Microservicios: Arquitectura para el Desarrollo de Software Escalable

Microservices: Architecture for Scalable Software Development

Microservicios: Arquitectura para el Desarrollo de Software Escalable

**Carlos Eduardo Guzmán Torres**

cedguzman@poligran.edu.co

**Dana Yuredt Castro Cuevas**

dyucastro@poligran.edu.co

Politécnico Grancolombiano, Facultad de Ingeniería, Diseño e Innovación

Colombia

*Carlos Eduardo Guzmán Torres y Dana Yuredt Castro Cuevas son estudiantes de Ingeniería de Sistemas en el Politécnico Grancolombiano. Actualmente, se estudian la asignatura de Ingeniería de Software II, donde trabajan en proyectos relacionados con la arquitectura de software, desarrollo de aplicaciones escalables y metodologías ágiles. Sus intereses académicos incluyen la computación en la nube, los microservicios y la inteligencia artificial aplicada al desarrollo de software.*

*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

La evolución de la ingeniería de software ha dado lugar a múltiples paradigmas y metodologías para el desarrollo eficiente de aplicaciones. Entre estas tendencias, la arquitectura de Microservicios ha emergido como una solución viable para la escalabilidad y mantenibilidad de sistemas complejos.

Este artículo presenta una revisión sistemática de literatura sobre la arquitectura de Microservicios, con el propósito de consolidar el conocimiento teórico y proporcionar una base para la implementación práctica de una aplicación que refleje los principios y ventajas de esta tecnología. La investigación abarca la identificación de antecedentes, estudios previos y aplicaciones recientes en la industria.

El desarrollo de la aplicación requerirá un entorno específico que incluye herramientas de desarrollo integrado (IDE’s), sistemas de gestión de versiones, plataformas de pruebas y un marco metodológico para el seguimiento del proyecto.

## DEFINICIÓN DEL AMBIENTE DE DESARROLLO

### Herramientas y Tecnologías

Para el desarrollo de la aplicación basada en microservicios, se utilizará el siguiente entorno de desarrollo:

* **IDE:** Eclipse con Spring Boot, que facilita la creación de microservicios con una estructura modular y eficiente. Estas herramientas proporcionan un marco sólido para el desarrollo de microservicios, facilitando la integración de componentes y la gestión de dependencias.
* **Sistema de Control de Versiones:** Git y GitHub, permitiendo una gestión eficiente del código y la colaboración en equipo. GitHub permite una gestión eficiente del código fuente, garantizando la trazabilidad y facilitando la colaboración entre los integrantes del equipo.
* **Gestión de Proyectos:** Notion, que proporciona una plataforma flexible para la planificación, seguimiento y documentación del desarrollo del software, ayudando a la planificación y seguimiento de tareas de manera estructurada.
* **Plataforma de Pruebas:** Postman, que permitirá la validación y pruebas de las APIs desarrolladas dentro del entorno de microservicios. Esto asegura la correcta comunicación entre los microservicios y la estabilidad del sistema

# REVISIÓN LITERARIA

## ORIGEN Y EVOLUCIÓN DE LA ARQUITECTURA DE MICROSERVICIOS

Es un enfoque para el desarrollo de una aplicación única como un conjunto de pequeños servicios, cada uno ejecutándose en su propio proceso y mecanismos ligeros de comunicación, a menudo un recurso de una interfaz de programación de aplicaciones (API) sobre protocolo de transferencia de hipertexto (HTTP). Estos servicios están construidos alrededor de las capacidades del negocio y con independencia de despliegue e implementación totalmente automatizada. Existe un mínimo de gestión centralizada de estos servicios, los que pueden estar escritos en lenguajes de programación diferentes y utilizar diferentes tecnologías de almacenamiento de datos.

El término microservicios no es relativamente nuevo, este estilo arquitectural fue acuñado por Martin Fowler en un taller de arquitectos de software como una descripción del nuevo campo que los participantes estaban explorando. No existe una definición en concreto para microservicio, sin embargo, una aproximación que la realiza [5] lo define como: “Pequeños servicios autónomos que trabajan juntos”. Una arquitectura de microservicios promueve el desarrollo y despliegue de aplicaciones compuestas por unidades independientes, autónomas, modulares y auto- contenidas, lo cual difiere de la forma tradicional o monolítico.

López, D., & Maya, E. (2017). Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web.

La primera pregunta es de dónde nace el término microservicio; pues bien, de acuerdo con Martin Fowler (Ingeniero de sistemas británico, autor de múltiples publicaciones, y orador público en diseño de software empresarial) el término microservice (microservicio en español) fue discutido en un taller para arquitectos de software cerca de la ciudad de Venecia en mayo del 2011, se utilizó para describir un estilo arquitectural que muchos habían empezado a explorar recientemente. Más tarde, en mayo de 2012, este mismo grupo decide optar por el nombre de microservicios como el término más apropiado para describir sus experiencias [1]. Algo que se debe tener en cuenta es que aunque el término puede ser relativamente nuevo — especialmente a partir del 2014 cuando se convirtió en uno de los términos más populares, atrayendo mucha atención, como una nueva manera de pensar en la estructuración de las aplicaciones—, su idea base ha existido por mucho tiempo; adopta, aparentemente, los mismos ideales de SOA (Service Oriented Architecture), tanto que para muchos no es fácil ver una diferencia real.

Sin embargo, es importante recalcar que durante los últimos años el término ha sido utilizado para describir un modo particular para diseñar y estructurar software como un conjunto de servicios desplegables independientemente. Y aunque a pesar de su auge no existe una definición precisa de este estilo arquitectural, sí cuenta con una serie de características que giran alrededor de la organización, su lógica del negocio, despliegues automáticos, control descentralizado del lenguaje — de programación— y datos [2]

Contreras, D. A. B. (2018). Arquitectura de microservicios. *Tecnología Investigación y Academia*, *6*(1), 36-46.

## FUNDAMENTOS DE LA ARQUITECTURA DE MICROSERVICIOS

Una arquitectura de microservicios promueve el desarrollo y despliegue de aplicaciones compuestas por unidades independientes, autónomas, modulares y autocontenidas, lo cual difiere de la forma tradicional o monolítico

López, D., & Maya, E. (2017). Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web.}

El software construido en base a microservicios se podrá descomponer en varias partes funcionales independientes. Siendo así, cada uno de estos servicios podrá ser desplegado, modificado y re-desplegado sin comprometer los otros aspectos funcionales de la aplicación; y como resultado en caso de necesitarlo, sólo tendremos que modificar un par de servicios en lugar de redesplegar toda la aplicación al completo nuevamente.

La forma en la que se organizan los microservicios suele ser en torno a las necesidades, capacidades y prioridades del cliente o negocio en el que se implantará. A diferencia de un entorno monolítico donde cada equipo de trabajo tiene un enfoque específico sobre un apartado de la aplicación, en la arquitectura de microservicios se utilizan módulos multifuncionales, adaptando así un módulo común a todos para que ofrezca un servicio determinado. El ahorro en tiempo de desarrollo es inmenso, por no hablar de la comodidad a la hora de programar tareas de mantenimiento, donde podemos revisar un módulo mientras el resto del equipo de trabajo no ve interrumpida su jornada.

El funcionamiento del software de microservicios puede parecerse al sistema de trabajo clásico de UNIX (se recibe una petición, se procesa y se genera una respuesta consecuente. Al contrario que los entornos ESB (Enterprise Service Buses, o Bus de servicio empresarial) donde se utilizan equipos para enrutar mensajes, redireccionar tráfico, aplicar reglas de denegación de acceso etc.… Se podría decir que la arquitectura de microservicios cuenta con puntos finales “inteligentes” que procesan la información en término y aplican la lógica establecida por el desarrollador.

La arquitectura de microservicios mantiene un sistema similar a un gobierno descentralizado, donde cada módulo contará por ejemplo con su propia base de datos, en lugar de acudir todos a la misma sobrecargándola así de solicitudes y arriesgándonos a que si falla ésta, todas las aplicaciones caigan.

Cuando varios servicios están comunicados entre sí, por lo general contarán con un sistema de aviso y actuación si alguno de éstos servicios llega a fallar (como mostrar una advertencia, enviar un mail a soporte, avisar a los usuarios de un fallo temporal, etc.), filtrando adecuadamente la información destinada a este módulo y favoreciendo la correcta gestión de los recursos entre los módulos funcionales restantes.

Rendón, J. MICROSERVICIOS.

## PATRONES Y BUENAS PRÁCTICAS EN MICROSERVICIOS

Utilizando un API Gateway En arquitecturas basadas en microservicios cada microservicio expone una serie de endpoints. Los clientes que se conectan a nuestros servicios (ya sean aplicaciones móviles o aplicaciones web ejecutándose en el navegador de nuestros usuarios) pueden acceder directamente a ellos, por lo que estarían expuestos directamente a internet y deberían tener una dirección pública. En entornos de producción y en la nube la dirección pública que exponen podría ser un balanceador que distribuya la carga entre los microservicios.

La comunicación directa entre clientes y microservicios está bien cuando hablamos de una aplicación pequeña, sin embargo, cuando la aplicación es compleja este enfoque presenta una serie de problemas. Si necesitamos realizar múltiples llamadas a distintos microservicios para cargar una página o pantalla en una aplicación móvil incrementaremos el número de peticiones aumentando la latencia y la complejidad en la parte del cliente.

En este punto podríamos introducir un sistema intermediario Ilamado API Gateway que proporcione una interfaz API REST para hacer de enrutador desde un único punto de entrada hacia un grupo de microservicios y/o API’s de terceros.

Este API Gateway permitirá unificar o desacoplar la interfaz que ven los clientes (en este caso, los consumidores de la API que podrían ser aplicaciones móviles o web) de la propia implementación de los microservicios. El objetivo sería de hacer de proxy optimizado para la autenticación y el control de acceso a los diferentes microservicios y/o APIs de terceros.

Este patrón es especialmente útil para evitar exponer los servicios internos a clientes externos, separando las API públicas externas de las API internas de los microservicios. También permite ocultar el descubrimiento de los servicios publicados en la API’s principal y los detalles de las diferentes versiones que puedan tener los microservicios, ya que proporciona un único punto de entrada para todos los microservicios y API's.

Este servicio nos expone un único punto de entrada para un grupo de microservicios. Este servicio podría implementarse como una Web API ejecutándose en un contenedor o podríamos usar otros servicios que ofrezcan los distintos proveedores Cloud donde estemos desplegando nuestros microservicios. El API Gateway permitiría añadir funcionalidades adicionales como:

* Autenticación y autorización (por ejemplo, OAuth).
* Protección contra amenazas (por ejemplo, DoS, inyección de código, etc.).
* Análisis y supervisión (quién utiliza sus API's, cuándo y cómo).
* Monitorización de uso (KPI's y métricas).
* Administración del ciclo de vida y versionado de las APIs.
* Orquestación de API propias o de terceros.

Candel, J. M. O. (2020). *Tecnologías para arquitecturas basadas en microservicios: Patrones y soluciones para aplicaciones desplegadas en contenedores*. José Manuel Ortega.

## TECNOLOGÍAS Y HERRAMIENTAS PARA IMPLEMENTAR MICROSERVICIOS

Ejecutar microservicios no es complicado, porque tenemos muchas tecnologías disponibles para ayudar a su implementación, como Spring Boot, Django, Lumen framework, todas ellas pueden ayudar a solventar los problemas relacionados con su construcción. Desplegar este tipo de arquitecturas no es muy complejo, ya que actualmente existen softwares que ayudan con el proceso como son PodMan, Docker, Kubernetes, entre otros. En este punto se obtienen problemas con arquitectura de microservicios como son la de poder controlar todos los microservicios, las comunicaciones y la trazabilidad. Para solventar este problema se implementa una malla de servicios (service mesh) que es una infraestructura encargada de controlar toda la comunicación y seguridad de los microservicios.

De esta manera se reduce la complejidad que se tiene al implementar este tipo de arquitectura. Para desarrollar la malla de servicios, se utiliza Istio, que es una plataforma de código abierto la cual nos permite gestionar la comunicación, la seguridad y los indicadores de los microservicios.

Diego Navarro, R., & Cabrera Lozada, R. D. (2020). Aplicación basada en arquitectura de microservicios.

Algunas de las herramientas más usadas para la creación de una arquitectura de microservicios son:

* Amazon Simple Queue Service (SQS)
* Apache Kafka
* Apache Kafka
* Kubernetes
* Spring Boot
* Google Cloud Functions
* AWS Lambda
* Microsoft Azure Functions

Rendón, J. MICROSERVICIOS.

## MICROSERVICIOS Y ESCALABILIDAD DEL SOFTWARE

Los microservicios se conciben como un estilo arquitectónico enfocado en desarrollar una aplicación mediante un conjunto de servicios, independientes, escalables, colaborativos, evolutivos, capaces de autoadaptarse a ecosistemas complejos. (Richardson, 2019; Bandeira y otros, 2019).

Rodríguez, Z. M., Rodríguez, L. D. P., & Suarez, J. C. G. (2020). Arquitectura basada en Microservicios y DevOps para una ingeniería de software continua. *Industrial Data*, *23*(2), 141-149.

La automatización de la infraestructura usando prácticas de DevOps reducen el esfuerzo manual involucrado en la construcción, implementación y operación de microservicios, lo que facilita la entrega y despliegue continuo. El gobierno descentralizado y la gestión de datos permiten que los servicios sean independientes y evitan que una aplicación estandarice una sola tecnología. Las arquitecturas de microservicio son especialmente adecuadas para las infraestructuras en la nube, ya que se benefician enormemente de la elasticidad y el rápido aprovisionamiento de recursos [5]

Vera-Rivera, F. H. (2018). Método de automatización del despliegue continuo en la nube para la implementación de microservicios. In *Proc. 21st Conf. Iberoamericana Ingeniería Softw.(CIBSE)* (pp. 597-604).

La escalabilidad en microservicios se concibe desde tres dimensiones: por duplicación horizontal de recursos de máquinas (Xaxis), por particionamiento de datos (eje Z), y por descomposición funcional (eje Y). Alineamiento organizacional: Es habilitado por la organización de los microservicios alrededor de las capacidades del negocio, y motiva a 122 UNACIENCIA. Revista de Estudios e Investigaciones los equipos más pequeños a componentes como unidades pequeñas, lo cual facilita la mantenibilidad. Incremento en la agilidad: El hecho de tener servicios escalables, adaptables, modulares y rápidamente accesibles hace que los microservicios favorezcan la evolución del software y así dar una mejor respuesta a las demandas de los clientes en entornos cambiantes. El esfuerzo del desarrollo es realizado alrededor de múltiples equipos y cada equipo es responsable por uno o más servicios simples y puede desarrollar, desplegar y escalar sus servicios independientemente de todos los otros equipos [8]

Suárez, K. T. G., Anaya, R., & Cano, A. F. (2017). Un acercamiento a los microservicios. *Unaciencia Revista de Estudios e Investigaciones*, *10*(19), 116-126.

## RETOS Y TENDENCIAS ACTUALES

La descomposición de aplicaciones en microservicios incrementa la complejidad de integración, por lo que se opta por usar comunicación asíncrona basada en eventos para desacoplar interfaces de integración entre los servicios. Un desafío al que se debe enfrentar los que desarrollan microservicios es identificar correctamente cuales son los límites que debe tener cada microservicio, más que todo cuando estos microservicios exponen sus interfaces a otros microservicios, ya que cualquier cambio en la interfaz puede afectar a otros microservicios. Otro desafío es ver cómo se autoorganizan los equipos de trabajo, pues un equipo de trabajo con poco conocimiento técnico puede crear microservicios en desorden y en lugar de simplificar se hace más complejo.

Velepucha, V., Flores, P., & Torres, J. (2019). MOMMIV: Modelo para descomposición de una arquitectura monolítica hacia una arquitectura de microservicios bajo el principio de ocultación de información. *Revista Ibérica de Sistemas e Tecnologias de Informação*, (E17), 1000-1009.

Los retos en la fase de diseño se centran en proponer herramientas o lenguajes de modelado, determinar la granularidad correcta de los microservicios, determinar qué tan pequeño debe ser, cuántas funcionalidades debe incluir y manejar, también definir patrones de diseño y lenguajes arquitecturales para gestionar la complejidad de las aplicaciones basadas en microservicios. DevOps es fundamental, proponer trabajos que profundicen DevOps, microservicios y IoT (Osses, Márquez, & Astudillo, 2018). En la fase de desarrollo los retos de investigación se presentan en la gestión de los datos es uno de los retos más importantes y muy discutidos. Gestionar bases de datos distribuidas, con transacciones y consultas distribuidas es un tema abierto que requiere más investigación.

Mantener la consistencia de los datos en un sistema distribuido sin afectar el rendimiento es un tema de mucho interés (Mishra, Kunde, & Nambiar, 2018). Gestionar la interfaz de usuario con diferentes dispositivos y de diferentes tipos, se requiere ayudar a las organizaciones a implementar un entorno de microservicio de pila completa (Full-Stack microservices) (Jamshidi et al., 2018)

Vera-Rivera, F. H., Astudillo, H., & Gaona, C. (2019). Desarrollo de aplicaciones basadas en microservicios: tendencias y desafíos de investigación. *RISTI-Revista Ibérica de Sistemas e Tecnologias de Informação*, (E23 (2019)), 107-120.

Seguridad: La seguridad es otro de los retos que deben ser cuidadosamente abordados en una solución de microservicios. Es vital que exista un usuario identificado a lo largo de toda la cadena de comunicaciones que sucede entre los microservicios.

Suárez, K. T. G., Anaya, R., & Cano, A. F. (2017). Un acercamiento a los microservicios. *Unaciencia Revista de Estudios e Investigaciones*, *10*(19), 116-126.

# MÉTODO

## ENFOQUE METODOLÓGICO

El presente proyecto se desarrolló bajo un enfoque de investigación aplicada, orientado a la implementación y evaluación de un sistema distribuido basado en microservicios. Se adoptó una metodología experimental, en la cual se diseñaron, construyeron y probaron microservicios independientes con capacidades específicas de procesamiento y persistencia de datos. El modelo de arquitectura de microservicios fue seleccionado debido a su idoneidad para promover el escalamiento, la resiliencia y la mantenibilidad de aplicaciones modernas (Newman, 2015). Asimismo, se empleó un enfoque iterativo e incremental, permitiendo validar cada componente de manera aislada antes de su integración completa.

Este enfoque metodológico facilitó la captura de información a través de pruebas funcionales y de conectividad, además del procesamiento y análisis de los resultados obtenidos en cada fase del desarrollo.

## DISEÑO METODOLÓGICO

La metodología empleada consistió en el diseño de una arquitectura de microservicios conformada por tres componentes principales:

* **Discovery-Microservice**, encargado de actuar como servidor de descubrimiento mediante la tecnología Eureka Server.
* **Product-Microservice**, orientado a la gestión de productos y persistencia en una base de datos NoSQL (MongoDB).
* **Booking-Microservice,** enfocado en la gestión de reservas y persistencia en una base de datos relacional (MySQL).

El desarrollo de los microservicios se realizó de manera modular, asegurando que cada uno cumpliera funciones específicas y se pudiera probar de forma independiente. La comunicación entre los servicios se gestionó a través de Eureka Server, que permitió registrar y descubrir dinámicamente las instancias desplegadas.

El proceso metodológico contempló las siguientes fases:

1. Preparación del entorno de trabajo mediante la instalación de herramientas de desarrollo y bases de datos.
2. Construcción del Discovery Server utilizando Spring Boot y Netflix Eureka.
3. Desarrollo individual de los microservicios Product y Booking, con sus respectivas configuraciones de conexión a bases de datos.
4. Pruebas funcionales de cada servicio mediante la herramienta Postman, validando la exposición de los endpoints REST.
5. Integración de los microservicios en el ecosistema de Eureka Server y ejecución de pruebas de descubrimiento y conectividad.

Cada fase del procedimiento fue documentada y validada a través de experimentos controlados, garantizando la funcionalidad esperada de los componentes y la correcta interacción entre ellos.

## HERRAMIENTAS Y TECNOLOGÍAS UTILIZADAS

El proyecto se apoyó en una serie de herramientas y tecnologías específicas que permitieron su implementación eficiente:

* **Entorno de desarrollo**: Visual Studio Code, elegido por su ligereza y amplia compatibilidad con el ecosistema de Spring Boot.
* **Frameworks**:
  + **Spring Boot**: Framework principal para la construcción de aplicaciones Java basadas en microservicios.
  + **Spring Cloud Netflix Eureka**: Solución de descubrimiento de servicios para la integración dinámica de microservicios.
* **Bases de datos**:
  + **MongoDB**: Base de datos NoSQL utilizada para la persistencia de datos del microservicio de productos.
  + **MySQL**: Sistema de gestión de bases de datos relacional utilizado en el microservicio de reservas.
* **Cliente de APIs**: Postman, empleado para realizar pruebas de consumo de servicios RESTful y validar el comportamiento de los endpoints.
* **Manejo de dependencias**: Maven, utilizado para la gestión automática de librerías y construcción de los proyectos Spring Boot.
* **Control de versiones**: Git (opcional, si se usó), para el control y seguimiento de cambios en el código fuente.

Estas herramientas se seleccionaron considerando criterios de compatibilidad, robustez y facilidad de integración en proyectos basados en arquitecturas de microservicios.

## DISEÑO DE LA ARQUITECTURA DE LA APLICACIÓN

La arquitectura de la aplicación propuesta está basada en el patrón de microservicios, un enfoque moderno de desarrollo de software que permite dividir un sistema grande en servicios independientes que se comunican entre sí. Cada microservicio es responsable de una función de negocio específica y tiene su propia lógica, base de datos y mecanismos de comunicación. Esto facilita la escalabilidad, el despliegue independiente y el mantenimiento del sistema.

### DESCRIPCIÓN GENERAL DE LA ARQUITECTURA

La aplicación está compuesta por los siguientes microservicios principales:

* **Discovery-Service**: Servicio de registro y descubrimiento basado en Eureka Server. Permite que todos los microservicios se registren y se descubran dinámicamente entre sí.
* **Product-Microservice**: Encargado de la gestión de productos. Utiliza MongoDB como sistema de almacenamiento de datos NoSQL.
* **Booking-Microservice**: Administra las reservas de productos realizadas por los usuarios. Utiliza MySQL como base de datos relacional.
* **Payment-Microservice**: Gestiona los pagos de las reservas, conectándose a un servicio externo de procesamiento de pagos.

Cada microservicio expone su funcionalidad a través de APIs REST y se comunica mediante peticiones HTTP.

### DESCRIPCIÓN DEL DIAGRAMA DE LA ARQUITECTURA

El diagrama de arquitectura de la aplicación se estructura de la siguiente manera:

* **Discovery-Service** actúa como el centro de registro donde todos los demás microservicios se registran y descubren entre sí.
* **Product-Microservice** se encarga de las operaciones CRUD sobre productos. Cada producto está almacenado en MongoDB, permitiendo una gestión flexible y eficiente de los datos.
* **Booking-Microservice** permite a los usuarios realizar reservas de productos. Antes de confirmar una reserva, este servicio consulta el Product-Microservice para validar la disponibilidad de los productos.
* Posteriormente, el **Booking-Microservice** se comunica con el Payment-Microservice para iniciar el proceso de pago.
* **Payment-Microservice** se conecta a una API externa de pagos, gestionando el procesamiento de las transacciones.

La comunicación entre los microservicios es directa utilizando HTTP y está mediada por el Discovery-Service, el cual facilita la ubicación de instancias de servicios activos.

### VENTAJAS DE LA ARQUITECTURA PROPUESTA

Entre las principales ventajas de esta arquitectura se encuentran:

* **Escalabilidad independiente**: Cada microservicio puede escalarse de manera autónoma según la demanda.
* **Facilidad de mantenimiento**: Las actualizaciones y cambios pueden aplicarse a un solo microservicio sin afectar a los demás.
* **Tolerancia a fallos**: Un error en un microservicio no necesariamente impacta en el funcionamiento de todo el sistema.
* **Diversidad de tecnologías**: Es posible usar diferentes bases de datos y tecnologías según las necesidades específicas de cada servicio.

### REPRESENTACIÓN GRÁFICA DE LA ARQUITECTURA

### DIAGRAMAS DE SECUENCIA

A diagram of a project

AI-generated content may be incorrect.La Figura 1 muestra el flujo de interacción entre el cliente, el API Gateway y el microservicio de reservas (Booking), el cual gestiona operaciones CRUD sobre las reservas, conectándose con una base de datos MySQL. El diagrama completo se encuentra en este enlace:

**Figura 1**  
Diagrama de secuencia de Booking Microservice

*Nota*. Diagrama de Secuencia de Booking Microservice. Elaborado por los autores del documento.

El siguiente diagrama ilustra las operaciones CRUD del microservicio de productos, que se comunica con una base de datos MongoDB a través del API Gateway, permitiendo la gestión de los productos disponibles en el sistema.

**Figura 2**  
Diagrama de Secuencia de Product Microservice

A diagram with different colored labels

AI-generated content may be incorrect.

*Nota*. Diagrama de Secuencia de Product Microservice. Elaborado por los autores del documento.

Este diagrama describe cómo el microservicio de pagos se conecta con una API externa para realizar las operaciones CRUD relacionadas con los pagos, mostrando la interacción entre el cliente, el sistema y servicios externos.

**Figura 3**  
Diagrama de Secuencia de Payment Microservice

*Nota*. Diagrama de Secuencia de Payment Microservice. Elaborado por los autores del documento.

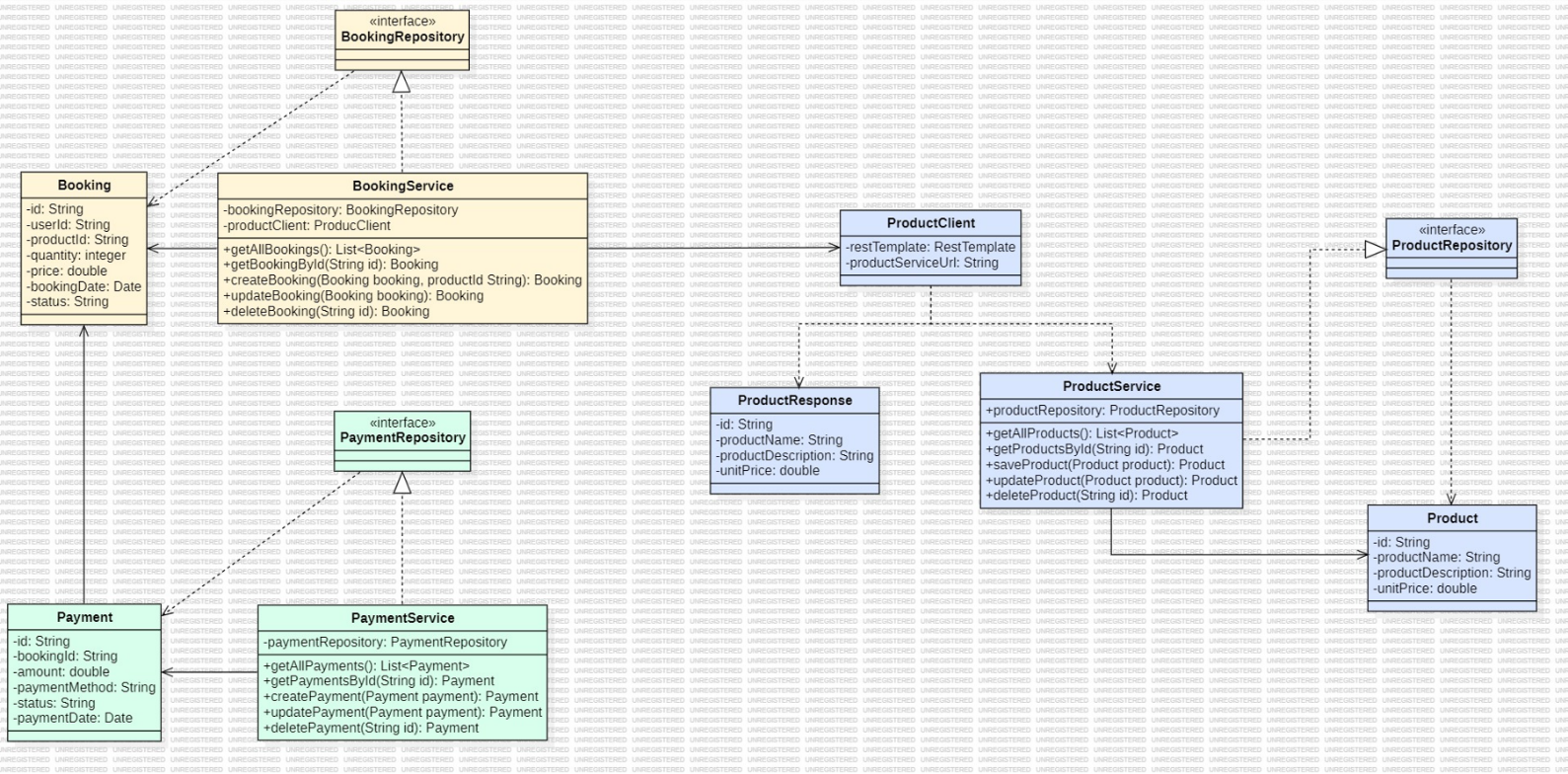
A diagram of a company

AI-generated content may be incorrect.

### DIAGRAMA DE CLASES UML

El Diagrama de Clases de la aplicación está representado en la Figura 2, la cual se presenta a continuación:

**Figura 2**  
Diagrama de clases UML de la aplicación



*Nota*. El Diagrama de Clases de la aplicación presenta el diseño de la arquitectura de la aplicación. Diagrama elaborado por los autores del documento.

El diagrama se encuentra ubicado en el repositorio de GitHub, en la siguiente ruta: <https://github.com/chartorresgg/microservices-architecture/tree/main/doc>

## PRODECIMIENTO DE DESARROLLO

El procedimiento seguido para el desarrollo del proyecto se estructuró en fases secuenciales, asegurando la correcta implementación y validación de cada componente:

### Configuración del Entorno de Trabajo

Para iniciar el proceso de desarrollo, se procedió con la instalación y configuración de las herramientas necesarias. Se utilizó Visual Studio Code como entorno de desarrollo integrado (IDE), debido a su compatibilidad con proyectos de Spring Boot y su flexibilidad para gestionar múltiples microservicios. Asimismo, se instalaron extensiones específicas para facilitar la gestión de proyectos Maven y la ejecución de servicios Java. Posteriormente, se configuraron las conexiones a las bases de datos MySQL y MongoDB, destinadas al almacenamiento persistente de la información.

La primera etapa del desarrollo consistió en la creación de un servidor de descubrimiento de servicios utilizando Spring Cloud Netflix Eureka, nombrado **Discovery-Microservice**. Este microservicio fue configurado para actuar como un registro centralizado donde los demás microservicios pudieran inscribirse y descubrirse entre sí de manera dinámica.

### Desarrollo de Microservicios

Posterior a la configuración del Discovery Server, se procedió con el diseño y construcción de los microservicios principales.

Primero, se desarrolló el Product-Microservice, orientado a la gestión de productos. Este servicio fue implementado utilizando Spring Boot y configurado para interactuar con una base de datos NoSQL, específicamente MongoDB, debido a su eficiencia en el manejo de documentos flexibles.

Luego, se implementó el Booking-Microservice, encargado de la gestión de reservas. En este caso, se optó por utilizar MySQL como sistema de gestión de bases de datos relacional, dada su robustez para garantizar integridad referencial en las operaciones de reserva.

Ambos microservicios fueron diseñados siguiendo principios de arquitectura RESTful, permitiendo la exposición de sus operaciones a través de APIs claras y estructuradas.

### Integración y Comunicación entre Microservicios

La siguiente fase consistió en la integración de los microservicios dentro del ecosistema Eureka. Tanto el Product-Microservice como el Booking-Microservice se registraron exitosamente en el Discovery Server, posibilitando la comunicación entre servicios mediante descubrimiento dinámico.

Para facilitar esta comunicación, se implementó el uso de **Feign Client**, una herramienta que permite simplificar las llamadas HTTP entre microservicios declarando las interfaces de cliente.

### Persistencia de Datos

Cada microservicio fue configurado para interactuar de manera independiente con su respectiva base de datos.

* El **Product-Microservice** se conectó a MongoDB, donde se definieron colecciones para almacenar la información de los productos.
* El **Booking-Microservice** se integró con MySQL, mediante la creación de esquemas y tablas específicas para el almacenamiento de las reservas.

Las configuraciones de persistencia se establecieron a través de archivos *application.properties*, incluyendo credenciales, direcciones de conexión y parámetros de inicialización de base de datos.

### Exposición de APIs y Pruebas

Finalmente, se diseñaron y desarrollaron los endpoints REST para ambos microservicios, permitiendo operaciones de creación, consulta, actualización y eliminación de registros (CRUD). Para validar su funcionamiento, se realizaron pruebas funcionales utilizando **Postman**, donde se verificó la correcta respuesta de los servicios ante distintas solicitudes HTTP.

## DESCRIPCIÓN DE PRUEBAS Y VALIDACIONES

### Pruebas de Conectividad entre Servicios

Se llevaron a cabo pruebas de conectividad para verificar que los microservicios pudieran registrarse y descubrirse adecuadamente a través de Eureka Server. Se comprobó que, una vez iniciados, los servicios se listaran en el panel de control de Eureka y pudieran ser localizados mediante sus nombres de servicio.

### Pruebas de Persistencia de Datos

Cada microservicio fue sometido a pruebas de persistencia, enviando solicitudes a través de Postman para almacenar datos en sus respectivas bases de datos. Posteriormente, se realizaron consultas directas a MongoDB y MySQL para validar que los datos hubieran sido almacenados de manera correcta y consistente.

### Pruebas de Funcionamiento de APIs REST

Se diseñó una batería de pruebas funcionales en Postman, evaluando operaciones CRUD (Create, Read, Update, Delete) para ambos microservicios. Cada prueba incluyó la validación de los códigos de respuesta HTTP, el tiempo de respuesta y la consistencia de los datos devueltos en formato JSON.

### Pruebas de Registro en Discovery Server

A screenshot of a computer

AI-generated content may be incorrect.Finalmente, se realizaron pruebas de disponibilidad, registrando múltiples instancias de los microservicios en Eureka para comprobar el manejo de escalabilidad y la tolerancia ante la caída de servicios. Se validó que, en caso de interrupción de un servicio, el Discovery Server actualizara el estado correspondiente en su panel de monitoreo.

**Figura 4**  
Despliegue de Eureka Server

## CRONOGRAMA DE IMPLEMENTACIÓN

La implementación del proyecto se organizó en fases claramente delimitadas para asegurar el cumplimiento de los objetivos planteados en los tiempos establecidos. A continuación, se describe el cronograma de actividades:

### FASE 1: CONFIGURACIÓN DEL ENTORNO (SEMANA 1)

Durante esta fase, se procedió a:

* Instalar y configurar Visual Studio Code.
* Integrar extensiones necesarias para proyectos de Spring Boot y Maven.
* Configurar las conexiones iniciales a bases de datos MySQL y MongoDB.
* Instalar y configurar Eureka Server.

Esta etapa fue fundamental para establecer el ambiente de trabajo adecuado para el desarrollo de los microservicios.

### FASE 2: DESARROLLO DEL DISCOVERY SERVICE (SEMANA 1)

En paralelo a la configuración del entorno, se construyó el **Discovery-Microservice** utilizando Spring Cloud Netflix Eureka. Durante esta fase:

* Se configuró el proyecto con dependencias necesarias.
* Se implementaron las propiedades de servidor de descubrimiento en el archivo *application.properties*.
* Se realizaron pruebas iniciales de despliegue para validar la correcta activación del servidor de Eureka.

### FASE 3: DESARROLLO DE PRODUCT Y BOOKING MICROSERVICE (SEMANA 2 Y 3)

Posteriormente, se desarrollaron los microservicios principales:

* **Semana 2:** Desarrollo del **Product-Microservice** basado en MongoDB. Se implementaron las operaciones CRUD y se configuraron las entidades de base de datos.
* **Semana 3:** Desarrollo del **Booking-Microservice** con conexión a MySQL. Se diseñaron las tablas necesarias y se desarrollaron los endpoints correspondientes.

Ambos microservicios fueron conectados al Discovery Server.

### FASE 4: INTEGRACIÓN Y COMUNICACIÓN ENTRE MICROSERVICIOS (SEMANA 4)

En esta etapa:

* Se integró el Discovery Client en los microservicios.
* Se implementó Feign Client para facilitar la comunicación entre servicios.
* Se realizaron pruebas de descubrimiento y llamadas entre microservicios.

### FASE 5: PRUEBAS FUNCIONALES Y VALIDACIONES (SEMANA 5)

Durante esta semana, se llevaron a cabo pruebas exhaustivas:

* Validaciones de conexión a Eureka.
* Pruebas de persistencia de datos en MongoDB y MySQL.
* Validación del funcionamiento de los endpoints a través de Postman.
* Verificación del registro y monitoreo de servicios en Eureka.

### FASE 6: AJUSTES FINALES Y DOCUMENTACIÓN (SEMANA 6)

Finalmente, se realizaron ajustes finales de configuración, optimización de código y documentación del proceso, incluyendo:

* Revisión de configuraciones de seguridad y despliegue.
* Documentación técnica de APIs REST.
* Consolidación de evidencias de pruebas y resultados obtenidos.

### CRONOGRAMA DEL PROYECTO

Para garantizar el cumplimiento de los objetivos planteados, se estableció un cronograma de actividades detallado que orientó el proceso de desarrollo e implementación del proyecto. Dicho cronograma distribuye las fases de trabajo de manera secuencial y progresiva, permitiendo una gestión organizada del tiempo y los recursos disponibles. A continuación, se presenta la planificación realizada:

|  |  |  |  |
| --- | --- | --- | --- |
| Fase | Actividad | Semana(s) | Descripción |
| 1 | Configuración del Entorno de Desarrollo | Semana 1 | Instalación de Visual Studio Code, extensiones, configuración de Bases de Datos y Eureka Server.   |  | | --- | |  | |
| 2 | Desarrollo del Discovery Server | Semana 1 | Implementación del Discovery-Microservice con Eureka Server y pruebas iniciales. |
| 3 | Desarrollo de Product y Booking Microservices | Semanas 2 y 3 | Desarrollo de Product-Microservice (MongoDB y Booking-Microservice (MySQL), con operaciones CRUD. |
| 4 | Integración y Comunicación entre Microservicios | Semana 4 | Implementación de Discovery Client y Feign Client, pruebas de comunicación entre servicios. |
| 5 | Pruebas Funcionales y Validaciones | Semana 4 | Validaciones de persistencia de datos, conexiones a Eureka y pruebas de endpoints usando Postman. |
| 6 | Ajustes Finales y Documentación | Semana 6 | Documentación técnica, comentarios del código fuente. |

El cronograma permitió llevar a cabo la implementación de los microservicios de manera ordenada, asegurando la correcta integración de los componentes, el cumplimiento de los tiempos establecidos y la calidad de los resultados obtenidos.

## OPERACIONES SOBRE LAS ESTRUCTURAS DE DATOS

En el presente sistema, los microservicios interactúan con diversas estructuras de datos, que son almacenadas y gestionadas en diferentes bases de datos, incluyendo MongoDB, MySQL y, opcionalmente, otros servicios de almacenamiento en el futuro. Las operaciones realizadas sobre estas estructuras de datos son fundamentales para el funcionamiento del sistema y permiten una comunicación efectiva entre los microservicios.

### ESTRUCTURAS DE DATOS SELECCIONADAS

Los microservicios seleccionados en este proyecto gestionan los siguientes tipos de estructuras de datos:

* **Product Service**: La estructura de datos principal es el **producto**, que se almacena en una **base de datos MongoDB**.
* **Booking Service**: La estructura de datos principal es la **reserva de un pedido**, que se almacena en una **base de datos MySQL**.
* **Payment Service**: La estructura de datos principal es el **pago**, que puede ser gestionado a través de un **API externo** como Stripe o PayPal.

Cada microservicio realiza operaciones sobre estas estructuras de datos de acuerdo con sus responsabilidades y funcionalidades.

### OPERACIONES SOBRE LAS ESTRUCTURAS DE DATOS

A continuación, se detallan las operaciones que se realizan sobre las estructuras de datos principales de cada microservicio:

**Tabla 1**  
Operaciones sobre las estructuras de datos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Microservicio | Operación | Método HTTP | Ruta | Descripción |
| Product | Obtener todos los productos | GET | |  | | --- | | /api/products |  |  | | --- | |  | | Recupera todos los productos almacenados en Product DB. |
|  | |  | | --- | | Obtener producto por ID |  |  | | --- | |  | | GET | /api/products/{id} | Recupera un producto específico desde Product DB. |
|  | |  | | --- | | Crear nuevo producto |  |  | | --- | |  | | POST | |  | | --- | | /api/products |  |  | | --- | |  | | Crea un nuevo producto en Product DB. |
|  | Actualizar producto | PUT | /api/products/{id} | Actualiza un producto en Product DB. |
|  | Eliminar producto | DELETE | /api/products/{id} | Elimina un producto de Product DB. |
| Booking | Obtener todas las reservas | GET | /api/bookings | Recupera todas las reservas almacenadas en Booking DB. |
|  | Obtener reserva por ID | GET | /api/bookings/{id} | Recupera una reserva específica de Booking DB. |
|  | Crear una nueva reserva | GET | /api/bookings | Crea una nueva reserva en Booking DB. |
|  | Actualizar reserva | POST | /api/bookings/{id} | Actualiza una reserva en Booking DB. |
|  | Eliminar reserva | DELETE | /api/bookings/{id} | Elimina una reserva de Booking DB. |
| Payment | Obtener todos los pagos | GET | /api/payments | Recupera todos los pagos almacenados en Payment DB. |
|  | Obtener pago por ID | GET | /api/payments/{id} | Recupera un pago específico desde Payment DB. |
|  | Crear nuevo pago | POST | /api/payments | Crea un nuevo pago en Payment DB. |
|  | Actualizar pago | PUT | /api/payments/{id} | Actualiza un pago en Payment DB. |
|  | Eliminar pago | DELETE | /api/payments/{id} | Elimina un pago de Payment DB. |

*Nota*. Esta tabla presenta las operaciones clave en cada microservicio. Tabla elaborada por los autores del documento.

### GRÁFICO DE LAS OPERACIONES DE LAS ESTRUCTURAS DE DATOS

En la Figura 1 se desarrolló la representación gráfica de las operaciones.

**Figura 5**  
Operaciones sobre las estructuras de datos

*Nota*. Imagen elaborada por los autores del documento.

A diagram of a software company

AI-generated content may be incorrect.

## REPOSITORIOS Y GESTIÓN DEL PROYECTO

Para la gestión del proyecto y el control de versiones, se utilizaron dos herramientas clave: **GitHub** y **Notion**.

* **GitHub**: El código fuente y todos los recursos relacionados con el desarrollo de los microservicios fueron almacenados y gestionados a través de GitHub, una plataforma de control de versiones ampliamente utilizada para proyectos de software. GitHub permitió una gestión eficiente del código, facilitando el trabajo colaborativo y el seguimiento de los cambios realizados en el desarrollo. A través de esta herramienta, se logró mantener un registro detallado de todas las modificaciones y la implementación de nuevas funcionalidades. El repositorio de GitHub utilizado para este proyecto es el siguiente: <https://github.com/chartorresgg/microservices-architecture>
* **Notion**: Para la gestión de las actividades y la planificación del proyecto, se utilizó **Notion**, una plataforma de productividad que combina herramientas de gestión de tareas, bases de datos y documentación en un solo lugar. En Notion, se organizaron las tareas, se gestionaron los plazos y se mantuvo un seguimiento de los avances del proyecto. Además, Notion facilitó la documentación colaborativa, permitiendo a los miembros del equipo acceder a información clave de manera rápida y organizada. El enlace al proyecto en Notion, donde se detallan las actividades realizadas, cronograma y decisiones clave, es el siguiente: <https://chartorresg.notion.site/PIF-Microservicios-en-Arquitectura-de-Software-156a5d686cd2802abc77d47e645872fc?pvs=4>

# RESULTADOS

*(Este apartado será presentado en la entrega del tercer corte).*

*Expone los hallazgos y datos resultantes del análisis de datos presentados anteriormente en los métodos, los cuales deben tener coherencia y ética respecto al tratamiento de la pregunta de la investigación planteada. En este espacio, se realiza la constatación de las hipótesis y resultados de los objetivos de investigación planteados en el estudio argumentando sus diferentes perspectivas/hallazgos. Se recomienda:*

* *Evidenciar situaciones o hallazgos con argumentos en hechos facticos.*
* *No hacer de este espacio o sección una lista de situaciones*
* *No usar o insertar notas al pie, ni notas al final*
* *Usar subtítulos que aporten mejoras en la legibilidad del documento*

# DISCUSIÓN Y CONCLUSIÓN

*(Este apartado será presentado en la entrega del tercer corte).*

*Interpreta los resultados de la investigación demostrando la relevancia de la investigación e innovación en el área de conocimiento.*

*Necesariamente, establece recomendaciones a nuevos estudios para su abordaje o continuidad de conformidad a los resultados encontrados.*

# REFERENCIAS BIBLIOGRÁFICAS

[1] avis Kažemaks and J. Decouchant, “SoK: Microservice Architectures from a Dependability Perspective”.

[2] Contreras, D. A. B. (2018). Arquitectura de microservicios. *Tecnología Investigación y Academia*, *6*(1), 36-46.

[2] D. Faustino, N. Gonçalves, M. Portela, and A. Rito Silva, “Stepwise migration of a monolith to a microservice architecture: Performance and migration effort evaluation,” *Performance Evaluation*, vol. 164, p. 102411, May 2024, doi: 10.1016/J.PEVA.2024.102411.

[3] “Semantic Dependency in Microservice Architecture”.

[4] K. Yin and Q. Du, “On Representing Resilience Requirements of Microservice Architecture Systems”.

[5] J. Bonilla, J. A. Carballo, V. Abad-Alcaraz, M. Castilla, J. D. Álvarez, and J. Fernández-Reche, “A real-time and modular weather station software architecture based on microservices,” *Environmental Modelling & Software*, vol. 186, p. 106337, Mar. 2025, doi: 10.1016/J.ENVSOFT.2025.106337.

[6] A. S. Abdelfattah *et al.*, “Multivocal study on microservice dependencies,” *Journal of Systems and Software*, vol. 222, p. 112334, Apr. 2025, doi: 10.1016/J.JSS.2025.112334.

[7] M. Hui *et al.*, “Unveiling the microservices testing methods, challenges, solutions, and solutions gaps: A systematic mapping study,” *Journal of Systems and Software*, vol. 220, p. 112232, Feb. 2025, doi: 10.1016/J.JSS.2024.112232.

[8] S. Schneider *et al.*, “Comparison of Static Analysis Architecture Recovery Tools for Microservice Applications,” *Proceedings of Mining Software Repositories (MSR’24)*, vol. 1, 2024, doi: XXXXXXX.XXXXXXX.

[9] J. Sebastian and V. Quintero, “Implementación de técnicas de auto escalado proactivo en Kubernetes para mejorar la elasticidad de los microservicios”.

[10] J. José Rojas Pérez and L. Daniela Sandoval Perez, “UNIDADES TECNOLÓGICAS DE SANTANDER CIENCIAS NATURALES E INGENIERÍAS INGENIERÍA DE SISTEMAS BUCARAMANGA Y FECHA (16, septiembre del 2024)”.

[11] D. López and E. Maya, “Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web”.

[12] “Vista de Arquitectura de software para el desarrollo de aplicaciones web orientada a micro-servicios en TecNM campus Escárcega.” Accessed: Mar. 09, 2025. [Online]. Available: https://progmat.uaem.mx/progmat/index.php/progmat/article/view/v15n02a02/v15n02a02pdf

[13] R. Fresno-Aranda, “Automated Capacity Analysis of Limitation-Aware Microservices Architectures,” 2022, doi: 10.1145/3540250.3558905.

[14] I. DE Postgrado and I. Alexis Fernando Saransig Chiza, “UNIVERSIDAD TÉCNICA DEL NORTE”.

[15] A. Bruno, M. López, J. Liriano, G. Tutor, and J. Bernal Bermúdez, “UNIVERSIDAD POLITÉCNICA DE MADRID Trabajo Fin de Máster”.

[16] T. M. Arturo Ramírez Romero en C Chadwick Carreto Arellano, “Instituto Politécnico Nacional,” 2021.

[17] B. Barua and M. S. Kaiser, “Microservices-Based Framework for Predictive Analytics and Real-time Performance Enhancement in Travel Reservation Systems”.

# GESTIÓN Y USO DE MENDELEY

**Figura 1**

A screenshot of a computer

AI-generated content may be incorrect.*Gestión de referencias bibliográficas con Mendeley, en Microsoft Word.*

*Nota.* La figura muestra las referencias bibliográficas añadidas al gestor de Mendeley para Microsoft Word y su disponibilidad para ser insertadas. Fuente: Elaboración propia (2025).

**Figura 2**

A computer screen shot of a computer screen

AI-generated content may be incorrect.*Listado de las referencias bibliográficas seleccionadas, en el gestor de referencias Mendeley.*

*Nota*. La figura muestra las referencias bibliográficas de los artículos de investigación que fueron seleccionados para el documento. Fuente: Elaboración propia (2025).

**Figura 3**

A screenshot of a computer

AI-generated content may be incorrect.*Uso del gestor de referencias de Mendeley, para agregar artículos de investigación.*

*Nota*. La figura muestra el procedimiento para agregar un artículo de investigación, al gestor de referencias de Mendeley. Fuente: Elaboración propia (2025).