Universidad Tecmilenio

Campus Las Torres

Materia: Desarrollo FullStack

Actividad 4 – Aplicación Web para la Gestión de Productos

Alumno: Patricio Calvo

Matricula: 07097795

Febrero / 2026

Link de Github:

https://github.com/Diablo650cc/Desarrollo-FullStack/tree/main/ACT%204

DOCUMENTACIÓN DEL PROYECTO

Requerimientos Funcionales

Gestión de Usuarios

- Registro de nuevos usuarios (email y contraseña)
- Inicio de sesión con email y contraseña
- Cierre de sesión
- Las contraseñas se guardan encriptadas en la base de datos

Gestión de Productos (CRUD)

- Crear: Agregar nuevos productos
- Leer: Ver lista de todos los productos disponibles
- Actualizar: Modificar datos de un producto existente
- Eliminar: Borrar productos del catálogo

Seguridad

- Solo usuarios autenticados pueden crear, actualizar o eliminar productos
- Usuarios no autenticados solo pueden VER los productos
- Cada usuario solo puede modificar/eliminar sus propios productos

Requerimientos No Funcionales

Técnicos

- Backend: Node.js con Express
- Base de datos: MongoDB
- Autenticación: Tokens JWT
- Pruebas: Jest

Rendimiento

- Tiempo de respuesta < 2 segundos
- API RESTful con códigos HTTP apropiados

Despliegue

- Código en GitHub con Actions para pruebas automáticas
- App desplegada en Vercel

UX/UI

- Mensajes de error claros para el usuario
- Confirmación antes de eliminar productos

CONFIGURACIÓN DEL SERVIDOR

Estructura de Carpetas Creada

```
motogear-crud/
├── src/
│   ├── config/ # Configuración de base de datos
│   ├── models/ # Modelos de Usuario y Producto
│   ├── controllers/ # Lógica de negocio
│   ├── routes/ # Definición de endpoints
│   ├── middlewares/ # Autenticación y protección
│   └── app.js # Configuración de Express
├── .env # Variables de entorno
├── package.json # Dependencias y scripts
└── server.js # Punto de entrada
```

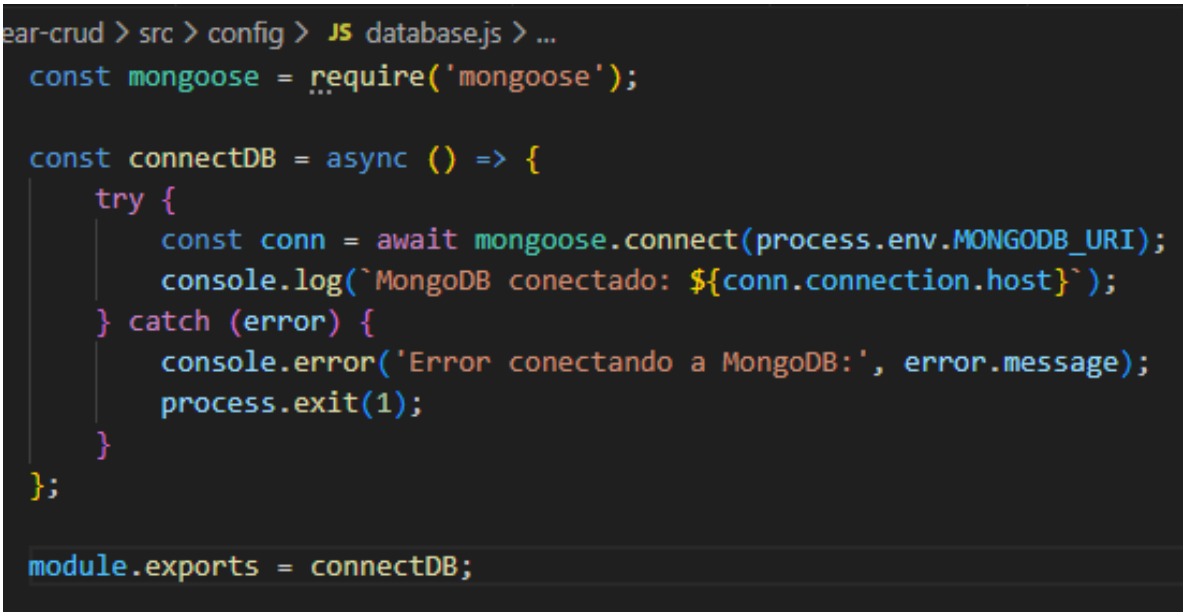
Configuración Inicial

- Inicialización del proyecto con npm init -y.
- Instalación de dependencias principales: express, mongoose, bcryptjs, jsonwebtoken, dotenv, cors.
- Instalación de dependencias de desarrollo: nodemon, jest, supertest.
- Configuración de scripts en package.json (start, dev, test).

```json
{
  "name": "motogear-crud",
  "version": "1.0.0",
  "description": "CRUD de productos de motociclismo con autenticación JWT",
  "main": "server.js",
  Debug
  "scripts": {
    "start": "node server.js",
    "dev": "nodemon server.js",
    "test": "jest"
  },
  "dependencies": {
    "express": "^4.18.2",
    "mongoose": "^7.5.0",
    "bcryptjs": "^2.4.3",
    "jsonwebtoken": "^9.0.2",
    "dotenv": "^16.3.1",
    "cors": "^2.8.5"
  },
  "devDependencies": {
    "nodemon": "^3.0.1",
    "jest": "^29.7.0",
    "supertest": "^6.3.3"
```

Base de Datos (MongoDB)

- Creación de archivo de conexión en src/config/database.js.
- Configuración de conexión usando mongoose.
- Manejo de errores y cierre graceful.



Modelo de Usuario

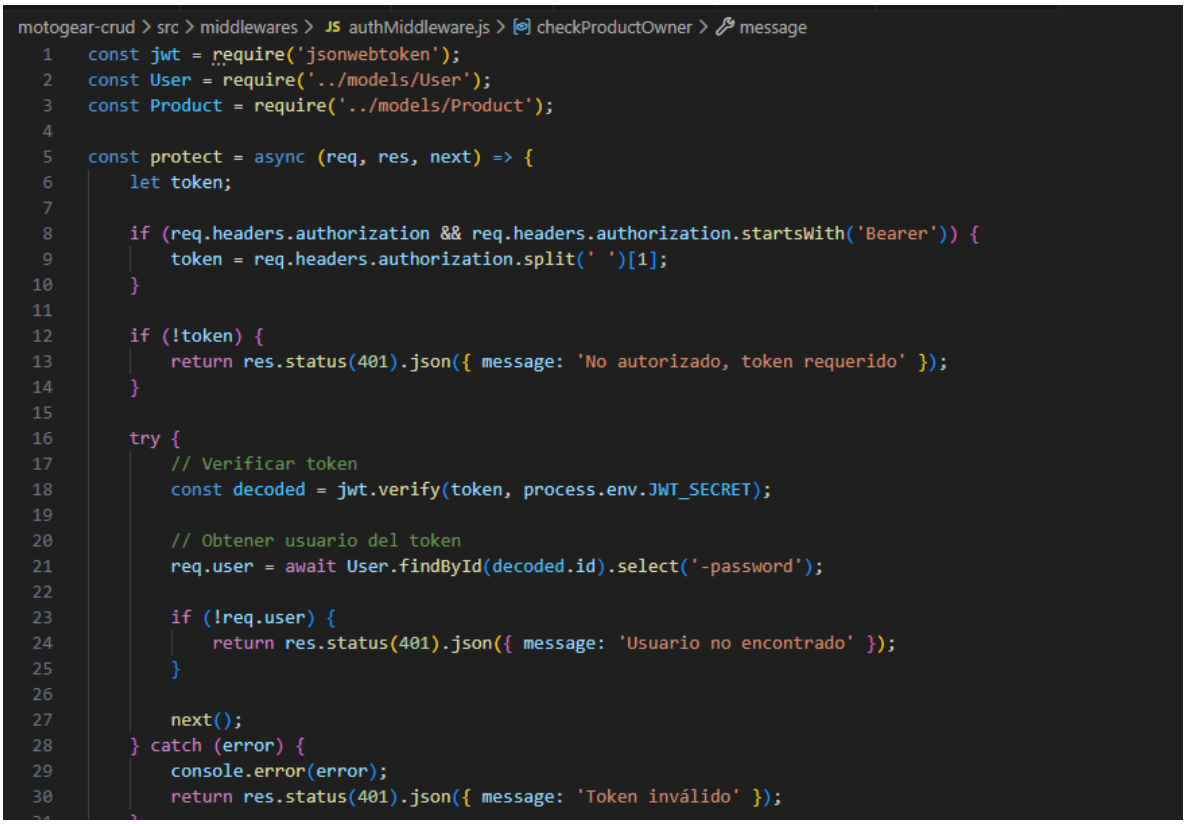
- Schema con email único y password.
- Encriptación automática de contraseñas con bcryptjs mediante hash y salt.
- Método personalizado para comparar contraseñas.
- Validaciones de campos requeridos.

Modelo de Producto

- Schema con campos: nombre, categoría, precio, talla, color, stock.
- Categorías predefinidas: Cascos, Chaquetas, Guantes, Botas, Accesorios.
- Referencia al usuario creador mediante ObjectId.
- Validaciones de tipos y valores mínimos.

Middleware de Autenticación

- Extracción de token JWT del header Authorization.
- Verificación de token usando jwt.verify.
- Búsqueda del usuario asociado al token.
- Adjuntar usuario al objeto req para uso en rutas protegidas.



Controladores Implementados

- Auth Controller: register para creación de nuevos usuarios con email y password.
- login para verificación de credenciales y generación de token JWT.
- getProfile para obtención de datos del usuario autenticado.
- Product Controller: getProducts para lista pública de todos los productos.
- getProductById para detalle de producto específico.
- createProduct para creación de producto requiriendo autenticación.
- updateProduct para modificación de producto existente.
- deleteProduct para eliminación de producto.

Rutas Configuradas

- Rutas de Autenticación en /api/auth: POST /register para registro de usuario, POST /login para inicio de sesión, GET /profile para perfil protegido.

```
const express = require('express');
const { register, login, getProfile } = require('../controllers/authController');
const { protect } = require('../middlewares/authMiddleware');

const router = express.Router();

router.post('/register', register);
router.post('/login', login);
router.get('/profile', protect, getProfile);

module.exports = router;
```

- Rutas de Productos en /api/products: GET / para listar productos públicos, GET /:id para ver producto público, POST / para crear producto protegido, PUT /:id para actualizar protegido requiriendo propietario, DELETE /:id para eliminar protegido requiriendo propietario.

```
const express = require('express');
const {
    getProducts,
    getProductById,
    createProduct,
    updateProduct,
    deleteProduct
} = require('../controllers/productController');
const { protect } = require('../middlewares/authMiddleware');

const router = express.Router();

// Rutas públicas
router.get('/', getProducts);
router.get('/:id', getProductById);

// Rutas protegidas
router.post('/', protect, createProduct);
router.put('/:id', protect, updateProduct);
router.delete('/:id', protect, deleteProduct);
```

Seguridad Implementada

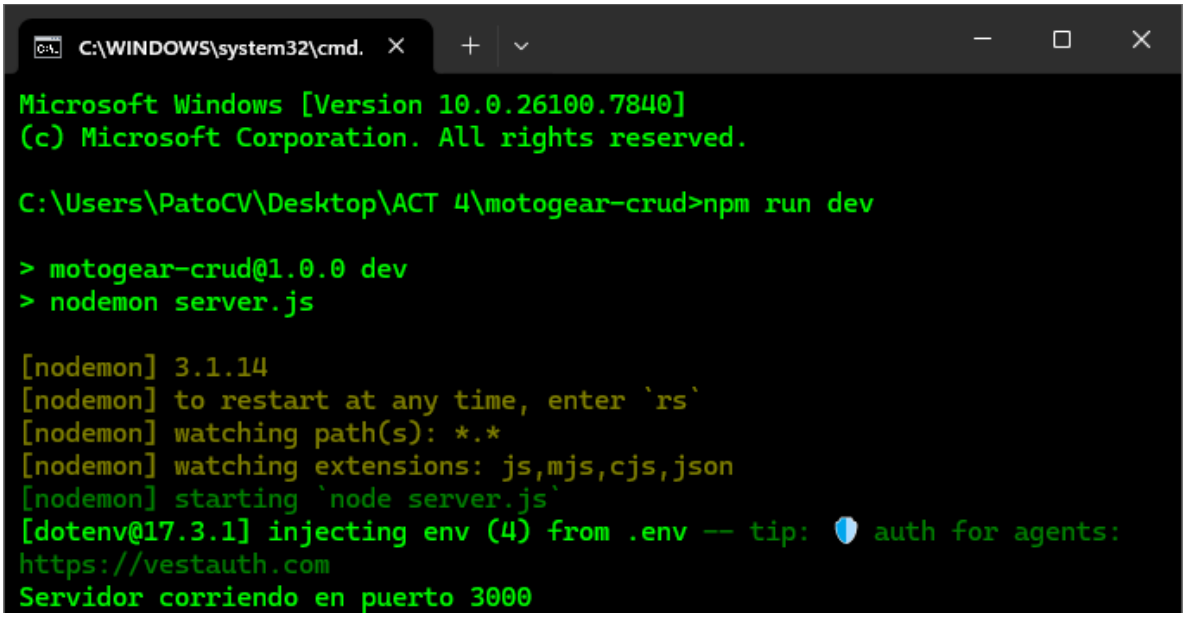
- Contraseñas encriptadas con bcryptjs.
- Tokens JWT con expiración de 7 días.
- Verificación de propiedad para modificar o eliminar productos.
- Validaciones en todos los endpoints.
- Middleware de protección para rutas sensibles.
- Manejo de errores global.

Variables de Entorno (.env)

- PORT=3000
- MONGODB_URI=mongodb://localhost:27017/motogear
- JWT_SECRET=mi_secreto_super_seguro_cambiame_en_produccion
- JWT_EXPIRE=7d

Ejecución del Proyecto

- Servidor iniciado con npm run dev usando nodemon para auto-reload.
- Conexión exitosa a MongoDB local. API disponible en http://localhost:3000



GENERACIÓN DE RUTAS

Añadimos y ajustamos los middlewares de authMiddleware.js:

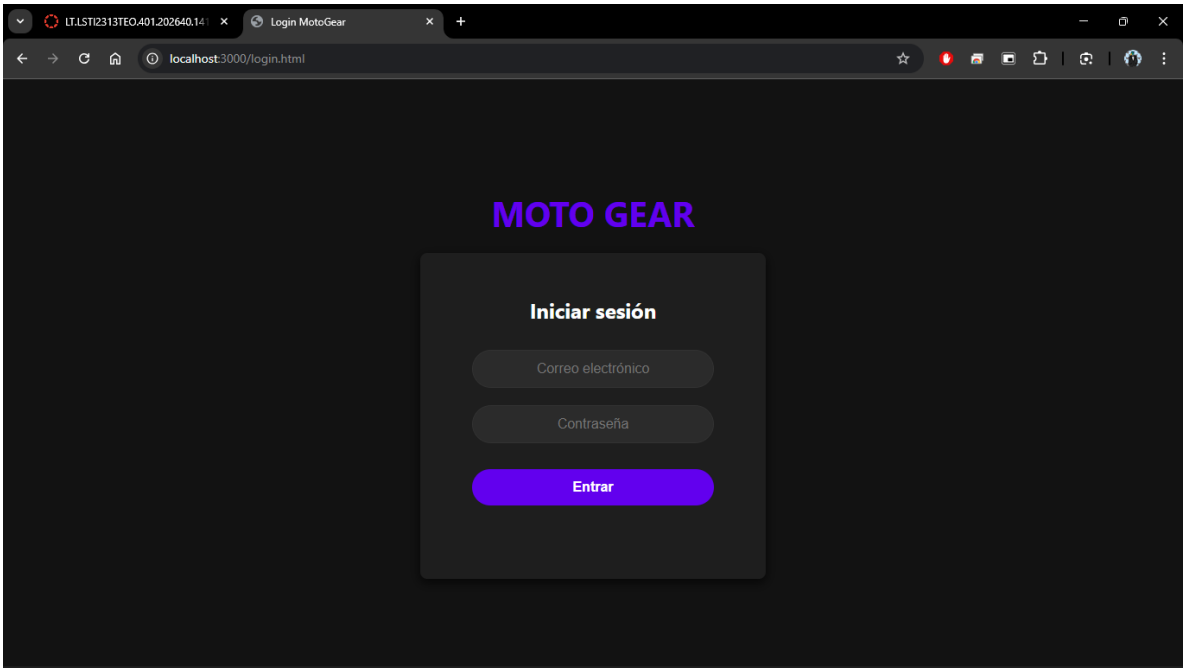
- protect valida token JWT y carga req.user.
- checkProductOwner deja pasar sólo al creador o a un admin.
- admin restringe rutas a administradores.

```
const jwt = require('jsonwebtoken');
const User = require('../models/User');
const Product = require('../models/Product');

const protect = async (req, res, next) => {
    let token;

    if (req.headers.authorization && req.headers.authorization.startsWith('Bearer')) {
        token = req.headers.authorization.split(' ')[1];
    }

    if (!token) {
        return res.status(401).json({ message: 'No autorizado, token requerido' });
    }

    try {
        // Verificar token
        const decoded = jwt.verify(token, process.env.JWT_SECRET);

        // Obtener usuario del token
        req.user = await User.findById(decoded.id).select('-password');

        if (!req.user) {
            return res.status(401).json({ message: 'Usuario no encontrado' });
```

```
// Middleware para verificar propiedad del producto
const checkProductOwner = async (req, res, next) => {
    try {
        const product = await Product.findById(req.params.id);

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

        // permitir al propietario o a un administrador
        if (
            product.usuario.toString() !== req.user._id.toString() &&
            req.user.role !== 'admin'
        ) {
            return res.status(403).json({ message: 'No tienes permiso para modificar este producto' });
        }

        req.product = product;
        next();
    } catch (error) {
        res.status(500).json({ message: 'Error verificando propietario' });
    }
};

// middleware para rutas sólo accesibles por administradores
const admin = (req, res, next) => {
    if (!req.user || req.user.role !== 'admin') {
        return res.status(403).json({ message: 'Se requiere rol de administrador' });
```

IMPLEMENTACIÓN DE AUTENTICACIÓN Y AUTORIZACIÓN

- Se añadió JWT en authController y protect en el middleware para validar el token.
- Las rutas de productos (productRoutes.js) usan protect y checkProductOwner para asegurar creación/edición/borrado.
- Se sirvió un HTML estático (public/login.html) con un formulario que pide email/clave y guarda el token.
- app.js expone la carpeta public/ para que login.html sea accesible en /login.html.



GENERACIÓN DE CONTROLADORES

Modelos Mongoose:

- User.js: Esquema con email (único), contraseña (encriptada con bcrypt), timestamps. Método pre-save para encriptación y método comparePassword().
- Product.js: Esquema con nombre, categoría (enum), precio, talla, color, stock, y relación con usuário via ObjectId.
- Controladores:

AuthController.js:

- register(): User.findOne() para verificar existencia, User.create() para crear usuario
- login(): User.findOne() para buscar, comparePassword() método del modelo
- getProfile(): Retorna datos del usuario autenticado

GENERACIÓN DE PRUEBAS UNITARIAS

Configuración Jest:

- jest.config.js (test environment, coverage paths)
- jest.setup.js (timeout global)
- .env.test (variables de entorno)
- Scripts NPM agregados (test, test:watch, test:coverage, etc.)

60 Tests Creados:

- 28 tests - Controladores (auth + products)
- 10 tests - Middlewares (protección)
- 22 tests - Rutas/Integración HTTP

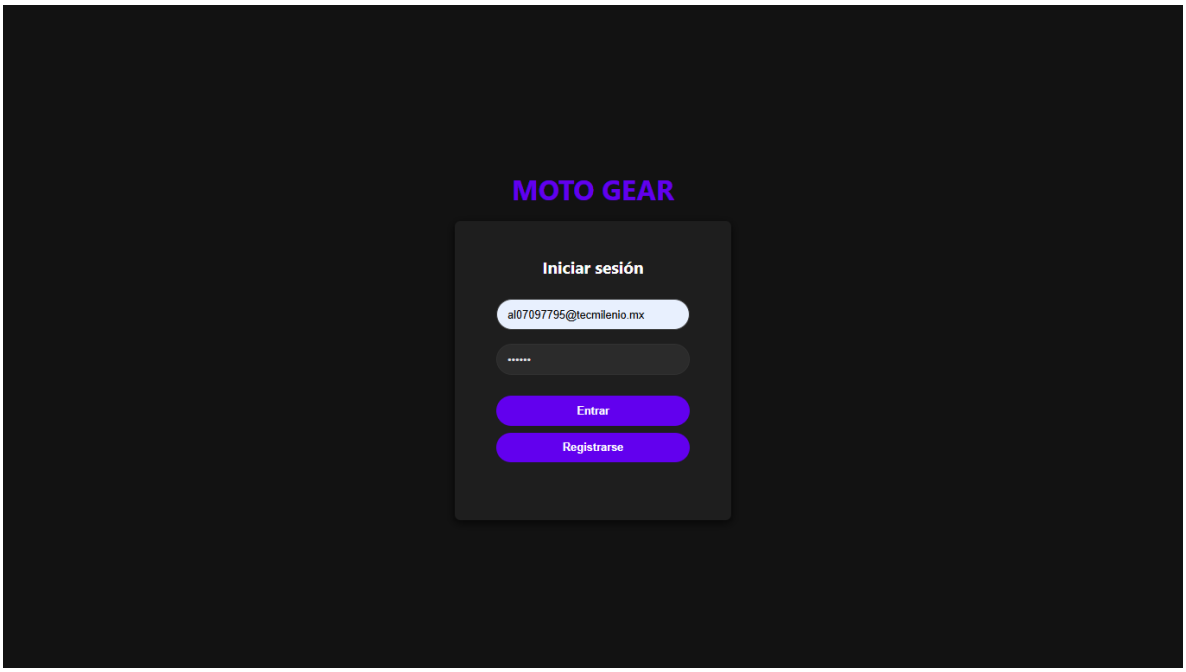
Validaciones Cubiertas:

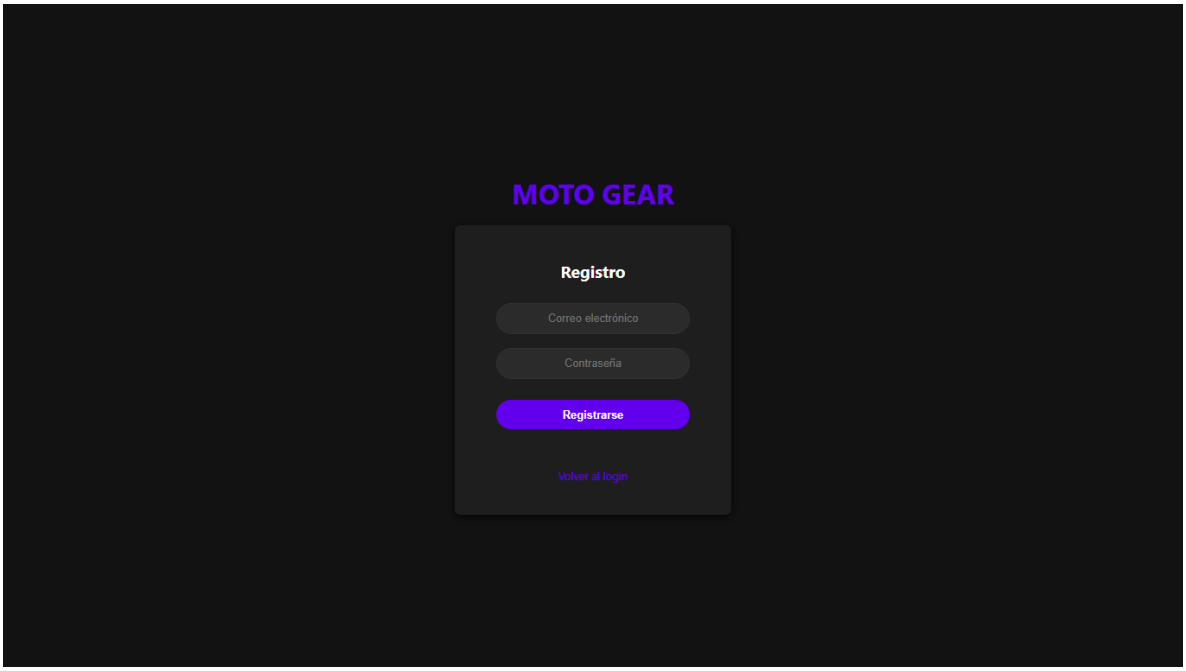
- Registro y login usuarios
- CRUD completo productos
- Autenticación JWT
- Autorización/propiedad
- Errores HTTP (400, 401, 403, 404, 500)
- Validación de campos

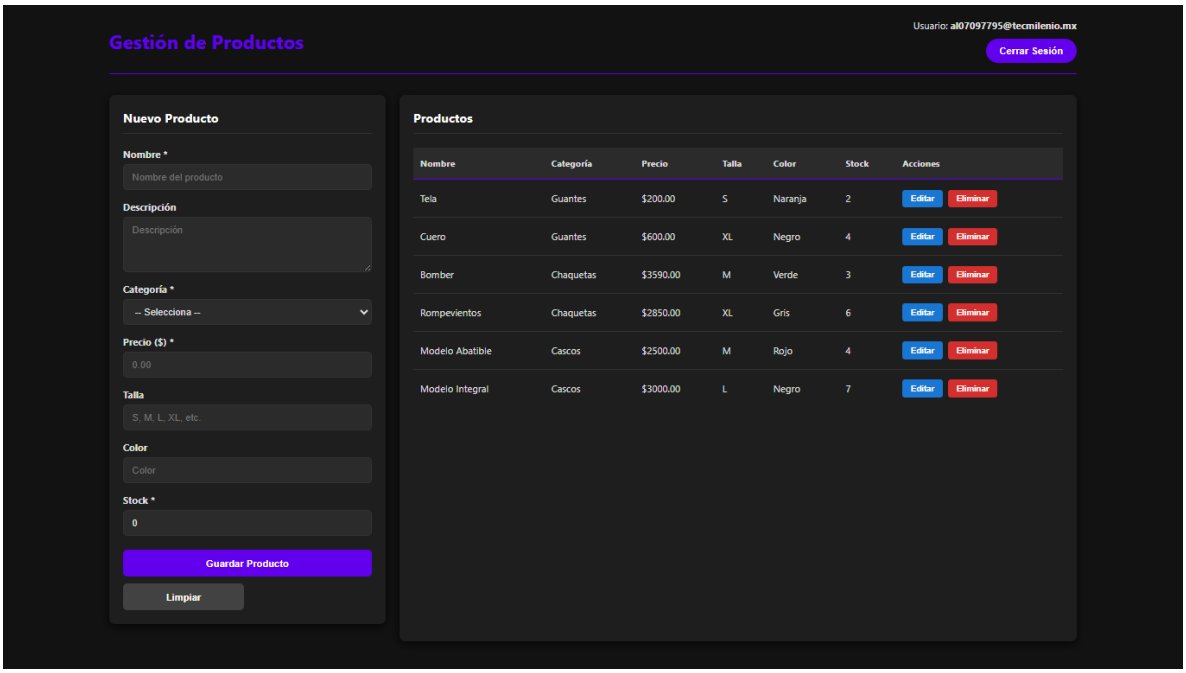
Documentación:

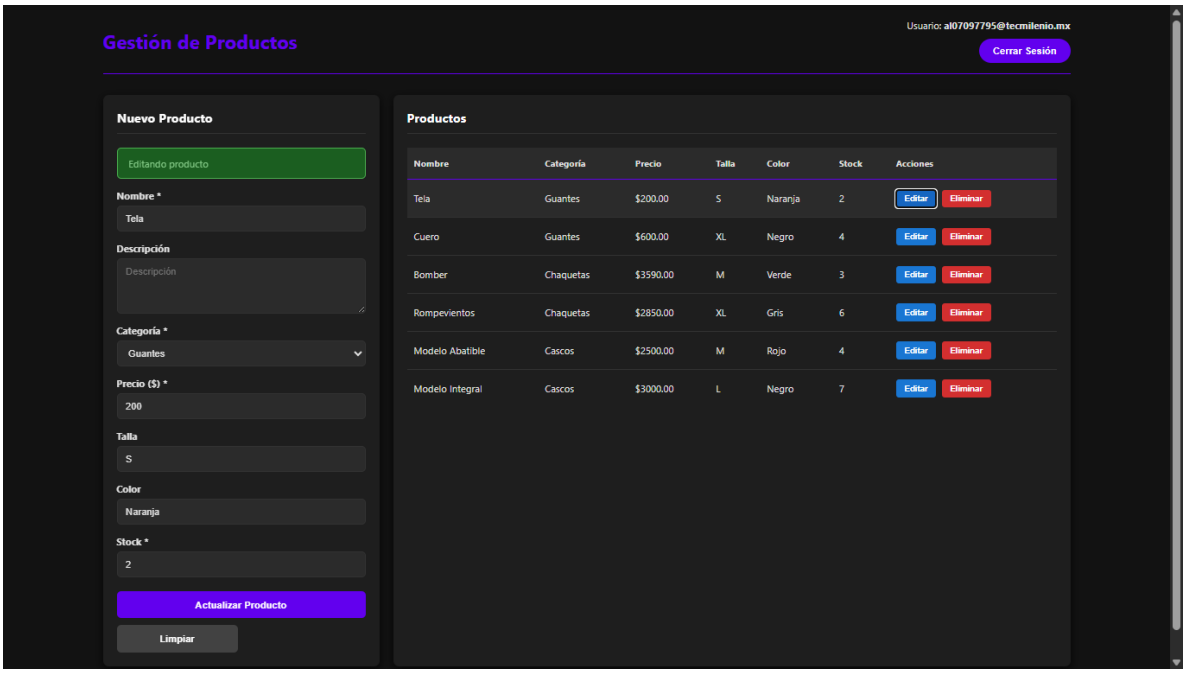
- TESTING.md (guía completa)
- TESTING_SUMMARY.md (resumen)
- TESTING_CHECKLIST.md (checklist)
- RESUMEN_PRUEBAS.md (guía rápida)

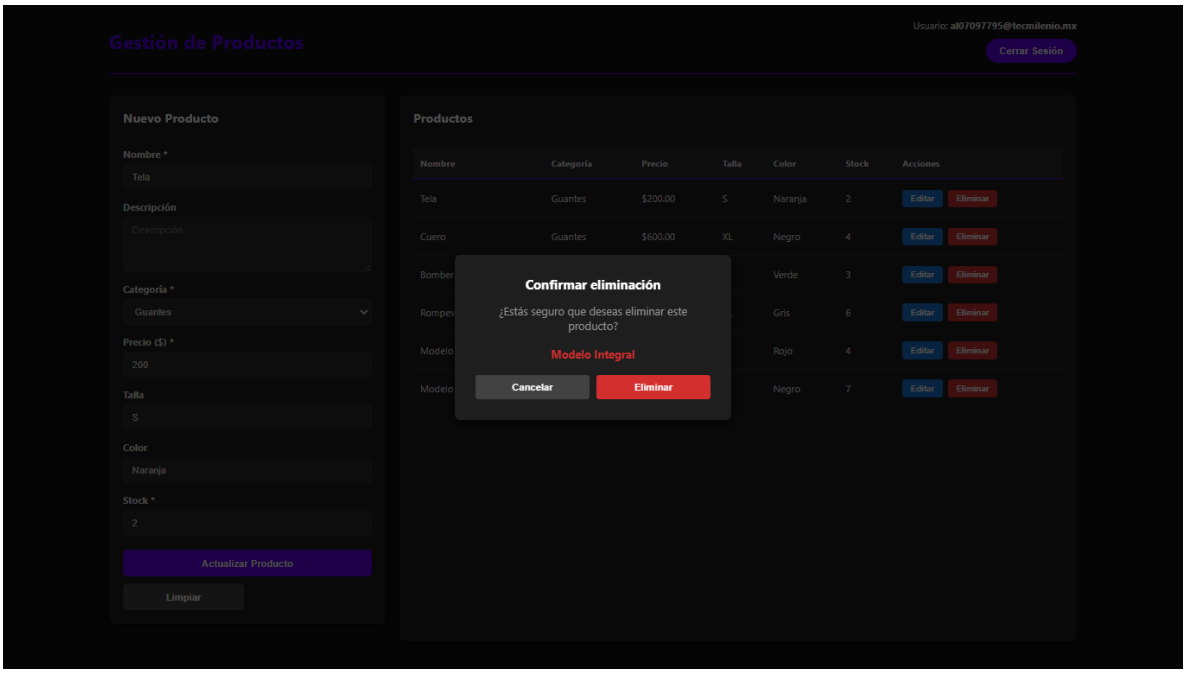
DESPLIEGUE


MOTO GEAR
Iniciar sesión
al07097795@tecmilenio.mx
Entrar
Registrarse


MOTO GEAR
Registro
Correo electrónico
Contraseña
Registrarse
Volver al login


Gestión de Productos
Usuario: al07097795@tecmilenio.mx
Cerrar Sesión
Nuevo Producto
Nombre *
Nombre del producto
Descripción
Descripción
Categoría *
-- Selecciona --
Precio ($) *
0.00
Talla
S, M, L, XL, etc.
Color
Color
Stock *
0
Guardar Producto
Limpiar
Productos
Nombre Categoría Precio Talla Color Stock Acciones
Tela Guantes $200.00 S Naranja 2 Editar Eliminar
Cuero Guantes $600.00 XL Negro 4 Editar Eliminar
Bomber Chaquetas $3590.00 M Verde 3 Editar Eliminar
Rompevientos Chaquetas $2850.00 XL Gris 6 Editar Eliminar
Modelo Abatible Cascos $2500.00 M Rojo 4 Editar Eliminar
Modelo Integral Cascos $3000.00 L Negro 7 Editar Eliminar


Gestión de Productos
Usuario: al07097795@tecmilenio.mx
Cerrar Sesión
Nuevo Producto
Editando producto
Nombre *
Tela
Descripción
Descripción
Categoría *
Guantes
Precio ($) *
200
Talla
S
Color
Naranja
Stock *
2
Actualizar Producto
Limpiar
Productos
Nombre Categoría Precio Talla Color Stock Acciones
Tela Guantes $200.00 S Naranja 2 Editar Eliminar
Cuero Guantes $600.00 XL Negro 4 Editar Eliminar
Bomber Chaquetas $3590.00 M Verde 3 Editar Eliminar
Rompevientos Chaquetas $2850.00 XL Gris 6 Editar Eliminar
Modelo Abatible Cascos $2500.00 M Rojo 4 Editar Eliminar
Modelo Integral Cascos $3000.00 L Negro 7 Editar Eliminar


Gestión de Productos
Usuario: al07097795@tecmilenio.mx
Cerrar Sesión
Nuevo Producto
Nombre *
Tela
Descripción
Descripción
Categoría *
Guantes
Precio ($) *
200
Talla
S
Color
Naranja
Stock *
2
Actualizar Producto
Limpiar
Productos
Nombre
Categoría
Precio
Talla
Color
Stock
Acciones
Tela
Guantes
$200.00
S
Naranja
2
Editar
Eliminar
Cuero
Guantes
$600.00
XL
Negro
4
Editar
Eliminar
Bomber
Verde
3
Editar
Eliminar
Rompev
Gris
6
Editar
Eliminar
Modelo
Rojo
4
Editar
Eliminar
Modelo
Negro
7
Editar
Eliminar
Confirmar eliminación
¿Estás seguro que deseas eliminar este producto?
Modelo Integral
Cancelar
Eliminar