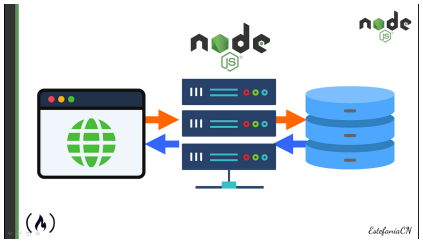
Práctica Sugerida

Desarrollo Backend con Node.js y Express



Bootcamp Desarrollo Fullstack
Julian F. Latorre

14 de noviembre de 2024

Información del Laboratorio

- **Duración:** 3 horas
- **Puntuación total:** 100 puntos
- **Requisitos previos:** Node.js instalado, MongoDB, editor de código
- **Entregables:** Repositorio Git con el código completo

1. Línea de Comandos y Node.js Básico (20 puntos)

Ejercicio 1.1: Estructura del Proyecto

Cree la siguiente estructura de directorios utilizando la línea de comandos:

```
proyecto-api/
├── src/
│   ├── controllers/
│   ├── models/
│   ├── routes/
│   ├── middleware/
│   └── config/
├── tests/
├── docs/
├── node_modules/
├── .env
├── .gitignore
└── package.json
```

Tareas:

1. Escriba los comandos exactos para crear esta estructura
2. Inicialice un repositorio git
3. Configure un archivo .gitignore apropiado
4. Documente los comandos utilizados

Ejercicio 1.2: Script Node.js Básico

Cree un script que realice las siguientes operaciones:

```
// file: src/utils/fileAnalyzer.js
const fs = require('fs');

// 1. Leer un archivo de configuración JSON
// 2. Procesar su contenido
// 3. Generar un reporte en formato txt
// 4. Implementar manejo de errores

// Implemente el código aquí
```

2. NPM y Gestión de Dependencias (20 puntos)

Ejercicio 2.1: Configuración del Proyecto

Configure un nuevo proyecto Node.js:

1. Inicialice un nuevo proyecto con npm
2. Instale las siguientes dependencias:

```
npm install express mongoose dotenv cors
npm install --save-dev nodemon jest supertest
```

3. Configure los scripts en package.json:

```
{
  "scripts": {
    "start": "node src/index.js",
    "dev": "nodemon src/index.js",
    "test": "jest --verbose",
    "lint": "eslint src/**/*.js"
  }
}
```

3. API con Express (30 puntos)

Ejercicio 3.1: Configuración de Express

Implemente una API REST para un sistema de gestión de productos:

```
// src/index.js
const express = require('express');
const cors = require('cors');
require('dotenv').config();

const app = express();

// Configuración de middleware
app.use(cors());
app.use(express.json());

// Implementar:
// 1. Conexión a MongoDB
// 2. Rutas básicas
// 3. Manejo de errores
// 4. Validación de datos

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
    console.log(`Server running on port ${PORT}`);
});
```

Estructura del modelo de Producto:

```
// src/models/Product.js
const mongoose = require('mongoose');

const productSchema = new mongoose.Schema({
    name: { type: String, required: true },
    price: { type: Number, required: true },
    // Complete el esquema
});
```

4. Bases de Datos y Operaciones CRUD (30 puntos)

Ejercicio 4.1: Implementación CRUD

Implemente los siguientes endpoints con conexión a MongoDB:

```
API Endpoints
├── GET /api/products
├── POST /api/products
├── GET /api/products/:id
├── PUT /api/products/:id
└── DELETE /api/products/:id
```

Ejemplo de implementación de una ruta:

```
// src/routes/productRoutes.js
const express = require('express');
const router = express.Router();
const Product = require('../models/Product');

router.get('/', async (req, res) => {
    try {
        const products = await Product.find();
        res.json(products);
    } catch (error) {
        res.status(500).json({ error: error.message });
    }
});

// Implemente las demás rutas...
```

Ejercicio 4.2: Queries Avanzadas

Implemente endpoints adicionales que demuestren:

- Filtrado por múltiples campos
- Ordenamiento
- Paginación
- Agregaciones

```
// Ejemplo de query avanzada
router.get('/search', async (req, res) => {
    const { minPrice, maxPrice, category, sort } = req.
       query;
    const query = {};

    // Implemente los filtros y la lógica
    // de búsqueda avanzada
});
```

Criterios de Evaluación

- Funcionalidad completa (40 %)
- Manejo de errores y validaciones (20 %)
- Estructura y organización del código (20 %)
- Documentación y claridad (10 %)
- Buenas prácticas y patrones de diseño (10 %)