**GESTOCK – Documentación del Proyecto**

**Universidad del Valle**

**1. INTRODUCCIÓN**

Gestock es un sistema para gestionar inventario desarrollado como proyecto académico del curso de Desarrollo de Software I. Su objetivo principal es optimizar la gestión de inventarios en pequeñas y medianas empresas mediante el uso de tecnologías web modernas.

Este sistema permite llevar un control detallado de productos, proveedores, categorías, usuarios y movimientos de inventario, incluyendo entradas y salidas. Además, incorpora notificaciones automáticas por bajo stock, autenticación de usuarios con control de acceso por roles, y una interfaz moderna e intuitiva.

**2. OBJETIVOS**

**Objetivo General**

Desarrollar una solución informática modular para la gestión de inventarios, que integre control de productos, seguimiento de movimientos y administración de usuarios, implementada con tecnologías actuales del ecosistema web.

**Objetivos Específicos**

* Diseñar una arquitectura cliente-servidor basada en Angular y Laravel.
* Implementar CRUD completos para productos, proveedores, movimientos y categorías.
* Incorporar autenticación de usuarios y gestión de roles.
* Automatizar notificaciones y alertas críticas.
* Integrar reportes y dashboards estadísticos para toma de decisiones.
* Aplicar principios de seguridad, escalabilidad y buenas prácticas de desarrollo.

**3. TECNOLOGÍAS UTILIZADAS**

| **Tecnología** | **Propósito** |
| --- | --- |
| **Laravel** | Backend, API REST, autenticación |
| **Angular** | Frontend SPA, UI responsiva |
| **MySQL** | Base de datos relacional |
| **Laravel Sanctum** | Autenticación con token |
| **XAMPP / Composer / Node.js / Angular CLI** | Herramientas de entorno de desarrollo |

**ARQUITECTURA GENERAL DEL SISTEMA**

Gestock está construido bajo una arquitectura cliente-servidor desacoplada. El frontend, desarrollado en Angular, consume los servicios expuestos por una API RESTful creada con Laravel. Esta separación permite escalar, mantener y desplegar cada componente de manera independiente.

**Flujo General:**

1. El usuario interactúa con la interfaz Angular.
2. Angular realiza peticiones HTTP a la API Laravel.
3. Laravel procesa, valida y responde mediante JSON.
4. Los datos son almacenados en una base de datos MySQL.
5. Se disparan notificaciones y logs según reglas del negocio.

**5. ESTRUCTURA DEL PROYECTO**

**Backend (Laravel)**

Texto

El contenido generado por IA puede ser incorrecto.

**Frontend (Angular)**

Pantalla de computadora con letras

El contenido generado por IA puede ser incorrecto.

**Integración entre Angular y Laravel**

* Las peticiones HTTP se hacen desde **api.service.ts,** incluyendo los tokens de autenticación proporcionados por Laravel Sanctum.
* La API responde con estructuras JSON que el frontend procesa para mostrar la información o manejar errores.
* Las rutas del backend están protegidas por middleware (**auth:sanctum**), garantizando que solo usuarios autenticados puedan acceder a operaciones sensibles.

**6. BACKEND – LARAVEL**

El backend está construido con Laravel, un framework PHP que permite crear APIs RESTful de forma limpia y estructurada. Toda la lógica del servidor, incluyendo autenticación, validaciones, notificaciones y control de acceso, se desarrolla en esta capa.

**6.1 MODELOS PRINCIPALES**

Los modelos de Eloquent representan las entidades de la base de datos. Cada modelo está vinculado a una tabla y define sus relaciones.

* **Producto**
  + Atributos: codigo, nombre, descripcion, precio, stock, stock\_minimo, activo, categoria\_id, proveedor\_id, user\_id
  + Relaciones: pertenece a una **Categoría**, un **Proveedor** y un **Usuario**
  + Funciones clave: validación de stock bajo, notificaciones, filtros
* **User**
  + Atributos: name, email, password, role, is\_admin, activo
  + Funciones clave: autenticación, control de acceso, bloqueo de usuario
  + Relación con movimientos y productos creados
* **Categoría**
  + Atributos: nombre, descripcion, activo
  + Relación: tiene muchos productos
* **Proveedor**
  + Atributos: nombre, nit, telefono, direccion, email, contacto
  + Relación: tiene muchos productos
* **Movimiento**
  + Atributos: tipo (entrada/salida), cantidad, producto\_id, user\_id, fecha, observacion
  + Registra cambios de inventario con trazabilidad completa

**6.2 CONTROLADORES**

Los controladores reciben las solicitudes, las validan, gestionan la lógica de negocio y devuelven respuestas estructuradas.

* **AuthController**
  + Métodos: login, register, logout, user
  + Implementa Laravel Sanctum para autenticación basada en tokens
* **ProductoController**
  + Métodos: index, store, update, destroy, show
  + Lógica: validaciones, control de stock, búsqueda, filtrado, imágenes
* **MovimientoController**
  + Registra cada entrada o salida de inventario
  + Actualiza el stock automáticamente
* **CategoriaController y ProveedorController**
  + CRUD completo, validaciones, relaciones

**6.3 RUTAS PRINCIPALES (routes/api.php)**

Texto

El contenido generado por IA puede ser incorrecto.

**6.4 SEGURIDAD Y MIDDLEWARE**

* Se utiliza auth:sanctum para proteger todas las rutas que requieren autenticación.
* Se verifica el rol del usuario manualmente o mediante políticas para operaciones críticas.
* Los datos se validan usando FormRequest personalizados o directamente en los controladores.

**6.5 NOTIFICACIONES Y LOGGING**

* Cuando el stock de un producto baja del mínimo, se genera una **notificación automática por correo** a los administradores.
* Laravel utiliza el sistema de **eventos y notificaciones** para enviar correos de forma asíncrona.
* Todos los movimientos, accesos y operaciones sensibles se registran en archivos de log (storage/logs/laravel.log), lo que permite auditoría y trazabilidad.

**7. FRONTEND – ANGULAR**

La interfaz del sistema está desarrollada con **Angular**, un framework de frontend moderno y robusto que permite construir aplicaciones de una sola página (SPA). El frontend se comunica directamente con la API de Laravel a través de peticiones HTTP autenticadas.

**7.1 ESTRUCTURA GENERAL**

**Texto

El contenido generado por IA puede ser incorrecto.**

**7.2 COMPONENTES PRINCIPALES**

| **Componente** | Funcionalidad |
| --- | --- |
| **LoginComponent** | Formulario de inicio de sesión |
| **DashboardComponent** | Vista principal con estadísticas del sistema |
| **ProductoComponent** | Listado y formulario para productos |
| **ProveedorComponent** | Gestión de proveedores |
| **MovimientoComponent** | Registro de entradas y salidas de inventario |
| **NavbarComponent** | Barra de navegación lateral o superior con logout y enlaces de acceso rápido |
| **UsuarioComponent** | Administración de usuarios y roles (opcional) |

**7.3 SERVICIOS Y CONSUMO DE API**

**api.service.ts**

Este servicio centraliza todas las peticiones HTTP al backend. Utiliza HttpClient y observables (RxJS) para manejar asincronía y errores.

**Ejemplo de métodos:**

**Texto

El contenido generado por IA puede ser incorrecto.**

Todos los métodos agregan el token JWT en el encabezado usando un interceptor o directamente desde el localStorage.

**7.4 AUTENTICACIÓN Y GUARDIAS**

* El token generado por Laravel Sanctum se almacena localmente y se envía en cada petición.
* Las rutas están protegidas con AuthGuard, que verifica si el usuario está autenticado.
* El menú de navegación cambia dinámicamente según el rol del usuario (admin o estándar).

**7.5 RUTEO Y NAVEGACIÓN**

El archivo app-routing.module.ts define las rutas del frontend:

**Texto

El contenido generado por IA puede ser incorrecto.**

**7.6 INTERFAZ Y EXPERIENCIA DE USUARIO**

* Se utiliza Angular Material para una UI moderna, responsiva y accesible.
* Validaciones de formularios con mensajes claros.
* Confirmaciones al eliminar o modificar datos.
* Alertas visuales en tiempo real ante errores o eventos exitosos.
* Vista tipo dashboard con gráficas, estadísticas y alertas de stock.

**8. BASE DE DATOS Y MIGRACIONES**

La base de datos está diseñada con relaciones claras y normalizadas. Laravel gestiona la creación de tablas mediante migraciones, y el poblado de datos de prueba mediante seeders.

**8.1 ESTRUCTURA DE TABLAS**

| **Tabla** | Descripción |
| --- | --- |
| **productos** | Almacena información de productos: nombre, stock, precio, proveedor, etc. |
| **categorias** | Agrupa productos por tipo. Relación uno a muchos. |
| **proveedores** | Información de proveedores asociados a productos. |
| **movimientos** | Registra entradas y salidas con fecha, cantidad, tipo, usuario y producto. |
| **users** | Usuarios con roles y autenticación. |

**8.2 MIGRACIONES Y SEEDERS**

* Las migraciones definen la estructura de cada tabla.
* Los seeders crean datos base como: productos de ejemplo, usuarios, categorías y movimientos.
* Se ejecutan con:

**Texto

El contenido generado por IA puede ser incorrecto.**

**9. ROLES Y PERMISOS**

Gestock implementa control de acceso mediante roles definidos en el modelo **User**.

| **Rol** | **Permisos** |
| --- | --- |
| **Administrador** | Acceso total a todos los módulos, usuarios, configuración y reportes. |
| **Usuario estándar** | Acceso limitado a inventario, consultas y creación de movimientos. |

* Las validaciones de permisos se realizan en los controladores y en el frontend (Angular) para ocultar opciones no permitidas.
* Laravel usa middleware para proteger las rutas y asegurar que cada acción tenga el permiso correspondiente.

**10. FLUJO DE AUTENTICACIÓN Y SEGURIDAD**

**Paso a paso**

1. El usuario ingresa sus credenciales en el formulario de Angular.
2. Angular envía una petición POST /api/login al backend.
3. Laravel valida las credenciales y genera un token con Sanctum.
4. El token se guarda en el frontend (localStorage o cookies seguras).
5. En cada solicitud protegida, el token se envía automáticamente.
6. Laravel valida el token antes de procesar la solicitud.
7. Si el usuario no tiene permisos suficientes, retorna un 403 Unauthorized.

**Buenas prácticas implementadas**

* Protección de rutas con auth:sanctum.
* Middleware personalizado para verificar roles.
* Validación estricta en formularios tanto en frontend como en backend.
* Uso de logs para auditoría de movimientos, accesos y errores.
* Separación de ambientes mediante .env.

**11. GLOSARIO DE TÉRMINOS**

| **Término** | Definición |
| --- | --- |
| **CRUD** | Operaciones básicas: Crear, Leer, Actualizar, Eliminar. |
| **SPA** | Single Page Application. Aplicación web que no recarga la página completa. |
| **Sanctum** | Sistema de autenticación por tokens en Laravel. |
| **Seeder** | Script que genera datos de prueba en la base de datos. |
| **Middleware** | Interceptor de peticiones para aplicar lógica antes de continuar. |
| **Eloquent** | ORM de Laravel para manejar modelos y relaciones de BD. |

**14. CRÉDITOS Y COLABORADORES**

Este proyecto fue desarrollado por estudiantes de la Universidad del Valle para la asignatura Desarrollo de Software I:

* Jhon Gomez
* Janer Mena
* Hinojosa Lopez
* Leornardo Barahona
* Santiago Cadavid

**Profesor:** Lucio Gómez