RECIE
Revista Electrónica Científica de Investigación Educativa

Volumen 9, enero-diciembre 2025
ISSN: 2594-200X

Integración de microservicios con Laravel, Python y Django para la enseñanza del procesamiento de datos

Integration of microservices with Laravel, Python and Django for teaching data processing

José de Jesús Solorzano Rangel • Mariana Carolyn Cruz Mendoza • Cesar Primero Huerta

José de Jesús Solorzano Rangel. Tecnológico Nacional de México-Tecnológico de Estudios Superiores de Valle de Bravo, Estado de México. Es colaborador del proyecto "Methodology for Experiments with Machine Learning Algorithms" y actualmente estudia Ingeniería en Sistemas Computacionales. Correo electrónico: jesus.solorzano.rangel@gmail.com. ORCID: https://orcid.org/0009-0007-8398-6870.

Mariana Carolyn Cruz Mendoza. Tecnológico Nacional de México- Tecnológico de Estudios Superiores de Valle de Bravo, Estado de México. Es Maestra en Ingeniería en Sistemas Computacionales por el TecNM-Tecnológico de Estudios Superiores de Valle de Bravo. Cuenta con el reconocimiento al perfil deseable Prodep. Es cofundadora del Cuerpo Académico "Procesamiento de datos con industria 4.0". Sus intereses de investigación son el aprendizaje automático y la ciencia de datos. Miembro de la Red Nacional de Investigación Aplicada de las Tecnologías de la Industria 4.0, reconocida por el TecNM. Correo electrónico: marianacarolyn@hotmail.com. ORCID: https://orcid.org/0000-0003-3500-2500.

Cesar Primero Huerta. Tecnológico Nacional de México- Tecnológico de Estudios Superiores de Valle de Bravo, Estado de México. Es Maestro en Ingeniería en Sistemas Computacionales por el TecNM-Tecnológico de Estudios Superiores de Ecatepec e Ingeniero en Sistemas Computacionales por el TecNM-Tecnológico de Estudios Superiores de Valle de Bravo. Cuenta con el

Resumen

Este artículo muestra cómo integrar diversas tecnologías en un entorno educativo para facilitar la enseñanza del procesamiento de datos mediante el uso de microservicios. El proyecto emplea Laravel, Alpine.js y Django para la gestión de datos y su transformación mediante microservicios. La implementación de una arquitectura basada en microservicios permite dividir la aplicación en componentes manejables, lo que no solo mejora la escalabilidad y el mantenimiento del sistema sino que también ofrece un modelo práctico para que los estudiantes comprendan estas tecnologías. Se desarrolló un prototipo siguiendo la metodología de prototipo, lo que permitió un diseño iterativo del sistema con un enfoque en la enseñanza de conceptos clave, como la carga de datos, homogenización de categorías, eliminación de duplicados y transformación de variables categóricas. Las evaluaciones de rendimiento indican tiempos de respuesta satisfactorios al procesar datos numéricos con conjuntos de datos pequeños, mientras que se identificaron áreas de mejora al manejar grandes volúmenes de datos categóricos. Este enfoque educativo busca reducir la brecha de habilidades técnicas en estudiantes interesados en el aprendizaje automático y el procesamiento de datos.

Palabras clave: acceso a la información, algoritmo, procesamiento de la información.

Abstract

This article demonstrates how to integrate various technologies into an educational environment to facilitate the teaching of data processing using microservices. The project employs Laravel, Alpine.js and Django for data management and transformation via microservices. Implementing a microservice-based architecture allows the application to be divided into manageable components, which not only improves the system's scalability and maintainability but also provides a practical model for students to understand these technologies. A prototype was developed following the prototyping

Recibido: 6 de diciembre de 2024 | Aprobado: 3 de junio de 2025 | Publicado: 13 de junio de 2025
DOI: https://doi.org/10.33010/recie.v9i0.2460

reconocimiento al perfil deseable Prodep. Es cofundador y miembro del Cuerpo Académico "Procesamiento de datos con industria 4.0" y miembro activo del colectivo Ambiente Cielo Rojo. Sus intereses de investigación son el aprendizaje automático y la ciencia de datos. Correo electrónico: iscprimerocesar@gmail.com. ORCID: https://orcid.org/0000-0002-8083-6988.

methodology, enabling an iterative system design with a focus on teaching key concepts such as data loading, category homogenization, duplicate elimination and categorical variable transformation. Performance evaluations indicate satisfactory response times when processing numerical data with small datasets, while areas for improvement were identified when handling large volumes of categorical data. This educational approach aims to reduce the technical skills gap among students interested in machine learning and data processing.

Keywords: information access, algorithm, information processing.

Introducción

Las interfaces de programación de aplicaciones –API, por sus siglas en inglés, *Application Programming Interfaces*– son ampliamente utilizadas por los servicios web y, desde el año 2000, se han convertido en un estándar para el desarrollo y uso de microservicios (Felício et al., 2023). Una característica principal de las API es su versatilidad, ya que se implementan en diversos lenguajes de programación, debido a esto es crucial contar con conocimientos técnicos avanzados para la definición y operaciones de las mismas, lo cual tiene como consecuencia una posible generación de errores durante su ejecución (De Luca et al., 2011).

Durante el desarrollo de una aplicación web es común que exista la falta de cohesión entre las tecnologías con las que se desarrolla, debido a que no existen modelos tradicionales para la integración de diferentes sistemas interconectados. La incapacidad de mantener una comunicación fluida entre los microservicios y las tecnologías descentralizadas ha resultado en ineficiencias durante la ejecución de los microservicios, como lo son pérdida de información, retrasos en procesamiento de datos o menor escalabilidad (Pujadas et al., 2024).

Las API, y especialmente las arquitecturas basadas en microservicios, han permitido a los desarrolladores dividir aplicaciones monolíticas en componentes más pequeños y manejables. Esta estrategia facilita el despliegue independiente, mejora la escalabilidad y simplifica el mantenimiento del sistema.

Sin embargo, también plantea el reto de lograr una cohesión efectiva entre los microservicios, ya que cada componente puede estar desarrollado en un lenguaje distinto o emplear *frameworks* variados. En aplicaciones web que integran Laravel, Alpine.js y Django con Python, es fundamental garantizar una comunicación fluida entre los microservicios para mantener la consistencia de los datos, pese a sus diferencias tecnológicas (Harris, 2020).

Diversos estudios han demostrado que la demanda de graduados en ingeniería con habilidades técnicas en ciencia de datos, aprendizaje automático e inteligencia artificial ha crecido considerablemente en los años recientes. En este contexto, se propuso un curso de aprendizaje automático –ML, por sus siglas en inglés, *Machine Learning*– para estudiantes de posgrado, diseñado específicamente para que aplicaran

técnicas de ML en la investigación. Los resultados mostraron una respuesta positiva por parte de los estudiantes, destacando la efectividad de enfoques educativos orientados a la integración de habilidades técnicas (Pilario et al., 2024).

Este artículo se centra en el desarrollo de un proyecto que integra diferentes tecnologías las cuales interactúan a lo largo del proceso de envío de información desde el *frontend* hasta el *backend* para garantizar una comunicación eficiente, con la finalidad de desarrollar una herramienta que disminuya la brecha de habilidades necesarias en estudiantes para utilizar herramientas de ML.

En el *frontend* se decidió implementar un *framework* ligero para la manipulación de la interfaz de usuario sin necesidad de recargar la página; de entre las opciones que se consideraron, se optó por utilizar Alpine.js debido a que Laravel facilita su integración en el proyecto.

Para gestionar la capa intermedia del *backend* se implementó Laravel, ya que este tiene la capacidad de trabajar con lenguaje PHP para recibir solicitudes del *frontend* y enviarlas a un servicio externo en el *backend*, en este caso un servidor Django en Python. Este último se encarga de transformar los datos enviados al aplicar algoritmos. La correcta integración de estas tecnologías permite un intercambio eficiente de datos a través de microservicios.

En este artículo se presenta la implementación de la metodología de prototipo, lo que permitió un desarrollo iterativo y enfocado en la integración de tecnologías como Laravel, Alpine.js y Python con Django. Acto seguido se puso en práctica esta metodología, la cual abarca, en primer lugar, una descripción del diseño y la construcción de la arquitectura basada en microservicios, seguida del desarrollo del prototipo y los desafíos enfrentados durante su implementación.

Posteriormente se presentan las soluciones aplicadas a los problemas identificados y se detallan las evaluaciones de rendimiento realizadas al sistema para garantizar su eficacia y adaptabilidad en un entorno educativo. Finalmente se agrega la discusión de los resultados obtenidos para tener como resultado las conclusiones de la investigación.

Antecedentes

Las investigaciones sobre tecnologías basadas en API proporcionan una base esencial para comprender su evolución y su aplicación efectiva en proyectos actuales. Estos estudios permiten analizar cómo han sido adoptadas en distintos contextos tecnológicos.

Explorar casos previos y soluciones implementadas ayuda a identificar patrones, desafíos y logros relevantes. En especial, el análisis de experiencias en integración de API, procesamiento de datos y mejora funcional aporta lecciones valiosas que orientan el diseño y desarrollo de nuevas aplicaciones.

Se desarrolló una API utilizando tecnologías de Google, como Google Cloud y Google API Gateway, que transformó, cargó y procesó datos en BigQuery mediante Apache Beam. Esta API permitió ejecutar consultas de SQL sobre un almacén de

datos, lo que proporcionó a los usuarios un control más detallado sobre las consultas hechas, ejemplificando con este sistema cómo se puede facilitar el uso y acceso a los pronósticos NWM para aplicaciones críticas para el mejoramiento en áreas como la gestión del agua, gestión de inundaciones, recreación y agricultura (Gopalakrishnan y Ramaswamy, 2017).

En el año 2022 se desarrolló una API web para generar condiciones meteorológicas y simular el desarrollo de plagas en América del Norte. Debido a que los generadores meteorológicos no se integraban fácilmente en marcos de modelado, se optó por implementarlos como una API web. Esto facilitó su integración en diversos entornos de simulación y mejoró su aplicabilidad en múltiples contextos. (Fortin et al., 2022).

Se ha desarrollado una aplicación móvil y web que utiliza técnicas de *deep learning* para facilitar la detección de diversas enfermedades mediante imágenes médicas. Este proyecto surgió como respuesta a los desafíos relacionados con la detección y clasificación manual de enfermedades, procesos tradicionalmente lentos y complejos. La aplicación busca asistir a los especialistas en la toma de decisiones terapéuticas más adecuadas, ofreciendo un seguimiento continuo de la evolución de las enfermedades. Actualmente, DLDiagnosis identifica ocho enfermedades, entre ellas el queratocono, cáncer de mama y neumonía (Mustapha et al., 2023).

Una estrategia eficaz para superar las limitaciones históricas en el análisis de datos de uso y cobertura del suelo, especialmente las asociadas a costos elevados, es el uso de la API de Python de Google Earth Engine –GEE– junto con imágenes aéreas NAIP gratuitas del 2017. Este enfoque fue aplicado para clasificar el uso del suelo en el Panhandle de Florida. Se identificaron ocho clases con una precisión del 86% y un valor Kappa del 79%. Los resultados demostraron que esta metodología reduce significativamente el tiempo y los costos, beneficiando proyectos globales de teledetección, en particular en regiones con acceso limitado a *software* comercial (Prasai et al., 2021).

La gestión heterogénea de redes representa un desafío significativo para muchas organizaciones, afectando el funcionamiento y la calidad del servicio, y provocando pérdidas económicas. Una solución efectiva consiste en integrar la información de gestión proporcionada por diferentes sistemas en una única aplicación mediante el uso de API. Este enfoque propone un procedimiento para implementar una gestión integrada utilizando API, cuya efectividad fue demostrada en una entidad al lograr una integración eficiente de los sistemas de gestión (Markert et al., 2024).

Una revisión sistemática de la literatura –RSL– examinó el impacto del ML en el despliegue de API y funcionalidades, destacando su capacidad para optimizar procesos mediante técnicas, algoritmos y predicciones de errores. El estudio identificó 176,378 artículos en bases de datos digitales, de los cuales se seleccionaron 85 relevantes tras aplicar criterios de exclusión y calidad. Los resultados indican que el ML ofrece soluciones para mejorar la eficiencia y detectar errores en procesos de despliegue de API y *software* que fallan regularmente (Rojas et al., 2023).

Las investigaciones recientes destacan el papel crucial de las API en la mejora y optimización de diversos sistemas y procesos. Los estudios revisados muestran cómo las API no solo facilitan la integración y el acceso a datos críticos, sino que también abordan desafíos específicos en diferentes áreas.

Estos avances subrayan la importancia de las API como herramientas versátiles y esenciales para el desarrollo de soluciones tecnológicas efectivas, resaltando su impacto positivo en la optimización de procesos y la resolución de problemas complejos en distintos dominios.

Metodología

Para el desarrollo de la investigación se implementó una arquitectura basada en microservicios con énfasis en la calidad del *software*. El proceso de desarrollo se basó en la metodología *Prototype*, esta se compone de seis etapas: recopilación y análisis de requisitos, diseño rápido, desarrollo del prototipo, evaluación del prototipo, refinamiento, e implementación y mantenimiento.

Sin embargo, para el caso de este artículo de investigación se omitirán por el momento las últimas dos etapas de refinamiento e implementación, estas se desarrollarán en un futuro para optimizar la calidad del *software* desarrollado (ver Figura 1).

Figura 1
Metodología utilizada

Nota: Se usan las primeras cuatro fases de la metodología original.
Fuente: Elaboración propia.

En el diseño de esta herramienta se priorizó la creación de módulos que fueran fácilmente comprensibles por estudiantes con diferentes niveles de experiencia técnica. Por ejemplo, cada etapa del *pipeline* de transformación de datos incluye explicaciones y sugerencias sobre su uso, lo que facilita el aprendizaje incremental de conceptos complejos como la homogeneización de categorías y la codificación de variables categóricas.

El sistema de microservicios permite la autenticación y autorización de usuarios mediante un inicio de sesión basado en *token*, esto garantiza el acceso controlado según roles. Facilitará la transformación de datos estructurados a través de herramientas seleccionables por el usuario, ejecutándose en un servidor independiente para optimizar el procesamiento. Además proporcionará una previsualización de los resultados transformados antes de su descarga y garantizará alto rendimiento, con tiempos de respuesta óptimos para las solicitudes y una carga completa de la interfaz interactiva con el flujo de información.

El enfoque de diseño consideró las necesidades específicas de estudiantes universitarios, con interfaces intuitivas y módulos preconfigurados que facilitan el aprendizaje incremental de las técnicas de transformación de datos. Estas características aseguran que la herramienta sea inclusiva y accesible para usuarios con diversos niveles de experiencia.

Una vez validados los requisitos, se procedió a realizar un diseño rápido del prototipo. Esta fase se centró en esbozar las principales características y funcionalidades del sistema mediante herramientas y técnicas específicas del desarrollo de *software*, como la maquetación de la interfaz del sistema y la elaboración del modelo entidad-relación para la base de datos, donde se utilizó como apoyo *software* especializado en diseño.

El maquetado del sistema se realizó con especial énfasis en que este sea intuitivo en su uso, no solo para usuarios familiarizados con transformación de datos sino para cualquiera que busque hacer uso de estas herramientas, es por ello que el diseño tuvo en todo momento un enfoque minimalista y sencillo, de esta manera se proporcionará el acceso a las herramientas del sistema en pocos movimientos por parte del usuario.

Herramientas

Para la construcción del prototipo del sistema basado en microservicios se adoptó un enfoque de desarrollo en paralelo para cada tecnología involucrada. Para desarrollar el sistema se utilizó la siguiente lista de herramientas:

- Laravel: *framework* de PHP orientado al desarrollo de aplicaciones web con énfasis en la simplicidad y eficiencia. Se destaca para el desarrollo rápido de aplicaciones a gran escala y es mejor que otros *frameworks* en cuanto a manejo de peticiones por segundo, uso de memoria, tiempo de respuesta y número de archivos (Laaziri et al., 2019).
- Alpine.js: *framework* ligero de JavaScript utilizado para gestionar la interactividad en el *frontend* de la aplicación. Se usó gracias a ser un gran sustituto de marcos de trabajo antiguos como jQuery (Ekholm, 2024).
- Django: *framework* de desarrollo web basado en Python, enfocado en la creación de aplicaciones complejas y seguras; fue utilizado para la parte *backend* del sistema debido a sus beneficios de flexibilidad y seguridad adicionales que brindan sus canales (Chapkovski y Kujansuu, 2019).

La combinación de estas tecnologías permitió la construcción de una aplicación web *full stack* capaz de gestionar datos de manera eficiente y proporcionar herramientas accesibles para la transformación de datos.

Se evaluará el *software* mediante el algoritmo *pipeline* de procesamiento de datos, que examina la eficiencia del sistema a partir de métricas clave como precisión, tiempo de respuesta y rendimiento general. El proceso implica el flujo secuencial de datos a través de diferentes etapas de validación y transformación, donde se asegura que cada módulo cumpla con los estándares esperados. Para el cálculo del rendimiento se aplicará la fórmula (Aude et al., 1988):

$$T_r = n/t$$

En la cual T_r es el rendimiento, n el número de solicitudes procesadas, y t el tiempo total en segundos. Este enfoque garantiza una evaluación consistente y alineada con los objetivos de optimización del sistema.

Métodos

La herramienta desarrollada se adapta especialmente al entorno académico al proporcionar módulos de transformación que permiten a los estudiantes aplicar técnicas de preprocesamiento y análisis de datos. Estas capacidades son útiles en proyectos de aprendizaje automático, investigaciones académicas y cursos avanzados en ingeniería de datos.

Requisitos del sistema

La metodología propuesta se diseñó a partir de los objetivos establecidos en la investigación, de esta forma se asegura el desarrollo de un prototipo funcional. Cada etapa se centró en asegurar la eficiencia y cohesión de cada uno de los microservicios involucrados en el sistema.

El proyecto establece como requisito principal la implementación de microservicios que soporten la transformación eficiente de datos, lo cual permite a los usuarios autenticarse mediante un sistema seguro basado en *tokens* y gestionar permisos según su rol. El sistema debe integrar diversas tecnologías a través de API para facilitar la comunicación entre el *frontend* y el *backend*, esto garantiza la actualización dinámica de los datos.

Entre las necesidades clave se encuentra la capacidad de transformar datos estructurados, principalmente archivos CSV, y ofrecer a los usuarios la opción de elegir la herramienta de transformación adecuada. La ejecución de estas transformaciones se realizará en un servidor independiente, para asegurar de esta manera una comunicación fluida entre el usuario y los microservicios. Además, el sistema deberá proporcionar una previsualización del conjunto de datos transformado para validar los cambios antes de su descarga, con un enfoque en mantener un alto rendimiento tanto en el procesamiento de las solicitudes como en la carga de la interfaz.

Diseño rápido

El diseño de una vista intuitiva facilita y acelera el proceso de preprocesamiento para el usuario. A continuación se presenta un análisis de las funciones clave en un maquetado de transformación de datos para explicar cómo la arquitectura visual y funcional puede hacer que el flujo de trabajo sea autoexplicativo.

Homogeneización de categorías

Descripción general de estandarización de dominios del conjunto de datos y las características de cada columna, lo cual permite identificar posibles dominios divergentes.

Revista Electrónica Científica de Investigación Educativa
Volumen 9, enero-diciembre 2025, e2460

Esto es importante para estandarizar las variables. Este paso es clave para asegurar que los datos que se presenten dentro de esta columna sean uniformes y que no existan múltiples dominios que representen el mismo tipo de datos de manera inconsistente.

Limpieza de registros duplicados

Se presentan columnas con una descripción del conjunto de datos donde se indica igualmente posibles columnas de registros duplicados que podría no tener valor en el análisis. Se presenta la posibilidad de eliminar estas columnas para limpiar el conjunto de datos. Internamente, la función "drop_duplicates()" de la librería Pandas sigue un conjunto de operaciones bien optimizadas para eliminar registros duplicados de un *DataFrame*. El sistema indica si existen duplicados o no en el conjunto de datos, y se da la opción de eliminarlos automáticamente si se detectan. Si no se detectan aparecerá una vista diferente para indicar que no hay datos duplicados.

Eliminación de columnas no útiles

La opción de eliminar columnas que no agregan valor significativo, esto asegura que solo se mantengan los datos más relevantes. También ayuda a detectar si tienen poca o ninguna variación, que puede ser eliminada si no es relevante. Hay razones por las que una columna puede ser no informativa, como por ejemplo las columnas con valores únicos o constantes que no ofrece ninguna información adicional que pueda ayudar a los algoritmos, así como columnas con demasiados valores nulos o vacíos (ver Figura 2).

Figura 2
Eliminación de columnas no útiles

Eliminación de columnas no útiles

#	COLUMN	NON-NULL COUNT	DTYPE	DROP COLUMN
0	wine_id	50	int64	Delete this column
1	color	50	object	Delete this column
2	acidity	50	float64	Delete this column
3	sugar_content	50	float64	Delete this column
4	alcohol_content	50	float64	Delete this column
5	region	50	object	Delete this column

Nota: Se presenta como ejemplo la vista de la sección de eliminación de columnas no útiles.
Fuente: Elaboración propia.

Transformación de variables categóricas a numéricas

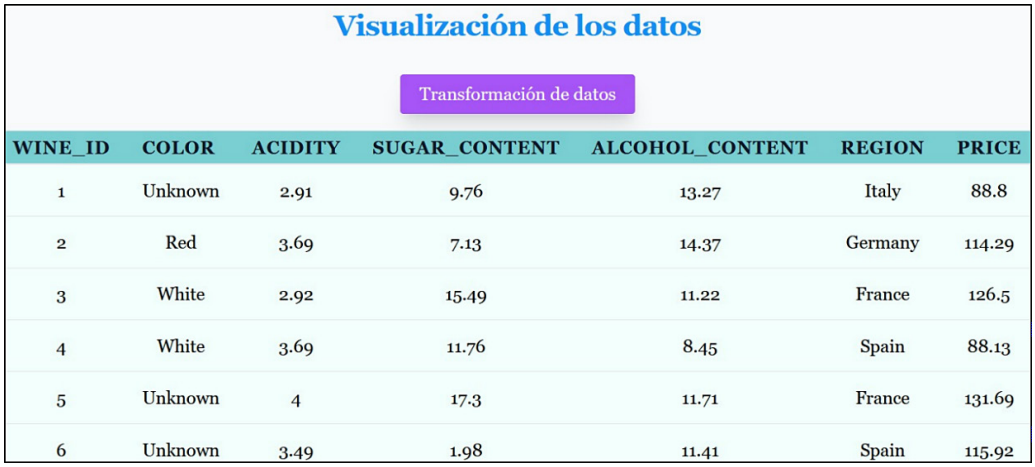
Permite la opción de seleccionar el tipo de codificación y permite a los usuarios aplicar codificación nominal u ordinal de forma intuitiva al transformar cadenas en valores numéricos para que sean utilizables en algoritmos de aprendizaje automático.

La mayoría de los algoritmos de aprendizaje automático y análisis estadístico no pueden trabajar directamente con datos en formato de cadena o texto, necesitan que las variables sean numéricas para poder realizar operaciones matemáticas, como cálculos de distancia o creación de modelos predictivos; esto no solo facilita que los algoritmos procesen los datos de manera más eficiente sino que también optimiza el uso de recursos computacionales, reduce el riesgo de errores y permite identificar patrones más claros en el análisis.

Visualización de los datos

Se realiza un resumen de los datos después de la recolección y el preprocesamiento, con los valores ya estandarizados y listos para ser analizados (ver Figura 3).

Figura 3
Visualización de los datos



WINE_ID	COLOR	ACIDITY	SUGAR_CONTENT	ALCOHOL_CONTENT	REGION	PRICE
1	Unknown	2.91	9.76	13.27	Italy	88.8
2	Red	3.69	7.13	14.37	Germany	114.29
3	White	2.92	15.49	11.22	France	126.5
4	White	3.69	11.76	8.45	Spain	88.13
5	Unknown	4	17.3	11.71	France	131.69
6	Unknown	3.49	1.98	11.41	Spain	115.92

Nota: Vista inicial del sistema desarrollado.

Fuente: Elaboración propia.

Desarrollo del prototipo

El sistema se desarrolló con base en tres tecnologías clave, cada una seleccionada por sus beneficios específicos en el desarrollo de aplicaciones web basadas en microservicios. Laravel fue elegido por su simplicidad y eficiencia en el manejo de peticiones y la gestión de datos en el *backend*. Además, al ser un *framework* PHP escalable y rápido, permite un manejo óptimo de las solicitudes provenientes del *frontend*.

Alpine.js se utilizó en el *frontend* por su ligereza y capacidad para gestionar interacciones dinámicas sin necesidad de recargar la página, lo que mejora la experiencia

del usuario. Esta tecnología es una alternativa moderna y eficiente frente a soluciones como jQuery, integrándose fácilmente con Laravel. Finalmente, Django fue empleado en el *backend* para tareas de procesamiento de datos complejos, destacándose por su seguridad, flexibilidad y estructura modular, ideal para aplicaciones que requieren un manejo eficiente y seguro de los datos.

Se adoptó un enfoque de desarrollo en paralelo para cada una de las tecnologías mencionadas. Esto permitió que diferentes equipos trabajaran de manera independiente, lo cual aseguró que los diferentes servicios se integraran al final del proceso. Los microservicios se encargan de manejar tareas específicas del sistema, como la gestión de solicitudes y la transformación de datos. Este enfoque permitió que los equipos de desarrollo trabajaran en paralelo, de esta manera se aseguró la correcta cohesión entre los componentes.

Arquitectura del sistema y comunicación entre componentes

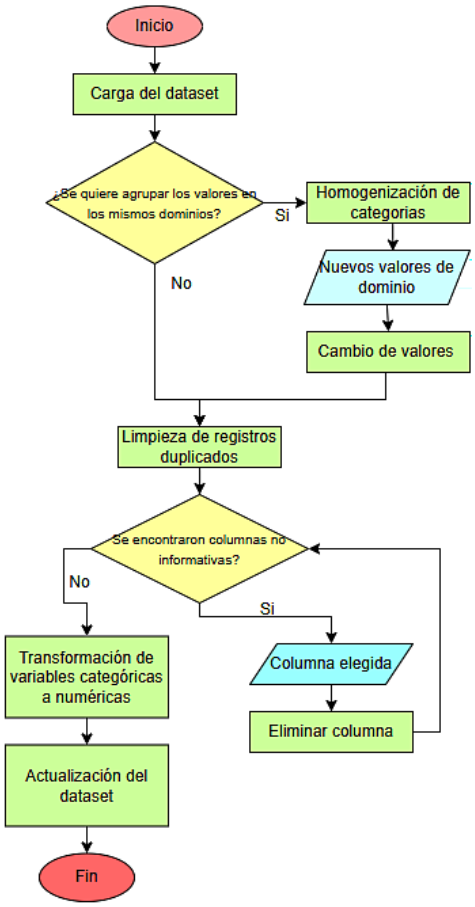
La aplicación se desarrolló utilizando como base una arquitectura de microservicios distribuida. A continuación se describe cómo se comunicaron las diferentes tecnologías:

- *Frontend* (Alpine.js) → Laravel → Django (*Backend*): la interfaz del usuario, gestionada por Alpine.js, envía las solicitudes a través de Laravel mediante peticiones HTTP (JSON). Laravel actúa como intermediario y maneja las solicitudes hacia Django, que se encarga de procesar y transformar los datos.
- Alpine.js: el usuario interactúa con la interfaz web, que es manejada por Alpine.js. Las acciones del usuario, como el envío de formularios o la selección de opciones, generan peticiones HTTP.
- Laravel: recibe las peticiones en formato JSON desde Alpine.js, las procesa y las redirige al *backend* correspondiente, en este caso, un servicio gestionado por Django.
- Django: una vez que recibe las solicitudes, Django aplica los algoritmos necesarios para transformar los datos. Responde a Laravel con los resultados en formato JSON, que son luego enviados al *frontend*.
- Devolución de resultados: Alpine.js actualiza dinámicamente la interfaz de usuario con los datos transformados, sin necesidad de recargar la página.

Las etapas seguidas para implementar un algoritmo *pipeline* para la transformación de datos se organizaron en un diagrama de flujo (ver Figura 4).

- Carga del *dataset*: se recibe el archivo desde el *frontend* y se almacena para su procesamiento.
- Homogeneización de categorías: se estandarizan los valores en las columnas categóricas del *dataset*.
- Limpieza de registros duplicados: se identifican y eliminan registros duplicados.
- Eliminación de columnas no útiles: el usuario selecciona qué columnas eliminar basándose en su relevancia.

Figura 4
Diagrama de flujo



Nota: Este diagrama de flujo muestra el recorrido de la información y las funciones que ofrece el sistema.
Fuente: Elaboración propia.

- Transformación de variables categóricas: se aplican transformaciones de codificación.

Este proceso asegura que los datos estén listos para ser utilizados en posteriores análisis o integraciones dentro de la misma plataforma.

Evaluación del proyecto

Para realizar la evaluación del proyecto basado en el algoritmo *pipeline* de procesamiento de datos se abordó la métrica del tiempo de respuesta, que es fundamental para medir la eficiencia del sistema en cuanto a cómo responde a cada solicitud que pasa a través del sistema y por consecuencia la velocidad de este. Para esto se utilizó la herramienta ClockWork, la cual proporciona información detallada sobre las solicitudes HTTP como tiempo de ejecución en diferentes parámetros de cada solicitud (Amarulloh et al., 2023).

La primera fase del algoritmo *pipeline* es la ingesta de datos, que consiste en preparar la información necesaria para su posterior preprocesamiento, en el caso del

proyecto este paso se realiza al cargar un archivo de tipo CSV. En esta etapa, los datos se reciben a través de Laravel y Alpine.js, donde se procesan las solicitudes HTTP para enviar los datos a los microservicios en Django (Leipzig, 2017).

Fórmula:

$$\textit{Tiempo de respuesta total (Ingesta)} = \textit{Tiempo de solicitud HTTP} + \textit{Tiempo de procesamiento en Laravel}$$

Los datos obtenidos en la solicitud para la ingesta son los siguientes:

$$1048\ ms = 868\ ms + 170\ ms$$

El tiempo de respuesta del servidor en esta etapa incluye el tiempo que toma a Laravel recibir los datos y enviarlos al *backend* para su procesamiento y guardado.

Una vez que se sabe el tiempo de la ingesta de datos de un conjunto de datos en general, es momento de cargar diferentes conjuntos de datos al sistema, con la finalidad de conocer su comportamiento bajo distintos tipos de escenarios. Esto ayudará a evaluar el estrés causado en cada una de las fases y a determinar los microservicios que pueden ser mejorados o que tienen un mal funcionamiento.

El primer conjunto de datos cargado contiene 919 registros, de los cuales 735 corresponden a casos positivos y el resto a casos negativos. La mayoría de los registros son datos numéricos, por lo que se esperaba un tiempo de respuesta corto, sin embargo, los tiempos de respuesta promedio fueron de 500 milisegundos. Este tiempo es lento, pero aceptable. No obstante, hay un valor atípico que se aleja considerablemente de la media, correspondiente al proceso de limpieza de datos duplicados (Kaplarevic, 2023) (ver Tabla 1).

Tabla 1
Resultados primer conjunto de pruebas

Fase	Memoria (MB)	Controlador (MS)	Django (MS)	Tiempo total (MS)
Visualización de datos	24	319	248	567
Target y *features*	22	230	127	357
Homogenización categorías	20	292	138	430
Limpieza duplicados	20	560	442	1002
Eliminar columnas no útiles	20	273	112	385
Variable categórica a numérica	20	247	121	367
Media	21	320.17	198	518

Nota: Se agregaron las medidas de tendencia central para obtener los datos más representativos de las pruebas.

Fuente: Elaboración propia.

El segundo conjunto de datos es considerablemente más grande, contiene 4,653 registros, de los cuales 3,053 corresponden a casos negativos, el resto a positivos. La mayoría de los registros son datos categóricos, por lo que se esperaba un tiempo de

respuesta más largo que el del conjunto anterior. El tiempo promedio de las solicitudes fue de 600 milisegundos, lo cual indica que se tiene un tiempo de respuesta más lento debido a la cantidad de registros y a que la mayoría de las columnas contiene datos categóricos (ver Tabla 2).

Tabla 2
Resultados segundo conjunto de pruebas

Fase	Memoria (MB)	Controlador (MS)	Django (MS)	Tiempo total (MS)
Visualización de datos	24	802	227	1078
Target y *features*	22	567	154	721
Homogenización categorías	20	343	491	834
Limpieza duplicados	20	543	181	724
Eliminar columnas no útiles	20	323	147	470
Variable categórica a numérica	20	326	159	485
Media	21	484	226.5	618.3

Nota: En este segundo conjunto de resultados se incluyeron de igual manera las medidas de tendencia central.

Fuente: Elaboración propia.

Se realizaron pruebas con cinco conjuntos de datos en total, con una media de 2,786 registros y un tiempo de respuesta final de media de 520 milisegundos, lo cual arroja una respuesta que puede ser considerada lenta.

Solicitudes por segundo

Finalmente se realizó una prueba de rendimiento con la finalidad de determinar la cantidad de solicitudes que el sistema puede procesar en 90 segundos, esto se aplicó mediante el uso de la siguiente formula (Aude et al., 1988):

$$T_r = n/t$$

Donde T_r es el rendimiento del sistema, n el número de solicitudes procesadas, y t el tiempo total en segundos. Cada solicitud corresponde a una parte diferente del sistema, este enfoque garantiza una evaluación consistente para todos los microservicios utilizados dentro del proyecto. El resultado de las pruebas arrojó un total de 126 solicitudes procesadas en 70 segundos, lo que da un total de 1.8 solicitudes por segundo, como se muestra en la formula desarrollada.

$$1.8 = \frac{126}{70}$$

Discusión

En los experimentos realizados se observó que el sistema alcanzó un rendimiento satisfactorio en la mayoría de los escenarios, especialmente al trabajar con datos numéricos. El tiempo de respuesta fue consistente y se mantuvo dentro de los parámetros

esperados, con un promedio de 500 milisegundos para conjuntos de datos menores a mil registros. Este resultado se alinea con lo reportado en estudios previos.

Así mismo, los tiempos de respuesta obtenidos en pruebas aplicadas a contextos educativos fueron suficientemente rápidos para entornos académicos. Esto permite que los estudiantes realicen iteraciones ágiles durante sus análisis y se concentren en la interpretación de los resultados, sin distraerse con los aspectos técnicos del procesamiento (Kaplarevic, 2023).

Sin embargo, al procesar datos categóricos, especialmente en conjuntos de datos mayores a cuatro mil registros, se detectó un incremento significativo en los tiempos de respuesta, estos tiempos superaron los 600 milisegundos en promedio. Este comportamiento sugiere que la optimización del procesamiento de variables categóricas es un área que requiere atención para mejorar la eficiencia general del sistema.

Además, en las pruebas de limpieza de duplicados se encontró un valor atípico que incrementó los tiempos de procesamiento, lo cual resalta la necesidad de refinar este módulo para reducir su impacto en el rendimiento general del sistema.

Conclusiones

En este artículo se presenta el desarrollo de un proyecto que integra diferentes tecnologías las cuales interactúan a lo largo del proceso de envío de información desde el *frontend* hasta el *backend* para garantizar una comunicación eficiente.

En el desarrollo de este artículo se abordó la integración de diferentes tecnologías de comunicación entre microservicios en un entorno de desarrollo web que utiliza diversas tecnologías (Laravel, Alpine.js y Django) para gestionar datos y permitir su transformación.

La implementación de una arquitectura basada en microservicios permitió dividir la aplicación en componentes más manejables, lo cual facilitó la escalabilidad y el mantenimiento del sistema, aunque también planteó retos de cohesión y optimización de rendimiento.

Para enfrentar la problemática, se desarrolló un prototipo mediante la metodología *Prototype*, que permitió un diseño iterativo del sistema. A través de un enfoque de desarrollo en paralelo para cada tecnología, se logró implementar un *pipeline* de procesamiento de datos, que incluye la carga de datos, la homogeneización de categorías, la limpieza de duplicados y la transformación de variables categóricas. Estos procesos fueron evaluados mediante el uso de métricas de tiempo de respuesta y carga de memoria, donde se aplicaron algoritmos específicos para asegurar que los microservicios funcionaran de forma óptima.

En el contexto educativo, este proyecto busca ser una herramienta innovadora para estudiantes universitarios interesados en la transformación de datos. Su diseño facilita que los alumnos se familiaricen con tecnologías modernas y comprendan los principios fundamentales de la integración de microservicios.

Esto contribuye a fortalecer su formación práctica en el manejo de datos y el desarrollo de aplicaciones escalables. Además, el uso conjunto de tecnologías como Laravel, Alpine.js y Django ofrece un ejemplo claro de cómo los microservicios pueden aplicarse en proyectos educativos para resolver desafíos complejos de procesamiento de datos.

Los resultados obtenidos mostraron que el sistema responde adecuadamente al procesar datos numéricos en conjuntos menores a 1,000 registros, con un tiempo promedio de 500 milisegundos. Estos valores se consideran aceptables dentro del contexto de uso previsto.

Sin embargo, al manejar grandes volúmenes de datos categóricos el rendimiento disminuyó notablemente, superando los 600 milisegundos. Esto señala oportunidades de mejora en el módulo de transformación de datos categóricos. Además, las pruebas de limpieza de duplicados arrojaron valores atípicos en los tiempos de respuesta, lo que sugiere la necesidad de optimizar este módulo para evitar ralentizaciones significativas.

Este prototipo presenta no solo el cumplimiento de los objetivos iniciales, sino que también constituye una herramienta valiosa para la enseñanza y el aprendizaje de la integración de microservicios y la transformación de datos. En el futuro, se optimizarán los módulos de transformación de datos categóricos y de limpieza de duplicados, y se desarrollarán las dos últimas etapas de la metodología *Prototype*. Estas mejoras reducirán los tiempos de respuesta y maximizarán la eficiencia del sistema, fortaleciendo su robustez y escalabilidad en entornos de producción complejos.

Referencias

Amarulloh, A., Kurniasih, K., y Muchlis, M. (2023). Analisis perbandingan performa web service rest menggunakan framework Laravel, Django, dan node js pada aplikasi berbasis website. *Jurnal Teknik Informatika*, *9*(1), 14-19. https://ejournal.antarbangsa.ac.id/jti/article/view/515

Aude, J. S., Paiva, E. B. M. O., Aude, E. P. L., Filho, E. P. L., Martins, M. F., y Pinto, S. B. (1988). *Acelerador modular para roteamento*. En *International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2 (pp. 90-97). https://doi.org/10.5753/sbac-pad.1988.23523

Chapkovski, P., y Kujansuu, E. (2019). Real-time interactions in oTree using Django Channels: Auctions and real effort tasks. *Journal of Behavioral and Experimental Finance*, *23*, 114-123. https://doi.org/https://doi.org/10.1016/j.jbef.2019.05.008

De Luca, V., Epicoco, I., Lezzi, D., y Aloisio, G. (2011). A web API framework for developing grid portals. *Procedia Computer Science*, *4*, 392-401. https://doi.org/10.1016/j.procs.2011.04.041

Ekholm, E. (2024). Review of HTML-oriented state managers for web development [Tesis de grado, Aalto University]. https://urn.fi/URN:NBN:fi:aalto-202405283848

Felício, D., Simão, J., y Datia, N. (2023). RapiTest: Continuous black-box testing of RESTful Web APIs. *Procedia Computer Science*, *219*, 537-545. https://doi.org/10.1016/j.procs.2023.01.322

Fortin, M., Lavoie, J.-F., Régnière, J., y Saint-Amant, R. (2022). A Web API for weather generation and pest development simulation in North America. *Environmental Modelling & Software*, *157*, 105476. https://doi.org/https://doi.org/10.1016/j.envsoft.2022.105476

Gopalakrishnan, V., y Ramaswamy, C. (2017). Patient opinion mining to analyze drugs satisfaction using supervised learning. *Journal of Applied Research and Technology*, *15*(4), 311-319. https://doi.org/10.1016/j.jart.2017.02.005

Harris, C. (2020). Arquitectura de microservicios. *Atlassian*. https://www.atlassian.com/es/microservices/microservices-architecture

Kaplarevic, V. (2023, dic. 20). 7 ways to reduce server response time. *PhoenixNAP*. https://phoenixnap.com/kb/reduce-server-response-time

Laaziri, M., Benmoussa, K., Khoulji, S., y Kerkeb, M. L. (2019). A comparative study of PHP frameworks performance. *Procedia Manufacturing*, *32*, 864-871. https://doi.org/https://doi.org/10.1016/j.promfg.2019.02.295

Leipzig, J. (2017). A review of bioinformatic pipeline frameworks. *Briefings in Bioinformatics*, *18*(3), 530-536. https://doi.org/10.1093/bib/bbw020

Markert, K. N., da Silva, G., Ames, D. P., Maghami, I., Williams, G. P., Nelson, E. J., Halgren, J., Patel, A., Santos, A., y Ames, M. J. (2024). Design and implementation of a BigQuery dataset and application programmer interface (API) for the U.S. National Water Model. *Environmental Modelling and Software*, *179*, 106123. https://doi.org/10.1016/j.envsoft.2024.106123

Mustapha, A., Abdellah, K., Mohamed, L., Khalid, L., Hamid, H., y Ali, K. (2023). DLDiagnosis: A mobile and web application for diseases classification using Deep Learning. *SoftwareX*, *23*, 101488. https://doi.org/10.1016/j.softx.2023.101488

Pilario, K. E. (2024). Teaching classical machine learning as a graduate-level course in chemical engineering: An algorithmic approach. *Digital Chemical Engineering*, *11*, 100163. https://doi.org/https://doi.org/10.1016/j.dche.2024.100163

Prasai, R., Schwertner, T. W., Mainali, K., Mathewson, H., Kafley, H., Thapa, S., Adhikari, D., Medley, P., y Drake, J. (2021). Application of Google earth engine python API and NAIP imagery for land use and land cover classification: A case study in Florida, USA. *Ecological Informatics*, *66*, 101474. https://doi.org/https://doi.org/10.1016/j.ecoinf.2021.101474

Pujadas, R., Valderrama, E., y Venters, W. (2024). The value and structuring role of web APIs in digital innovation ecosystems: The case of the online travel ecosystem. *Research Policy*, *53*(2), 104931. https://doi.org/10.1016/j.respol.2023.104931

Rojas, J., Gamboa-Cruzado, J., y de la Cruz, P. (2023). Systematic literature review on machine learning and its impact on APIs deployment. *Computacion y Sistemas*, *27*(4), 1107-1124. https://doi.org/10.13053/CyS-27-4-4371

Cómo citar este artículo:

Solorzano Rangel, J. d. J., Cruz Mendoza, M. C., y Primero Huerta, C. (2025). Integración de microservicios con Laravel, Python y Django para la enseñanza del procesamiento de datos. *RECIE. Revista Electrónica Científica de Investigación Educativa*, *9*, e2460. https://doi.org/10.33010/recie.v9i0.2460