TRABAJO DE GRADO
Opción Investigación o Proyecto de Grado

Sistema de Vuelos POC basado en Microservicios

Corporación Universitaria Remington.
Facultad de Ingenierías
Ingeniería en Sistemas

Brayan Pistala Cuaspud
Tutor: Diego Fernando Marín Lozano

Opción de Trabajo de grado Seminario-Diplomado
2025

.

Dedicatoria

Este trabajo está dedicado a mis padres quienes con su ejemplo han forjado en mí un hombre de bien, y me han permitido llegar a la culminación de esta etapa, a mi familia que siempre me ha apoyado y que ha confiado en mí.

Agradecimientos

Agradezco a Dios en primera instancia, a mis padres, a mi familia, a mis tutores que a lo largo de mi vida académica me permitieron crecer como profesional, a la universidad que me ha ayudado a formarme de manera integral no solo como ingeniero sino como una persona que busca aportar con su conocimiento y carácter a su comunidad, a mis compañeros que han sido cómplices de esta maravillosa experiencia del saber.

Tabla de contenido

Lista de figuras

Resumen

El desarrollo del Sistema de Reservas de Vuelos se desarrolló empleando una arquitectura de microservicios, seleccionada por su idoneidad para sistemas en crecimiento. Entre las ventajas que encontramos en esta arquitectura están: la facilidad del mantenimiento, la independencia de sus componentes, permiten una mayor escalabilidad y permiten la entrega de los servicios de manera más continua.

El uso de un lenguaje de programación como Python, en conjunto con Docker, Pytest y MkDocs, proporciona un entorno sólido para el desarrollo, implementación, pruebas y mantenimiento del sistema. En cuanto al manejo de las bases de datos se hace necesario la integración de PostgreSQL y MongoDB, la primera para gestionar datos estructurados (usuarios, horarios, aviones) y la segunda para la gestión de datos no estructurados (registro de transacciones, historial de reservas) lo que mejora sustancialmente la flexibilidad y el rendimiento.

El sistema de Vuelos ofrece grandes ventajas para las aerolíneas debido a que logrará optimizar las operaciones que se ejecutan a diario, reduce costos tanto de operación como de mantenimiento y garantiza que el servicio estará disponible para los usuarios.

El proyecto busca desde la perspectiva académica poder aplicar los conceptos teóricos que se han adquirido a lo largo de la carrera de manera práctica, entender cómo funciona la arquitectura de microservicios, utilizar bases de datos estructuradas y no estructuradas, qué son y para qué sirven las herramientas de contenerización, realización de pruebas, entre otros aspectos, que consolidan las competencias que como estudiantes se han alcanzado.

Palabras clave

Arquitectura, Microservicios, Contenedores, FastAPI, Flask, Scrumban

Pregunta orientadora de la búsqueda

¿Cuál es la razón para adoptar una arquitectura de microservicios desplegada en contenedores, utilizando bases de datos relacionales (PostgreSQL) y no relacionales (MongoDB), dentro de un enfoque DevOps y CI/CD para el desarrollo e implementación del Sistema de Vuelos?

Sustentación teórica de la pregunta

Los sistemas de software en la actualidad, enfrentan un entorno que por la demanda de su uso requiere responder a exigencias de escalabilidad, disponibilidad y agilidad en el desarrollo (Di Francesco, 2019), y son estas exigencias las que han hecho que las organizaciones empiecen a adoptar arquitecturas basadas en microservicios, aunque tradicionalmente muchas aplicaciones empresariales se desarrollaban bajo una arquitectura monolítica, puesto que ofrecía buenos resultados tanto en proyectos pequeños como en implementaciones a gran escala, en esta arquitectura sus diferentes componentes: presentación, lógica de negocio y persistencia, se agrupan dentro de una misma estructura. (Armin Balalaie, 2016)

Sin embargo, se ha podido determinar que esta arquitectura ya no satisface por completo las necesidades actuales del mercado en términos de eficiencia y optimización de recursos tanto humanos como tecnológicos (Kamisetty, 2023). Esta arquitectura tiende a bajar su rendimiento en la medida en que el volumen de datos supera la capacidad prevista (Tapia F. M., 2020).

En este contexto, los microservicios constituyen una alternativa eficaz, ya que permiten separar los componentes mediante interfaces bien definidas y ofrecen ventajas como escalabilidad, resiliencia, bajo acoplamiento, y la capacidad de escalar únicamente los servicios que lo requieren. Sin embargo, también se presentan desafíos relevantes entre

ellos: orquestación de servicios, complejidad operativa, mantenimiento de consistencia de datos y los costos que pueden incrementarse (Parizi, 2018)

Específicamente, en el sistema de reserva de vuelos, la adopción de este tipo de arquitectura ofrece ventajas operativas y estratégicas, las aerolíneas pueden reducir el tiempo de respuesta, mejora la experiencia del cliente y la disponibilidad de sus servicios y reduce los costos de mantenimiento, lo que optimiza la gestión de sus diferentes procesos, la adopción de microservicios ha cambiado la forma en como las empresas interactúan con los usuarios finales, al permitir la modularización de sus componentes, por ejemplo en la implementación de motores de búsqueda de vuelos, pasarelas de pago y sistemas de notificación (Kaiserb, 2018).

En cuanto a las bases de datos se hace necesario combinar bases de datos relacionales para la gestión de datos estructurados como la información de los usuarios, pagos, vuelos, reservas, y no relacionales, para almacenar y procesar datos no estructurados como logs de transacciones, historial de reservas, comentarios de usuarios y preferencias de viaje. Esta arquitectura híbrida de datos combina lo mejor de cada enfoque, por una parte, la consistencia y organización de bases de datos estructuradas y la capacidad de adaptación de las bases de datos no estructuradas (Ying, 2025).

En cuanto al enfoque académico, la implementación del sistema representa una oportunidad de aprendizaje y práctica en arquitectura de software, bases de datos híbridas, pruebas, documentación, gestión de datos, que permiten el desarrollo de competencias que a futuro se utilizaran teniendo en cuenta la demanda del mercado

Desarrollo de APIs y Backend

Cuando hablamos de frameworks y específicamente los que corresponden a Python podemos notar claramente que estos han tenido una evolución considerable: se han convertido en plataformas que son un soporte para potenciar el desarrollo de APIs más

escalables y con mejor rendimiento. Los frameworks surgen por una necesidad que demanda el mercado y son aplicaciones más rápidas, ligeras y modulares, dentro de este conjunto destaca FastAPI, que al utilizar programación asíncrona puede manejar muchas solicitudes de forma simultánea, disminuyendo los tiempos de respuesta, además que facilita la separación de servicios, genera de forma instantánea documentación interactiva mejorando la comunicación entre equipos de desarrollo, en cuanto a rendimiento se ubica entre los más rápidos de Python compitiendo incluso con lenguajes como Node.js.

Es bien conocido que Python se sigue manteniendo como líder en temas de inteligencia artificial y ciencia de datos gracias a la gran cantidad de desarrolladores que lo prefieren, su sintaxis simple, y compatibilidad con librerías como PyTorch, TensorFlow, entre otras, frameworks como FastAPI aprovechan estas ventajas para integrarse de forma eficiente y ágil con los modelos de machine learning y procesamiento de datos, convirtiéndose en una herramienta moderna que optimiza el ciclo de vida del software (Bednarz, 2025).

Arquitectura y Despliegue

La transición de arquitecturas monolíticas a arquitecturas basadas en microservicios ha resultado óptima en pro de mejorar y facilitar el mantenimiento del sistema, aumentar la escalabilidad y acelerar los despliegues de sistemas complejos. Sin embargo, este cambio se debe hacer de forma progresiva, de tal manera que cada microservicio se vea reflejado en un componente funcional e individual, para que cualquier proceso que se lleve a cabo si se pueda hacer de forma independiente (Diogo Faustino, 2024).

Se ha comprobado a través de diferentes prácticas que el uso de microservicios optimiza el consumo de recursos en comparación con la arquitectura monolítica, la utilización de contenedores como Docker permite empaquetar cada microservicio con sus dependencias y su entorno de ejecución, lo que se constituye como una ventaja a la hora de querer actualizar, modificar, implementar o escalar cada servicio según sea necesario. El uso de contenedores entonces ofrece múltiples ventajas, entre las que podemos destacar

su portabilidad, eliminando la problemática de trabajar únicamente en una máquina, los programadores tienen la facilidad de trabajar en una dependencia sin interferir en otra, además, el consumo de recursos se disminuye al ser el contenedor quien se encarga de la gestión y de las aplicaciones que contenga (D. Panji Dirgantara, 2024).

En la arquitectura basada en microservicios, el componente conocido como API Gateway, adquiere una importancia relevante al convertirse en un componente diferenciador respecto al uso que se le da en la arquitectura monolítica. Los API Gateway se encargan de recibir, procesar y enrutar la información hacia los microservicios necesarios, permite la coordinación entre las solicitudes de los clientes y microservicios, redirigiéndolas al servicio que requieren dependiendo de cuál sea su necesidad, en este sitio se llevan a cabo las tareas de autenticación y autorización, actúa como un intermediario estable, permitiendo a los servicios actualizarse de forma individual e incluso intercambiar componentes sin afectar al usuario (Michael Matias, 2024).

Frontend y Experiencia de Usuario

La tendencia actual en las arquitecturas de software nos ha mostrado la necesidad de separar totalmente la interfaz de usuario frontend del backend, implementando un enfoque basado en microservicios y comunicación mediante APIS. Este enfoque permite tener componentes individuales y autónomos que logran acelerar los ciclos de desarrollo, facilitan la escalabilidad y evolución independiente de cada componente, además de que se diferencian claramente las responsabilidades de cada desarrollador. (Canedo., 2025)

De esta forma el componente de frontend se encarga de la experiencia directa con el usuario final, con el objetivo de brindar una interfaz que sea intuitiva, fácil de utilizar, entendible, constituye la interacción directa con el cliente, es todo lo que el usuario ve y con lo que puede interactuar, mientras el backend se encarga de la lógica del negocio, manejo de servicios mediante APIs RESTful, gestión de la persistencia de los datos, conexión con la base de datos y el servidor, esta parte se sitúa del lado del servidor, en

conclusión permite que la aplicación funcione. El mantener separados estos dos componentes brinda varias ventajas entre las que podemos resaltar: la escalabilidad traducida en optimización, actualización más sencilla, despliegue más rápido, modularidad, desarrollo de proyectos complejos en múltiples niveles (Kulkarni, 2024).

Calidad con Pruebas Automatizadas

La realización de pruebas resulta un tema primordial que debe ir acorde al Ciclo de Vida de desarrollo del software, de tal forma que a medida que se desarrolla el sistema se vayan realizando diferentes tipos de pruebas para evitar que al final haya que realizar reestructuraciones críticas al mismo que pueden derivar en mayores costos, baja calidad del sistema e insatisfacción por parte del cliente. Las pruebas son esenciales para garantizar el óptimo funcionamiento y la fiabilidad de los sistemas informáticos (Sánchez Peño, 2015).

En este sentido, se hace necesario desarrollar una "Pirámide de pruebas automatizadas", que se convierten en un apoyo para determinar la cantidad de pruebas que se deben realizar en cada una de las capas, partiendo del precepto: "Mientras más alta la capa, menor la cantidad de pruebas" (Sánchez Peño, 2015). En este punto se realizarán los siguientes tipos de pruebas:

- Pruebas unitarias (unit testing): En este tipo de pruebas se analizarán componentes básicos del proyecto de forma individual, esto con el fin de conocer el correcto funcionamiento del código, son pruebas que se realizan en la fase inicial del proyecto, de no hacerlo así puede incrementar riesgos y costos debido a la complejidad de las modificaciones en etapas posteriores. La capa de pruebas unitarias es la que más pruebas debe tener, porque esto garantiza que la base del proyecto es sólida (Celis Rubiano, 2022).

- Pruebas de integración: Aquí se evalúa la forma en que los diferentes módulos del sistema actúan interactuando entre sí, una vez se han realizado las pruebas unitarias y estas se ejecutan correctamente, es momento de establecer que las conexiones entre los módulos son funcionales. En esta capa es importante establecer puntos esenciales a tener en cuenta para conseguir un proyecto sin errores y acorde a los requerimientos solicitados: crear un plan de pruebas, priorizar los módulos, realizar pruebas en diferentes entornos y evaluar los resultados obtenidos de forma minuciosa (Serna, 2016).

Operaciones y DevOps (CI/CD)

DevOps resulta una práctica esencial en el proceso de desarrollo de aplicaciones ligado estrechamente a la calidad del software, la entrega frecuente de avances resulta en mejoras en el control de versiones, reducción del ciclo de desarrollo y detección temprana de bugs que pueden incrementar costos en etapas posteriores, además de permitir la integración entre el equipo de desarrollo y operaciones. (Manuel Pastrana, 2025).

La automatización de pruebas resulta un aspecto clave en DevOps, esto básicamente es lo que garantiza la entrega continua de los productos, al permitirle a los desarrolladores lanzar cambios de código rápidamente con una alta confianza en la calidad. Entre las principales pruebas que podemos mencionar están: pruebas unitarias, de integración, de extremo a extremo y pruebas exploratorias (Anum Bahar, 2025).

En cuanto a CI (Integración Continua) y CD (Distribución Continua) podemos decir que la primera está enfocada como su nombre lo indica al proceso de automatización que busca la fusión de los cambios del código de una forma sencilla y periódica, garantizando la confiabilidad de los mismos. En este punto, el objetivo es que el equipo de desarrollo pueda trabajar de forma simultánea en distintas funciones del sistema sin que existan conflictos (Eliezio Soares, 2022).

Una vez se ha llevado a cabo el proceso anterior, la distribución continua se encarga de automatizar el proceso de lanzamiento del código validado en un repositorio (como GitHub) después de haber realizado pruebas en el mismo. Al finalizar estos dos procesos, el equipo de operaciones puede implementar por completo la aplicación de una manera ágil y sencilla (Muhammad Owais Khan, 2020).

Persistencia de Datos

Un ORM (Mapeador Objeto – Relacional) actúa como una capa de abstracción entre la estructura de la base de datos y la aplicación. ORM utiliza el lenguaje de backend que el desarrollador esté empleando, convirtiendo las tablas en clases, los registros en objetos y los campos en propiedades. Esto simplifica las tareas para el programador al evitarse la necesidad de construir consultas SQL extensas y complejas.

Entra las ventajas que destacan del uso de ORM están:

- Reducción del código y mejora en su mantenibilidad.
- Estandarización de las operaciones con bases de datos.
- ORM trabaja con una gran variedad de lenguajes de programación.
- Mejora de la seguridad del sistema al evitar inyecciones SQL.
- Va en sintonía con las metodologías ágiles, facilitando el trabajo colaborativo.

Documentación

La automatización de la documentación busca que esta se actualice, versione y alinee en sincronía con el código, lo que incrementa la claridad y minimiza las discrepancias. Este enfoque garantiza que la documentación tenga una estructura y un lenguaje coherentes, se encuentre organizada y disponible para el equipo de desarrollo, y que sean los miembros del equipo quienes puedan gestionarla sin necesidad de un redactor externo.

Problema

Actualmente, el sector aéreo ha experimentado un crecimiento sustancial en el uso de plataformas digitales a través de las cuales los usuarios pueden buscar, reservar, pagar y administrar sus vuelos de manera ágil, segura y confiable. Sin embargo, muchas empresas del sector todavía operan con arquitecturas monolíticas, lo que dificulta la interoperabilidad de datos, dificultando la integración de la información aumentando los tiempos de respuesta, limitando la escalabilidad y vulnerando los datos de los usuarios (Hossam Hassan, 2018).

La Asociación internacional del Transporte Aereo (IATA), informó que el número de pasajeros aéreos a nivel mundial superó los 4.5 mil millones en el año 2024, esto ha hecho que el uso de plataformas digitales también se incremente, por lo que se requiere de tecnologías que soporten esta demanda de personas, que sean seguras y escalables para gestionar reservas y pagos. En contraste, todavía existen aerolíneas que siguen utilizando sistemas monolíticos que enfrentan tiempos más lentos de respuesta, mayores costos de mantenimiento, inconvenientes al realizar búsquedas de vuelos, confirmar reservas, o efectuar pagos en línea, sobre todo cuando el incremento de personas es relevante (Kumar, 2021).

La ausencia de arquitecturas basadas en microservicios genera múltiples limitaciones operativas dentro de los sistemas, entre ellos podemos listar los siguientes: Al existir un único bloque funcional aumenta la latencia en la transmisión y procesamiento de información, duplicidad de datos y una falta de división de responsabilidades entre los diferentes módulos. Esta arquitectura impide el escalamiento de los componentes de forma modular, lo que afecta de forma directa la disponibilidad y el rendimiento del sistema (Tapia F. M., 2020).

Es por esta razón, que se ha decidido que el Sistema de Reservas de Vuelos se base en una arquitectura basada en microservicios, utilizando bases de datos híbridas (PostgresSQL

y MongoDB), esto permite optimizar la gestión de información, incrementar la escalabilidad, reforzar la seguridad de las transacciones y al usuario final le ofrece una experiencia satisfactoria en términos de seguridad, uso de interfaces, rapidez y eficiencia. Se ha confirmado que la combinación de estas tecnologías mejora la disponibilidad y reducen hasta un 30% los tiempos de respuesta en sistemas de reserva y gestión de pagos (Goze, 2024).

Metodología

La metodología utilizada para la realización del proyecto se basa en Scrumban como una combinación de dos metodologías que resultan ser muy efectivas en el desarrollo ágil de aplicaciones (Scrum y Kanban), sobre todo porque permiten enfrentar desafíos como la gestión del tiempo, el control del flujo del trabajo y la entrega del producto final en los tiempos solicitados (Alqudah, 2018).

Scrumban busca disminuir el uso excesivo de historias de usuarios limitándose a aquellas que realmente se ajustan a las necesidades del proyecto y del equipo, lo que hace que el flujo de trabajo sea continuo (Castaño, 2016).

En cuanto a las prácticas de Kanban el tablero sigue siendo una parte fundamental en el desarrollo normal del trabajo, debido a que la visualización de las diferentes tareas que se requieren para completar el proyecto resulta ser crucial para el dinamismo del mismo. Conocer quiénes son los responsables de las actividades y en qué punto estas se encuentran permite hacer un seguimiento al equipo de tal forma que no haya retrasos significativos que pueden derivar en el fracaso del proyecto (Castillo-Nuñez, 2023).

Las columnas en el tablero representan las etapas del flujo de trabajo que va desde el listado de las actividades hasta la finalización de cada una de estas, se mantiene un límite de las actividades que se pueden realizar al mismo tiempo (WIP) (Damij., 2021), esta característica garantiza que se respete la cantidad de tareas empleadas en cada etapa según el rendimiento del equipo, debajo de cada etapa se especifica claramente los recursos con los que se cuenta, las operaciones que se van a hacer y cuando se puede considerar que una tarea está Hecha, para evitar confusiones.

Scrumban trabaja en coordinación con la planificación del Backlog, priorizando tareas para cada iteración, esto permite que el trabajo sea más fluido y se puedan evacuar tareas de una manera más ágil. Es importante que este proceso se realice diariamente para que

realmente el trabajo sea continuo Scrumban trabaja en coordinación con la planificación del Backlog, priorizando tareas para cada iteración, esto permite que el trabajo sea más fluido y se puedan evacuar tareas de una manera más ágil. Es importante que este proceso se realice diariamente para que realmente el trabajo sea continuo (Krunal Bhavsar, 2020).

Esta metodología, además, permite identificar debilidades y oportunidades de mejora, enfocado siempre en la mejora continua.

Los artefactos esenciales en la metodología Scrumban se listan a continuación:

- Product Backlog: Se traduce en las necesidades del cliente con respecto al proyecto, cada una se convierte en una tarea representada en historia de usuario, aquí es importante determinar cuáles son las tareas prioritarias.
- Sprint Backlog: Se especifica en detalle lo que debe cumplir cada tarea, además de tener en cuenta el tiempo estimado que tomará el desarrollo de cada actividad.
- Tablero Kanban: Puede ser físico o digital, lo que mejor se ajuste al equipo de trabajo, lo importante en este artefacto es que estén relacionadas todas las tareas que se deben desarrollar empezando desde la columna Por Hacer (listado de las actividades), en Proceso (actividades que ya se han empezado a desarrollar) y Terminado (actividades que se han completado a satisfacción) (Aysha Abdullah Albarqi, 2018).

En cuanto a la documentación del proyecto esta se trabaja como una tarea más y no como un proceso paralelo, puesto que se entiende la importancia de documentar las diferentes etapas del proyecto como un soporte para el equipo de desarrollo. Es importante que se vayan registrando los avances del proyecto y que no sea una tarea para desarrollar al final, puesto que pueden quedar registros importantes por fuera (Hasan., 2023).

Desarrollo

Sprint 1

El Sistema de Vuelos POC, es una aplicación desarrollada en Python, basada en la arquitectura de microservicios y en conjunto con Docker, integrando bases de datos relacionales (PostgresSQL, Puerto 5432) para auth, flights y payments y no relacionales (MongoDB, Puerto 27017) para bookings.

Se establecen cuatro microservicios: Auth_service – Autenticación y gestión de usuarios (Puerto 8001), Flights Service – Gestión de vuelos y disponibilidad (Puerto 8002), Bookings Service - Reservas y tiquetes electrónicos (Puerto 8003) y Payment Services – Procesamiento de Pagos y Facturación – Procesamiento de pagos y facturación (Puerto 8004).

En la Ilustración 1 se observan salidas de Log, donde se incluyen los puertos, el método, el token y el código de respuesta que indica que el microservicio está funcionando correctamente.

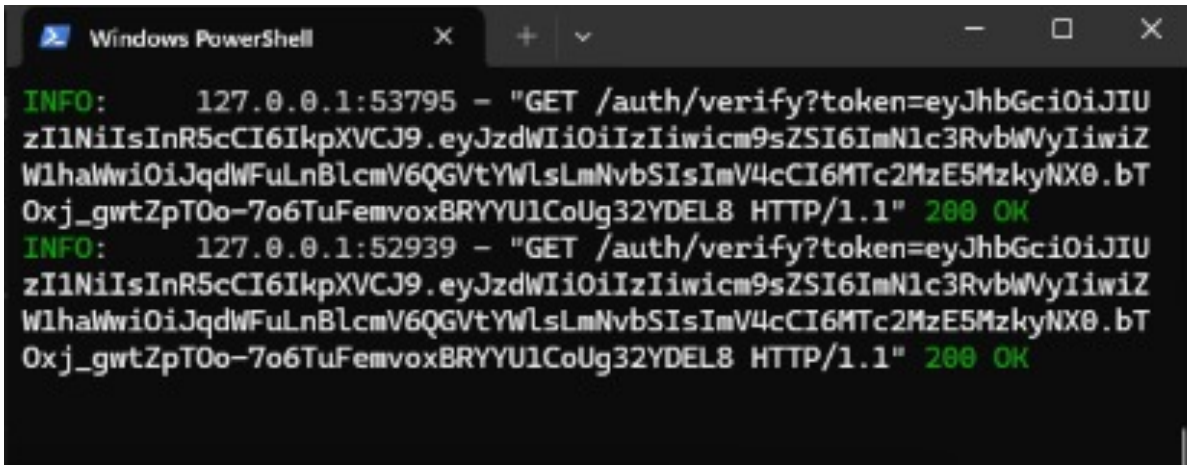

Ilustración 1.Microservicios

Se observan los logs generados (Ilustración 2) por el microservicio booking, se muestran las solicitudes que el servicio está recibiendo, los errores se muestran en color rojo relacionados con los parámetros de envío y el código 200 que indica que el endpoint funciona correctamente.

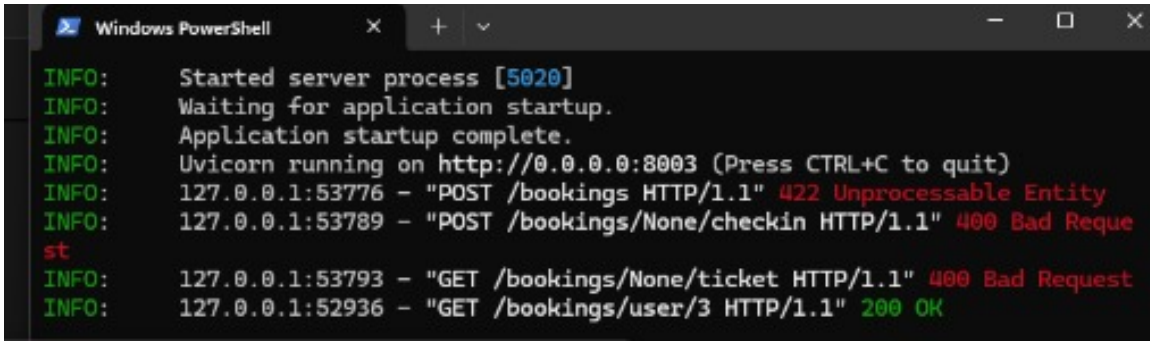

Ilustración 2. Microservicios

La Ilustración 3 detalla las pruebas realizadas a los microservicios en los procesos Creando vuelo (como Aerolínea) donde se observa un error debido a un fallo del servidor, Buscando Vuelos que responde correctamente, Creando Reservas y Procesando Pagos que indica Status 422 lo que evidencia que el servidor no puede procesar la solicitud.

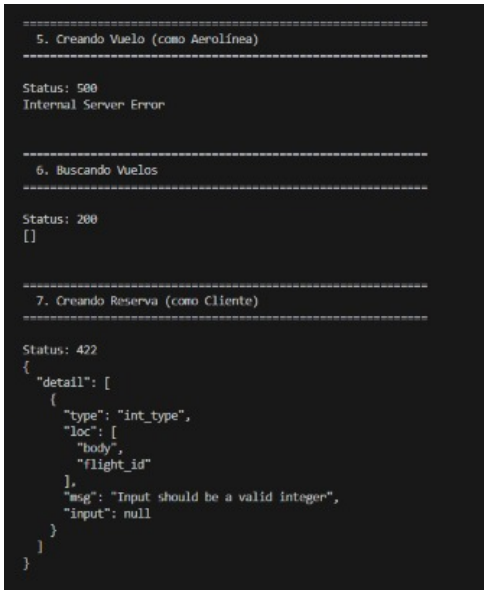

Ilustración 3. Microservicios

Sprint 2

En la Ilustración 4 se observa la tabla de usuarios y la tabla de vuelos con sus respectivos campos. Y en la Ilustración 5 se observan la tabla de pagos y los índices.

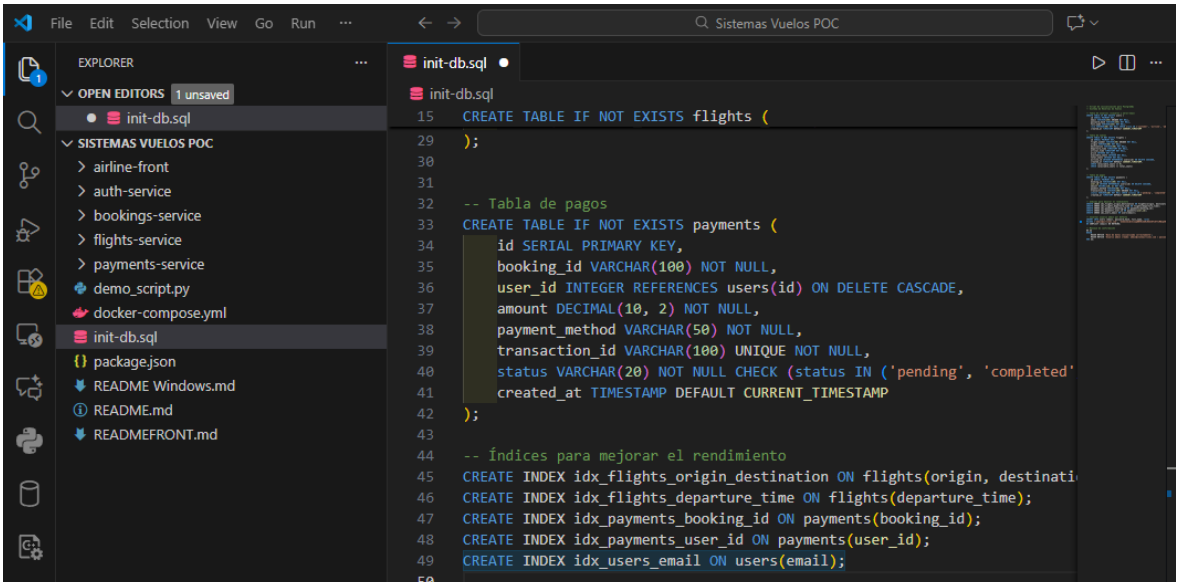

Ilustración 4. Base de Datos

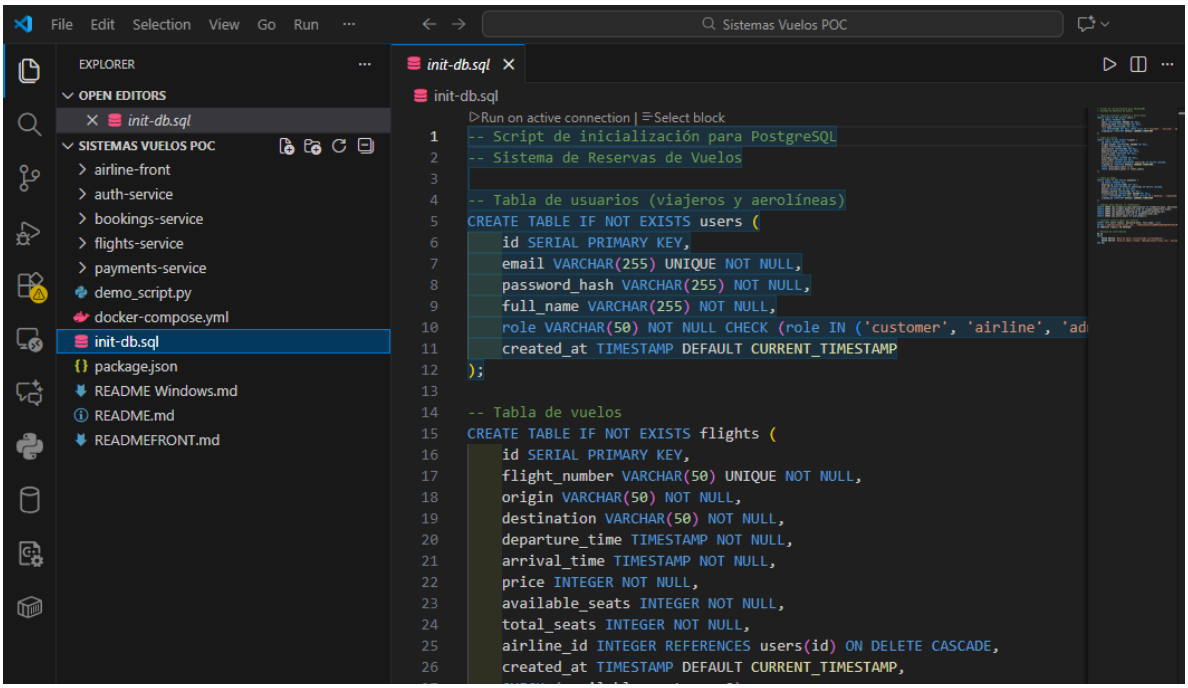

Ilustración 5. Base de Datos

La Ilustración 6 muestra la configuración de las bases de datos tanto relacionales (PostgreSQL) como no relacionales (MongoDB).

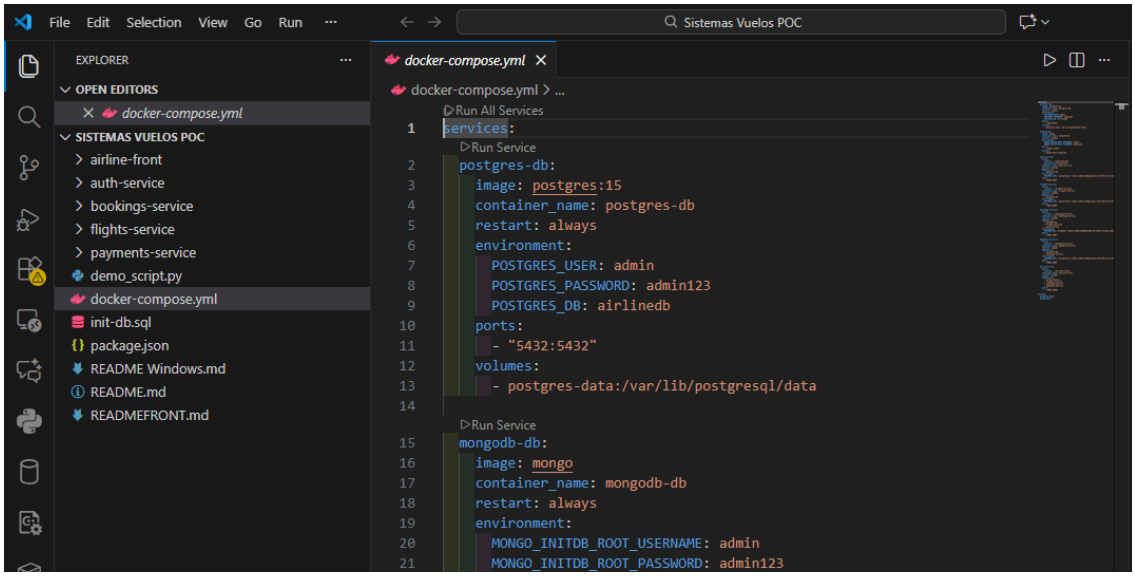

Ilustración 6. Docker-compose.yml

Ejecución de demo_script.py (Ilustración 7): lista el resultado de pruebas realizadas sobre los microservicios. Se observan los procesos de reservas y el Historial de Pagos del Usuario y al final un mensaje que muestra que todos los microservicios se ejecutaron correctamente.

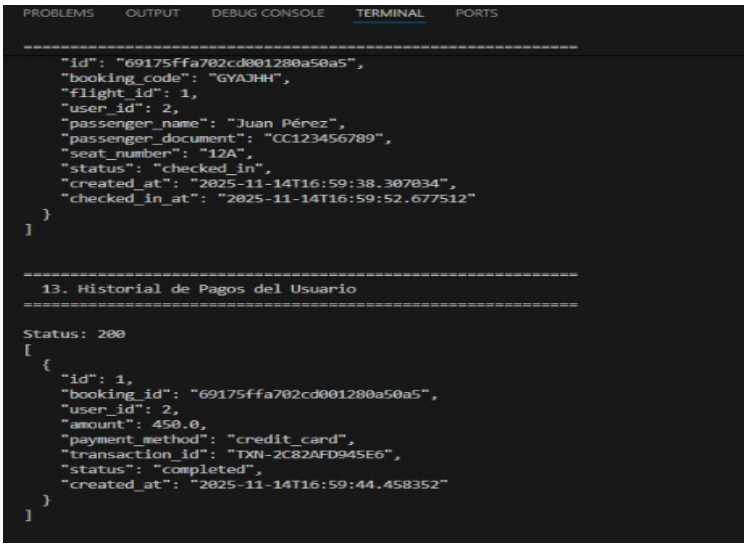

Ilustración 7. Ejecución de demo_script.py

Sprint 3

Interfaces de la aplicación:

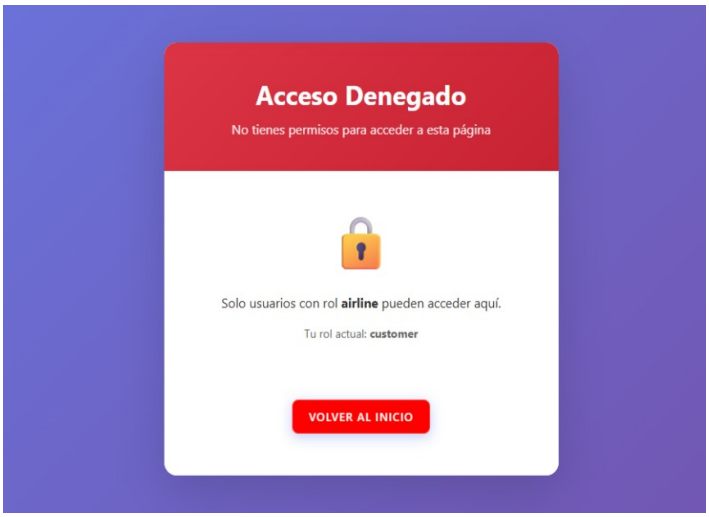

Ilustración 8. Acceso Denegado

El sistema restringe el acceso a usuarios no autorizados (Ilustración 8). Solo los usuarios con rol Airline pueden acceder al sistema. En la parte superior de la interfaz se encuentran las opciones Buscar vuelos, Mis Reservas, Mis Pagos y Cerrar Sesión.

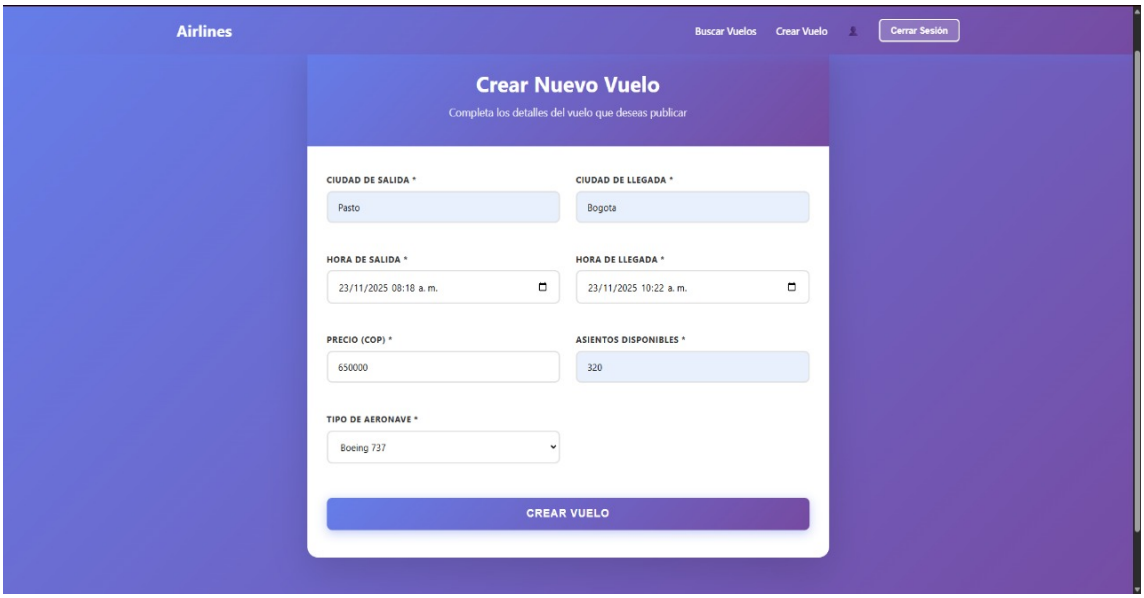

Ilustración 9. Creación de vuelos

Esta interfaz (Ilustración 9) permite ingresar ciudad de salida, ciudad de llegada, hora de salida, hora de llegada, precios, asientos disponibles y tipo de aeronave. En la parte superior de la interfaz se visualiza un menú horizontal donde se encuentran las siguientes opciones: Buscar vuelos, crear vuelos y cerrar sesión.

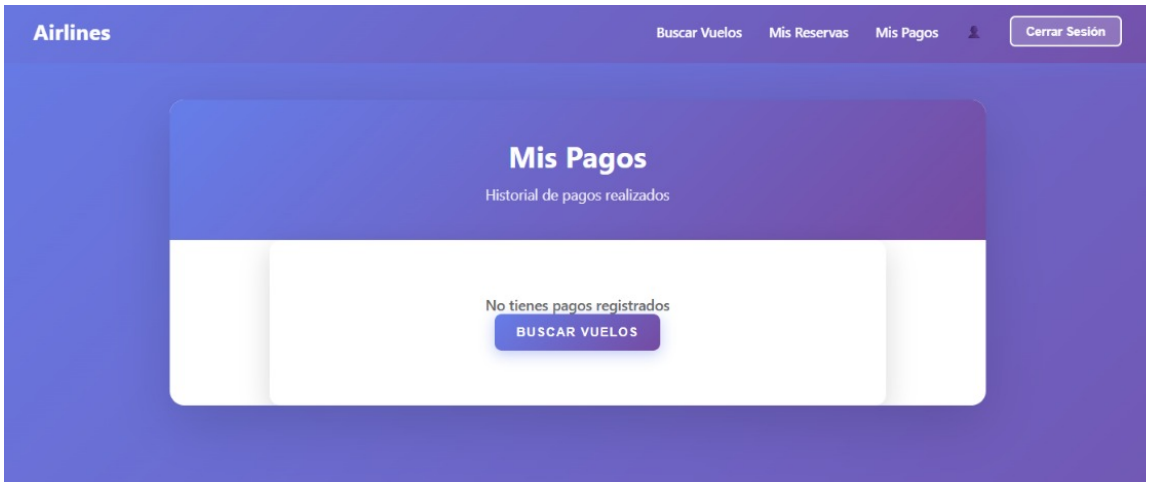

Ilustración 10. Interfaz de pagos

Esta interfaz (Ilustración 10) lista los pagos realizados por el usuario y permite buscar vuelos disponibles.

Ilustración 11. Interfaz Buscar Vuelos

Esta interfaz (Ilustración 11) permite buscar vuelos ingresando la ciudad de salida, la ciudad de llegada y la fecha de vuelo.

Esta interfaz (Ilustración 12) lista los vuelos encontrados teniendo en cuenta los parámetros solicitados en la Ilustración 11.

Ilustración 12. Interfaz de lista de vuelos

Se almacenó el código en un repositorio de GitHub (Ilustración 13), principalmente para utilizar el control de versiones que este ofrece de tal manera que sea posible identificar los cambios que se realizan en el código y resulte más sencillo su mantenimiento.

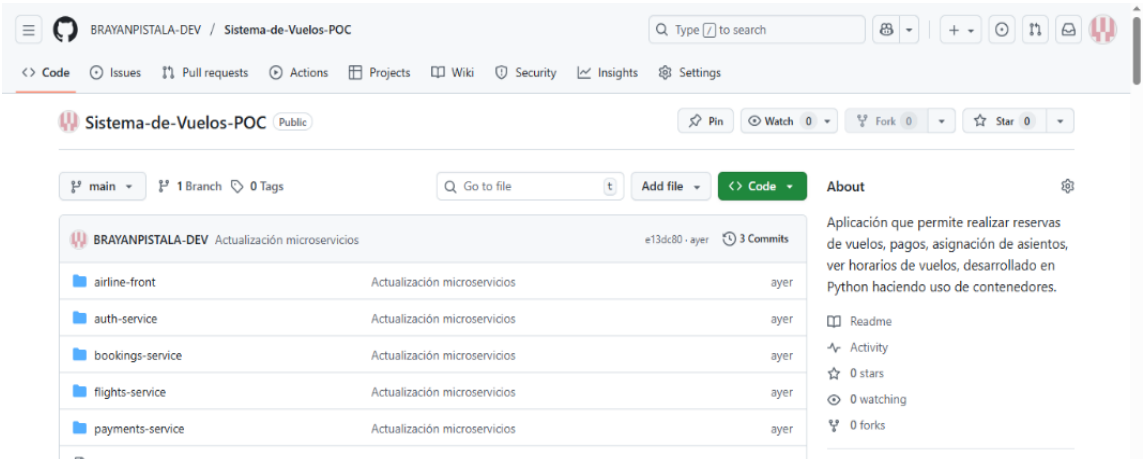

Ilustración 13. Repositorio de GitHub

Conclusiones

- El uso de microservicios se enfoca en servicios que trabajan de forma independiente, de tal manera que si uno falla los otros pueden seguir funcionando sin verse afectados, lo que agiliza el desarrollo y facilita la optimización de las aplicaciones actuales. Esta arquitectura está dirigida a entornos escalables y flexibles.

- El control de versiones de los microservicios es una herramienta esencial para los desarrolladores, para mantener de forma automática actualizado el código y permitir el trabajo colaborativo entre los integrantes del equipo de desarrollo, lo que facilita la gestión del código sin que se interrumpan ningún servicio, se hace un seguimiento individual de todos los cambios que hayan ocurrido evitando que el trabajo simultáneo entre en conflicto.

- Los contenedores como un nuevo nivel de virtualización se integran perfectamente con la arquitectura de los microservicios, donde cada uno de los servicios se empaquetan con sus propias bibliotecas, enlaces, configuración y su entorno de ejecución, lo que los hace eficientes en términos de uso de recursos, tiempo de arranque y portabilidad.

- ORM facilita el desarrollo ágil de las aplicaciones, convirtiéndose en un puente entre la estructura de la base de datos y la estructura de la aplicación, se encarga de realizar tareas repetitivas como la gestión de consultas y transacciones, lo que permite que los desarrolladores aumenten su enfoque en la lógica del negocio.

Bibliografía

Alqudah, R. (2018). Un estudio empírico de la formación de Scrumban basado en la selección de prácticas Scrum y Kanban.

Armin Balalaie, A. H. (2016). La arquitectura de microservicios habilita DevOps: un informe de experiencia sobre la migración a una arquitectura nativa de la nube.

Aysha Abdullah Albarqi, R. Q. (2018). The Proposed L-Scrumban Methodology to Improve the Efficiency of Agile Software Development.

Bednarz, B. y. (2025). Evaluación comparativa del rendimiento de frameworks web de Python. Journal of Computer Sciences Institute .

Castaño, J. M. (2016). Proyecto de investigación para optar al título de Magister en ingeniería con especialidad en Teleinformática. Universidad EAFIT.

Castillo-Nuñez, J. P. (2023). Innovación en la Gestión Visual: Prácticas clave en Kanban digital . Pädi Boletín Científico De Ciencias Básicas E Ingenierías Del ICBI, 11(Especial3), 50–56. https://doi.org/10.29057/icbi.v11iEspecial3.11336.

Celis Rubiano, E. F. (2022). Mitigación de costos de defectos en el proceso de pruebas de calidad de software.

D. Panji Dirgantara, D. S. (2024). “COMPARACIÓN DEL RENDIMIENTO DE ARQUITECTURA MONOLÍTICA Y DE MICROSERVICIOS BASADA EN DOCKER”.

Damij., N. (2021). Un enfoque para optimizar el flujo de trabajo del tablero Kanban y acortar el plan de gestión de proyectos.

Di Francesco, P. L. (2019). Architecting with microservices: A systematic mapping study . *Journal of Systems and Software*.

Diogo Faustino, N. G. (2024). Stepwise migration of a monolith to a microservice architecture: Performance and migration effort evaluation, Performance Evaluation.

Goze, I. E. (2024). Arquitectura híbrida de gestión de datos con microservicios que integra bases de datos SQL y NoSQL para empresas de gestión de activos.

Hasan., M. A. (2023). Prácticas de documentación en el desarrollo ágil de software: una revisión sistemática de la literatura.

Hossam Hassan, M. A.-F. (2018). Migración de una arquitectura monolítica a una arquitectura de microserviciosArquitecturas: Una revisión sistemática de la literatura .

Idris, N. M. (2021). Revisión general de la tecnología web: Django y Flask. International Journal of Advanced Science Computing and Engineering.

Kaiserb, B. B. (2018). Enhancing Resilience and Scalability in Travel Booking Systems: A Microservices Approach to Fault Tolerance, Load Balancing, and Service Discovery.

Kamisetty, A. N. (2023). (2Microservicios vs. Monolitos: Análisis comparativo para el diseño de arquitecturas de software escalables. Engineering International. https://doi.org/10.1803. 99-112.

Krunal Bhavsar, V. S. (2020). Scrumban: An Agile Integration of Scrum and Kanban in Software Engineering.

Kulkarni, K. G. (2024). Integración de microservicios y microfrontends: una revisión bibliográfica exhaustiva sobre arquitectura, patrones de diseño y desafíos de implementación. . *Journal of Scientific Research and Technology*.

Kumar, H. (2021). Microservices Architecture for Streamlining Airline Management Systems.

Microservices as an Evolutionary Architecture of Component-Based Development: A Think-aloud Study Reza M. Parizi. (s.f.).

Parizi, R. M. (2018). (Microservices as an Evolutionary Architecture of Component-Based Development: A Think-aloud Study Reza M. Parizi) .

Sánchez Peño, J. M. (2015). Pruebas de Software. Fundamentos y Técnicas.

Serna, E. (2016). Metodología orientada al Proceso de Testing Funcional en la evaluación de productos de software. . *Revista [SciELO]*.

Tapia, F. M. (2020). De sistemas monolíticos a microservicios: un estudio comparativo del rendimiento. Applied Sciences.

Tapia, F. M. (2020). De sistemas monolíticos a microservicios: un estudio comparativo del rendimiento. Applied Sciences. https://doi.org/10.3390/app10175797.

Ying, C. S. (2025). Análisis comparativo de modelos de bases de datos relacionales y no relacionales: un estudio de caso sobre sistemas de reservas de viajes. .