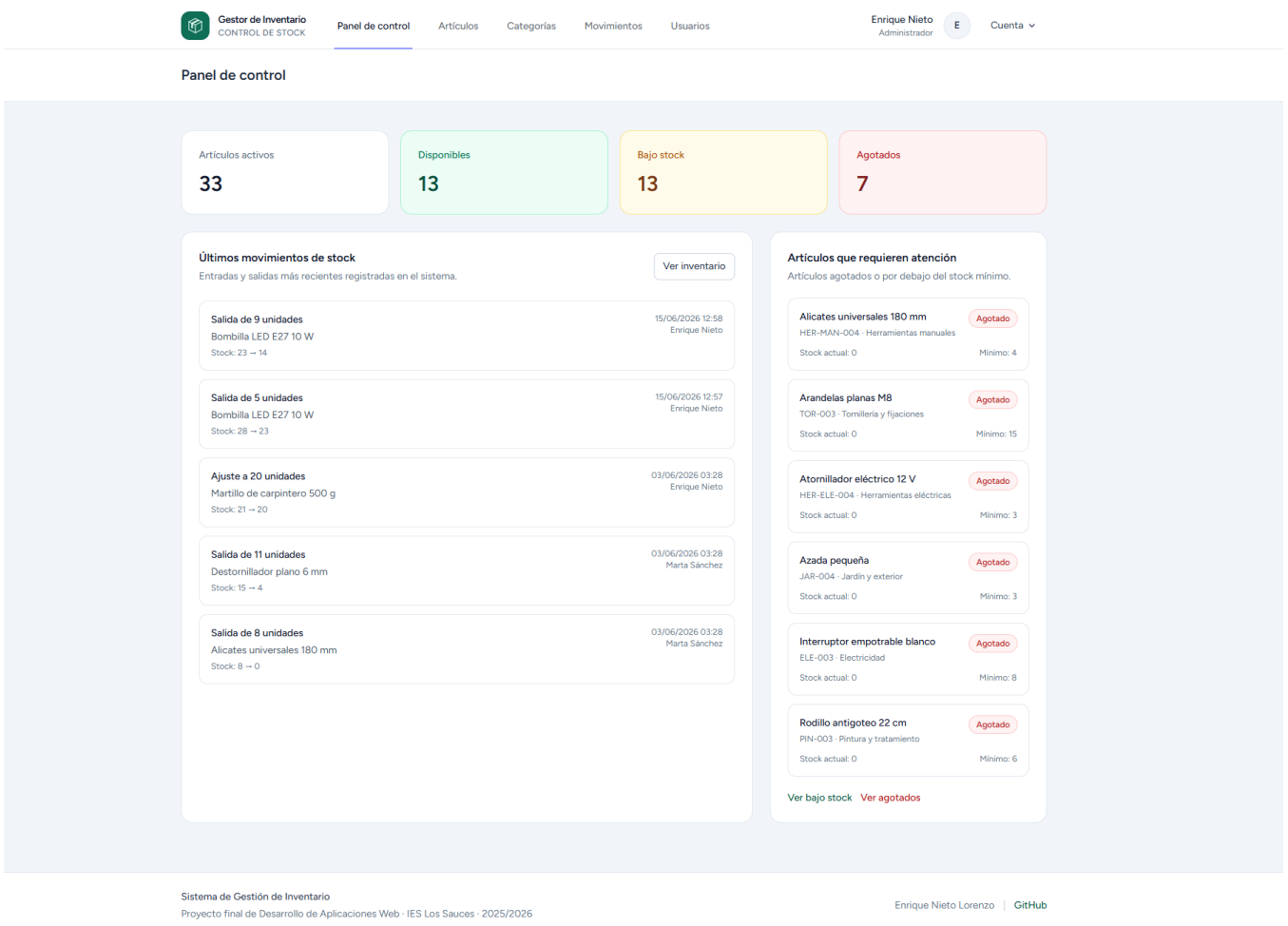
# Sistema de Gestión de Inventario

Aplicación web Laravel para la gestión de artículos, categorías, stock, usuarios y API REST

**Enrique Nieto Lorenzo**
Proyecto Intermodular DAW
IES Los Sauces — Benavente
Curso 2025/2026



# Introducción y contextualización

### Contexto organizativo

- Las pequeñas organizaciones también necesitan controlar sus existencias.
- Muchas no requieren un sistema ERP completo.

### Situación inicial

- La información puede quedar repartida entre documentos y hojas de cálculo.
- La aplicación se plantea como una herramienta web interna.

### Ámbito de aplicación

- El caso representativo utilizado es una pequeña ferretería.
- El planteamiento puede adaptarse a otros pequeños almacenes.

**El objetivo es centralizar el inventario en una herramienta web sencilla de utilizar.**

# Problema detectado y necesidades

## Problemas detectados

- Dificultad para conocer el stock real disponible.
- Falta de un historial completo de operaciones.
- No se puede identificar al responsable de cada cambio.
- Riesgo de introducir datos incoherentes.
- Dificultad para localizar artículos bajo mínimos.
- Información dispersa entre diferentes documentos.

## Necesidades planteadas

- Centralizar la información del inventario.
- Registrar todas las entradas, salidas y ajustes.
- Mantener la trazabilidad de las operaciones.
- Diferenciar los permisos según el tipo de usuario.
- Consultar rápidamente los artículos críticos.
- Disponer de una solución sencilla de utilizar.

# Solución propuesta y alcance

## Solución propuesta

- Aplicación web interna desarrollada con Laravel.
- Gestión centralizada de artículos y categorías.
- Registro de entradas, salidas y ajustes de stock.
- Historial completo de todos los movimientos.
- Panel de control con indicadores del inventario.
- API REST pública de solo lectura.

## Alcance de la primera versión

- Control de existencias mediante unidades agregadas.
- Dos perfiles internos: empleado y administrador.
- Acceso mediante cuentas creadas por un administrador.
- Gestión por número de serie aún no implementada.
- Los ejemplares individuales se plantean como mejora futura.

# Objetivo general

**Desarrollar**, **documentar** y **desplegar** una aplicación web de gestión de inventario para pequeñas y medianas organizaciones, utilizando Laravel y una base de datos relacional, con control de artículos, categorías, usuarios y movimientos de stock, y aplicando criterios de **seguridad**, **trazabilidad** y mantenibilidad.

# Objetivos específicos

## Análisis y diseño

1. Estudiar Laravel y aplicar sus componentes al proyecto.
2. Identificar los requisitos funcionales y no funcionales.
3. Diseñar los casos de uso y la navegación.
4. Diseñar el modelo conceptual y físico de datos.

## Desarrollo e implantación

5. Implementar artículos, categorías, usuarios y movimientos.
6. Incorporar búsqueda, filtros, dashboard e historial.
7. Aplicar autenticación, autorización y validación.
8. Crear una API REST pública de solo lectura.
9. Desplegar la aplicación mediante Plesk y HTTPS.
10. Documentar el desarrollo y las decisiones técnicas.

# Tecnologías utilizadas

### Backend

- PHP 8.3
- Laravel 13

### Persistencia de datos

- MariaDB
- Eloquent ORM
- Migraciones

### Interfaz

- Blade
- Tailwind CSS
- Laravel Breeze

### Herramientas de desarrollo

- Composer · Node.js y npm · Vite
- Git y GitHub para el control de versiones
- Visual Studio Code y WSL2 como entorno

### Producción

- Plesk
- HTTPS

# Justificación del uso de Laravel

## Por qué encaja en el proyecto

- Proporciona una organización basada en convenciones.
- Separa rutas, controladores, modelos y vistas.
- Eloquent facilita el trabajo con relaciones entre entidades.
- Las migraciones permiten versionar la base de datos.

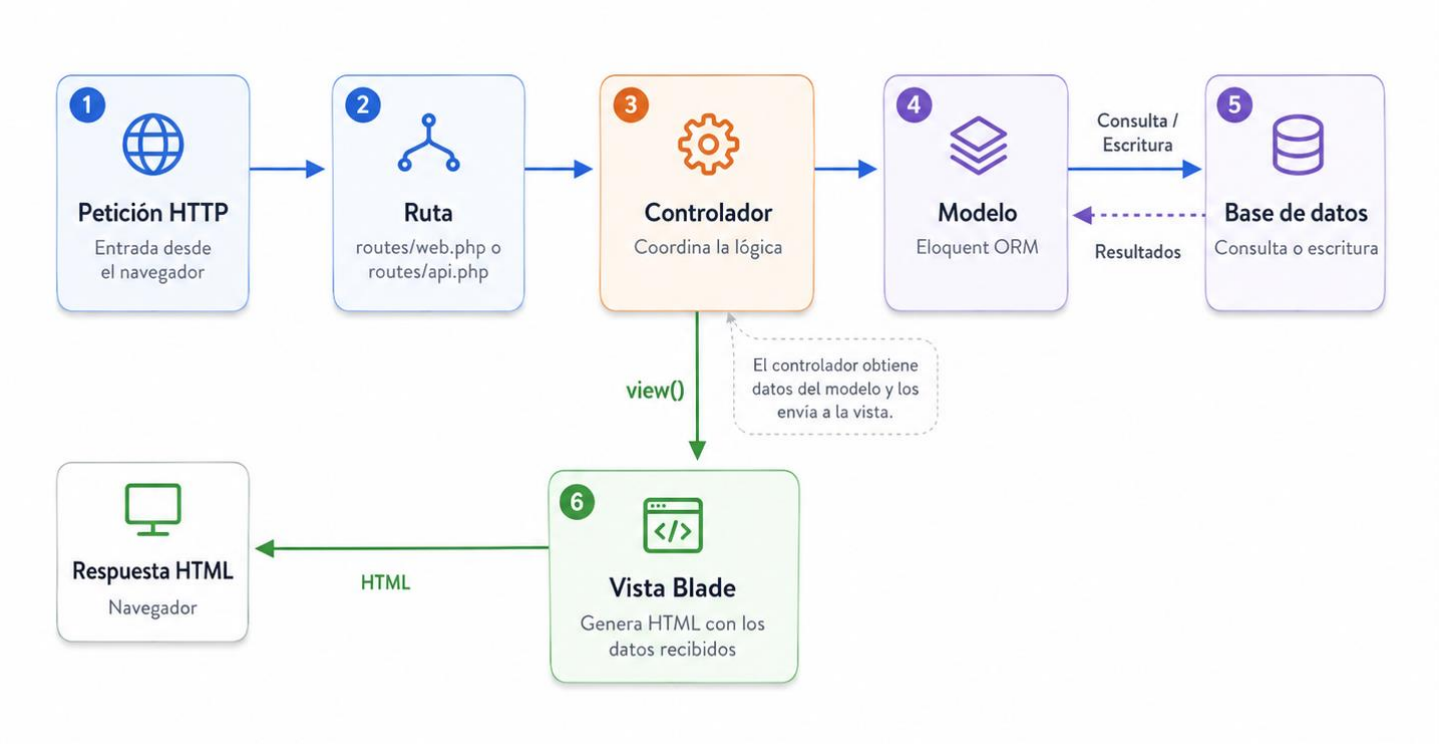
Ver migraciones

## Componentes que se aprovechan

- FormRequest centraliza la validación de los formularios.
- Middleware y Policies controlan los permisos.
- Blade facilita la creación de las vistas.
- Artisan, Composer y Vite agilizan el desarrollo.

**Laravel permite mantener separadas las distintas responsabilidades de la aplicación.** Estructura de directorios

# Arquitectura MVC de la aplicación




## RECORRIDO DE UNA PETICIÓN

- La ruta recibe la petición del usuario.
- El middleware comprueba el acceso.
- FormRequest valida los datos recibidos.
- El controlador coordina la operación.
- Eloquent consulta o modifica MariaDB. Eloquent
- Blade genera HTML o la API devuelve JSON.

- **Ejemplo aplicado: registro de una salida de stock.**

Ver diagrama de clases completo

# Actores, roles y casos de uso

### Usuario no registrado

- Consulta la página pública.
- Accede a la documentación.
- Inicia sesión.
- Consulta los endpoints públicos.

### Empleado

- Consulta el dashboard.
- Consulta artículos y categorías.
- Usa búsquedas y filtros.
- Consulta detalles e historiales.
- Gestiona su perfil.

### Administrador

- Incluye lo del empleado.
- Gestiona artículos y categorías.
- Gestiona usuarios.
- Registra movimientos de stock.

### Cliente API

- Lista artículos activos.
- Consulta artículos críticos.
- Busca artículos por SKU.

- **No existe registro público; las cuentas internas las crea un administrador.**

Ver diagrama de casos de uso

# Navegación del sistema

## ACCESO Y RECORRIDO PRINCIPAL


### Desde el dashboard

- Inventario
- Categorías
- Perfil
- Usuarios (solo administrador)
- Movimientos de stock (solo administrador)

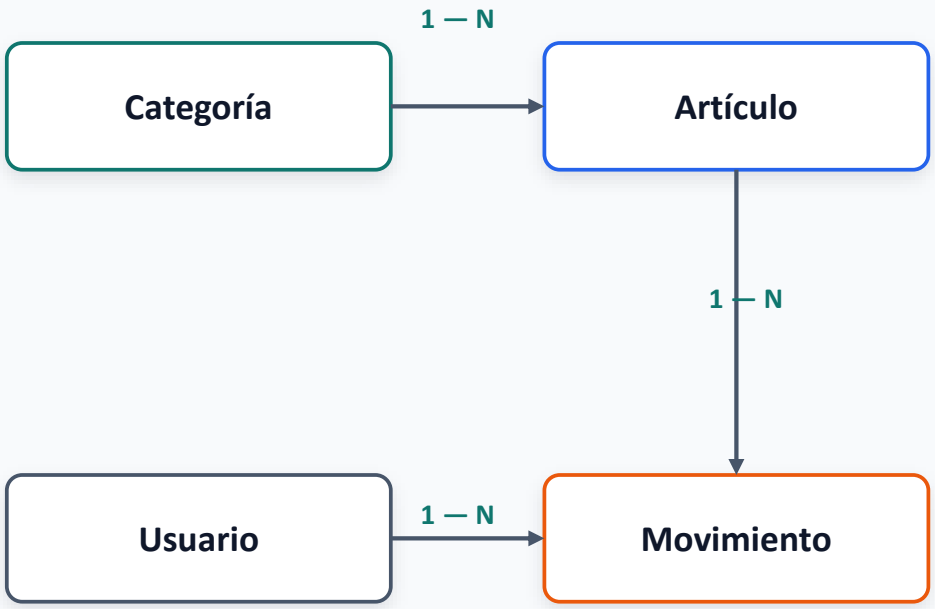
## ACCESO EXTERNO (API)


Acceso de solo lectura, separado del recorrido interno.

Acceso público · Consulta · Administración · API

Ver árbol de navegación

# Modelo entidad-relación




## RELACIONES

- Una categoría clasifica varios artículos; cada artículo pertenece a una categoría.
- Un artículo registra varios movimientos; cada movimiento pertenece a un artículo.
- Un usuario realiza varios movimientos; cada movimiento se asocia a un usuario.

- **El estado del artículo es un atributo derivado del stock actual y del stock mínimo.**

Ver modelo entidad-relación completo

# Modelo físico de datos

## TABLAS PRINCIPALES

- `users`
- `categories`
- `items`
- `stock_movements`

## RESTRICCIONES RELEVANTES

- El correo electrónico del usuario es único.
- El SKU del artículo es único.
- Cada artículo tiene una clave foránea hacia su categoría.
- Cada movimiento se relaciona con un artículo y un usuario.
- Los movimientos conservan el stock anterior y posterior.
- Los artículos usan baja lógica mediante is_active.
- Los movimientos no se modifican ni se eliminan.
- No se permite eliminar categorías con artículos asociados.

Ver modelo físico completo

# Desarrollo técnico: movimientos de stock y trazabilidad

**Stock anterior** → **Validación** → **Operación** → **Transacción** → **Stock posterior y registro**

## Tipos de movimiento

- Entrada: aumenta el stock disponible.
- Salida: reduce el stock disponible.
- Ajuste: fija el stock real tras un recuento.

## Información registrada

- Artículo afectado y usuario responsable.
- Tipo de movimiento y cantidad.
- Stock anterior y stock posterior.
- Fecha y observaciones.

## Reglas técnicas

- El stock no puede modificarse directamente.
- Una salida no puede dejar el stock en negativo.
- Las correcciones usan un movimiento compensatorio.
- Actualización y registro en una transacción.

**Cada cambio de stock queda registrado y puede ser explicado posteriormente.**

# Validación, autorización e integridad

## Autenticación

- Identifica al usuario mediante inicio de sesión.
- Mantiene la sesión del usuario autenticado.

## Autorización

- Los roles determinan las acciones permitidas.
- Los middleware protegen las rutas.
- Las Policies comprueban los permisos en el servidor.

## Validación

- FormRequest centraliza las reglas de los formularios.
- Se comprueban formatos, campos obligatorios y unicidad.
- Se comprueban las cantidades introducidas.

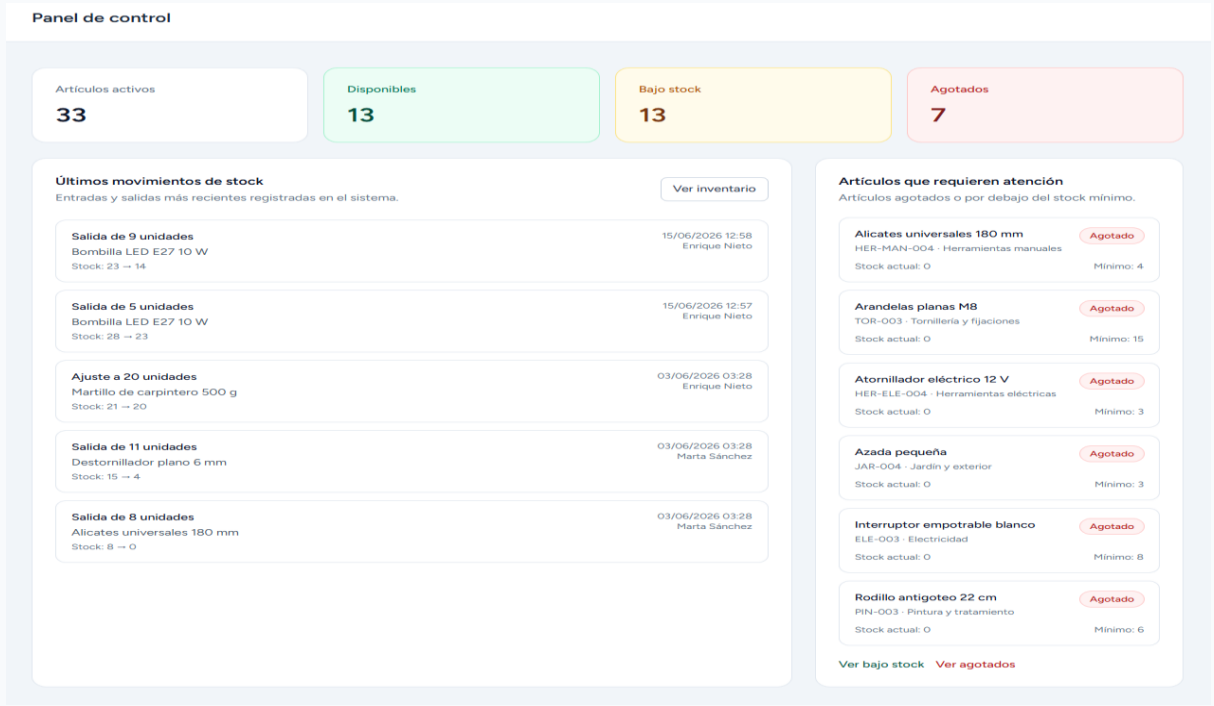
## Integridad de negocio

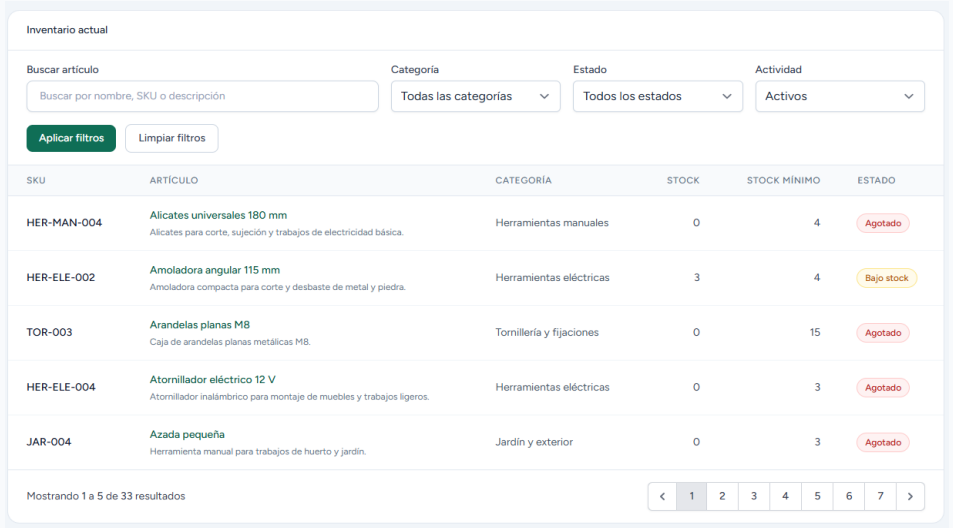
- No se permiten salidas superiores al stock.
- No se eliminan categorías con artículos asociados.
- El estado del artículo se calcula automáticamente.

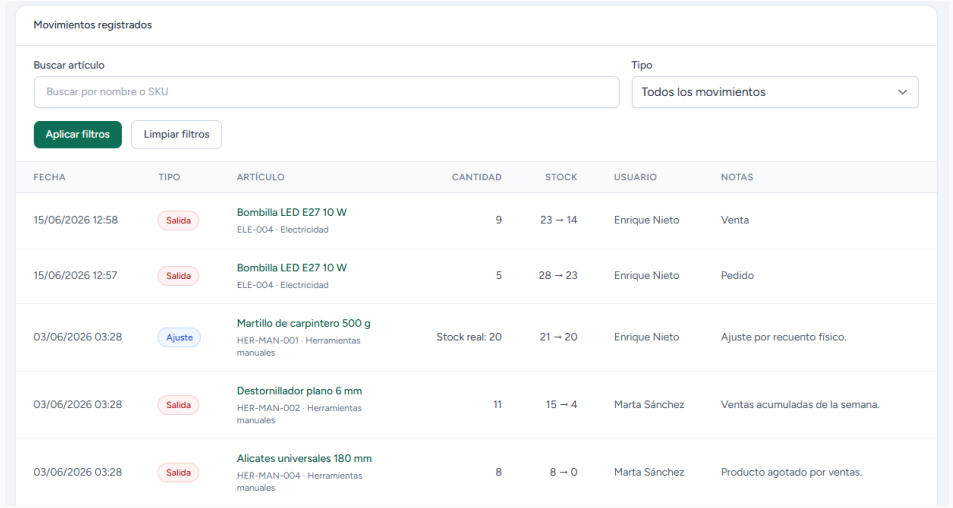
- **Ocultar botones en la interfaz no sustituye la autorización en el servidor.**

Ver Policies y FormRequest en el diagrama de clases

# Interfaz de usuario




## CARACTERÍSTICAS VISIBLES

- Consulta rápida del estado del inventario.
- Búsqueda por nombre, descripción o SKU.
- Filtros por categoría y estado.
- Diferenciación visual de los estados.
- Opciones adaptadas al rol del usuario.

# API REST

## ENDPOINTS DISPONIBLES

`GET /api/items`

Devuelve los artículos activos.

`GET /api/items/critical`

Devuelve artículos agotados o bajo mínimos.

`GET /api/items/{sku}`

Consulta un artículo mediante su SKU.

## CARACTERÍSTICAS

- Respuestas en formato JSON.
- Servicio público de solo lectura.
- No permite modificar el inventario.
- Puede ser consumida por un sistema externo.
- Utiliza los mismos datos que la interfaz web.

# Despliegue en producción

## ARQUITECTURA SIMPLIFICADA


## CONFIGURACIÓN

- Servidor administrado mediante Plesk con PHP 8.3.
- Base de datos MariaDB y certificado HTTPS activo.
- Raíz del sitio dirigida al directorio public/.
- Archivo .env fuera del acceso público.
- Recursos de frontend compilados con Vite.

### Incidencia real de despliegue

Error 500 en las rutas web por ausencia de la carpeta de vistas compiladas y problemas de permisos.

### Solución aplicada

Creación de la carpeta necesaria y corrección de permisos para permitir la compilación de las vistas Blade.

# Demostración funcional

# Conclusiones y líneas futuras

## Conclusiones

- Se ha desarrollado una aplicación funcional y desplegada.
- El inventario se gestiona de forma centralizada.
- Todos los cambios de stock mantienen trazabilidad.
- Los permisos diferencian consulta y administración.
- Laravel separa las responsabilidades del sistema.
- La API ofrece consultas externas sin modificar datos.

## Líneas futuras

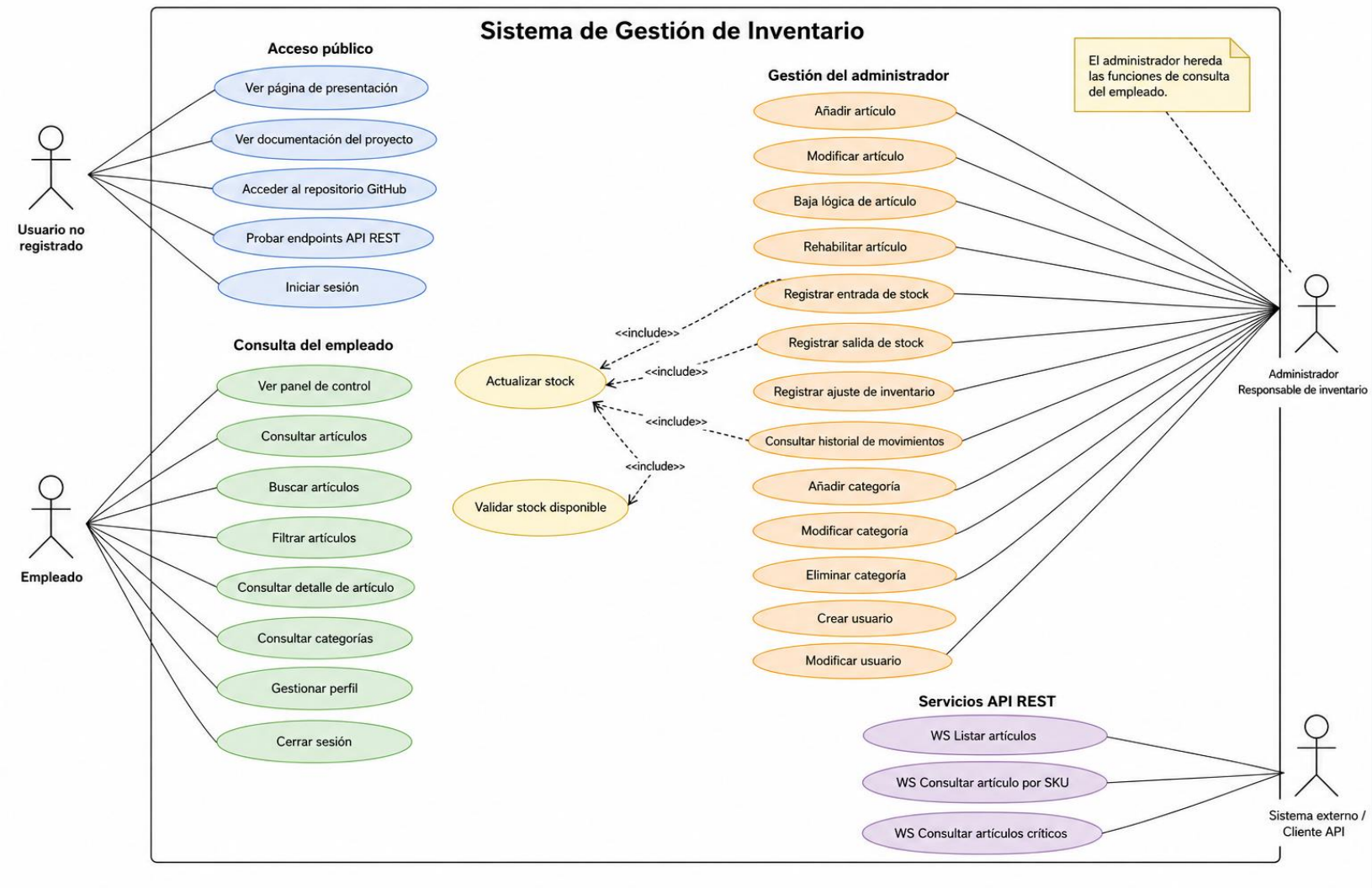
- Gestión de ejemplares con número de serie.
- Incorporación de proveedores y ubicaciones.
- Importación y exportación de datos.
- Notificaciones de stock mínimo.
- Gráficos e informes históricos.
- Más pruebas automatizadas y API con tokens.

**GRACIAS POR SU ATENCIÓN**
**Preguntas**

# Árbol de navegación completo

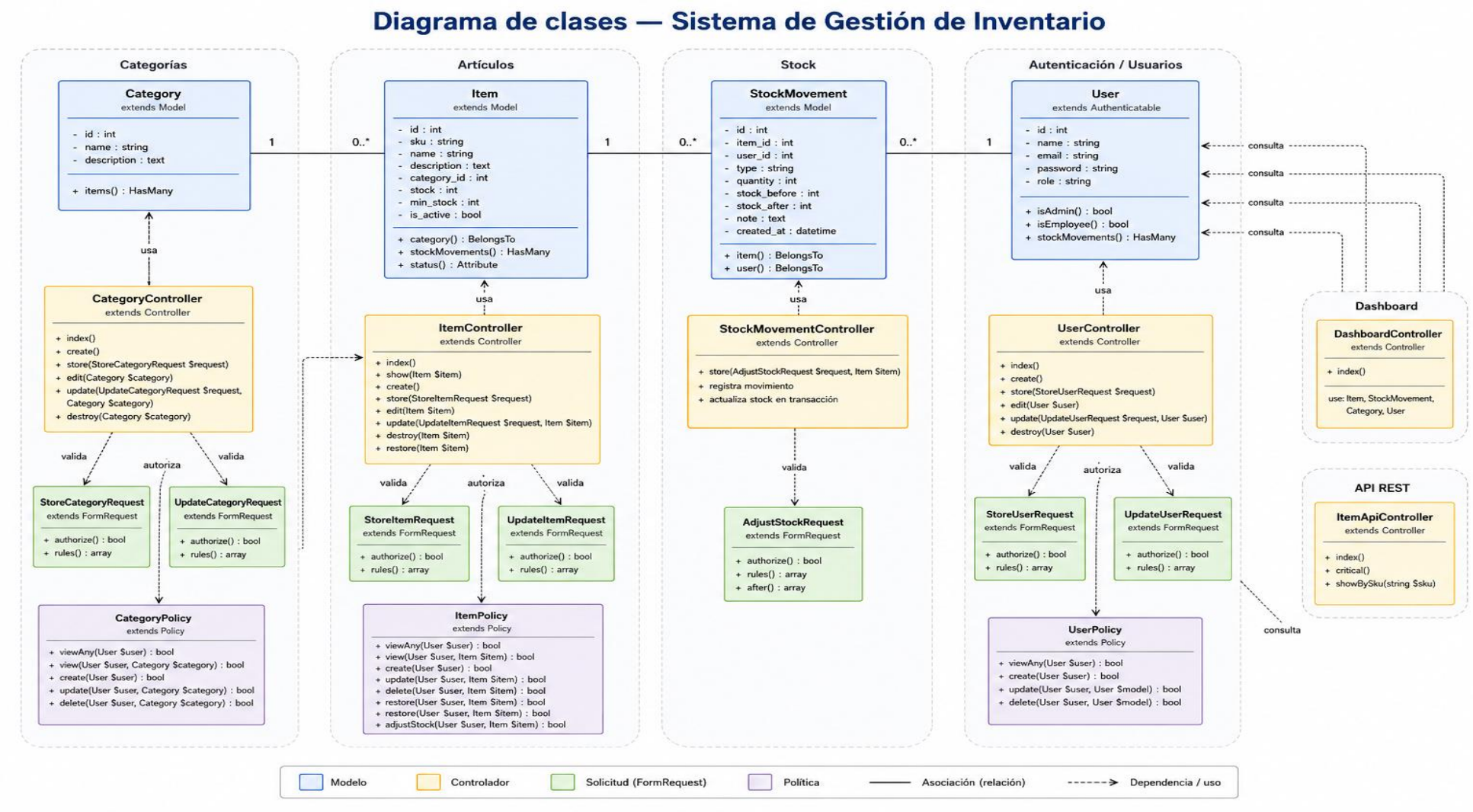

← Volver a navegación del sistema

# Diagrama completo de casos de uso




← Volver a actores y roles

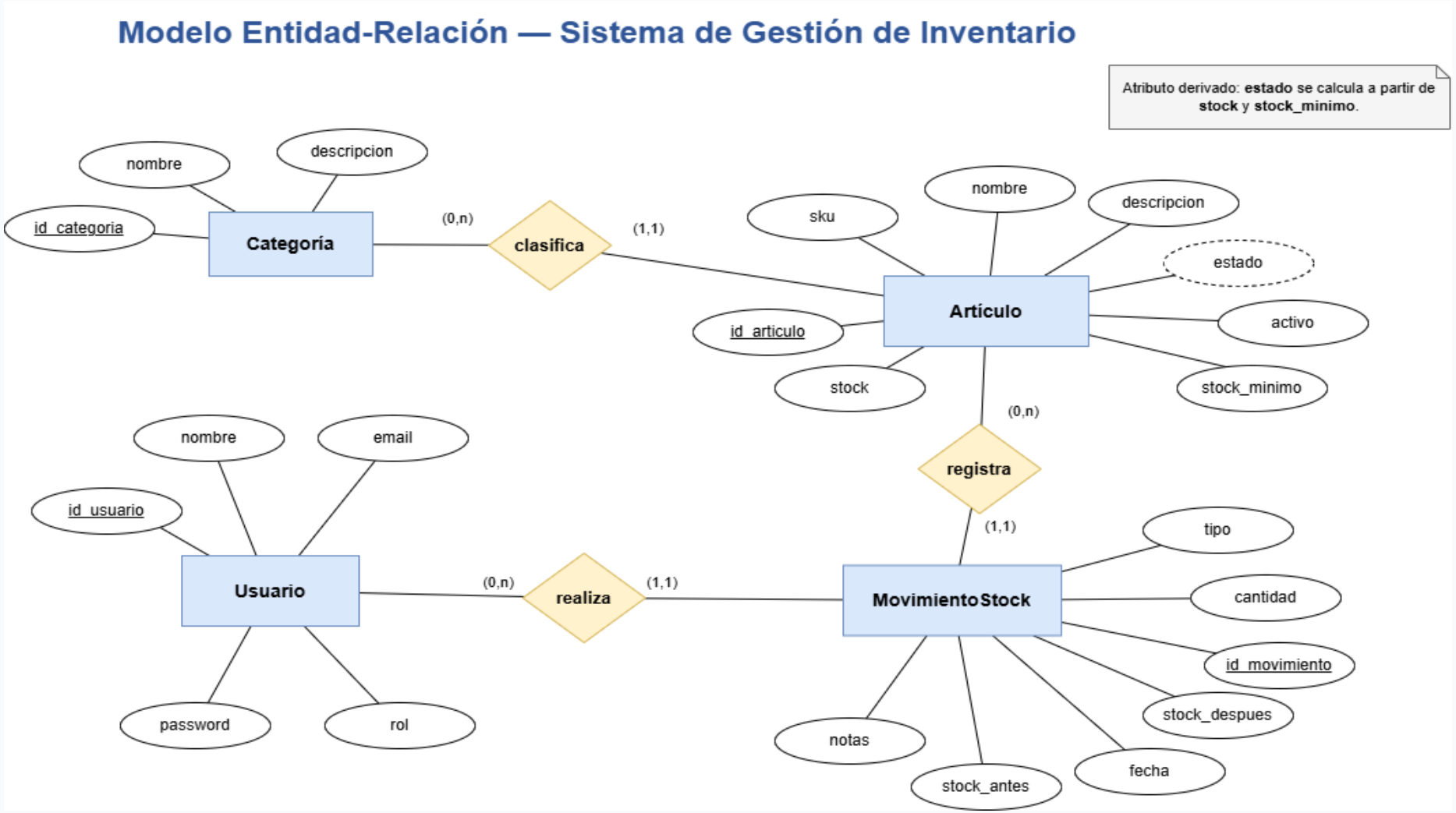
# Diagrama de clases




← Volver a arquitectura MVC

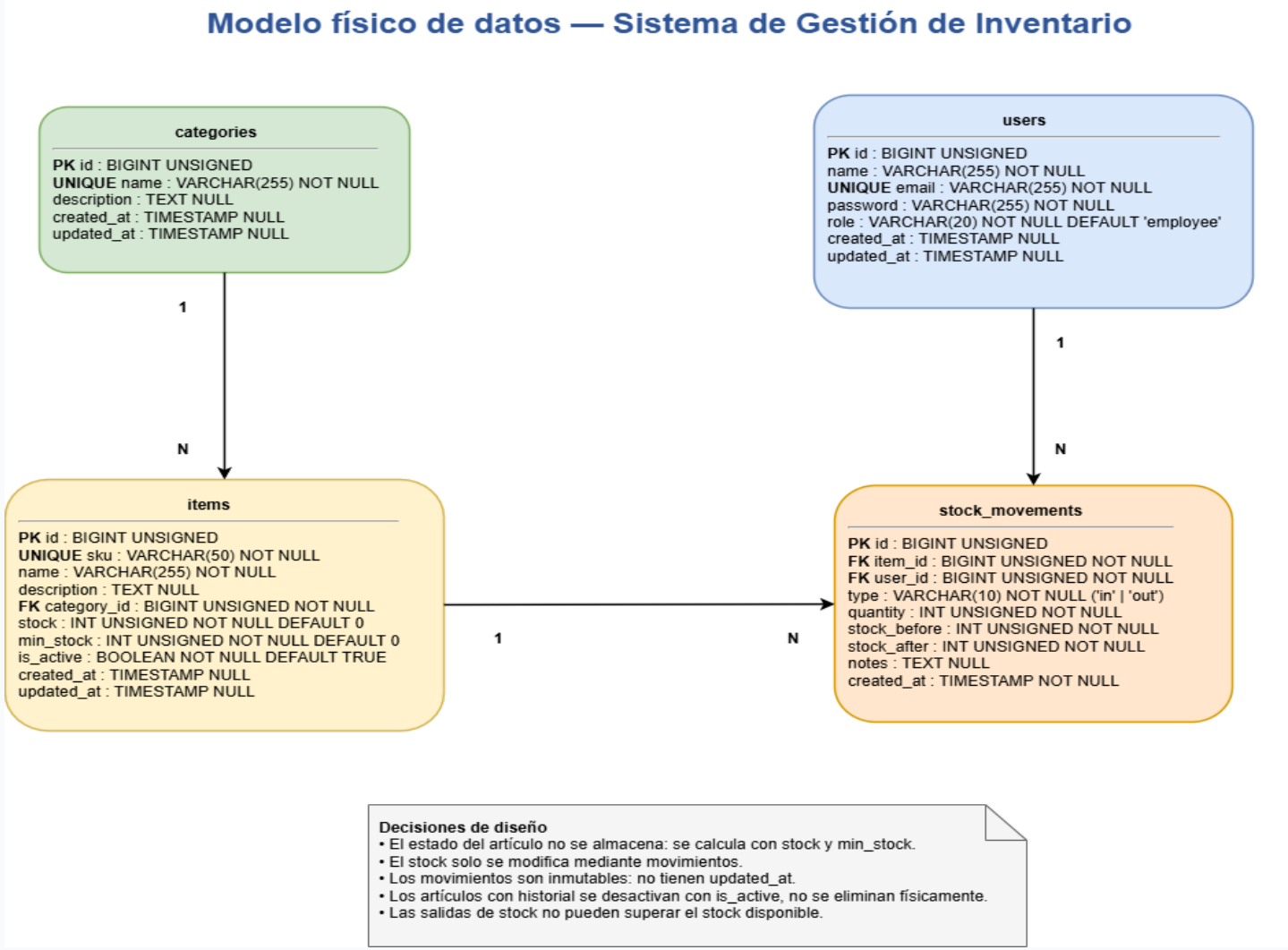
← Volver a validación y autorización

# Modelo entidad-relación completo




← Volver al modelo entidad-relación

# Modelo físico de datos completo




← Volver al modelo físico

# Estructura de directorios


← Volver

# Migraciones en Laravel

## Migraciones: control de versiones de la base de datos

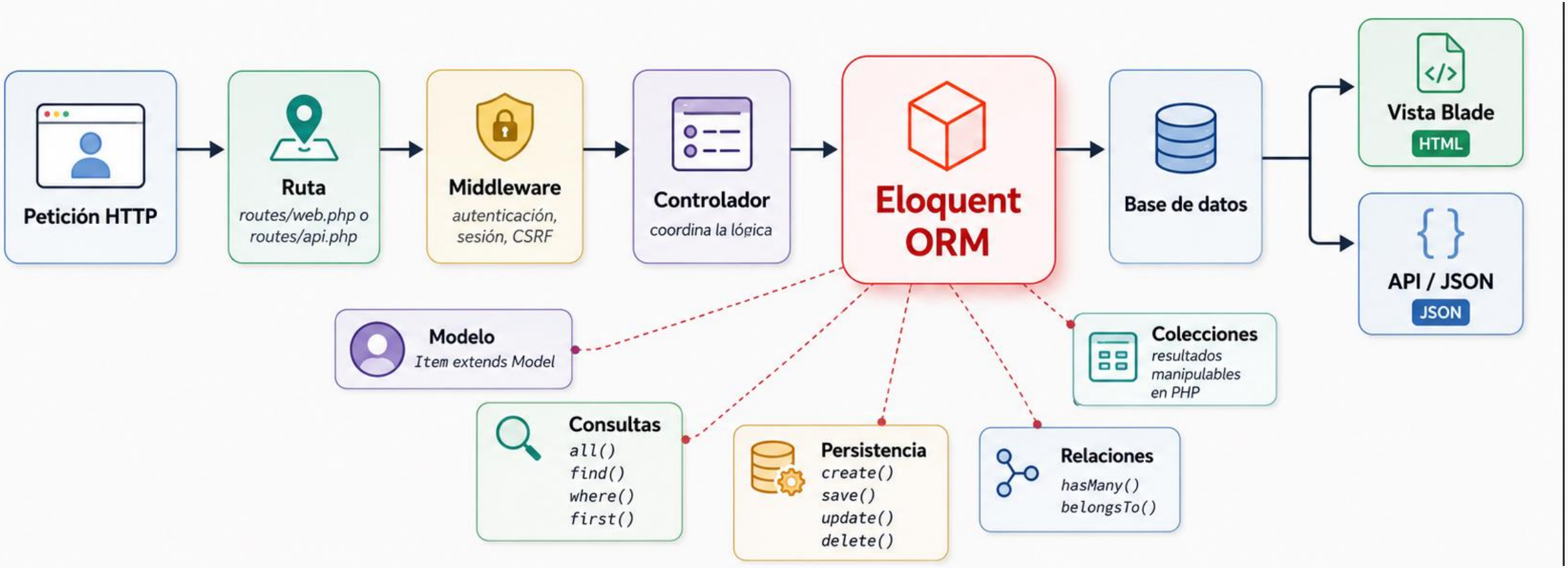
Laravel define los cambios de estructura de la base de datos mediante código reproducible.


**Ventaja principal**

La estructura de la base de datos permanece documentada, versionada y reproducible en cualquier entorno mediante comandos del proyecto.

← Volver

# Eloquent ORM en Laravel




← Volver