# Creación de API con Expressjs

Julian F. Latorre

5 de noviembre de 2024

# Contenido

# ¿Qué es Express.js?

- Framework minimalista y flexible para Node.js
- Proporciona características robustas para aplicaciones web y móviles
- Ampliamente utilizado para crear APIs RESTful

# ¿Qué es una API?

- API: Interfaz de Programación de Aplicaciones
- Conjunto de definiciones y protocolos
- Utilizada para desarrollar e integrar software
- Facilita la comunicación entre diferentes sistemas

# Configuración inicial

1. Verificar la instalación de Node.js:

```
node --version
```

2. Crear un nuevo directorio para el proyecto:

```
mkdir mi-api-express
cd mi-api-express
```

3. Inicializar un nuevo proyecto Node.js:

```
npm init -y
```

4. Instalar Express.js:

```
npm install express
```

# Servidor básico con Express.js

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

app.get('/', (req, res) => {
  res.send(' Hola , mundo!');
});

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

# Métodos HTTP comunes

- GET: Para recuperar recursos
- POST: Para crear nuevos recursos
- PUT: Para actualizar recursos existentes
- DELETE: Para eliminar recursos

# Ejemplo de rutas con diferentes métodos

```
// GET
app.get('/api/usuarios', (req, res) => {
  res.json({ mensaje: 'Obtener todos los usuarios' });
});

// POST
app.post('/api/usuarios', (req, res) => {
  const nuevoUsuario = req.body;
  res.status(201).json({ mensaje: 'Usuario creado',
      usuario: nuevoUsuario });
});

// PUT
app.put('/api/usuarios/:id', (req, res) => {
  const id = req.params.id;
  const datosActualizados = req.body;
  res.json({ mensaje: `Usuario ${id} actualizado`,
      datos: datosActualizados });
});
```

# Manejo de parámetros

```
// Parámetros de ruta
app.get('/api/usuarios/:id', (req, res) => {
  const id = req.params.id;
  res.json({ mensaje: `Detalles del usuario ${id}` });
});

// Parámetros de consulta
app.get('/api/productos', (req, res) => {
  const { categoria, orden } = req.query;
  res.json({ mensaje: `Productos de ${categoria}
      ordenados por ${orden}` });
});
```

# Middleware personalizado

```
const loggerMiddleware = (req, res, next) => {
  console.log(`${new Date().toISOString()} - ${req.
      method} ${req.url}`);
  next();
};

app.use(loggerMiddleware);
```

# Middleware de manejo de errores

```
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({ error: 'Algo salió mal!' });
});
```

# Estructura recomendada para proyectos grandes

- `routes/`: Archivos de ruta para diferentes recursos
- `controllers/`: Lógica de negocio
- `models/`: Definiciones de modelos de datos
- `middleware/`: Funciones de middleware personalizadas
- `config/`: Archivos de configuración

# Herramientas para documentar APIs

- Swagger/OpenAPI
- API Blueprint
- Postman

# Herramientas para pruebas

- Mocha y Chai para pruebas unitarias
- Supertest para pruebas de integración
- Postman para pruebas manuales y automatizadas

# Conclusión

- Hemos cubierto los conceptos básicos de la creación de API con Express.js
- La práctica es clave para dominar estos conceptos
- Continúe explorando y construyendo proyectos más complejos
- 🚀 ¡Empiece a construir sus propias APIs!