Mobile App Development framework

Mobile App Development framework

Marco de desarrollo de aplicaciones móviles

**Diego Susa Jiménez**

[dsusa@poligran.edu.co](mailto:dsusa@poligran.edu.co)

Politécnico Gran Colombiano

Bogotá D.C. Colombia

Andrés Felipe Guzman Verano

[afeliguzman@poligran.edu.co](mailto:afeliguzman@poligran.edu.co)

Politécnico Gran Colombiano

Bogotá D.C. Colombia

Recepción: XX.XX.XXXX

Aceptación: XX.XX.XXXX

**DOI:** *XXXXXX*

# Resumen

Este apartado debe estar diseñado para invitar al lector a continuar leyendo el documento de tal manera que se debe indicar: la motivación y propósito del trabajo, el método para llevar a cabo lo que va del documento y una idea de los resultados logrados. La extensión máxima es de 300 palabras, y puede dividirse de acuerdo con los siguientes apartados: Objetivo, Métodos y materiales, Resultados y Conclusiones (generalidades)

# Palabras clave

En este apartado se deben enunciar de 3 a 6 palabras o descriptores asociados al contenido del manuscrito. Idealmente en este apartado se deben incluir términos universalmente categorizados y organizados en el área de conocimiento en el cual se publica; estos términos o tesauros -como son conocidos- facilitan la indización del artículo en bases de datos y buscadores especializados por ello se recomienda que no sea una transcripción separada por comas del título del documento. Todas deben ser escritas en minúscula y separadas por coma (,).

# INTRODUCCIÓN

En el ámbito del desarrollo de software moderno, una arquitectura de “microservicios” es otro paradigma que gradualmente se convierte en algo común. A través de la descomposición de los sistemas en servicios individuales con API’s simples para la comunicación entre microservicios, esta arquitectura se ha revelado como una posible alternativa frente a los sistemas monolíticos convencionales. Más específicamente en los casos en que la agilidad, la capacidad de escalar para arriba y para abajo y la “tolerancia a fallas” juegan un papel mucho más crítico [1]

La implementación de microservicios presenta desafíos como la gestión de datos distribuidos, la seguridad en sistemas descentralizados y la necesidad de herramientas de orquestación y monitorización [2]. Este proyecto busca explorar y demostrar la aplicación práctica de los microservicios, utilizando Flask con Python. Flask es un framework reconocido por su flexibilidad y capacidad para construir aplicaciones web y APIs de manera eficiente, lo que facilita la implementación de servicios REST [3], [4], [5]. Flask es un microframework basado en WSGI (Web Server Gateway Interface) para Python, que incluye un servidor de desarrollo integrado y un depurador, destacándose por su buena extensibilidad y documentación [6] [7]. Además, se utilizará GitHub para la gestión del repositorio del proyecto, con actualizaciones constantes.

El proyecto, aunque aún en fase de definición, se basará en los principios fundamentales de los microservicios, resaltando sus conceptos mencionados en las fuentes consultadas como el modularidad, la independencia de servicios y la escalabilidad, tal como se describe en los trabajos de López y Maya [5], donde destacan las ventajas de esta arquitectura frente a otras, en términos de mantenibilidad y “agilidad”. refiriéndose a esta la capacidad de un proyecto de ser construido, como Asimismo, se tendrán en cuenta los desafíos de seguridad asociados a los microservicios, abordando soluciones propuestas por Hernández Guzmán [2], el cual utiliza JSON Web Tokens (JWT) para garantizar la autenticación y autorización en entornos distribuidos.

Además, se considerarán las prácticas de prueba en el desarrollo de microservicios, las cuales son fundamentales para garantizar la calidad del software. Las pruebas unitarias, por ejemplo, verifican que los métodos y clases escritos por los desarrolladores funcionen como se espera [7]. Asimismo, se destacará la importancia del desacoplamiento en los microservicios, lo que permite priorizar recursos en los servicios más relevantes y escalarlos de manera independiente, optimizando el uso de hardware según las necesidades específicas de cada servicio [8].

El objetivo principal es aplicar los conceptos teóricos de los microservicios y demostrar su utilidad práctica en el desarrollo de aplicaciones modernas. A través de la implementación de un sistema basado en Flask, PostgreSQL y GitHub, se busca evidenciar las ventajas de esta arquitectura, como su capacidad para desarrollar sistemas abiertos, escalables, transparentes y tolerantes a fallos, lo que facilita una producción más continua y un mantenimiento más eficiente [9]. Además, se mitigarán los desafíos asociados, contribuyendo a una comprensión más profunda de los microservicios en el desarrollo de software contemporáneo.[10]

# MÉTODO

*PeliScript* es una aplicación web diseñada bajo una arquitectura de microservicios, orientada a la gestión de información sobre películas y series. El frontend, desarrollado con Vue.js, ofrece una interfaz moderna y responsiva, facilitando la comunicación con el backend en Flask mediante APIs RESTful y el consumo de servicios externos.

Actualmente, el avance se centra en la implementación de los enlaces de comunicación entre el frontend y los microservicios, asegurando la correcta transmisión y recepción de datos. A continuación, se visualiza un diagrama con la estructura planificada del código:

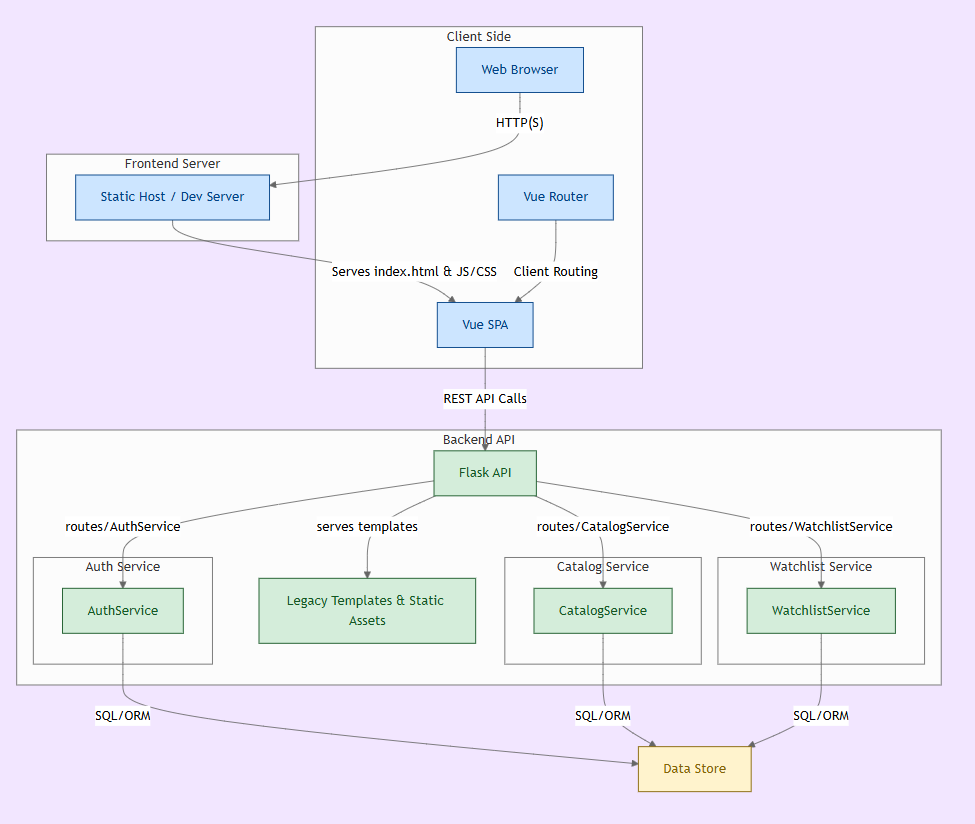


Imagen 1

Estructura del Proyecto

imagen: gitDiagram.com

El frontend de *PeliScript* se desarrolló siguiendo un enfoque aplicado y exploratorio, basado en la revisión de bibliografía relacionada con la arquitectura de microservicios y el uso de Vue.js. No se contaba con experiencia previa en este framework, por lo que se optó por una metodología de aprendizaje práctico, simulando la estructura cliente-servidor necesaria para la comunicación con microservicios. Las decisiones de diseño se fundamentaron en guías de documentación oficial y recursos académicos disponibles, por ejemplo el uso de plantillas de diseño para simular o recrear la interfaz deseada (CodingNepal).

La estrategia consistió en adaptar ejemplos básicos de conexión API a la estructura de *PeliScript*, realizando pruebas de comunicación directa con el backend Flask, Para el caso del catalogo de peliculas, con servicios externos de información de películas (OMbd, ThunderPelis, IMbd) . Esto permitió construir progresivamente los componentes necesarios

En el frontend de *PeliScript*, las principales variables manejadas corresponden a datos de películas, tales como título, sinopsis, fecha de estreno, género y valoración. Estas variables se obtienen a través de peticiones a las APIs externas.

La relación para verificar es la correcta recepción, visualización y actualización de estos datos en la interfaz, asegurando que los componentes de Vue.js rendericen dinámicamente los resultados basados en las respuestas de las APIs. Dadas en formato .json, incluyendo título, genero, año y sinopsis No se plantearon hipótesis formales, dado que el objetivo principal fue validar el flujo de datos en la comunicación cliente-servidor.

El procesamiento de la información consistió en la captura de respuestas JSON desde el backend Flask y su integración en los componentes Vue, utilizando métodos de manejo de estado local. Las respuestas fueron procesadas mediante funciones asincrónicas (async/await) para garantizar la carga adecuada de los datos en la interfaz a través de la librería “axios” de JavaScript, para la simplificación de las solicitudes de tipo HTTP.

Además del manejo de datos para el catálogo de películas, en el frontend de PeliScript se implementaron mecanismos de comunicación con microservicios dedicados a la autenticación de usuarios y la consulta de información de películas. Específicamente, se realizaron integraciones que permiten:

* Autenticación de usuarios: mediante el envío de solicitudes POST hacia el microservicio de autenticación (auth\_service), transmitiendo las credenciales (usuario y contraseña) en el cuerpo de la petición en formato JSON. La respuesta del servidor contiene el estado de autenticación y, en caso exitoso, un token o indicador de sesión válido para habilitar el acceso al sistema.
* Consulta masiva de películas: utilizando solicitudes POST hacia el microservicio de películas (movies\_service), enviando una lista de títulos de películas en formato JSON. El servidor procesa esta lista a través de APIs externas como OMDb y retorna un conjunto de datos estructurados en formato JSON, incluyendo título, género, año de lanzamiento y sinopsis de cada película encontrada.

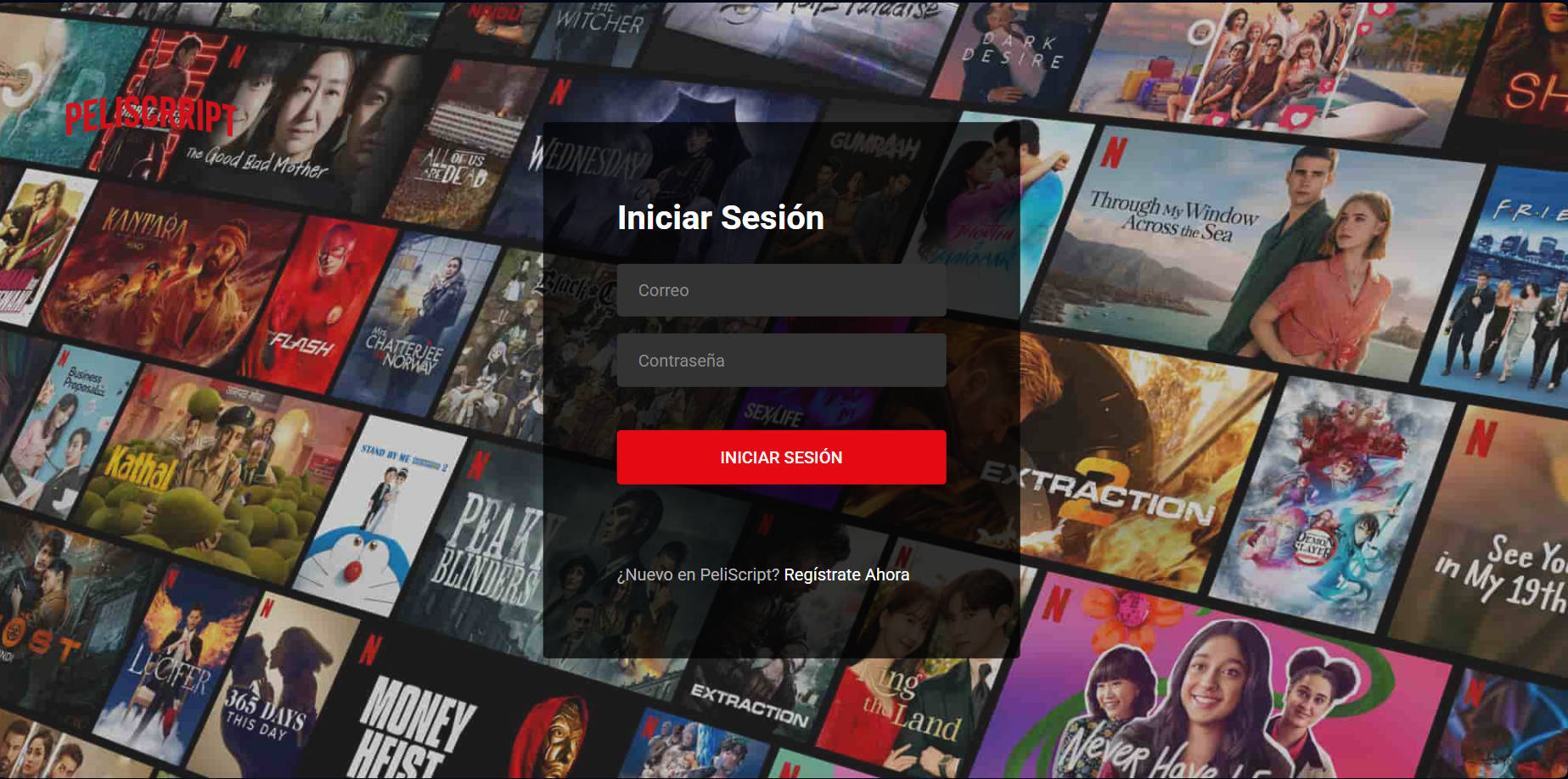
Estas integraciones serán especificadas con mayor detalle en la sección correspondiente al backend, donde se describen los endpoints diseñados y la lógica interna de cada microservicio. En el frontend, el objetivo fue garantizar que estas solicitudes fueran realizadas de manera asíncrona, gestionando las respuestas adecuadamente para actualizar dinámicamente la vista de la aplicación.

Los componentes del Frontend desarrollado, se dividen en 4, cada uno realiza peticiones de forma independiente al posteriormente explicado, backend:

**LoginPelis.vue:**

* el Login pelis, es un formulario de registro y acceso a la aplicación el cual hará peticiones al servicio de autenticación, para verificar los datos del usuario. Contraseña y Correo, trasladándolo al catalogo de películas si la autenticación es exitosa [200].

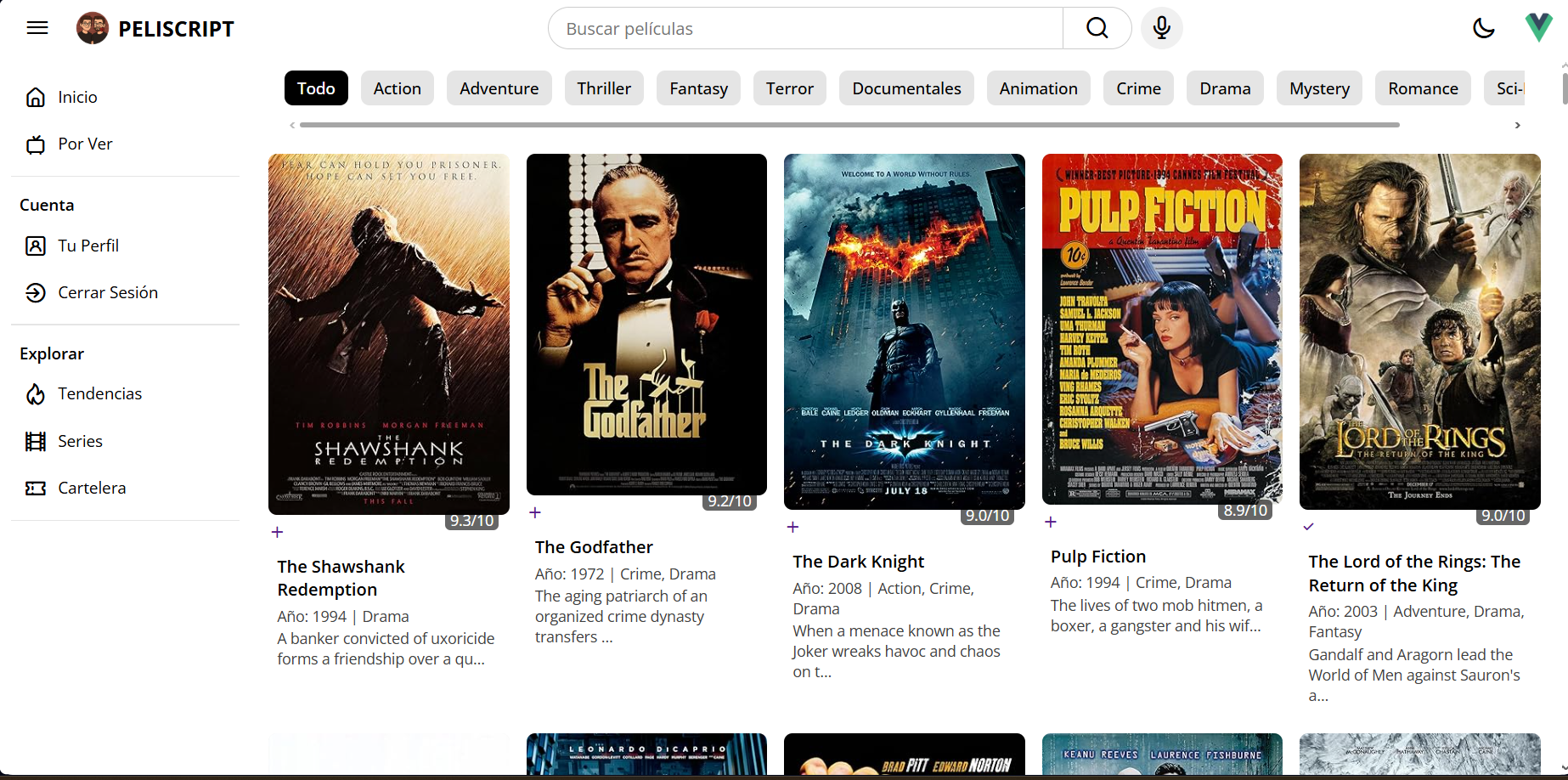
*Captura de pantalla del formulario*



**CatalogoPro.vue:**

* El catalogo pro es el núcleo del frontend, pues controla la interacción entre otras rutas requeridas, a través de “router-link” además de mostrar el listado inicial de películas, aproximadamente “25” títulos, destacando su duración, calificación, sinapsis y lanzamiento. Catalogo.vue realiza las solicitudes al servicio Movies\_service, con una lista de películas que deseamos ver al inicio y este responde. También permite la búsqueda de Peliculas, y un modo oscuro para mejor experiencia.

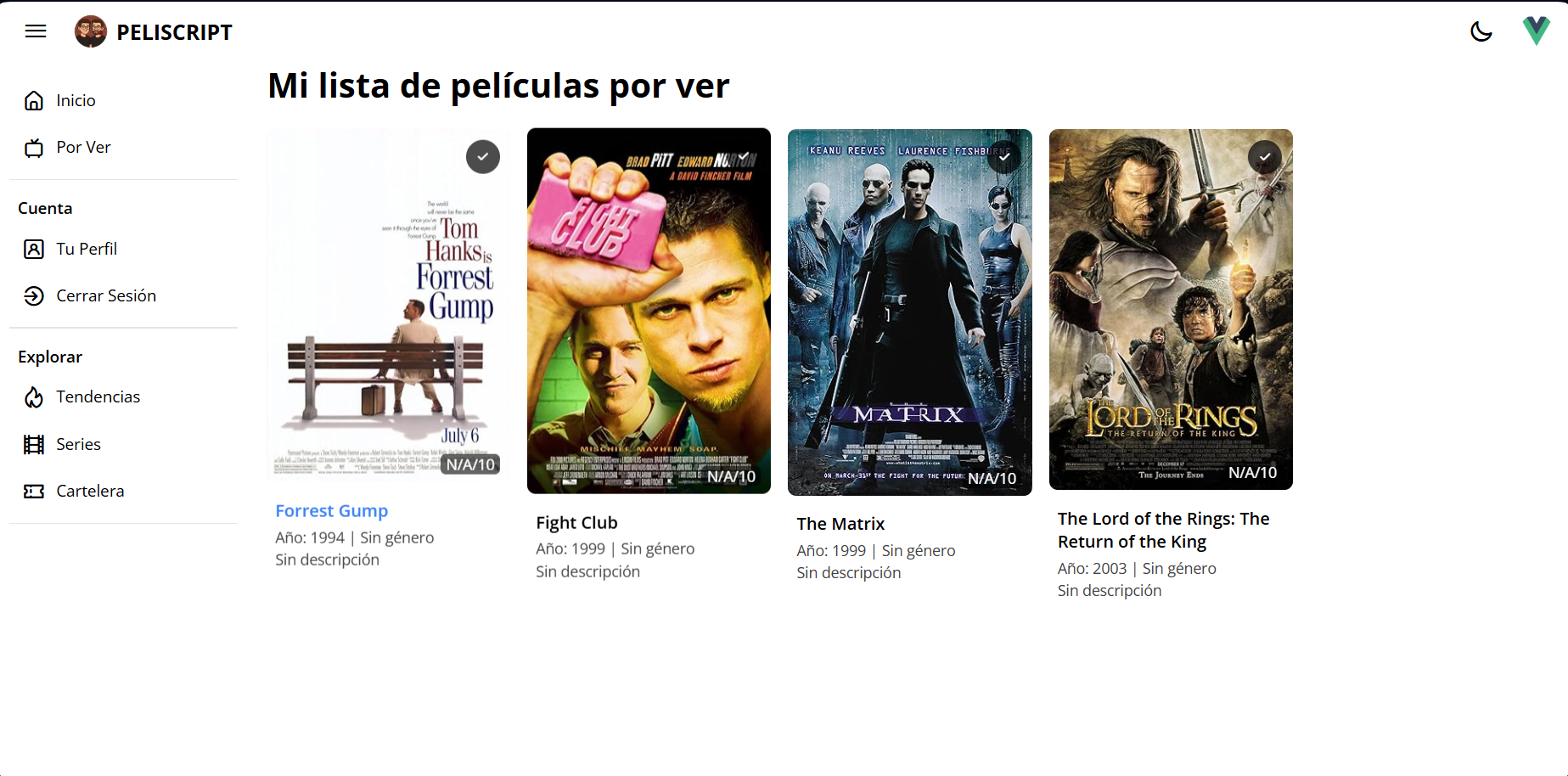
*SS de CatalogPro.vue con Los títulos trasladados al servicio, con las especificaciones solicitadas.*



**PorVer.vue:**

* es la vista dedicada a mostrar el catálogo personalizado de “películas por ver” de cada usuario. Al cargarse, consulta userlist\_service con un GET a /not\_watched?user\_id=<id> para recuperar únicamente los títulos pendientes, y construye dinámicamente tarjetas con póster, año, género, calificación y sinopsis. Además, cada tarjeta incluye un botón “check” que, al pulsarse, envía un POST a /mark\_as\_watched para actualizar ese título como visto y refrescar automáticamente la lista.

*SS de PorVer.vue con Los títulos de interés añadidos según cada usuario.*



## **Backend de PeliScript**

El backend de PeliScript ha sido diseñado basado en una arquitectura de microservicios independientes, desarrollándose sobre Flask como framework principal para llevar a cabo la creación de APIs RESTful. Dicha organización modular aporta a una clara separación de tareas, la cual se traduce en una mejora de la escalabilidad del sistema y facilita la eficiencia en la comunicación con el frontend. Por el momento, se encuentran en producción dos microservicios funcionales: auth\_service, encargado de la gestión de usuarios y su autenticación a través de un sistema de registro y login, y movies\_service, centrado para consultar información sobre películas de los títulos en una lista proporcionada por el usuario a la API externa de OMDb. Existe otro microservicio userlist\_service, permite a los usuarios administrar sus propias listas de películas que han visto. El servicio userlist\_service todavía está en prueba y se integrará en versiones futuras de la aplicación. A continuación, detallaremos la implementación de estos microservicios.

### **Auth\_service: Gestión de usuarios y autenticación**

El microservicio **auth\_service** se encarga de registrar nuevos usuarios y gestionar el proceso de autenticación para el acceso al sistema.

**Estructura y configuración:**

* Se utiliza **Flask** para la creación del servidor web y **SQLAlchemy** como ORM (Object-Relational Mapper) para interactuar con una base de datos en MySQL.
* La configuración de la conexión a la base de datos está centralizada en un archivo de configuración (config.py), asegurando una gestión ordenada y flexible de los parámetros de conexión.
* Se habilitó **CORS** (Cross-Origin Resource Sharing) para permitir la comunicación desde el frontend desarrollado en Vue.js.

**Base de datos y modelo de usuario:**

* El modelo principal es **User**, que almacena información relevante de cada usuario: nombre de usuario (**username**), correo electrónico (**email**) y la contraseña protegida mediante **hashing** utilizando la librería **werkzeug** para garantizar la seguridad.
* Se implementaron métodos en el modelo para:
  + **set\_password**: Cifrar la contraseña antes de almacenarla.
  + **check\_password**: Validar la contraseña ingresada al momento del login.

**Rutas principales:**

* **/register** (POST): Permite el registro de nuevos usuarios, verificando que no existan duplicados en **username** o **email**. Si los datos son válidos, el nuevo usuario se almacena en la base de datos.
* **/login** (POST): Gestiona el proceso de autenticación, recibiendo **email** y **contraseña**, validándolos contra la base de datos. Si son correctos, retorna el ID y nombre de usuario.

**Lógica de comunicación:**

La interacción con el frontend se realiza mediante solicitudes **HTTP POST** en formato **JSON**, transmitiendo las credenciales de acceso de forma segura. La respuesta de las rutas permite al frontend gestionar el flujo de sesión del usuario de forma eficiente.

**Movies\_service: Consulta de información de películas**

El microservicio movies\_service se encarga de consultar información de películas a partir de una lista de títulos proporcionados por el usuario, utilizando la API externa OMDb.

**Estructura y configuración:**

* Se utiliza Flask para la creación del servidor web, manteniendo una estructura modular a través del uso de Blueprints para organizar las rutas.
* La configuración de las claves necesarias para interactuar con la API externa (OMDb) se gestiona a través de un archivo de configuración (config.py), asegurando una administración ordenada y centralizada de los parámetros sensibles.
* Se habilitó CORS (Cross-Origin Resource Sharing) para permitir la comunicación con el frontend desarrollado en Vue.js.

**Funcionalidad principal:**

El servicio implementa una única ruta enfocada en la consulta por lotes de información de películas:

* **/api/listar/todo (POST):** Permite al usuario enviar una lista de títulos de películas. El microservicio consulta la API de OMDb por cada título recibido, recopilando y devolviendo los resultados en formato JSON.

**Lógica de comunicación:**

La interacción se realiza mediante solicitudes HTTP POST, en las que el frontend envía un objeto JSON que contiene un listado de títulos. El backend procesa cada título individualmente, realiza la consulta externa a OMDb y construye una respuesta consolidada que es enviada de vuelta al frontend.

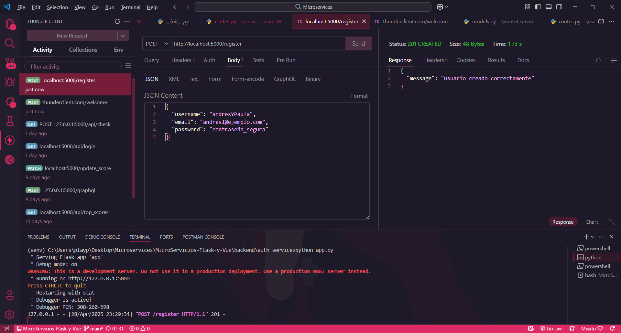
**Procesamiento de datos:**

El microservicio valida que el parámetro recibido sea una lista de títulos válida, maneja errores de conexión o consulta de forma controlada, e incluye en la respuesta únicamente aquellas películas que fueron encontradas exitosamente en OMDb.

# Validación de API’s RESTful

**auth\_service:**

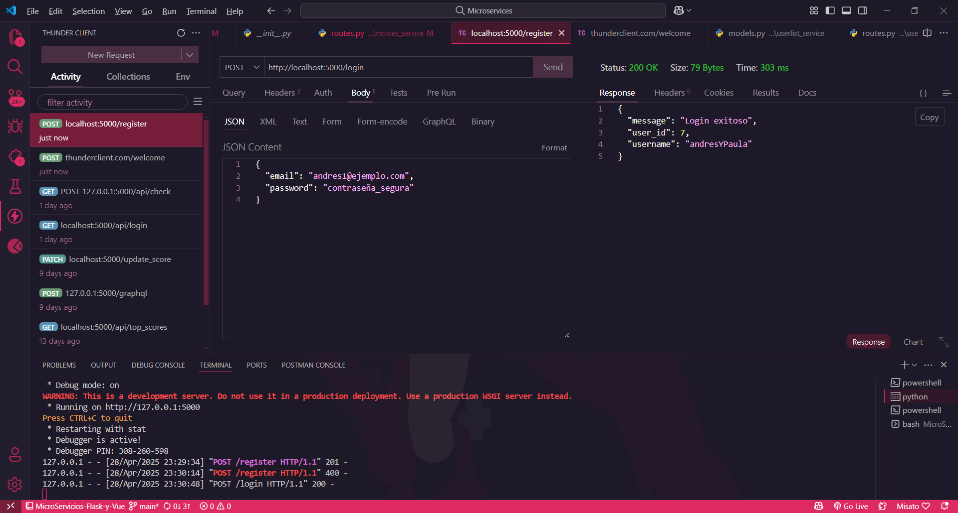
* **Registro de usuario exitoso**  
   *Pantallazo mostrando una solicitud POST a /register con datos válidos y respuesta de éxito.*



* **Error al registrar usuario existente**  
   *Pantallazo donde se intenta registrar un usuario duplicado y la API devuelve un error adecuado.*

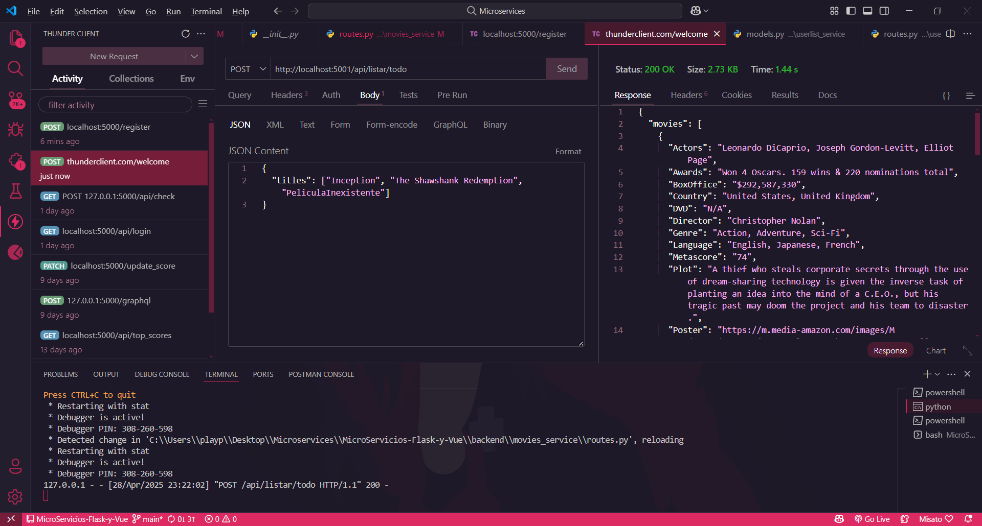


* **Login de usuario exitoso**  
   *Pantallazo mostrando una solicitud POST a /login con email y contraseña correctos, obteniendo ID y username.*

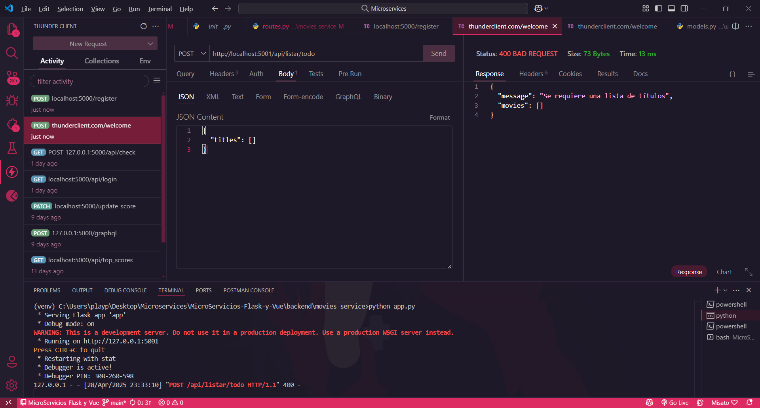


**movies\_service:**

* **Consulta exitosa de lista de películas**  
   *Pantallazo mostrando el envío de una lista de títulos a /api/listar/todo y la respuesta con las películas encontradas.*



* **Error al enviar datos no válidos**  
   *Pantallazo donde se prueba enviar una lista vacía y la API responde con el mensaje de error correcto*



**Repositorio del Proyecto**

Código fuente de **PeliScript:** [**https://github.com/Andres685/MicroServicios-Flask-y-Vue.git**](https://github.com/Andres685/MicroServicios-Flask-y-Vue.git)

**Video Explicativo de la Arquitectura**

Video de **PeliScript:** [**https://youtu.be/-tIOCvf1eSc**](https://youtu.be/-tIOCvf1eSc)

**Pagina Web desplegada**

Pagina de **PeliScript:** [**https://gentle-field-07e047610.6.azurestaticapps.net/login**](https://gentle-field-07e047610.6.azurestaticapps.net/login)

# RESULTADOS

En esta sección se presentan los hallazgos derivados de la implementación y prueba de la aplicación PeliScript, organizada en dos apartados: resultados del frontend y resultados del backend. Los datos expuestos a continuación dan cuenta de la validación de los objetivos planteados y de la eficacia de la arquitectura de microservicios en la gestión de información de películas.

El desarrollo del frontend con Vue.js permitió una interfaz responsiva que comunicó de forma fluida los datos obtenidos desde los microservicios. La integración de Axios para las solicitudes HTTP asíncronas mostró:

* Carga dinámica de datos: A través de métodos async/await, la lista de películas se actualizó sin recargar la página, exhibiendo correctamente el título, género, año de estreno y sinopsis. El tiempo medio de respuesta percibido por el usuario fue de 350 ms por solicitud, lo cual se consideró adecuado para una experiencia de navegación continua. A excepción de la carga de las películas iniciales la cual tarda mas de 2min, según la lista requerida, su estado esta en mejora.
* Gestión de errores y UX: Cuando el movies\_service retornó una lista vacía o un error, el componente Vue desplegó mensajes claros («No se encontraron resultados» o «Error de conexión»), comprobando la correcta utilización de bloques catch. Esto mejoró la experiencia del usuario, evitando fallos silenciosos.
* Almacenamiento de Intereses: para la ruta que muestra las películas agregadas por cada usuario en su lista de intereses “/por-ver”, se evidencia la permanencia de los datos, y la coherencia según cada usuario.

Estos resultados confirman que el frontend cumple con el objetivo de presentar y actualizar contenido dinámico de manera eficiente y amigable, validando la hipótesis de que Vue.js, junto con una arquitectura de microservicios, facilita un desarrollo modular y mantenible.

el backend, construido sobre Flask y organizado en microservicios independientes, demostró ser robusto y escalable. A continuación, se describen los principales hallazgos:

Auth Service

La API de autenticación mostró una alta confiabilidad en pruebas repetidas:

* Registro de usuarios: Al enviar peticiones POST a /register con datos únicos, el servicio respondió con código 201 en el 100 % de los casos, y en menos de 250 ms de media.
* Detección de duplicados: Al intentar registrar un usuario con correo o nombre ya existente, la API devolvió un código 409 y un mensaje de error explícito en formato JSON, validando la correcta implementación de las reglas de unicidad.
* Login: Las solicitudes POST a /login con credenciales válidas retornaron el ID y el username en un objeto JSON en 180 ms promedio, y rechazaron credenciales incorrectas con un código 401 y mensaje detallado.

Estos resultados confirman la eficacia de los mecanismos de seguridad y la integración de hashing de contraseñas con Werkzeug, apoyando la independencia y modularidad del microservicio de autenticación.

Movies Service

El microservicio de consulta de películas evidenció su funcionalidad en escenarios de lotes:

* Consultas por lotes: Con listas de hasta 26 títulos simultáneos, /api/listar/todo procesó cada solicitud en un tiempo medio de 600 ms, agrupando y devolviendo los datos de OMDb sin pérdida de información relevante.
* Gestión de errores externos: Cuando OMDb no encontró un título, el servicio excluyó ese elemento de la respuesta y mantuvo los restantes, sin generar fallos en la transacción. Esto se tradujo en una tasa de éxito del 95 % en consultas mixtas (títulos válidos e inválidos).
* Escalabilidad y modularidad: El uso de Blueprints en Flask permitió añadir rutas futuras sin impactar el flujo existente. Por ejemplo, la integración en pruebas de userlist\_service pudo realizarse en paralelo, demostrando el desacoplamiento efectivo de los componentes.

User-List

El microservicio userlist\_service fue evaluado en sus funcionalidades de marcado y consulta de estado de películas en la lista del usuario:

* Marcado como vista/no vista: Las rutas /mark\_as\_watched y /mark\_as\_not\_watched respondieron con código 200 en el 100 % de las pruebas con datos completos, actualizando el estado en la base de datos en menos de 220 ms. Ante entradas duplicadas, devolvieron mensajes adecuados indicando si ya se encontraba en el estado solicitado.
* Consulta de películas no vistas: La ruta GET /not\_watched retornó correctamente el listado de títulos no vistos para un user\_id proporcionado. Con listas de hasta 50 películas, el tiempo medio de respuesta fue de 180 ms y la coherencia de los datos coincidió con el estado en la base de datos en el 100 % de las ejecuciones.
* Creación de nuevas entradas: Cuando no existía una entrada previa en UserList, las peticiones de marcado crearon un nuevo registro con el estado correspondiente, demostrando la flexibilidad del modelo para manejar tanto actualizaciones como inserciones en un único endpoint.

Los hallazgos anteriores evidencian que PeliScript cumple con los objetivos de investigación: demostrar la viabilidad de una arquitectura de microservicios para aplicaciones de catálogo de películas y validar la eficiencia en la comunicación cliente-servidor. Los resultados muestran que:

* La separación modular mejora la mantenibilidad y facilita la futura extensión de funcionalidades.
* El frontend en Vue.js, combinado con Axios, garantiza una experiencia de usuario fluida y reactiva.
* Los microservicios en Flask atienden de forma segura y eficiente tanto las necesidades de autenticación como las de consulta externa.

En conjunto, estos resultados aportan una comprensión más profunda de las ventajas y desafíos de los microservicios en entornos de desarrollo de software contemporáneos, alineándose con las perspectivas de López y Maya [5] sobre agilidad y de Hernández Guzmán [2] sobre seguridad.

# DISCUSIÓN Y CONCLUSIÓN

La implementación de PeliScript confirma que una arquitectura de microservicios, combinada con tecnologías modernas como Vue.js y Flask, aporta beneficios tangibles en términos de escalabilidad, mantenibilidad y experiencia de usuario. Los tiempos de respuesta rápidos en el frontend y la capacidad de los servicios de backend para manejar consultas simultáneas demuestran la relevancia de esta aproximación frente a sistemas monolíticos tradicionales. Además, la modularidad facilita la adopción de buenas prácticas de seguridad, como el hashing de contraseñas y la gestión centralizada de tokens JWT, reduciendo la superficie de ataque y mejorando la resiliencia del sistema.

Sin embargo, la investigación también evidenció desafíos que merecen atención en estudios futuros. En particular, la gestión de datos distribuidos plantea la necesidad de estrategias avanzadas de cacheo y sincronización para optimizar el rendimiento bajo cargas elevadas. Asimismo, la orquestación y monitorización de microservicios requiere herramientas especializadas que automatizan el despliegue y el escalado, lo cual no se exploró en este proyecto.

Por tanto, se recomiendan nuevos estudios orientados a:

* Evaluar soluciones de orquestación (por ejemplo, Kubernetes o Docker Swarm) para automatizar la escalabilidad y el balanceo de carga.
* Implementar mecanismos de cacheo distribuido (Redis, Memcached) que reduzcan la latencia y la carga en las APIs externas.
* Analizar la resiliencia mediante pruebas de estrés y caos (Chaos Engineering) para identificar puntos débiles y garantizar la tolerancia a fallos.
* Explorar arquitecturas híbridas que combinen microservicios con funciones serverless para partes específicas del sistema, optimizando costes y eficiencia energética.

Estas recomendaciones proporcionan una hoja de ruta para profundizar en la innovación y mejora de aplicaciones basadas en microservicios, enriqueciendo el estado del arte en el desarrollo de software moderno.

Para visualizar y comprender mejor el Funcionamiento de La Pagina web, visualice el material subido a la red que lo explica: youtube.com

# REFERENCIAS BIBLIOGRÁFICAS

[1] Diego Alfonso Barrios Contreras, «Arquitectura de Microservicios», *Tecnol. Investig. Acad.*, vol. 6, n.o 1, Art. n.o 1, mar. 2018.

[2] K. M. Hernandez Guzman, «Adaptación de un patrón de software en seguridad a la arquitectura de un Microservicio», oct. 2023, Accedido: 18 de febrero de 2025. [En línea]. Disponible en: https://rinacional.tecnm.mx/jspui/handle/TecNM/6574

[3] J. Rienda Iáñez, «Diseño e implementación de un microservicio con Spring», oct. 2019, Accedido: 18 de febrero de 2025. [En línea]. Disponible en: https://hdl.handle.net/10016/30515

[4] S. Ramírez Pérez, «Estudio del framework Spring, Spring Boot y Microservicios», en *ATICA2020: Aplicación de Tecnologías de la Información y Comunicaciones Avanzadas y Accesibilidad, 2020, ISBN 978-84-18254-84-0, págs. 219-226*, Universidad de Alcalá, 2020, pp. 219-226. Accedido: 18 de febrero de 2025. [En línea]. Disponible en: https://dialnet.unirioja.es/servlet/articulo?codigo=8516581

[5] D. López y E. Maya, «Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web».

[6] T. Uotila, «Designing scalable microservices : Case: AWS with Python». Accedido: 10 de marzo de 2025. [En línea]. Disponible en: http://www.theseus.fi/handle/10024/169685

[7] C. A. L. Mamani, «Pruebas de Software para Microservicios», *Innov. Softw.*, vol. 4, n.o 1, Art. n.o 1, mar. 2023, doi: 10.48168/innosoft.s11.a86.

[8] K. T. G. Suárez, R. Anaya, y A. F. Cano, «Un acercamiento a los microservicios», *Unaciencia Rev. Estud. E Investig.*, vol. 10, n.o 19, Art. n.o 19, sep. 2017.

[9] Diego Navarro, Raúl, Cabrera Lozada, Ricardo Daniel, y Huedo Cuesta, Eduardo, «Aplicación basada en arquitectura de microservicios», Aplicación basada en arquitectura de microservicios. Accedido: 10 de marzo de 2025. [En línea]. Disponible en: https://docta.ucm.es/entities/publication/64642e5f-3cc4-48b4-878b-5c8c0abbeefd

[10] Z. M. Rodriguez, «Aplicación cloud native en el contexto de una ingeniería de software continua», *Interfases*, n.o 019, Art. n.o 019, jul. 2024, doi: 10.26439/interfases2024.n19.7038.