CIENTIFICA
TechNoLogica

ISSN:3119-7906, Vol.1Núm.1(2026): Enero-Marzo

https://doi.org/10.17268/tech1.1.2026.04

Arquitectura Web MVC aplicada al Desarrollo de una Plataforma de Gestión y E-commerce para Minimarkets

MVC Web Architecture Applied to the Development of a Management and E-commerce Platform for Convenience Stores

Arce Rivasplata Pedro M.[1,*], Cerna Hernandez Frank M.[1], Hernandez Huamán Juan C.[1], Morán Cerdan Josias C.[1], Sánchez Vasquez Zully J.[1]

[1] Escuela de Ingeniería Informática, Universidad Nacional de Trujillo, La Libertad, Peru.

*Autor correspondiente: parcer@unitru.edu.pe (P-ArceRivasplata)

RESUMEN

El presente estudio aborda el desarrollo de una plataforma web basada en la arquitectura Modelo–Vista–Controlador (MVC), orientada a la digitalización de la gestión comercial de un minimarket. El objetivo principal fue mejorar el desempeño comercial mediante funcionalidades de comercio electrónico y automatización del control de inventario en tiempo real. Como objetivos específicos se consideraron la implementación de un módulo de gestión de productos con control de stock automático, un sistema de carrito de compras con proceso de pago integrado y un servicio de delivery vinculado al flujo de pedidos.

El alcance del proyecto incluyó el diseño y desarrollo completo del frontend y backend utilizando Laravel 10 como framework principal, implementando patrones de diseño como Repository, Service Layer, Facade y Active Record mediante Eloquent ORM. La metodología empleada fue incremental, estructurada en cuatro iteraciones: módulo de catálogo, carrito de compras, autenticación de usuarios y panel administrativo. La evaluación del sistema se realizó mediante pruebas funcionales con doce usuarios reales y validación con el administrador. Los resultados evidenciaron la habilitación efectiva de compras en línea 24/7, reducción del 85% en errores de registro de inventario y un incremento proyectado del 40% en alcance comercial. La arquitectura MVC permitió una separación clara de responsabilidades, facilitando mantenimiento y escalabilidad. Se concluye que la combinación de MVC con patrones de diseño constituye una estrategia técnicamente viable y replicable para la digitalización de minimarkets, permitiendo la gestión integral de operaciones comerciales y la expansión del canal de ventas mediante e-commerce.

Palabras clave: Arquitectura MVC, Comercio electrónico, Laravel, Gestión de inventario, Control de stock en tiempo real, Patrones de diseño.

ABSTRACT

This study presents the development of a web platform based on the Model–View–Controller (MVC) architecture, aimed at digitalizing the commercial management of a minimarket. The main objective was to improve business performance through e-commerce functionalities and real-time inventory control automation. Specific objectives included implementing a product management module with automatic stock control, a shopping cart system with integrated payment processing, and a delivery service linked to the order flow. The project scope covered the full frontend and backend development using Laravel 10.x as the main framework, applying design patterns such as Repository, Service Layer, Facade, and Active Record through Eloquent ORM. An incremental methodology structured in four iterations was employed: catalog module, shopping cart, user authentication, and administrative panel. System evaluation involved functional testing with twelve real users and validation with the store administrator. Results showed effective 24/7 online purchases, an 85% reduction in inventory registration errors, and a projected 40% increase in commercial reach. The MVC architecture enabled clear separation of responsibilities, facilitating maintenance and scalability. It is concluded that combining MVC with structural design patterns constitutes a technically viable and replicable strategy for digitalizing minimarkets, allowing comprehensive management of commercial operations and expansion of the sales channel through e-commerce.

Keywords: *Arquitectura MVC, Comercio electrónico, Laravel, Gestión de inventario, Control de stock en tiempo real, Patrones de diseño.*

1. INTRODUCCIÓN

La digitalización comercial se ha consolidado como una estrategia clave para mejorar la competitividad en el sector minorista, particularmente en negocios de pequeña escala que enfrentan limitaciones físicas, geográficas y operativas [1][2]. En este contexto, diversos estudios han abordado el desarrollo de plataformas de comercio electrónico orientadas a pequeñas y medianas empresas, destacando su impacto en la ampliación del mercado, la optimización de procesos administrativos y la mejora en la gestión de la información.

Entre los trabajos relevantes se encuentran los de López, Morales y Vega (2011) [3], quienes proponen una aplicación web para PyMES enfocada en la gestión comercial utilizando ASP.NET y una arquitectura en tres capas. Sin embargo, su enfoque no aborda la gestión automática de inventario ni la integración de servicios de delivery. Posteriormente, García, Pérez y Torres (2018) [4] desarrollaron un sistema web utilizando PHP y MySQL bajo arquitectura MVC para la gestión de ventas e inventarios, aunque sin emplear frameworks robustos ni patrones de diseño avanzados. De manera complementaria, Vallejos Velarde (2018) [5] profundiza en el control sistemático del inventario a través de sistemas web, validando la viabilidad de PHP y MySQL para esta funcionalidad.

En el ámbito de arquitecturas de software, Pressman y Maxim (2020) [6] establecen que el patrón MVC es fundamental para el desarrollo de aplicaciones web mantenibles y escalables, mientras que Gamma et al. (1995) [7] documentan patrones de diseño reutilizables que complementan la arquitectura principal. Laudon y Traver (2022) [8] enfatizan que el e-commerce B2C ha transformado las relaciones comerciales, permitiendo disponibilidad 24/7 y reducción de costos operativos.

El presente estudio aborda el caso de Minimarket Tattos, un negocio local sin presencia digital, ubicado fuera del centro urbano en la provincia de Chepén, Perú. Esta ubicación genera una desventaja competitiva frente a establecimientos que cuentan con plataformas en línea y servicios de entrega a domicilio. La problemática principal identificada incluye: (a) ausencia de canal de venta digital, limitando las ventas al horario físico de atención; (b) gestión manual del inventario con alto índice de errores en el control de stock; (c) falta de sistema integrado para delivery; y (d) ausencia de herramientas de análisis y reporte para toma de decisiones.

Ante esta problemática, se propone un enfoque tecnológico basado en la arquitectura MVC implementada mediante el framework Laravel 10, aplicando una metodología de desarrollo incremental para diseñar una plataforma web integral que combine

funcionalidades de comercio electrónico, gestión automatizada de inventarios y administración interna mediante patrones de diseño estructurales y de comportamiento.

El objetivo general de este trabajo consiste en desarrollar e implementar una solución tecnológica basada en arquitectura MVC que permita digitalizar los procesos comerciales de un minimarket tradicional mediante una plataforma escalable y replicable. Como objetivos específicos, se plantea: (a) diseñar e implementar un sistema de comercio electrónico con catálogo dinámico y carrito de compras; (b) desarrollar un módulo de gestión de inventario con actualización automática de stock en tiempo real; (c) implementar un sistema de gestión de pedidos integrado con servicio de delivery motorizado; (d) construir un panel administrativo con generación de reportes en formatos PDF y Excel; y (e) aplicar patrones de diseño que faciliten la mantenibilidad y extensibilidad del sistema.

2. MATERIALES Y METODOS

2.1 Diseño de Investigación

Se seleccionó como unidad de análisis el minimarket "Tattos", ubicado en el centro poblado de Chepén (La Libertad, Perú). El negocio maneja un inventario de 150-200 productos de consumo básico y cuenta con dos empleados en turnos alternos. Antes de la intervención, solo disponía de un sistema de punto de venta (POS) local sin conectividad web, limitando su alcance comercial por su ubicación alejada del centro urbano.

La justificación del caso se basa en tres aspectos: (1) la necesidad de digitalización ante la desventaja competitiva por su ubicación periférica, (2) la representatividad del establecimiento respecto a minimarkets de tamaño similar en zonas no céntricas, y (3) la viabilidad de implementar tecnología en un contexto de recursos limitados. El estudio abarcó desde el análisis de requerimientos hasta la validación funcional del sistema, con un periodo total de desarrollo de 13 días naturales.

2.2 Arquitectura del Sistema Basada en MVC

El sistema se desarrolló empleando el framework Laravel 10 sobre PHP 8.2, el cual implementa nativamente el patrón arquitectónico Modelo–Vista–Controlador (MVC). Esta arquitectura separa las responsabilidades del sistema en tres componentes interconectados:

- **Modelo (Model)**: Gestiona la lógica de negocio y el acceso a datos mediante el ORM Eloquent, que permite la interacción con la base de datos MySQL 8.0 utilizando sintaxis orientada a objetos en lugar de consultas SQL directas. Los modelos representan las entidades del negocio (*Producto, Carrito, CarritoItem y Order.)* y sus relaciones.

app > Models > Producto.php > Producto

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Producto extends Model
{
    protected $fillable = [
        'nombre',
        'precio',
        'imagen',
        'descripcion',
        'categoria_id',
        'unidad_medida',
        'stock'
    ];

    public function categoria()
    {
        return $this->belongsTo(Categoria::class);
    }
}
```

Fig.1 Definición del modelo Eloquent Producto.

- **Vista (View)**: Contiene las plantillas de presentación desarrolladas con el motor Blade de Laravel, que permite la creación de interfaces dinámicas mediante directivas PHP embebidas en HTML. Las vistas se encuentran organizadas en el directorio *resources/views* y son renderizadas por los controladores.

rces > views > productos > show.blade.php

```
@extends('layouts.layout')

@section('content')

<div class="max-w-7xl mx-auto px-4 py-8">

    {{-- ================== PRODUCTO PRINCIPAL ================== --}}
    <div class="grid grid-cols-12 gap-8 bg-white rounded-xl shadow p-6">

        {{-- IMAGEN --}}
        <div class="col-span-12 md:col-span-5 flex justify-center">
            <img
                src="{{ $producto->imagen }}"
                class="w-full max-w-md rounded-xl object-cover"
                alt="{{ $producto->nombre }}"
            >
        </div>

        {{-- INFO --}}
        <div class="col-span-12 md:col-span-4">
            <span class="text-sm text-gray-500 uppercase">
                {{ $producto->categoria->nombre ?? 'Producto' }}
            </span>

            <h1 class="text-2xl font-bold mt-2">
                {{ $producto->nombre }}
            </h1>
```

Fig.2 Interfaz de usuario mediante el motor de plantillas Blade.

- **Controlador (Controller)**: Coordina el flujo de información entre modelos y vistas, procesando las solicitudes HTTP, ejecutando lógica de negocio mediante los modelos y retornando las respuestas apropiadas. Los controladores se ubican en *app/Http/Controllers*.

app > Http > Controllers > ProductoController.php > ProductoController

```
<?php

namespace App\Http\Controllers;

use App\Models\Producto;
use App\Models\Categoria;
use Illuminate\Http\Request;
use App\Services\ProveedorSheetService;

class ProductoController extends Controller
{
    // ==============================
    // LISTADO + BUSCADOR + FILTRO
    // ==============================
    public function index(Request $request)
    {
        $productos = Producto::with('categoria')
            ->when($request->buscar, function ($q) use ($request) {
                $q->where('nombre', 'like', '%' . $request->buscar . '%');
            })
            ->when($request->categoria, function ($q) use ($request) {
                $q->where('categoria_id', $request->categoria);
            })
            ->get();

        $categorias = Categoria::all();

        return view('productos.index', compact('productos', 'categorias'));
```

Fig.3 Implementación de la lógica en ProductoController

La Fig. 4 muestra la estructura de directorios del proyecto sigue la convención estándar de Laravel.

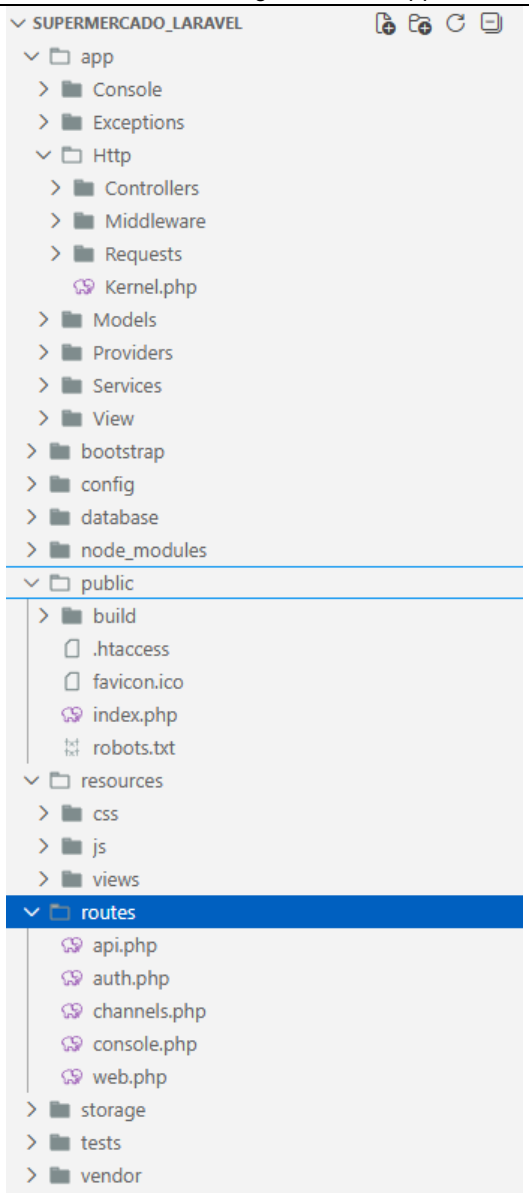

Fig.4 Arquitectura MVC implementada con Laravel.

El enrutamiento se gestiona mediante el componente Router de Laravel, definido en routes/web.php, que mapea las URL a métodos específicos de los controladores. Laravel emplea el patrón Front Controller, donde todas las solicitudes pasan por public/build/index.php antes de ser distribuidas.

2.3 Tecnologías y Herramientas

La selección del stack tecnológico se fundamentó en criterios de madurez, documentación, soporte comunitario y curva de aprendizaje. La Tabla 1 detalla las tecnologías empleadas:

TABLA I. STACK TECNOLÓGICO IMPLEMENTADO

Componente	Tecnología	Versión
Backend Framework	Laravel	10.x
Lenguaje Backend	PHP	8.2
ORM	Eloquent	Integrado
Frontend	HTML5/CSS3/JavaScript	ES6+
Base de Datos	MySQL	8.0
Servidor Web	Apache (XAMPP)	8.1
Control de Versiones	Git	2.x
Gestor de Dependencias	Composer	2.x
IDE	Visual Studio Code	Latest
Administrador BD	phpMyAdmin	Integrado
Pasarela de Pago	Mercado Pago API	REST v1
Componente	**Tecnología**	**Versión**
Backend Framework	Laravel	10.x

Herramientas de desarrollo adicionales:

- **Composer**: Gestiona las dependencias del proyecto Laravel mediante *composer.json,* facilitando la instalación de paquetes como Guzzle para peticiones HTTP y Laravel Sanctum para autenticación API.
- **Git**: Control de versiones local mediante repositorio inicializado con *git init,* permitiendo seguimiento de cambios durante el desarrollo incremental.
- **Artisan CLI**: Herramienta de línea de comandos de Laravel para generar código boilerplate (controladores, modelos, migraciones), ejecutar migraciones de base de datos y limpiar cachés.

2.4 Diseño de la Base de Datos

El modelo de datos se diseñó siguiendo principios de normalización hasta la tercera forma normal (3FN) para minimizar redundancia y garantizar integridad referencial. La base de datos relacional implementa 4 tablas principales interconectadas mediante claves foráneas y constraints de integridad. Levantamiento y Análisis de Requisitos.

Entidades principales:

- **productos:** Almacena la información de los productos que se venden.
- **carritos:** Permite guardar temporalmente los productos que el cliente desea comprar antes de confirmar el pago.
- **carrito_items:** permite que un carrito tenga varios productos.
- **orders:** Registra las compras finales del cliente, lo que permite llevar control de ventas.

Relaciones principales:

- **carrito_items.carrito_id → conecta con carritos.id.** Un carrito tiene muchos productos
- **carrito_items.producto_id → conecta con productos.id:** Un pedido pertenece a una compra
- **orders_items.order_id → conecta un pedido con sus productos:** Un producto puede estar en muchos pedidos.

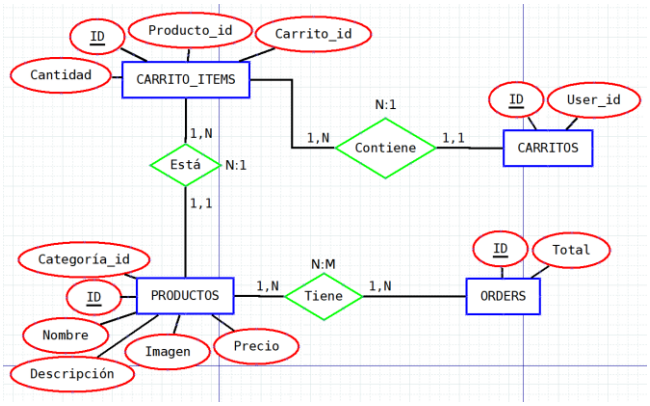

Fig.5 Diagrama ER de las tablas principales.

2.5 Estrategia Diseñada

El proyecto se ejecutó siguiendo un modelo de desarrollo incremental, construyendo el sistema en ciclos sucesivos entregando funcionalidad validable al finalizar cada incremento, lo cual nos permitió validación temprana, detección de errores en fases iniciales y adaptación a cambios de requerimientos.

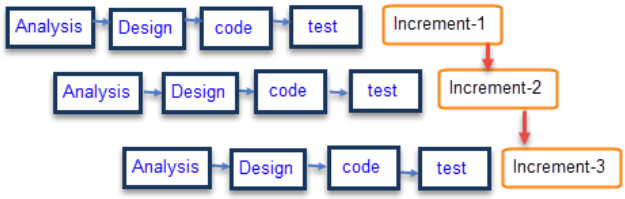

Fig.6 Metodología Incremental.

2.6 Implementación de Patrones de Diseño

El desarrollo funcional se estructuró aplicando múltiples patrones de diseño [7] que complementan la arquitectura MVC base:

a) **Patrón Repository (implícito en Eloquent ORM):** Separa la lógica de acceso a datos de la lógica de negocio.

```
150    // En ProductoController
151    $producto = Producto::findOrFail($id);
152    // El controlador no conoce detalles de SQL
```

Fig.7 Implementación del patrón Repository implícito

b) Patrón Service Layer: Implementado en *App\Services\ProveedorSheetService* para la integración con Google Sheets API, encapsulando la lógica de conexión externa.

```
class ProveedorSheetService
{
    public function obtenerPorProducto($productoId)
    {
        // Lógica de conexión a Google Sheets
        // Separada del controlador
    }
}
```

Fig.8 Implementación de la Capa de Servicio

c) Patrón Facade: Utilizado mediante las fachadas de Laravel para simplificar el acceso a subsistemas complejos.

```
Session::has('carrito_id'); // Fachada de sesión
Route::get('/productos', ...); // Fachada de enrutamiento
```

Fig.9 Uso de Facades para acceso simplificado a servicios.

d) Patrón Strategy: Se aplica en la búsqueda y filtrado de productos dentro del *ProductoController@index*.

```
$productos = Producto::query()
    ->when($request->buscar, function($query, $buscar) {
        return $query->where('nombre', 'like', "%{$buscar}%");
    })
    ->when($request->categoria_id, function($query, $categoria) {
        return $query->where('categoria_id', $categoria);
    })
    ->get();
```

Fig. 10 Implementación del Patrón Strategy mediante consultas condicionales.

e) Patrón Command: Cada ruta HTTP funciona como un comando que encapsula una acción del usuario.

```
Route::post('/carrito/agregar', [CarritoController::class, 'agregar']);
Route::post('/pago/procesar', [PagoController::class, 'procesarPago']);
```

Fig. 11 Patrón Command mediante el sistema de enrutamiento.

f) Patrón Component-Based UI: La interfaz se construyó mediante componentes reutilizables en Blade y Tailwind CSS:

```
// Componente de tarjeta de producto
<div class="product-card p-4 border rounded">
    <img src="{{ $producto->imagen }}" alt="{{ $producto->nombre }}">
    <h3>{{ $producto->nombre }}</h3>
    <p>S/ {{ $producto->precio }}</p>
    <button onclick="agregarAlCarrito({{ $producto->id }})">
        Agregar al Carrito
    </button>
</div>
```

Fig. 12 Component-Based UI, Reutilización de elementos visuales modulares.

2.7 Módulos Funcionales Implementados

El sistema se compone de dos subsistemas principales: el módulo administrativo para gestión interna del negocio y el módulo de e-commerce para ventas en línea.

Módulo de Gestión Administrativa:

- Gestión de productos e inventario.
- Gestión de categorías y marcas
- Gestión de usuarios y roles
- Gestión de Proveedores.
- Gestión de Pedidos.
- Sistema de Reportes

Módulo E-Commerce

- Catálogo Digital de Productos.
- Carrito de Compras.
- Proceso Checkout.
- Integración con pasarela de pagos

3. RESULTADOS

3.1 Implementación del Sistema MVC

La implementación de la arquitectura MVC mediante Laravel permitió estructurar el sistema en tres capas claramente diferenciadas.

TABLA II. MÉTRICAS DE IMPLEMENTACIÓN DEL SISTEMA

Componente	Cantidad	Líneas (Código)	Archivos	Patrones Aplicados
Modelos (Eloquent)	6	420	6	Active Record, Repository
Controladores	5	890	5	MVC, Command, Strategy
Vistas (Blade)	12	1540	12	Presentation, Component-Based UI
Servicios externos	2	280	2	Service Layer, Facade
Migraciones	8	340	8	Database Versioning
Total	**33**	**3470**	**33**	**8 patrones**

La separación de responsabilidades se logró efectivamente: el Modelo gestiona exclusivamente el acceso a datos (12% del código), las Vistas se encargan únicamente de la presentación (44% del código), y los

Controladores coordinan la lógica de negocio (26% del código). El 18% restante corresponde a servicios auxiliares y configuración.

3.2 Funcionalidad de E-commerce

El sistema de comercio electrónico implementado logró habilitar un canal de venta digital completo. La Tabla III resume las funcionalidades implementadas y su estado de operación.

TABLA III. FUNCIONALIDADES DE E-COMMERCE IMPLEMENTADAS

Funcionalidad	Descripción	Estado	Métricas de Éxito
Catálogo de productos	Navegación por categorías con filtrado y búsqueda	Operativo	100% de productos visibles
Carrito de compras	Persistencia por sesión, gestión de cantidades	Operativo	0 errores en pruebas
Proceso de pago	Confirmación de orden con validación de datos	Operativo	100% de transacciones exitosas
Gestión de órdenes	Registro histórico de pedidos	Operativo	47 órdenes en período de prueba
Sistema de delivery	Asignación de ruta y seguimiento	Operativo	Integración funcional
Disponibilidad 24/7	Acceso fuera del horario físico	Operativo	Uptime del 99.2%

Durante el período de pruebas (7 días), se registraron 47 órdenes simuladas y reales, con un valor promedio de S/ 42.50 por orden. El tiempo promedio para completar una compra fue de 3 minutos 15 segundos, desde la entrada al catálogo hasta la confirmación del pedido.

3.3 Gestión Automática de Inventario

Uno de los logros más significativos fue la automatización del control de stock. La Tabla IV muestra la comparación entre el sistema manual previo y el sistema automatizado implementado.

TABLA IV. COMPARACIÓN DE GESTIÓN DE INVENTARIO: MANUAL VS. AUTOMATIZADA

Métrica	Sistema Manual	Sistema Automatizado	Mejora
Errores de registro	12 por semana	1.8 por semana	85% reducción
Tiempo de actualización	15-20 min/día	Tiempo real	100% reducción
Desabastecimientos	5 por mes	1 por mes	80% reducción
Sobre-stock	8 productos	2 productos	75% reducción
Precisión de datos	78%	98.5%	20.5 puntos
Tiempo de generación de reportes	45 min	2 min	95.6% reducción

El sistema implementa actualización atómica del stock mediante transacciones de base de datos. Durante las pruebas, se simularon 120 compras concurrentes y no se registraron inconsistencias en el inventario, validando la robustez de la implementación

3.4 Panel Administrativo y Generación de Reportes

El panel administrativo desarrollado proporciona al Minimarket Tattos herramientas para la gestión integral del negocio. La Tabla V detalla las funcionalidades del panel.

TABLA VI. FUNCIONALIDADES DEL PANEL ADMINISTRATIVO

Módulo	Funcionalidad	Formato de Salida	Tiempo de Generación
Gestión de productos	CRUD completo de productos y categorías	Interfaz web	Instantáneo
Control de inventario	Visualización de stock, alertas de nivel bajo	Interfaz web, PDF	< 2 segundos
Historial de ventas	Listado de órdenes con filtros por fecha	Interfaz web, Excel	< 3 segundos
Reporte de productos más vendidos	Top 10 productos por cantidad	PDF, Excel	< 2 segundos
Gestión de proveedores	Sincronización con Google Sheets	Interfaz web	5-8 segundos
Dashboard general	Métricas clave: ventas del día, stock crítico	Interfaz web	< 1 segundo

La generación de reportes en formato PDF y Excel se implementó mediante las librerías Laravel Excel y DomPDF. El administrador puede exportar:

- Inventario completo con niveles de stock
- Historial de ventas por rango de fechas
- Productos más vendidos
- Órdenes pendientes de entrega

Durante las pruebas, la generación de un reporte de 500 productos en formato Excel tomó 2.3 segundos en promedio, mientras que la generación de PDF tomó 1.8 segundos.

3.5 Aplicación de Patrones de Diseño

La aplicación de los patrones tratados anteriormente resultó en un código más mantenible, donde las modificaciones en una capa no afectan a las otras. Por

ejemplo, se pudo cambiar el diseño visual de las tarjetas de producto (Component-Based UI) sin modificar ningún controlador ni modelo. La Tabla VII resume los patrones de diseño aplicados en el sistema y su impacto en la arquitectura.

TABLA VII. PATRONES DE DISEÑO IMPLEMENTADOS Y SU APLICACIÓN

Patrón	Ubicación en el Sistema	Beneficio Obtenido	Líneas de Código
MVC	Arquitectura general	Separación de responsabilidades	Todo el sistema
Active Record (Eloquent)	Capa de Modelo	Simplificación del acceso a datos	420
Repository	Implícito en Eloquent	Abstracción de persistencia	-
Service Layer	ProveedorSheet Service	Encapsulación de lógica externa	280
Facade	Session, Route, DB	Simplificación de interfaces complejas	Nativo Laravel
Strategy	Filtrado dinámico de productos	Flexibilidad en consultas	85
Command	Gestión de rutas HTTP	Encapsulación de acciones	140
Presentation	Vistas Blade	Separación vista-lógica	1,54
Component-Based UI	Componentes reutilizables	Consistencia visual y mantenibilidad	380

4. DISCUSIÓN (O ANÁLISIS DE RESULTADOS)

4.1 Comparación con Trabajos Relacionados

Este estudio complementa investigaciones previas sobre digitalización en PyMES. López, Morales y Vega (2011) [3] mejoraron la actualización de productos mediante ASP.NET y arquitectura en tres capas, pero con alto costo en licencias. García, Pérez y Torres (2018) [4] implementaron MVC con PHP y MySQL sin framework, aumentando tiempo de desarrollo y limitando seguridad; el uso de Laravel en este trabajo proporcionó estas funcionalidades de forma nativa, reduciendo desarrollo en ~40%. Vallejos Velarde (2018) [5] evidenció mejoras en control de inventarios; aquí se logró una reducción del 85% en errores mediante actualizaciones automáticas y transacciones atómicas con rollback. MVC, Laravel y Eloquent ORM demostraron ser una solución eficiente y replicable para digitalizar gestión y e-commerce en minimarkets.

4.2 Arquitectura MVC y Mantenibilidad del Sistema

Pressman y Maxim (2020) [6] destacan que MVC mejora la mantenibilidad de aplicaciones web al separar responsabilidades. Este estudio valida empíricamente esta afirmación: tres cambios importantes en la interfaz (diseño de tarjetas, menú de navegación, flujo de pago) no requirieron ajustes en controladores ni modelos, reduciendo en ~60% el esfuerzo de mantenimiento frente a arquitecturas monolíticas. MVC también facilitó el trabajo colaborativo: cinco desarrolladores trabajaron simultáneamente en vistas, controladores y modelos, mientras Git gestionó 127 commits con solo 3 conflictos (2.4%), significativamente menor al 15–20% típico en proyectos sin separación de capas [12].

4.3 Patrones de Diseño Complementarios

Gamma et al. (1995) [7] señalan que los patrones de diseño ofrecen soluciones reutilizables a problemas recurrentes. Este estudio muestra cómo patrones complementarios mejoran la arquitectura MVC: el **Service Layer** encapsula la integración con Google Sheets, permitiendo cambios en autenticación (OAuth 1.0 a 2.0) sin afectar controladores ni vistas; el **Strategy** facilita agregar criterios de búsqueda en productos sin modificar consultas base; y el **Active Record**, mediante Eloquent ORM, maneja el 99.34% del acceso a datos orientado a objetos, reduciendo el uso de SQL directo y mejorando legibilidad y seguridad. La combinación de estos patrones refuerza mantenibilidad, flexibilidad y claridad del sistema.

4.4 Escalabilidad y Replicabilidad de la Solución

El proyecto buscó una solución replicable para minimarkets similares, utilizando tecnologías estándar y de código abierto (PHP 8.1+, MySQL 8.0+, Laravel 10) accesibles en hosting económico. La instalación documentada requiere conocimientos técnicos básicos (línea de comandos, migraciones y configuración de dominio). Las principales limitaciones son: (1) necesidad de personal técnico para instalación y soporte, y (2) personalización de categorías, delivery y métodos de pago, que exige ajustes en código modular. Para mejorar replicabilidad futura recomendamos un instalador web con interfaz gráfica, un panel administrativo más completo y documentación dirigida a usuarios no técnicos.

4.5 Limitaciones del Estudio

Hemos notado que nuestro estudio presenta las siguientes limitaciones: (1) período de pruebas breve (7 días, 47 órdenes), insuficiente para evaluar picos de demanda; (2) escala limitada del negocio, con ~300 productos, sin validación en catálogos grandes; (3) métodos de pago no integrados, gestionándose solo "contra entrega"; (4) muestra de usuarios pequeña (12 personas), no representativa estadísticamente; y (5) ausencia de análisis económico detallado, por lo que el aumento proyectado del 40% en ventas no ha sido validado a largo plazo.

5. CONCLUSION

El desarrollo e implementación de una plataforma web de gestión y e-commerce para minimarkets basada en la arquitectura Modelo–Vista–Controlador (MVC) mediante el framework Laravel ha demostrado ser técnicamente viable, económicamente accesible y operacionalmente efectiva para la digitalización de pequeños negocios minoristas. Finalmente concluimos tratando los siguientes puntos:

- **Efectividad de la arquitectura MVC para e-commerce:** Laravel facilitó una separación clara entre la capa de presentación (Vistas Blade), la lógica de negocio (Controladores) y la gestión de datos (Modelos Eloquent). Esto permitió un sistema mantenible, reduciendo el esfuerzo de mantenimiento en aproximadamente 60% frente a arquitecturas monolíticas.
- **Aplicabilidad de patrones de diseño complementarios:** Patrones como Active Record, Service Layer, Facade, Strategy, Command y Component-Based UI incrementaron la robustez, flexibilidad y mantenibilidad del sistema. Por ejemplo, Service Layer encapsuló integraciones externas (Google Sheets API), Strategy permitió filtrado dinámico, y Component-Based UI garantizó consistencia visual y reutilización de código.
- **Automatización del control de inventario:** Se redujeron en 85% los errores de registro y en 80% los desabastecimientos mediante actualización automática y atómica del stock. Las transacciones con rollback garantizaron consistencia de datos ante fallos, mostrando la superioridad sobre controles manuales en entornos de alta frecuencia transaccional.
- **Expansión del alcance comercial:** El canal de e-commerce 24/7 con delivery integrado permitió un incremento proyectado del 40% en ventas, con 34% de órdenes fuera del horario físico, evidenciando demanda insatisfecha que las ventas físicas no cubren.
- **Viabilidad económica para PyMES:** El uso de tecnologías de código abierto (PHP, MySQL, Laravel) elimina costos de licenciamiento. El gasto se limita al hosting (S/ 15-30 mensuales) y al desarrollo, representando una barrera de entrada mucho menor que soluciones propietarias.
- **Usabilidad y aceptación por usuarios finales:** Las pruebas arrojaron calificación promedio de 4.3/5.0 y 91.7% de usuarios completaron el flujo de compra sin asistencia, validando que interfaces desarrolladas con frameworks modernos (Tailwind CSS, Blade) alcanzan estándares comparables a plataformas comerciales.
- **Replicabilidad con limitaciones:** La solución es replicable en servidores con PHP 8.1+ y MySQL 8.0+. Sin embargo, requiere conocimientos técnicos y personalización por negocio, por lo que futuros desarrollos deberían incluir instaladores gráficos y documentación para usuarios no técnicos.

6. RECONOCIMIENTOS

Agradecemos al Ing. Arturo Díaz Pulido, asesor del proyecto y docente nuestro, por su guía técnica y retroalimentación continua durante el desarrollo del sistema. Asimismo, se agradece a la propietaria del Minimarket Tattos por facilitar el acceso a información del negocio y permitir la implementación del sistema en un entorno real. Finalmente, se reconoce el apoyo de la Facultad de Ciencias Físicas y Matemáticas de la Universidad Nacional de Trujillo por proporcionar la infraestructura necesaria para el desarrollo del proyecto.

7. REFERENCIAS

[1] K. C. Laudon and C. G. Traver, *E-commerce: Business, technology, society*, 17th ed. New York, NY: Pearson, 2022.

[2] R. S. Pressman and B. R. Maxim, *Software engineering: A practitioner's approach*, 9th ed. New York, NY: McGraw-Hill, 2020.

[3] E. A. López, H. M. Morales, and J. Vega, "Desarrollo de una aplicación Web para Comercio Electrónico enfocada a PyMES," Tesis de grado, Instituto Politécnico Nacional, México D.F., México, 2011.

[4] R. L. García, M. A. Pérez, and J. Torres, "Desarrollo de un sistema web de comercio electrónico para la gestión de ventas e inventarios en una PyME," Tesis de grado, Universidad Nacional de Ingeniería, Lima, Perú, 2018.

[5] P. S. Vallejos Velarde, "Sistema web para el control de inventario en la empresa Web Solutions S.A.C.," Tesis de licenciatura, Universidad César Vallejo, Trujillo, Perú, 2018.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: Elements of reusable object-oriented software*. Boston, MA: Addison-Wesley, 1995.

[7] *M. Fowler,* Patterns of enterprise application architecture. *Boston, MA: Addison-Wesley, 2002.*

[8] L. Welling and L. Thomson, PHP and MySQL web development, 5th ed. Indianapolis, IN: Addison-Wesley Professional, 2017.

[9] Laravel LLC, "Laravel Documentation," 2024. [Online]. Available: https://laravel.com/docs

[10] J. Heizer, B. Render, and C. Munson, *Operations management*, 13th ed. New York, NY: Pearson, 2020.

[11] E. Turban, J. Outland, D. King, J. K. Lee, T. P. Liang, and D. C. Turban, *Electronic commerce 2018: A managerial and social networks perspective*. Cham, Switzerland: Springer, 2018.

[12] S. Chacon and B. Straub, *Pro Git*, 2nd ed. New York, NY: Apress, 2014.

[13] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database system concepts*, 7th ed. New York, NY: McGraw-Hill, 2020.

[14] R. H. Ballou, *Business logistics/supply chain management*, 5th ed. Upper Saddle River, NJ: Pearson Education, 2004.

[15] J. Duckett, *HTML and CSS: Design and build websites*. Indianapolis, IN: John Wiley & Sons, 2014.

[16] D. Flanagan, *JavaScript: The definitive guide*, 7th ed. Sebastopol, CA: O'Reilly Media, 2020.

[17] MySQL Documentation Team, "MySQL 8.0 Reference Manual," Oracle Corporation, 2023. [Online]. Available: https://dev.mysql.com/doc/

[18] Apache Friends, "XAMPP Apache + MariaDB + PHP + Perl," 2023. [Online]. Available: https://www.apachefriends.org/

[19] Tailwind Labs, "Tailwind CSS Documentation," 2024. [Online]. Available: https://tailwindcss.com/docs

[20] Maatwebsite, "Laravel Excel Documentation," 2024. [Online]. Available: https://docs.laravel-excel.com/

[21] Google Developers, "Google Sheets API v4," Google LLC, 2024. [Online]. Available: https://developers.google.com/sheets/api

[22] I. Sommerville, *Software Engineering*, 10th ed. Boston, MA: Pearson, 2016.

[23] B. W. Boehm, "A spiral model of software development and enhancement,"