Laboratorio 6: Crear backend para la app React con conexión a base de datos y login (CRUD)

Objetivo del laboratorio

Desarrollar un backend completo con Node.js y Express que sirva como API RESTful para la SPA de React creada en el Laboratorio 5, implementando:

- Servidor con endpoints para todas las funcionalidades del frontend
- Conexión a base de datos MongoDB Atlas
- Sistema de autenticación JWT para usuarios
- Operaciones CRUD completas para productos y mensajes de contacto
- Integración completa frontend-backend con despliegue profesional

Actividades por pasos

Paso 1: Configuración Inicial del Proyecto Backend

El objetivo de este primer paso es establecer los cimientos sólidos del backend, creando la estructura organizada del proyecto, instalando todas las dependencias necesarias y configurando el entorno de desarrollo profesional. Una configuración adecuada desde el inicio es crucial para garantizar un desarrollo escalable, mantenible y libre de problemas técnicos.

Acciones Específicas

1. Crear la Estructura de Carpetas del Proyecto

Ubicación dentro del proyecto completo:

```
proyecto-spa/
├── frontend/          (Proyecto React del Laboratorio 5)
└── backend/           (Nuevo backend - Laboratorio 6)
```

Comandos para ejecutar en terminal:

```
# Navegar al directorio principal del proyecto
cd proyecto-spa

# Crear carpeta para el backend
mkdir backend

# Navegar a la carpeta backend
cd backend
```

2. Inicializar el Proyecto Node.js

Ejecutar en la terminal:

```
# Inicializar package.json con valores por defecto
npm init -y
```

Modificar el archivo package.json resultante:

```
{
  "name": "backend",
  "version": "1.0.0",
  "description": "Backend para la SPA de React - Laboratorio 6",
  "main": "server.js",
  "type": "module",
  "scripts": {
    "start": "node server.js",
    "dev": "nodemon server.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": ["nodejs", "express", "mongodb", "jwt", "api"],
  "author": "Tu Nombre",
  "license": "ISC"
}
```

Explicación de la configuración:

- "type": "module": Habilita el uso de import/export (ES6 modules)

- "main": "server.js": Punto de entrada de la aplicación
- "dev" **script**: Para desarrollo con reinicio automático
- "start" **script**: Para producción

3. Instalar Dependencias Principales

Dependencias de producción (esenciales):

```
npm install express mongoose jsonwebtoken bcryptjs cors dotenv
```

Dependencias de desarrollo (para facilitar el coding):

```
npm install -D nodemon
```

Explicación de cada dependencia:

- express: Framework web minimalista y flexible para Node.js
- mongoose: ODM (Object Data Modeling) para MongoDB y Node.js
- jsonwebtoken: Implementación de tokens JWT para autenticación
- bcryptjs: Para hashing seguro de contraseñas
- cors: Middleware para habilitar Cross-Origin Resource Sharing
- dotenv: Carga variables de entorno desde archivo .env
- nodemon: Reinicia automáticamente el servidor al detectar cambios

4. Crear la Estructura de Carpetas Organizada

Ejecutar los siguientes comandos en terminal:

```
# Crear estructura de carpetas organizada
mkdir config controllers middleware models routes utils

# Crear archivos principales
touch server.js .env .gitignore
```

Estructura final de carpetas:

```
backend/
├── config/          # Configuraciones (base de datos, cloudinary, etc.)
├── controllers/     # Lógica de negocio - manejan request/response
├── middleware/      # Funciones intermedias (auth, validations, logs)
├── models/          # Modelos de datos de MongoDB (Esquemas)
├── routes/          # Definición de endpoints de la API
├── utils/           # Utilidades y helpers personalizados
├── .env             # Variables de entorno (NO subir a GitHub)
├── .gitignore       # Archivos a ignorar por Git
├── package.json     # Dependencias y scripts
└── server.js        # Punto de entrada de la aplicación
```

5. Configurar Variables de Entorno Iniciales

Crear archivo .env con configuración básica:

```
# ==================================
# SERVER CONFIGURATION
# ==================================
PORT=5000
NODE_ENV=development

# ==================================
# DATABASE CONFIGURATION
# ==================================
MONGODB_URI=mongodb+srv://username:password@cluster0.xxxxx.mongodb.net/nombre-db?retryWrites=true&w=
majority

# ==================================
# JWT CONFIGURATION
# ==================================
JWT_SECRET=tu_jwt_super_secreto_aqui_minimo_32_caracteres_2024
JWT_EXPIRE=1h
JWT_COOKIE_EXPIRE=30
```

```
# ==================================
# FRONTEND CONFIGURATION
# ==================================
FRONTEND_URL=http://localhost:3000

# ==================================
# OTHER CONFIGURATIONS
# ==================================
BCRYPT_SALT_ROUNDS=12
```

Verificación de la Configuración

1. Verificar Instalación de Dependencias

Ejecutar en terminal:

```
# Verificar que las dependencias se instalaron correctamente
npm list --depth=0
```

Resultado esperado:

```
backend@1.0.0
├── bcryptjs@2.4.3
├── cors@2.8.5
├── dotenv@16.3.1
├── express@4.18.2
├── jsonwebtoken@9.0.2
└── mongoose@8.0.3
```

2. Verificar Estructura del Proyecto

Ejecutar en terminal:

```
# Verificar la estructura de carpetas creada
ls -la
```

Resultado esperado:

```
drwxr-xr-x  10 user  staff   320 Jan 15 10:30 .
drwxr-xr-x   4 user  staff   128 Jan 15 10:28 ..
-rw-r--r--   1 user  staff   456 Jan 15 10:30 .env
```

```
drwxr-xr-x   2 user  staff    64 Jan 15 10:29 config
drwxr-xr-x   2 user  staff    64 Jan 15 10:29 controllers
drwxr-xr-x   2 user  staff    64 Jan 15 10:29 middleware
drwxr-xr-x   2 user  staff    64 Jan 15 10:29 models
-rw-r--r--   1 user  staff  1324 Jan 15 10:29 package.json
drwxr-xr-x   2 user  staff    64 Jan 15 10:29 routes
drwxr-xr-x   2 user  staff    64 Jan 15 10:29 utils
-rw-r--r--   1 user  staff     0 Jan 15 10:29 server.js
```

3. Probar Ejecución del Servidor

Crear contenido básico en server.js para prueba:

```
console.log('✅ Backend configurado correctamente!');
console.log('📦 Dependencias instaladas:');
console.log('   - Express.js');
console.log('   - MongoDB Mongoose');
console.log('   - JWT Authentication');
console.log('   - Bcrypt para passwords');
console.log('   - CORS habilitado');
console.log('   - Variables de entorno cargadas');
```

Ejecutar el servidor:

```
npm run dev
```

Resultado esperado:

```
> backend@1.0.0 dev
> nodemon server.js

[nodemon] 3.0.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`

✅ Backend configurado correctamente!
📦 Dependencias instaladas:
   - Express.js
   - MongoDB Mongoose
   - JWT Authentication
   - Bcrypt para passwords
   - CORS habilitado
   - Variables de entorno cargadas
```

Solución de Problemas Comunes

Error: EACCES permissions

Solución: Configurar npm para no requerir sudo

```
# Crear directorio para instalaciones globales
mkdir ~/.npm-global

# Configurar npm para usar este directorio
npm config set prefix '~/.npm-global'

# Agregar a PATH en el perfil de shell
echo 'export PATH=~/.npm-global/bin:$PATH' >> ~/.bashrc
# o para zsh:
echo 'export PATH=~/.npm-global/bin:$PATH' >> ~/.zshrc

# Recargar el perfil
source ~/.bashrc  # o source ~/.zshrc
```

Error: Port already in use

Solución: Cambiar puerto o terminar proceso

```
# Verificar procesos usando el puerto 5000
lsof -ti:5000

# Terminar el proceso
kill -9 $(lsof -ti:5000)

# O cambiar el puerto en el archivo .env
```

Error: Module not found

Solución: Reinstalar dependencias

```
# Eliminar node_modules y package-lock
rm -rf node_modules package-lock.json

# Reinstalar dependencias
npm install
```

Paso 2: Configuración del Servidor Express Básico

El objetivo de este paso es crear un servidor Express funcional con toda la configuración esencial necesaria para una API RESTful moderna. Configuraremos middleware crucial, habilitaremos CORS para comunicación con el frontend React, implementaremos manejo básico de errores y crearemos endpoints de prueba para verificar que todo funciona correctamente.

Acciones Específicas

1. Configurar el Archivo Principal del Servidor

Crear/editar el archivo server.js en la raíz del backend:

```
import express from 'express';
import cors from 'cors';
import dotenv from 'dotenv';
import path from 'path';

// Importar rutas (las crearemos después)
import authRoutes from './routes/auth.js';
import productRoutes from './routes/products.js';

// Cargar variables de entorno
dotenv.config();

// Crear aplicación Express
const app = express();

// Middleware básico
app.use(express.json({ limit: '10mb' }));
app.use(express.urlencoded({ extended: true }));
```

```
// Configurar CORS para desarrollo
app.use(cors({
    origin: process.env.FRONTEND_URL || 'http://localhost:3000',
    credentials: true,
    methods: ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS'],
    allowedHeaders: ['Content-Type', 'Authorization']
}));

// Middleware de logging para desarrollo
if (process.env.NODE_ENV === 'development') {
    app.use((req, res, next) => {
        console.log(`${new Date().toISOString()} - ${req.method} ${req.path}`);
        next();
    });
}
```

```
// Routes básicas de salud y prueba
app.get('/api/health', (req, res) => {
    res.status(200).json({
        success: true,
        message: '🚀 Servidor funcionando correctamente',
        timestamp: new Date().toISOString(),
        environment: process.env.NODE_ENV
    });
});
```

```
app.get('/api/test', (req, res) => {
    res.status(200).json({
        success: true,
        message: '✅ Endpoint de prueba funcionando',
        data: {
            version: '1.0.0',
            author: 'Tu Nombre',
            description: 'Backend para SPA React - Lab 6'
        }
    });
});
```

```
// Montar rutas principales
app.use('/api/auth', authRoutes);
app.use('/api/products', productRoutes);

// Manejo de rutas no encontradas (404)
app.use('*', (req, res) => {
    res.status(404).json({
        success: false,
        message: `Ruta no encontrada: ${req.originalUrl}`,
        suggestion: 'Verifica la URL o consulta la documentación de la API'
    });
});
```

```
// Middleware de manejo de errores global
app.use((error, req, res, next) => {
    console.error('❌ Error:', error);

    res.status(error.status || 500).json({
        success: false,
        message: error.message || 'Error interno del servidor',
        ...(process.env.NODE_ENV === 'development' && { stack: error.stack })
    });
});

// Obtener puerto desde variables de entorno
const PORT = process.env.PORT || 5000;
```

```
// Iniciar servidor
app.listen(PORT, () => {
    console.log('\n' + '='.repeat(50));
    console.log('🎯 Backend Server iniciado exitosamente!');
    console.log('📊 Información del servidor:');
    console.log(`   → Entorno: ${process.env.NODE_ENV}`);
    console.log(`   → Puerto: ${PORT}`);
    console.log(`   → URL: http://localhost:${PORT}`);
    console.log(`   → Health Check: http://localhost:${PORT}/api/health`);
    console.log('='.repeat(50) + '\n');
});

// Manejo graceful de shutdown
process.on('SIGINT', () => {
    console.log('\n🔴 Apagando servidor gracefulmente...');
    process.exit(0);
});

export default app;
```

2. Crear Estructura Básica de Rutas

Crear archivo routes/auth.js:

```
import express from 'express';
const router = express.Router();

// Ruta de prueba para auth
router.get('/test', (req, res) => {
    res.status(200).json({
        success: true,
        message: 'Auth routes funcionando correctamente',
        endpoints: {
            login: 'POST /api/auth/login',
            register: 'POST /api/auth/register',
            profile: 'GET /api/auth/profile'
        }
    });
});

export default router;
```

Crear archivo routes/products.js:

```
import express from 'express';
const router = express.Router();

// Datos de prueba para productos (temporal)
const mockProducts = [
    {
        id: 1,
        name: 'Producto Demo 1',
        price: 29.99,
        category: 'electronics',
        description: 'Producto de demostración para testing'
    },
    {
        id: 2,
        name: 'Producto Demo 2',
        price: 49.99,
        category: 'clothing',
        description: 'Segundo producto de prueba'
    }
];
```

```
// GET /api/products - Obtener todos los productos
router.get('/', (req, res) => {
    try {
        res.status(200).json({
            success: true,
            count: mockProducts.length,
            data: mockProducts
        });
    } catch (error) {
        res.status(500).json({
            success: false,
            message: 'Error al obtener productos'
        });
    }
});
```

```
// GET /api/products/:id - Obtener producto por ID
router.get('/:id', (req, res) => {
    try {
        const productId = parseInt(req.params.id);
        const product = mockProducts.find(p => p.id === productId);

        if (!product) {
            return res.status(404).json({
                success: false,
                message: 'Producto no encontrado'
            });
        }

        res.status(200).json({
            success: true,
            data: product
        });
    } catch (error) {
        res.status(500).json({
            success: false,
            message: 'Error al obtener el producto'
        });
    }
});

export default router;
```

3. Configurar Variables de Entorno Completas

Actualizar archivo .env con más configuración:

```
# ==================================
# SERVER CONFIGURATION
# ==================================
PORT=5000
NODE_ENV=development
HOST=localhost

# ==================================
# DATABASE CONFIGURATION
# ==================================
MONGODB_URI=mongodb+srv://username:password@cluster0.xxxxx.mongodb.net/your-database-name?retryWrite
s=true&w=majority

# ==================================
# JWT CONFIGURATION
# ==================================
JWT_SECRET=tu_jwt_super_secreto_minimo_32_caracteres_aqui_2024
JWT_EXPIRE=24h
JWT_COOKIE_EXPIRE=30

# ==================================
# FRONTEND CONFIGURATION
# ==================================
FRONTEND_URL=http://localhost:3000
```

```
# ==================================
# API CONFIGURATION
# ==================================
API_VERSION=v1
API_RATE_LIMIT_WINDOW_MS=900000
API_RATE_LIMIT_MAX_REQUESTS=100

# ==================================
# SECURITY CONFIGURATION
# ==================================
BCRYPT_SALT_ROUNDS=12
SESSION_SECRET=tu_session_secret_aqui
```

4. Crear Middleware de Logging Mejorado

Crear archivo middleware/logger.js:

```
const requestLogger = (req, res, next) => {
    const start = Date.now();

    res.on('finish', () => {
        const duration = Date.now() - start;
        console.log(
            `${new Date().toISOString()} - ${req.method} ${req.originalUrl} - ` +
            `Status: ${res.statusCode} - Duration: ${duration}ms - ` +
            `IP: ${req.ip} - User-Agent: ${req.get('User-Agent')?.substring(0, 50)}...`
        );
    });

    next();
};
```

```
const errorLogger = (error, req, res, next) => {
    console.error('❌ ERROR LOG:');
    console.error('Timestamp:', new Date().toISOString());
    console.error('Method:', req.method);
    console.error('URL:', req.originalUrl);
    console.error('Error Message:', error.message);
    console.error('Stack:', error.stack);
    console.error('---'.repeat(20));

    next(error);
};

export { requestLogger, errorLogger };
```

5. Actualizar server.js con Middleware Mejorado

Actualizar server.js con los nuevos middlewares:

```
import express from 'express';
import cors from 'cors';
import dotenv from 'dotenv';
import { requestLogger, errorLogger } from './middleware/logger.js';

// ... resto de imports ...

// Cargar variables de entorno
dotenv.config();

const app = express();

// Middleware esencial
app.use(requestLogger); // ← Nuevo middleware de logging
app.use(express.json({ limit: '10mb' }));
app.use(express.urlencoded({ extended: true }));
```

```
// CORS configuration
app.use(cors({
    origin: process.env.FRONTEND_URL || 'http://localhost:3000',
    credentials: true,
    methods: ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS'],
    allowedHeaders: ['Content-Type', 'Authorization', 'X-Requested-With']
}));

// ... resto del código ...

app.use(errorLogger); // ← Middleware de logging de errores

// ... resto del código ...
```

Verificación del Servidor

1. Ejecutar el Servidor

```
npm run dev
```

2. Probar Endpoints con Thunder Client/Postman

Health Check:

```
GET http://localhost:5000/api/health
```

Response esperado:

```
{
    "success": true,
    "message": "🚀 Servidor funcionando correctamente",
    "timestamp": "2024-01-15T10:30:00.000Z",
    "environment": "development"
}
```

Test Products:

```
GET http://localhost:5000/api/products
```

Response esperado:

```
{
    "success": true,
    "count": 2,
    "data": [
        {
            "id": 1,
            "name": "Producto Demo 1",
            "price": 29.99,
            "category": "electronics",
            "description": "Producto de demostración para testing"
        }
    ]
}
```

3. Probar Ruta No Existente (404)

```
GET http://localhost:5000/api/nonexistent
```

Response esperado:

```
{
    "success": false,
    "message": "Ruta no encontrada: /api/nonexistent",
    "suggestion": "Verifica la URL o consulta la documentación de la API"
}
```

Solución de Problemas Comunes

Error: CORS bloqueado

Solución**:** Verificar que FRONTEND_URL esté correcto en .env

```
// Verificar en consola:
console.log('Frontend URL:', process.env.FRONTEND_URL);
```

Error: Body parsing fallido

Solución**:** Verificar que los headers sean correctos

```
Content-Type: application/json
```

Error: Puerto en uso

Solución: Cambiar puerto o terminar proceso

```
# Buscar proceso usando puerto 5000
lsof -ti:5000

# Terminar proceso
kill -9 $(lsof -ti:5000)

# O cambiar puerto en .env
PORT=5001
```

Paso 3: Conexión a MongoDB Atlas

El objetivo de este paso es establecer una conexión segura y eficiente entre nuestro servidor Express y MongoDB Atlas, configurar los modelos de datos con Mongoose, e implementar un sistema robusto de manejo de conexiones. Esto nos

permitirá persistir datos de usuarios, productos y mensajes de contacto de forma profesional.

Crear Cuenta en MongoDB Atlas y Obtener Cadena de Conexión

Pasos para configurar MongoDB Atlas:

1. Registrarse en MongoDB Atlas
2. Crear un nuevo cluster (elegir tier gratuito M0)
3. Configurar seguridad:
    - Crear usuario de base de datos
    - Configurar IP whitelist (0.0.0.0/0 para desarrollo)
4. Obtener connection string:
    - Ir a Cluster → Connect → Connect your application
    - Copiar la cadena de conexión

2. Configurar el Archivo de Conexión a la Base de Datos

Crear archivo config/database.js**:**

```
import mongoose from 'mongoose';
import dotenv from 'dotenv';

// Cargar variables de entorno
dotenv.config();

class Database {
    constructor() {
        this.connection = null;
        this.connect();
    }

    async connect() {
        try {
            // Opciones de conexión para Mongoose 7+
            const options = {
                maxPoolSize: 10, // Máximo de conexiones simultáneas
                serverSelectionTimeoutMS: 5000, // Timeout para seleccionar servidor
                socketTimeoutMS: 45000, // Timeout de inactividad
                family: 4, // Usar IPv4
            };

            console.log('🔄 Intentando conectar a MongoDB Atlas...');

            this.connection = await mongoose.connect(
                process.env.MONGODB_URI,
                options
            );

            console.log('✅ Conectado a MongoDB Atlas exitosamente!');
            console.log(`   → Base de datos: ${this.connection.connection.name}`);
            console.log(`   → Host: ${this.connection.connection.host}`);
            console.log(`   → Puerto: ${this.connection.connection.port}`);

        } catch (error) {
            console.error('❌ Error conectando a MongoDB:', error.message);
            process.exit(1); // Salir del proceso con error
        }
    }
```

```
    async disconnect() {
        try {
            if (this.connection) {
                await mongoose.disconnect();
                console.log('✅ Desconectado de MongoDB Atlas');
            }
        } catch (error) {
            console.error('❌ Error desconectando de MongoDB:', error.message);
        }
    }

    getConnection() {
        return this.connection;
    }

    getStatus() {
        return {
            connected: mongoose.connection.readyState === 1,
            state: this.getStateString(mongoose.connection.readyState),
            dbName: mongoose.connection.name,
            host: mongoose.connection.host,
            port: mongoose.connection.port
        };
    }
```

```
    getStateString(state) {
        const states = {
            0: 'disconnected',
            1: 'connected',
            2: 'connecting',
            3: 'disconnecting',
            99: 'uninitialized'
        };
        return states[state] || 'unknown';
    }
}

// Manejo de eventos de conexión
mongoose.connection.on('connected', () => {
    console.log('📊 Mongoose conectado a MongoDB Atlas');
});

mongoose.connection.on('error', (err) => {
    console.error('❌ Error de conexión de Mongoose:', err);
});

mongoose.connection.on('disconnected', () => {
    console.log('⚠️  Mongoose desconectado de MongoDB');
});
```

```
// Manejar cierre graceful de la aplicación
process.on('SIGINT', async () => {
    await mongoose.connection.close();
    console.log('✅ Conexión a MongoDB cerrada por terminación de app');
    process.exit(0);
});

// Crear y exportar instancia única de Database
const database = new Database();
export default database;
```

3. Actualizar Variables de Entorno para MongoDB

Actualizar archivo .env con la conexión real:

```
# ==================================
# MONGODB ATLAS CONFIGURATION
# ==================================
MONGODB_URI=mongodb+srv://usuario:password@cluster0.xxxxx.mongodb.net/nombre-db?retryWrites=true&w=majority

# ==================================
# DATABASE SETTINGS
# ==================================
DB_NAME=spa_database
DB_POOL_SIZE=10
DB_CONNECTION_TIMEOUT=5000
DB_SOCKET_TIMEOUT=45000
```

4. Crear Modelos de Datos con Mongoose

Crear modelo de Usuario (models/User.js):

```
import mongoose from 'mongoose';
import bcrypt from 'bcryptjs';

const userSchema = new mongoose.Schema({
    name: {
        type: String,
        required: [true, 'El nombre es requerido'],
        trim: true,
        maxlength: [50, 'El nombre no puede exceder 50 caracteres'],
        minlength: [2, 'El nombre debe tener al menos 2 caracteres']
    },
    email: {
        type: String,
        required: [true, 'El email es requerido'],
        unique: true,
        lowercase: true,
        match: [
            /^\w+([.-]?\w+)*@\w+([.-]?\w+)*(\.\w{2,3})+$/,
            'Por favor ingresa un email válido'
        ]
```

```
    },
    password: {
        type: String,
        required: [true, 'La contraseña es requerida'],
        minlength: [6, 'La contraseña debe tener al menos 6 caracteres'],
        select: false // No incluir en queries por defecto
    },
    role: {
        type: String,
        enum: ['user', 'admin'],
        default: 'user'
    },
    isActive: {
        type: Boolean,
        default: true
    },
    lastLogin: {
        type: Date,
        default: Date.now
    }
```

```
}, {
    timestamps: true, // Crea createdAt y updatedAt automáticamente
    toJSON: { virtuals: true },
    toObject: { virtuals: true }
});

// Middleware pre-save para hashear password
userSchema.pre('save', async function(next) {
    // Solo hashear si el password fue modificado
    if (!this.isModified('password')) return next();

    try {
        const salt = await bcrypt.genSalt(Number(process.env.BCRYPT_SALT_ROUNDS) || 12);
        this.password = await bcrypt.hash(this.password, salt);
        next();
    } catch (error) {
        next(error);
    }
});
```

```
// Middleware pre-save para hashear password
userSchema.pre('save', async function(next) {
    // Solo hashear si el password fue modificado
    if (!this.isModified('password')) return next();

    try {
        const salt = await bcrypt.genSalt(Number(process.env.BCRYPT_SALT_ROUNDS) || 12);
        this.password = await bcrypt.hash(this.password, salt);
        next();
    } catch (error) {
        next(error);
    }
});
```

```
// Método para comparar passwords
userSchema.methods.comparePassword = async function(candidatePassword) {
    return await bcrypt.compare(candidatePassword, this.password);
};

// Virtual para nombre completo
userSchema.virtual('fullName').get(function() {
    return `${this.name}`;
});

// Método estático para buscar por email
userSchema.statics.findByEmail = function(email) {
    return this.findOne({ email: email.toLowerCase() });
};

// Index para mejorar performance en búsquedas
userSchema.index({ email: 1 });
userSchema.index({ createdAt: -1 });

const User = mongoose.model('User', userSchema);
export default User;
```

Crear modelo de Producto (models/Product.js):

```
import mongoose from 'mongoose';

const productSchema = new mongoose.Schema({
    name: {
        type: String,
        required: [true, 'El nombre del producto es requerido'],
        trim: true,
        maxlength: [100, 'El nombre no puede exceder 100 caracteres']
    },
    description: {
        type: String,
        required: [true, 'La descripción es requerida'],
        maxlength: [1000, 'La descripción no puede exceder 1000 caracteres']
    },
    price: {
        type: Number,
        required: [true, 'El precio es requerido'],
        min: [0, 'El precio no puede ser negativo']
    },
    originalPrice: {
        type: Number,
        min: [0, 'El precio original no puede ser negativo']
    },
```

```
    category: {
        type: String,
        required: [true, 'La categoría es requerida'],
        enum: ['electronics', 'clothing', 'books', 'home', 'sports', 'other']
    },
    image: {
        type: String,
        default: 'https://via.placeholder.com/300x300?text=Product+Image'
    },
    images: [{
        type: String
    }],
    stock: {
        type: Number,
        required: true,
        min: [0, 'El stock no puede ser negativo'],
        default: 0
    },
    sku: {
        type: String,
        unique: true,
        sparse: true
    },
```

```
    sku: {
        type: String,
        unique: true,
        sparse: true
    },
    brand: {
        type: String,
        trim: true
    },
    rating: {
        average: {
            type: Number,
            min: 0,
            max: 5,
            default: 0
        },
        count: {
            type: Number,
            default: 0
        }
    },
```

```
    features: [{
        type: String
    }],
    specifications: {
        type: Map,
        of: String
    },
    isActive: {
        type: Boolean,
        default: true
    },
    tags: [{
        type: String,
        trim: true
    }]
}, {
    timestamps: true,
    toJSON: { virtuals: true },
    toObject: { virtuals: true }
});
```

```
// Virtual para discount percentage
productSchema.virtual('discountPercentage').get(function() {
    if (this.originalPrice && this.originalPrice > this.price) {
        return Math.round(((this.originalPrice - this.price) / this.originalPrice) * 100);
    }
    return 0;
});

// Virtual para inStock
productSchema.virtual('inStock').get(function() {
    return this.stock > 0;
});

// Indexes para búsquedas eficientes
productSchema.index({ name: 'text', description: 'text' });
productSchema.index({ category: 1, price: 1 });
productSchema.index({ createdAt: -1 });
productSchema.index({ 'rating.average': -1 });

// Método estático para productos activos
productSchema.statics.getActiveProducts = function() {
    return this.find({ isActive: true });
};
```

```
// Virtual para discount percentage
productSchema.virtual('discountPercentage').get(function() {
    if (this.originalPrice && this.originalPrice > this.price) {
        return Math.round(((this.originalPrice - this.price) / this.originalPrice) * 100);
    }
    return 0;
});

// Virtual para inStock
productSchema.virtual('inStock').get(function() {
    return this.stock > 0;
});

// Indexes para búsquedas eficientes
productSchema.index({ name: 'text', description: 'text' });
productSchema.index({ category: 1, price: 1 });
productSchema.index({ createdAt: -1 });
productSchema.index({ 'rating.average': -1 });

// Método estático para productos activos
productSchema.statics.getActiveProducts = function() {
    return this.find({ isActive: true });
};

const Product = mongoose.model('Product', productSchema);
export default Product;
```

5. Integrar la Conexión en el Servidor Principal

Actualizar server.js para incluir la conexión a MongoDB:

```
import express from 'express';
import cors from 'cors';
import dotenv from 'dotenv';
import database from './config/database.js'; // ← Nueva importación
import { requestLogger, errorLogger } from './middleware/logger.js';

// Importar modelos para verificar conexión
import './models/User.js';
import './models/Product.js';
import './models/Cart.js';

// ... resto de imports ...

const app = express();
```

```
// Verificar conexión a la base de datos al iniciar
app.use((req, res, next) => {
    const dbStatus = database.getStatus();
    if (!dbStatus.connected) {
        return res.status(503).json({
            success: false,
            message: 'Servicio no disponible - Base de datos desconectada',
            error: 'database_connection_error'
        });
    }
    next();
});

// ... resto del middleware ...
```

```
// Endpoint adicional para verificar estado de la DB
app.get('/api/db-status', (req, res) => {
    const status = database.getStatus();
    res.status(200).json({
        success: true,
        database: status
    });
});

// ... resto del código del servidor ...
```

6. Crear Script de Población de Datos de Prueba

Crear archivo scripts/seedDatabase.js:

```
import mongoose from 'mongoose';
import dotenv from 'dotenv';
import Product from '../models/Product.js';
import User from '../models/User.js';

dotenv.config();
```

```
const seedProducts = [
    {
        name: 'Smartphone Premium',
        description: 'Último modelo con cámara de 108MP y 5G',
        price: 899.99,
        originalPrice: 999.99,
        category: 'electronics',
        stock: 50,
        brand: 'TechBrand',
        features: ['5G', '108MP Camera', '8GB RAM', '256GB Storage'],
        rating: { average: 4.5, count: 120 }
    },
    {
        name: 'Laptop Ultradelgada',
        description: 'Laptop para trabajo y entretenimiento',
        price: 1299.99,
        category: 'electronics',
        stock: 25,
        brand: 'ComputerPro',
        features: ['Intel i7', '16GB RAM', '1TB SSD', '15.6" Display'],
        rating: { average: 4.3, count: 85 }
    }
];
```

```
const seedUsers = [
    {
        name: 'Admin User',
        email: 'admin@example.com',
        password: 'password123',
        role: 'admin'
    },
    {
        name: 'John Doe',
        email: 'john@example.com',
        password: 'password123',
        role: 'user'
    }
];
```

```
const seedDatabase = async () => {
    try {
        await mongoose.connect(process.env.MONGODB_URI);
        console.log('✅ Conectado a MongoDB para seeding');

        // Limpiar colecciones existentes
        await Product.deleteMany({});
        await User.deleteMany({});

        console.log('🧹 Colecciones limpiadas');

        // Insertar datos de prueba
        const products = await Product.insertMany(seedProducts);
        const users = await User.insertMany(seedUsers);

        console.log('🌱 Datos de prueba insertados:');
        console.log(`   → Productos: ${products.length}`);
        console.log(`   → Usuarios: ${users.length}`);

        process.exit(0);
    } catch (error) {
        console.error('❌ Error en seeding:', error);
        process.exit(1);
    }
};
```

```
// Ejecutar solo si se llama directamente
if (import.meta.url === `file://${process.argv[1]}`) {
    seedDatabase();
}

export { seedDatabase, seedProducts, seedUsers };
```

Agregar script al package.json:

```
{
  "scripts": {
    "start": "node server.js",
    "dev": "nodemon server.js",
    "seed": "node scripts/seedDatabase.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  }
}
```

Verificación de la Conexión

1. Ejecutar el Servidor y Verificar Conexión

```
npm run dev
```

Output esperado en consola:

```
Intentando conectar a MongoDB Atlas...
Conectado a MongoDB Atlas exitosamente!
→ Base de datos: spa_database
→ Host: cluster0-shard-00-00.xxxxx.mongodb.net
→ Puerto: 27017
Mongoose conectado a MongoDB Atlas
```

2. Probar Endpoint de Estado de la DB

```
GET http://localhost:5000/api/db-status
```

Response esperado:

```
{
    "success": true,
    "database": {
        "connected": true,
        "state": "connected",
        "dbName": "spa_database",
        "host": "cluster0-shard-00-00.xxxxx.mongodb.net",
        "port": 27017
    }
}
```

3. Poblar la Base de Datos con Datos de Prueba

```
npm run seed
```

Output esperado:

```
Conectado a MongoDB para seeding
Colecciones limpiadas
Datos de prueba insertados:
→ Productos: 2
→ Usuarios: 2
```

Solución de Problemas Comunes

Error: Authentication failed

Solución**:** Verificar usuario y password en la connection string

```
# La cadena debe verse así:
mongodb+srv://usuario:password@cluster0.xxxxx.mongodb.net/
```

Error: Network timeout

Solución: Verificar whitelist de IPs en MongoDB Atlas

- Ir a Network Access → Add IP Address → 0.0.0.0/0

Error: Invalid schema

Solución: Verificar que la connection string incluce opciones

```
// Debe incluir retryWrites y w=majority
mongodb+srv://...mongodb.net/dbname?retryWrites=true&w=majority
```

Paso 4: Implementar Sistema de Autenticación JWT

El objetivo de este paso es implementar un sistema completo de autenticación basado en JSON Web Tokens (JWT) que permita a los usuarios registrarse, iniciar sesión, y acceder a recursos protegidos. Incluiremos middleware de autenticación, manejo seguro de contraseñas, y endpoints RESTful para gestionar la autenticación de usuarios.

Acciones Específicas

1. Crear Utilities para JWT y Validaciones

Crear archivo utils/authUtils.js:

```
import jwt from 'jsonwebtoken';
import bcrypt from 'bcryptjs';

// Generar JWT Token
export const generateToken = (userId, role = 'user') => {
    return jwt.sign(
        {
            userId,
            role,
            iss: 'spa-backend',
            aud: 'spa-frontend'
        },
        process.env.JWT_SECRET,
        {
            expiresIn: process.env.JWT_EXPIRE || '24h',
            algorithm: 'HS256'
        }
    );
};
```

```
// Verificar JWT Token
export const verifyToken = (token) => {
    try {
        return jwt.verify(token, process.env.JWT_SECRET, { algorithms: ['HS256'] });
    } catch (error) {
        throw new Error('Token inválido o expirado');
    }
};

// Hashear contraseña
export const hashPassword = async (password) => {
    const saltRounds = parseInt(process.env.BCRYPT_SALT_ROUNDS) || 12;
    return await bcrypt.hash(password, saltRounds);
};

// Comparar contraseña con hash
export const comparePassword = async (password, hashedPassword) => {
    return await bcrypt.compare(password, hashedPassword);
};
```

```
// Extraer token de headers
export const extractTokenFromHeader = (authHeader) => {
    if (!authHeader) {
        throw new Error('Authorization header required');
    }

    const parts = authHeader.split(' ');

    if (parts.length !== 2 || parts[0] !== 'Bearer') {
        throw new Error('Authorization header format: Bearer <token>');
    }

    return parts[1];
};
```

```
// Generar respuesta de autenticación estándar
export const generateAuthResponse = (user, token) => {
    return {
        success: true,
        message: 'Autenticación exitosa',
        data: {
            user: {
                id: user._id,
                name: user.name,
                email: user.email,
                role: user.role,
                lastLogin: user.lastLogin
            },
            token,
            expiresIn: process.env.JWT_EXPIRE || '24h'
        }
    };
};
```

Crear archivo utils/validationUtils.js:

```
import validator from 'validator';

// Validar email
export const validateEmail = (email) => {
    if (!email) {
        return 'Email es requerido';
    }

    if (!validator.isEmail(email)) {
        return 'Email no tiene formato válido';
    }

    return null;
};
```

```
// Validar password
export const validatePassword = (password) => {
    if (!password) {
        return 'Contraseña es requerida';
    }

    if (password.length < 6) {
        return 'Contraseña debe tener al menos 6 caracteres';
    }

    if (!/[A-Z]/.test(password)) {
        return 'Contraseña debe contener al menos una mayúscula';
    }

    if (!/[0-9]/.test(password)) {
        return 'Contraseña debe contener al menos un número';
    }

    return null;
};
```

```
// Validar nombre
export const validateName = (name) => {
    if (!name) {
        return 'Nombre es requerido';
    }

    if (name.length < 2) {
        return 'Nombre debe tener al menos 2 caracteres';
    }

    if (name.length > 50) {
        return 'Nombre no puede exceder 50 caracteres';
    }

    if (!/^[a-zA-ZáéíóúÁÉÍÓÚñÑ\s]+$/.test(name)) {
        return 'Nombre solo puede contener letras y espacios';
    }

    return null;
};
```

```
// Validar datos de registro
export const validateRegistrationData = (data) => {
    const errors = {};

    const nameError = validateName(data.name);
    if (nameError) errors.name = nameError;

    const emailError = validateEmail(data.email);
    if (emailError) errors.email = emailError;

    const passwordError = validatePassword(data.password);
    if (passwordError) errors.password = passwordError;

    return {
        isValid: Object.keys(errors).length === 0,
        errors
    };
};
```

```
// Validar datos de login
export const validateLoginData = (data) => {
    const errors = {};

    const emailError = validateEmail(data.email);
    if (emailError) errors.email = emailError;

    if (!data.password) {
        errors.password = 'Contraseña es requerida';
    }

    return {
        isValid: Object.keys(errors).length === 0,
        errors
    };
};
```

2. Crear Middleware de Autenticación

Crear archivo middleware/authMiddleware.js:

```
import { verifyToken, extractTokenFromHeader } from '../utils/authUtils.js';
import User from '../models/User.js';

// Middleware para verificar JWT
export const authenticateToken = async (req, res, next) => {
    try {
        const authHeader = req.headers.authorization;
        const token = extractTokenFromHeader(authHeader);

        const decoded = verifyToken(token);

        // Verificar que el usuario aún existe
        const user = await User.findById(decoded.userId).select('-password');
        if (!user) {
            return res.status(401).json({
                success: false,
                message: 'Usuario no encontrado - token inválido'
            });
        }

        if (!user.isActive) {
            return res.status(401).json({
                success: false,
                message: 'Cuenta desactivada'
            });
        }
```

```
        // Agregar usuario al request
        req.user = user;
        next();

    } catch (error) {
        console.error('Error en autenticación:', error.message);

        return res.status(401).json({
            success: false,
            message: 'No autorizado - token inválido',
            error: error.message
        });
    }
};
```

```
// Middleware para verificar roles
export const requireRole = (allowedRoles) => {
    return (req, res, next) => {
        if (!req.user) {
            return res.status(401).json({
                success: false,
                message: 'Autenticación requerida'
            });
        }

        if (!allowedRoles.includes(req.user.role)) {
            return res.status(403).json({
                success: false,
                message: 'Permisos insuficientes para esta acción'
            });
        }

        next();
    };
};
```

```
// Middleware opcional de autenticación (para rutas públicas/privadas)
export const optionalAuth = async (req, res, next) => {
    try {
        const authHeader = req.headers.authorization;
        if (!authHeader) {
            return next(); // Continuar sin usuario
        }

        const token = extractTokenFromHeader(authHeader);
        const decoded = verifyToken(token);

        const user = await User.findById(decoded.userId).select('-password');
        if (user && user.isActive) {
            req.user = user;
        }

        next();
    } catch (error) {
        // Si hay error en el token, continuar sin autenticación
        next();
    }
};
```

3. Crear Controladores de Autenticación

Crear archivo controllers/authController.js:

```
import User from '../models/User.js';
import {
    generateToken,
    generateAuthResponse,
    comparePassword,
    hashPassword
} from '../utils/authUtils.js';
import {
    validateRegistrationData,
    validateLoginData
} from '../utils/validationUtils.js';

// Registrar nuevo usuario
export const register = async (req, res) => {
    try {
        const { name, email, password } = req.body;
```

```
// Validar datos de entrada
const validation = validateRegistrationData({ name, email, password });
if (!validation.isValid) {
    return res.status(400).json({
        success: false,
        message: 'Datos de registro inválidos',
        errors: validation.errors
    });
}

// Verificar si el usuario ya existe
const existingUser = await User.findOne({ email: email.toLowerCase() });
if (existingUser) {
    return res.status(409).json({
        success: false,
        message: 'El email ya está registrado',
        error: 'EMAIL_ALREADY_EXISTS'
    });
}
```

```
        // Crear nuevo usuario
        const user = new User({
            name: name.trim(),
            email: email.toLowerCase(),
            password,
            role: 'user'
        });

        await user.save();

        // Generar token JWT
        const token = generateToken(user._id, user.role);

        // Actualizar último login
        user.lastLogin = new Date();
        await user.save();

        res.status(201).json(generateAuthResponse(user, token));

    } catch (error) {
        console.error('Error en registro:', error);
        res.status(500).json({
            success: false,
            message: 'Error interno del servidor durante el registro',
            error: process.env.NODE_ENV === 'development' ? error.message : undefined
        });
    }
};
```

```
// Login de usuario
export const login = async (req, res) => {
    try {
        const { email, password } = req.body;

        // Validar datos de entrada
        const validation = validateLoginData({ email, password });
        if (!validation.isValid) {
            return res.status(400).json({
                success: false,
                message: 'Datos de login inválidos',
                errors: validation.errors
            });
        }
```

```
// Buscar usuario incluyendo password
const user = await User.findOne({ email: email.toLowerCase() })
    .select('+password');

if (!user) {
    return res.status(401).json({
        success: false,
        message: 'Credenciales inválidas',
        error: 'INVALID_CREDENTIALS'
    });
}

if (!user.isActive) {
    return res.status(401).json({
        success: false,
        message: 'Cuenta desactivada',
        error: 'ACCOUNT_DEACTIVATED'
    });
}
```

```
// Verificar contraseña
const isPasswordValid = await comparePassword(password, user.password);
if (!isPasswordValid) {
    return res.status(401).json({
        success: false,
        message: 'Credenciales inválidas',
        error: 'INVALID_CREDENTIALS'
    });
}

// Generar token JWT
const token = generateToken(user._id, user.role);

// Actualizar último login
user.lastLogin = new Date();
await user.save();

// Eliminar password de la respuesta
user.password = undefined;

res.status(200).json(generateAuthResponse(user, token));
```

```
    } catch (error) {
        console.error('Error en login:', error);
        res.status(500).json({
            success: false,
            message: 'Error interno del servidor durante el login',
            error: process.env.NODE_ENV === 'development' ? error.message : undefined
        });
    }
};
```

```
// Obtener perfil de usuario actual
export const getProfile = async (req, res) => {
    try {
        const user = await User.findById(req.user._id);

        res.status(200).json({
            success: true,
            data: {
                user: {
                    id: user._id,
                    name: user.name,
                    email: user.email,
                    role: user.role,
                    lastLogin: user.lastLogin,
                    createdAt: user.createdAt
                }
            }
        });
```

```
    } catch (error) {
        console.error('Error obteniendo perfil:', error);
        res.status(500).json({
            success: false,
            message: 'Error obteniendo perfil de usuario'
        });
    }
};
```

```
// Actualizar perfil de usuario
export const updateProfile = async (req, res) => {
    try {
        const { name } = req.body;
        const updates = {};

        if (name) {
            const nameError = validateName(name);
            if (nameError) {
                return res.status(400).json({
                    success: false,
                    message: nameError
                });
            }
            updates.name = name.trim();
        }
```

```
        const user = await User.findByIdAndUpdate(
            req.user._id,
            updates,
            { new: true, runValidators: true }
        );

        res.status(200).json({
            success: true,
            message: 'Perfil actualizado exitosamente',
            data: {
                user: {
                    id: user._id,
                    name: user.name,
                    email: user.email,
                    role: user.role
                }
            }
        });
```

```
    } catch (error) {
        console.error('Error actualizando perfil:', error);
        res.status(500).json({
            success: false,
            message: 'Error actualizando perfil de usuario'
        });
    }
};

// Verificar token (para frontend)
export const verifyAuth = async (req, res) => {
    try {
        res.status(200).json({
            success: true,
            message: 'Token válido',
            data: {
                user: {
                    id: req.user._id,
                    name: req.user.name,
                    email: req.user.email,
                    role: req.user.role
                },
                valid: true
            }
        });
```

```
    } catch (error) {
        res.status(401).json({
            success: false,
            message: 'Token inválido',
            valid: false
        });
    }
};

// Logout (manejado en frontend, pero podemos invalidar token si es necesario)
export const logout = async (req, res) => {
    try {
        // En un sistema más avanzado, podríamos invalidar el token
        // Para este ejemplo, el logout se maneja en el frontend

        res.status(200).json({
            success: true,
            message: 'Sesión cerrada exitosamente'
        });
```

```
        res.status(200).json({
            success: true,
            message: 'Sesión cerrada exitosamente'
        });

    } catch (error) {
        console.error('Error en logout:', error);
        res.status(500).json({
            success: false,
            message: 'Error cerrando sesión'
        });
    }
};
```

4. Crear Rutas de Autenticación

Actualizar archivo routes/auth.js:

```
import express from 'express';
import {
    register,
    login,
    getProfile,
    updateProfile,
    verifyAuth,
    logout
} from '../controllers/authController.js';
import { authenticateToken, optionalAuth } from '../middleware/authMiddleware.js';

const router = express.Router();

// Rutas públicas
router.post('/register', register);
router.post('/login', login);
router.get('/verify', optionalAuth, verifyAuth); // Verificación opcional

// Rutas protegidas (requieren autenticación)
router.get('/profile', authenticateToken, getProfile);
router.put('/profile', authenticateToken, updateProfile);
router.post('/logout', authenticateToken, logout);
```

```
// Ruta de información (pública)
router.get('/info', (req, res) => {
    res.status(200).json({
        success: true,
        message: 'Auth API funcionando',
        endpoints: {
            register: 'POST /api/auth/register',
            login: 'POST /api/auth/login',
            profile: 'GET /api/auth/profile (protected)',
            verify: 'GET /api/auth/verify'
        },
        security: {
            type: 'JWT',
            algorithm: 'HS256'
        }
    });
});

export default router;
```

5. Actualizar Servidor Principal con Nuevas Rutas

Actualizar server.js para incluir rutas de autenticación:

```
import express from 'express';
import cors from 'cors';
import dotenv from 'dotenv';
import database from './config/database.js';
import { requestLogger, errorLogger } from './middleware/logger.js';

// Importar rutas
import authRoutes from './routes/auth.js';
import productRoutes from './routes/products.js';

// Cargar variables de entorno
dotenv.config();

const app = express();

// Middleware esencial
app.use(requestLogger);
app.use(express.json({ limit: '10mb' }));
app.use(express.urlencoded({ extended: true }));
```

```
// CORS configuration
app.use(cors({
    origin: process.env.FRONTEND_URL || 'http://localhost:3000',
    credentials: true,
    methods: ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS'],
    allowedHeaders: ['Content-Type', 'Authorization', 'X-Requested-With']
}));

// Verificar conexión a la base de datos
app.use((req, res, next) => {
    const dbStatus = database.getStatus();
    if (!dbStatus.connected) {
        return res.status(503).json({
            success: false,
            message: 'Servicio no disponible - Base de datos desconectada',
            error: 'database_connection_error'
        });
    }
    next();
});
```

```
// Routes básicas
app.get('/api/health', (req, res) => {
    res.status(200).json({
        success: true,
        message: '🚀 Servidor funcionando correctamente',
        timestamp: new Date().toISOString(),
        environment: process.env.NODE_ENV,
        database: database.getStatus()
    });
});

// Montar rutas principales
app.use('/api/auth', authRoutes);
app.use('/api/products', productRoutes);

// ... resto del código del servidor ...
```

6. Crear Ejemplos de Uso para Testing

Crear archivo examples/authExamples.http (para Thunder Client):

```
### Health Check
GET http://localhost:5000/api/health

### Get Auth Info
GET http://localhost:5000/api/auth/info

### Register New User
POST http://localhost:5000/api/auth/register
Content-Type: application/json

{
    "name": "Juan Pérez",
    "email": "juan@example.com",
    "password": "Password123"
}

### Login User
POST http://localhost:5000/api/auth/login
Content-Type: application/json

{
    "email": "juan@example.com",
    "password": "Password123"
}
```

```
### Get Profile (Protected)
GET http://localhost:5000/api/auth/profile
Authorization: Bearer {{auth_token}}

### Verify Token
GET http://localhost:5000/api/auth/verify
Authorization: Bearer {{auth_token}}

### Update Profile
PUT http://localhost:5000/api/auth/profile
Authorization: Bearer {{auth_token}}
Content-Type: application/json

{
    "name": "Juan Carlos Pérez"
}

### Logout
POST http://localhost:5000/api/auth/logout
Authorization: Bearer {{auth_token}}
```

Verificación del Sistema de Autenticación

1. Probar Endpoints de Autenticación

```
npm run dev
```

2. Test de Flujo Completo

1. Registro: Crear un nuevo usuario
2. Login: Obtener token JWT
3. Perfil: Acceder a ruta protegida con token
4. Verificación: Validar token
5. Logout: Cerrar sesión

3. Response Esperados

Registro exitoso:


```
{
    "success": true,
    "message": "Autenticación exitosa",
    "data": {
        "user": {
            "id": "507f1f77bcf86cd799439011",
            "name": "Juan Pérez",
            "email": "juan@example.com",
            "role": "user",
            "lastLogin": "2024-01-15T10:30:00.000Z"
        },
        "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
        "expiresIn": "24h"
    }
}
```


Login exitoso:

```
{
    "success": true,
    "message": "Autenticación exitosa",
    "data": {
        "user": {
            "id": "507f1f77bcf86cd799439011",
            "name": "Juan Pérez",
            "email": "juan@example.com",
            "role": "user",
            "lastLogin": "2024-01-15T10:30:00.000Z"
        },
        "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
        "expiresIn": "24h"
    }
}
```


Solución de Problemas Comunes

Error: JWT Secret not defined

Solución: Verificar variable JWT_SECRET en .env

```
JWT_SECRET=tu_jwt_super_secreto_minimo_32_caracteres
```

Error: Password hashing failed

Solución: Verificar que bcrypt esté instalado correctamente

```
npm list bcryptjs
```

Error: CORS en frontend

Solución: Verificar configuración CORS en server.js

```
origin: process.env.FRONTEND_URL || 'http://localhost:3000'
```

Paso 5: Implementar CRUD para Productos

El objetivo de este paso es implementar un sistema completo de operaciones CRUD (Create, Read, Update, Delete) para productos, incluyendo endpoints RESTful, validaciones de datos, búsquedas avanzadas, paginación, y control de acceso basado en roles. Esto permitirá gestionar el catálogo de productos de manera profesional.

Acciones Específicas

1. Crear Utilities para Productos

Crear archivo utils/productUtils.js:

```
import Product from '../models/Product.js';

// Constantes para paginación y ordenamiento
export const PRODUCT_DEFAULT_LIMIT = 10;
export const PRODUCT_MAX_LIMIT = 50;
export const SORT_OPTIONS = {
    'price-asc': { price: 1 },
    'price-desc': { price: -1 },
    'name-asc': { name: 1 },
    'name-desc': { name: -1 },
    'newest': { createdAt: -1 },
    'oldest': { createdAt: 1 },
    'rating': { 'rating.average': -1 }
};
```

```
// Validar datos de producto
export const validateProductData = (productData, isUpdate = false) => {
    const errors = {};

    if (!isUpdate || productData.name !== undefined) {
        if (!productData.name || productData.name.trim().length === 0) {
            errors.name = 'El nombre del producto es requerido';
        } else if (productData.name.length > 100) {
            errors.name = 'El nombre no puede exceder 100 caracteres';
        }
    }

    if (!isUpdate || productData.description !== undefined) {
        if (!productData.description || productData.description.trim().length === 0) {
            errors.description = 'La descripción es requerida';
        } else if (productData.description.length > 1000) {
            errors.description = 'La descripción no puede exceder 1000 caracteres';
        }
    }
```

```
    if (!isUpdate || productData.price !== undefined) {
        if (productData.price === undefined || productData.price === null) {
            errors.price = 'El precio es requerido';
        } else if (typeof productData.price !== 'number' || productData.price < 0) {
            errors.price = 'El precio debe ser un número positivo';
        }
    }

    if (!isUpdate || productData.category !== undefined) {
        const validCategories = ['electronics', 'clothing', 'books', 'home', 'sports', 'other'];
        if (productData.category && !validCategories.includes(productData.category)) {
            errors.category = 'Categoría no válida';
        }
    }

    if (productData.stock !== undefined && (typeof productData.stock !== 'number' || productData.sto
ck < 0)) {
        errors.stock = 'El stock debe ser un número positivo';
    }
```

```
    return {
        isValid: Object.keys(errors).length === 0,
        errors
    };
};

// Construir query de búsqueda
export const buildProductQuery = (filters = {}) => {
    let query = { isActive: true };

    // Filtro por categoría
    if (filters.category && filters.category !== 'all') {
        query.category = filters.category;
    }

    // Filtro por rango de precios
    if (filters.minPrice || filters.maxPrice) {
        query.price = {};
        if (filters.minPrice) query.price.$gte = parseFloat(filters.minPrice);
        if (filters.maxPrice) query.price.$lte = parseFloat(filters.maxPrice);
    }
```

```
    // Filtro por stock disponible
    if (filters.inStock === 'true') {
        query.stock = { $gt: 0 };
    }

    // Búsqueda por texto
    if (filters.search) {
        query.$text = { $search: filters.search };
    }

    // Filtro por rating mínimo
    if (filters.minRating) {
        query['rating.average'] = { $gte: parseFloat(filters.minRating) };
    }

    // Filtro por tags
    if (filters.tags) {
        const tagsArray = Array.isArray(filters.tags) ? filters.tags : [filters.tags];
        query.tags = { $in: tagsArray };
    }

    return query;
};
```

```
// Construir opciones de paginación y ordenamiento
export const buildPaginationOptions = (queryParams = {}) => {
    const page = Math.max(1, parseInt(queryParams.page) || 1);
    const limit = Math.min(
        PRODUCT_MAX_LIMIT,
        Math.max(1, parseInt(queryParams.limit) || PRODUCT_DEFAULT_LIMIT)
    );
    const skip = (page - 1) * limit;

    const sort = SORT_OPTIONS[queryParams.sort] || { createdAt: -1 };

    return {
        page,
        limit,
        skip,
        sort
    };
};
```

```
// Calcular metadatos de paginación
export const calculatePaginationMetadata = (total, page, limit) => {
    const totalPages = Math.ceil(total / limit);

    return {
        total,
        totalPages,
        currentPage: page,
        hasNext: page < totalPages,
        hasPrev: page > 1,
        nextPage: page < totalPages ? page + 1 : null,
        prevPage: page > 1 ? page - 1 : null
    };
};
```

```
// Formatear respuesta de productos
export const formatProductResponse = (products, pagination = null) => {
    const response = {
        success: true,
        data: {
            products: products.map(product => ({
                id: product._id,
                name: product.name,
                description: product.description,
                price: product.price,
                originalPrice: product.originalPrice,
                discountPercentage: product.discountPercentage,
                category: product.category,
                image: product.image,
                images: product.images,
                stock: product.stock,
                inStock: product.inStock,
                sku: product.sku,
                brand: product.brand,
                rating: product.rating,
                features: product.features,
                specifications: product.specifications,
                tags: product.tags,
                createdAt: product.createdAt,
                updatedAt: product.updatedAt
            }))
        }
    };

    if (pagination) {
        response.data.pagination = pagination;
    }

    return response;
};
```

2. Crear Controladores de Productos

Crear archivo controllers/productController.js:

```
import Product from '../models/Product.js';
import {
    validateProductData,
    buildProductQuery,
    buildPaginationOptions,
    calculatePaginationMetadata,
    formatProductResponse
} from '../utils/productUtils.js';

// Obtener todos los productos (público)
export const getProducts = async (req, res) => {
    try {
        const query = buildProductQuery(req.query);
        const { page, limit, skip, sort } = buildPaginationOptions(req.query);

        // Ejecutar query con paginación
        const [products, total] = await Promise.all([
            Product.find(query)
                .sort(sort)
                .skip(skip)
                .limit(limit)
                .lean(),
            Product.countDocuments(query)
        ]);
```

```
        const pagination = calculatePaginationMetadata(total, page, limit);

        res.status(200).json(formatProductResponse(products, pagination));

    } catch (error) {
        console.error('Error obteniendo productos:', error);
        res.status(500).json({
            success: false,
            message: 'Error al obtener productos',
            error: process.env.NODE_ENV === 'development' ? error.message : undefined
        });
    }
};
```

```
// Obtener producto por ID (público)
export const getProductById = async (req, res) => {
    try {
        const product = await Product.findOne({
            _id: req.params.id,
            isActive: true
        });

        if (!product) {
            return res.status(404).json({
                success: false,
                message: 'Producto no encontrado'
            });
        }

        res.status(200).json({
            success: true,
            data: { product: formatProductResponse([product]).data.products[0] }
        });
```

```
    } catch (error) {
        console.error('Error obteniendo producto:', error);

        if (error.name === 'CastError') {
            return res.status(400).json({
                success: false,
                message: 'ID de producto no válido'
            });
        }

        res.status(500).json({
            success: false,
            message: 'Error al obtener el producto'
        });
    }
};
```

```
// Crear nuevo producto (solo admin)
export const createProduct = async (req, res) => {
    try {
        const validation = validateProductData(req.body);
        if (!validation.isValid) {
            return res.status(400).json({
                success: false,
                message: 'Datos del producto inválidos',
                errors: validation.errors
            });
        }
```

```
        // Generar SKU automático si no se proporciona
        const productData = { ...req.body };
        if (!productData.sku) {
            const baseSku = productData.name
                .toLowerCase()
                .replace(/[^a-z0-9]+/g, '-')
                .replace(/(^-|-$)/g, '')
                .substring(0, 20);

            const count = await Product.countDocuments({
                sku: new RegExp(`^${baseSku}`)
            });

            productData.sku = count > 0 ? `${baseSku}-${count + 1}` : baseSku;
        }

        const product = new Product(productData);
        await product.save();
```

```
        res.status(201).json({
            success: true,
            message: 'Producto creado exitosamente',
            data: { product: formatProductResponse([product]).data.products[0] }
        });

    } catch (error) {
        console.error('Error creando producto:', error);

        if (error.code === 11000) {
            return res.status(409).json({
                success: false,
                message: 'El SKU del producto ya existe',
                error: 'DUPLICATE_SKU'
            });
        }

        res.status(500).json({
            success: false,
            message: 'Error al crear el producto',
            error: process.env.NODE_ENV === 'development' ? error.message : undefined
        });
    }
};
```

```
// Actualizar producto (solo admin)
export const updateProduct = async (req, res) => {
    try {
        const validation = validateProductData(req.body, true);
        if (!validation.isValid) {
            return res.status(400).json({
                success: false,
                message: 'Datos del producto inválidos',
                errors: validation.errors
            });
        }

        const product = await Product.findByIdAndUpdate(
            req.params.id,
            req.body,
            {
                new: true,
                runValidators: true,
                context: 'query'
            }
        );
```

```
        res.status(200).json({
            success: true,
            message: 'Producto actualizado exitosamente',
            data: { product: formatProductResponse([product]).data.products[0] }
        });

    } catch (error) {
        console.error('Error actualizando producto:', error);

        if (error.name === 'CastError') {
            return res.status(400).json({
                success: false,
                message: 'ID de producto no válido'
            });
        }
```

```
        if (error.code === 11000) {
            return res.status(409).json({
                success: false,
                message: 'El SKU del producto ya existe',
                error: 'DUPLICATE_SKU'
            });
        }

        res.status(500).json({
            success: false,
            message: 'Error al actualizar el producto'
        });
    }
};
```

```
// Eliminar producto (soft delete - solo admin)
export const deleteProduct = async (req, res) => {
    try {
        const product = await Product.findByIdAndUpdate(
            req.params.id,
            { isActive: false },
            { new: true }
        );
```

```
        if (!product) {
            return res.status(404).json({
                success: false,
                message: 'Producto no encontrado'
            });
        }

        res.status(200).json({
            success: true,
            message: 'Producto eliminado exitosamente'
        });
```

```
    } catch (error) {
        console.error('Error eliminando producto:', error);

        if (error.name === 'CastError') {
            return res.status(400).json({
                success: false,
                message: 'ID de producto no válido'
            });
        }

        res.status(500).json({
            success: false,
            message: 'Error al eliminar el producto'
        });
    }
};
```

```
// Obtener productos por categoría (público)
export const getProductsByCategory = async (req, res) => {
    try {
        const { category } = req.params;
        const query = buildProductQuery({ ...req.query, category });
        const { page, limit, skip, sort } = buildPaginationOptions(req.query);

        const [products, total] = await Promise.all([
            Product.find(query)
                .sort(sort)
                .skip(skip)
                .limit(limit)
                .lean(),
            Product.countDocuments(query)
        ]);
```

```
        const pagination = calculatePaginationMetadata(total, page, limit);

        res.status(200).json({
            ...formatProductResponse(products, pagination),
            category: category
        });
```

```
    } catch (error) {
        console.error('Error obteniendo productos por categoría:', error);
        res.status(500).json({
            success: false,
            message: 'Error al obtener productos por categoría'
        });
    }
};

// Buscar productos (público)
export const searchProducts = async (req, res) => {
    try {
        const { q } = req.query;

        if (!q || q.trim().length === 0) {
            return res.status(400).json({
                success: false,
                message: 'Término de búsqueda requerido'
            });
        }
```

```
        const query = buildProductQuery({ ...req.query, search: q });
        const { page, limit, skip, sort } = buildPaginationOptions(req.query);

        const [products, total] = await Promise.all([
            Product.find(query)
                .sort(sort)
                .skip(skip)
                .limit(limit)
                .lean(),
            Product.countDocuments(query)
        ]);

        const pagination = calculatePaginationMetadata(total, page, limit);

        res.status(200).json({
            ...formatProductResponse(products, pagination),
            searchTerm: q,
            resultsCount: total
        });
```

```
    } catch (error) {
        console.error('Error buscando productos:', error);
        res.status(500).json({
            success: false,
            message: 'Error al buscar productos'
        });
    }
};

// Obtener productos destacados (público)
export const getFeaturedProducts = async (req, res) => {
    try {
        const limit = Math.min(8, parseInt(req.query.limit) || 4);

        const products = await Product.find({
            isActive: true,
            'rating.average': { $gte: 4.0 }
        })
        .sort({ 'rating.average': -1, 'rating.count': -1 })
        .limit(limit)
        .lean();
```

```
        res.status(200).json({
            success: true,
            data: {
                products: formatProductResponse(products).data.products,
                featured: true
            }
        });

    } catch (error) {
        console.error('Error obteniendo productos destacados:', error);
        res.status(500).json({
            success: false,
            message: 'Error al obtener productos destacados'
        });
    }
};
```

3. Actualizar Rutas de Productos

Actualizar archivo routes/products.js:

```
import express from 'express';
import {
    getProducts,
    getProductById,
    createProduct,
    updateProduct,
    deleteProduct,
    getProductsByCategory,
    searchProducts,
    getFeaturedProducts
} from '../controllers/productController.js';
import { authenticateToken, requireRole } from '../middleware/authMiddleware.js';

const router = express.Router();
```

```
// Rutas públicas
router.get('/', getProducts);
router.get('/search', searchProducts);
router.get('/featured', getFeaturedProducts);
router.get('/category/:category', getProductsByCategory);
router.get('/:id', getProductById);

// Rutas protegidas (solo admin)
router.post('/', authenticateToken, requireRole(['admin']), createProduct);
router.put('/:id', authenticateToken, requireRole(['admin']), updateProduct);
router.delete('/:id', authenticateToken, requireRole(['admin']), deleteProduct);
```

```
// Ruta de información de API
router.get('/info/api', (req, res) => {
    res.status(200).json({
        success: true,
        message: 'Products API funcionando',
        endpoints: {
            getAll: 'GET /api/products',
            getById: 'GET /api/products/:id',
            search: 'GET /api/products/search?q=term',
            featured: 'GET /api/products/featured',
            byCategory: 'GET /api/products/category/:category',
            create: 'POST /api/products (admin only)',
            update: 'PUT /api/products/:id (admin only)',
            delete: 'DELETE /api/products/:id (admin only)'
        },
```

```
        queryParameters: {
            page: 'Número de página',
            limit: 'Items por página (max 50)',
            sort: 'price-asc, price-desc, name-asc, name-desc, newest, oldest, rating',
            category: 'Filtrar por categoría',
            minPrice: 'Precio mínimo',
            maxPrice: 'Precio máximo',
            inStock: 'true/false para filtrar stock',
            minRating: 'Rating mínimo (1-5)'
        }
    });
});

export default router;
```

4. Crear Middleware de Validación Adicional

Crear archivo middleware/validationMiddleware.js:

```
import { ObjectId } from 'mongodb';

// Validar ObjectId de MongoDB
export const validateObjectId = (req, res, next) => {
    const { id } = req.params;

    if (!ObjectId.isValid(id)) {
        return res.status(400).json({
            success: false,
            message: 'ID no válido'
        });
    }

    next();
};
```

```
// Validar parámetros de query para productos
export const validateProductQuery = (req, res, next) => {
    const { limit, page, minPrice, maxPrice, minRating } = req.query;

    if (limit && (isNaN(limit) || parseInt(limit) < 1)) {
        return res.status(400).json({
            success: false,
            message: 'El parámetro limit debe ser un número positivo'
        });
    }

    if (page && (isNaN(page) || parseInt(page) < 1)) {
        return res.status(400).json({
            success: false,
            message: 'El parámetro page debe ser un número positivo'
        });
    }
```

```
    if (minPrice && (isNaN(minPrice) || parseFloat(minPrice) < 0)) {
        return res.status(400).json({
            success: false,
            message: 'El parámetro minPrice debe ser un número positivo'
        });
    }

    if (maxPrice && (isNaN(maxPrice) || parseFloat(maxPrice) < 0)) {
        return res.status(400).json({
            success: false,
            message: 'El parámetro maxPrice debe ser un número positivo'
        });
    }

    if (minRating && (isNaN(minRating) || parseFloat(minRating) < 1 || parseFloat(minRating) > 5)) {
        return res.status(400).json({
            success: false,
            message: 'El parámetro minRating debe estar entre 1 y 5'
        });
    }

    next();
};
```

5. Actualizar Rutas con Validación

Actualizar routes/products.js con validaciones:

```
import express from 'express';
import {
    getProducts,
    getProductById,
    // ... otros imports ...
} from '../controllers/productController.js';
import { authenticateToken, requireRole } from '../middleware/authMiddleware.js';
import { validateObjectId, validateProductQuery } from '../middleware/validationMiddleware.js';

const router = express.Router();

// Aplicar validación de query a rutas relevantes
router.get('/', validateProductQuery, getProducts);
router.get('/search', validateProductQuery, searchProducts);
router.get('/category/:category', validateProductQuery, getProductsByCategory);

// Aplicar validación de ObjectId donde sea necesario
router.get('/:id', validateObjectId, getProductById);
router.put('/:id', validateObjectId, authenticateToken, requireRole(['admin']), updateProduct);
router.delete('/:id', validateObjectId, authenticateToken, requireRole(['admin']), deleteProduct);

// ... resto de las rutas ...
```

Verificación del CRUD de Productos

1. Probar Endpoints de Productos

```
npm run dev
```

2. Test de Flujo Completo CRUD

1. Listar productos con paginación y filtros
2. Buscar productos por término
3. Obtener producto por ID
4. Crear producto (como admin)
5. Actualizar producto (como admin)
6. Eliminar producto (soft delete como admin)

3. Response Esperados

Lista de productos:

```
{
    "success": true,
    "data": {
        "products": [...],
        "pagination": {
            "total": 25,
            "totalPages": 3,
            "currentPage": 1,
            "hasNext": true,
            "hasPrev": false,
            "nextPage": 2,
            "prevPage": null
        }
    }
}
```

Producto individual:

```
{
    "success": true,
    "data": {
        "product": {
            "id": "507f1f77bcf86cd799439011",
            "name": "Smartphone Premium",
            "price": 899.99,
            "category": "electronics",
            "inStock": true,
            "rating": {
                "average": 4.5,
                "count": 120
            }
        }
    }
}
```

Paso 6: Implementar Endpoint de Contacto

El objetivo de este paso es implementar un sistema completo de gestión de mensajes de contacto que permita a los usuarios enviar consultas, sugerencias o solicitudes de soporte a través de la aplicación. Incluiremos validación de datos, almacenamiento en base de datos, y notificaciones opcionales por email.

Acciones Específicas

1. Crear Modelo de Mensaje de Contacto

Crear archivo models/ContactMessage.js**:**

```
import mongoose from 'mongoose';

const contactMessageSchema = new mongoose.Schema({
    name: {
        type: String,
        required: [true, 'El nombre es requerido'],
        trim: true,
        maxlength: [100, 'El nombre no puede exceder 100 caracteres'],
        minlength: [2, 'El nombre debe tener al menos 2 caracteres']
    },
    email: {
        type: String,
        required: [true, 'El email es requerido'],
        lowercase: true,
        match: [
            /^\w+([.-]?\w+)*@\w+([.-]?\w+)*(\.\w{2,3})+$/,
            'Por favor ingresa un email válido'
        ]
```

```
    },
    phone: {
        type: String,
        trim: true,
        maxlength: [20, 'El teléfono no puede exceder 20 caracteres'],
        match: [
            /^[\+]?[0-9\s\-\(\)]{10,20}$/,
            'Por favor ingresa un número de teléfono válido'
        ]
    },
    subject: {
        type: String,
        required: [true, 'El asunto es requerido'],
        trim: true,
        maxlength: [200, 'El asunto no puede exceder 200 caracteres'],
        minlength: [5, 'El asunto debe tener al menos 5 caracteres']
    },
    message: {
        type: String,
        required: [true, 'El mensaje es requerido'],
        trim: true,
        maxlength: [2000, 'El mensaje no puede exceder 2000 caracteres'],
        minlength: [10, 'El mensaje debe tener al menos 10 caracteres']
    },
```

```
    category: {
        type: String,
        required: true,
        enum: {
            values: ['general', 'support', 'sales', 'technical', 'complaint', 'suggestion'],
            message: 'Categoría no válida'
        },
        default: 'general'
    },
    status: {
        type: String,
        enum: ['new', 'in_progress', 'resolved', 'closed'],
        default: 'new'
    },
    priority: {
        type: String,
        enum: ['low', 'medium', 'high', 'urgent'],
        default: 'medium'
    },
    ipAddress: {
        type: String,
        trim: true
    },
```

```
    userAgent: {
        type: String,
        trim: true
    },
    userId: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'User',
        default: null
    },
    response: {
        message: String,
        respondedBy: {
            type: mongoose.Schema.Types.ObjectId,
            ref: 'User'
        },
        respondedAt: Date
    },
    assignedTo: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'User',
        default: null
    }
```

```
// Virtual para tiempo desde creación
contactMessageSchema.virtual('ageInHours').get(function() {
    return Math.floor((Date.now() - this.createdAt) / (1000 * 60 * 60));
});

// Virtual para saber si tiene respuesta
contactMessageSchema.virtual('hasResponse').get(function() {
    return !!this.response && !!this.response.message;
});
```

```
}, {
    timestamps: true,
    toJSON: { virtuals: true },
    toObject: { virtuals: true }
});

// Índices para mejor performance
contactMessageSchema.index({ email: 1 });
contactMessageSchema.index({ status: 1 });
contactMessageSchema.index({ category: 1 });
contactMessageSchema.index({ createdAt: -1 });
contactMessageSchema.index({ priority: 1 });
```

```
// Método estático para contar mensajes por estado
contactMessageSchema.statics.countByStatus = function() {
    return this.aggregate([
        { $group: { _id: '$status', count: { $sum: 1 } } }
    ]);
};

// Método estático para obtener mensajes recientes
contactMessageSchema.statics.getRecentMessages = function(days = 7) {
    const date = new Date();
    date.setDate(date.getDate() - days);

    return this.find({ createdAt: { $gte: date } })
        .sort({ createdAt: -1 })
        .limit(50);
};

const ContactMessage = mongoose.model('ContactMessage', contactMessageSchema);
export default ContactMessage;
```

2. Crear Utilities para Validación y Email

Crear archivo utils/contactUtils.js:

```
import validator from 'validator';

// Validar datos del formulario de contacto
export const validateContactData = (data) => {
    const errors = {};

    // Validar nombre
    if (!data.name || data.name.trim().length === 0) {
        errors.name = 'El nombre es requerido';
    } else if (data.name.length < 2) {
        errors.name = 'El nombre debe tener al menos 2 caracteres';
    } else if (data.name.length > 100) {
        errors.name = 'El nombre no puede exceder 100 caracteres';
    }

    // Validar email
    if (!data.email || data.email.trim().length === 0) {
        errors.email = 'El email es requerido';
    } else if (!validator.isEmail(data.email)) {
        errors.email = 'Por favor ingresa un email válido';
    }
```

```
    // Validar asunto
    if (!data.subject || data.subject.trim().length === 0) {
        errors.subject = 'El asunto es requerido';
    } else if (data.subject.length < 5) {
        errors.subject = 'El asunto debe tener al menos 5 caracteres';
    } else if (data.subject.length > 200) {
        errors.subject = 'El asunto no puede exceder 200 caracteres';
    }

    // Validar mensaje
    if (!data.message || data.message.trim().length === 0) {
        errors.message = 'El mensaje es requerido';
    } else if (data.message.length < 10) {
        errors.message = 'El mensaje debe tener al menos 10 caracteres';
    } else if (data.message.length > 2000) {
        errors.message = 'El mensaje no puede exceder 2000 caracteres';
    }
```

```
    // Validar teléfono (opcional)
    if (data.phone && data.phone.trim().length > 0) {
        const phoneRegex = /^[\+]?[0-9\s\-\(\)]{10,20}$/;
        if (!phoneRegex.test(data.phone)) {
            errors.phone = 'Por favor ingresa un número de teléfono válido';
        }
    }

    // Validar categoría
    const validCategories = ['general', 'support', 'sales', 'technical', 'complaint', 'suggestion'];
    if (data.category && !validCategories.includes(data.category)) {
        errors.category = 'Categoría no válida';
    }

    return {
        isValid: Object.keys(errors).length === 0,
        errors
    };
};
```

```
// Sanitizar datos del formulario
export const sanitizeContactData = (data) => {
    return {
        name: validator.trim(validator.escape(data.name || '')),
        email: validator.normalizeEmail(validator.trim(data.email || '')),
        phone: validator.trim(data.phone || ''),
        subject: validator.trim(validator.escape(data.subject || '')),
        message: validator.trim(validator.escape(data.message || '')),
        category: validator.trim(data.category || 'general')
    };
};
```

```
// Formatear respuesta del mensaje
export const formatContactResponse = (contactMessage) => {
    return {
        id: contactMessage._id,
        name: contactMessage.name,
        email: contactMessage.email,
        subject: contactMessage.subject,
        message: contactMessage.message,
        category: contactMessage.category,
        status: contactMessage.status,
        priority: contactMessage.priority,
        createdAt: contactMessage.createdAt,
        hasResponse: contactMessage.hasResponse,
        ageInHours: contactMessage.ageInHours
    };
};
```

```
export const generateConfirmationEmail = (contactData) => {
    return {
        subject: `Confirmación de recepción: ${contactData.subject}`,
        html: `
            <!DOCTYPE html>
            <html>
            <head>
                <meta charset="utf-8">
                <style>
                    body { font-family: Arial, sans-serif; line-height: 1.6; color: #333; }
                    .container { max-width: 600px; margin: 0 auto; padding: 20px; }
                    .header { background: #007bff; color: white; padding: 20px; text-align: center;
}
                    .content { background: #f9f9f9; padding: 20px; }
                    .footer { background: #333; color: white; padding: 10px; text-align: center; }
                </style>
            </head>
```

```
            <body>
                <div class="container">
                    <div class="header">
                        <h1>Gracias por contactarnos</h1>
                    </div>
                    <div class="content">
                        <p>Hola <strong>${contactData.name}</strong>,</p>
                        <p>Hemos recibido tu mensaje y te responderemos dentro de las próximas 24 ho
ras.</p>
```

```
                        <p><strong>Asunto:</strong> ${contactData.subject}</p>
                        <p><strong>Mensaje:</strong></p>
                        <p>${contactData.message}</p>
                        <p>Número de referencia: <strong>${contactData._id}</strong></p>
                    </div>
                    <div class="footer">
                        <p>&copy; 2024 Mi Empresa. Todos los derechos reservados.</p>
                    </div>
                </div>
            </body>
            </html>
        `
    };
};
```

```
// Generar email de notificación para admin
export const generateAdminNotificationEmail = (contactData) => {
    return {
        subject: `🔔 Nuevo mensaje de contacto: ${contactData.subject}`,
        html: `
            <!DOCTYPE html>
            <html>
            <head>
                <meta charset="utf-8">
                <style>
                    body { font-family: Arial, sans-serif; line-height: 1.6; color: #333; }
                    .container { max-width: 600px; margin: 0 auto; padding: 20px; }
                    .header { background: #dc3545; color: white; padding: 20px; text-align: center;
}
                    .content { background: #f9f9f9; padding: 20px; }
                    .info-item { margin-bottom: 10px; }
                    .label { font-weight: bold; color: #555; }
                </style>
            </head>
```

```
            <body>
                <div class="container">
                    <div class="header">
                        <h1>Nuevo Mensaje de Contacto</h1>
                    </div>
                    <div class="content">
                        <div class="info-item"><span class="label">Nombre:</span> ${contactData.nam
e}</div>
                        <div class="info-item"><span class="label">Email:</span> ${contactData.emai
l}</div>
                        <div class="info-item"><span class="label">Teléfono:</span> ${contactData.ph
one || 'No proporcionado'}</div>
```

```
                        <div class="info-item"><span class="label">Categoría:</span> ${contactData.c
ategory}</div>
                        <div class="info-item"><span class="label">Asunto:</span> ${contactData.subj
ect}</div>
                        <div class="info-item"><span class="label">Mensaje:</span></div>
                        <p>${contactData.message}</p>
                        <div class="info-item"><span class="label">ID:</span> ${contactData._id}</di
v>
                        <div class="info-item"><span class="label">Fecha:</span> ${new Date(contactD
ata.createdAt).toLocaleString()}</div>
                    </div>
                </div>
            </body>
            </html>
        `;
    };
};
```

3. Crear Controlador de Contacto

Crear archivo controllers/contactController.js:

```
import ContactMessage from '../models/ContactMessage.js';
import {
    validateContactData,
    sanitizeContactData,
    formatContactResponse,
    generateConfirmationEmail,
    generateAdminNotificationEmail
} from '../utils/contactUtils.js';
```

```
// Enviar mensaje de contacto
export const submitContact = async (req, res) => {
    try {
        const rawData = req.body;
```

```
// Validar datos
const validation = validateContactData(rawData);
if (!validation.isValid) {
    return res.status(400).json({
        success: false,
        message: 'Datos del formulario inválidos',
        errors: validation.errors
    });
}
```

```
// Sanitizar datos
const sanitizedData = sanitizeContactData(rawData);

// Crear mensaje de contacto
const contactMessage = new ContactMessage({
    ...sanitizedData,
    ipAddress: req.ip || req.connection.remoteAddress,
    userAgent: req.get('User-Agent'),
    ...(req.user && { userId: req.user._id }) // Si está autenticado
});

await contactMessage.save();
```

```
// TODO: Implementar envío de emails (paso opcional)
// await sendConfirmationEmail(contactMessage);
// await sendAdminNotification(contactMessage);

res.status(201).json({
    success: true,
    message: 'Mensaje enviado exitosamente. Te responderemos pronto.',
    data: {
        contact: formatContactResponse(contactMessage),
        referenceId: contactMessage._id
    }
});
```

```
    } catch (error) {
        console.error('Error enviando mensaje de contacto:', error);
        res.status(500).json({
            success: false,
            message: 'Error al enviar el mensaje. Por favor, intenta nuevamente.',
            error: process.env.NODE_ENV === 'development' ? error.message : undefined
        });
    }
};
```

```
// Obtener mensajes de contacto (solo admin)
export const getContactMessages = async (req, res) => {
    try {
        const {
            page = 1,
            limit = 10,
            status,
            category,
            sortBy = 'createdAt',
            sortOrder = 'desc'
        } = req.query;

        // Construir query
        const query = {};
        if (status) query.status = status;
        if (category) query.category = category;
```

```
        // Construir query
        const query = {};
        if (status) query.status = status;
        if (category) query.category = category;
```

```
// Opciones de paginación
const options = {
    page: Math.max(1, parseInt(page)),
    limit: Math.min(50, Math.max(1, parseInt(limit))),
    sort: { [sortBy]: sortOrder === 'desc' ? -1 : 1 },
    populate: {
        path: 'userId',
        select: 'name email',
        match: { _id: { $ne: null } }
    }
};
```

```
// Ejecutar query con paginación
const messages = await ContactMessage.find(query)
    .sort(options.sort)
    .limit(options.limit)
    .skip((options.page - 1) * options.limit)
    .populate(options.populate);

const total = await ContactMessage.countDocuments(query);
```

```
res.status(200).json({
    success: true,
    data: {
        messages: messages.map(formatContactResponse),
        pagination: {
            total,
            totalPages: Math.ceil(total / options.limit),
            currentPage: options.page,
            hasNext: options.page < Math.ceil(total / options.limit),
            hasPrev: options.page > 1
        }
    }
});
```

```
    } catch (error) {
        console.error('Error obteniendo mensajes de contacto:', error);
        res.status(500).json({
            success: false,
            message: 'Error al obtener los mensajes de contacto'
        });
    }
};
```

```
// Obtener estadísticas de contactos (solo admin)
export const getContactStats = async (req, res) => {
    try {
        const stats = await ContactMessage.aggregate([
            {
                $facet: {
                    totalMessages: [{ $count: "count" }],
                    byStatus: [{ $group: { _id: "$status", count: { $sum: 1 } } }],
                    byCategory: [{ $group: { _id: "$category", count: { $sum: 1 } } }],
                    recentActivity: [
                        {
                            $match: {
                                createdAt: {
                                    $gte: new Date(Date.now() - 7 * 24 * 60 * 60 * 1000)
                                }
                            }
                        },
```

```
                    {
                        $group: {
                            _id: {
                                $dateToString: {
                                    format: "%Y-%m-%d",
                                    date: "$createdAt"
                                }
                            },
                            count: { $sum: 1 }
                        }
                    },
                    { $sort: { _id: 1 } }
                ]
            }
        }
    ]);
```

```
    res.status(200).json({
        success: true,
        data: {
            total: stats[0].totalMessages[0]?.count || 0,
            byStatus: stats[0].byStatus,
            byCategory: stats[0].byCategory,
            recentActivity: stats[0].recentActivity
        }
    });
```

```
} catch (error) {
    console.error('Error obteniendo estadísticas:', error);
    res.status(500).json({
        success: false,
        message: 'Error al obtener estadísticas de contacto'
    });
}
};
```

```
// Obtener mensaje específico (solo admin)
export const getContactMessage = async (req, res) => {
    try {
        const { id } = req.params;

        const message = await ContactMessage.findById(id)
            .populate('userId', 'name email')
            .populate('assignedTo', 'name email')
            .populate('response.respondedBy', 'name email');

        if (!message) {
            return res.status(404).json({
                success: false,
                message: 'Mensaje no encontrado'
            });
        }
```

```
        res.status(200).json({
            success: true,
            data: {
                message: {
                    id: message._id,
                    name: message.name,
                    email: message.email,
                    phone: message.phone,
                    subject: message.subject,
                    message: message.message,
                    category: message.category,
                    status: message.status,
                    priority: message.priority,
                    ipAddress: message.ipAddress,
                    userAgent: message.userAgent,
                    userId: message.userId,
                    assignedTo: message.assignedTo,
                    response: message.response,
                    createdAt: message.createdAt,
                    updatedAt: message.updatedAt
                }
            }
        });
```

```
    } catch (error) {
        console.error('Error obteniendo mensaje:', error);

        if (error.name === 'CastError') {
            return res.status(400).json({
                success: false,
                message: 'ID de mensaje no válido'
            });
        }

        res.status(500).json({
            success: false,
            message: 'Error al obtener el mensaje'
        });
    }
};
```

```
// Actualizar estado de mensaje (solo admin)
export const updateMessageStatus = async (req, res) => {
    try {
        const { id } = req.params;
        const { status, priority, assignedTo } = req.body;

        const updateData = {};
        if (status) updateData.status = status;
        if (priority) updateData.priority = priority;
        if (assignedTo) updateData.assignedTo = assignedTo;

        const message = await ContactMessage.findByIdAndUpdate(
            id,
            updateData,
            { new: true, runValidators: true }
        ).populate('assignedTo', 'name email');
```

```
        if (!message) {
            return res.status(404).json({
                success: false,
                message: 'Mensaje no encontrado'
            });
        }
```

```
        res.status(200).json({
            success: true,
            message: 'Mensaje actualizado exitosamente',
            data: {
                message: formatContactResponse(message)
            }
        });
```

```
    } catch (error) {
        console.error('Error actualizando mensaje:', error);

        if (error.name === 'CastError') {
            return res.status(400).json({
                success: false,
                message: 'ID de mensaje no válido'
            });
        }

        res.status(500).json({
            success: false,
            message: 'Error al actualizar el mensaje'
        });
    }
};
```

```
export const respondToMessage = async (req, res) => {
    try {
        const { id } = req.params;
        const { responseMessage } = req.body;

        if (!responseMessage || responseMessage.trim().length === 0) {
            return res.status(400).json({
                success: false,
                message: 'El mensaje de respuesta es requerido'
            });
        }
```

```
        const message = await ContactMessage.findByIdAndUpdate(
            id,
            {
                status: 'resolved',
                response: {
                    message: responseMessage.trim(),
                    respondedBy: req.user._id,
                    respondedAt: new Date()
                }
            },
```

```
            { new: true, runValidators: true }
        ).populate('response.respondedBy', 'name email');

        if (!message) {
            return res.status(404).json({
                success: false,
                message: 'Mensaje no encontrado'
            });
        }
```

```
        // TODO: Enviar email de respuesta al usuario
        // await sendResponseEmail(message);

        res.status(200).json({
            success: true,
            message: 'Respuesta enviada exitosamente',
            data: {
                message: formatContactResponse(message)
            }
        });
```

```
    } catch (error) {
        console.error('Error respondiendo mensaje:', error);

        if (error.name === 'CastError') {
            return res.status(400).json({
                success: false,
                message: 'ID de mensaje no válido'
            });
        }
```

```
        res.status(500).json({
            success: false,
            message: 'Error al enviar la respuesta'
        });
    }
};
```

```
// Eliminar mensaje (solo admin)
export const deleteMessage = async (req, res) => {
    try {
        const { id } = req.params;

        const message = await ContactMessage.findByIdAndDelete(id);

        if (!message) {
            return res.status(404).json({
                success: false,
                message: 'Mensaje no encontrado'
            });
        }

        res.status(200).json({
            success: true,
            message: 'Mensaje eliminado exitosamente'
        });
```

```
    } catch (error) {
        console.error('Error eliminando mensaje:', error);

        if (error.name === 'CastError') {
            return res.status(400).json({
                success: false,
                message: 'ID de mensaje no válido'
            });
        }

        res.status(500).json({
            success: false,
            message: 'Error al eliminar el mensaje'
        });
    }
};
```

4. Crear Rutas de Contacto

Crear archivo routes/contact.js:

```
import express from 'express';
import {
    submitContact,
    getContactMessages,
    getContactMessage,
    getContactStats,
    updateMessageStatus,
    respondToMessage,
    deleteMessage
} from '../controllers/contactController.js';
import { authenticateToken, requireRole } from '../middleware/authMiddleware.js';

const router = express.Router();
```

```
// Ruta pública para enviar mensajes
router.post('/submit', submitContact);

// Rutas protegidas (solo admin)
router.use(authenticateToken);
router.use(requireRole(['admin']));

router.get('/messages', getContactMessages);
router.get('/stats', getContactStats);
router.get('/messages/:id', getContactMessage);
router.put('/messages/:id/status', updateMessageStatus);
router.post('/messages/:id/respond', respondToMessage);
router.delete('/messages/:id', deleteMessage);
```

```
// Ruta de información
router.get('/info', (req, res) => {
    res.status(200).json({
        success: true,
        message: 'Contact API funcionando',
        endpoints: {
            submit: 'POST /api/contact/submit (pública)',
            getMessages: 'GET /api/contact/messages (admin)',
            getMessage: 'GET /api/contact/messages/:id (admin)',
            getStats: 'GET /api/contact/stats (admin)',
            updateStatus: 'PUT /api/contact/messages/:id/status (admin)',
            respond: 'POST /api/contact/messages/:id/respond (admin)',
            delete: 'DELETE /api/contact/messages/:id (admin)'
        },
        categories: ['general', 'support', 'sales', 'technical', 'complaint', 'suggestion'],
        statuses: ['new', 'in_progress', 'resolved', 'closed']
    });
});

export default router;
```

5. Configurar Middleware de Validación

Crear archivo middleware/contactValidation.js:

```
import { validateContactData } from '../utils/contactUtils.js';

// Middleware de validación para contacto
export const validateContactInput = (req, res, next) => {
    const validation = validateContactData(req.body);

    if (!validation.isValid) {
        return res.status(400).json({
            success: false,
            message: 'Datos del formulario inválidos',
            errors: validation.errors
        });
    }

    next();
};
```

```
// Middleware para validar ObjectId
export const validateMessageId = (req, res, next) => {
    const { id } = req.params;

    if (!id.match(/^[0-9a-fA-F]{24}$/)) {
        return res.status(400).json({
            success: false,
            message: 'ID de mensaje no válido'
        });
    }

    next();
};
```

```
// Middleware para validar respuesta
export const validateResponse = (req, res, next) => {
    const { responseMessage } = req.body;

    if (!responseMessage || responseMessage.trim().length === 0) {
        return res.status(400).json({
            success: false,
            message: 'El mensaje de respuesta es requerido'
        });
    }
```

```
    if (responseMessage.length > 2000) {
        return res.status(400).json({
            success: false,
            message: 'La respuesta no puede exceder 2000 caracteres'
        });
    }

    next();
};
```

6. Actualizar Rutas con Validaciones

Actualizar routes/contact.js:

```
import express from 'express';
import {
    submitContact,
    getContactMessages,
    getContactMessage,
    getContactStats,
    updateMessageStatus,
    respondToMessage,
    deleteMessage
} from '../controllers/contactController.js';
import { authenticateToken, requireRole } from '../middleware/authMiddleware.js';
import { validateContactInput, validateMessageId, validateResponse } from '../middleware/contactVali
dation.js';

const router = express.Router();
```

```
// Ruta pública con validación
router.post('/submit', validateContactInput, submitContact);

// Rutas protegidas (solo admin)
router.use(authenticateToken);
router.use(requireRole(['admin']));

router.get('/messages', getContactMessages);
router.get('/stats', getContactStats);
router.get('/messages/:id', validateMessageId, getContactMessage);
router.put('/messages/:id/status', validateMessageId, updateMessageStatus);
router.post('/messages/:id/respond', validateMessageId, validateResponse, respondToMessage);
router.delete('/messages/:id', validateMessageId, deleteMessage);

// ... resto del código ...
```

7. Integrar Rutas en el Servidor Principal

Actualizar server.js para incluir rutas de contacto:

```
import express from 'express';
import cors from 'cors';
import dotenv from 'dotenv';
import database from './config/database.js';
import { requestLogger, errorLogger } from './middleware/logger.js';

// Importar rutas
import authRoutes from './routes/auth.js';
import productRoutes from './routes/products.js';
import cartRoutes from './routes/cart.js';
import contactRoutes from './routes/contact.js'; // ← Nueva ruta

// ... resto del código ...

// Montar rutas principales
app.use('/api/auth', authRoutes);
app.use('/api/products', productRoutes);
app.use('/api/cart', cartRoutes);
app.use('/api/contact', contactRoutes); // ← Nueva ruta

// ... resto del código ...
```

8. Crear Ejemplos de Uso para Testing

Crear archivo examples/contactExamples.http:

```
### Get Contact Info
GET http://localhost:5000/api/contact/info

### Submit Contact Message (Public)
POST http://localhost:5000/api/contact/submit
Content-Type: application/json
```

```
{
    "name": "María García",
    "email": "maria@example.com",
    "phone": "+34 612 345 678",
    "subject": "Consulta sobre productos",
    "message": "Hola, me gustaría obtener más información sobre los productos premium que ofrecen.
¿Tienen disponibilidad inmediata?",
    "category": "sales"
}

### Submit Contact Message (Minimal)
POST http://localhost:5000/api/contact/submit
Content-Type: application/json
```

```
{
    "name": "Juan Pérez",
    "email": "juan@example.com",
    "subject": "Problema técnico",
    "message": "Estoy teniendo problemas para acceder a mi cuenta. ¿Me pueden ayudar?"
}

### Get Contact Messages (Admin)
GET http://localhost:5000/api/contact/messages?page=1&limit=10
Authorization: Bearer {{admin_token}}

### Get Contact Stats (Admin)
GET http://localhost:5000/api/contact/stats
Authorization: Bearer {{admin_token}}
```

```
### Get Specific Message (Admin)
GET http://localhost:5000/api/contact/messages/{{message_id}}
Authorization: Bearer {{admin_token}}

### Update Message Status (Admin)
PUT http://localhost:5000/api/contact/messages/{{message_id}}/status
Authorization: Bearer {{admin_token}}
Content-Type: application/json

{
    "status": "in_progress",
    "priority": "high"
}
```

```
### Respond to Message (Admin)
POST http://localhost:5000/api/contact/messages/{{message_id}}/respond
Authorization: Bearer {{admin_token}}
Content-Type: application/json

{
    "responseMessage": "Hola María, gracias por tu consulta. Nuestros productos premium tienen dispo
nibilidad inmediata. Te enviaremos el catálogo completo por email."
}

### Delete Message (Admin)
DELETE http://localhost:5000/api/contact/messages/{{message_id}}
Authorization: Bearer {{admin_token}}
```

Verificación del Endpoint de Contacto

1. Probar Endpoints de Contacto

```
npm run dev
```

2. Test de Flujo Completo

1. Enviar mensaje como usuario público
2. Obtener mensajes como administrador

3. Actualizar estado de mensaje
4. Responder mensaje
5. Ver estadísticas del sistema

3. Response Esperados

Envío exitoso:


```
{
    "success": true,
    "message": "Mensaje enviado exitosamente. Te responderemos pronto.",
    "data": {
        "contact": {
            "id": "message_id",
            "name": "María García",
            "email": "maria@example.com",
            "subject": "Consulta sobre productos",
            "message": "Hola, me gustaría obtener más información...",
            "category": "sales",
            "status": "new",
            "priority": "medium",
            "createdAt": "2024-01-15T10:30:00.000Z",
            "hasResponse": false,
            "ageInHours": 0
        },
        "referenceId": "message_id"
    }
}
```


Lista de mensajes (admin):

```
{
    "success": true,
    "data": {
        "messages": [...],
        "pagination": {
            "total": 15,
            "totalPages": 2,
            "currentPage": 1,
            "hasNext": true,
            "hasPrev": false
        }
    }
}
```

Paso 7: Conectar Frontend con Backend

El objetivo de este paso es transformar la SPA de React del Laboratorio 5 de una aplicación con datos mock a una aplicación full-stack completamente funcional que consume datos reales del backend. Implementaremos servicios API robustos, manejo de autenticación JWT, gestión de estados de carga y errores, y actualizaremos todos los componentes para usar datos reales.

Acciones Específicas

1. Instalar Dependencias Necesarias en el Frontend

Ejecutar en la terminal del frontend:

```
cd frontend
npm install axios js-cookie
```

2. Crear Hook Personalizado para Manejo de API

Crear archivo frontend/src/hooks/useApi.js:

```
import { useState, useCallback } from 'react';
import { apiRequest } from '../services/api';

export const useApi = () => {
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);
  const [data, setData] = useState(null);

  const execute = useCallback(async (apiCall, onSuccess = null, onError = null) => {
    setLoading(true);
    setError(null);
```

```
    const result = await apiRequest(apiCall);

    if (result.success) {
      setData(result.data);
      if (onSuccess) onSuccess(result.data);
    } else {
      setError(result.error);
      if (onError) onError(result.error);
    }

    setLoading(false);
    return result;
  }, []);
```

```
  const reset = useCallback(() => {
    setLoading(false);
    setError(null);
    setData(null);
  }, []);

  return {
    execute,
    loading,
    error,
    data,
    reset
  };
};
```

```
// Hook específico para operaciones CRUD
export const useApiOperation = (apiFunction) => {
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);

  const execute = useCallback(async (...args) => {
    setLoading(true);
    setError(null);

    const result = await apiRequest(() => apiFunction(...args));

    if (!result.success) {
      setError(result.error);
    }
```

```
    setLoading(false);
    return result;
  }, [apiFunction]);

  return {
    execute,
    loading,
    error
  };
};
```

3. Actualizar ProtectedRoute para Verificar Autenticación

Actualizar archivo frontend/src/components/ProtectedRoute.js:

```
import React, { useEffect, useState } from 'react';
import { Navigate } from 'react-router-dom';
import { authAPI } from '../services/api';

const ProtectedRoute = ({ children, requireAuth = true }) => {
  const [isAuthenticated, setIsAuthenticated] = useState(null);
  const [loading, setLoading] = useState(true);
```

```
  useEffect(() => {
    const checkAuth = async () => {
      const token = localStorage.getItem('authToken');

      if (!token) {
        setIsAuthenticated(false);
        setLoading(false);
        return;
      }
```

```
      try {
        // Verificar token con el backend
        const response = await authAPI.verifyToken();
        if (response.data.success) {
          setIsAuthenticated(true);
        } else {
          setIsAuthenticated(false);
          localStorage.removeItem('authToken');
        }
      } catch (error) {
        console.error('Error verifying token:', error);
        setIsAuthenticated(false);
        localStorage.removeItem('authToken');
      } finally {
        setLoading(false);
      }
    };

    checkAuth();
  }, []);
```

```
// Escuchar eventos de logout desde otros componentes
useEffect(() => {
  const handleLogout = () => {
    setIsAuthenticated(false);
  };

  window.addEventListener('authenticationFailed', handleLogout);
  return () => window.removeEventListener('authenticationFailed', handleLogout);
}, []);

if (loading) {
  return (
    <div className="loading-container">
      <div className="spinner"></div>
      <p>Verificando autenticación...</p>
    </div>
  );
}

 if (requireAuth && !isAuthenticated) {
   return <Navigate to="/login" replace />;
 }

 if (!requireAuth && isAuthenticated) {
   return <Navigate to="/dashboard" replace />;
 }

 return children;
};

export default ProtectedRoute;
```

4. Crear Componentes de Utilidad

Crear archivo frontend/src/components/LoadingSpinner.js**:**

```jsx
import React from 'react';

const LoadingSpinner = ({ size = 'medium', message = 'Cargando...' }) => {
  return (
    <div className={`loading-spinner ${size}`}>
      <div className="spinner"></div>
      {message && <p className="loading-message">{message}</p>}
    </div>
  );
};

export default LoadingSpinner;
```

Crear archivo frontend/src/components/ErrorMessage.js**:**

```jsx
import React from 'react';

const ErrorMessage = ({ message, onRetry, showRetry = true }) => {
  return (
    <div className="error-message-container">
      <div className="error-icon">⚠️</div>
      <h3>Algo salió mal</h3>
      <p>{message || 'Ocurrió un error inesperado'}</p>
      {showRetry && onRetry && (
        <button onClick={onRetry} className="retry-button">
          Reintentar
        </button>
      )}
    </div>
  );
};

export default ErrorMessage;
```

5. Configurar Interceptor Global para Errores

Actualizar archivo frontend/src/index.js**:**

```jsx
import React from 'react';
import ReactDOM from 'react-dom/client';
import { BrowserRouter } from 'react-router-dom';
import { AuthProvider } from './context/AuthContext';
import App from './App';
import './index.css';
```

```jsx
// Interceptor global para errores de autenticación
window.addEventListener('authenticationFailed', () => {
  localStorage.removeItem('authToken');
  localStorage.removeItem('refreshToken');
  localStorage.removeItem('userData');

  if (window.location.pathname !== '/login') {
    window.location.href = '/login';
  }
});
```

```jsx
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <BrowserRouter>
      <AuthProvider>
        <App />
      </AuthProvider>
    </BrowserRouter>
  </React.StrictMode>
);
```

6. Actualizar Variables de Entorno

Crear/actualizar archivo frontend/.env**:**

```
REACT_APP_API_URL=http://localhost:5000/api
REACT_APP_ENVIRONMENT=development
REACT_APP_VERSION=1.0.0
```

Verificación de la Conexión

1. Probar Endpoints desde el Frontend

```
# Iniciar frontend y backend
cd frontend && npm start
cd backend && npm run dev
```

2. Verificar Funcionalidades

1. Registro y Login - Crear cuenta e iniciar sesión
2. Navegación de productos - Listar productos reales
3. Formulario de contacto - Enviar mensajes
4. Protección de rutas - Acceso a rutas privadas

3. Verificar Almacenamiento de Tokens

```
// Los tokens deben guardarse automáticamente
localStorage.getItem('authToken'); // debería existir después del login
```

Solución de Problemas Comunes

Error: CORS

Solución: Verificar que el backend tenga CORS configurado para el frontend

```
// En backend/server.js
app.use(cors({
  origin: 'http://localhost:3000',
  credentials: true
}));
```

Error: Token no incluido

Solución: Verificar que el interceptor esté funcionando

```
// Los requests deben incluir: Authorization: Bearer <token>
```

Error: Rutas protegidas no redirigen

Solución: Verificar que ProtectedRoute verifique el token real

```
// Debe hacer una request a /auth/verify en lugar de solo check localStorage
```

Construye hoy el backend que impulsará el mañana: donde cada línea de código acerca tu visión a la realidad.

—Lina Giraldo