**Trabajo Práctico Extensivo: Creando Servidores con Express.js**

Esta guía te llevará paso a paso a través de la creación de servidores web y APIs utilizando Express.js, el framework más popular para Node.js. Aprenderás a manejar rutas, recibir datos de los usuarios y enviar respuestas en diferentes formatos.

**Requisitos Previos:**

* Tener Node.js y npm instalados.
* Conocimientos básicos de JavaScript (variables, funciones, objetos, arrays).

**Configuración Inicial para Todos los Ejercicios:**

1. Crea una nueva carpeta para tu proyecto (ej: proyecto-express).
2. Ábrela en tu terminal.
3. Inicia un proyecto de Node.js: npm init -y
4. Instala Express: npm install express

**1. El Servidor Básico**

**Documentación y Explicación**

* **require('express')**: Importa el módulo de Express a tu archivo, permitiéndote usar sus funcionalidades.
* **const app = express()**: Crea una instancia de una aplicación de Express. La variable app es el objeto principal que representa tu aplicación web.
* **app.listen(puerto, callback)**: Inicia el servidor y lo pone a "escuchar" peticiones en un puerto específico. El callback es una función opcional que se ejecuta una vez que el servidor está listo y escuchando.

**Ejemplo 1: "Hola Mundo"**

// server.js  
const express = require('express');  
const app = express();  
const PUERTO = 3000;  
  
// Cuando alguien visita la ruta principal ('/'), le respondemos.  
app.get('/', (req, res) => {  
  res.send('¡Hola Mundo!');  
});  
  
// Iniciamos el servidor  
app.listen(PUERTO, () => {  
  console.log(`Servidor escuchando en http://localhost:${PUERTO}`);  
});

*Para probarlo, guarda este código en un archivo server.js, ejecuta node server.js en tu terminal y visita http://localhost:3000 en tu navegador.*

**Ejercicios de Servidor Básico**

Ejercicio 1.1: Mi Primer Servidor

Tarea: Crea un archivo app.js. Escribe el código para iniciar un servidor en el puerto 8080 que, al ser visitado en su ruta principal (/), responda con el texto "¡Mi primer servidor con Express!".

Ejercicio 1.2: Mensaje de Bienvenida

Tarea: Crea un nuevo servidor desde cero en un archivo welcome.js. Debe correr en el puerto 4000 y responder en la ruta principal con un <h1> que diga "Bienvenido a mi sitio web".

Pista: res.send() puede enviar texto HTML.

**2. Rutas (Routing)**

**Documentación y Explicación**

El enrutamiento se refiere a cómo la aplicación responde a una petición de un cliente a un endpoint particular (una URI o ruta).

* **app.METHOD(PATH, HANDLER)**: Es la estructura básica.
* METHOD: Es un método HTTP en minúsculas (get, post, put, delete, etc.).
* PATH: Es la ruta en el servidor (ej: /, /productos, /usuarios).
* HANDLER: Es la función que se ejecuta cuando la ruta coincide. Siempre recibe dos parámetros principales: req (el objeto de la petición) y res (el objeto de la respuesta).
* **res.send('...')**: Envía una respuesta de texto, HTML o incluso JSON al cliente.
* **res.status(codigo)**: Establece el código de estado HTTP de la respuesta (ej: 200 para OK, 404 para No Encontrado).

**Ejercicios de Rutas**

Ejercicio 2.1: Múltiples Páginas

Tarea: Crea un servidor que tenga tres rutas diferentes, cada una con un mensaje distinto:

1. GET /: "Página de Inicio".
2. GET /nosotros: "Acerca de Nosotros".
3. GET /contacto: "Página de Contacto".

Ejercicio 2.2: Mi Tienda Online

Tarea: Crea un nuevo servidor que simule una tienda online con las siguientes rutas:

1. GET /: "Bienvenido a MiTienda".
2. GET /productos: "Listado de todos nuestros productos".
3. GET /categorias: "Categorías disponibles: Electrónica, Ropa, Hogar".
4. GET /ofertas: "¡No te pierdas nuestras ofertas especiales!".

Ejercicio 2.3: Manejo de Rutas no Encontradas (404)

Tarea: A veces un usuario intenta acceder a una ruta que no existe. Podemos manejar esto para dar una respuesta personalizada.

1. Toma el servidor del ejercicio 2.2.
2. Al final de todas tus definiciones de rutas, añade un manejador especial que capture todas las demás peticiones.
3. Este manejador debe responder con un código de estado 404 y el mensaje "Error 404: La página que buscas no existe."

**Pista:** app.use((req, res) => { ... }) se puede usar para capturar cualquier petición que no haya coincidido con las rutas anteriores.

**3. Parámetros de Ruta**

**Documentación y Explicación**

Los parámetros de ruta son segmentos de la URL que se usan para capturar valores. Son muy útiles para, por ejemplo, obtener el ID de un producto o el nombre de un usuario.

* **Definición:** Se definen en la ruta con dos puntos (:), ej: /usuarios/:id.
* **Acceso:** El valor capturado está disponible en el objeto req.params. Para la ruta /usuarios/:id, si un usuario visita /usuarios/123, req.params.id será el string "123".

**Ejemplo 3: Obtener un Usuario por ID**

app.get('/usuarios/:id', (req, res) => {  
  const userId = req.params.id;  
  res.send(`Mostrando información del usuario con ID: ${userId}`);  
});

*Visita http://localhost:3000/usuarios/123 para ver el resultado.*

**Ejercicios con Parámetros de Ruta**

Ejercicio 3.1: Saludo Personalizado

Tarea: Crea una ruta GET /saludo/:nombre que capture el nombre de la URL y devuelva un saludo personalizado. Por ejemplo, si visitas /saludo/Juan, debe responder "¡Hola, Juan! Bienvenido.".

Ejercicio 3.2: Detalles de un Producto

Tarea: Crea una ruta GET /productos/:idProducto que simule mostrar los detalles de un producto. Debe responder con un mensaje como "Información del producto con ID: [idProducto]".

Ejercicio 3.3: Operaciones Matemáticas

Tarea: Crea una ruta GET /calcular/sumar/:num1/:num2. Debe tomar los dos números de la URL, realizar la suma y devolver el resultado.

Pista: Los valores en req.params son strings. Necesitarás convertirlos a números usando parseInt() o Number().

Ejercicio 3.4: Búsqueda por Categoría y Subcategoría

Tarea: Crea una ruta que maneje dos parámetros, como /productos/:categoria/:subcategoria. Debe responder con un mensaje que confirme ambos parámetros, como "Mostrando productos de la categoría [categoria], subcategoría [subcategoria]".

**4. Query Strings (Consultas en la URL)**

**Documentación y Explicación**

Los query strings son pares clave-valor que aparecen al final de una URL después de un ? y se separan con &. Se usan para pasar datos opcionales, como filtros, opciones de ordenamiento o términos de búsqueda.

* **URL de Ejemplo:** http://localhost:3000/buscar?termino=libros&ordenar=precio
* **Acceso:** Express parsea estos valores automáticamente y los pone a tu disposición en el objeto req.query. Para el ejemplo anterior, req.query sería { termino: 'libros', ordenar: 'precio' }.

**Ejercicios con Query Strings**

Ejercicio 4.1: Búsqueda Simple

Tarea: Crea una ruta GET /buscar que espere un query string q. Debe responder con "Resultados de búsqueda para: [valor de q]". Si q no está presente, debe responder "Por favor, ingresa un término de búsqueda".

Ejercicio 4.2: Filtrar Productos Avanzado

Tarea: Crea una ruta GET /productos que pueda filtrar por categoria y precio\_max.

* Si se visita /productos?categoria=tecnologia, debe responder "Filtrando por categoría: tecnologia".
* Si se visita /productos?precio\_max=500, debe responder "Filtrando por precio máximo: 500".
* Si se visita /productos?categoria=hogar&precio\_max=200, debe responder "Filtrando por categoría hogar y precio máximo 200".
* Si se visita /productos, debe responder "Mostrando todos los productos".

**5. Middleware**

**Documentación y Explicación**

Los **Middleware** son funciones que se ejecutan en medio del ciclo de petición-respuesta. Tienen acceso a los objetos req y res, y a una función especial llamada next. Se usan para tareas como logging, autenticación, parseo de datos, etc.

* **app.use(miFuncionMiddleware)**: Aplica el middleware a todas las peticiones que lleguen después de esta línea.
* **next()**: Es una función que se llama dentro del middleware para pasar el control al siguiente middleware o a la ruta final. **Si no llamas a next(), la petición se quedará "colgada".**

**Ejemplo 5: Logger Simple**

const logger = (req, res, next) => {  
  console.log(`Petición recibida: ${req.method} en ${req.url}`);  
  next(); // ¡Crucial! Pasa al siguiente manejador.  
};  
  
app.use(logger); // Aplicar a todas las rutas

**Ejercicios con Middleware**

Ejercicio 5.1: Logger de Fecha y Hora

Tarea: Crea un middleware que imprima en la consola la fecha y hora actual cada vez que se recibe una petición.

Ejercicio 5.2: Middleware de Autenticación Falso

Tarea: Crea un middleware que simule una autenticación.

1. El middleware debe verificar si un query string apiKey tiene el valor 12345.
2. Si es correcto, debe llamar a next() para permitir el acceso.
3. Si es incorrecto o no está presente, debe enviar una respuesta de error con estado 401 (No autorizado) y el mensaje "Acceso denegado. Se requiere una API Key válida.".
4. Aplica este middleware **solo a una ruta específica**, por ejemplo /admin, no a toda la aplicación.

**Pista:** Puedes pasar una función de middleware como argumento antes del manejador final de la ruta: app.get('/ruta', miMiddleware, (req, res) => { ... });

**6. Manejo de JSON y APIs**

**Documentación y Explicación**

Las APIs modernas se comunican principalmente con JSON.

* **res.json(objeto)**: Envía una respuesta en formato JSON. Express establece automáticamente la cabecera Content-Type a application/json.
* **app.use(express.json())**: Un middleware nativo de Express que es **esencial** para recibir datos JSON. Parsea las peticiones entrantes con Content-Type: application/json y pone los datos parseados en req.body.

**Ejercicios de API con JSON**

Ejercicio 6.1: API de Productos

Tarea: Crea una ruta GET /api/productos que devuelva un array de objetos de producto en formato JSON.

Ejercicio 6.2: Obtener un Producto por ID

Tarea: Crea una ruta GET /api/productos/:id que busque un producto en un array (simulando una base de datos) por su id.

* Si encuentra el producto, lo devuelve como JSON.
* Si no lo encuentra, devuelve un objeto de error { error: 'Producto no encontrado' } con un estado 404.

Ejercicio 6.3: Crear un Nuevo Producto (POST)

Tarea: Crea una ruta POST /api/productos para añadir un nuevo producto.

1. Asegúrate de tener app.use(express.json()); al principio de tu script.
2. El nuevo producto vendrá en el cuerpo de la petición (req.body).
3. Añade el nuevo producto a tu array de productos.
4. Responde con el producto recién creado (incluyendo un nuevo id) y un estado 201 (Creado).

**7. Ejercicio Final Integrador: Mini API CRUD para Libros**

**Tarea:** Crear una API completa para gestionar una colección de libros. CRUD significa **C**reate, **R**ead, **U**pdate, **D**elete.

1. **Configuración:**

* Crea un archivo api.js.
* Usa express.json().
* Crea un array inicial de objetos libros, donde cada libro tenga id, titulo y autor.

1. **Implementa los siguientes endpoints:**

* **GET /libros**: Devuelve todos los libros.
* **GET /libros/:id**: Devuelve un libro específico por su id.
* **POST /libros**: Crea un nuevo libro. El id debe generarse automáticamente.
* **PUT /libros/:id**: Actualiza un libro existente. El cuerpo de la petición contendrá la nueva información.
* **DELETE /libros/:id**: Elimina un libro por su id.

¡Felicidades por llegar hasta aquí! Completar este trabajo práctico te dará una base muy sólida para construir cualquier tipo de aplicación web con Node.js y Express.