# Taller Práctico: Creación de API con Expressjs

Julian F. Latorre
Bootcamp Desarrollo Web Full Stack

## Introducción

En este taller, vamos a poner en práctica los conceptos aprendidos sobre la creación de API con Express.js. Construiremos una API sencilla para gestionar una lista de tareas (TODO list).

## Objetivos

Al finalizar este taller, serás capaz de:

- Configurar un proyecto de Express.js desde cero
- Crear rutas y manejar métodos HTTP
- Implementar funcionalidades CRUD (Crear, Leer, Actualizar, Eliminar) para tareas
- Utilizar middleware para procesar solicitudes y manejar errores
- Documentar tu API usando Postman

## Requisitos previos

- Conocimientos básicos de JavaScript
- Familiaridad con Node.js y npm
- Tener instalado Node.js en tu sistema

## Instrucciones

### Paso 1: Inicializar el proyecto

1. Crea un nuevo directorio para tu proyecto: `mkdir todo-api`
2. Navega al directorio: `cd todo-api`

3. Inicializa un nuevo proyecto Node.js: `npm init -y`
4. Instala Express.js: `npm install express`
5. Crea un nuevo archivo llamado `app.js` en el directorio raíz del proyecto.

## Paso 2: Crear un servidor básico

Abre `app.js` y agrega el siguiente código:

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

app.get('/', (req, res) => {
  res.send('  Bienvenido   a la API de tareas!');
});

app.listen(port, () => {
  console.log(`Servidor escuchando en http://localhost:${port}`);
});
```

Guarda el archivo y ejecuta el servidor con el comando: `node app.js`. Abre tu navegador y visita `http://localhost:3000` para verificar que el servidor se esté ejecutando correctamente.

## Paso 3: Crear rutas para la gestión de tareas

Ahora vamos a agregar las rutas necesarias para crear, leer, actualizar y eliminar tareas.

1. Agrega el siguiente código en `app.js`:

```
// Middleware para procesar JSON
app.use(express.json());

// Ruta para obtener todas las tareas
app.get('/api/tareas', (req, res) => {
  // Aquí irá la lógica para obtener todas las tareas
  res.json([
    { id: 1, titulo: 'Terminar el taller de Express.js',
    completada: false },
    { id: 2, titulo: 'Hacer las compras', completada: true },
    { id: 3, titulo: 'Limpiar el apartamento', completada:
    false }
  ]);
});

// Ruta para crear una nueva tarea
app.post('/api/tareas', (req, res) => {
  const nuevaTarea = req.body;
  // Aquí irá la lógica para crear una nueva tarea
  res.status(201).json(nuevaTarea);
});
```

```
// Ruta para actualizar una tarea
app.put('/api/tareas/:id', (req, res) => {
  const id = req.params.id;
  const datosActualizados = req.body;
  // Aquí irá la lógica para actualizar una tarea
  res.json({ mensaje: `Tarea ${id} actualizada`, datos:
    datosActualizados });
});

// Ruta para eliminar una tarea
app.delete('/api/tareas/:id', (req, res) => {
  const id = req.params.id;
  // Aquí irá la lógica para eliminar una tarea
  res.json({ mensaje: `Tarea ${id} eliminada` });
});
```

2. Guarda el archivo y reinicia el servidor. Ahora puedes probar las diferentes rutas en tu navegador o usando una herramienta como Postman.

## Paso 4: Implementar la lógica de negocio

Hasta ahora, hemos definido las rutas, pero no hemos implementado la lógica real para manejar las tareas. En este paso, vamos a agregar la funcionalidad CRUD (Crear, Leer, Actualizar, Eliminar) para las tareas.

1. Crea un nuevo archivo llamado `tareas.js` en el directorio raíz del proyecto.

2. En `tareas.js`, agrega el siguiente código:

```
let tareas = [
  { id: 1, titulo: 'Terminar el taller de Express.js',
    completada: false },
  { id: 2, titulo: 'Hacer las compras', completada: true },
  { id: 3, titulo: 'Limpiar el apartamento', completada: false
     }
];

exports.obtenerTareas = () => {
  return tareas;
};

exports.crearTarea = (nuevaTarea) => {
  const id = tareas.length + 1;
  const tarea = { id, ...nuevaTarea };
  tareas.push(tarea);
  return tarea;
};

exports.actualizarTarea = (id, datosActualizados) => {
  const index = tareas.findIndex((tarea) => tarea.id ===
    parseInt(id));
  if (index === -1) {
    return null;
  }
```

```
  tareas[index] = { ...tareas[index], ...datosActualizados };
  return tareas[index];
};

exports.eliminarTarea = (id) => {
  const index = tareas.findIndex((tarea) => tarea.id ===
    parseInt(id));
  if (index === -1) {
    return null;
  }
  const eliminada = tareas.splice(index, 1)[0];
  return eliminada;
};
```

3. Ahora, actualiza las rutas en `app.js` para usar las funciones definidas en `tareas.js`:

```
const tareas = require('./tareas');

// Ruta para obtener todas las tareas
app.get('/api/tareas', (req, res) => {
  res.json(tareas.obtenerTareas());
});

// Ruta para crear una nueva tarea
app.post('/api/tareas', (req, res) => {
  const nuevaTarea = req.body;
  const tarea = tareas.crearTarea(nuevaTarea);
  res.status(201).json(tarea);
});

// Ruta para actualizar una tarea
app.put('/api/tareas/:id', (req, res) => {
  const id = req.params.id;
  const datosActualizados = req.body;
  const tareaActualizada = tareas.actualizarTarea(id,
    datosActualizados);
  if (tareaActualizada) {
    res.json(tareaActualizada);
  } else {
    res.status(404).json({ mensaje: `No se encontró la tarea $
    {id}` });
  }
});

// Ruta para eliminar una tarea
app.delete('/api/tareas/:id', (req, res) => {
  const id = req.params.id;
  const tareaEliminada = tareas.eliminarTarea(id);
  if (tareaEliminada) {
    res.json({ mensaje: `Tarea ${id} eliminada` });
  } else {
    res.status(404).json({ mensaje: `No se encontró la tarea $
    {id}` });
  }
});
```

4. Guarda los cambios y reinicia el servidor. Ahora puedes probar las rutas CRUD para las tareas.

### Paso 5: Documentar la API con Postman

Para que otros desarrolladores puedan entender y utilizar tu API, es importante documentarla adecuadamente. Vamos a usar Postman para este propósito.

1. Descarga e instala Postman si aún no lo tienes.
2. Abre Postman y crea una nueva colección para tu API de tareas.
3. Agrega las diferentes rutas a la colección y configura los métodos HTTP, parámetros, cuerpos de solicitud, etc.
4. Documenta cada ruta con una descripción, ejemplos de solicitud y respuesta, y cualquier otra información relevante.
5. Guarda la colección de Postman para poder compartirla con otros desarrolladores.

## Conclusión

¡Felicidades! Has completado el taller y has construido una API básica para la gestión de tareas utilizando Express.js. Ahora tienes las habilidades necesarias para continuar explorando y desarrollando APIs más complejas.

Algunos próximos pasos podrían ser:

- Agregar autenticación y autorización a tus APIs
- Integrar una base de datos para almacenar las tareas
- Mejorar la estructura del proyecto siguiendo mejores prácticas
- Agregar pruebas unitarias y de integración