Desarrollo de una Plataforma Web para Gestión Inteligente de Depósitos: Un Trabajo en Progreso

Joaquín Sedoff
Facultad de Ingeniería - UNaM
Oberá, Argentina
joasedoff@gmail.com

***Abstract*—This paper presents the ongoing development of a web platform for intelligent warehouse management with dynamic pricing capabilities. The system, currently in prototype phase, integrates Flask, MariaDB, and Telegram Bot, deployed through Docker on a Debian VPS, which is programmed via SSH. The platform automatically calculates the required storage surface and estimates costs based on requested products by the user. Although the current version implements a simplified pricing mechanism, the proposed model is designed to be extensible, allowing future inclusion of additional variables such as occupancy rates and seasonal demand. This paper describes current progress, encountered technical challenges, and the roadmap for completing development.**

***Keywords*—Web Application; Flask; Docker; Telegram Bot; Warehouse Management; Prototype**

***Resumen*—Este trabajo presenta el desarrollo en curso de una plataforma web para la gestión inteligente de depósitos con capacidades básicas de fijación dinámica de precios. El sistema, actualmente en fase de prototipo, integra Flask, MariaDB y un Bot de Telegram, desplegado mediante Docker en un VPS Debian y programado a través de SSH. La plataforma calcula automáticamente la superficie de almacenamiento requerida y estima los costos según los productos solicitados por el usuario. Aunque la versión actual implementa un mecanismo de cálculo de precios simplificado, el modelo propuesto está diseñado para ser extensible, permitiendo en el futuro la incorporación de variables adicionales como tasas de ocupación y demanda estacional. Este artículo describe el progreso actual, los desafíos técnicos encontrados y la hoja de ruta para completar el desarrollo.**

***Palabras-clave*—Aplicación Web; Flask; Docker; Telegram Bot; Gestión de Depósitos; Prototipo**

I. Introducción

El auge del comercio electrónico ha generado una demanda crítica por sistemas automatizados para la gestión de depósitos. Este proyecto aborda dicha problemática mediante el desarrollo de una plataforma web que, partiendo de una necesidad real del mercado local, ofrece una solución accesible para pequeñas y medianas empresas (PyMEs). El sistema combina el cálculo de la superficie requerida por un usuario, la estimación de costos en tiempo real y notificaciones push a través de un Bot de Telegram.

El objetivo principal es ofrecer una solución integral basada en una arquitectura cliente-servidor con Flask, MariaDB y Telegram. Actualmente en fase de prototipo avanzado, este artículo presenta los avances del desarrollo, los desafíos técnicos encontrados y la hoja de ruta para su implementación final.

II. Estado del Arte y Motivación

La optimización de almacenes, capaz de reducir costos operacionales hasta en un 30% [1], depende de Sistemas de Gestión de Almacenes (WMS). El mercado está dominado por soluciones comerciales de alto costo como SAP WM, inaccesibles para PyMEs [3]. Como alternativa, existen soluciones de código abierto como OpenBoxes [10] y Odoo [9], que si bien son potentes, presentan una considerable curva de aprendizaje y complejidad en su configuración.

En un punto intermedio se encuentran frameworks ligeros como Flask, utilizados en proyectos de gestión de inventario más simples [4], [5]. Nuestra propuesta se diferencia al ofrecer una solución ultraligera, contenerizada con Docker para un despliegue sencillo en infraestructuras de bajo costo, y con un sistema de notificaciones en tiempo real vía Telegram, enfocado en las necesidades de comunicación directa de los pequeños negocios.

III. Arquitectura del Sistema

El sistema se ha diseñado siguiendo una arquitectura cliente-servidor, implementada sobre un entorno de contenedores Docker que garantiza la modularidad y el despliegue controlado de los servicios. La estructura general, ilustrada en la Figura 1, se compone de cuatro capas principales que operan de manera coordinada para ofrecer la funcionalidad completa de la plataforma.

A. Diseño General

La arquitectura sigue un patrón distribuido que separa la lógica de presentación (frontend), de negocio (backend) y de persistencia de datos. El flujo inicia cuando el usuario interactúa con el frontend de Flask; el backend procesa la

lógica de negocio, almacena los datos de registro en MariaDB y finalmente envía una notificación al administrador a través de un Bot de Telegram. Todo el entorno opera sobre un Servidor Privado Virtual (VPS) con Debian para proveer un entorno de producción estable.

B. Capa de Presentación: El Frontend

La interfaz de usuario es una aplicación web responsiva, desarrollada con **HTML5**, **CSS3** y **JavaScript**. Para asegurar la compatibilidad con dispositivos móviles y de escritorio, se utilizó el framework **Bootstrap 5**. Esta capa tiene dos componentes principales:

- **La Calculadora de Depósito:** Una página pública donde los usuarios pueden seleccionar objetos predefinidos o ingresar las dimensiones de objetos personalizados, cuenta con la posibilidad de elegir entre modo oscuro y claro. El cálculo de la superficie y la estimación del precio se actualizan dinámicamente en el lado del cliente mediante JavaScript para ofrecer una experiencia interactiva.
- **El Panel de Administración:** Una sección de acceso restringido que requiere autenticación. Permite a los administradores visualizar la tabla completa de registros de clientes almacenados en la base de datos.

C. Capa de Aplicación: El Backend

El núcleo del sistema es un backend desarrollado en **Python** utilizando el microframework **Flask**. Este componente es responsable de:

- **Manejar las Peticiones HTTP:** Procesa las solicitudes POST provenientes de los formularios de registro de la calculadora y del login de administración.
- **Lógica de Negocio:** Ejecuta la lógica para calcular el costo final del almacenamiento, validar los datos de entrada del usuario y gestionar las credenciales de administrador, las cuales se almacenan de forma segura mediante *hashing*.
- **Coordinación de Servicios:** Actúa como orquestador, interactuando con la base de datos para persistir y recuperar información, y con el servicio de notificaciones para enviar alertas.

D. Capa de Datos

Para la persistencia de la información, se utiliza el sistema de gestión de bases de datos relacional **MariaDB**, una bifurcación de MySQL compatible y de alto rendimiento. El esquema de la base de datos está diseñado para almacenar:

- **Registros de Clientes:** Incluye el nombre, email, teléfono y los detalles de los objetos que el cliente desea almacenar.
- **Credenciales de Administrador:** Almacena los nombres de usuario y las contraseñas *hasheadas* para el acceso al panel de administración.

La gestión y el mantenimiento de la base de datos se facilitan mediante la herramienta web **phpMyAdmin**.

E. Capa de Notificaciones

El sistema de notificaciones en tiempo real se implementa a través de un **Bot de Telegram**. Utilizando la biblioteca **python-telegram-bot**, el backend de Flask envía una solicitud a la API de Telegram de forma asíncrona cada vez que un nuevo cliente completa un registro. El mensaje, que contiene un resumen del pedido (cliente, objetos, superficie y costo estimado), se envía a un ID de chat preconfigurado, garantizando que la notificación llegue exclusivamente al administrador del negocio.

IV. Implementación Actual

A. Módulos Completados

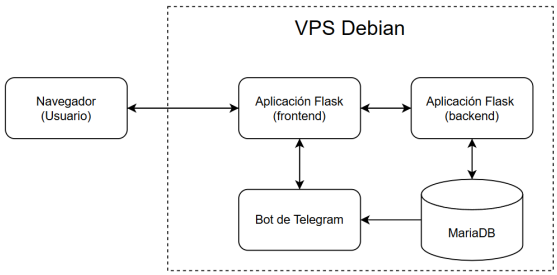

Fig. 1. Diagrama de flujo actual de la plataforma web (en desarrollo).

El flujo básico del sistema está implementado: los usuarios seleccionan objetos del catálogo predefinido o ingresan manualmente las especificaciones de objetos personalizados, el sistema procesa la información de superficie y genera estimaciones de costo, los datos se persisten en MariaDB y se envían notificaciones automáticas al administrador.

B. Sistema de Registro de Objetos

La plataforma ofrece dos modalidades para el registro de objetos:

- **Objetos Predefinidos**: Catálogo de objetos estándar organizados por categorías del hogar (sillones, mesas, escritorios, camas), cada uno con superficie y características preestablecidas.
- **Objetos Personalizados**: Formulario para ingreso manual donde el usuario especifica la superficie y una descripción del objeto a almacenar.

El sistema procesa esta información para estimar el espacio requerido y calcular el costo de almacenamiento correspondiente, integrando los datos con la base de datos MariaDB para su posterior gestión.

C. Interfaz de Usuario

Fig. 2. Prototipo actual de la interfaz principal.

La interfaz web principal, cuyo prototipo se muestra en la Figura 2, consiste en una calculadora interactiva con formularios responsivos para el registro de objetos. Adicionalmente, el sistema cuenta con una sección de administración con acceso restringido por usuario y contraseña, donde se listan todos los registros realizados. Para asegurar la integridad de los datos, se implementó validación tanto en el lado del cliente (*client-side*) como en el servidor (*server-side*).

Una vez que el usuario completa el registro y proporciona sus datos de contacto, el sistema envía una notificación al administrador. Un ejemplo de este mensaje, recibido a través de un bot de Telegram, se puede observar en la Figura 3. El mensaje está destinado solamente a los administradores del negocio, ya que se vincula con sus ID de chat de Telegram.

Fig. 3. Notificación de Telegram enviada al administrador.

Además de la sección de calculadora, la cual está orientada a los usuarios de la página, se cuenta con una sección destinada a los administradores del negocio, en donde se muestran todos los registros realizados, junto con su información más relevante. Para acceder a esta página, se requieren credenciales de usuario, las cuales se almacenan en la base de datos de forma *hasheada* para así poder asegurar que solo los administradores tengan acceso a la sección.

V. Pruebas y Validación Preliminar

Para asegurar el correcto funcionamiento del prototipo, se llevó a cabo un proceso de validación en varias etapas, enfocado tanto en la funcionalidad de los componentes individuales como en la integración del sistema completo.

- **Validación del Frontend:** Se comprobó el comportamiento de la ”Calculadora de Depósito”. Las pruebas incluyeron la selección de múltiples objetos predefinidos y la introducción de dimensiones para objetos personalizados, verificando que los cálculos de superficie y precio estimado se actualizaran correctamente en la interfaz en tiempo real. Se validaron también los formularios de contacto para asegurar que los datos del usuario se capturaran de forma adecuada antes de ser enviados al servidor.
- **Pruebas del Backend y API:** Se testeó el endpoint principal que recibe los datos del formulario. Se enviaron peticiones de prueba (simulando un registro de usuario) para confirmar que el backend procesaba la información correctamente, aplicaba la lógica de negocio para el cálculo final y devolvía una respuesta adecuada. Se incluyeron pruebas de validación de datos en el lado del servidor para rechazar entradas incompletas o malformadas.
- **Integración con la Base de Datos:** Se verificó el ciclo completo de persistencia de datos. Tras una prueba de registro exitosa, se consultó directamente la base de datos MariaDB para confirmar que un nuevo registro se había creado en la tabla correspondiente con la información correcta (nombre, email, objetos, superficie, etc.). Del mismo modo, se probó la funcionalidad de login del administrador para asegurar la correcta recuperación y validación de credenciales *hasheadas*.
- **Verificación del Sistema de Notificaciones:** Se realizaron múltiples registros de prueba para validar la integración con el bot de Telegram. Se confirmó que, por cada registro exitoso, se enviaba un único mensaje al ID de chat del administrador. Se inspeccionó el contenido de estos mensajes para asegurar que el formato y la información (datos del cliente, resumen del pedido) fueran precisos.

A. Compatibilidad y Usabilidad

Finalmente, se validó la accesibilidad de la plataforma, confirmando su correcto funcionamiento en los principales navegadores web (Google Chrome, Mozilla Firefox, Safari y

Microsoft Edge). Gracias al uso del framework Bootstrap 5, se aseguró además una experiencia de usuario consistente y responsiva, adaptándose la interfaz de manera fluida a dispositivos móviles, tabletas y monitores de escritorio.

VI. Algoritmo de Fijación Dinámica de Precios

A. Problemática de la Fijación de Precios en Depósitos

La determinación de precios en servicios de almacenamiento tradicionalmente se basa en tarifas fijas por metro cúbico, lo que no refleja adecuadamente las variaciones en costos operacionales ni la demanda del mercado [1]. Para PyMEs del sector, implementar un sistema de precios más flexible puede significar una ventaja competitiva importante.

B. Propuesta de Implementación

Se propone implementar un algoritmo básico de cálculo de precios que considere factores simples pero efectivos:

$$P_{final} = P_{base} \cdot F_{superficie} \cdot F_{ocupacion} \tag{1}$$

donde P_{base} es el precio base por metro cúbico, $F_{superficie}$ aplica descuentos por superficie para incentivar clientes grandes, y $F_{ocupacion}$ ajusta precios según la disponibilidad actual del depósito.

Esta aproximación permite optimizar ingresos sin complejidad excesiva, manteniendo la simplicidad operativa necesaria para un negocio en crecimiento.

VII. Trabajo futuro: Roadmap de Desarrollo

Los próximos pasos del proyecto se centrarán en la maduración del prototipo actual hacia una versión beta, con un enfoque en las siguientes áreas clave:

- **Autenticación Multiusuario:** Expandir el sistema de autenticación básico para soportar diferentes roles y permisos (ej. administrador, gestor de depósitos), mejorando la seguridad y la flexibilidad operativa.
- **Implementación del Algoritmo de Precios Dinámico:** Desarrollar la lógica del backend para implementar el algoritmo de fijación de precios propuesto, permitiendo ajustes automáticos basados en la demanda y la ocupación.
- **Módulo de Reportes:** Crear un panel de control para administradores que incluya métricas de negocio, visualizaciones de datos y la capacidad de exportar informes en formato PDF.
- **Integración Externa:** Desarrollar una API REST para permitir la comunicación con sistemas de facturación externos y pasarelas de pago.
- **Optimización y Pruebas de Usabilidad:** Realizar pruebas de rendimiento para optimizar la escalabilidad y ejecutar un ciclo de pruebas con usuarios finales para recolectar feedback y mejorar la experiencia de la interfaz.
- **Documentación Completa:** Generar la documentación técnica y de usuario necesaria para facilitar el mantenimiento y la adopción de la plataforma.

VIII. Conclusiones

El prototipo desarrollado demuestra la viabilidad de crear una solución de gestión de depósitos accesible para PyMEs, utilizando tecnologías web estándar y de código abierto. La arquitectura propuesta, basada en Flask y Docker, no solo facilita el despliegue y futuro mantenimiento del sistema, sino que también asegura una solución de bajo costo, cuyo principal componente de inversión reside en el servidor de alojamiento (VPS). La integración con un bot de Telegram se confirmó como un canal de comunicación simple y eficiente para el envío de notificaciones en tiempo real a los administradores.

Los resultados preliminares validan que es posible ofrecer una alternativa funcional a soluciones comerciales de alto costo, con un enfoque centrado en las necesidades del mercado local. El trabajo futuro se enfocará en evolucionar el prototipo hacia una versión beta, implementando las funcionalidades de pricing dinámico y reportes avanzados, lo que permitirá evaluar su impacto directo en la eficiencia operativa del negocio objetivo.

Referencias

[1] G. Richards, *Warehouse Management: A Complete Guide to Improving Efficiency and Minimizing Costs in the Modern Warehouse*, 2nd ed. Kogan Page, 2015.
[2] R. de Koster, T. Le-Duc, and K. J. Roodbergen, "Design and control of warehouse order picking: A literature review," *European Journal of Operational Research*, vol. 182, no. 2, pp. 481-501, 2007.
[3] A. Gola and A. Kłos, "Open Source Software in Warehouse Management Systems," in *Information Systems Architecture and Technology: Proceedings of 40th Anniversary International Conference on Information Systems Architecture and Technology – ISAT 2020*, Springer, 2021, pp. 119-129.
[4] P. Swiecki, "Basic Warehouse Management System built with Python on Flask," GitHub repository, 2022. [Online]. Available: https://github.com/pawelswiecki/basic-wms
[5] Marination, "Inventory Manager: An inventory management system using Flask," GitHub repository, 2021. [Online]. Available: https://github.com/marination/Inventory-Manager
[6] Pallets Projects, "Flask Documentation," 2025. [Online]. Available: https://flask.palletsprojects.com/
[7] python-telegram-bot developers, "python-telegram-bot Documentation," 2025. [Online]. Available: https://python-telegram-bot.org/
[8] Bootstrap Core Team, "Bootstrap Documentation," 2025. [Online]. Available: https://getbootstrap.com/docs/
[9] Odoo S.A., "Odoo Inventory," 2025. [Online]. Available: https://www.odoo.com/app/inventory
[10] OpenBoxes, "OpenBoxes: Supply Chain Management Software," 2025. [Online]. Available: https://openboxes.com/