Curso de ProgramaciÃ³n Desde Cero

Tema 5: Backend con Node.js

5.1 IntroducciÃ³n al Desarrollo Backend

El desarrollo backend se refiere a la parte del servidor de las aplicaciones web. Mientras que el frontend es lo que ven los usuarios, el backend maneja la lÃ³gica de negocio, bases de datos, autenticaciÃ³n, APIs y toda la infraestructura que hace que las aplicaciones funcionen de manera robusta y segura.

Â¿QuÃ© es Node.js?

Node.js es un entorno de ejecuciÃ³n de JavaScript del lado del servidor, construido sobre el motor V8 de Chrome. Permite usar JavaScript para desarrollo backend, lo que significa que puedes usar el mismo lenguaje tanto en frontend como en backend.

Ventajas de Node.js

â€¢ Un Solo Lenguaje: JavaScript en frontend y backend
â€¢ Alto Rendimiento: Arquitectura asÃ-ncrona y no bloqueante
â€¢ NPM: Ecosistema masivo de paquetes y bibliotecas
â€¢ Escalabilidad: Ideal para aplicaciones en tiempo real
â€¢ Comunidad: Soporte activo y recursos abundantes
â€¢ Velocidad de Desarrollo: RÃ¡pido prototipado y desarrollo

Arquitectura de Node.js

Node.js utiliza una arquitectura event-driven con un bucle de eventos Ãºnico:
â€¢ Event Loop: Bucle principal que maneja operaciones asÃ-ncronas
â€¢ Thread Pool: Pool de hilos para operaciones intensivas
â€¢ Non-blocking I/O: Operaciones de entrada/salida no bloqueantes
â€¢ Callbacks/Promises: Manejo de operaciones asÃ-ncronas

Casos de Uso Ideales

Node.js es especialmente bueno para:
â€¢ APIs RESTful y GraphQL
â€¢ Aplicaciones en tiempo real (chat, juegos online)
â€¢ Microservicios y arquitecturas distribuidas
â€¢ Aplicaciones con muchas conexiones concurrentes
â€¢ Proxies y sistemas de streaming
â€¢ Herramientas de desarrollo y automatizaciÃ³n

5.2 ConfiguraciÃ³n del Entorno

InstalaciÃ³n de Node.js

5.3 Primeros Pasos con Node.js

Hola Mundo en Node.js

Comencemos con un servidor bÃ¡sico:

```
// server.js
const http = require('http');

const server = http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/plain; charset=utf-8' });
  res.end('Â¡Hola Mundo desde Node.js!');
});

const PORT = 3000;
server.listen(PORT, () => {
  console.log(`Servidor ejecutÃ¡ndose en http://localhost:${PORT}`);
});
```

MÃ³dulos en Node.js

Node.js usa el sistema de mÃ³dulos CommonJS:

```
// utils/matematicas.js
function sumar(a, b) {
  return a + b;
}

function restar(a, b) {
  return a - b;
}

// Exportar funciones
module.exports = {
  sumar,
  restar
};
```

```
// app.js - Importar mÃ³dulo
const { sumar, restar } = require('./utils/matematicas');

console.log(sumar(5, 3));  // 8
console.log(restar(10, 4)); // 6
```

File System (Sistema de Archivos)

Node.js puede interactuar con el sistema de archivos:

```
const fs = require('fs');
const path = require('path');

// Leer archivo de forma sÃ-ncrona
try {
  const data = fs.readFileSync('archivo.txt', 'utf8');
  console.log(data);
} catch (err) {
  console.error('Error leyendo archivo:', err);
}

// Leer archivo de forma asÃ-ncrona (recomendado)
fs.readFile('archivo.txt', 'utf8', (err, data) => {
  if (err) {
    console.error('Error:', err);
    return;
  }
  console.log('Contenido:', data);
});
```

5.4 Express.js: Framework Web

Â¿QuÃ© es Express.js?

Express es un framework web minimalista y flexible para Node.js que proporciona un conjunto robusto de caracterÃ-sticas para aplicaciones web y mÃ³viles. Simplifica la creaciÃ³n de servidores y APIs.

Servidor BÃ¡sico con Express

```
const express = require('express');
const app = express();
const PORT = process.env.PORT || 3000;

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

// Ruta bÃ¡sica
app.get('/', (req, res) => {
  res.json({ mensaje: 'Â¡Hola desde Express!' });
});

// Iniciar servidor
app.listen(PORT, () => {
  console.log(`Servidor Express ejecutÃ¡ndose en puerto ${PORT}`);
});
```

Rutas y MÃ©todos HTTP

```
// GET - Obtener datos
app.get('/usuarios', (req, res) => {
  res.json({ usuarios: ['Ana', 'Carlos', 'MarÃ-a'] });
});

// POST - Crear nuevo recurso
app.post('/usuarios', (req, res) => {
  const { nombre, email } = req.body;
  res.status(201).json({
    mensaje: 'Usuario creado',
    usuario: { nombre, email }
  });
});

// PUT - Actualizar recurso completo
app.put('/usuarios/:id', (req, res) => {
  const { id } = req.params;
  const { nombre, email } = req.body;
  res.json({
    mensaje: `Usuario ${id} actualizado`,
    usuario: { id, nombre, email }
  });
});

// PATCH - Actualizar recurso parcial
app.patch('/usuarios/:id', (req, res) => {
  const { id } = req.params;
  const updates = req.body;
  res.json({
    mensaje: `Usuario ${id} actualizado parcialmente`,
    updates
  });
});

// DELETE - Eliminar recurso
app.delete('/usuarios/:id', (req, res) => {
```

5.5 APIs RESTful

Â¿QuÃ© es REST?

REST (Representational State Transfer) es un estilo arquitectÃ³nico para diseÃ±ar servicios web. Define un conjunto de principios para crear APIs escalables, mantenibles y fÃ¡ciles de usar.

Principios REST

â€¢ Sin Estado: Cada peticiÃ³n debe contener toda la informaciÃ³n necesaria
â€¢ Interfaz Uniforme: URLs y mÃ©todos HTTP estÃ¡ndar
â€¢ Cacheable: Las respuestas deben indicar si son cacheables
â€¢ Cliente-Servidor: SeparaciÃ³n clara de responsabilidades
â€¢ Sistema por Capas: Arquitectura en capas transparente

API Completa de Tareas

Implementemos una API RESTful completa para gestionar tareas:

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

app.use(express.json());

// Base de datos temporal (en memoria)
let tareas = [
  { id: 1, titulo: 'Aprender Node.js', completada: false, fechaCreacion: new Date() },
  { id: 2, titulo: 'Crear API REST', completada: true, fechaCreacion: new Date() }
];
let siguienteId = 3;

// GET /api/tareas - Obtener todas las tareas
app.get('/api/tareas', (req, res) => {
  const { completada, buscar } = req.query;
  let resultado = tareas;

  // Filtrar por estado
  if (completada !== undefined) {
    resultado = resultado.filter(t => t.completada === (completada === 'true'));
  }

  // Buscar por tÃ-tulo
  if (buscar) {
    resultado = resultado.filter(t =>
      t.titulo.toLowerCase().includes(buscar.toLowerCase())
    );
  }

  res.json({
    total: resultado.length,
    tareas: resultado
  });
});

// GET /api/tareas/:id - Obtener tarea especÃ-fica
app.get('/api/tareas/:id', (req, res) => {
  const id = parseInt(req.params.id);
  const tarea = tareas.find(t => t.id === id);

  if (!tarea) {
    return res.status(404).json({ error: 'Tarea no encontrada' });
  }

  res.json(tarea);
```

5.6 ValidaciÃ³n y Manejo de Errores

ValidaciÃ³n con express-validator

La validaciÃ³n es crucial para la seguridad y consistencia de datos:

```
npm install express-validator
```

```
const { body, validationResult } = require('express-validator');
// Middleware de validaciÃ³n
const validarTarea = [
  body('titulo')
    .notEmpty()
    .withMessage('El tÃ-tulo es requerido')
    .isLength({ min: 3, max: 100 })
    .withMessage('El tÃ-tulo debe tener entre 3 y 100 caracteres'),
  body('descripcion')
    .optional()
    .isLength({ max: 500 })
    .withMessage('La descripciÃ³n no puede exceder 500 caracteres'),
  body('completada')
    .optional()
    .isBoolean()
    .withMessage('Completada debe ser un valor booleano')
];
// Middleware para manejar errores de validaciÃ³n
const manejarErroresValidacion = (req, res, next) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json({
      error: 'Datos invÃ¡lidos',
      detalles: errors.array()
    });
  }
  next();
};
// Usar validaciÃ³n en rutas
app.post('/api/tareas', validarTarea, manejarErroresValidacion, (req, res) => {
  // LÃ³gica para crear tarea
});
```

Manejo Centralizado de Errores

```
// Clase personalizada para errores
class AppError extends Error {
  constructor(message, statusCode) {
    super(message);
    this.statusCode = statusCode;
    this.isOperational = true;
    Error.captureStackTrace(this, this.constructor);
  }
}
// Middleware para capturar errores async
const catchAsync = (fn) => {
  return (req, res, next) => {
    fn(req, res, next).catch(next);
  };
};
// Middleware global de manejo de errores
```

5.7 AutenticaciÃ³n y AutorizaciÃ³n

JWT (JSON Web Tokens)

JWT es un estÃ¡ndar para transmitir informaciÃ³n de forma segura:

```
npm install jsonwebtoken bcryptjs
```

```
const jwt = require('jsonwebtoken');
const bcrypt = require('bcryptjs');
// Base de datos temporal de usuarios
const usuarios = [];
// Registro de usuario
app.post('/api/auth/registro', async (req, res) => {
  try {
    const { nombre, email, password } = req.body;
    // Verificar si el usuario ya existe
    const usuarioExistente = usuarios.find(u => u.email === email);
    if (usuarioExistente) {
      return res.status(400).json({ error: 'Usuario ya existe' });
    }
    // Hashear password
    const salt = await bcrypt.genSalt(10);
    const passwordHasheada = await bcrypt.hash(password, salt);
    // Crear usuario
    const nuevoUsuario = {
      id: usuarios.length + 1,
      nombre,
      email,
      password: passwordHasheada,
      fechaRegistro: new Date()
    };
    usuarios.push(nuevoUsuario);
    // Generar JWT
    const token = jwt.sign(
      { id: nuevoUsuario.id, email: nuevoUsuario.email },
      process.env.JWT_SECRET,
      { expiresIn: '7d' }
    );
    res.status(201).json({
      mensaje: 'Usuario registrado exitosamente',
      token,
      usuario: {
        id: nuevoUsuario.id,
        nombre: nuevoUsuario.nombre,
        email: nuevoUsuario.email
      }
    });
  } catch (error) {
    res.status(500).json({ error: 'Error interno del servidor' });
  }
});
// Login de usuario
app.post('/api/auth/login', async (req, res) => {
  try {
    const { email, password } = req.body;
    // Encontrar usuario
    const usuario = usuarios.find(u => u.email === email);
```

5.8 Testing y DocumentaciÃ³n

Testing con Jest y Supertest

Las pruebas aseguran la calidad y confiabilidad del cÃ³digo:

```
npm install --save-dev jest supertest
```

```
// tests/api.test.js
const request = require('supertest');
const app = require('../app');

describe('API de Tareas', () => {
  test('GET /api/tareas debe retornar todas las tareas', async () => {
    const res = await request(app)
      .get('/api/tareas')
      .expect(200);

    expect(res.body).toHaveProperty('tareas');
    expect(Array.isArray(res.body.tareas)).toBe(true);
  });

  test('POST /api/tareas debe crear una nueva tarea', async () => {
    const nuevaTarea = {
      titulo: 'Tarea de prueba',
      descripcion: 'DescripciÃ³n de prueba'
    };

    const res = await request(app)
      .post('/api/tareas')
      .send(nuevaTarea)
      .expect(201);

    expect(res.body).toHaveProperty('id');
    expect(res.body.titulo).toBe(nuevaTarea.titulo);
  });
});
```

DocumentaciÃ³n con Swagger/OpenAPI

```
npm install swagger-jsdoc swagger-ui-express
```

```
const swaggerJsdoc = require('swagger-jsdoc');
const swaggerUi = require('swagger-ui-express');

const swaggerOptions = {
  definition: {
    openapi: '3.0.0',
    info: {
      title: 'API de Tareas',
      version: '1.0.0',
      description: 'API RESTful para gestiÃ³n de tareas'
    },
    servers: [
      { url: 'http://localhost:3000', description: 'Desarrollo' }
    ]
  },
  apis: ['./routes/*.js']
};

const specs = swaggerJsdoc(swaggerOptions);
app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(specs));
```

Deployment y ProducciÃ³n

Consideraciones para poner la API en producciÃ³n: