UNIVERSIDAD NACIONAL
SANTIAGO ANTÚNEZ DE MAYOLO

FACULTAD DE CIENCIAS

ESCUELA PROFESIONAL
DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

"ARQUITECTURA DE SOFTWARE APH PARA MEJORAR EL DESARROLLO DE MICROSERVICIOS EN LA EMPRESA HIPER, AÑO 2022"

TESIS

PARA OPTAR EL TÍTULO PROFESIONAL DE
INGENIERO DE SISTEMAS E INFORMÁTICA

PRESENTADO POR:

Bach. MORALES CARLOS, ALDO OMAR

ASESOR

Mag. SILVA ZAPATA, MIGUEL ANGEL

HUARAZ – PERÚ

2022

N° Registro: T124

DEDICATORIA

A mi querida madre, por brindarme su cariño y apoyo incondicional.

AGRADECIMIENTO

A mi madre, por acompañarme y darme las fortalezas para lograr avanzar cada peldaño de mi vida.

Mis familiares, por brindarme las motivaciones constantes que me llena de gratitud por ser parte de mis alegrías y fracasos.

A los profesionales que son parte de mi crecimiento, por compartir grandes ideas, sugerencias, críticas, los valores y cada detalle que sumaron para lograr sobrepasar los obstáculos.

Al divino creador, por darme vida y salud, que gracias a ello lograr cada objetivo anhelado.

Aldo Omar Morales Carlos

PRESENTACIÓN

Señores miembros del jurado:

En cumplimiento con el reglamento de grados y títulos de la Escuela Profesional de Ingeniería de Sistemas e Informática, de la Facultad de Ciencias, de la Universidad Nacional Santiago Antúnez de Mayolo, me presento ante ustedes y presento mi tesis titulada "ARQUITECTURA DE SOFTWARE APH PARA MEJORAR EL DESARROLLO DE MICROSERVICIOS EN LA EMPRESA HIPER, AÑO 2022".

Hiper, es una organización dedicada al servicio de tecnologías de información, hasta la actualidad desarrolladas e implementadas diferentes productos comerciales en el ámbito local e internacional. Uno de los problemas principales se ha identificado dentro del departamento de desarrollo, quienes administran diferentes proyectos de gran envergadura para clientes variados. Los proyectos con enfoque a microservicios que se vienen implementando actualmente no sigue un estándar y/o control de calidad, la finalidad de la tesis es desarrollar e implementar una arquitectura que permita mejorar el desarrollo de los microservicios.

El informe de la investigación está conformado por los siguientes capítulos:

- Capítulo I: Planteamiento y formulación del problema, objetivos y justificación de la investigación.
- Capitulo II: Antecedentes, teorías, la hipótesis y las variables que sustentan la investigación
- Capitulo III: Tipo de estudio e investigación, descripción de la población, instrumentos de recolección y técnicas de análisis y prueba de hipótesis.
- Capitulo IV: Descripción del trabajo de campo, presentación de resultados y prueba de hipótesis y la discusión de resultados.
- Capitulo V: Conclusiones

RESUMEN

La presente tesis está orientada al desarrollo e implementación de la arquitectura de software APH usando herramientas de código libre para mejorar el desarrollo de microservicios en la organización Hiper. Al iniciar la tesis, Hiper no contaba con una arquitectura de software que le permita mejorar y estandarizar los microservicios que construyen, estas fueron realizadas a criterio y experticia del equipo asignado al proyecto.

El tipo de investigación es aplicada, descriptivo y experimental. Para el desarrollo de la arquitectura APH se utilizó el lenguaje de programación Java, el marco de desarrollo de Spring. El principal objetivo es mejorar el desarrollo de los microservicios en base a la arquitectura APH, para ello se ha implementado tres arquetipos principales para la generación de un API Básico, API CRUD y API Batch. Cada una de ellas compuesta por los artefactos necesarios para cubrir las funcionalidades esperadas.

A partir de los resultados alcanzados se concluyó que la arquitectura APH mejora el desarrollo de los microservicios, en cuanto a la flexibilidad y agilidad en el desarrollo, la modularización de los microservicios, mejorando las capacidades de interconectividad con servicios de clientes, integrando funcionalidades para el envío de notificaciones masivos, registro de la trazabilidad completa y la documentación intuitiva que describe e integra las operaciones a través de la interfaz.

Palabras clave: Arquitectura, Microservicios, Arquetipos, Artefactos, APH (Arquitectura de Programación Hiper)

ABSTRACT

This thesis is oriented to the development and implementation of the APH software architecture using open-source tools to improve the development of microservices in the Hiper organization. When starting the thesis, Hiper did not have a software architecture that allows it to improve and standardize the microservices they build, these were carried out at the discretion and expertise of the team assigned to the project.

The type of research is applied, descriptive and experimental. For the development of the APH architecture, the Java programming language, the Spring development framework, was used. The main objective is to improve the development of microservices based on the APH architecture, for which three main archetypes have been implemented for the generation of a Basic API, CRUD API and Batch API. Each of them composed of the necessary artifacts to cover the expected functionalities.

Based on the results achieved, it was concluded that the APH architecture improves the development of microservices, in terms of flexibility and agility in development, the modularization of microservices, improving interconnectivity capabilities with customer services, integrating functionalities for the sending massive notifications, complete traceability record and intuitive documentation that describes and integrates the operations through the interface.

Keywords: Architecture, Microservices, Archetypes, Artifacts, APH (Hyper Programming Architecture)

ÍNDICE GENERAL

INDICE DE TABLAS

INDICE DE GRÁFICOS

1. INTRODUCCIÓN

1.1. Planteamiento del problema

A menudo, los equipos de desarrollo al iniciar con una aplicación tienden a realizar cierta cantidad de módulos y/o proyectos para cubrir los requerimientos, orientado en la construcción de servicios que en su mayoría comparten una serie de características en común. En la actualidad, al encontrarse en estos escenarios se suele crear el servicio totalmente nuevo o una copia de uno existente, provocando incremento en los recursos y con las probabilidades altas en riesgo de errores.

El departamento de desarrollo está compuesto por varios equipos asignados en diferentes proyectos de gran envergadura, cada integrante con niveles de conocimiento y experiencia variada que se encuentran involucrados en la construcción de los microservicios, al finalizar los proyectos no se cuenta con una estructura definida, porque no se aplica un estándar de desarrollo, lo que imposibilita la administración y mantenimiento continuo en tiempos óptimos.

Al tener proyectos planificados y con tiempos estimados, el equipo de desarrollo debe concentrarse en la lógica del negocio de acuerdo con los requerimientos planteados; sin embargo, esta no se realiza, porque no se cuenta con los utilitarios que permita crear un proyecto base integrado los componentes principales. Los tiempos dedicados a estructurar los servicios nuevos conlleva retrasos y a mayor costo en la planificación.

Dada las ineficiencias y atendiendo a las necesidades que presenta la organización, es necesario desarrollar la ARQUITECTURA DE SOFTWARE APH PARA MEJORAR EL DESARROLLO DE MICROSERVICIOS EN LA EMPRESA HIPER, con la

finalidad de automatizar los procesos, evitando tareas repetitivas y brindando mejor experiencia al usuario, así marchen de acuerdo con sus objetivos y metas establecidas.

1.2. Formulación del problema

1.2.1. Problema general

¿En qué medida mejora el desarrollo de microservicios en la empresa Hiper con la arquitectura de software APH?

1.2.2. Problemas específicos

1. ¿Cómo implementar la arquitectura de software APH que satisfaga los requerimientos de construcción de los microservicios?
2. ¿Determinar en qué medida se incrementa la interoperabilidad de los microservicios construidos con la arquitectura de software APH?
3. ¿Determinar en qué medida mejora la mantenibilidad de los microservicios construidos con la arquitectura de software APH?
4. ¿Determinar en qué medida se aumenta la fiabilidad de los microservicios construidos con la arquitectura de software APH?
5. ¿Determinar en qué medida se maximiza la escalabilidad de los microservicios construidos con la arquitectura de software APH?

1.3. Objetivos de la investigación

1.3.1. Objetivo general

Mejorar el desarrollo de microservicios mediante la arquitectura de software APH en la empresa Hiper.

1.3.2. Objetivos específicos

1. Implementar la arquitectura de software APH que satisfaga los requerimientos de construcción de los microservicios.
2. Incrementar la interoperabilidad en la construcción de los microservicios en base a la arquitectura de software APH.
3. Mejorar la mantenibilidad de los microservicios en base a la arquitectura de software APH.
4. Aumentar la fiabilidad de los microservicios en base a la arquitectura software APH.
5. Maximizar la escalabilidad de los microservicios con la arquitectura de software APH.

1.4. Justificación de la investigación

La presente investigación tiene vital importancia, puesto que está encaminado a automatizar y mejorar el desarrollo de microservicios en la empresa Hiper, considerando que se pretende diseñar una arquitectura de software APH flexible y dinámica apoyada en microservicios que permita el desarrollo estándar de los servicios, brindando énfasis a la reutilización, escalabilidad y mantenibilidad.

1.4.1. Justificación tecnológica

Con el desarrollo de nuevas tecnologías, se busca que esté en la capacidad de brindar mejor experiencia al usuario, procesos automatizados, estandarización y reducción de las incidencias.

La arquitectura de software APH se basa en tecnologías de código abierto, con la finalidad de brindar las mismas capacidades en el mundo distribuido y reduce los

costos generales del sistema. Así lo mismo, una vez que los arquetipos sean creados e implementados en el repositorio de la organización, estarán disponibles sin considerar el tiempo ni el lugar donde se encuentra los desarrolladores.

Con la arquitectura de software APH se brindará a los departamentos de desarrollo, arquitectura, seguridad y calidad de la organización un mecanismo que permitirá optimizar los flujos de trabajo en determinadas tareas relacionadas al desarrollo de microservicios.

Para llevar el desarrollo y su respectiva implementación la organización dispone con equipos y tecnologías necesarias.

1.4.2. Justificación económica

La arquitectura de software propuesto para el desarrollo de microservicios reducirá los costos, el tiempo, conduciendo a los departamentos involucrados a tener mayor eficiencia en cada uno de sus procesos. Los equipos de desarrollo podrán enfocarse en los requerimientos funcionales logrando cumplir con los objetivos planificados, evitando las tareas repetitivas.

Para hacer realidad la presente investigación la organización está en la posibilidad de cubrir con todos los gastos desde su inicio hasta culminar la investigación.

1.4.3. Justificación operativa

La arquitectura propuesta, establecerá un trabajo innovador y fructífero en la organización objeto de estudio, centralizando los componentes reusables, automatizando los procesos, brindando seguridad, disponibilidad, interoperabilidad, mantenibilidad y fiabilidad, así lo mismo, tener una estructura estandarizada en la construcción de los microservicios.

Así, la organización contempla del departamento de arquitectura con conocimientos sobre el ámbito informático, quienes serán los responsables de instruir el funcionamiento de la arquitectura de software, frente a la existencia de problemas complejos el tesista estará en la disposición de brindar la asistencia técnica del sistema.

1.4.4. Justificación social

El presente proyecto de investigación ayudará al departamento de desarrollo, arquitectura, seguridad y calidad de la empresa Hiper, tenga un estándar de desarrollo de los microservicios fiable y segura, evitando al equipo de tareas repetitivas, retrasos, errores, etc. Así todo el personal del departamento de desarrollo que labora en dicha organización tendrá la disponibilidad para interactuar mediante APH CLI para administrar los componentes de la arquitectura de software APH.

Al tener una arquitectura centralizada, nos permitirá construir microservicios de calidad, que cumplan los estándares de desarrollo, buscando la alta probabilidad de darle mayor oportunidad de atención y brindar un servicio con expectativas de crecimiento, fortaleciendo la confianza con los clientes de la empresa Hiper.

1.4.5. Justificación legal

La presente investigación se justifica legalmente debido a que su desarrollo no contraviene ninguna ley ni normativa vigente, más por el contrario ayuda a que la organización Hiper cumpla con las leyes y reglamentos que se detallan a continuación.

- ISO/IEC TS 23167, proporciona una descripción (especificaciones técnicas) de un conjunto de tecnologías y técnicas comunes utilizadas junto a la computación en la nube, entre ellas la arquitectura de microservicios y la automatización. Estas definiciones están dirigidas a desarrolladores y personal de operaciones, cada vez más vinculados entre sí. En un enfoque unificado llamado DevOps, que el objetivo es acelerar y simplificar la creación y operación de soluciones basadas en el uso de servicios en la nube.
- NTP-ISO/IEC 12207, establece un marco de referencia para los procesos del ciclo de vida del software, con la finalidad de mejorar los procesos, el marco de referencia engloba todo el ciclo de vida del software a partir de la conceptualización de ideas, procesos, actividades y tareas que son participes en la adquisición de servicios de software.
- NTP-ISO/IEC 17799, su objetivo es direccionar y brindar soporte a la gestión de la seguridad de la información en correspondencia con los requisitos de la organización, la legislación y las regulaciones. La presente investigación contribuye con esta ley, porque se podrán implementar mejores estrategias de seguridad para el acceso a los microservicios que son objetos encargados de procesar las informaciones confidenciales de los clientes.

2. MARCO TEÓRICO

2.1. Antecedentes de la investigación

2.1.1. Antecedentes internacionales

a) López, J. (2017). *ARQUITECTURA DE SOFTWARE BASADA EN MICROSERVICIOS PARA EL DESARROLLO DE APLICACIONES WEB*

DE LA ASAMBLEA NACIONAL (Tesis de maestría). Universidad Técnica del Norte. Ecuador – Ibarra.

El estudio de la presente tesis abarca en la identificación de tecnologías, metodología aplicada y la arquitectura que utiliza la CGTIC para el desarrollo e implementación de aplicaciones web y los microservicios, en el marco de la investigación tiene un enfoque cualitativo, descriptiva con un diseño fundamental.

El objetivo se centra en presentar una arquitectura de software enfocado a aplicar los lineamientos de los microservicios para la construcción de aplicaciones web en la CGTIC.

Al finalizar este proyecto se llega a las siguientes conclusiones:

- La arquitectura de software propuesta permite al equipo de desarrollo a tomar mejores decisiones durante la implementación o migración de una funcionalidad para una aplicación web.
- En el marco de la infraestructura y operaciones, se ha demostrado que la aplicación del diseño propuesto bosqueja una nueva organización de esta, lo que permite a la organización a nivel de la infraestructura física, lógica y de comunicaciones proveer soporte a la solución mediante esta arquitectura.
- Los diseños de arquitectura bajo microservicios traen consigo una serie de beneficios en el desarrollo de software, operaciones y cultura organizacional ya que plantea una nueva forma de implementación de sistemas de información.

Apreciación del investigador

De acuerdo con el propósito de la tesis mencionada se rescata que la investigación ha permitido identificar tecnologías y metodologías a usar para el desarrollo e implementación de los microservicios, así como la identificación de los requisitos funcionales y necesidades concernientes al desarrollo de aplicaciones web para satisfacerlas mediante el prototipo y diseño de una arquitectura de software. Por lo mencionado realzo la relevancia del desarrollo de tesis relacionadas a una arquitectura de software basada en microservicios.

b) Cols, A. & Salcedo, M. (2017). *ARQUITECTURA DE MICROSERVICIOS CON RESTFUL* (Tesis de maestría). Universidad Politécnica de Madrid. España - Madrid.

La investigación está orientada a detallar y/o explicar respecto al contexto de la arquitectura con enfoque hacia los microservicios, se define las ventajas y desventajas que nos proporciona al aplicar este tipo de diseño, describir los lineamientos a seguir para componer un sistema en microservicios, finalmente orientar el uso adecuado de acuerdo al contexto del proyecto.

Su objetivo se basa en:

- Concebir la estructura de un microservicio, describir sus principales ventajas, desventajas y características principales.
- Contrastar una arquitectura monolítica con los microservicios.

Al término de la investigación se llega a la conclusión que los módulos por el hecho de estar en aplicaciones separadas, se puede implementar de modo más

conveniente y/o óptimo con la finalidad de cubrir las necesidades, así como la disposición para escalar horizontal como verticalmente cada microservicio.

Apreciación del investigador

De la tesis mencionada se puede apreciar que el enfoque basado microservicios nos permite ejecutar y escalar de manera horizontal y vertical. Así, la aplicación puede alterar su tamaño en función de las necesidades reales del negocio, por ende, es necesario automatizar y tener el control desde una arquitectura superior que nos permita optimizar y centralizar los microservicios.

c) Gesteira, R. (2020). *IMPLEMENTACIÓN DE UNA ARQUITECTURA DE MICROSERVICIOS PARA UNA RED DE SENSORES IOT SOBRE ARDUINO* (Tesis de grado). Universidad Pontificie Comillas. España – Madrid.

En la tesis de grado se estudia la viabilidad de utilizar una arquitectura de microservicios con el propósito de gestionar una red de sensores conectados a una placa de Arduino. Para darle solución se ha desarrollado una aplicación web de domótica que se apoya en una arquitectura de microservicios desplegada en Kubernetes para comunicarse con los sensores.

Los objetivos de esta investigación son:

- Desarrollar una arquitectura basado en microservicios con el objetivo de usar y sacar provecho al máximo los beneficios que ofrecen los microservicios.

- Configurar una red de sensores IoT conectadas en la placa de Arduino, que estará encargado de enviar datos de los sensores mediante una red de WIFI hacia los correspondientes microservicios.

Al culminar esta investigación se concluye que el objetivo esencial se ha logrado, y que es completamente viable implementar una arquitectura de microservicios para gestionar una red de sensores.

Apreciación del investigador

En la tesis mencionada el autor llega a una conclusión que es viable la implementación de una arquitectura de microservicios, esto nos orienta a plantear y estructurar una arquitectura centralizada orientada a gestionar diferentes microservicios que se van construyendo.

d) Noraimar, Y. & Mora, M. (2019). *ARQUITECTURA BASADA EN MICROSERVICIOS PARA SISTEMAS DE E-COMMERCE DE LA EMPRESA LCC OPENTECH, C.A.* (Tesis de pregrado). Universidad Católica Andrés Bello. Venezuela – Guayana.

El presente estudio se realiza en una empresa dedicada al e-commerce, debido al avance tecnológico los usuarios se están integrando a la nueva forma de interactuar con las plataformas tecnológicas, así se ha observado un incremento en el número de usuarios en operabilidad que repercute un aumento en la exigencia de los sistemas, ocasionando problemas de rendimiento y disponibilidad. Por tal razón el estudio está focalizado en el estudio de la arquitectura de microservicios ya que nos proporciona una serie de beneficios como la alta disponibilidad y rendimiento optimo.

El objetivo de esta investigación es plantear una propuesta de arquitectura de software basada en microservicios para la empresa LCC OpenTech C.A. la cual sería utilizada para llevar a cabo el desarrollo de sistemas de e-commerce con alto nivel de disponibilidad y escalabilidad.

Algunas conclusiones llegadas en esta investigación son:

- El período de diseño de una arquitectura de software es crucial, donde se toman las decisiones acerca de los patrones que serán utilizadas para satisfacer los requerimientos del sistema.
- La valoración de la arquitectura de microservicios se realizó en base a pruebas de carga, resistencia y funcionalidad, permitiendo validar que la arquitectura propuesta está en la capacidad de brindar el soporte a los sistemas de e-commerce.

Apreciación del investigador

La tesis mencionada resalta que el boceto de la arquitectura apoyada en microservicios nos permite tener un desarrollo sofisticado en componentes modularizados de acuerdo con los requerimientos del sistema. Estos resultados nos dan la viabilidad que los desarrollos futuros deben diseñarse usando este enfoque para dar escalabilidad, tolerancia a fallos, etc.

2.1.2. Antecedentes nacionales

a) Villaizán, H. (2019). *ARQUITECTURA DE SOFTWARE BASADA EN MICROSERVICIOS PARA IMPLEMENTACIÓN DE LA APLICACIÓN WEB DE COBRANZA DIGITAL EN FINANCIAL SYSTEMS COMPANY SAC* (Tesis de grado). Universidad Continental. Perú – Huancayo.

En la investigación de la presente tesis, se contempla la arquitectura de microservicios en la aplicación web de cobranza digital, para ello se hizo un estudio de la interacción entre sus módulos y/o componentes, definiendo la lista de atributos, considerando que la aplicación web interactúe por diversos canales como correo, redes sociales, mensajes de texto y chatbots durante el proceso de gestión de cobranza.

El objetivo principal es bosquejar la arquitectura de software orientada a microservicios para la llevar a cabo la implementación en la aplicación web de cobranza digital de la organización.

A partir del estudio realizado se llegó a las siguientes conclusiones:

- Se logró implementar con éxito la arquitectura de software orientada en microservicios en la aplicación web de cobranza digital de la organización.
- El prototipo de la arquitectura de software orientada en microservicios incorpora diferentes servicios de terceros aseverando la alta disponibilidad y transmitiendo la responsabilidad a terceros por canales digitales implementados.
- Los resultados obtenidos de la evaluación del uso de canales digitales, solución tecnológica basada en microservicios tiene la capacidad de realizar envíos masivos de mensajes sin perjudicar el rendimiento, así lo mismo manteniendo la alta disponibilidad.

Apreciación del investigador

La tesis mencionada nos muestra que la implementación de la arquitectura orientada en microservicios en la aplicación web de cobranza digital se ha logrado su capacidad para el envío de mensajes masivos manteniendo la disponibilidad y escalabilidad en el tiempo. Teniendo en cuenta el resultado obtenido por el autor de esta tesis se puede decir que una arquitectura orientada en microservicios permite modularizar las funcionalidades y que estas evolucionen por separado, para ello se debe realizar con pericia y a la vez disponer de una herramienta que nos facilite automatizar la construcción de cada microservicio, con la finalidad que el equipo de desarrollo se centre en actividades de lógica funcional más no en la estructura completa.

b) Ruelas, D. (2017). *MODELO DE COMPOSICIÓN DE MICROSERVICIOS PARA LA IMPLEMENTACIÓN DE UNA APLICACIÓN WEB DE COMERCIO ELECTRÓNICO UTILIZANDO KUBERNETES* (Tesis doctor). Universidad Nacional del Altiplano. Perú – Puno.

El objetivo principal en esta investigación es plantear un modelo de composición de microservicios orientado al comercio electrónico implementando una aplicación con la tecnología de Kubernetes, el diseño se ha contemplado a través de los modelos y notaciones de procesos de negocio del comercio electrónico logrando bosquejar el prototipo de la arquitectura de microservicios para llevar a cabo su implementación y al final realizar una serie de evaluaciones a través de pruebas de carga para validar el cumplimiento de los atributos de calidad.

Al culminar la investigación se llega a la conclusión que el modelo propuesto funciona efectivamente en comparación de un modelo monolítico en un 104% con respecto a los indicadores de disponibilidad, tiempo de respuesta y rendimiento.

Apreciación del investigador

Los resultados obtenidos de los microservicios frente a un modelo monolítico son fiables para aquellos indicadores en disponibilidad, tiempo de respuesta y rendimiento. Esto nos consta que una arquitectura centralizada y organizada nos podría permitir una alta disponibilidad, reusabilidad y seguridad.

2.2. Bases teóricas

2.2.1. Arquitectura de software

Existen diversas definiciones en relación con la arquitectura de software, en general “su principal objetivo radica en ofrecer cierta calidad al sistema de administración de datos, a partir de su desempeño, ahorro de tiempo, disponibilidad y usabilidad, la capacidad de modificarse y adecuarse a las nuevas necesidades del sistema” (KeepCoding, 2022).

Según (Albertos, 2018), describe como “un conjunto de patrones y abstracciones que proporcionan un marco de referencia para guiar en la construcción de sistemas informáticos, donde programadores, analistas y todos los ingenieros compartan una línea de trabajo común”.

La arquitectura de software es el proceso por el cual se define una solución para los requisitos técnicos y operacionales del mismo. Este proceso define qué componentes forman el software, como se relacionan entre ellos, y como mediante

su interacción llevan a cabo la funcionalidad especificada, cumpliendo con los criterios previamente establecidos; como seguridad, disponibilidad, eficiencia o usabilidad (Tupac, 2022).

La arquitectura de software es el diseño de más alto nivel de la estructura de un sistema, el cual consiste en un conjunto de patrones y abstracciones que proporcionan un marco claro para la implementación del sistema (Blancarte Iturralde, 2020, p. 30).

Es importante resaltar que la arquitectura de software contempla una serie de etapas que parte de análisis de requisitos, diseño de la solución, documentación y pruebas de su funcionamiento, la finalidad de cumplir estas etapas es mantener la calidad de datos y su confiabilidad.

2.2.2. Estilos arquitectónicos

Según (Blancarte Iturralde, Introducción a la arquitectura de software - Un enfoque práctico, 2020), un estilo arquitectónico establece un marco de referencia a partir del cual es posible construir aplicaciones que comparten un conjunto de atributos y características mediante el cual es posible identificarlos y clasificarlos. (p. 46)

Un estilo arquitectónico es una colección con nombre de decisiones de diseño arquitectónico que son aplicables a un contexto de desarrollo dado, restringen decisiones de diseño arquitectónico que son específicas de un sistema particular dentro de ese contexto, y obtienen cualidades beneficiosas en cada uno sistema resultante (Taylor y otros, 2009).

Un estilo arquitectónico define una familia de sistemas en términos de un patrón de organización estructural; un vocabulario de componente y conectores, con restricciones sobre cómo se pueden combinar (Shaw & Garlan, 1996).

Se debe tener en claro la diferencia que existe entre los estilos y patrones arquitectónicos, aunque puedan parecer similares, los estilos nos permiten determinar las características que debe tener un componente, lo cual hace que sea fácil de reconocer. Así lo mismo, no determinan la tecnología y los detalles técnicos para que sea construido el software.

A continuación, se describe algunos estilos arquitectónicos más comunes:

- **Microkernel**

  También conocido como estilo arquitectónico enchufable o Plug-in, se utiliza cuando se crean sistemas con componentes intercámbiales.

  Este estilo es aplicable a aquellos sistemas que deben ser capaces de adaptarse a los requisitos cambiantes. Los componentes de la arquitectura Microkernel constituye de un sistema central y módulos enchufables, que permite extensibilidad, flexibilidad y aislamientos de las características de la aplicación y la lógica de procesamiento personalizada. Así el sistema central se enfoca solo en la funcionalidad mínima necesaria para que el sistema sea operativo.

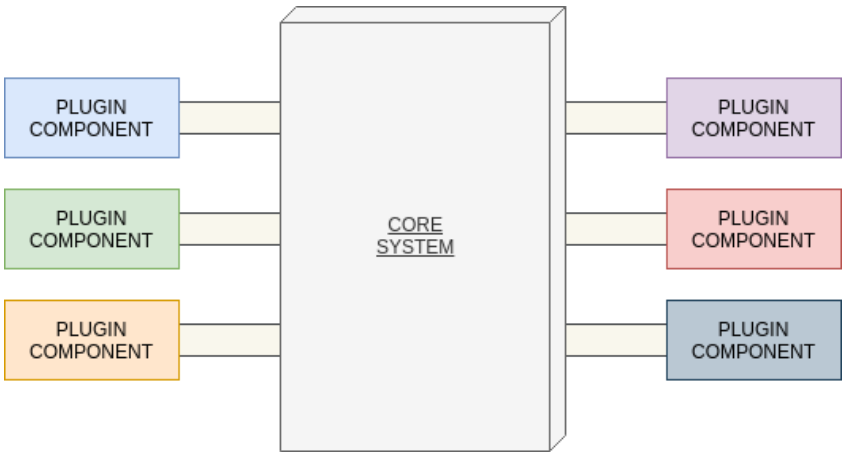
  Esta arquitectura proporciona ciertas ventajas como:

  - Flexibilidad y extensibilidad.
  - La funcionalidad de los módulos o plugins pueden probarse independientemente de forma aislada del sistema central.

- Facilidad de despliegue.
- Alto rendimiento, dado que permite caracterizar y sistematizar las aplicaciones para incluir las características necesarias.

El IDE de Eclipse está construida bajo la arquitectura del Microkernel es, donde el producto original proporciona poco más que un editor; sin embargo, al agregar las librerías o plugins, se transforma en un producto altamente personalizable y útil.

Gráfico 1: *Patrón arquitectónico Microkernel*



Fuente: Lopez, J. (2022). *Patrones de arquitectura de software*. Recuperado de https://www.dreams.es/transformacion-digital/desarrolladores-paginas-web/patrones-de-arquitectura-de-software

- **Microservicios**

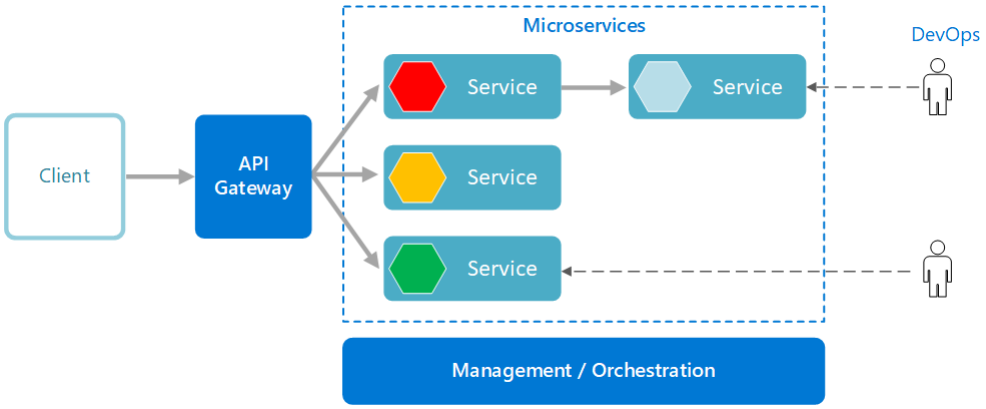
Esta arquitectura permite que las aplicaciones sean independientes, flexibles y manejables durante el tiempo, su pequeña escala permite que el mantenimiento sea más sencillo, una productividad mejorada, mayor tolerancia a fallos, mejor alineación comercial.

Según (Lewis & Fowler, 2014), el estilo arquitectónico de microservicio es un enfoque para desarrollar una sola aplicación como un conjunto de

pequeños servicios, cada uno ejecutándose en su propio proceso y comunicándose con mecanismos livianos, a menudo una API de recursos HTTP. Estos servicios se basan en capacidades comerciales y se implementan de forma independiente mediante maquinaria de implementación totalmente automatizada. Hay un mínimo indispensable de gestión centralizada de estos servicios, que pueden estar escritos en diferentes lenguajes de programación y utilizar diferentes tecnologías de almacenamiento de datos.

Las ventajas que nos proporciona esta arquitectura es que los microservicios son fáciles de escalar en torno a las capacidades empresariales, altamente mantenible y comprobable, más aún el despliegue independiente sin afectar a otros procesos.

Gráfico 2: *Arquitectura de microservicios*



Fuente: Microsoft. *Microservice architecture style*. Recuperado de https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices

Según (Blancarte Iturralde, 2020), "un microservicio es un pequeño programa que se especializa en realizar una pequeña tarea y se enfoca únicamente en eso, por ello, decimos que los microservicios son altamente cohesivos, pues

toda las operaciones o funcionalidad que tiene dentro está extremadamente relacionadas para resolver un único problema".

- **Arquitectura en capas**

  El estilo de arquitectura en capas está compuesto por niveles, se encuentran organizados los componentes en capas horizontales. Esta forma de trabajo permite descomponer en subtareas logrando un nivel de abstracción de alto nivel.

  Una de las características resaltantes es la separación de preocupaciones (SoC) entre los componentes. Aquellos componentes que integran a una determinada capa tienden a encargarse únicamente de la lógica de negocio de la capa a la que pertenece.

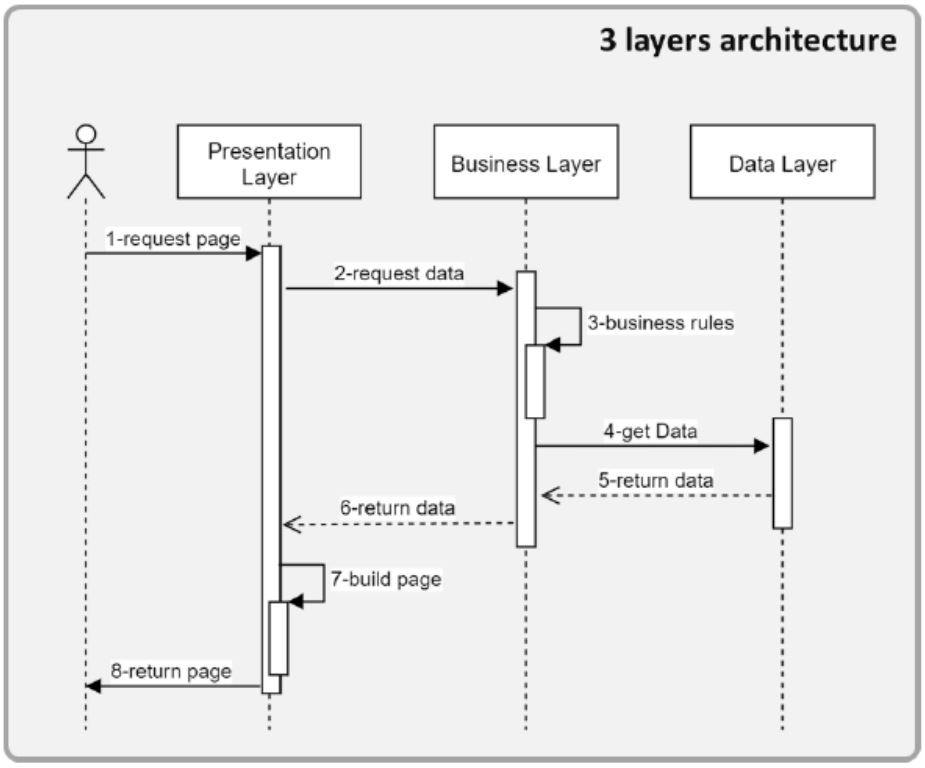
  Las capas usadas con frecuencia en un sistema de información son:

  - Capa de presentación (UI - Interfaz de usuario): Encargada de crear la interfaz gráfica del usuario.
  - Capa de negocio (Servicio): Contiene la lógica de negocio y las operaciones de alto nivel.
  - Capa de acceso a datos (Persistencia): Encargada de los datos, comunicación con los sistemas gestores de bases de datos como Oracle, PostgreSQL, SQL Server, MySQL y MongoDB.

  El orden jerárquico de las capas es muy importante, es posible saltarnos a una capa más abajo; sin embargo, esta suele ser un error, debido a que crearía una desorganización sobre el flujo de comunicación, lo que nos conduce a las

malas prácticas (espagueti), y en el transcurso del tiempo el mantenimiento se vuelve engorroso.

Gráfico 3: *Diagrama de secuencia de la ejecución en una arquitectura de 3 capas*



Fuente: Blancarte, O. (2020). *Introducción a la arquitectura de software.* Recuperado de https://reactiveprogramming.io/books/software-architecture/es

(Blancarte Iturralde, Introducción a la arquitectura de software - Un enfoque práctico, 2020) define las ventajas y desventajas que tiene una arquitectura en capas:

Ventajas

- **Separación de responsabilidades:** Cada una de las capas tienen responsabilidades independientes logrando la separación de preocupaciones (SoC).

- **Fácil de desarrollar:** Su implementación es sencillo, es bien conocido por muchos desarrolladores ya que en su mayoría de las aplicaciones la utilizan.
- **Fácil de probar:** La etapa de las pruebas es flexible, debido a que se puede ir realizando de manera individual por capas.
- **Fácil de mantener:** En caso de que ocurra errores, su identificación es inmediata, gracias a la estructura que se maneja por capas, esto nos permite corregir para aplicar algún cambio en tiempos bastante óptimos.
- **Seguridad:** Debido a que las capas están separadas nos permite configurar en servidores aislados cada una en diferentes subredes, asegurando la fiabilidad de los datos.

Desventajas

- **Performance:** La comunicación entre capas genera un alto degrado a nivel de performance y es una situación que depende de la estabilidad de la red o internet para llevar a cabo durante todo el proceso.
- **Escalabilidad:** En su mayoría de las aplicaciones que siguen este patrón, están orientados a ser monolíticas, ocasionando que estas sean difíciles de escalar en el tiempo, aunque se tiene las opciones de replicar en diferentes nodos esta provocaría un escalado monolítico.
- **Complejidad de despliegue:** El despliegue en la arquitectura de capas, sigue una orientación de abajo hacia arriba, esta crea una dependencia durante el despliegue, así mismo, si la aplicación tiende a ser monolítico

y exista cualquier cambio en la aplicación podría requerir el despliegue completo.

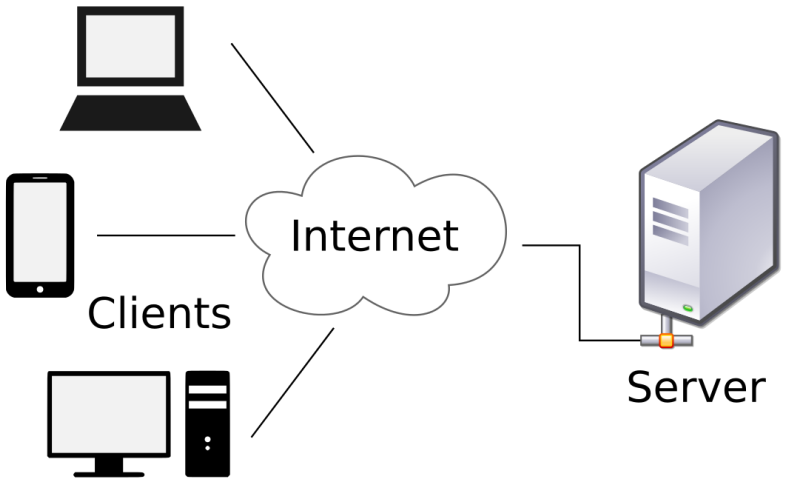
- **Stack tecnológico:** En lo general las capas de la aplicación son construidas usando la misma tecnología, si bien esto no es una regla, debido a que es posible usar diferentes tecnologías por capa.
- **Tolerancia a los fallos:** La arquitectura de capas sigue una dependencia, en caso de que ocurra una falla en una determinada capa, las otras capas superiores sufren en cascada el error repercutido.

- **Cliente – Servidor**

  La arquitectura está compuesta por el servidor y el cliente, el servidor que está encargado de alojar, gestionar los recursos y servicios, las cuales son consumidos por los clientes.

  La comunicación entre el servidor y el cliente por lo general es TCP/IP, lo que permite que la comunicación sea continua y bidireccional, con la finalidad que el cliente pueda enviar y recibir datos del servidor y viceversa.

  Gráfico 4: *Arquitectura Cliente - Servidor*



Fuente: *Estructura Cliente – Servidor*. Recuperado de https://chsosunal20181913034worpress.wordpress.com/2018/05/19/estructura-cliente-servidor/

- **Monolítico**

  Las aplicaciones monolíticas tienen como característica el uso de una base de código única para sus servicios o funcionalidades. En lo general se presentan algunos inconvenientes en esta arquitectura, ya que toda su funcionalidad está acoplada y en el tiempo es difícil de mantener.

  Resaltar que las aplicaciones monolíticas no necesariamente son aplicaciones grandes y que realizan infinidad de tareas, se puede considerar un monolítico a pesar de tener una sola clase o miles, el estilo monolítico no se define por la cantidad de archivos o clases, más bien por lo que es autosuficiente, es decir, que tiene toda la funcionalidad para operar por sí mismo sin la dependencia.

  Ventajas en las arquitecturas monolíticas:

  - **Desarrollo inicial rápido:** El equipo de desarrollo puede enfocarse en la implementación de todas las funcionalidades, sin considerar las integraciones entre distintos componentes que se encuentran en otros servidores.
  - **Identificar código fuente:** Navegar por el código fuente es sencillo debido a que se encuentra en un único paquete.
  - **Fácil despliegue:** Desplegar el paquete de aplicación dentro de un servidor, que efectivamente es sencillo ya que los IDEs actuales nos permite generar de forma automatizada.

- **Escalamiento horizontal**: Es posible desplegar varias copias de la misma aplicación en varios servidores.

Desventajas en las arquitecturas monolíticas:

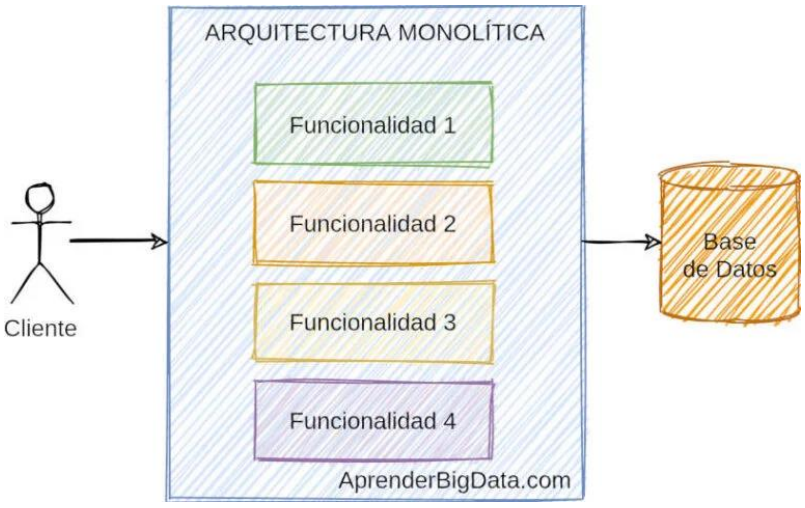
A medida que la aplicación va creciendo, nos enfrentamos a diversos problemas que se vuelven cada vez más significativos.

- **Dificultad de adaptación:** La adaptación de nuevos desarrolladores al equipo de trabajo tiende a ser lenta si el monolito tiene gran tamaño, esto conlleva a que la velocidad de desarrollo del equipo se vea afectada. Más aún, si no existe una buena metodología de trabajo, la aplicación cada vez más será más difícil de entender y de modificar, lo cual conlleva que cualquier nuevo cambio por parte de los nuevos desarrolladores puedan influir negativamente en la calidad del código (Richardson, Pattern: Monolithic Architecture, s.f.).
- **Incremento en el tiempo de despliegue:** A medida que el monolítico va creciendo, el tiempo de despliegue se verá afectado lo cual tendrá un impacto negativo en el avance de los desarrolladores, ya que los periodos de tiempo serán más dilatados.
- **Inconvenientes al escalar eficientemente la aplicación:** Si bien es cierto que es posible escalar horizontalmente en diferentes servidores, existen inconvenientes en los siguientes enfoques:
  - Debido a que todas las instancias tengan acceso a la totalidad de la data, dificulta el cacheo de esta, forzando un incremento en el acceso a la memoria, ocasionando re-lentitud en la aplicación.

- En caso de existir un error dentro de un componente especifico, podría afectar a toda la aplicación.
- Si un determinado componente o servicio posee más carga de trabajo y se desea tener varias instancias únicamente de esta, no sería posible con esta arquitectura.

- **Componentes con distintos requerimientos:** Los componentes del software necesitan distintos tipos de recursos, al tener un monolito no es posible escalar verticalmente cada componente independientemente según sus necesidades (Richardson, Pattern: Monolithic Architecture, s.f.).
- **Difícil de adaptar a nuevas tecnologías:** Una aplicación basada en la arquitectura monolítica está implementada con el mismo stack tecnológico desde el inicio de esta. (Chris Richardson)

Gráfico 5: *Arquitectura monolítica*



Fuente: Aprender Big Data. (2022). *Arquitectura de microservicios: Introducción.* Recuperado de https://aprenderbigdata.com/microservicios/

- **Modelo – Vista – Controlador**

Según (Harrop, 2005), es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos (Universidad de Alicante, s.f.).

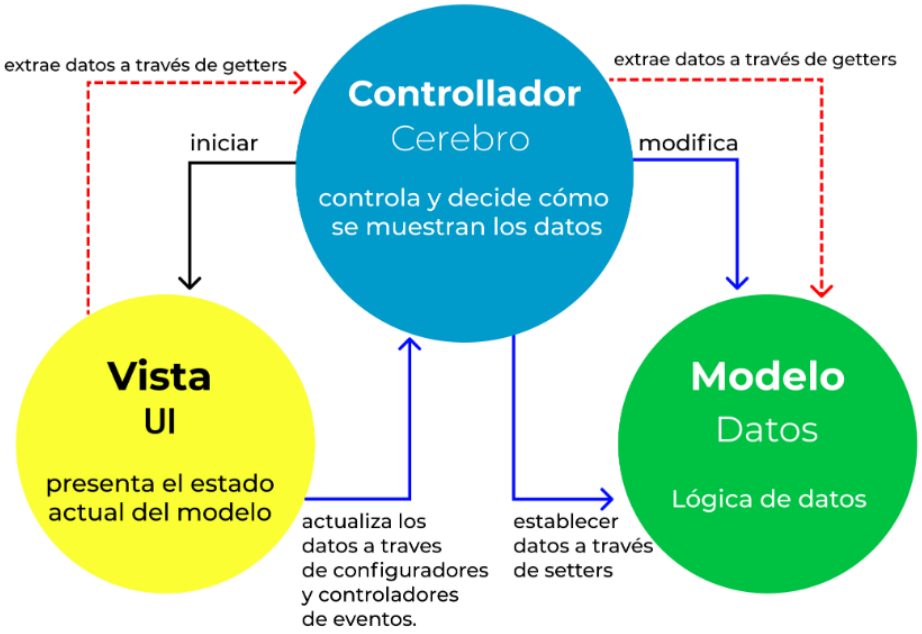
El propósito de este patrón es simplificar la implementación de aplicaciones de acuerdo con las peticiones de los usuarios y los datos a desplegar.

El patrón Modelo – Vista – Controlador (MVC) separa la funcionalidad en tres distintos conceptos: el Modelo, que es una representación de los datos y el estado de la aplicación, la Vista, que interactúa con el usuario ofreciendo una visualización de la información y recogiendo señales por distintos dispositivos o interfaz, y el Controlador, que es el intermediario y traduce los cambios de usuario en cambios al modelo o a la vista, notificando los cambios de estado. Normalmente, existen varias vistas y controladores asociados a un único modelo (Albertos, 2018, p. 5).

Este estilo de arquitectura es un modelo muy maduro, ya que se ha verificado su validez en diferentes aplicaciones, lenguajes y plataformas de desarrollo.

- El Modelo: Representa los datos que opera el sistema, la lógica de negocio y los mecanismos de persistencia.
- La Vista: Dispone la información que se envía al cliente y los mecanismos de interacción.
- El Controlador: Es la parte intermedia en la comunicación entre el modelo y la vista, gestiona el flujo de información entre ellos y las transformaciones para adecuar los datos a las necesidades de cada uno.

Gráfico 6: *Arquitectura MVC*



Fuente: Hernandez, R. (2021). *El patrón modelo-vista-controlador: Arquitectura y frameworks explicados.* Recuperado de https://www.freecodecamp.org/espanol/news/el-modelo-de-arquitectura-view-controller-pattern/

2.2.3. Patrones arquitectónicos

Según (Taylor y otros, 2009), es un conjunto de decisiones de diseño arquitectónico específicas que es aplicable a un problema de diseño recurrente, y parametrizado para considerar distintos contextos de desarrollo de software en los que se presenta el mismo problema.

Por lo tanto, un patrón arquitectónico está enfocado a una solución general y que esta sea reutilizable en los diferentes problemas comunes que se identifica dentro de la arquitectura de software. De esta forma proporcionamos agilidad en el desarrollo además de obtener una performance óptima.

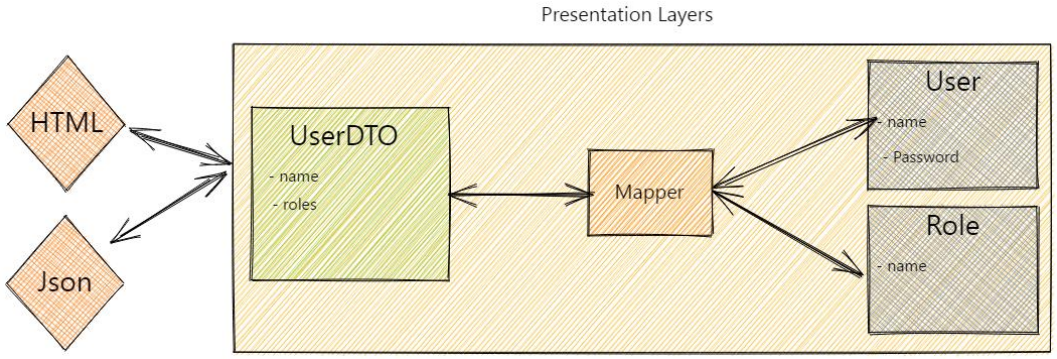
Veamos algunas definiciones respecto a los patrones arquitectónicos:

Data Transfer Object (DTO)

Según (Blancarte Iturralde, 2020), el patrón DTO tiene como finalidad la creación de objetos planos (POJO) con una serie de atributos que puedan ser enviados o recuperados del servidor en una sola invocación, de tal forma que un DTO puede contener información de múltiples fuentes o tablas y concentrarlas en una única clase simple.

Normalmente los DTO son estructuras de datos planas que no contiene lógica empresarial. Solo contienen almacenamiento, accesores y eventualmente, métodos relacionados con la serialización o el análisis.

Gráfico 7: *Interacción entre los componentes - DTO*



Fuente: Baeldung. (2022). *The DTO Pattern (Data Transfer Object)*. Recuperado de https://www.baeldung.com/java-dto-pattern

Si observamos en la imagen anterior, se está utilizando el Mapper que nos permite obtener los datos desde las entidades y pasarlos al DTO, en ambientes empresariales se hace uso del ModelMapper, esto hace que el mapeo de objetos sea sencillo y automatizado según las convenciones definidas.

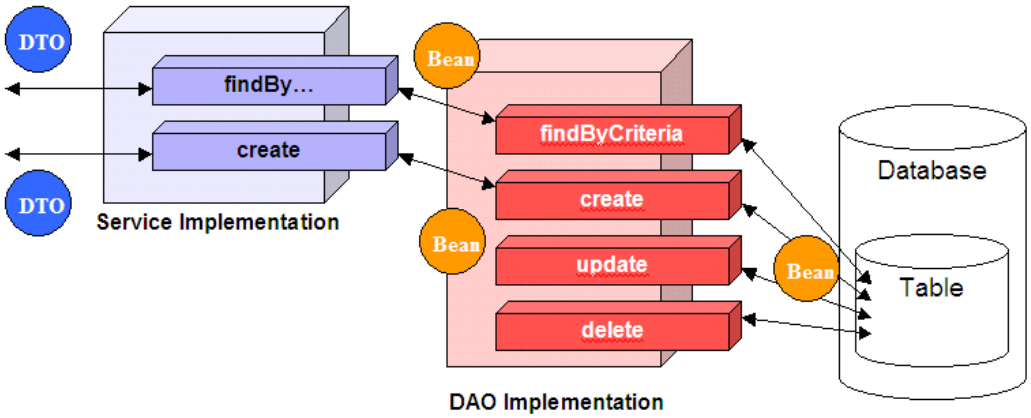
Data Access Object (DAO)

El patrón DAO, se usa para separar la lógica de persistencia de datos en una capa separada. De esta manera, el servicio permanece completamente a oscuras sobre como realizan las operaciones de bajo nivel para acceder a la base de datos (Shubham, 2022).

Es un patrón estructural que nos permite aislar la capa de aplicación/negocio de la capa de persistencia (generalmente una base de datos relacional, pero podría ser cualquier otro mecanismo de persistencia) utilizando una API abstracta (Baeldung, 2021).

Es uno de los patrones más utilizados en la actualidad, ya que es fácil de implementar y proporciona grandes beneficios, además esta funcionalidad se complementa perfectamente con el patrón de diseño Abstract Factory, en casos que es necesario conectarnos a más de un recurso que podría ser bases de datos relacionales, NoSQL, XML, archivos planos, servicios web o REST, etc.

Gráfico 8: *Implementación DAO*



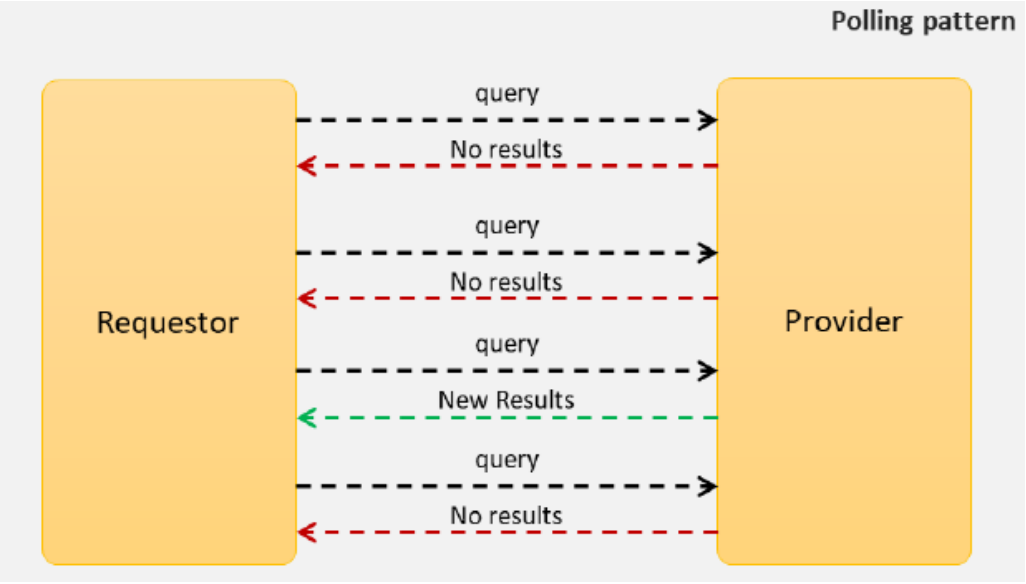
Fuente: *Spring boot – DTO vs DAO.* Recuperado de https://blog.hillpig.top/java-spring-dto-dao/

Polling

La integración con sistemas de terceros generalmente es requerida para realizar consultas o enviar datos que se nos solicita; sin embargo, si nuestra aplicación necesita saber si existe algún cambio en el sistema externo y deseamos saberlo. Una de las formas es aplicar mediante el Polling.

Según (Oscar Blancarte, 2020), describe que el Polling, consiste en realizar una serie de consultas repetitivas y con una periodicidad programada, en búsqueda de nueva información, de esta forma, el sistema interesado tendrá que ir al servidor y preguntar si hay nuevas actualizaciones, si las hay, el servidor las retornará, en otro caso, el cliente seguirá preguntando cada x tiempo hasta que encuentre nuevas actualizaciones.

Gráfico 9: *Patrón Polling*



Fuente: Blancarte, O. (2020). *Polling.* Recuperado de https://reactiveprogramming.io/blog/es/patrones-arquitectonicos/polling

- Requestor: Componente que realiza las consultas recurrentes.

- Provider: Servidor externo donde se encuentra alojado los datos que pueden cambiar.

Como se observa la imagen, las peticiones son muy recurrentes, así que el patrón Polling se debe utilizar con cuidado, podría llevar a un desgaste en el performance del Provider.

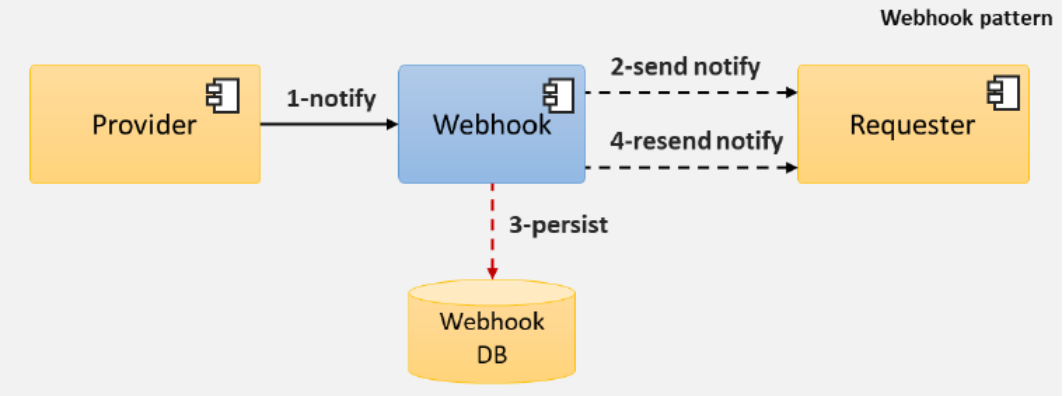
WebHook

Anteriormente observamos como el patrón Polling nos permitía realizar consultas recurrentes por el Requestor, ahora lo que nos gustaría es que el Provider sea quien nos notifique al instante cuando existe un cambio. En este contexto tratamos con este patrón WebHook

"Es un sistema de comunicación automático entre aplicaciones. Lo que hacen es aportar una solución sencilla para el intercambio de datos entre aplicaciones" (NewsMDirector, 2020).

En este patrón la responsabilidad de notificar los cambios relevantes pasa a ser del Provider, y el Requester debe contar con un mecanismo o componente que le permita recibir estas notificaciones. En este proceso, es importante considerar el número de reintentos, porque es posible que durante el envío de la notificación el destinatario no se encuentra disponible, así que reenviar estas notificaciones cada un cierto periodo por N reintentos nos permite asegurar que la notificación llega a su destino. Resaltar que debemos evaluar la cantidad de reintentos a realizar de acuerdo con el proceso que se está llevando.

Gráfico 10: *Arquitectura de reintentos*



Fuente: Blancarte, O. (2017). *Webhook una alternativa al Polling*. Recuperado de https://www.oscarblancarteblog.com/2017/09/11/webhook-una-alternativa-al-polling/

Algunas ventajas que se puede describir son:

- Ahorro de recursos y tiempo
- Eliminación de los retrasos
- Velocidad de las llamadas

Actualmente existen plataformas integradas con este patrón arquitectónico como es Github, Mailchimp, Shopify, Twilio, Sendgrid, etc.

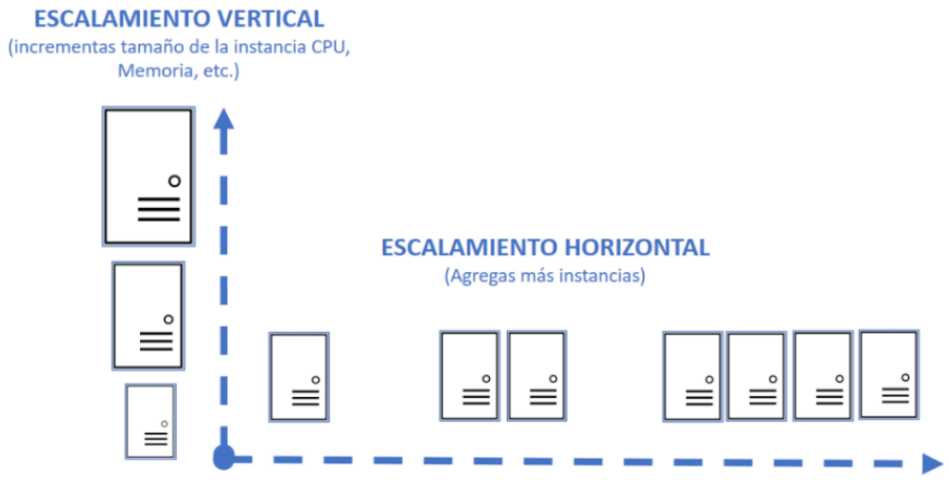
Load Balance

Un balanceador de carga distribuye el tráfico entrante de aplicaciones entre varios destinos, tales como instancias EC2, en varias zonas de disponibilidad. Esto aumenta la disponibilidad de la aplicación. Puede agregar uno o varios agentes de escucha al balanceador de carga (AWS, s.f.).

En la práctica nos encontramos con algunas limitaciones en un contexto donde es imposible que un solo servidor sea capaz de atender todas las solicitudes, como primera solución se podría aplicar el escalamiento vertical (aumentar más

recursos), pero sería una solución temporal ya que existe un límite de crecimiento en el hardware. Una solución más eficiente a este tipo de problemas es aplicar el escalamiento horizontal, que consiste en agregar más de un servidor a nuestro cluster, con el objetivo de dividir la carga de trabajo equitativo en todos los servidores.

Gráfico 11: *Escalamiento vertical y horizontal*



Fuente: Navarro, H. (2020). *Escalando aplicaciones digitales.* Recuperado de https://elementi.me/escalando-aplicaciones-digitales/

Access Token

Los tokens de acceso son lo que usan las aplicaciones para realizar solicitudes de API en nombre de un usuario. El token de acceso representa la autorización de una aplicación específica para acceder a partes específicas de los datos de un usuario (Okta, s.f.).

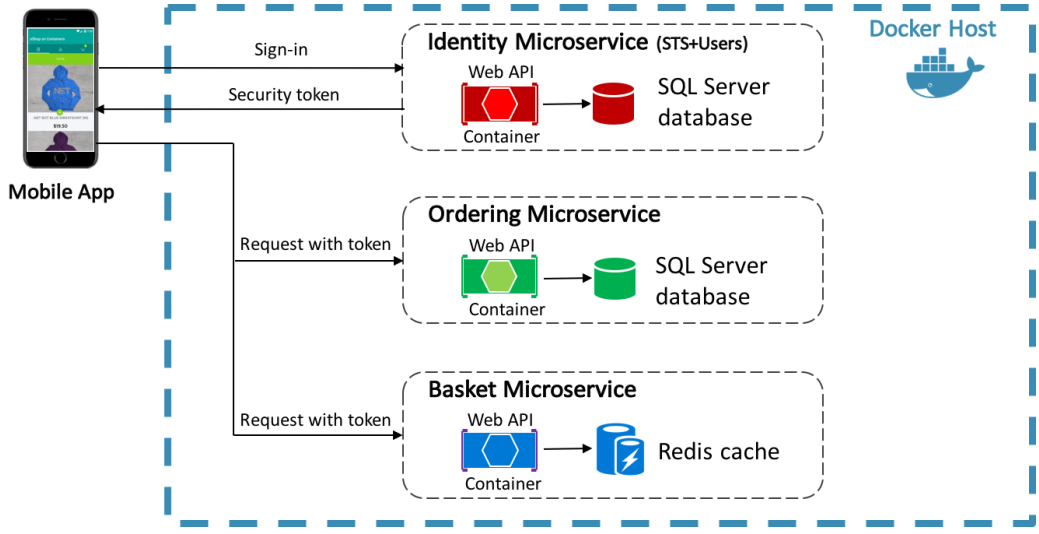
Este patrón nos permite que los clientes pueden invocar de forma segura a la API web protegidas. Las API Rest utilizan tokens de acceso para realizar la autenticación y la autorización.

JSON Web Token JWT: Es un estándar abierto que define una forma compacta y autónoma de trasmitir la información de forma segura entre las partes como un objeto JSON. Esta información se puede verificar y confiar porque está firmada digitalmente. Los JWT se pueden firmar usando un secreto (con el algoritmo HMAC) o un par de claves pública/privada usando RSA o ECDSA (JWT, s.f.).

La estructura de los JWT está confirmada por tres partes separadas por puntos, a continuación, se definen:

- Headers: Representa la cabecera donde se almacena informaciones como el tipo de token y algoritmos cifrados.
- Payload: Es la capa que engloba informaciones como el identificador del usuario, nombres y otros datos que se cree necesario del usuario, así lo mismo la codificación de las transacciones para identificar en la comunicación con los servicios externos.
- Firma: Es la firma digital, es un Hash generado a partir de la Payload, la cual tiene como objetivo validar la autenticidad del token.

Gráfico 12: *Autorización por token de acceso*



Fuente: Microsoft. (2022). *Autenticación y autorización.* Recuperado de https://docs.microsoft.com/es-es/dotnet/architecture/maui/authentication-and-authorization

Log Aggregation

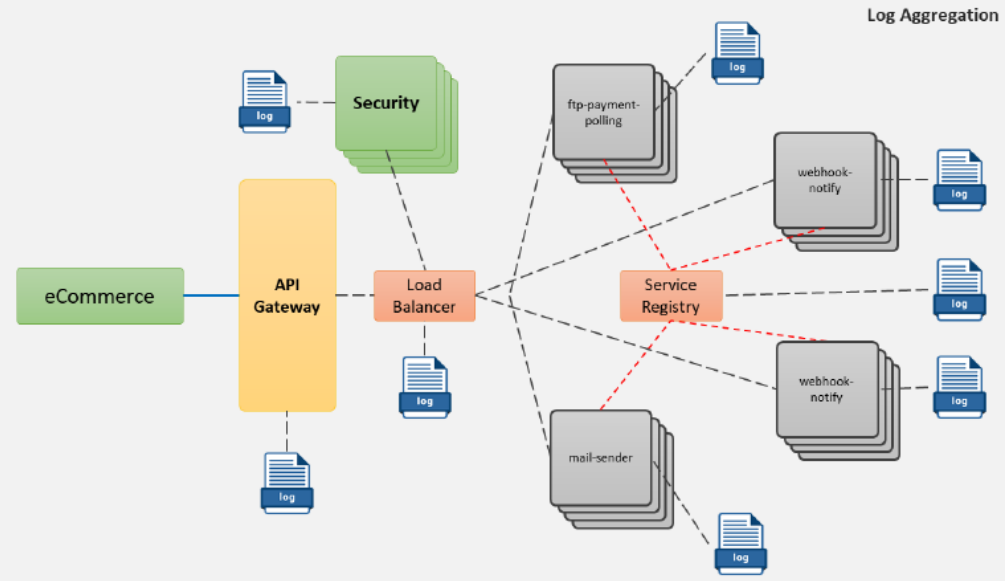
La centralización de las trazas de una aplicación es una de las actividades importantes que se debe darle prioridad, que gracias a estos registros podemos identificar y resolver múltiples errores, porque nos da la claridad para realizar el seguimiento oportuno de la aplicación.

La agregación de registros es una función de software que consolida los datos de registro de toda la infraestructura de TI en una única plataforma centralizada donde se puede revisar y analizar. Las herramientas de software de agregación de registros pueden admitir funciones adicionales, como la normalización de datos, la búsqueda de registros y el análisis de datos complejos. La agregación de registros es solo un aspecto de un proceso general de administración de registros que produce información en tiempo real sobre la seguridad y el rendimiento de las aplicaciones (Sumo Logic, s.f.).

Logback

Es una de las tecnologías con mayor uso en la actualidad dentro de la comunidad Java, es un reemplazo de su predecesor, Log4j. Logback ofrece una implementación más rápida, brinda más opciones de configuración y más flexibilidad para archivar archivos de registros antiguos.

Gráfico 13: *Logs distribuidos*



Fuente: Blancarte, O. (2020). *Log Aggregation*. Recuperado de https://reactiveprogramming.io/blog/es/patrones-arquitectonicos/log-aggregation

2.2.4. Patrones de diseño

Los patrones de diseño (design patterns) son soluciones habituales a problemas comunes en el diseño de software. Cada patrón es como un plano que se puede personalizar para resolver un problema de deseño particular de tu código (Refactoring Guru, s.f.).

Los patrones de diseño o design patterns, son una solución general, reutilizable y aplicable a diferentes problemas de diseño de software. Se trata de plantillas que identifican problemas en el sistema y proporcionan soluciones apropiadas a problemas generales a los que se han enfrentado los desarrolladores durante un largo periodo de tiempo, a través de prueba y error (Martinez Canelo, 2020).

Los tipos de patrones más usados se clasifican en tres categorías:

- Patrón creacional
- Patrón estructural

- Patrón de comportamiento

Gráfico 14: *Tipos de patrones de diseño*

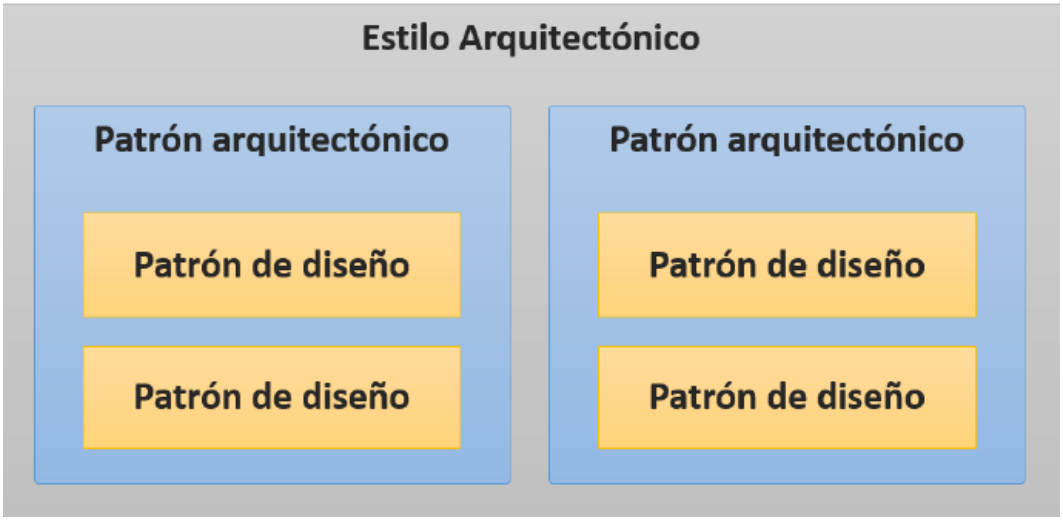
Fuente: Martínez, M. (2020). *¿Qué son los patrones de diseño de software?*. Recuperado de https://profile.es/blog/patrones-de-diseno-de-software/

2.2.5. Relación entre estilos, patrones y diseños arquitectónicos

Frecuentemente existe todavía la confusión de estos conceptos, debido que no existe definiciones concretas o no existe una línea especifica que define.

Según (Blancarte Iturralde, 2020), describe que "los estilos arquitectónicos son los de más alto nivel, y sirve como base para implementar muchos de los patrones arquitectónicos que conocemos, de la misma forma, un patrón arquitectónico puede ser implementado utilizando uno o más patrones de diseño". (p. 49)

Gráfico 15: *Relación entre patrones de diseño, patrones y estilos arquitectónicos*



Fuente: Blancarte, O. (2020). *Introducción a la arquitectura de software.* Recuperado de https://reactiveprogramming.io/books/software-architecture/es

2.2.6. Microservicios

Según (Fugaro & Vocale, 2019) define que "La arquitectura de microservicios está basada en el concepto de que una aplicación puede contener una colección de servicios de bajo acoplamiento, independientes y atómicas; los cuales implementan capacidades de negocio".

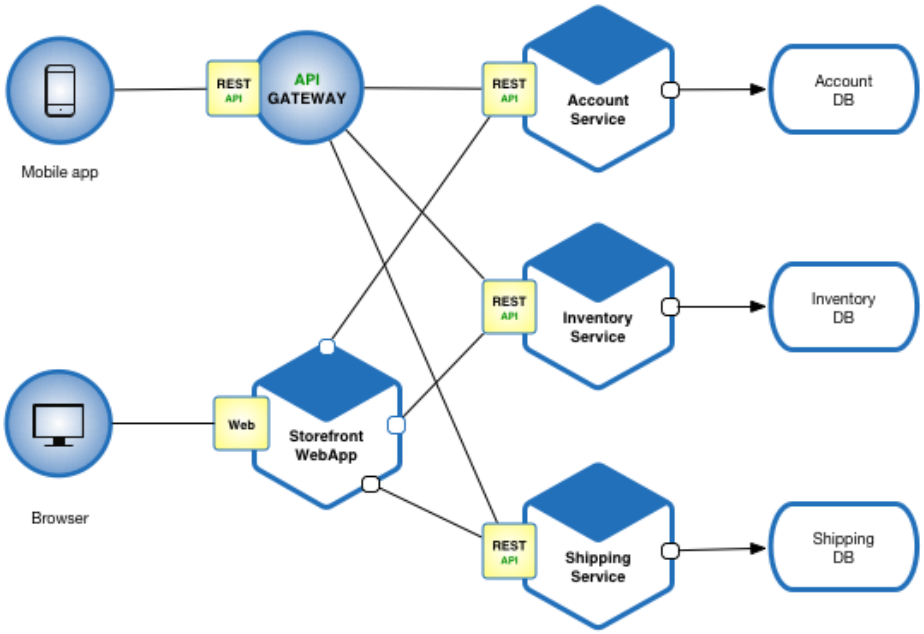
La arquitectura de microservicios está enfocada en crear soluciones con objetivos específicos y que estas sean independientes, flexibles con la capacidad de ejecutarse por sí mismos, al final que sean desplegadas independientemente (Indrasiri, 2018, p. 434).

La arquitectura de microservicios es un estilo arquitectónico que proporcionada beneficios al llevarse a cabo su implementación. Algunas de sus características son:

- Mantenibles y comprobables en el tiempo.
- Su despliegue es independiente y escalable

- La asignación de recursos es entorno a las capacidades de la organización
- Equipo de desarrollo pequeño

Gráfico 16: *Diseño de microservicios*



Fuente: Richardson, C. (2016). *What are microservices ?*. Recuperado de https://microservices.io/

2.2.7. Arquitectura de microservicios

Una arquitectura de microservicios es un tipo de arquitectura de aplicaciones en la que se desarrolla la aplicación como un conjunto de servicios. Este modelo proporciona el framework para desarrollar, desplegar y mantener diagramas y servicios de arquitectura de microservicios de forma independiente (Google Cloud, s.f.).

La arquitectura de los microservicios nos ofrece ciertos beneficios en las aplicaciones construidas en base a esto, como son:

- **Modular**
  - Desarrollo y despliegue independiente.

- Si existe algún fallo no afecta a otros servicios (aislamiento de fallos).

- **Escalable**
  - Adaptación a las necesitades de acuerdo con el rendimiento.
  - Capacidad de crear o eliminar instancias.
- **Versátil**
  - Implementación con diferentes tecnologías.
  - Elección de lenguajes de programación acuerdo a la experiencia del equipo de desarrollo.
- **Respuesta óptima**
  - Agilidad en el proceso de implementación.
  - Mantenimiento flexible en la integración de cambios.
- **Equipo especializado**
  - Equipos de desarrollo especializado en determinados microservicios.
  - Conocimientos centralizados con independencia en los equipos de desarrollo.
- **Reducir costos**
  - Las aplicaciones usan determinados servicios por tarea.
  - Los costos están orientados a la carga de uso del sistema.
- **Interoperabilidad**
  - Intercambio de información con servicios de terceros.
  - Flexibilidad en la integración de forma segura con servicios externos.

A igual que nos proporciona muchos beneficios, debemos considerar los siguientes puntos al elegir esta arquitectura:

- **Inversión inicial**
  - El tiempo de inicio es primordial para centralizar y distribuir los microservicios de acuerdo con las funcionalidades.
  - Definición de la arquitectura de solución basado a microservicios.
- **Gestión compleja**
  - La cantidad de microservicios es acorde a las funcionalidades del proyecto, sin embargo; al tener un gran número nos conlleva a tener dificultades en la administración y seguimiento de estas.
  - Es necesario contar con otras herramientas que nos permitan agilizar y tener mejor visión de los microservicios.
- **Perfil de desarrollador**
  - Amplia experiencia de los programadores en el desarrollo de los microservicios.
  - Conocimientos sobre el diseño, balanceo de carga, y latencias de red.
- **No uniformidad**
  - Al tener la ventaja de utilizar tecnologías variadas para la implementación de los microservicios, esta se puede volver difícil de gestionar y que no siga la uniformidad de las aplicaciones.
- **Costos en la implantación**
  - La implementación de los microservicios puede llevar a un costo alto según la distribución en la infraestructura.

La importancia de implementar esta arquitectura es que la pequeña escala y su relativo aislamiento pueden generar muchos beneficios adicionales, como:

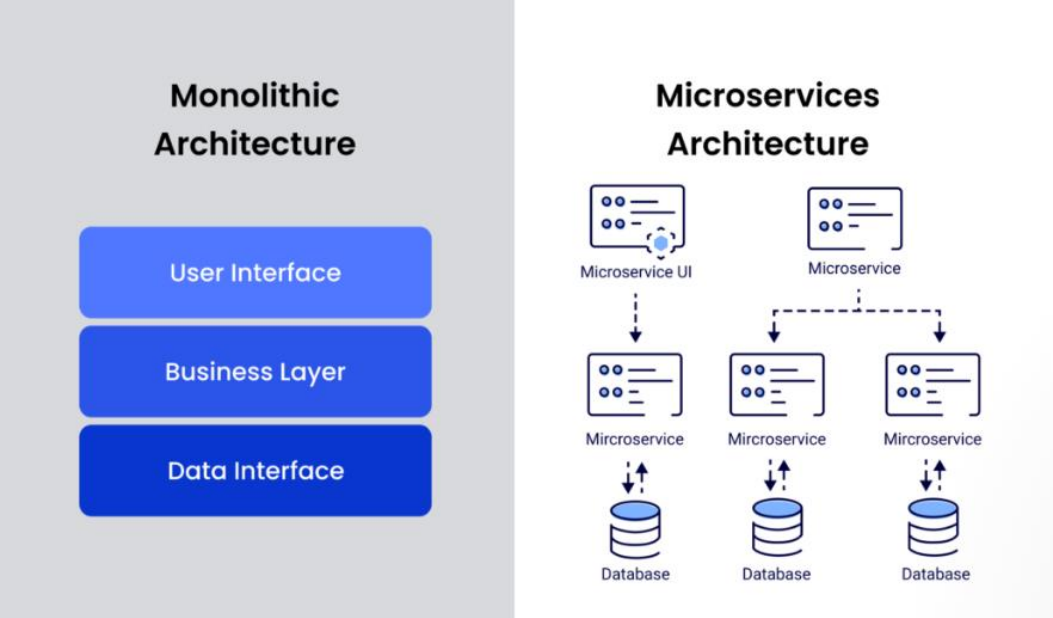
- **Mantenimiento sencillo:** Permite realizar mantenimiento o mejora de manera independiente sin afectar a los otros servicios.
- **Productividad mejorada:** El desarrollo es posible realizar al mismo tiempo con los otros equipos, además, el control de calidad y las puestas en producción son más eficientes.
- **Mayor tolerancia a fallas:** Si existe un error en un servicio determinado no afecta a los otros servicios desplegados.
- **Mejor alineación comercial:** Permite crear funcionalidades y/o componentes de manera independiente en cada microservicio y estas se pueden integrar en conjunto.

2.2.8. Microservicios vs Monolítico

Es importante resaltar la diferencia entre una arquitectura monolítica y microservicios, que al final nos permitirá elegir para la implementación de nuestras aplicaciones.

En referencia a la arquitectura monolítica podemos definir como un contenedor que contiene todos los componentes de software, en el transcurso del tiempo y a medida que la aplicación va creciendo con nuevas funcionalidades se tiene varias limitaciones, como inflexibilidad, falta de confiabilidad, dificultad para escalar, desarrollo lento, etc. Para solventar o evitar estos problemas nace la arquitectura de microservicios, que busca separar en unidades o componentes más pequeñas que pueda ser implementados y desplegados de forma independiente, obteniendo como resultado que el mantenimiento sea más fluido, flexible, una mayor disponibilidad y escalabilidad.

Gráfico 17: *Arquitectura Monolítica vs Microservicio*



Fuente: Middleware. (2021). *What are microservices? How microservices architecture works.* Recuperado de https://middleware.io/blog/microservices-architecture/

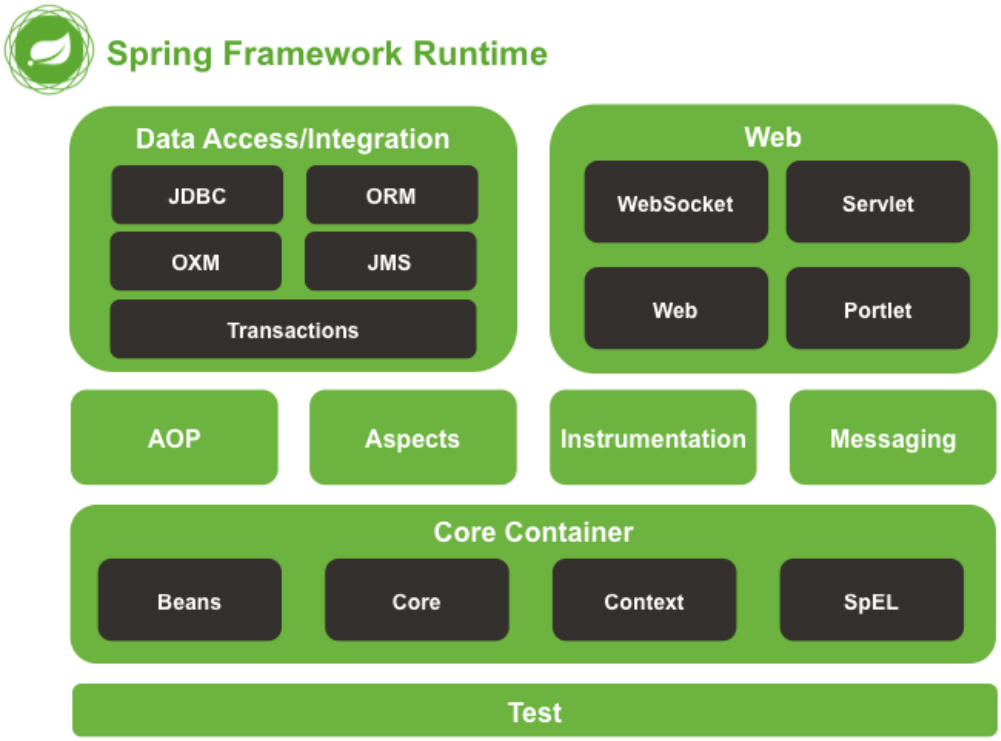
2.2.9. Stack tecnológico

Una de las características más importantes es la tecnología que usar, por ello, la arquitectura de software APH se basa al Framework Spring, con el objetivo de aumentar la productividad.

Spring Framework (Spring)

Es un marco para el desarrollo de aplicaciones de código abierto para la plataforma Java. Lanzado inicialmente en 2002 por Rod Johnson con la finalidad de facilitar el desarrollo del lado del servidor de Java y permitir a los equipos de desarrollo crear sus aplicaciones más rápidamente.

Gráfico 18: *Spring Framework Runtime*



Fuente: Spring. *Overview of Spring Framework.* Recuperado de https://docs.spring.io/spring-framework/docs/5.0.0.RC2/spring-framework-reference/overview.html

Spring Boot

Es una extensión de Spring para hacer que el desarrollo, las pruebas y la implementación sean más flexibles. Nos proporciona una gama de herramientas

que nos da la facilidad durante la construcción de los microservicios y que su configuración es automatizada.

Proporciona un conjunto de plantillas muy opuestas pero extensibles para crear varios proyectos basados a Spring a gran velocidad. Hace que sea realmente fácil crear aplicaciones Spring independientes con Tomcat integrado o un contenedor similar (Reclu IT, 2022).

Mientras Spring Framework se enfoca en proporcionar flexibilidad, Spring Boot busca reducir la longitud del código y simplificar el desarrollo de aplicaciones web. Al aprovechar las anotaciones y la configuración repetitiva, Spring Boot reduce el tiempo que lleva desarrollar aplicaciones. Esta capacidad lo ayuda a crear aplicaciones independientes con una sobrecarga de configuración mínima o casi nula (Bamboo Agile, 2022).

Componentes clave de Spring Boot Framework (JavabyKiran, s.f.)

- Spring Boot Starter: Es una de las principales características o componentes clave de Spring Boot Framework. La responsabilidad principal de Spring Boot Starter es combinar un grupo de dependencias comunes o relacionadas en dependencias únicas.
- Spring Boot AutoConfigurator: El marco tradicional de Spring requiere muchas configuraciones y, para ello, se utiliza la configuración XML o la configuración de anotaciones. La solución a este problema es Spring AutoConfigurator, su responsabilidad es reducir la configuración de Spring de forma automatizada.

- Spring Boot CLI: Es un software Spring Boot para ejecutar y probar aplicaciones Spring Boot desde el símbolo del sistema.

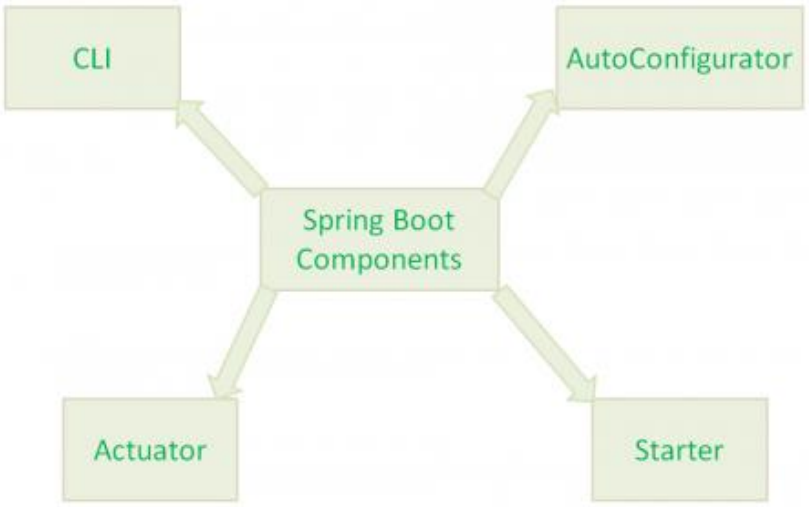
  Cuando ejecutamos aplicaciones Spring Boot usando CLI (Command Line Interface), entonces usa internamente los componentes de Spring Boot Starter y Spring Boot AutoConfigurator para resolver todas las dependencias y ejecutar la aplicación.

- Spring Boot Actuator: Proporciona muchas características, pero los principales son:

  - Proporcionar puntos finales de administración a aplicaciones Spring Boot.
  - Métricas de aplicaciones Spring Boot.

  Al ejecutar una aplicación web Spring Boot, Spring Boot Framework proporciona automáticamente el nombre de host como “localhost” y el número de puerto predeterminado como “8080”.

Gráfico 19: *Componentes de Spring Boot Framework*



Fuente: Rambabu, P. (2022). *Key Components and Internals of Spring Boot Framework.* Recuperado de https://www.digitalocean.com/community/tutorials/key-components-and-internals-of-spring-boot-framework

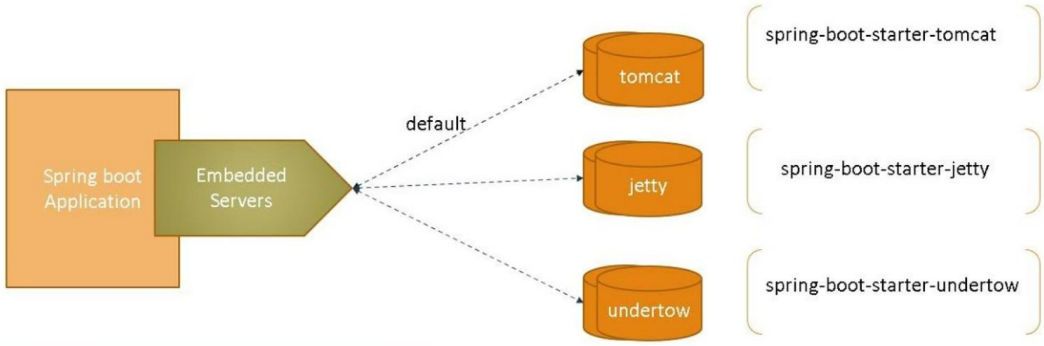
Características de Spring Boot (Rootstack, s.f.)

- Crea aplicaciones Spring independientes
- Incrusta Tomcat, Jetty o Undertow directamente (no es necesario implementar archivos WAR)
- Configura automáticamente Spring y bibliotecas de terceros siempre que sea posible.
- Proporciona funciones listas para producción, como métricas, controles de estado y configuración externalizada.
- Absolutamente sin generación de código y sin requisitos para la configuración XML

Apache Tomcat

El software de Apache Tomcat se desarrolla en un entorno abierto y participativo y se publica bajo la licencia Apache Version 2. El proyecto Apache Tomcat pretende ser una colaboración de los mejores desarrolladores de todo el mundo (Apache, 2022).

Gráfico 20: *Spring Boot Embedded Tomcat*



Fuente: Elaboración propia

Git

"Es un sistema de control de versiones distribuido gratuito y de código abierto diseñado para manejar todo, desde proyectos pequeños hasta proyectos muy grandes, con rapidez y eficiencia" (Git, 2022).

Linus Torvalds, el creador de Linux, desarrolló y lanzó Git en 2005. EL proyecto se emprendió originalmente porque los sistemas de control de versiones de código abierto disponibles en ese momento no estaban a la altura de los requisitos del desarrollo del kernel de Linux. Por un lado, el control de versiones para un esfuerzo de colaboración a gran escala requiere un mejor rendimiento del que eran capaces los sistemas disponibles (Tech Target, 2016).

Control de versiones

"El control de versiones ayuda a los desarrolladores a rastrear y administrar los cambios en el código de un proyecto de software. A medida que crece un proyecto de software, el control de versiones se vuelve esencial" (Kinsta, 2022).

Github

Es un servicio basado en la nube que aloja un sistema de control de versiones (VCS) llamado Git. Éste permite a los desarrolladores colaborar y realizar cambios en proyectos compartidos, a la vez que mantienen un seguimiento detallado de su progreso (Gustavo, 2022).

GitHub Packages

Es un servicio de alojamiento de paquetes de software de forma privada o pública y utilizar como dependencias en los proyectos.

Es una plataforma para hospedar y administrar paquetes, incluidos contenedores y otras dependencias. Paquetes de GitHub combina su código fuente y paquetes en un solo lugar para proporcionar administración y facturación de permisos integrados, para que pueda centralizar su desarrollo de software en GitHub (GitHub, s.f.).

Github Packages utiliza los comandos de herramientas de paquetes nativos con los que ya está familiarizado para publicar e instalar versiones de paquetes.

Tabla 1: *Soporte para registros de paquetes*

Lenguaje	Descripción	Package format	Package client
JavaScript	Administrador de paquetes de nodos	Package.json	npm
Rubí	Administrador de paquetes RubyGems	Gemfile	gem
Java	Herramienta de comprensión y gestión de proyectos Apache Maven	Pom.xml	mvn
Java	Herramienta de automatización de compilación de Gradle para Java	build.gradle o build.gradle.kts	gradle
.NET	Administración de paquetes NuGet para .NET	nupkg	dotnet CLI
N/A	Gestión de contenedores Docker	Dockerfile	docker

Fuente: GitHub. *Introduction to GitHub Packages.* Recuperado de https://docs.github.com/es/packages/learn-github-packages/introduction-to-github-packages

Según (Das, 2022) algunos beneficios que nos brinda los repositorios de alojamiento de paquetes de software son:

- El versionado de los softwares se realiza de forma sencilla y automática debido a que las herramientas actuales ya vienen integradas la gestión de repositorios.

- Agrupación de aplicaciones en paquetes, con el objetivo de administrar múltiples proyectos DevOps.
- La copia de seguridad de los diferentes proyectos se realiza de forma automatizada, debido a que el código se encuentra centralizadas en una determinada ubicación.
- Seguimiento y trazabilidad completa de los cambios introducidos en los proyectos.
- Colaboración automatizada con un gran número de desarrolladores que interactúan en tiempo real en diferentes proyectos.
- Punto de restauración sencilla a su versión anterior en caso de errores.
- Paquetes de software que están centralizados dentro de un repositorio es flexible y segura su entrega a los terceros, dado que estas no pueden realizar ninguna modificación.

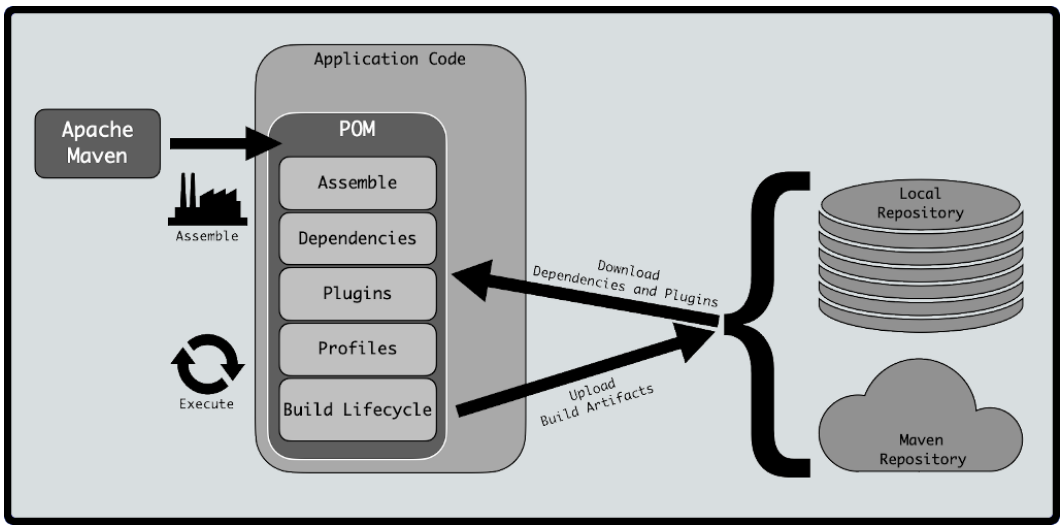
Apache Maven

Es una herramienta que estandariza la configuración de un proyecto en todo su ciclo de vida, como por ejemplo en todas las fases de compilación y empaquetado y la instalación de mecanismos de distribución de librerías, para que puedan ser utilizadas por otros desarrolladores y equipos de desarrollo (Yagüe, 2019).

Definición de Maven.

- Una herramienta de gestión de proyectos que fomenta, a través de la definición de un Project Object Model (el fichero POM que se define en cada uno de los proyectos o módulos), un conjunto de estándares que definen el ciclo de vida del proyecto (Yagüe, 2019).

Gráfico 21: *Proceso de ejecución Apache Maven*



Fuente: C.V. Guntur. (2020). *Understanding Apache Maven – Part 1 – Basics.* Recuperado de https://cguntur.me/2020/05/22/understanding-apache-maven-part-1-basics/

Java

Java es un lenguaje de programación ampliamente utilizado para codificar aplicaciones web. Ha sido una opción popular entre los desarrolladores durante más de dos décadas, con millones de aplicaciones Java en uso en la actualidad. Java es un lenguaje multiplataforma, orientado a objetos y centrado en la red que se puede utilizar como una plataforma en sí mismo. Es un lenguaje de programación rápido, seguro y confiable para codificarlo todo, desde aplicaciones móviles y software empresarial hasta aplicaciones de macrodatos y tecnologías del servidor (AWS, s.f.).

Logger

Un aspecto muy importante al crear arquitecturas orientadas a microservicios es el planteamiento de Logs, trazas o auditorias del sistema. Es importante tener en

cuenta que se trata de arquitecturas distribuidas, es decir, cada microservicio dispone de su sistema de log.

2.2.10. Arquetipos de Maven

Un arquetipo es una compilación que genera un proyecto basado en una plantilla. Está creado para facilitar las construcciones. En el presente proyecto estará enfocado al desarrollo de microservicios en base a estos arquetipos.

Arquetipos Maven

Se define como una plantilla reutilizable que nos permite crear nuevos proyectos bajo las características y estructuras definidas.

Un arquetipo, es un patrón o modelo sobre el que se pueden desarrollar todas aquellas tareas que son de un mismo tipo. Puede decirse que son plantillas, parametrizadas o configuradas para utilizar determinadas tecnologías que los desarrolladores utilizan como base para escribir y organizar el código de la aplicación (Garcia Puebla, 2008).

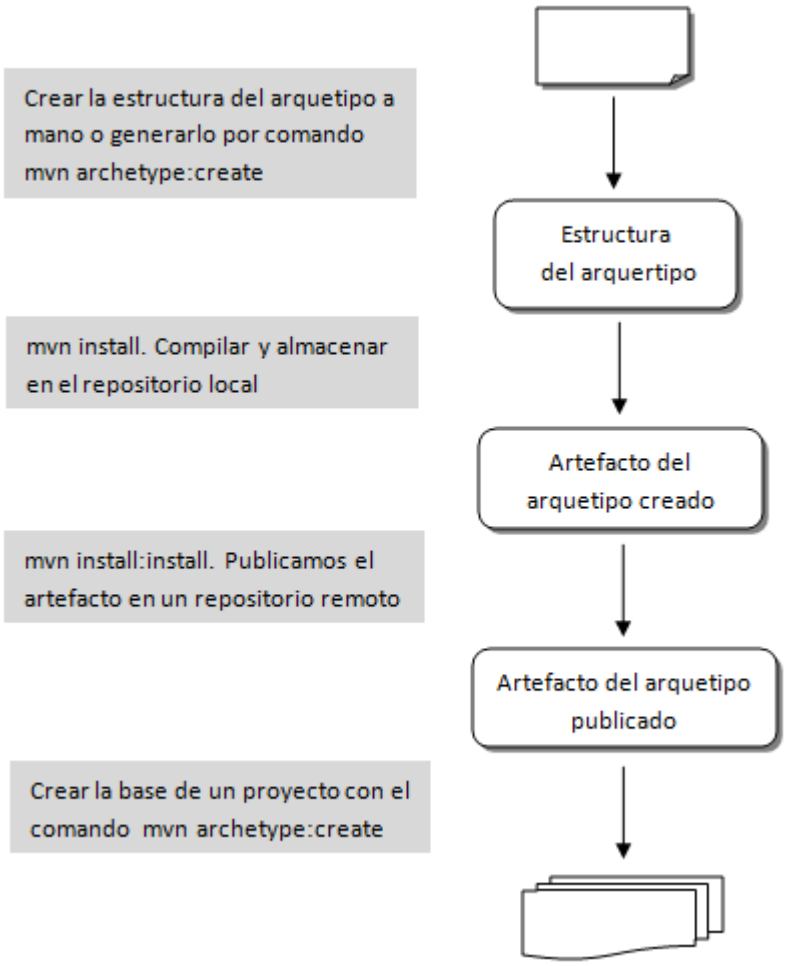
Todos los proyectos que involucren ciertas tecnologías portan de una base (arquetipo, plantilla o esqueleto configurado) común, tienen ventajas evidentes:

- Uniformidad entre distintos desarrollos que utilizan las mismas tecnologías.
- Reutilización y construcción de unos arquetipos como suma de otros.
- Estandarización de los proyectos dentro de una organización.
- Reducción del tiempo necesario (inicio de un nuevo proyecto) para la construcción de los diversos servicios.

Ciclo de vida de un arquetipo Maven

En el siguiente gráfico se representa la secuencia de la creación y uso de un arquetipo.

Gráfico 22: *Ciclo de vida de un arquetipo de Maven*



Fuente: García, P. I. (2008). *Arquetipos de Maven.* Recuperado de https://www.adictosaltrabajo.com/2008/06/09/creararquetiposmaven/

2.2.11. Artefactos de Maven

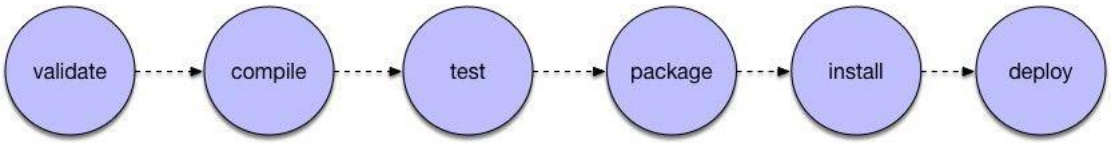
Dentro de los artefactos se define ciertos componentes reusables en cualquier proyecto de microservicios, estos se caracterizan por la disponibilidad permanente

y que en el tiempo pueden ser modificables de acuerdo con los nuevos requerimientos.

Artefacto de Maven, es una abstracción sobre el concepto de bloque de código reutilizable.

Según (Alvarez Caules, 2018) el ciclo de vida de un artefacto de Maven se representa como:

Gráfico 23: *Ciclo de vida de un artefacto de Maven*

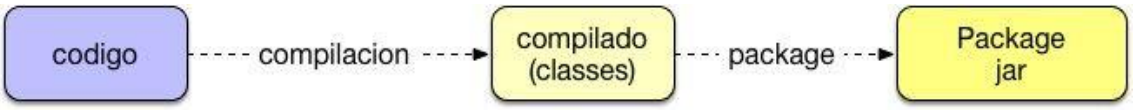


Fuente: Caules, C. A. (2018). ¿Qué es un Java Maven Artifacto?. Recuperado de https://www.arquitecturajava.com/que-es-un-java-maven-artifact/

- **Validate:** Simplemente comprueba que el proyecto tiene la estructura correcta y los ficheros están donde tienen que estar.
- **Compile:** Nos compila el código.
- **Test:** Se encarga de pasar las pruebas unitarias.
- **Package:** Recordemos que compilar nuestro código y generar los archivos .class es muy diferente que generar un empaquetado que se pueda reutilizar.

  Nota: Compilar no es lo mismo que empaquetar.

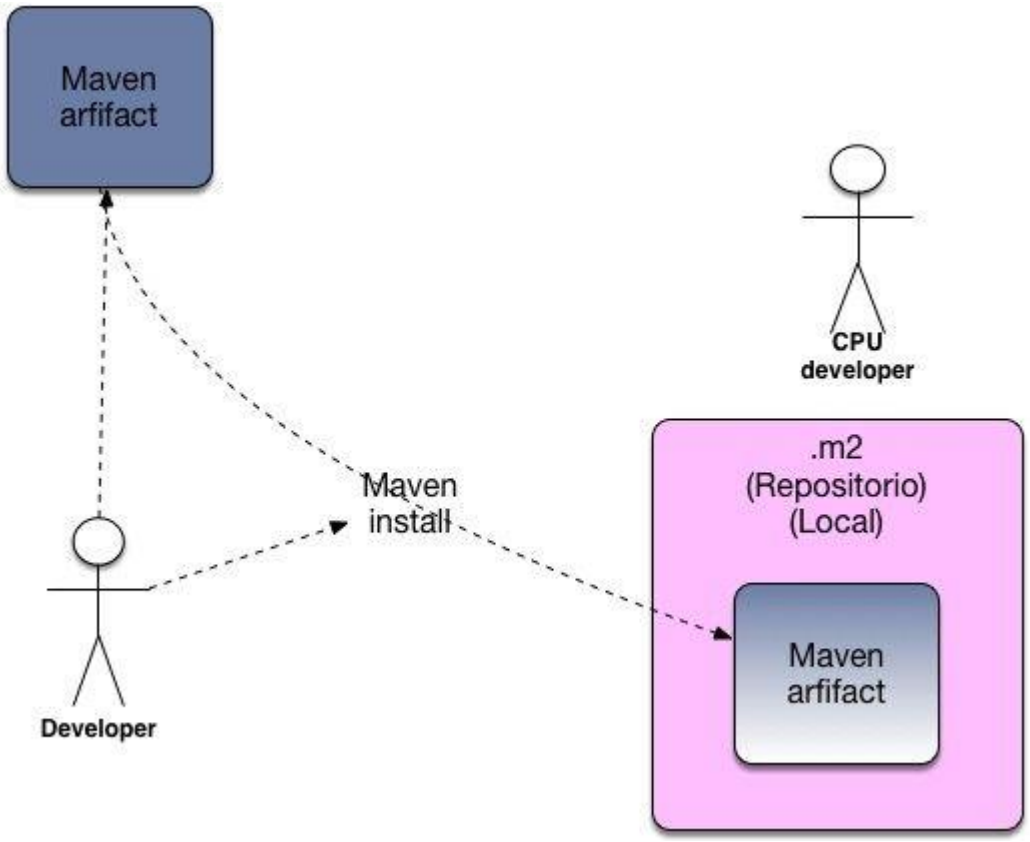
Gráfico 24: *Empaquetar un artefacto Maven*



Fuente: Caules, C. A. (2018). ¿Qué es un Java Maven Artifacto?. Recuperado de https://www.arquitecturajava.com/que-es-un-java-maven-artifact/

- **Install:** Proceso de instalación del artefacto Maven a un repositorio Maven.

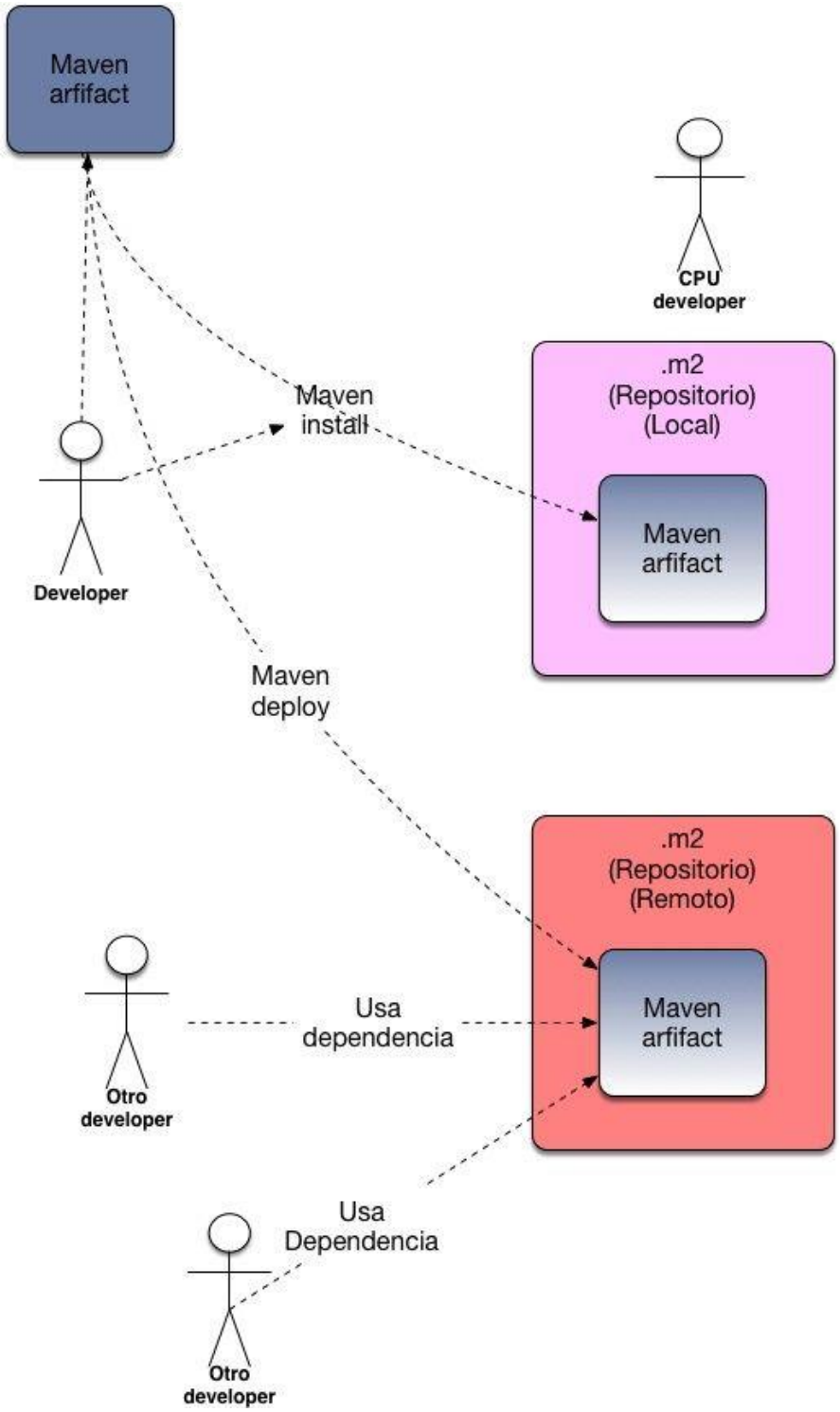
Gráfico 25: *Proceso de instalación del artefacto Maven*



Fuente: Caules, C. A. (2018). ¿Qué es un Java Maven Artifacto?. Recuperado de https://www.arquitecturajava.com/que-es-un-java-maven-artifact/

- **Deploy:** Realizado la fase de instalación de nuestro artefacto Maven al repositorio local específicamente en la carpeta .m2, si queremos que nuestro artefacto sea utilizado por otros desarrolladores necesitaremos realizar el Maven deploy que nos instalará en un repositorio de Maven remoto al que otros usuarios podrán acceder.

Gráfico 26: *Proceso de despliegue del artefacto Maven*



Fuente: Caules, C. A. (2018). ¿Qué es un Java Maven Artifacto?. Recuperado de https://www.arquitecturajava.com/que-es-un-java-maven-artifact/

2.3. Definición de términos

2.3.1. Arquitectura de software APH

La arquitectura de software APH (Arquitectura de Programación Hiper), es una extensión de la arquitectura backend, que nos permite desarrollar microservicios de una manera automatizada, mejorando significativamente la estandarización y la productividad en la integración de la lógica de negocio.

2.3.2. Patrón

Un patrón nos brinda una solución a un problema general que ocurre de forma recurrente, el objetivo de esta solución es que sea reutilizable.

2.3.3. Estilo

Es un nombre dado a un diseño arquitectónico recurrente. No existe para resolver un problema.

2.3.4. Separación de preocupaciones (SoC)

Su enfoque es separar un programa en secciones distintas, donde cada sección tiene por fin resolver tareas delimitadas.

Este concepto podemos reflejar en la arquitectura de capas, el código está separada por responsabilidades y estás capas buscan satisfacer el mismo objetivo.

2.3.5. Entidad

Las entidades son clases que mapean directamente contra la base de datos, en este sentido, una Entidad hace referencia a una tabla y tienen un atributo por cada columna de la tabla. Son muy utilizadas en los Frameworks ORM (Object-Relational Mapping) (Blancarte Iturralde, 2020, p. 351).

2.3.6. ModelMapper

El objetivo de ModelMapper es hacer que el mapeo de objetos sea fácil, determinando automáticamente como un modelo de objeto se asigna a otro, según las convenciones, de la misma manera que lo haría un ser humano, al tiempo que proporciona una API simple y seguridad para la refactorización para manejar cosas de usos específicos (Halterman, s.f.).

2.3.7. Agente de escucha

Un agente de escucha comprueba las solicitudes de conexión de los clientes mediante el protocolo y el puerto configurados. Las reglas que defina para un agente de escucha determinan como balanceador de carga va a direccionar las solicitudes a sus destinos registrados (AWS, s.f.).

2.3.8. Escalamiento vertical y horizontal

Vertical: Consiste en agregar recursos a un solo nodo, aumentando su capacidad, esto puede ser aumentar la memoria RAM del servidor, agregar discos duros de mayor capacidad, cambiar de CPU o incluso cambiar todo el servidor por uno de más capacidad (Covarrubias, s.f.).

Horizontal: Consiste en agregar nodos adicionales para adaptarse a la carga de trabajo. Si la aplicación o el sistema están llegando a su punto crítico, entonces se agregan nodos adicionales y se dive la carga entre los distintos nodos (Covarrubias, s.f.).

2.3.9. Cluster

"Grupo de múltiples ordenadores unidos mediante una red de alta velocidad, de tal forma que el conjunto es visto como un único ordenador, más potente que los comunes de escritorio" (Webmaster, s.f.).

La unión de estos servidores hace que los servicios brinden mejor rendimiento, disponibilidad alta, aislamiento de errores, balanceo de carga y escalabilidad.

2.3.10. API Rest

Es una interfaz de comunicación entre dos sistemas que se encuentran ubicados en distintos puntos, interactúan mediante el protocolo de HTTP con el objetivo de consultar o enviar datos.

2.3.11. API

Es una interfaz de programación que intermedia como un contrato entre dos aplicaciones.

2.3.12. Token

Cadena alfanumérica con caracteres aparentemente aleatorios.

2.3.13. Arquitectura del software

La arquitectura de software es el diseño de más alto nivel de la estructura de un sistema, el cual consiste en un conjunto de patrones y abstracciones que proporcionan un marco claro para la implementación del sistema (Blancarte Iturralde, 2020, p. 30).

2.3.14. Arquetipos

Un arquetipo es una compilación que genera un proyecto basado en una plantilla. Está creado para facilitar las construcciones.

Un arquetipo es un modelo, un patrón usado para generar un nuevo proyecto que comparte estructura/diseño general con otros proyectos. Técnicamente es una plantilla utilizada para generar proyectos Maven.

2.3.15. Artefacto

Un artefacto es un elemento que un proyecto puede usar o producir. En la terminología de Maven, un artefacto es una salida generada después de la construcción de un proyecto de Maven. Puede ser un jar, war o cualquier otro archivo ejecutable.

2.3.16. Batch

El procesamiento Batch o también conocido por lotes, es un proceso que ejecuta lotes de trabajo en orden secuencial de manera continuada. Este tipo de procesos se dividen en pequeñas partes que se realizan de forma repetitiva consiguiendo una mejor depuración.

2.3.17. CRUD

CRUD (Crear, Leer, Actualizar, Eliminar) es un acrónimo para las cuatro funciones del almacenamiento persistente.

- C: Crear registros
- R: Leer registros
- U: Actualizar registros
- D: Borrar registros

2.3.18. JMS

JMS (Java Message Service), API que nos permite acceder a los sistemas de mensajería. Tiene el objetivo de crear, enviar, recibir y leer mensajes.

2.3.19. Swagger

Es una herramienta que nos permite documentar, gestionar las especificaciones de nuestros servicios API mediante una interfaz intuitiva y fácil de entender.

2.4. Hipótesis

2.4.1. Hipótesis general

La arquitectura de software APH mejora favorablemente el desarrollo de microservicios en la empresa Hiper.

2.4.2. Hipótesis especificas

1. La implementación de la arquitectura de software APH satisface los requerimientos de construcción de los microservicios.
2. La arquitectura de software APH incrementa la interoperabilidad de los microservicios.
3. La arquitectura de software APH mejora la mantenibilidad de los microservicios.
4. La arquitectura de software APH aumenta la fiabilidad de los microservicios.
5. La arquitectura de software APH maximiza la escalabilidad de los microservicios.

2.5. Variables

2.5.1. Variable independiente

Arquitectura de software APH

2.5.2. Variable dependiente

Desarrollo de microservicios

2.5.3. Operacionalización de variables

Tabla 2: *Operacionalización de variables*

Variable	Definición conceptual	Definición operacional	Dimensiones	Indicadores	Escala
Arquitectura de software APH	Oscar Blancarte (2020), la arquitectura de software es el diseño de más alto nivel de la estructura de un sistema, el cual consiste en un conjunto de patrones y abstracciones que proporcionan un marco claro para la implementación del sistema.	Desarrollar microservicios en base a la arquitectura de software APH para su respectiva evaluación.	Reusabilidad	Componentes reusables	Likert
				Portabilidad de componentes	Likert
			Disponibilidad	Operación continua	Likert
			Modificabilidad	Susceptible de modificación especifica y estabilidad	Likert
			Seguridad	Confidencialidad de los datos	Likert
				Integridad de los datos	Likert
				Disponibilidad de los datos	Likert
Desarrollo de microservicios	Chris Richardson (2016), describe que normalmente, un servicio implementa un conjunto de características o funciones distintas. Cada microservicio es una pequeña aplicación que tiene su propia arquitectura hexagonal que consiste en lógica empresarial junto con varios adaptadores.	Medir el desarrollo de microservicios si son más flexibles y modulares.	Interoperabilidad	Integración entre sistemas nuevos y legados	Likert
			Mantenibilidad	Susceptible de modificación especifica y estabilidad	Likert
				Flexibilidad a los cambios	Likert
			Fiabilidad	Tolerancia a fallos	Likert
			Escalabilidad	Capacidad de crecimiento	Likert

Fuente: Elaboración propia

3. METODOLOGÍA

3.1. Tipo de estudio

Descriptivo, en el presente proyecto se obtendrán directamente de la realidad, sin que estos sean alterados o modificados. Estas nos permitirán establecer o plantear las soluciones adecuadas para los objetivos planteados.

3.2. Diseño de investigación

A continuación, se describe los diseños de investigación, acorde a lo planteado por Dankhe (1986).

3.2.1. Según la intervención del investigador

De acuerdo con la intervención el tipo de investigación es experimental, porque está orientada a determinar las causas y efectos de acuerdo los cambios aplicados en nuestra variable dependiente, por otra parte, no se dispone de información documental para describir nuestro objeto de estudio.

3.2.2. Según la planificación de la toma de datos

De acuerdo con la planificación el tipo de investigación es prospectivo, porque se busca anticipar o pronosticar los resultados a obtener con la implementación de la arquitectura de software APH en base a los conocimientos de ingeniería de sistemas e informática previo análisis de los requerimientos.

3.2.3. Según en que mide la variable de estudio

De acuerdo con la variable de estudio el tipo de investigación es longitudinal, dado que, durante el periodo de construcción de los componentes de la arquitectura de

software APH se va a ir recopilando datos basados a las pruebas del equipo de desarrollo.

3.2.4. Según variables de interés

De acuerdo con la variable de interés el tipo de estudio es analítico, porque los datos primarios y complementarios se obtendrán directamente de la realidad que nos permitirá analizar y explicar el origen de la problemática.

3.3. Tipo de investigación

3.3.1. De acuerdo con la orientación

De acuerdo con la orientación el tipo de investigación es aplicada, porque está orientada a la aplicación de conocimientos de ingeniería de sistemas e informática con la finalidad de buscar una solución práctica a una realidad concreta, respecto al desarrollo de microservicios en la empresa Hiper.

3.3.2. De acuerdo con la técnica de contrastación

La investigación es de tipo descriptiva, porque los datos primarios y complementarios se obtendrán directamente de la realidad, sin que estos sean alterados o modificados, con la finalidad de realizar estudios de las variables sin la manipulación de estas, además se describe como la arquitectura de software APH mejora el desarrollo de microservicios en la entidad objeto de estudio.

3.4. Descripción de la unidad de análisis población y muestra

3.4.1. Unidad de análisis

La unidad de análisis en el presente proyecto de investigación se considera al empleado que labora en la organización Hiper, específicamente en los departamentos de infraestructura, desarrollo, seguridad y calidad.

3.4.2. Población

La población del presente proyecto de investigación está conformada por los empleados de la organización Hiper, que conforman en los departamentos de infraestructura, desarrollo, seguridad y calidad. Según los datos de la gerencia de Recursos Humanos hacen un total de 218 empleados.

3.4.3. Muestra

Para este proyecto de investigación se obtendrá la muestra mediante el uso de la siguiente formula por ser una población finita.

$$n = \frac{N * Z^2 * p * q}{e^2 * (N - 1) + Z^2 * p * q}$$

Donde:

$n = Muestra$

$Z = 1.96$ $(Nivel\ de\ confianza\ al\ 95\%)$

$p = 0.5$ $(Proporción\ del\ éxito\ 50\%)$

$q = 0.5$ $(Proporción\ del\ fracaso\ 50\%)$

$e = Margen\ de\ error\ o\ precisión\ 0.05$

$N = Población$

Obteniendo como resultado de la muestra a considerar es de 139 empleados.

$$n = \frac{218 * 1.96^2 * 0.5 * 0.5}{0.05^2 * (218 - 1) + 1.96^2 * 0.5 * 0.5} = 139.30$$

3.4.4. Tipo de muestreo

Para identificar la población de la cual se va a recopilar la información se usa el Muestreo Probabilístico (Aleatorio). En este tipo de muestreo, todos los individuos de la población pueden formar parte de la muestra, por lo tanto, se adapta a nuestra necesidad porque en la investigación se busca representatividad de los datos y a la vez la funcionalidad de la arquitectura de software APH.

3.5. Técnicas de instrumentos de recolección de datos

Para obtener la información necesaria para esta investigación se utilizaron varías técnicas que se complementan así mismas.

3.5.1. Fuentes primarias

El proceso de recolección de datos para su posterior análisis de las fuentes primarias se llevará a cabo, haciendo uso de las encuestas, entrevistas, observación y registros de datos en el sistema.

Observación (directa o indirecta): Para el presente proyecto se realiza en ambos casos, directa porque el investigador está presente en el lugar de los hechos para realizar las observaciones de cada uno de los procesos involucrados para la investigación, por otro lado, indirecta porque se va a revisar documentaciones respecto al desarrollo de microservicios en diferentes proyectos por equipo de desarrollo.

Entrevista: El investigador formula una serie de preguntas dirigidas a los empleados que integran a los departamentos de arquitectura, desarrollo, seguridad y calidad que laboran en la entidad de manera presencial y virtual, estableciendo

un dialogo peculiar, de esta forma logrando recolectar la información necesaria de los involucrados directos en nuestra investigación.

Encuesta: Formato elaborado especialmente con los ítems y alternativas cerradas con base a las variables e indicadores de estudio, que serán aplicadas a la muestra establecida. Ver **Anexo 02.**

3.5.2. Fuentes secundarias

La recopilación de datos a través de las fuentes secundarias se realiza haciendo uso de la información documental como citas de autores de renombre, revistas, internet, proyectos de tesis, que tengan relación con la investigación.

3.5.3. Técnicas de procesamiento de datos

El procesamiento de la información recolectada se aplicará técnicas estadísticas en la que es imprescindible el uso y soporte de las herramientas informáticas como hojas de cálculo (Excel) y el SPSS para procesar los datos obtenidos.

3.6. Técnicas de análisis y prueba de hipótesis

3.6.1. Técnicas de análisis

Para la presente investigación se utiliza el análisis situacional, herramienta que permite representar la situación actual logrando entender y/o clarificar lo que ocurre dentro de la investigación.

Para el desarrollo del proyecto, se consideraron seis etapas que dieron el cumplimiento a los objetivos de esta investigación.

Tabla 3: *Descripción de etapas para el proyecto*

Etapa	Actividad

Identificación de requerimientos	Reunión con el jefe del proyecto y lideres técnicos
	Recopilación de información
	Modelado de diagramas de la situación actual
Análisis	Análisis situacional
	Definición de requerimientos del negocio
Diseño	Diseño del prototipo de solución
	Modelar los componentes de la arquitectura
Desarrollo	Desarrollo de la arquitectura APH
Pruebas	Pruebas funcionales del desarrollador
	Pruebas funcionales por el departamento de calidad
Implementación	Capacitación a los equipos de los departamentos involucrados
	Arquitectura puesta en producción

Fuente: Elaboración propia

3.6.2. Prueba de hipótesis

Para poder probar la hipótesis de la investigación, se utilizó la prueba de Wilcoxon en base a la evaluación previa de la estadística inferencial.

- **Estadística descriptiva**

  Se encargará de describir a los sujetos estudiados en relación con todos y cada una de las variables recogidas (variable cuantitativa y variable cualitativa).

- **Estadística inferencial**

  Dado que el tamaño de la muestra supera más de 50 empleados, se somete a la prueba Kolmogórov-Smirnov para determinar si la significancia adopta por una distribución normal o no, en base a esta elegir la prueba de hipótesis a utilizar.

4. RESULTADOS DE LA INVESTIGACIÓN

4.1. Descripción del trabajo de campo

4.1.1. Análisis de la situación actual

a) Reseña Histórica

La organización Hiper es una empresa peruana, creada en el año de 1983, está enfocada al diseño, desarrollo e implementación de soluciones tecnológicas empresariales que están relacionadas con la atención y gestión de transacciones cubriendo las necesidades y/o solicitudes de sus clientes.

Su principal actividad está orientada a brindar productos y servicios de alta calidad en diferentes países de Sudamérica, contribuyendo una vasta experiencia en procesos transaccionales financieros.

b) Análisis de organigrama funcional – estratégico

Nombre de la empresa: Hiper S.A.

Ubicación geográfica: Se encuentra establecida en la Urb. Parque Industrial Av. Colonial Cuadra 58 de la provincia constitucional del Callao, distrito de Callao.

En el presente proyecto de tesis tendrá específicamente como entorno de análisis en los departamentos de desarrollo, arquitectura, seguridad y calidad, en las cuales se identificó la problemática y que mediante el desarrollo e implementación de la arquitectura de software APH se quiere dar solución a la misma. Los departamentos se detallan a continuación.

- **Desarrollo:** Equipo de trabajo con los suficientes conocimientos y experiencias para afrontar el desarrollo de software a medida que ofrezcan la mejor solución a las necesidades.
- **Arquitectura:** Cubre la arquitectura tecnológica empresarial enfocada en la infraestructura global de los sistemas.
- **Seguridad:** Encargada de velar por la seguridad de las aplicaciones en las diferentes capas de los entornos de desarrollo y despliegue.
- **Calidad:** Asegurar la calidad de los productos bajo ciertos criterios de desarrollo de software para alcanzar la cobertura técnica y garantizar la flexibilidad y mantenimientos de estas.

La organización está enfocada en crear soluciones tecnológicas integrales por más 30 años, en tal sentido se tiene como:

Misión

- Con el cliente, satisfacerlo con proyectos integrales y exitosos.
- Con el personal, Asumiendo nuevos retos tecnológicos que ayuden al logro del desarrollo integral.
- Con la sociedad, facilitando las actividades de las personas e instituciones con soluciones creativas.

Visión

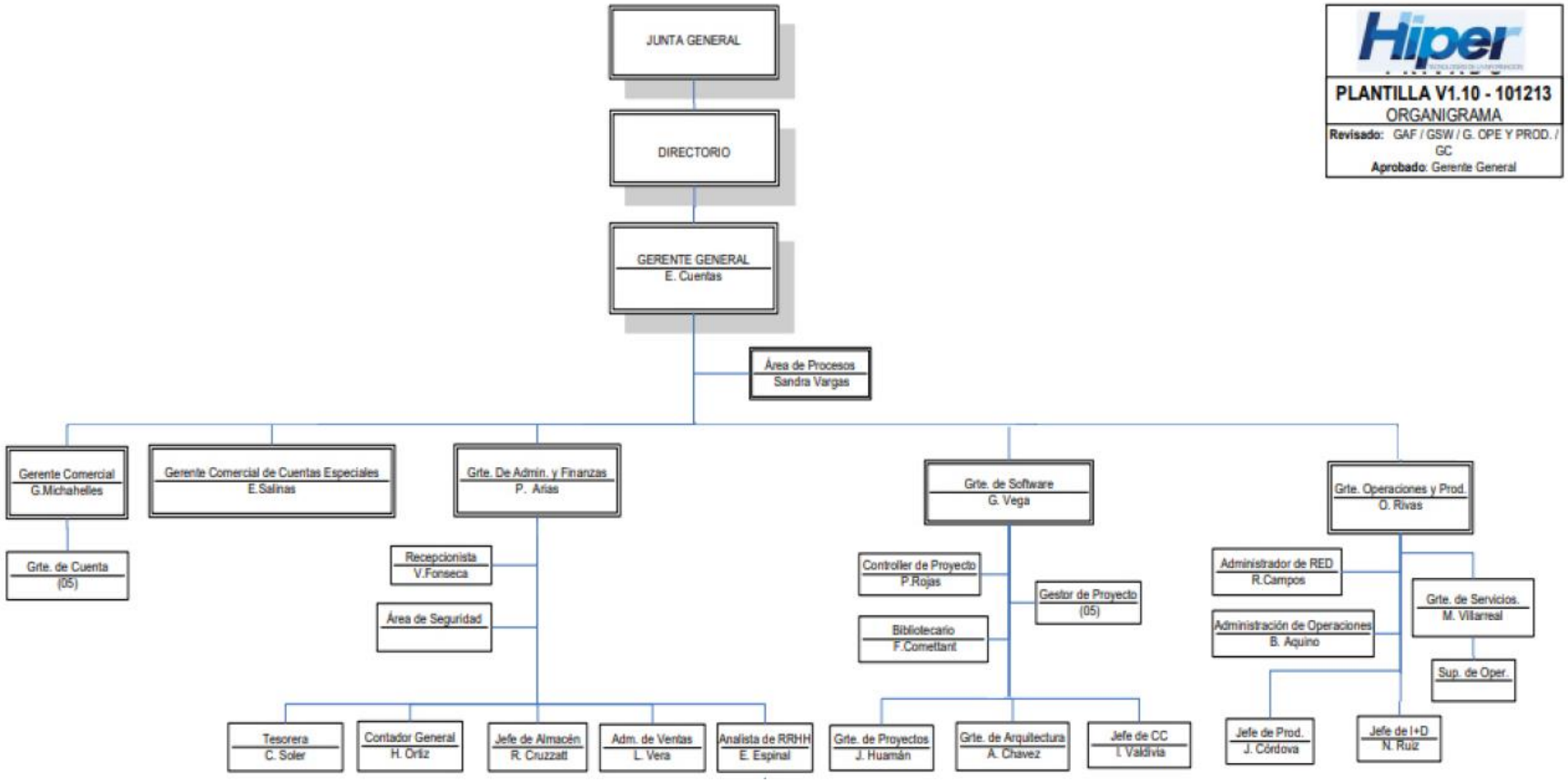
- Ser principales gestores tecnológicos en nuestro país con proyección a Latinoamérica.
- Ser socios tecnológicos de nuestros clientes, brindando calidad de servicio.

Objetivo

- Ser lideres en tecnologías de información.
- Convertirnos en lideres en el servicio de redes operativas y/o publicidad digital.

Organigrama

Gráfico 27: *Organigrama de Hiper*



Fuente: Organigrama de Hiper S.A. (Intranet Hiper, 2018)

c) Evaluación de la capacidad instalada

La organización cuenta con la disponibilidad de la infraestructura, recursos tecnológicos y humanos para la implementación de la arquitectura APH.

Personal

Se cuenta con personal capacitado con conocimientos sólidos en diferentes tecnologías de acuerdo con la necesidad de los departamentos de desarrollo, arquitectura, seguridad y calidad.

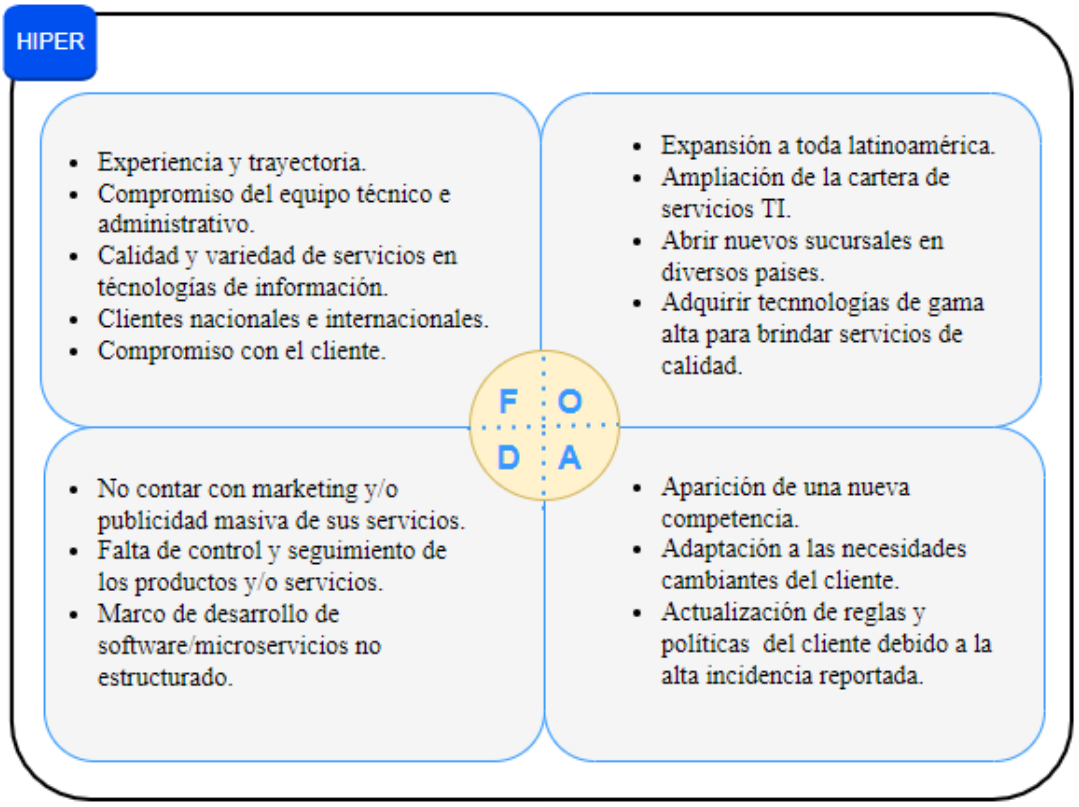
Equipos

Debido al rubro que está orientado la organización, cuenta con los equipos tecnológicos de gama alta, con servidores virtuales y repositorios basados en DevOps.

d) Análisis de FODA

Las fortalezas y las debilidades que están orientadas a identificar las capacidades y limitaciones internas respectivamente, las oportunidades y amenazas que son factores externos fueron identificadas haciendo uso de la herramienta FODA a partir de la información recolectada aplicando entrevistas, encuestas al personal técnico y administrativo de la organización Hiper.

Gráfico 28: *Análisis FODA*



Fuente: Elaboración propia

4.1.2. Requerimientos y procesos

a) Identificación y descripción de los requerimientos

- **Seguridad**

  Como sistema de autentificación y autorización debe adaptarse a los siguientes medios:

  Access Manager

  - La autentificación a través del sistema corporativo del cliente (Access Manager). En esta participa la arquitectura FrontEnd, que es la encargada de conectarse con los servicios REST que habilitan el token y la información del usuario que se ha autenticado.

- El sistema de seguridad del cliente proveerá de un token, que será enviado de forma transparente para el desarrollador en cada petición del FrontEnd al BackEnd.
- Cada microservicio será responsable de su seguridad, por lo que esta, formará parte de la arquitectura que la sustenta.
- La comunicación entre FrontEnd y BackEnd se realizará mediante el protocolo HTTPS.

Oauth2

Autenticación mediante el estándar Oath2 para la autorización de las APIs, que permita compartir la información con las aplicaciones de terceros.

- **Integraciones**

  Los microservicios deben tener la capacidad de integrarse con varios sistemas. Ver **Anexo 04**.

  Establecer las siguientes integraciones:

  - Mediante el uso de Topic del gestor de colas Tibco.
  - Mediante llamadas a servicios REST o servicios web de sistemas de terceros On Premise.

  Integración vía Topic

  Las integraciones con los gestores de colas implementar utilizando Java Message Service (JMS).

  JMS se separa de las APIs propietarias de cada proveedor para ofrecer un API estándar (mediante un conjunto de interfaces) para la mensajería

empresarial, de modo que mediante Java podamos enviar y recibir mensajes sin atarnos a ningún proveedor (Alvarez Villanueva, 2017).

La implementación del Driver JMS esta implementado desde la versión 2 de java, y por supuesto incorporada en Spring Boot mediante anotaciones.

- Para incorporar la integración con un gestor de colas determinado, se debe generar un Bean de configuración de Spring Boot con los parámetros de configuración correspondientes para conectar con el gestor de colas.
- La dependencia a nivel de pom de la librería correspondiente del gestor de colas, se debe proveer a nivel del microservicio que se vaya a implementar, y no a nivel de arquitectura general, ya que no todos los microservicios tendrán integración con el gestor de colas.
- Debido a la infraestructura de la mayoría de los clientes está en el cloud de AWS se debe plantear como gestor de colas el servicio Amazon SQS (Amazon Simple Queue Service).

Integración mediante API REST o Web Service

Otro tipo de integración será mediante llamadas a servicios REST o Web Services de sistemas terceros (clientes), que se encuentran ubicados en On Premise o Cloud.

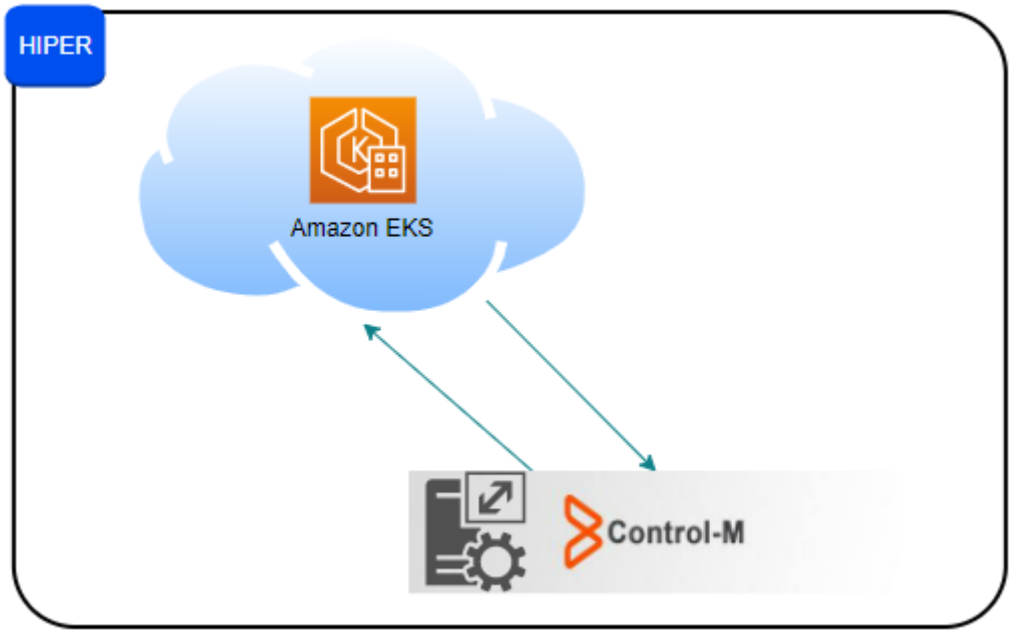
La arquitectura debe proveer de dos módulos funcionales que permita realizar llamadas tanto a REST como a Web Service. Para ello integrar las tecnologías de RestTemplate y Apache CXF.

Estos módulos se basan en interceptores que añaden funcionalidad extra de auditorías de las llamadas entrantes y salientes.

- **Procesos Batch**

  Los procesos Batch se ejecuta como lógica de negocio implementado como microservicios. Generalmente la ejecución de estos microservicios es la siguiente:

  Gráfico 29: *Ejecución de procesos batch*



Fuente: Elaboración propia

Desde Control-M se lanza una petición Restfull al microservicio, el cuál ejecuta el proceso batch. Al terminar recibe la respuesta con un indicador si el proceso se ha ejecutado correctamente.

Evidentemente, el marco de Spring nos proporciona la tecnología de Spring Batch, para ello la arquitectura APH debe incluir esta librería, dado que nos aporta funcionalidades básicas para ejecutar procesos por lotes.

Seguridad en la llamada al servicio REST Batch

El servicio REST que ejecutará el proceso Batch deberá verificar que es invocado por Control-M, por lo que la petición debe incluir un token pactado entre los dos sistemas para que el servicio REST pueda verificar unívocamente que es llamado por el planificador. La arquitectura APH habilita, dentro del módulo de seguridad, un mecanismo de interceptores ante una petición REST que ejecutará la lógica correspondiente que se necesite para comprobar el token o el identificador pactado entre Control-M y el microservicio.

b) Arquitectura tecnológica global

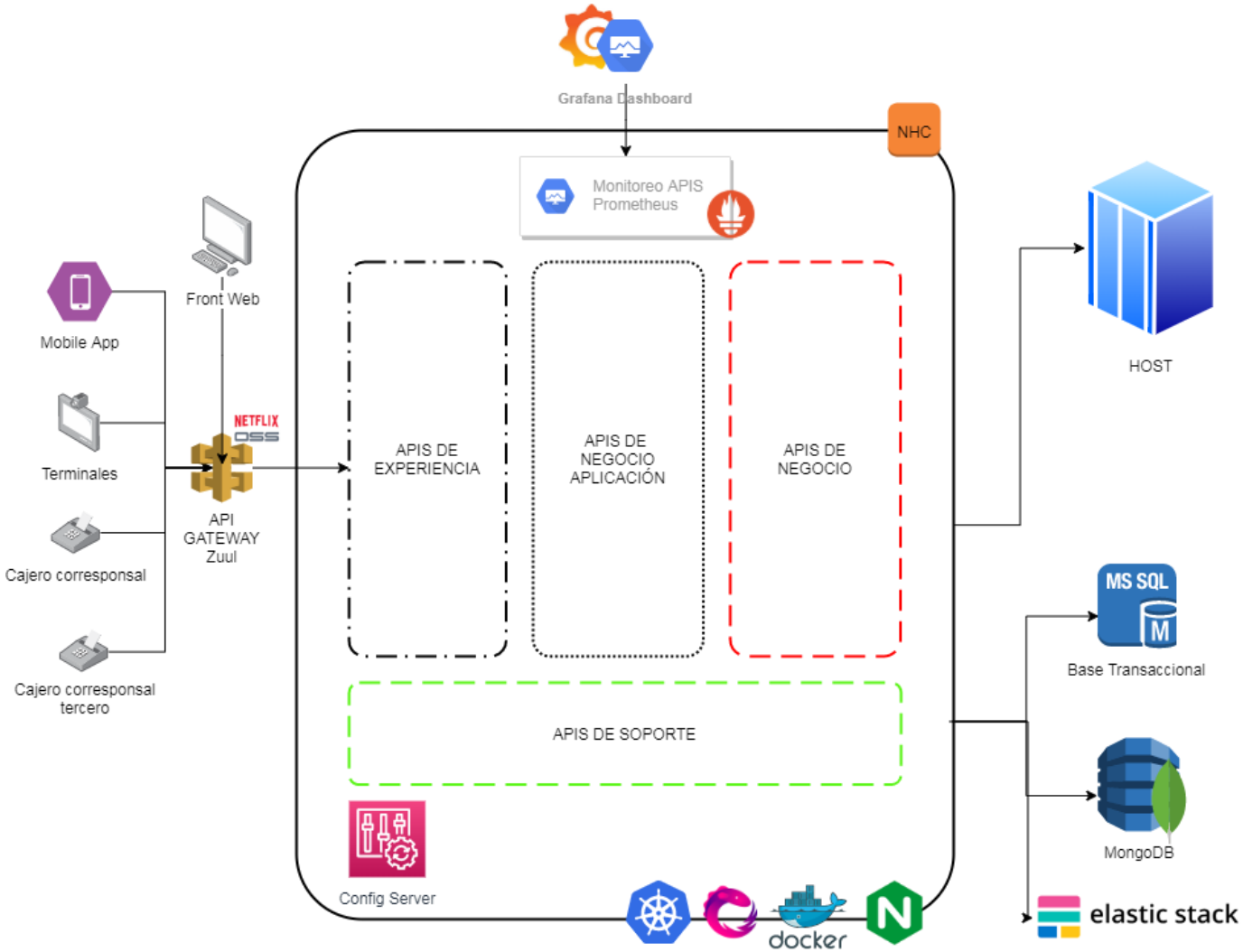
La arquitectura global nos muestra la interacción de diferentes componentes principales, de las cuales podemos resaltar:

Entrada: La solicitud ingresa desde aplicaciones móviles, plataforma web, terminales y cajeros corresponsales.

Proceso: De acuerdo con el punto de entrada se procesa la información a través de los microservicios ya sean propias del negocio o de terceros, estas tienen una comunicación con el motor de la base de datos para recuperar la información.

Salida: Los resultados obtenidos se devuelve a los puntos de entrada dado la confirmación de las transacciones enviadas.

Gráfico 30: *Arquitectura tecnológica global*

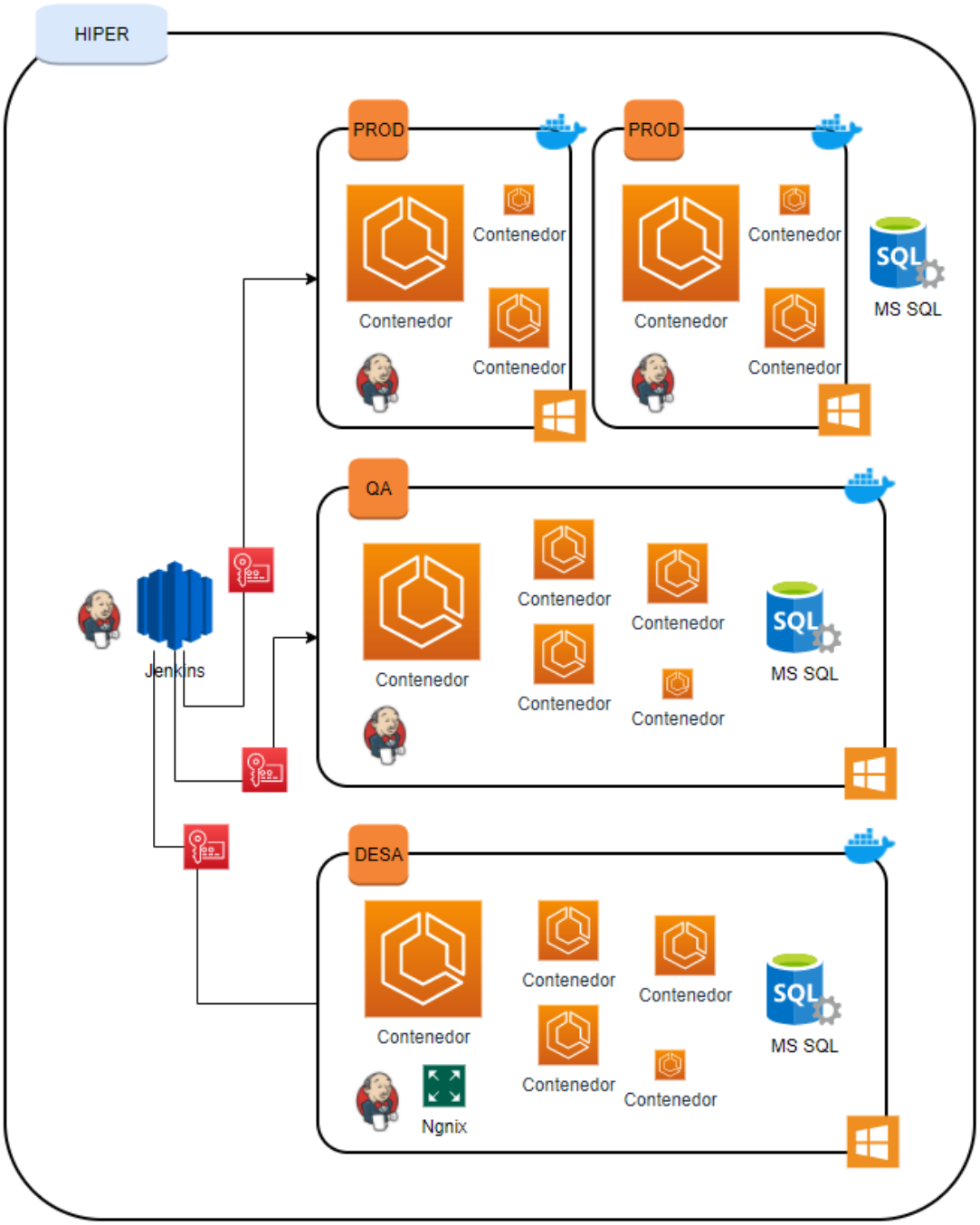


Fuente: Elaboración propia

c) Arquitectura de despliegue de los microservicios

El despliegue de los microservicios contenerizados en diferentes entornos se esquematiza en el siguiente gráfico.

Gráfico 31: *Arquitectura de despliegue de los microservicios*

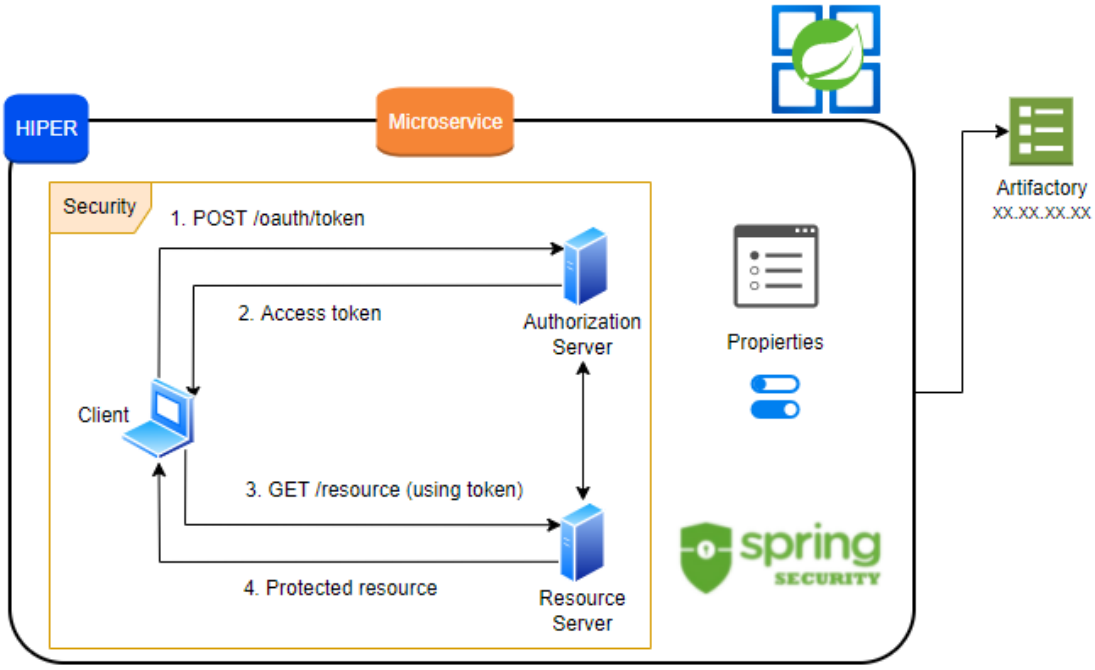


Fuente: Elaboración propia

d) Seguridad en los microservicios

El pilar fundamental en los microservicios es la seguridad, que permite tener con mayor control, en la actualidad todo microservicio desarrollado tiene integrado la tecnología Spring Security.

Gráfico 32: *Seguridad en los microservicios*

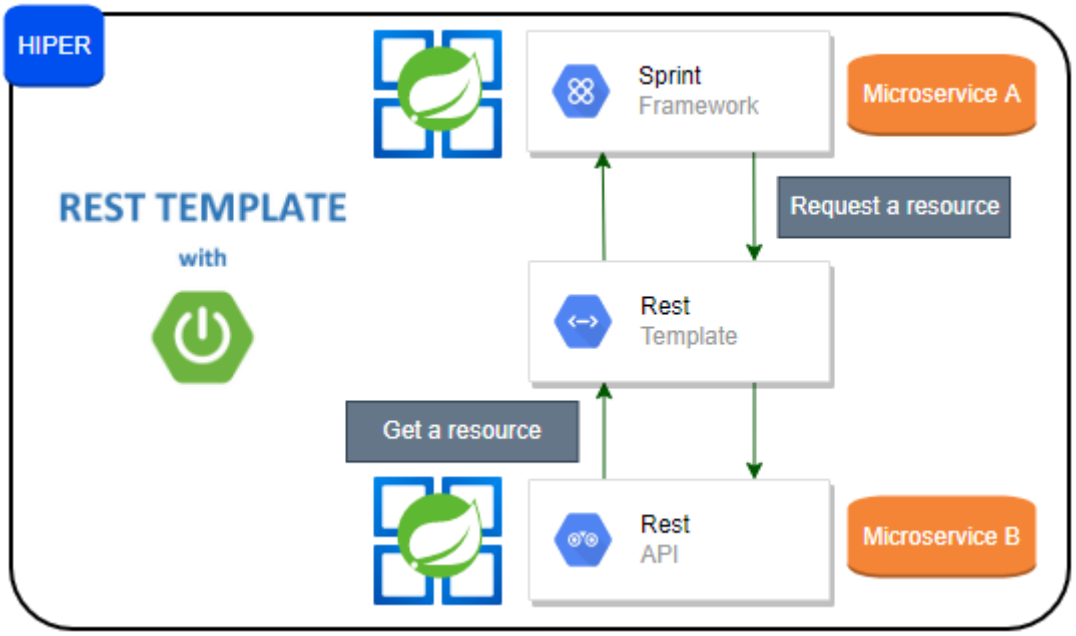


Fuente: Elaboración propia

e) RestTemplate en la comunicación de microservicios

El marco de Spring proporciona el RestTemplate para el acceso desde la parte cliente a servicios REST.

Gráfico 33: *RestTemplate en los microservicios*

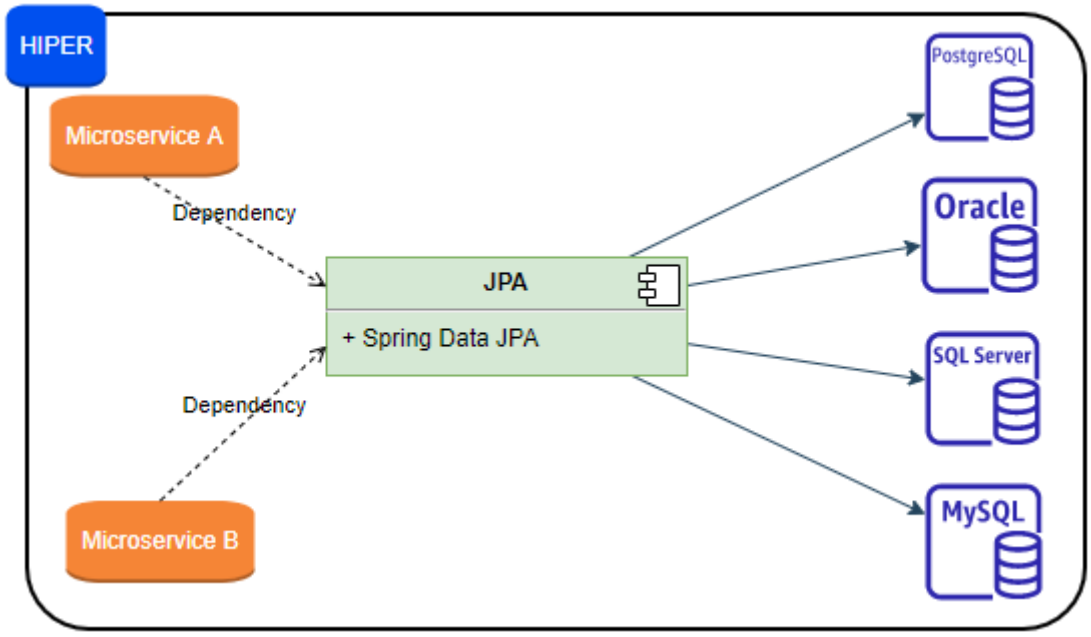


Fuente: Elaboración propia

f) Acceso a datos con JPA

Los microservicios integran Data JPA, que permite el acceso a los datos, marcos de reducción de mapas, servicios en la nube, así como compatibilidad con bases de datos relacionales muy avanzadas.

Gráfico 34: *Acceso a datos con JPA*



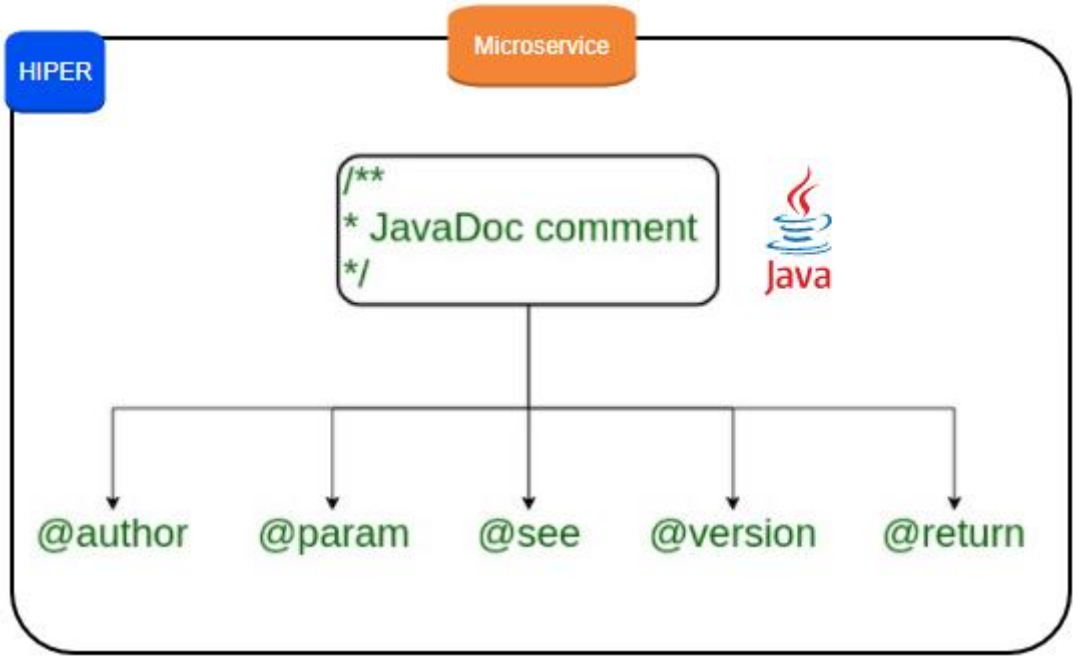
Fuente: Elaboración propia

Spring Data JPA, nos proporciona una serie de características sencillas para acceder y recuperar información desde la base de datos, los tipos de consultas usadas frecuentemente son JPQL (escrito en la sintaxis del lenguaje de consulta de persistencia de Java) y NativeQuery (escrito en sintaxis SQL simple)

g) Documentación de código

La documentación de los servicios es uno de los factores que contribuye al éxito general y las buenas prácticas del desarrollo de los microservicios, para ello se usa Javadoc que es una utilidad de Oracle para la generación de documentación de APIs en formato HTML a partir del código fuente de Java.

Gráfico 35: *Documentación Javadoc*

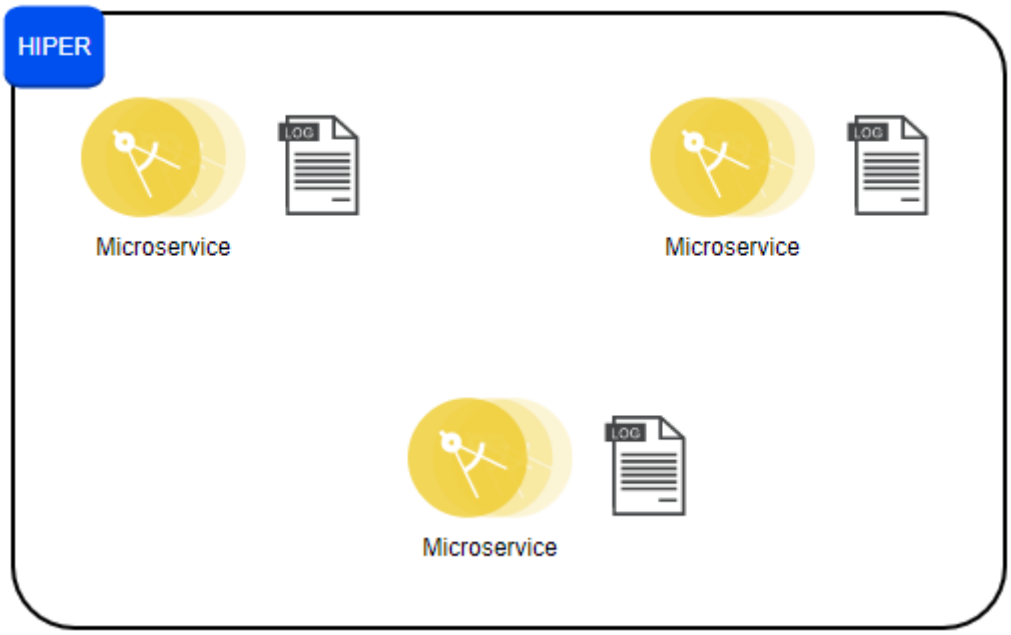


Fuente: Elaboración propia

h) Registro de logs en los microservicios

La trazabilidad de los microservicios se almacena en archivos independientes para su posterior seguimiento en caso de errores del sistema, estas se encuentran almacenadas en una unidad física, el formato y/o estructura es definida por cada desarrollador.

Gráfico 36: *Registro de logs en los microservicios*



Fuente: Elaboración propia

4.1.3. Nivel de satisfacción del desarrollo de microservicios actual

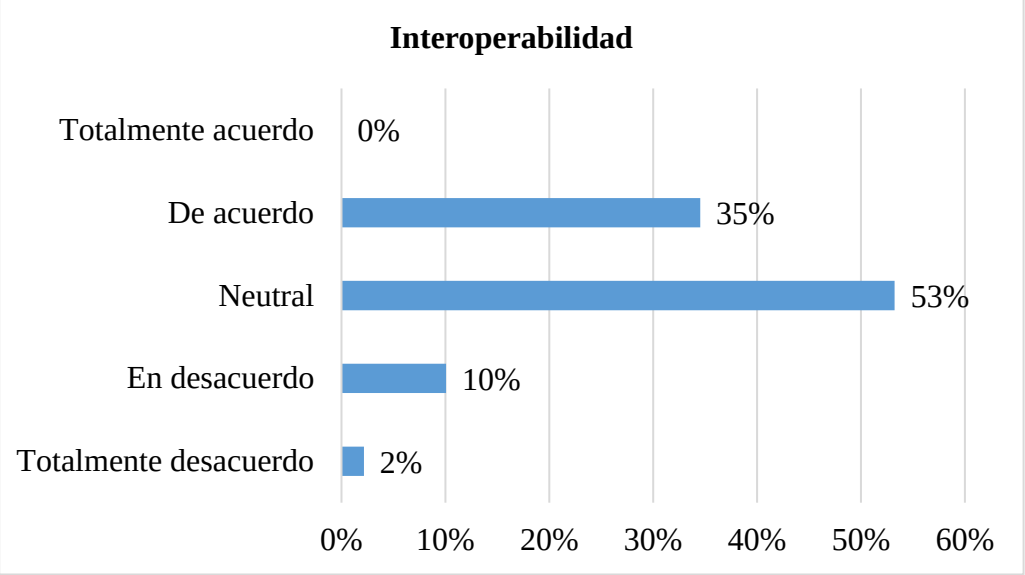
Para contrastar los datos hallados mediante el análisis de la situación actual se procedió con la aplicación de una encuesta dirigida al personal de los departamentos de desarrollo, arquitectura, seguridad y calidad de la organización Hiper, los resultados hallados por dimensión se obtuvieron.

Tabla 4: *Resultados de las dimensiones del desarrollo de microservicios*

	Desarrollo de microservicios			
Nivel\Dimensión	**Interoperabilidad**	**Mantenibilidad**	**Fiabilidad**	**Escalabilidad**
Totalmente desacuerdo	2%	3%	1%	1%
En desacuerdo	10%	15%	21%	9%
Neutral	53%	62%	47%	44%
De acuerdo	35%	19%	27%	37%
Totalmente acuerdo	0%	1%	4%	9%

Fuente: Elaboración propi

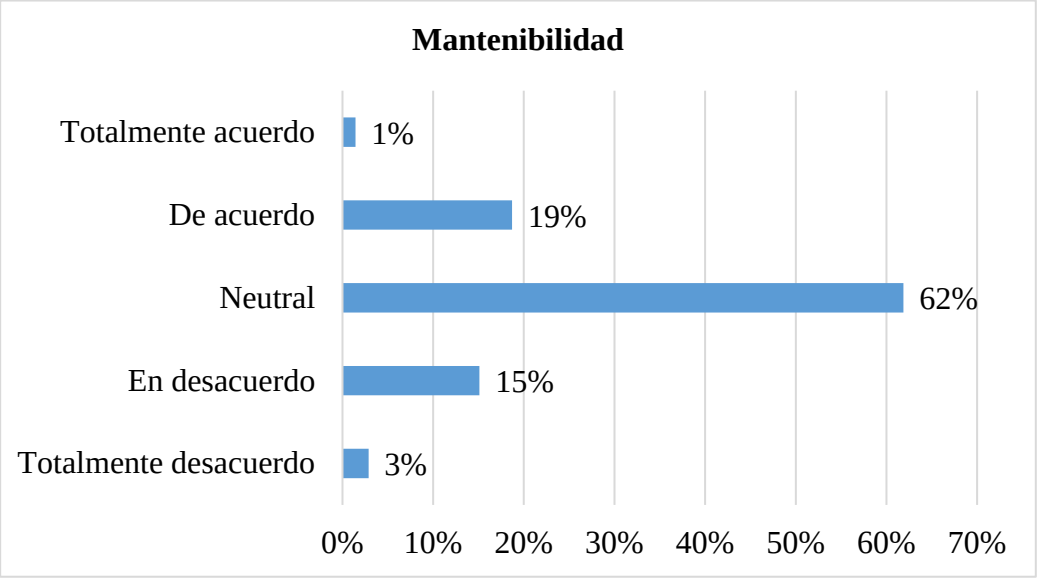
Gráfico 37: *Calificación de la interoperabilidad de los microservicios*



Fuente: Elaboración propia

Como se observa en la tabla N° 4 y gráfico 37 en cuánto a la dimensión de la interoperabilidad, la mayoría del personal de los departamentos involucrados califican que los microservicios que se vienen desarrollando tienen un nivel de aceptación como neutro (53%) y un porcentaje menor como de acuerdo (35%) y los que están en desacuerdo y totalmente desacuerdo con un 10% y 2% respectivamente.

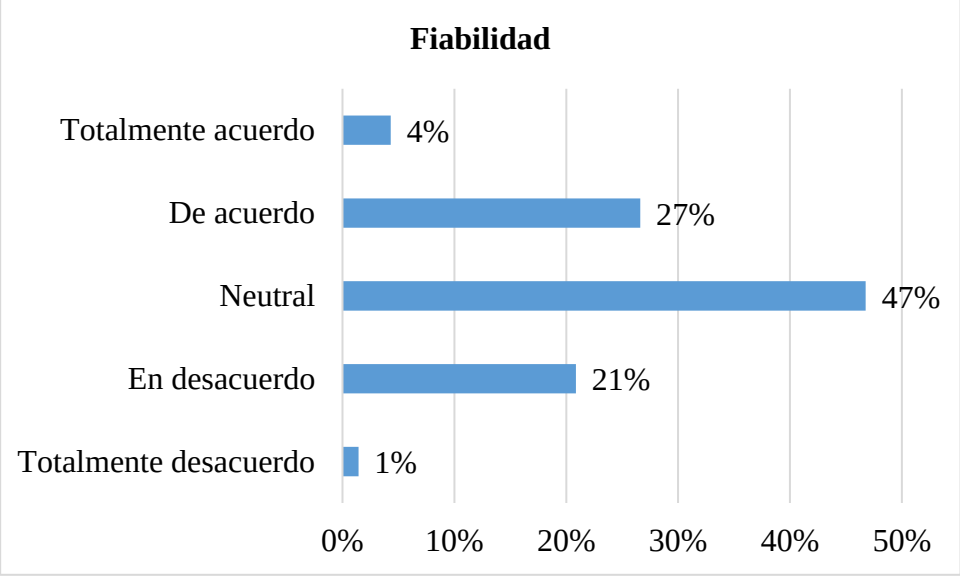
Gráfico 38: *Calificación de la mantenibilidad de los microservicios*



Fuente: Elaboración propia

Cómo se observa en la tabla N° 4 y gráfico 38 en cuánto a la dimensión de la mantenibilidad, el personal de los departamentos involucrados considera a nivel neutro (62%), los que se encuentran de acuerdo (19%), el 15% está en desacuerdo, existe un 3% que está totalmente desacuerdo y solo el 1% totalmente de acuerdo.

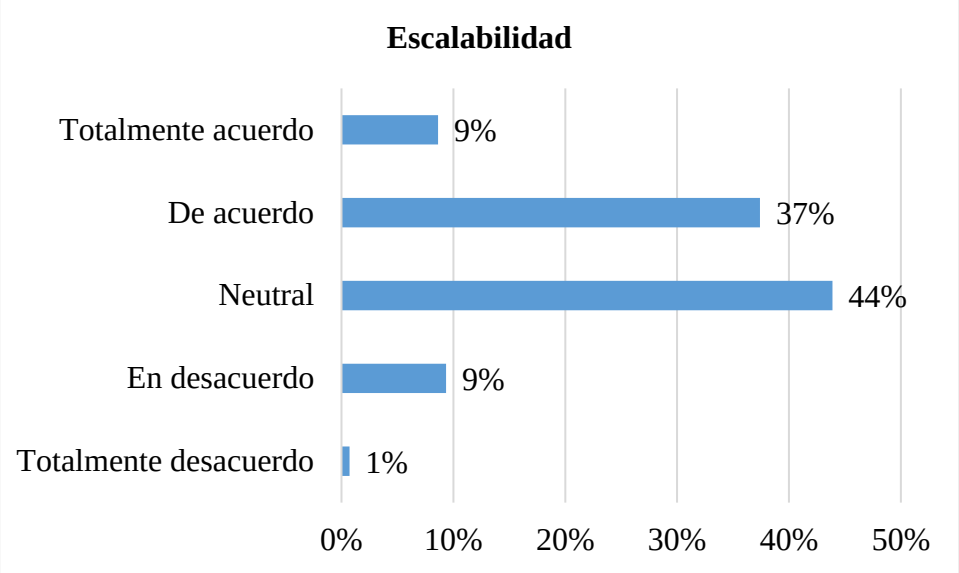
Gráfico 39: *Calificación de la fiabilidad de la mantenibilidad de los microservicios*



Fuente: Elaboración propia

Como se observa en la tabla N° 4 y gráfico 39 respecto a la dimensión de fiabilidad, la totalidad de los participantes en la encuesta de los departamentos involucrados califican de manera neutral (47%), los que están acuerdo (27%), aquellos que están en desacuerdo (21%), el 4% consideran que están totalmente de acuerdo y el 1% están totalmente desacuerdo.

Gráfico 40: *Calificación de la escalabilidad de los microservicios*



Fuente: Elaboración propia

Cómo se observa en la tabla N° 4 y gráfico 40 en cuánto a la dimensión de escalabilidad, el personal de los departamentos involucrados califica a un nivel de 44% neutral, el 37% que están de acuerdo, el 9% consideran totalmente acuerdo, así lo mismo coinciden el mismo porcentaje en desacuerdo y finalmente el 1% que están totalmente desacuerdo.

En base a la evaluación realizada se puede afirmar que el desarrollo de microservicios se viene construyendo a un nivel intermedio, debido a que el personal no se muestra conforme con lo que se viene realizando, pero ello puede

deberse a que muchos desarrolladores no siguen un estándar de desarrollo y que los microservicios se vienen implementando a un nivel de experticia independiente. A su vez se observa que existen muchos desarrolladores que califican como desacuerdo y totalmente desacuerdo, debido a las deficiencias existentes en la actualidad.

4.1.4. Diagnóstico de la situación actual

a. Informe de diagnóstico

A partir del análisis de la situación actual en la que se encuentra la organización Hiper, se llega al siguiente diagnóstico.

- No se cuenta con una arquitectura sólida que permita crear microservicios bajo un estándar establecido.
- Existe componentes reusables (seguridad, rest-template, acceso a datos, trazabilidad) que podemos abstraer de los microservicios para administrar desde la arquitectura.
- Los microservicios se integran con servicios de terceros mediante API REST, más no tienen la capacidad de integrarse por Web Service.
- La capa de seguridad está implementada de forma independiente por microservicio.
- Carencia en la documentación del código, si bien se usa algunas características de Javadoc; sin embargo, esta no tiene todas las funcionalidades para generar una documentación completa e intuitiva.
- La trazabilidad de los logs no es robusta, debido a que no se registra los interceptores con otros microservicios.

- La gestión de errores no está clasificada a nivel técnico y de negocio, así lo mismo no se tiene un control en el Timeout en las invocaciones a través de API REST a las interfaces de RestTemplate.
- Los procesos Batch se encuentran configurados con la anotación de Spring (Programador de tareas) que se adapta más a proyectos pequeños o de prueba debido a la carga directa dentro del microservicio.
- Se cuenta con personal capacitado en diferentes tecnologías en el ámbito de desarrollo de microservicios, así lo mismo con entornos de prueba, calidad y producción, para que la arquitectura desarrollada sea integrada y desplegada correctamente.

b. Medidas de diagnóstico

La organización Hiper, dispondrá de una arquitectura APH que permitirá la agilidad y la calidad en la construcción de microservicios, facilitando a los desarrolladores centrarse a implementar la funcionalidad solicitada por los clientes.

Con esta herramienta tecnológica se podrá desarrollar microservicios con capacidades de interoperabilidad con servicios de terceros, que el mantenimiento correctivo y preventivo sean flexibles, debido a que los componentes principales se administran desde la arquitectura como la seguridad son fiables y finalmente estén diseñados para escalar fácilmente en el futuro.

Con la implementación de la arquitectura APH se pretende que la organización Hiper maximice sus servicios teniendo un mejor control y calidad en sus productos.

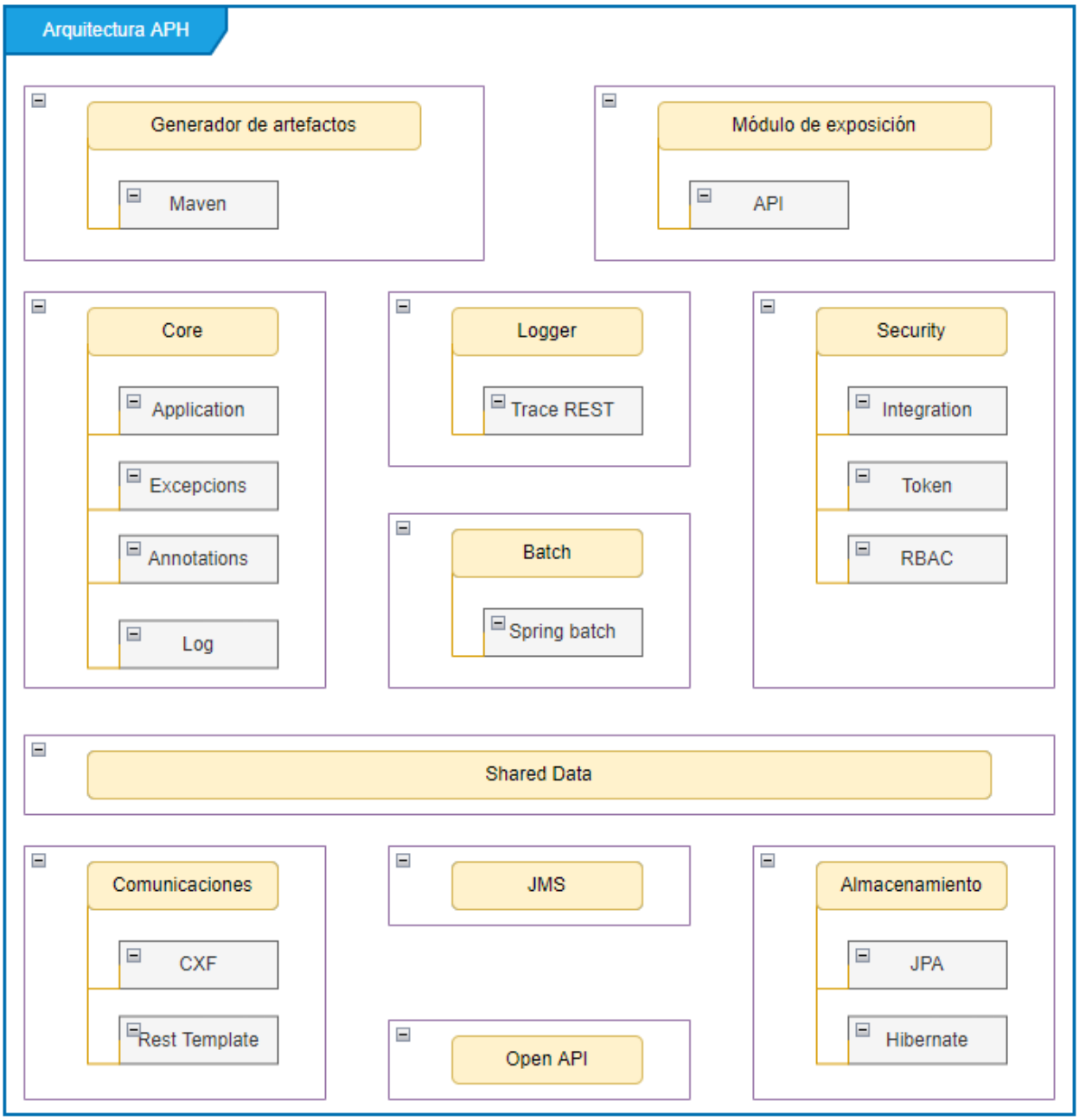
4.2. Presentación resultado y prueba de hipótesis

4.2.1. Arquitectura tecnológica de la solución

Arquitectura APH

A continuación, se muestra el esquema global de la arquitectura, que nos permitirá contextualizar funcionalidades que cubre, agrupadas por bloques.

Gráfico 41: *Arquitectura tecnológica de la solución*



Fuente: Elaboración propia

Ramas y entornos de despliegue

Para la implementación de la arquitectura APH, disponemos de diferentes ramas y entornos sobre la cual se aplican nuevas funcionalidades y cuya diferencia radica en la posibilidad de mantener dos flujos de trabajo distintos con funcionalidades

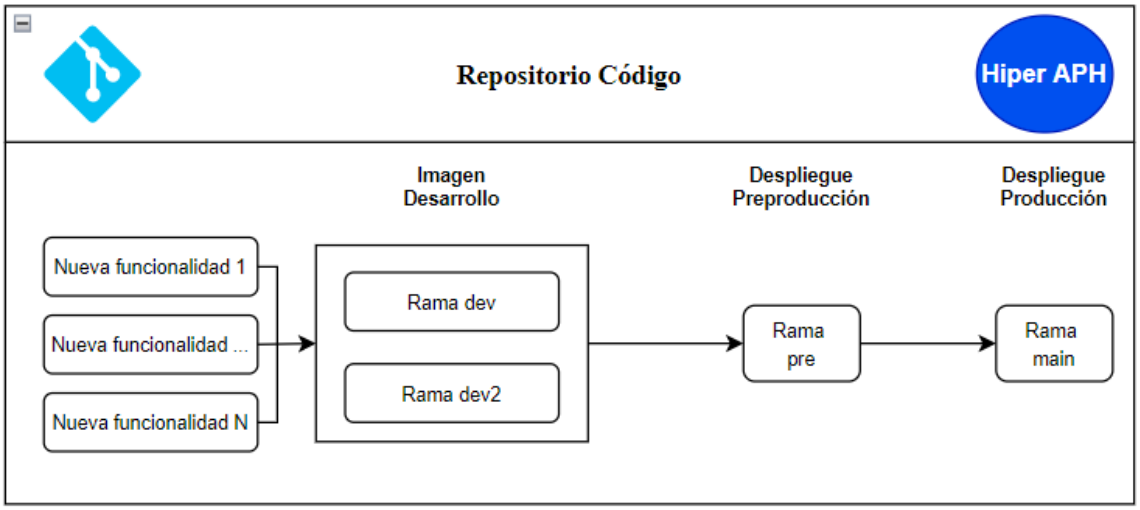
distintitas, el entorno de preproducción donde se realizan las pruebas de funcionamiento de la línea productiva y las pruebas funcionales de los evolutivos, y la otra el entorno de producción.

Los despliegues sobre estos entornos se realizan a través de GitHub, para los despliegues disponemos de ramas protegidas (únicamente se puede subir código mediante el pull request) que contiene el código a desplegar en cada entorno.

- Dev: Entorno de desarrollo al que se aplicaran las nuevas funcionalidades.
- Dev2: Entorno de desarrollo con funcionalidad especifica, al que se aplicaran las nuevas funcionalidades y los correctivos de mantenimiento.
- Pre: Entorno de preproducción en el que se realizan pruebas funcionales por parte del equipo de Arquitectura.
- Main: Entorno producción, en que se despliegan los cambios probados en preproducción.

Los cambios deben de seguir una evolución de entorno a entorno, siendo testeados y verificados en el entorno previo antes de pasar al siguiente.

Gráfico 42: *Ramas y entornos de despliegue*



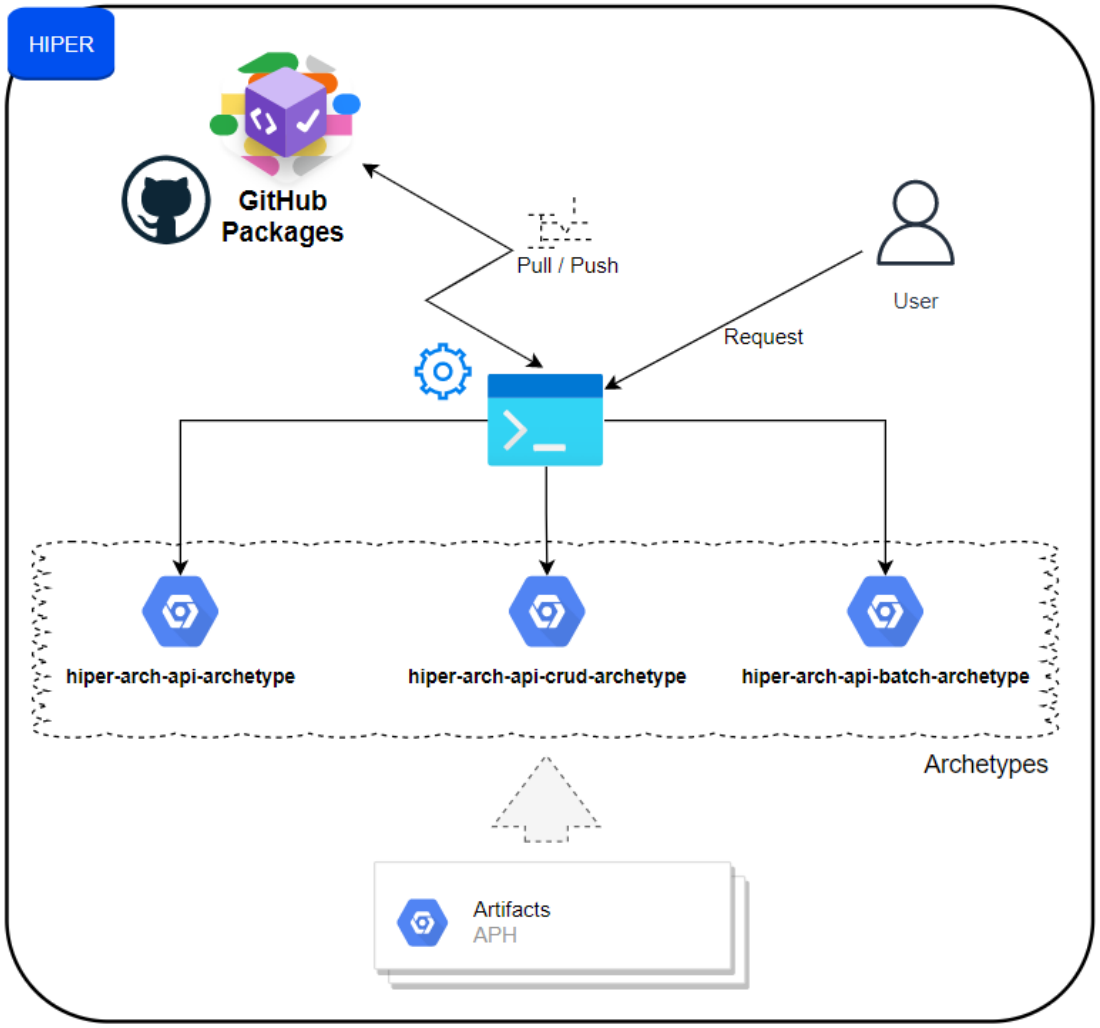
Fuente: Elaboración propia

Generación de soluciones con la arquitectura APH

La arquitectura APH se encuentra empaquetado y disponible en el repositorio de GitHub, hacemos uso de la interfaz de línea de comandos (CLI) que nos permite conectarnos a través de un token de seguridad.

Al ingresar comandos de instrucción en el CLI, generamos un arquetipo en particular en base a la arquitectura APH, que consigo trae las dependencias de Spring y los artefactos configurados para iniciar el desarrollo de los microservicios.

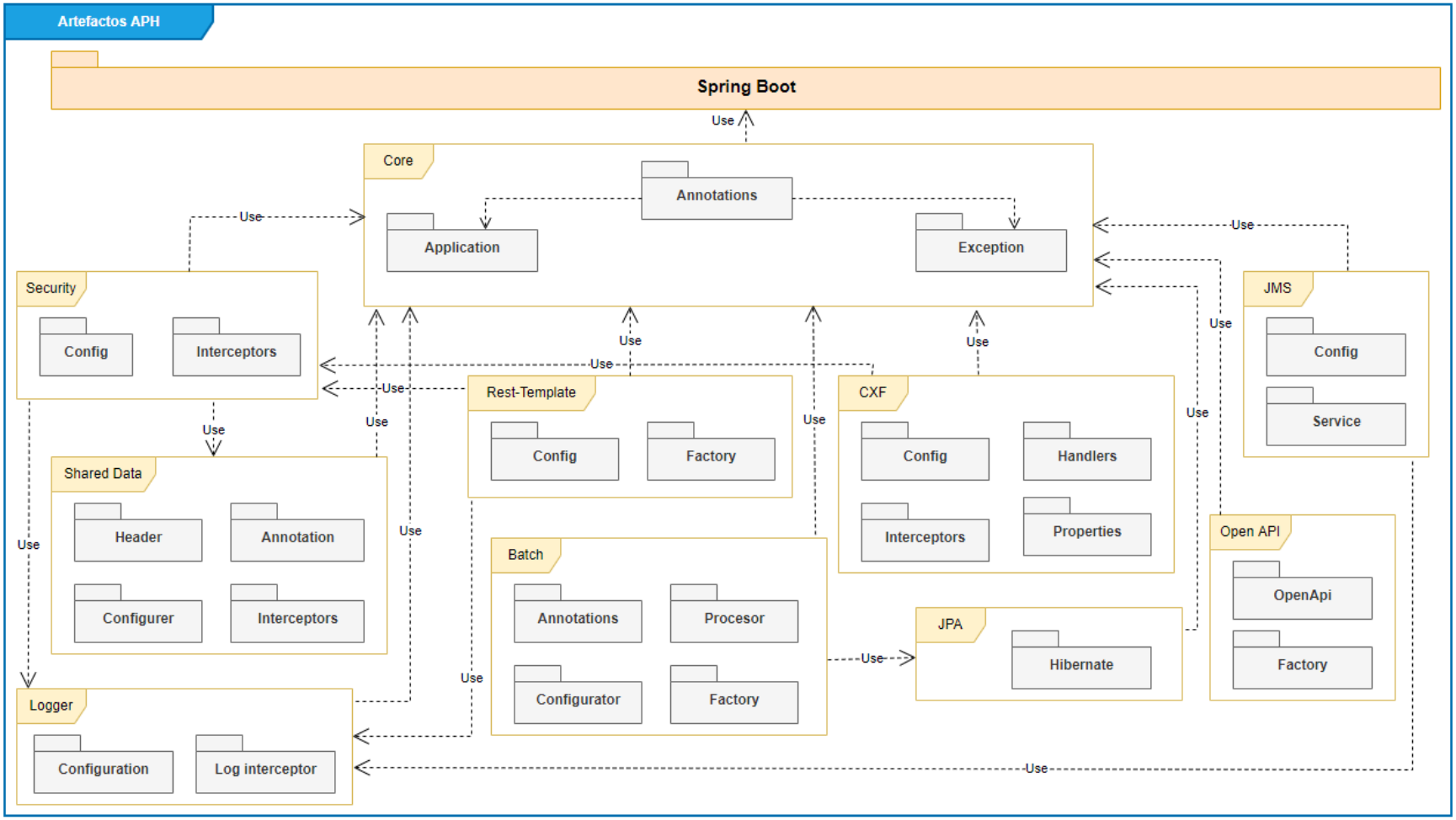
Gráfico 43: *Acceso a la arquitectura APH*



Fuente: Elaboración propia

4.2.2. Diseño de estructura de la solución

Gráfico 44: *Estructura de artefactos de la arquitectura APH*



Fuente: Elaboración propia

a) Core

Es el módulo principal de la arquitectura, principalmente contiene:

- **Annotations:** Gestión de anotaciones específicas que adaptan e integran la arquitectura de software dentro del Framework de Spring Boot.
- **Application:** Gestión de la clase principal de arranque de la aplicación que se apoyará en la arquitectura.

  Este paquete contiene la clase principal en la que se incluye y se gestiona las funcionalidades comunes que afectan a la aplicación de negocio. Alguna funcionalidad común necesaria para iniciar una aplicación de microservicios deberá ser llamada o implementada en esta clase.
- **Exception:** Gestión de las excepciones que se producirán tanto en la arquitectura, como en la aplicación de negocio.

  El sistema unificado de gestión de excepciones cubre dos aspectos importantes:

  - Por un lado, al tratarse de una arquitectura para aplicaciones que exponen servicios REST, la arquitectura gestiona todas las excepciones que se pueden producir en el protocolo HTTP. La arquitectura recubre todas las excepciones que se pueden producir en este sentido.
  - Por otro, se habilitan tres tipologías de excepciones que permitan la gestión de los errores producidos en la aplicación de negocio que se apoya sobre la arquitectura. Las tres tipologías son:
    - Excepciones de negocio (BusinessException): Durante la implementación de una aplicación de negocio, los desarrolladores

deben utilizar esta clase para lanzar las excepciones de negocio que consideran oportunas.

- Excepciones técnicas (TechnicalErrorException): Estas excepciones se corresponden con errores que no tienen relación con del negocio y su uso estará restringido a los problemas que surgen dentro del contexto técnico.
- Excepciones de seguridad (SecurityException): Excepciones referidas a la seguridad. Su uso es restringido al ámbito de la seguridad y/o autentificación. Como regla general este tipo de excepciones corresponde a la arquitectura.

El comportamiento que tiene predeterminado la arquitectura ante cada una de estas excepciones es el siguiente:

- Excepciones de negocio: Respuesta con código 422 y devolviendo el código el mensaje de error.
- Excepciones técnicas: Respuesta con código 500 y devolviendo el código y el mensaje de error.
- Excepciones de seguridad: Respuesta con código 400 y devolviendo el código y el mensaje de error.

Para aquellos microservicios que requieran un comportamiento diferente del establecido en la arquitectura debe utilizarse la clase ApplicationResponseEntityExceptionHandler donde se podrá sobrescribir el comportamiento de la arquitectura con un @Override del

manejador de la excepción cuyo comportamiento se desea particularizar.

Las constantes que se corresponden a los códigos de error y sus descripciones deben definirse en la clase AppErrorCode que se genera en cada microservicio. Así mismo debe evitarse definir códigos genéricos para mejorar la claridad y trazabilidad.

b) Security

Modulo que contiene los interceptores de las peticiones HTTP entrantes y salientes. Realiza el parseo del token JWT y se aplica la validación y/o extracción del contenido de dicho token.

Los interceptores de Spring son más adecuados que los filtros, ya que al ser Beans de Spring, se tiene acceso a todo el contexto desde ellos, y por lo tanto se puede implementar un comportamiento más sofisticado.

Desde el punto de vista de la seguridad, el planteamiento es interceptar todas las llamadas entrantes y salientes que se realicen a servicios REST.

- **Config:** Configuración del interceptor de peticiones entrantes y la configuración del redireccionamiento a la página de error con el código de error HTTP 403.
- **Interceptors:** Contiene los interceptores de peticiones entrantes y salientes donde se valida y gestiona el token JWT:
  - IncomingJWTInterceptor: Valida el token JWT y extrae su contenido.
  - OutgoingJWTInterceptor: Inserta el token JWT en las peticiones salientes (peticiones Restfull a otros microservicios)

c) RestTemplate

Este módulo proporciona la lógica de petición síncrona a otros microservicios a través de HTTP, controlando los errores a través de los convertidores de mensajes. Nos proporciona un enfoque simplificado con comportamientos predeterminados para realizar tareas complejas.

Config: Contiene la clase abstracta que permite la configuración alternativa; con y sin interceptor de peticiones salientes.

En su implementación se redefine el Bean de Spring RestTemplate para que incorpore el del interceptor de peticiones salientes, en caso de existir y registro de logs de las peticiones salientes.

d) CXF

Módulo que configura e integra la librería de Apache CXF en la arquitectura del software APH. Lo que aporta la librería Apache CXF es la posibilidad de comunicación con otros servicios mediante varios protocolos (SOAP, XML/HTTP, RESTful HTTP, CORBA). En la arquitectura APH se emplea para cubrir la necesidad de comunicación SOAP.

- **Config:** Define y configura los factory de CXF como Beans de Spring.
- **Handlers:** Manejadores para añadir la lógica de logs.
- **Interceptors:** Interceptor para la gestión de Timeout.
- **Properties:** Definición de las propiedades de timeout que serán usadas con CXF así como el prefijo Spring bajo el que se encuentra en un momento dado y poder modificarlas en los microservicios haciendo uso de los ficheros de propiedades.

e) Shared Data

Módulo que define el objeto y las propiedades que registra. En el objeto Shared Data viajará la cabecera de la petición y el token JWT. En este módulo también se definen los interceptores y manejadores necesarios para gestionar esta información. Hay que tener consideración que este módulo está estrechamente relacionado con el módulo de seguridad que lo consume para obtener el token y la cabecera de la petición entrante o para inyectar la cabecera en las peticiones salientes.

El paquete de contiene el objeto Shared Data así como su interceptor integrado con Spring MVC y la configuración de este interceptor para Spring.

f) JMS

En la arquitectura de software APH se integra la especificación JMS (Java Message Service) para conectar y operar con diferentes gestores de colas. Específicamente permite crear un registro de conexiones, tantas como gestores de colas queramos utilizar en todo momento.

Para la simplificación de la API JMS, está compuesto por el marco de integración JMS de Spring.

Config: Contiene la configuración y definición de un registro de conexiones integrado en Spring de forma que es accesible mediante inyección de dependencias.

Integración vía topic

Las integraciones con los gestores de colas se implementan con JMS, para ello, se hace uso del gestor de colas SQS (Amazon Simple Queue Service) que nos proporciona las siguientes características principales:

- Implementa y entiende el protocolo JMS.
- El servicio es auto escalable, por lo que se puede utilizar un número ilimitado de mensajes por cola, y las colas estándar proporcionan una capacidad de procesamiento casi ilimitada.
- Asegura la tolerancia a fallos y la entrega del mensaje. Se almacenan varias copias de cada mensaje de manera redundante en diferentes zonas de disponibilidad para que estén disponibles cuando se necesite.

Es importante destacar que al implementar el protocolo JMS y utilizarlo desde la arquitectura, el sistema se desacopla del proveedor del gestor de colas. Utilizando este protocolo, a pesar de usar el servicio de colas de AWS, no dependemos del proveedor Amazon, además podemos conectarnos siempre de la misma forma, tanto al gestor de colas corporativo del cliente (Tibco) como al de Amazon.

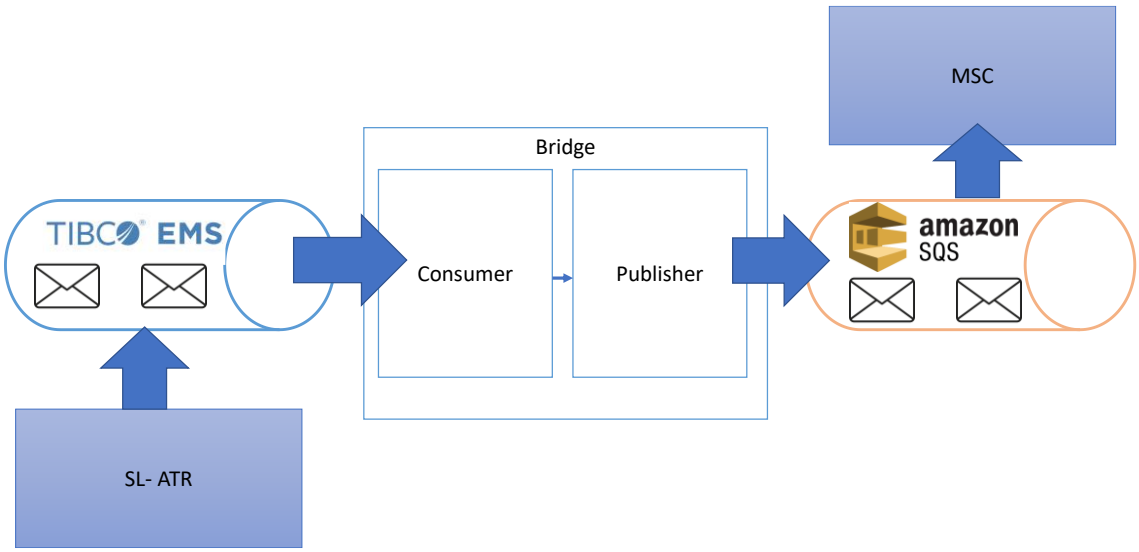
Para aquellas aplicaciones nuevas que necesitan utilizar topic que están implementadas en el gestor Tibco, pero que estas pasen a publicar en el nuevo gestor de colas de AWS, no es un proceso inmediato; sin embargo, debe existir un periodo de convivencia con los dos gestores de colas, para ello se plantea lo siguiente:

- Los sistemas de terceros On Premise siguen publicando mensajes en el gestor de colas de Tibco.

- Los servicios nuevos consumen mensajes del servicio SQS de AWS.
- Se debe implementar un pequeño componente de software “Bridge” (no es parte de la arquitectura APH) que tendrá un consumidor de la cola de Tibco y publicará los mensajes en la cola de AWS. De esta forma podrán convivir los sistemas cada uno publicando y consumiendo de donde corresponda.

En la siguiente gráfica se ilustra como los sistemas On Premise (SL-ATR) publican al gestor de colas Tibco y a través de un Brige se publican al servicio de SQS y estas son consumidas por las aplicaciones nuevas (MSC).

Gráfico 45: *Simulación del gestor de colas*



Fuente: Elaboración propia

- Una vez que todos los sistemas On Premise se hayan adaptado para publicar en SQS, se podrá desconectar el sistema sin tener que realizar ninguna modificación en las aplicaciones nuevas.

g) Batch Core

Para aislar a los equipos de desarrollo de la inclusión de la librería Spring Batch, de la implementación de las clases necesarias y la definición de los n

pasos (steps) del proceso, se ha implementado una serie de anotaciones específicas que nos permite:

- La posibilidad de añadir datasource propio para la gestión de Spring Batch (para la gestión de job, ejecuciones, steps, etc.).
- Se habilita una serie de anotaciones para simplificar, en la medida de lo posible, la definición del Job y los Steps de proceso por lotes.

En este módulo el framework de Spring batch que está enfocado en la creación de procesos batch. Principalmente se añade algunas características a la funcionalidad propia de Spring:

- **Activación de las funcionalidades de Batch:** Se realiza a través de la anotación @EnableBatch. Esta anotación activará tanto el Spring Batch como la funcionalidad extra.
- **Datasource:** Si bien Spring Batch puede tanto utilizar el origen de datos de nuestra aplicación como un datasource diferente. En la arquitectura APH podremos crear un datasource completamente distinto introduciendo en el fichero de propiedades del proyecto.
- **Anotaciones de ayuda:** Se diseña una serie de anotaciones con la idea de simplificar en la medida de lo posible la definición del Job y los Steps. Por supuesto estas anotaciones no contemplan todas las casuísticas y todas las probabilidades que ofrece Spring Batch; si se requieren algunas de estas características no implementadas se puede acudir a la definición vía Beans de Spring Batch

- @ConfigBatchJob: Define un job. Contiene como atributos los siguientes:
    - BeanName: Nombre que tendrá el Bean que define el Job y el Job en sí.
    - Steps: Lista ordenada de los Steps directamente vinculados al Job.
    - Restart: Indica si el Job puede relanzarse o no.
- @ConfigTaskletStep: Define un Step de tipo Tasklet:
    - BeanName: Nombre del Step y del Bean asociado.
    - Task: Clase que extiende de Task y que implementa la lógica de negocio realizada por este Step.
    - AllowStartComplete: Indica si el Step puede relanzarse una vez completado o no (tiene sentido si relanzas una instancia de Job concreto ya ejecutado). Por defecto falso.
    - ReturnBuilder: Indica si el Bean generado por la anotación se corresponde con un Step (false) o con un TaskletStepBuilder (true). Por defecto falso.
- @ConfigPartitionStep: Define un Step de tipo PartitionStep.
    - BeanName: Nombre del Step y del Bean asociado.
    - AllowStartComplete: Indica si este Step puede relanzarse una vez completado o no.
    - GridSize: Es definido para el PartitionStep.

- ReturnBuilder: Indica si el Bean generado por la anotación se corresponde con un Step (false) o con un PartitionStepBuilder (true). Por defecto falso.
- Partitioner: Nombre del Bean que define el Partitioner a utilizar para el Step, por defecto ninguno.
- Handler: Nombre del Bean que define el PartitionHandler a utilizar en el Step, por defecto ninguno.
- Step: Indica el Bean del Step que será invocado por cada partición.

- @ConfigChunkStep: Define un Step de tipo Chunk.
  - BeanName: Nombre del Step y del Bean asociado.
  - AllowStartComplete: Indica si el Step puede relanzarse una vez completado o no.
  - GridSize: El gridSize definido para el Chunk.
  - ReturnBuilder: Indica si el Bean generado por la anotación se corresponde con un Step (false) o con un SimpleStepBuilder<I,O> (true). Por defecto falso.
  - InputClass: Se corresponde con el tipo “I” del tipado <I,O> requerido por los factory SimpleChunkFactory y SimpleStepBuilder.
  - OutputClass: Se corresponde con el tipo “O” del tipado <I,O> requerido por los factory SimpleChunkFactory y SimpleStepBuilder.
  - ReaderBean: Bean que define el ItemReader.

- WriterBean: Bean que define el ItemWriter.
- BeanProcess: Bean que define el ItemProcessor.

En cuanto a la estructura de paquetes de este componente disponemos de lo siguiente:

- **Annotations:** Contiene todas las anotaciones definidas tanto como ayuda como la necesaria para activar el Spring Batch.
- **Configurator:** Contiene el configurador que lee las anotaciones y crea los Beans asociados junto con los Factory para instanciar estos Bean.
- **Factory:** Factorys que instanciarán los Job y Steps de las anotaciones.
- **Processor:** Únicamente contiene la clase BatchHelper ya referida con anterioridad.
- **Spring:** Contiene las clases necesarias para la integración con Spring tanto de las anotaciones como del datasource propio de Spring Batch.

La arquitectura generará dinámicamente todo el código necesario para disponer de estas funcionalidades y poder convertir la lógica de negocio del microservicio en un proceso que ejecutará controlado por Spring Batch.

h) Logger

Dentro de la arquitectura APH se incorpora Logback para Spring como sistema de log de toda la arquitectura. Cualquier funcionalidad de esta librería puede ser utilizada en cualquier aplicación que se apoye sobre la arquitectura de software.

Se tiene por objetivo que cada microservicio genere un fichero, sin entrar en disquisiciones de si uno o varios, ya que el conjunto de estos sea factible de

centralizar las trazas en un sistema (CloudWatch) que permita su almacenamiento y procesamiento.

Los ficheros serán generados sobre volúmenes físicos y que sigan una política de ficheros rotatorios (por tamaño) que nos permita controlar la eliminación o pasar a históricos los ficheros generados para aligerar los volúmenes y no crecer infinitamente.

Seguimiento: Por cada petición a un servicio REST se incorporan parámetros denominados "idTransaccion" que es un identificador único por cada petición y el "hiperCodeUsername" que son incorporados de forma transparente para el desarrollador. La arquitectura propaga estos parámetros a lo largo de todas las llamadas RESTfull entre microservicios.

Cada microservicio generará un fichero de log y los desarrolladores deberán incorporar el parámetro que consideren en cada una de las trazas para poder después correlacionar y hacer un seguimiento.

i) Data JPA

Un aspecto resaltante que incluir en la arquitectura de software APH es el Spring Data, que es uno de los frameworks que se encuentra dentro del marco de Spring. Su objetivo es simplificar el desarrollo a través de la persistencia de datos contra distintos repositorios de información.

El módulo Data JPA, nos permitirá el acceso a los datos, como bases de datos no relacionales, marcos de reducción de mapas, servicios en la nube, así como compatibilidad con bases de datos relacionales muy avanzadas.

j) Open API

Una característica fundamental en todo sistema es la documentación, para ello en la arquitectura de software APH se incluye el Open API, que es un estándar para la descripción de las interfaces de programación.

Uno de los beneficios principales del uso de Open API es para la documentación o descripción de tu API (Interfaz de programación de aplicaciones), este módulo nos permitirá incluir de manera sencilla en todos los microservicios las especificaciones estándar de Swagger 3 de Open API.

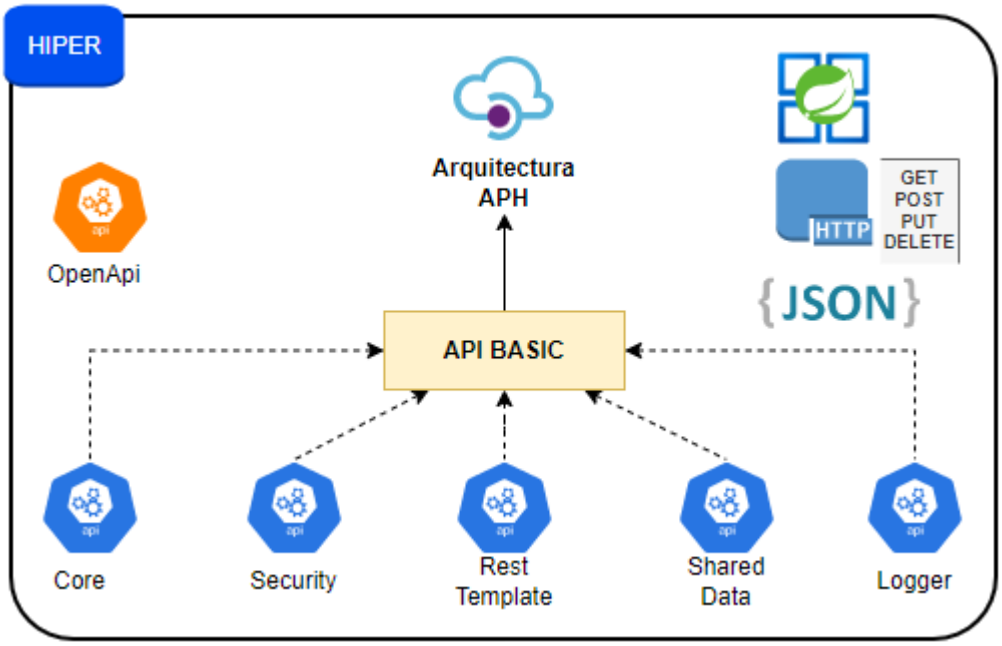
4.2.3. Diseño de la funcionalidad de la solución

A continuación, se presenta los diseños de las funcionalidades que compone la arquitectura APH.

a) API básico

En base a la arquitectura APH, se genera un microservicio básico incluido sus componentes (artefactos) principales. Esta permite las diferentes peticiones HTTP para realizar las operaciones necesarias en nuestra fuente de datos. Además, se puede resaltar que, si bien el OpenApi está integrado dentro de la arquitectura, de forma predeterminada no está incluida al generar los microservicios, esta con el objetivo que los componentes como OpenApi son opcionales y dejamos en libre elección de incluir por el equipo desarrollador.

Gráfico 46: *Diseño de componentes de API básico*



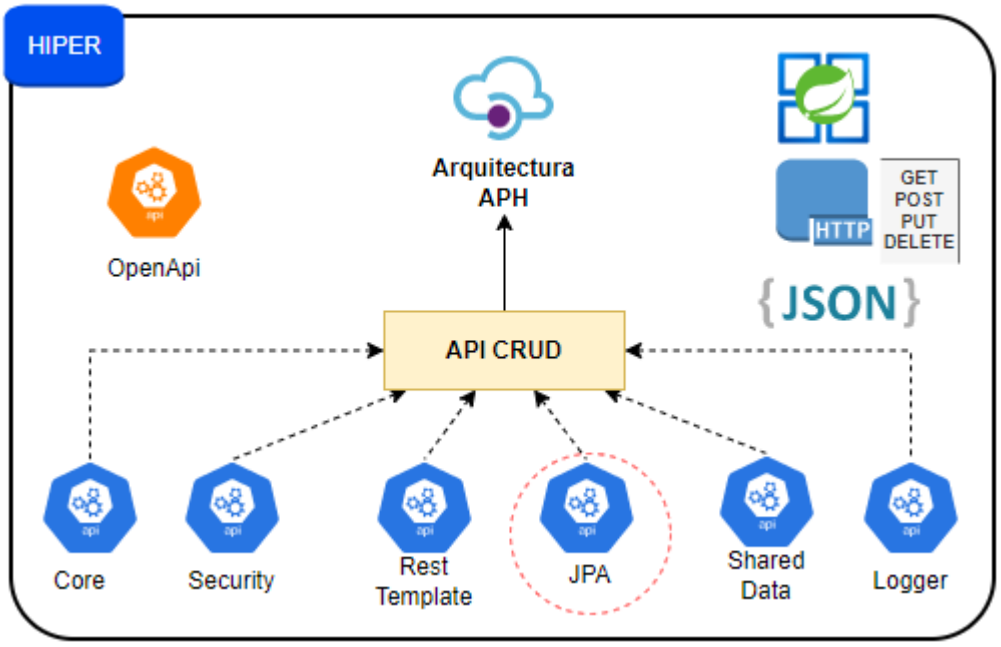
Fuente: Elaboración propia

b) API CRUD

El proceso de generación del API CRUD tiene el flujo similar como el básico. Además de incluir los componentes principales, podemos observar que se incluye el módulo de JPA que nos va a permitir la manipulación de datos en diferentes motores de bases de datos como Oracle, SQL Server, PostgreSQL y MySQL.

Es importante resaltar que la dependencia específica para realizar la conexión a los motores de bases de datos se define en cada microservicio, esto con la finalidad que el equipo desarrollador tenga la libertad de elección de acuerdo con las necesidades del proyecto.

Gráfico 47: *Diseño de componentes de API CRUD*



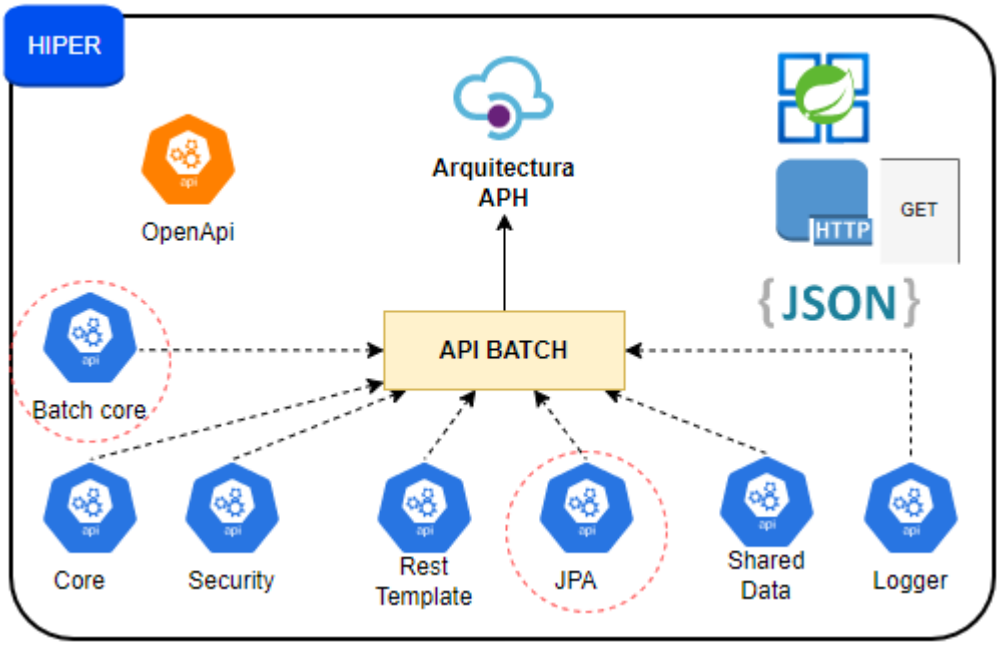
Fuente: Elaboración propia

c) API Batch

El arquetipo Api Batch, sigue el mismo flujo de generación como los arquetipos anteriores, en esta se incluye el módulo de Batch Core, que tiene las funcionalidades para configurar la ejecución de tareas programadas. Además, estos microservicios estarán en la capacidad de responder las peticiones configuradas desde Control-M.

Es importante resaltar, que la única petición aceptada por el microservicio es el GET, ya que tiene por objetivo de ejecutar tareas con determinados parámetros y no es necesario realizar otro tipo de peticiones HTTP.

Gráfico 48: *Diseño de componentes de API Batch*



Fuente: Elaboración propia

4.2.4. Construcción de la solución

Herramientas de desarrollo

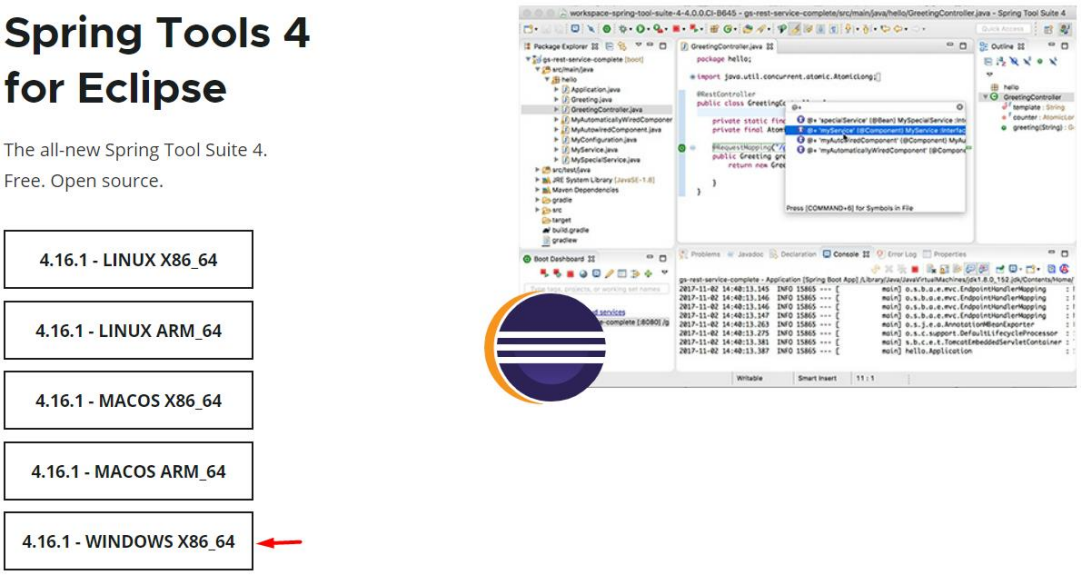
a) Herramientas CASE

Para el desarrollo de la arquitectura se requirió de los siguientes aplicativos:

- Spring Tool Suite 4

  Herramienta de Spring para el entorno de codificación de clase mundial, que nos permite desarrollar aplicaciones empresariales basadas en Spring.

Gráfico 49: *Spring Tools 4 for Eclipse*

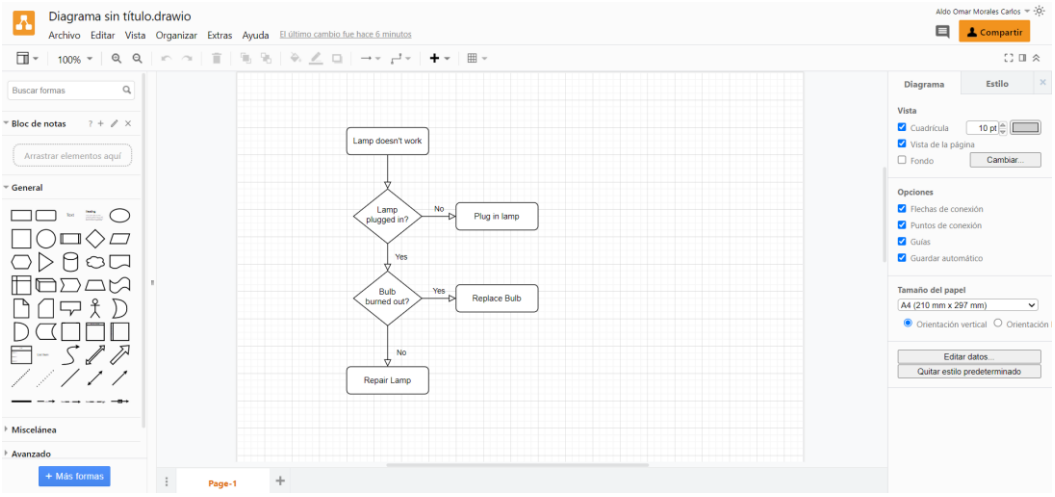


Fuente: Elaboración propia

- Diagrams.net

Software de dibujo gráfico multiplataforma gratuito y de código abierto. Su interfaz se puede utilizar para crear diferentes diagramas de flujo, UML, organigramas, diagramas de red, etc.

Gráfico 50: *Software diagrams.net*



Fuente: Elaboración propia

b) Framework de desarrollo

La arquitectura APH, se desarrolló en el lenguaje de programación Java, haciendo uso del framework Spring que nos proporciona un marco integral de infraestructura para desarrollar aplicaciones Java.

Gráfico 51: *Framework Spring*

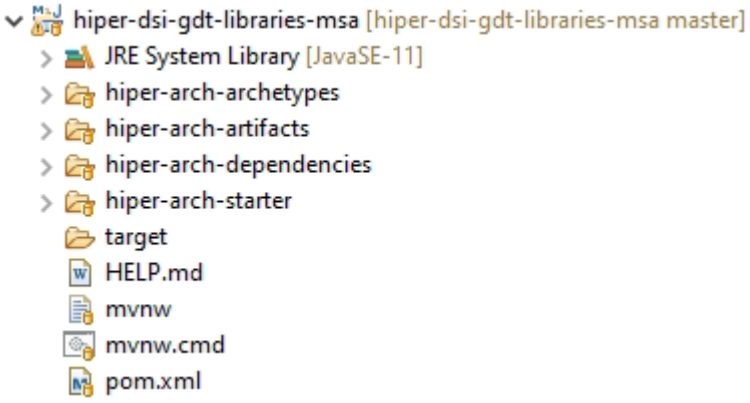
Fuente: Elaboración propia

Proceso de desarrollo de la arquitectura

a) Estructura de la arquitectura APH

La arquitectura APH está compuesto por los arquetipos, artefactos, las dependencias y el módulo principal para iniciar (starter) los microservicios.

Gráfico 52: *Estructura de la arquitectura APH*



Fuente: Elaboración propia

El modelo de objeto del proyecto de la arquitectura APH incluye los módulos principales que nos permitirá la generación de nuevos microservicios con todas las configuraciones y dependencias personalizadas de la arquitectura.

Dentro de esta configuración tenemos el repositorio de GitHub donde se publica la arquitectura APH empaquetado todos sus módulos y dependencias.

Gráfico 53: *POM de la arquitectura APH*

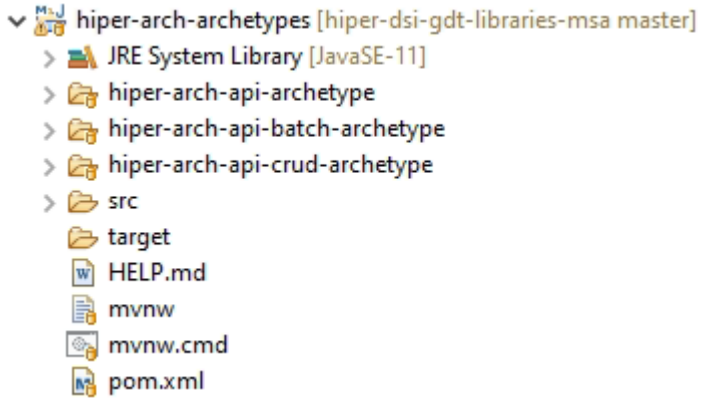
```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.7.2</version>
    <relativePath /> <!-- lookup parent from repository -->
</parent>
<groupId>pe.com.hiper.arch</groupId>
<artifactId>hiper-arch-build</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>pom</packaging>
<name>hiper-dsi-gdt-libraries-msa</name>
<description>Hiper Arch Build</description>
<properties>
    <arch.version>0.0.1-SNAPSHOT</arch.version>
    <java.version>11</java.version>
    <maven.compiler.target>${java.version}</maven.compiler.target>
    <maven.compiler.source>${java.version}</maven.compiler.source>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    <lombok.version>1.18.20</lombok.version>
</properties>


<modules>
    <module>hiper-arch-dependencies</module>
    <module>hiper-arch-starter</module>
    <module>hiper-arch-artifacts</module>
    <module>hiper-arch-archetypes</module>
</modules>

<distributionManagement>
    <repository>
        <id>github</id>
        <name>Repositorio de arquitectura microservicios (MSA)</name>
        <url>https://maven.pkg.github.com/aldomar-mc/hiper-dsi-gdt-libraries-msa</url>
    </repository>
</distributionManagement>
```

Fuente: Elaboración propia

b) Arquetipos de la arquitectura APH

Los arquetipos de la arquitectura APH está compuesto por el API básico, API CRUD y el API Batch.

Gráfico 54: *Arquetipos de la arquitectura APH*



Fuente: Elaboración propia

En el modelo de objeto encontramos la configuración de los módulos de los arquetipos implementados.

Gráfico 55: *POM de los arquetipos de la arquitectura APH*

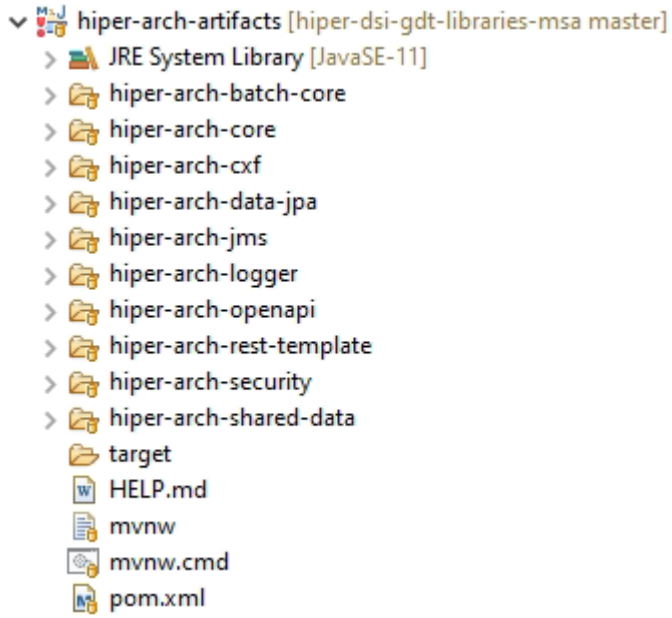
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="ht
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https:,
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>pe.com.hiper.arch</groupId>
        <artifactId>hiper-arch-dependencies</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <relativePath>../hiper-arch-dependencies</relativePath>
    </parent>

    <artifactId>hiper-arch-archetypes</artifactId>
    <packaging>pom</packaging>
    <name>hiper-arch-archetypes</name>
    <description>Hiper Arch Archetypes</description>

    <modules>
        <module>hiper-arch-api-archetype</module>
        <module>hiper-arch-api-crud-archetype</module>
        <module>hiper-arch-api-batch-archetype</module>
    </modules>

</project>
```

Fuente: Elaboración propia

c) Artefactos de la arquitectura APH

Dentro de esta capa nos encontramos con los diferentes artefactos modularizados, su implementación sigue el enfoque de microservicios que están interrelacionados unos con otros.

Gráfico 56: *Artefactos de la arquitectura APH*



Fuente: Elaboración propia

En el modelo de objeto del proyecto de los artefactos compuestos en la arquitectura APH, encontramos la configuración de todos los módulos implementados.

Importante: Los módulos de la arquitectura APH tienen un enfoque de los microservicios, considerando la escalabilidad que en el futuro podemos incluir nuevos módulos dentro de la arquitectura.

Gráfico 57: *POM de los artefactos de la arquitectura APH*

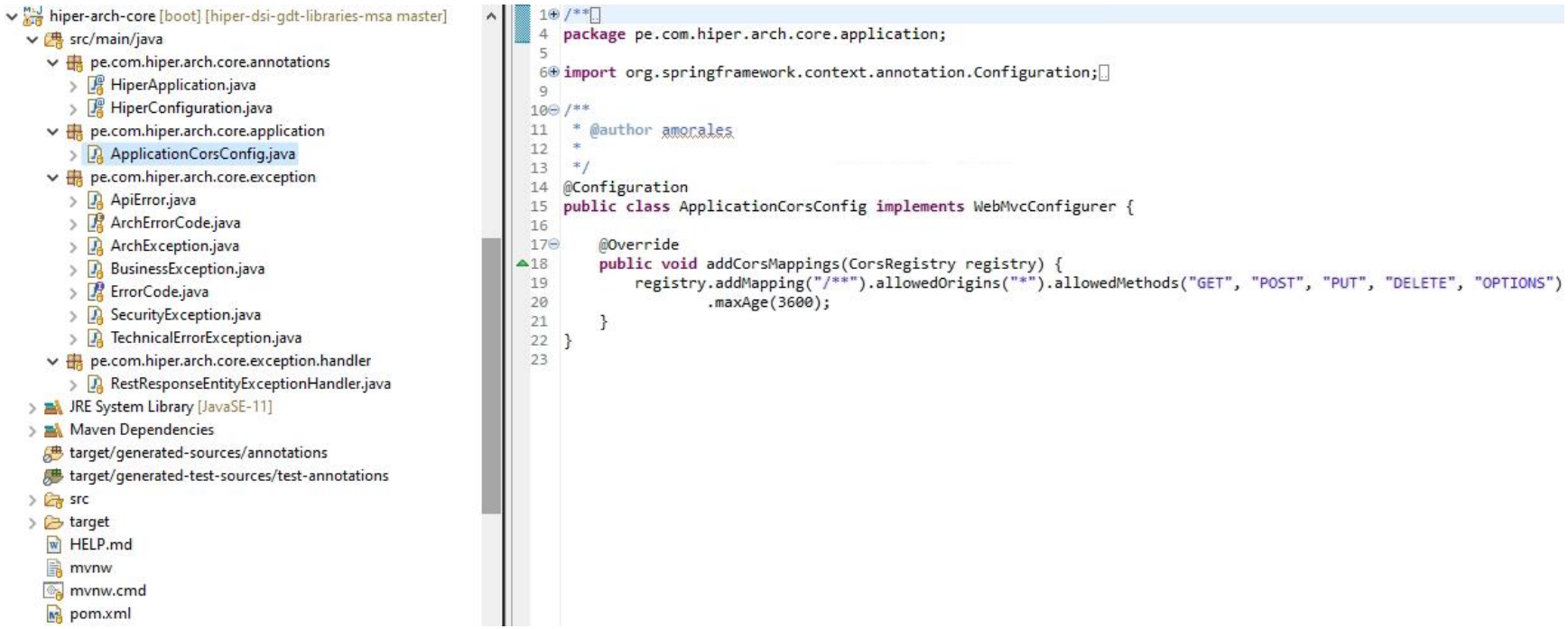
```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https:/
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>pe.com.hiper.arch</groupId>
        <artifactId>hiper-arch-dependencies</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <relativePath>../hiper-arch-dependencies</relativePath>
    </parent>

    <artifactId>hiper-arch-artifacts</artifactId>
    <packaging>pom</packaging>
    <name>hiper-arch-artifacts</name>
    <description>Hiper Arch Artifacts</description>

    <build>..

    <modules>
        <module>hiper-arch-core</module>
        <module>hiper-arch-data-jpa</module>
        <module>hiper-arch-cxf</module>
        <module>hiper-arch-shared-data</module>
        <module>hiper-arch-logger</module>
        <module>hiper-arch-security</module>
        <module>hiper-arch-rest-template</module>
        <module>hiper-arch-batch-core</module>
        <module>hiper-arch-jms</module>
        <module>hiper-arch-openapi</module>
    </modules>
</project>
```

Fuente: Elaboración propia

Módulo Core de la arquitectura APH

Es el módulo principal de la arquitectura, está compuesto por las anotaciones, la clase principal de la aplicación y la definición de las excepciones técnicas.

Gráfico 58: *Estructura del módulo Core de la arquitectura APH*

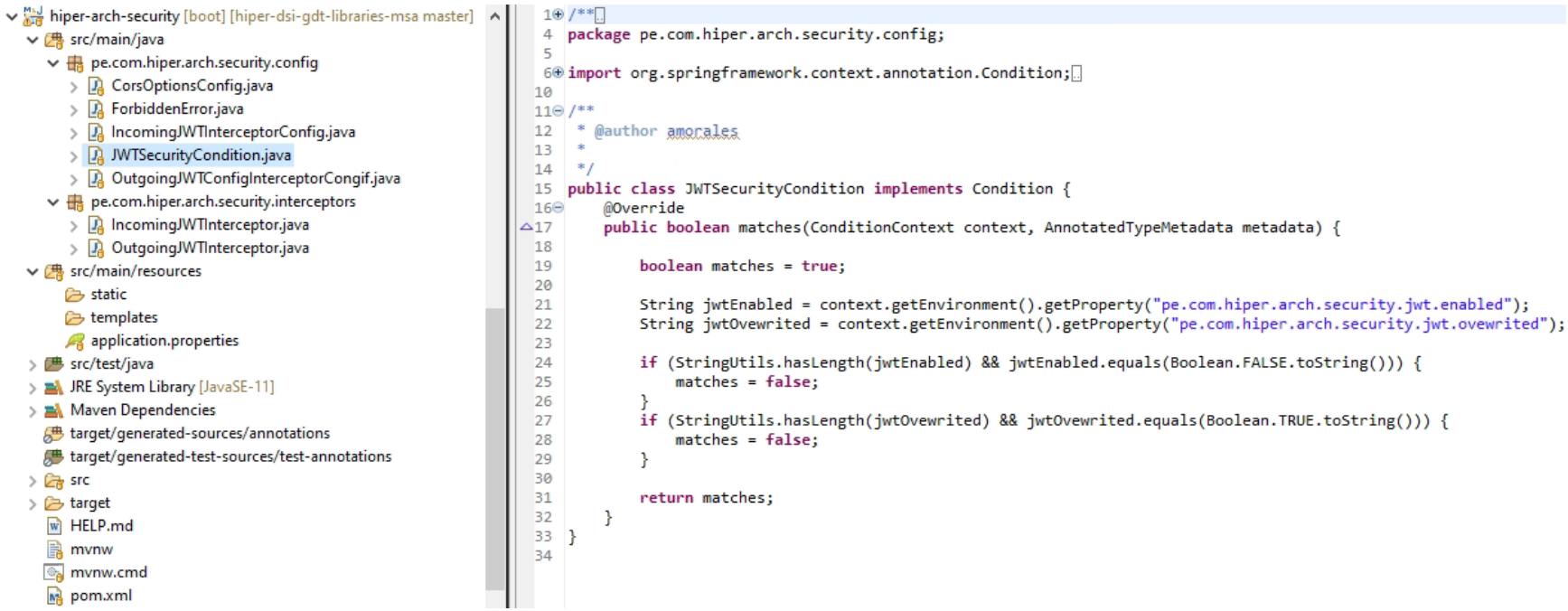


Fuente: Elaboración propia

Módulo de seguridad de la arquitectura APH

En el presente módulo se incluye la seguridad a través de JWT que nos permite tener un mejor control para acceder a diferentes microservicios implementados bajo la arquitectura APH.

Gráfico 59: *Módulo de seguridad de la arquitectura APH*

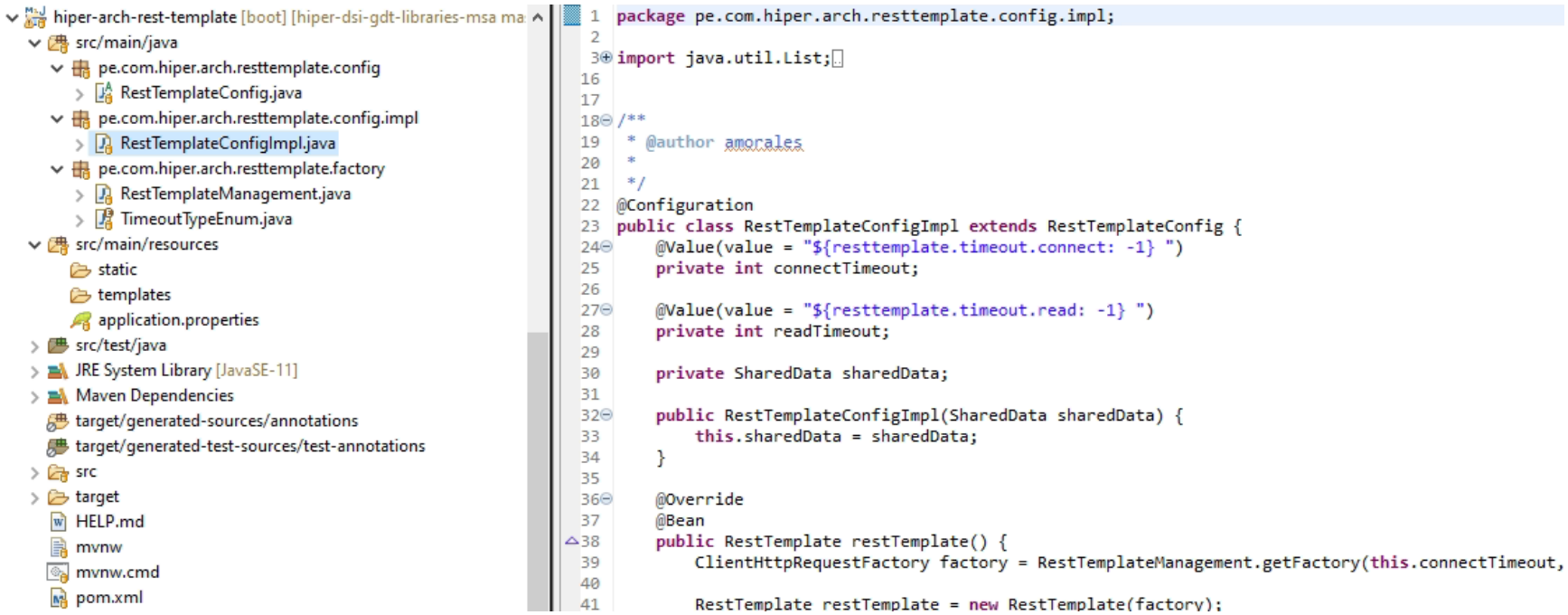


Fuente: Elaboración propia

Módulo de acceso desde servicios REST a la arquitectura APH

El presente módulo implementa y personaliza las funcionalidades de RestTemplate de Spring, que nos permite el acceso desde la parte de los clientes a nuestros servicios REST desarrollados por la organización Hiper.

***Gráfico 60**: Módulo de acceso desde servicios REST a la arquitectura APH*

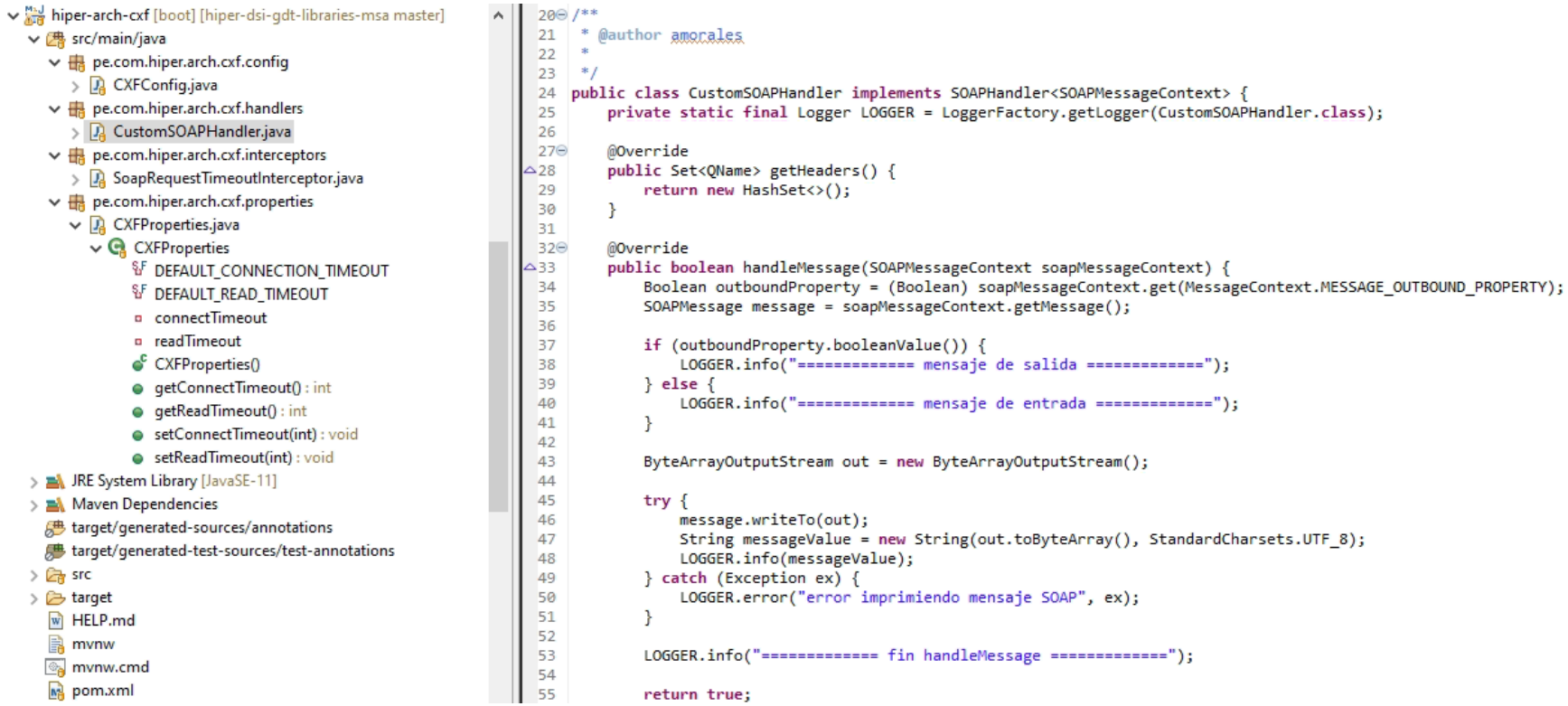


Fuente: Elaboración propia

Módulo de acceso desde servicios SOAP a la arquitectura APH

Dentro de la arquitectura APH se emplea para cubrir la comunicación SOAP desde la parte de los clientes a nuestros servicios REST.

Gráfico 61: *Módulo de acceso desde servicios SOAP a la arquitectura APH*

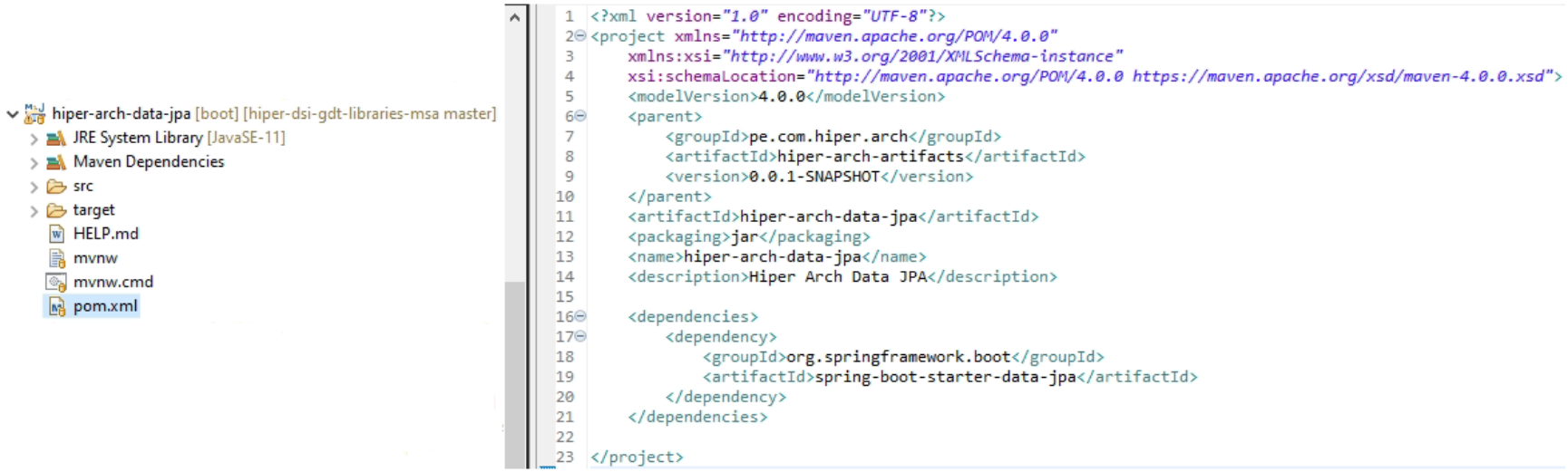


Fuente: Elaboración propia

Módulo de JPA en la arquitectura de APH

El módulo de Data JPA, está modularizado por una dependencia que nos provee el marco de desarrollo Spring, esta nos permite la manipulación de datos al tener conexión a los diferentes motores de bases de datos.

Gráfico 62: *Módulo de JPA en la arquitectura de APH*



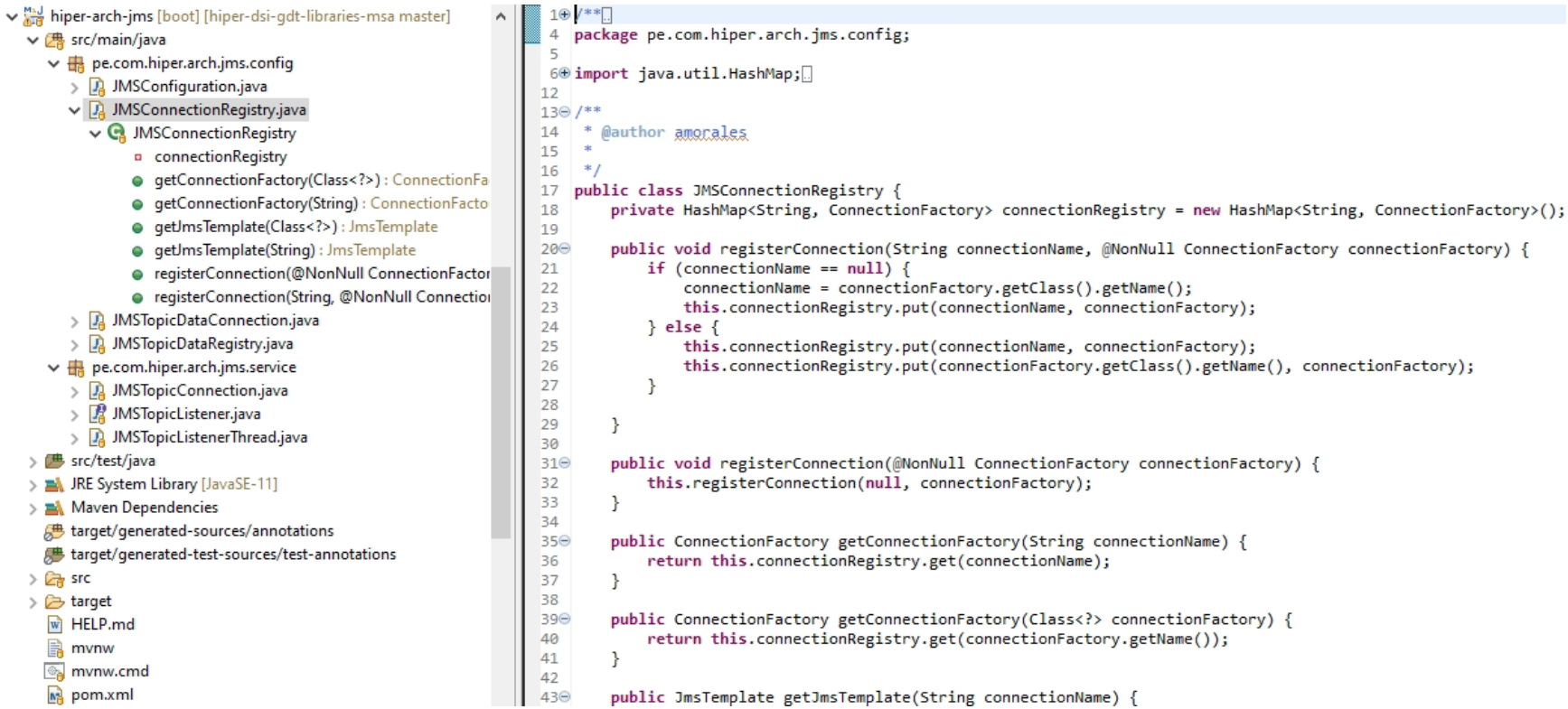
Fuente: Elaboración propia

Módulo de JMS en la arquitectura APH

En el presente módulo se implementa las especificaciones de JMS (Java Message Service) que nos permite conectar y operar con diferentes gestores de colas (Amazon SQS).

Gráfica X:

Gráfico 63: *Módulo de JMS en la arquitectura APH*

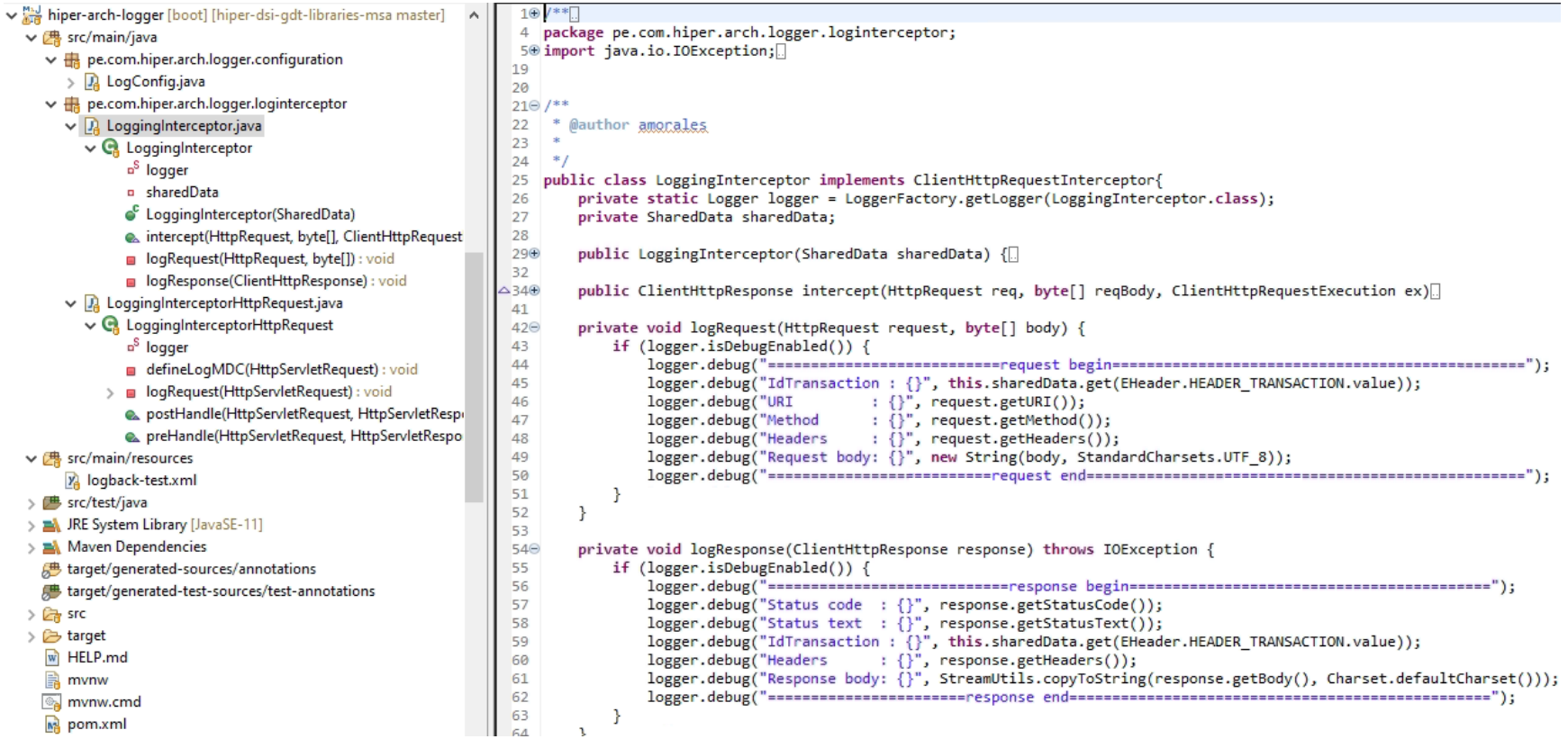


Fuente: Elaboración propia

Módulo de registro de logs en la arquitectura APH

En la arquitectura APH se incorpora Logback que nos provee el marco de desarrollo Spring, el objetivo de modularizar es que cualquier funcionalidad de esta librería sea utilizada en los microservicios que se apoye sobre la arquitectura APH.

Gráfico 64: *Módulo de registro de logs en la arquitectura APH*

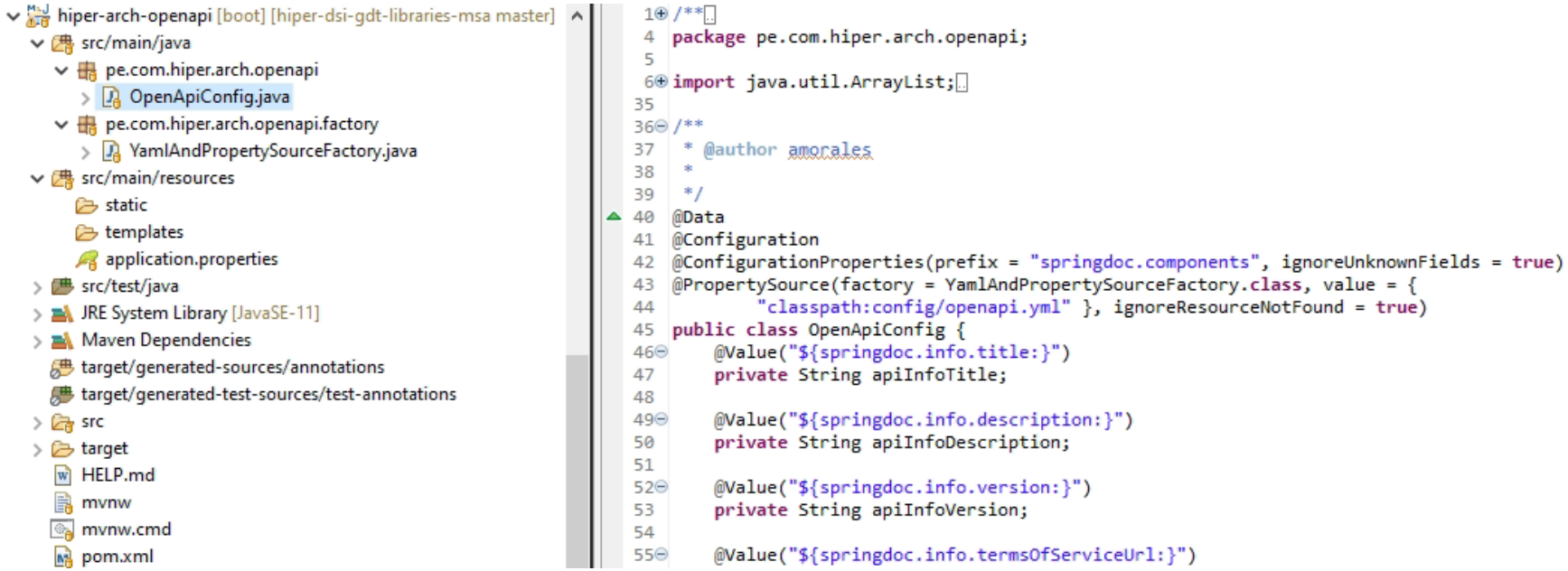


Fuente: Elaboración propia

Módulo de documentación en la arquitectura APH

Sobre la arquitectura APH se incluye el Open API, para describir de una manera sencilla en todos los microservicios las especificaciones estándar de Swagger 3 de Open API.

Gráfico 65: *Módulo de documentación en la arquitectura APH*

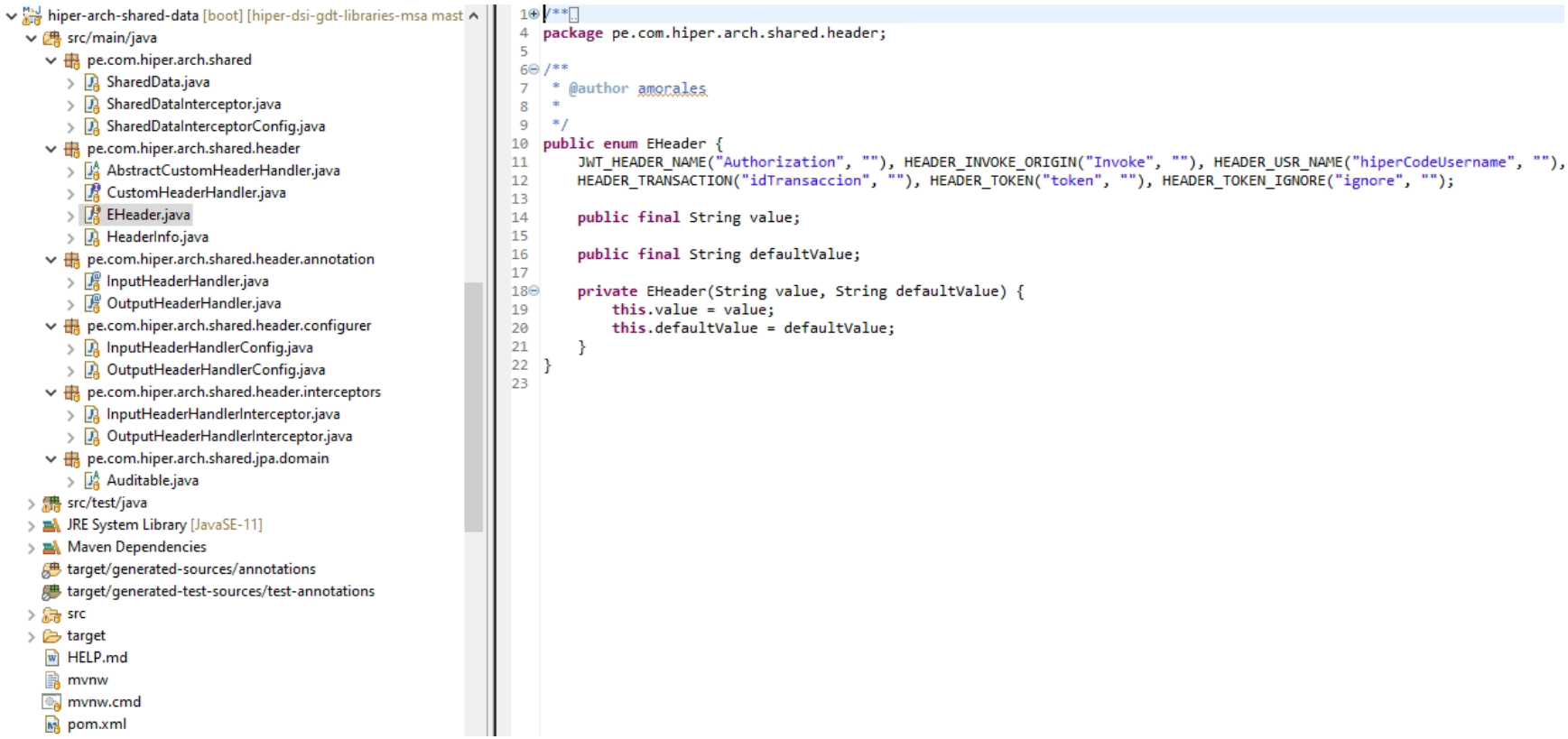


Fuente: Elaboración propia

Módulo de datos compartidos en la arquitectura APH

Módulo que define los datos compartidos para ser usados por los módulos que son parte de la arquitectura APH. Este módulo está estrechamente relacionado con el módulo de seguridad que lo consume para obtener el token y la cabecera de las peticiones salientes.

Gráfico 66: *Módulo de datos compartidos en la arquitectura APH*

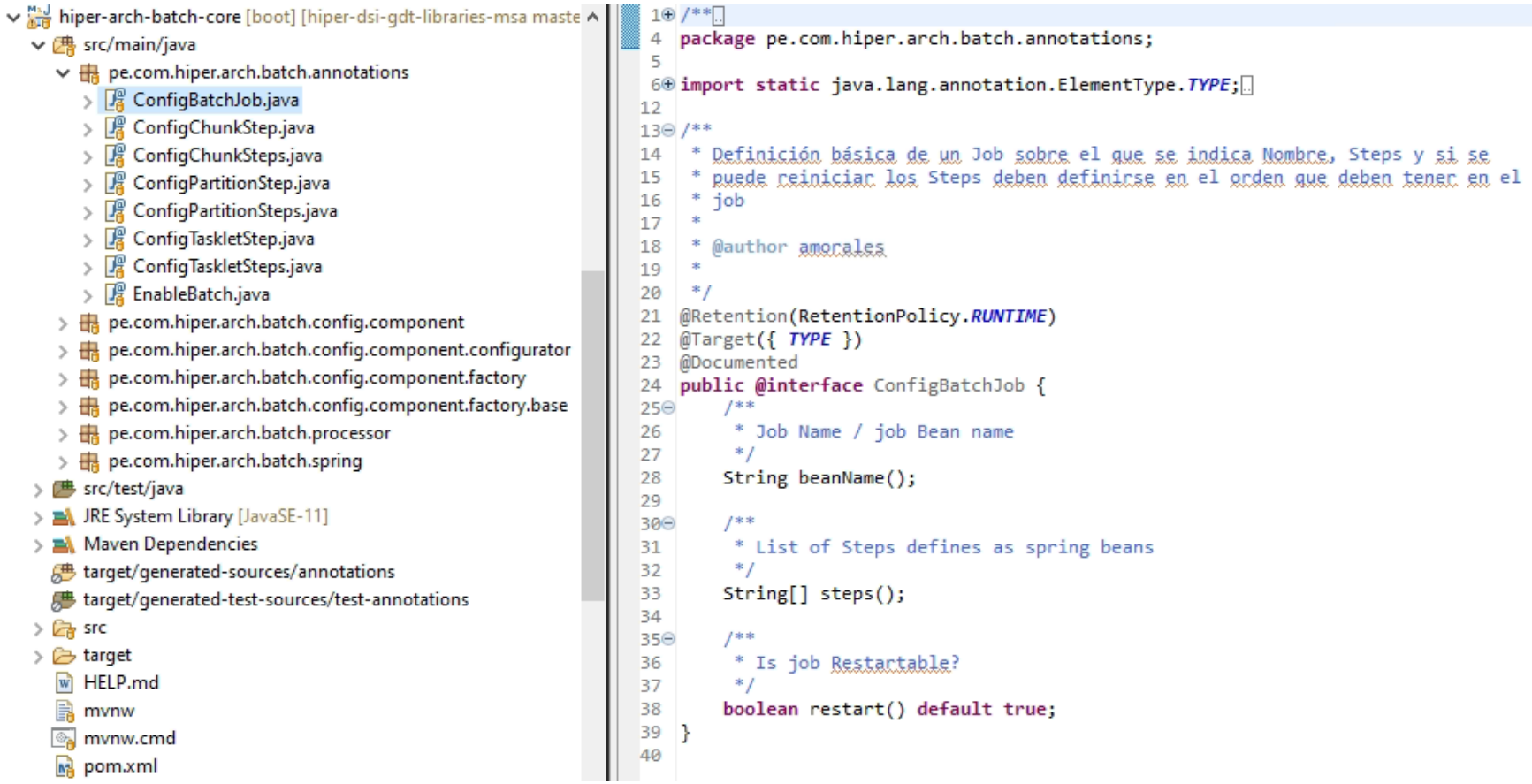


Fuente: Elaboración propia

Módulo de tareas en la arquitectura APH

Dentro de la arquitectura APH se integra el Spring Batch, con el objetivo de aislar a los equipos de desarrollo la inclusión de esta librería, además, se implementa una serie de anotaciones específicas que nos facilita durante la etapa de desarrollo.

Gráfico 67: *Módulo de tareas en la arquitectura APH*

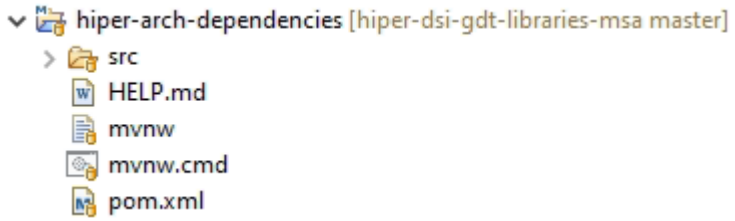


Fuente: Elaboración propia

d) Módulo de dependencias

La gestión de dependencias se realiza a través de este módulo, donde se define las dependencias que corresponden propiamente de la arquitectura APH, así como también librerías externas que son necesarias para suplir las necesidades durante el desarrollo.

Gráfico 68: *Gestión de dependencias en la arquitectura APH*



Fuente: Elaboración propia

Gráfico 69: *Dependencias externas en la arquitectura APH*

```
<dependency>
    <groupId>ch.qos.logback.contrib</groupId>
    <artifactId>logback-json-classic</artifactId>
    <version>${logback-jackson.version}</version>
</dependency>
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>${gson.version}</version>
</dependency>
<dependency>
    <groupId>com.nimbusds</groupId>
    <artifactId>nimbus-jose-jwt</artifactId>
    <version>${nimbus-jose-jwt.version}</version>
</dependency>
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>${javax.servlet-api.version}</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.modelmapper</groupId>
    <artifactId>modelmapper</artifactId>
    <version>${modelmapper.version}</version>
</dependency>
```

Fuente: Elaboración propia

Gráfico 70: *Dependencias internas de la arquitectura APH*

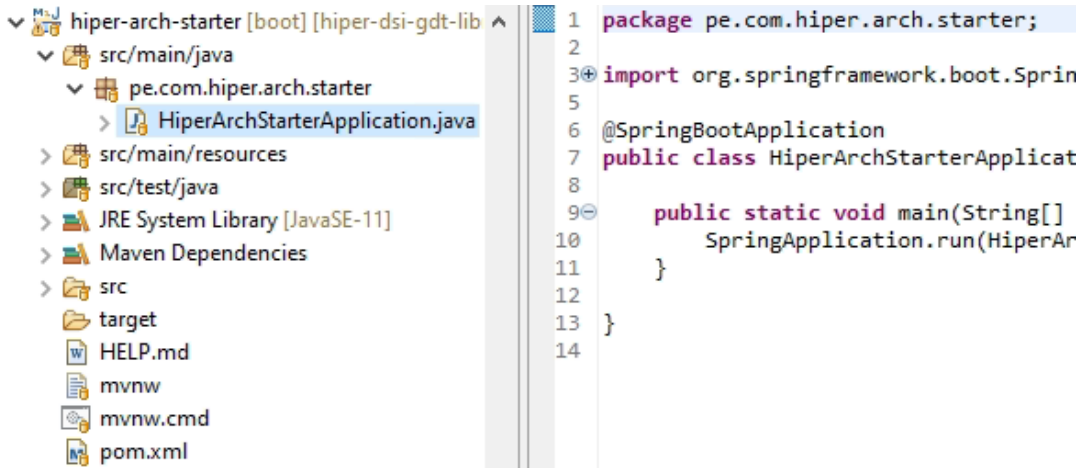
```
<dependency>
    <groupId>pe.com.hiper.arch</groupId>
    <artifactId>hiper-arch-core</artifactId>
    <version>${arch.version}</version>
</dependency>
<dependency>
    <groupId>pe.com.hiper.arch</groupId>
    <artifactId>hiper-arch-data-jpa</artifactId>
    <version>${arch.version}</version>
</dependency>
<dependency>
    <groupId>pe.com.hiper.arch</groupId>
    <artifactId>hiper-arch-cxf</artifactId>
    <version>${arch.version}</version>
</dependency>
<dependency>
    <groupId>pe.com.hiper.arch</groupId>
    <artifactId>hiper-arch-shared-data</artifactId>
    <version>${arch.version}</version>
</dependency>
<dependency>
    <groupId>pe.com.hiper.arch</groupId>
    <artifactId>hiper-arch-logger</artifactId>
    <version>${arch.version}</version>
</dependency>
<dependency>
    <groupId>pe.com.hiper.arch</groupId>
    <artifactId>hiper-arch-security</artifactId>
    <version>${arch.version}</version>
</dependency>
<dependency>
    <groupId>pe.com.hiper.arch</groupId>
    <artifactId>hiper-arch-rest-template</artifactId>
    <version>${arch.version}</version>
</dependency>
<dependency>
    <groupId>pe.com.hiper.arch.batch</groupId>
    <artifactId>hiper-arch-batch-core</artifactId>
    <version>${arch.version}</version>
</dependency>
<dependency>
    <groupId>pe.com.hiper.arch</groupId>
    <artifactId>hiper-arch-jms</artifactId>
    <version>${arch.version}</version>
</dependency>
<dependency>
    <groupId>pe.com.hiper.arch</groupId>
    <artifactId>hiper-arch-openapi</artifactId>
    <version>${arch.version}</version>
</dependency>
```

Fuente: Elaboración propia

e) Módulo de inicio de la aplicación en la arquitectura APH

En la arquitectura APH el punto de inicio de la aplicación se aísla de los otros módulos o componentes, con el fin de tener una configuración propia del marco de trabajo Spring.

Gráfico 71: *Módulo de inicio de la aplicación en la arquitectura APH*



Fuente: Elaboración propia

En el modelo de objeto del proyecto se define la dependencia principal de Spring Boot que nos permitirá ejecutar los microservicios, así lo mismo, se incluye una dependencia enfocada a las pruebas unitarias que son necesarias para cubrir la cobertura de la lógica de negocio que se implementa en los microservicios dependientes de la arquitectura APH.

Importante:

El presente módulo es dependiente del módulo de dependencias, que nos provee todas las librerías tanto internas como externas a la arquitectura APH.

Gráfico 72: *POM del módulo Starter*

```
<parent>
    <groupId>pe.com.hiper.arch</groupId>
    <artifactId>hiper-arch-dependencies</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <relativePath>../hiper-arch-dependencies</relativePath>
</parent>
<artifactId>hiper-arch-starter</artifactId>
<packaging>pom</packaging>
<name>hiper-arch-starter</name>
<description>Hiper Arch Starter</description>

<dependencies>
    <dependency>..
    <dependency>..
    <dependency>..
    <dependency>..
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
        <exclusions>..
    </dependency>
    <dependency>..
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-validation</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>

<build>..
```

Fuente: Elaboración propia

4.2.5. Pruebas de la solución

Para dar inicio las pruebas es necesario realizar las primeras configuraciones en nuestro entorno de desarrollo, principalmente en el archivo de configuración del repositorio local de Maven. Se establece la dirección del repositorio GitHub donde se encuentra empaquetado nuestra arquitectura APH, así lo mismo las credenciales de acceso a la misma.

Gráfico 73: *Configuración del repositorio local Maven*

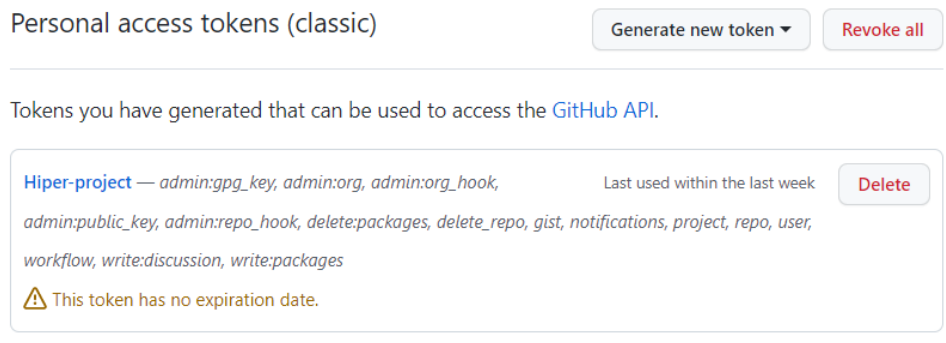
settings.xml

```
<pluginGroups>
</pluginGroups>

<activeProfiles>
  <activeProfile>github</activeProfile>
</activeProfiles>

<profiles>
  <profile>
    <id>github</id>
    <repositories>
      <repository>
        <id>central</id>
        <url>https://repo1.maven.org/maven2</url>
      </repository>
      <repository>
        <id>github</id>
        <url>https://maven.pkg.github.com/aldomar-mc/hiper-dsi-gdt-libraries-msa</url>
        <snapshots>
          <enabled>true</enabled>
        </snapshots>
      </repository>
    </repositories>
  </profile>
</profiles>

<servers>
  <server>
    <id>github</id>
    <username>aldomar-mc</username>
    <password>                                    </password>
  </server>
</servers>

</settings>
```

Fuente: Elaboración propia

La seguridad de acceso a la arquitectura APH nos proporciona la plataforma de GitHub a través de un token único.

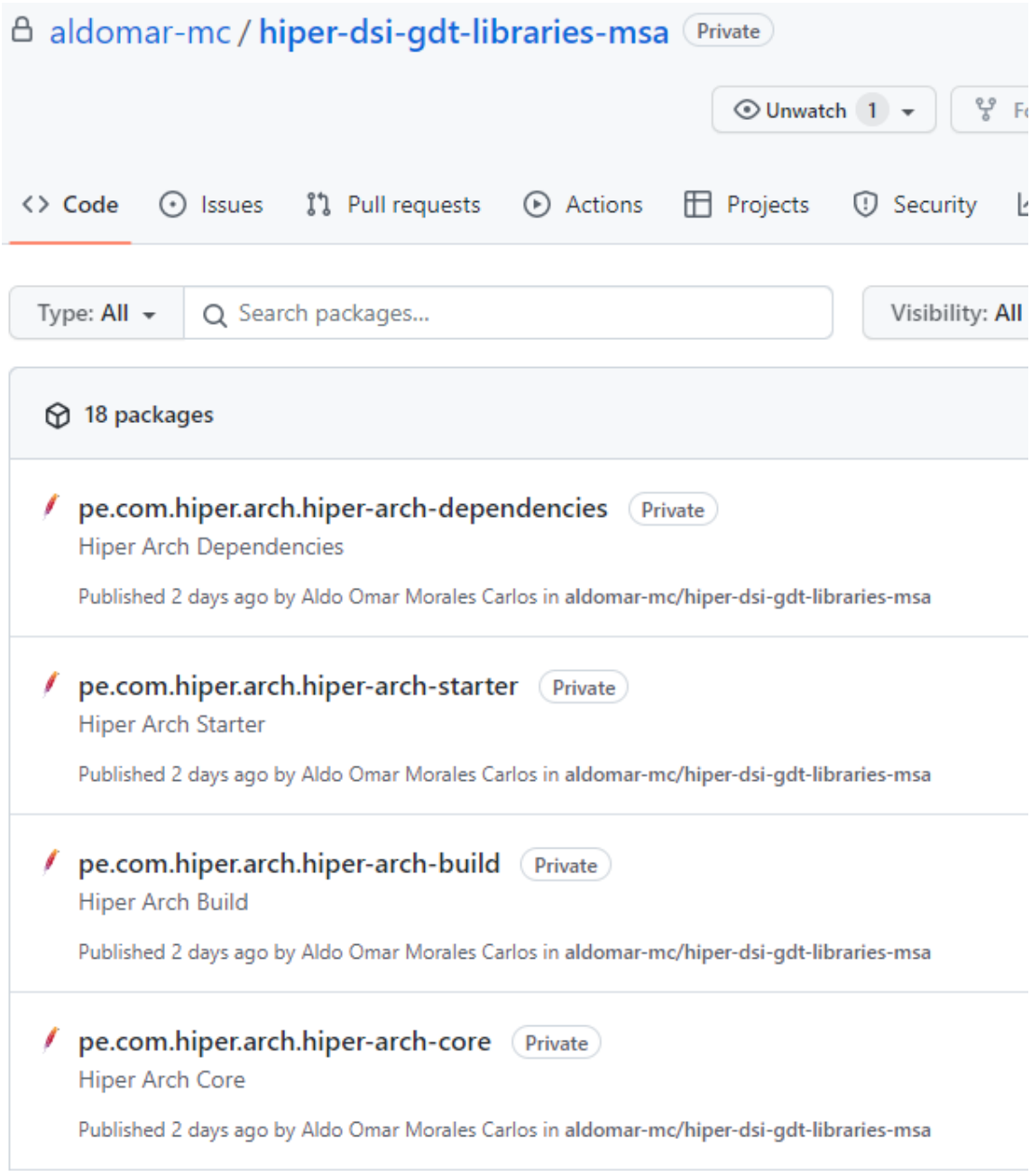
Gráfico 74: *Token de acceso a GitHub*



Fuente: Elaboración propia

Al tener configurado los accesos, el equipo de desarrollo podrá acceder a todos los componentes de la arquitectura que se encuentra ramificado para los entornos de desarrollo, preproducción y producción.

Gráfico 75: *Paquetes de la arquitectura APH en GitHub*

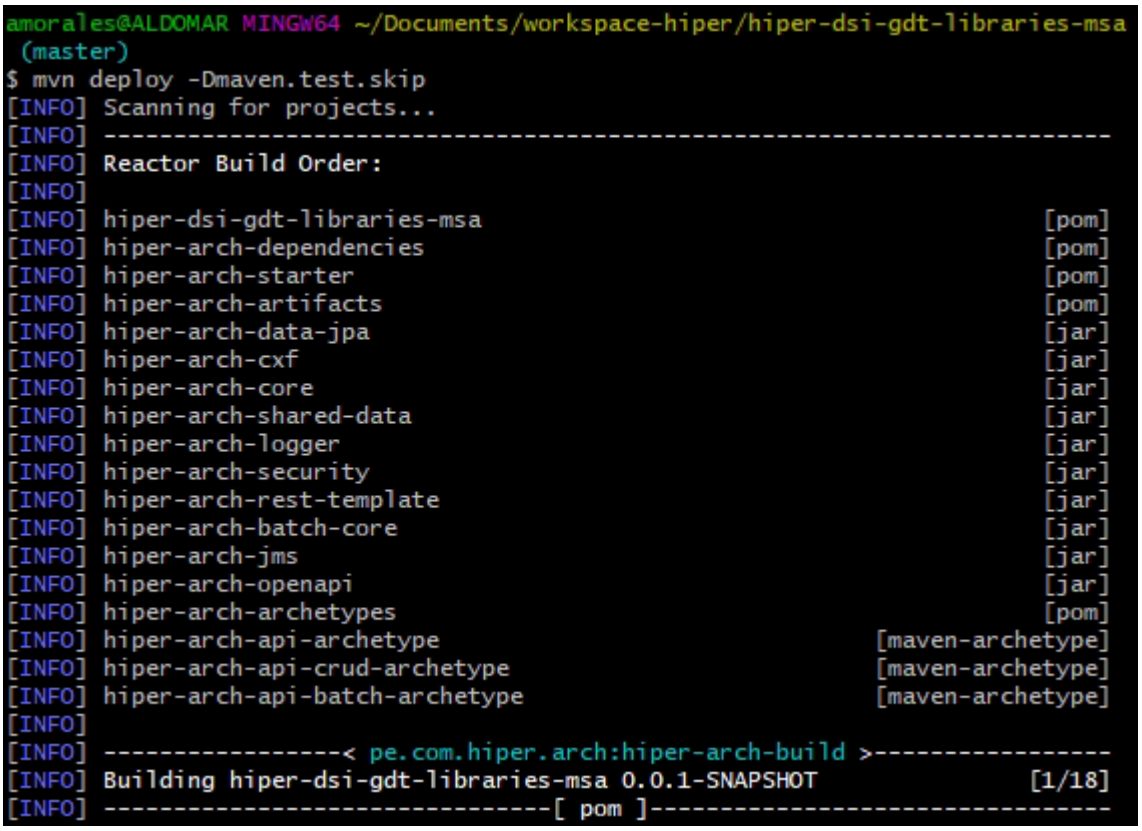


Fuente: Elaboración propia

Despliegue de los módulos de la arquitectura APH

Nos apoyamos de los comandos Maven para desplegar los módulos de la arquitectura.

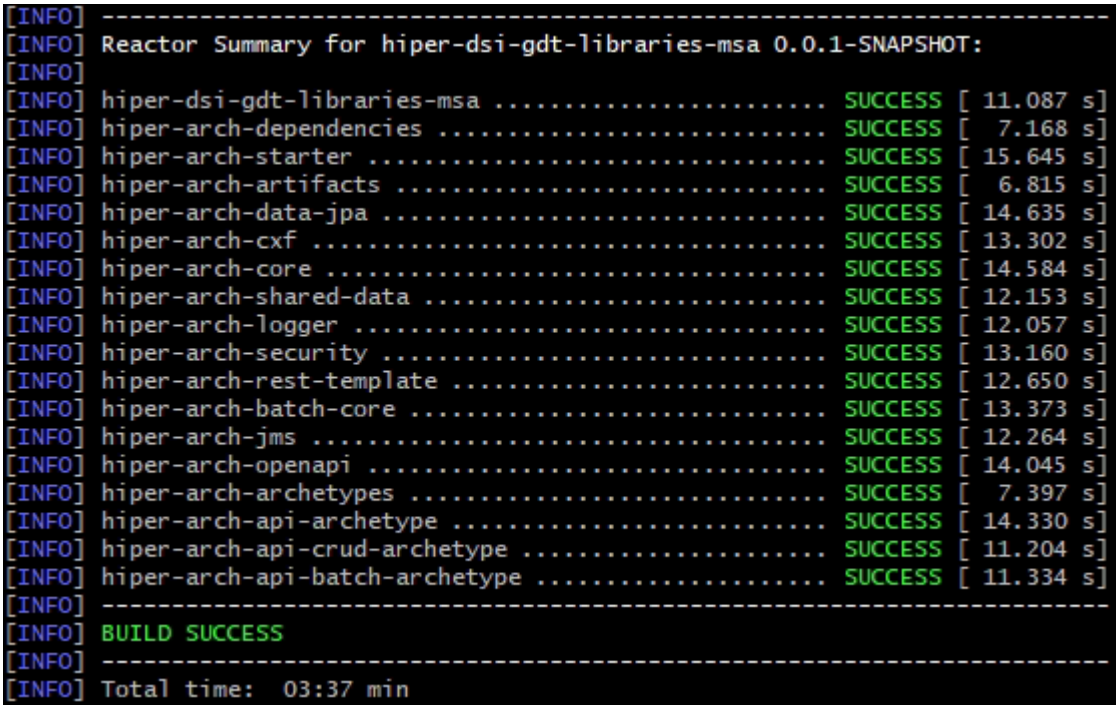
Gráfico 76: *Despliegue de los módulos de la arquitectura APH*



Fuente: Elaboración propia

Al ejecutar el comando Maven, cada uno de los microservicios que compone la arquitectura se ejecutan y se empaquetan en archivos Jar, y de manera automática son desplegados al repositorio GitHub.

Gráfico 77: *Despliegue de la arquitectura APH completado*



Fuente: Elaboración propia

Descarga de los módulos de la arquitectura APH

Mediante la ejecución del archivo CMD desde la interfaz de línea de comandos podemos descargar los módulos de la arquitectura APH disponibles en GitHub.

Gráfico 78: *Descarga de la arquitectura APH*

hiper-arch-download-archetypes.cmd

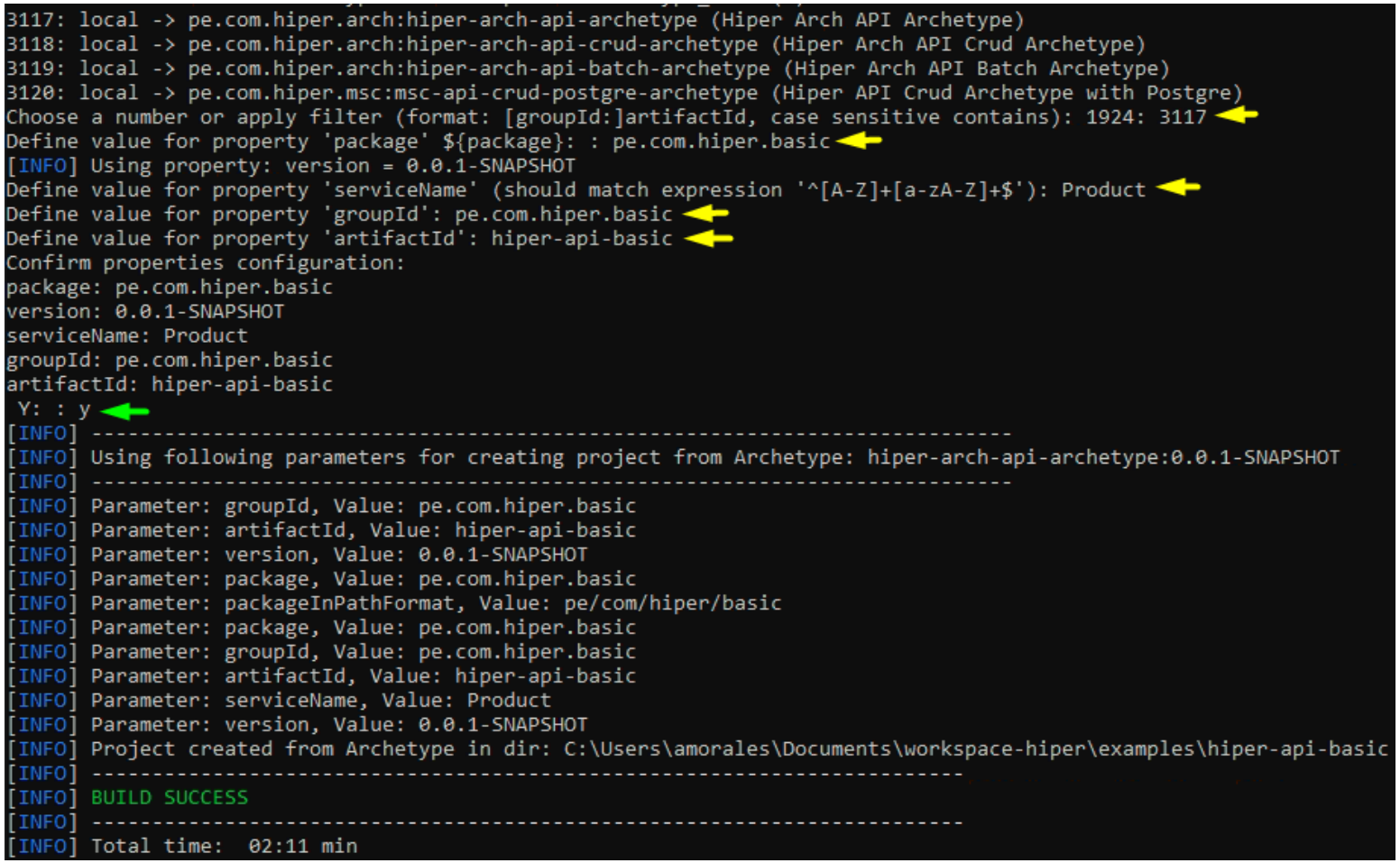
```
1  @echo off
2
3  call mvn dependency:tree -U
4
5  echo ------------------------------------------------------------
6  echo IMPORTANTE: Es necesario copiar los arquetipos descargados
7  echo Origen: \.m2\repository\pe\com\hiper\arch
8  echo Destino: \.m2\repository\org\apache\maven\archetype
9  echo ------------------------------------------------------------
```

Fuente: Elaboración propia

a) Generar API básico

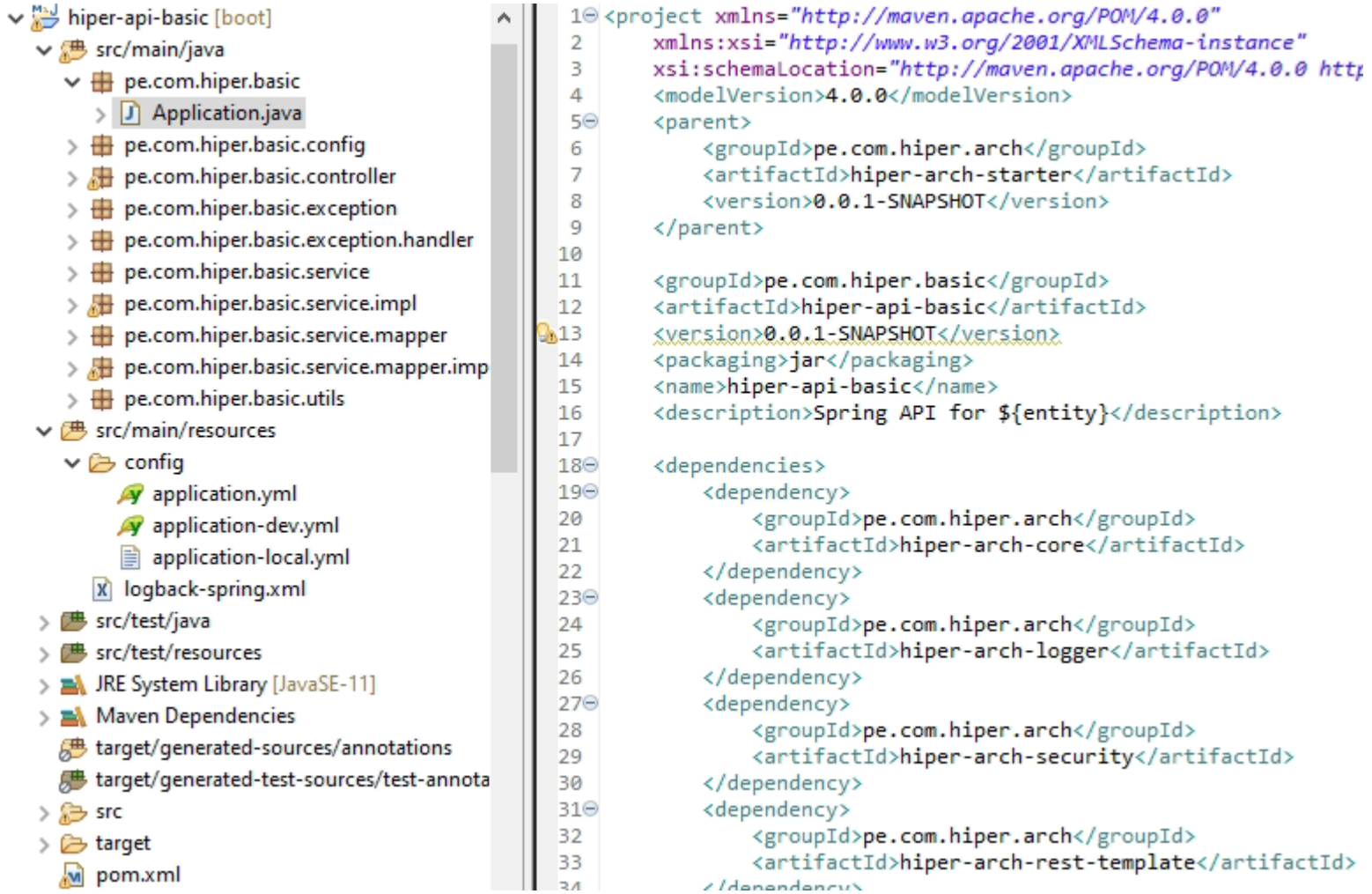
Para la generación de los diferentes arquetipos ejecutamos el siguiente comando mvn archetype:generate

Gráfico 79: *Generar API básico*



Fuente: Elaboración propia

Gráfico 80: *Estructura del microservicio básico*

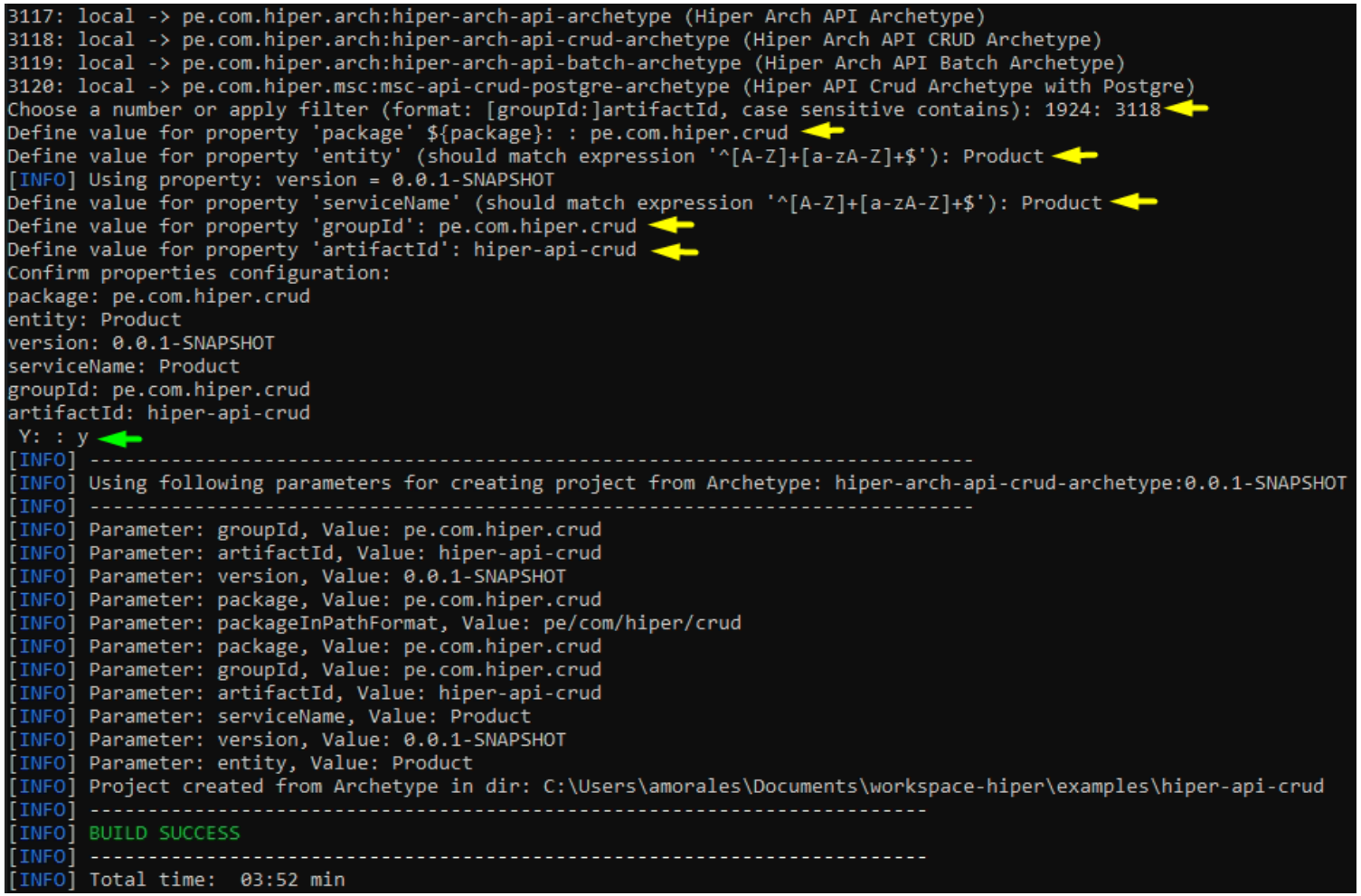


Fuente: Elaboración propia

b) Generar API CRUD

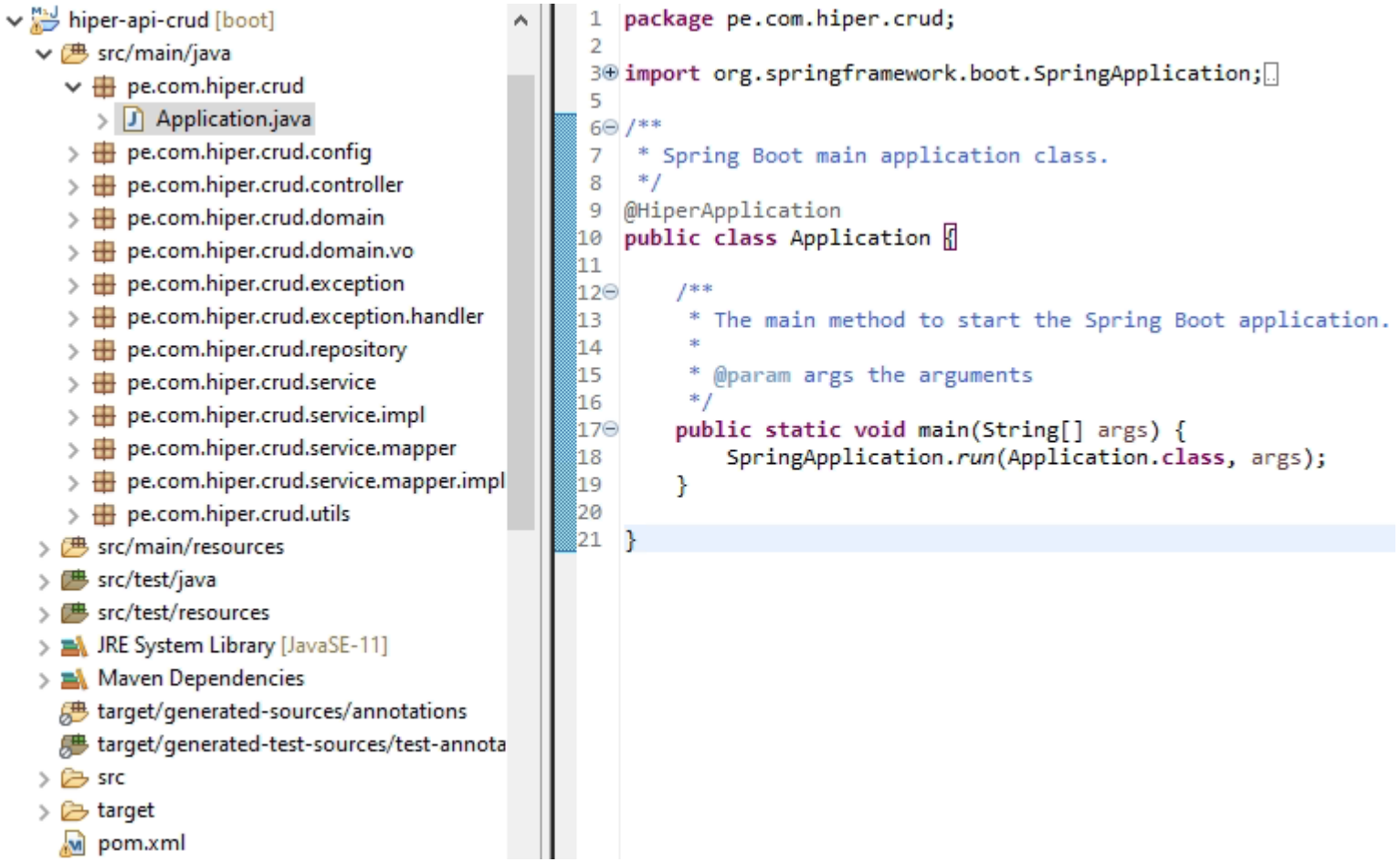
Se sigue el mismo procedimiento que la generación del API básico, donde ingresamos los parámetros necesarios.

Gráfico 81: *Generar API CRUD*



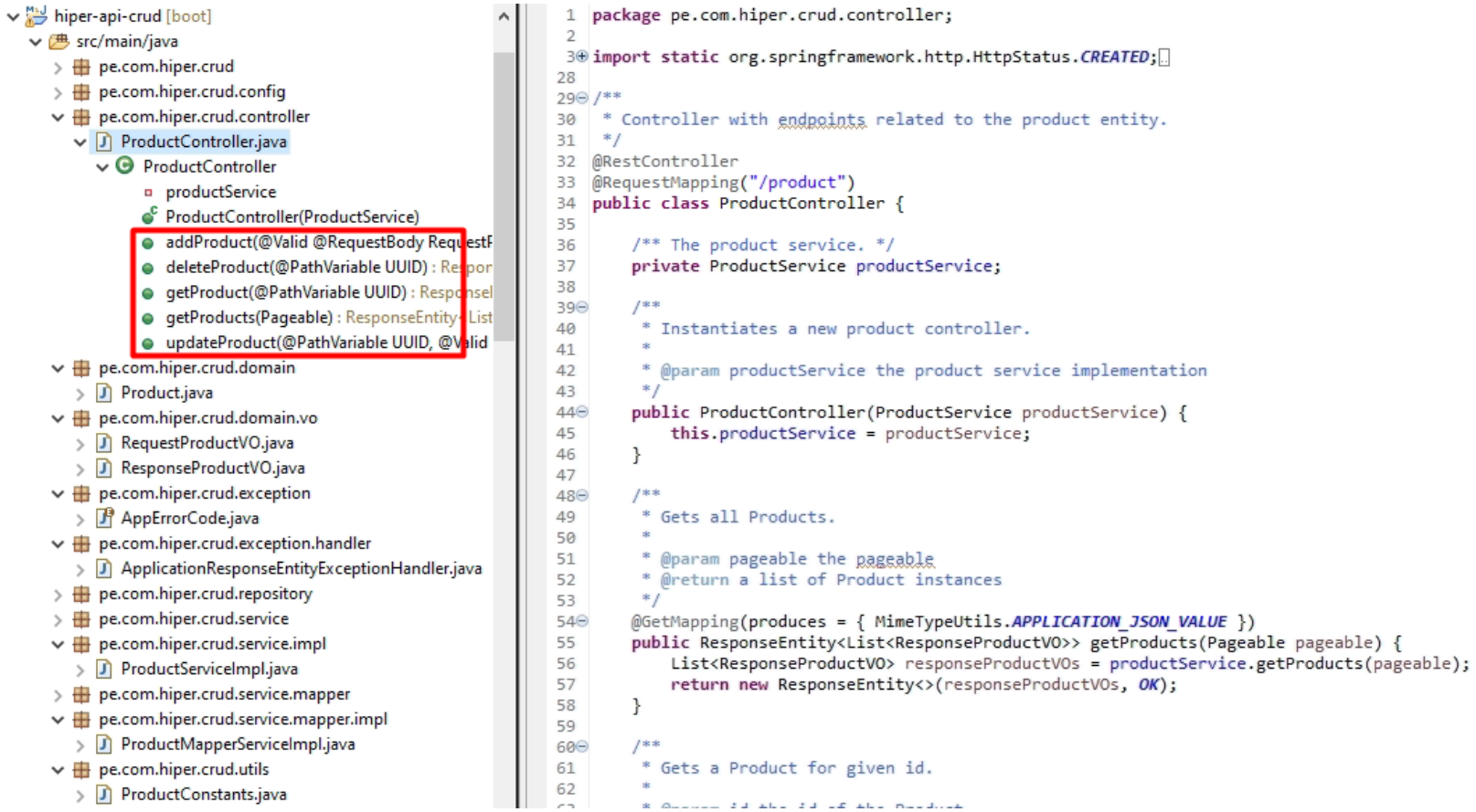
Fuente: Elaboración propia

Gráfico 82: *Estructura del microservicio CRUD*



Fuente: Elaboración propia

Gráfico 83: *Métodos del microservicio CRUD*



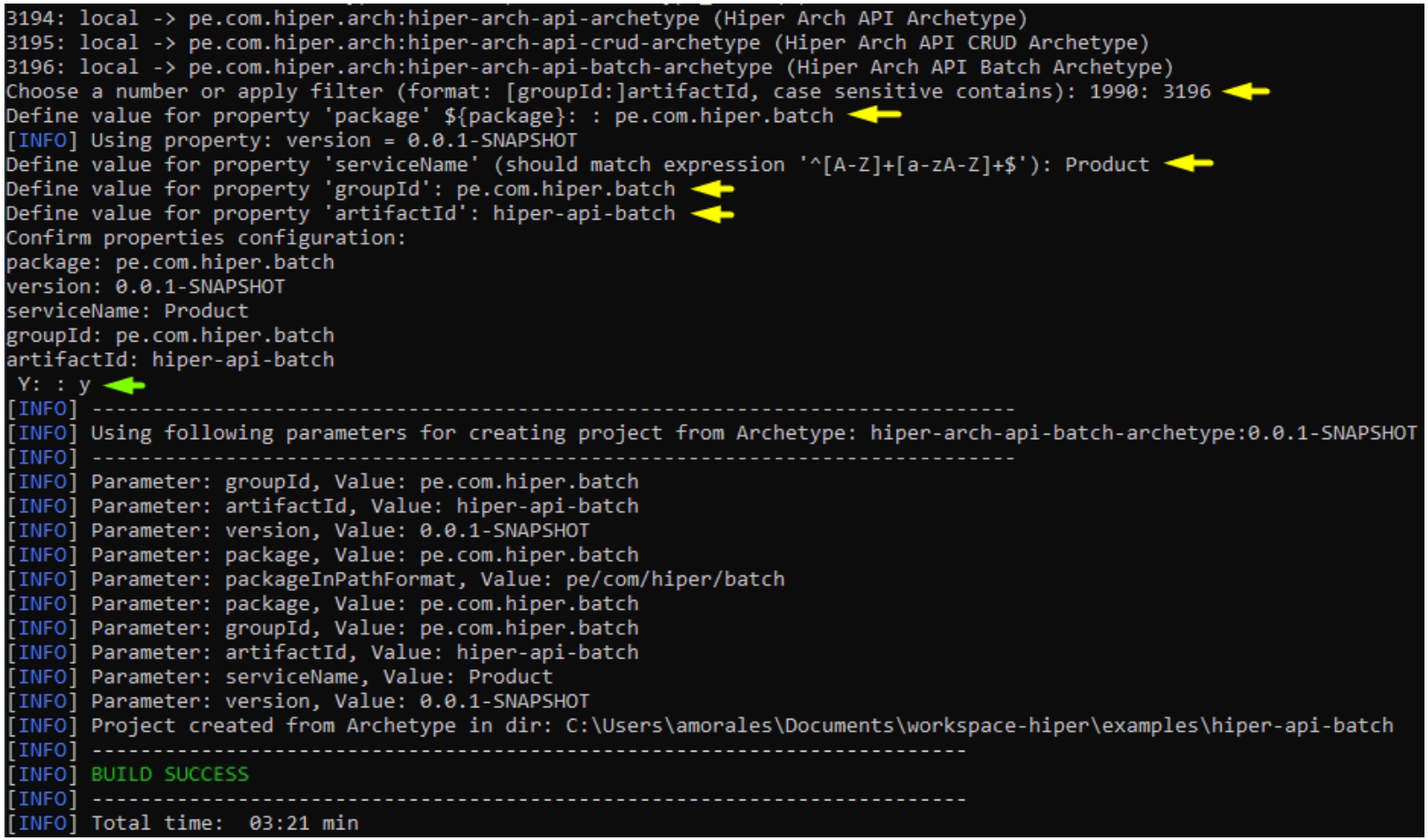

```java
package pe.com.hiper.crud.controller;

import static org.springframework.http.HttpStatus.CREATED;

/**
 * Controller with endpoints related to the product entity.
 */
@RestController
@RequestMapping("/product")
public class ProductController {

    /** The product service. */
    private ProductService productService;

    /**
     * Instantiates a new product controller.
     *
     * @param productService the product service implementation
     */
    public ProductController(ProductService productService) {
        this.productService = productService;
    }

    /**
     * Gets all Products.
     *
     * @param pageable the pageable
     * @return a list of Product instances
     */
    @GetMapping(produces = { MimeTypeUtils.APPLICATION_JSON_VALUE })
    public ResponseEntity<List<ResponseProductVO>> getProducts(Pageable pageable) {
        List<ResponseProductVO> responseProductVOs = productService.getProducts(pageable);
        return new ResponseEntity<>(responseProductVOs, OK);
    }

    /**
     * Gets a Product for given id.
     *
```

Fuente: Elaboración propia

c) Generar API Batch

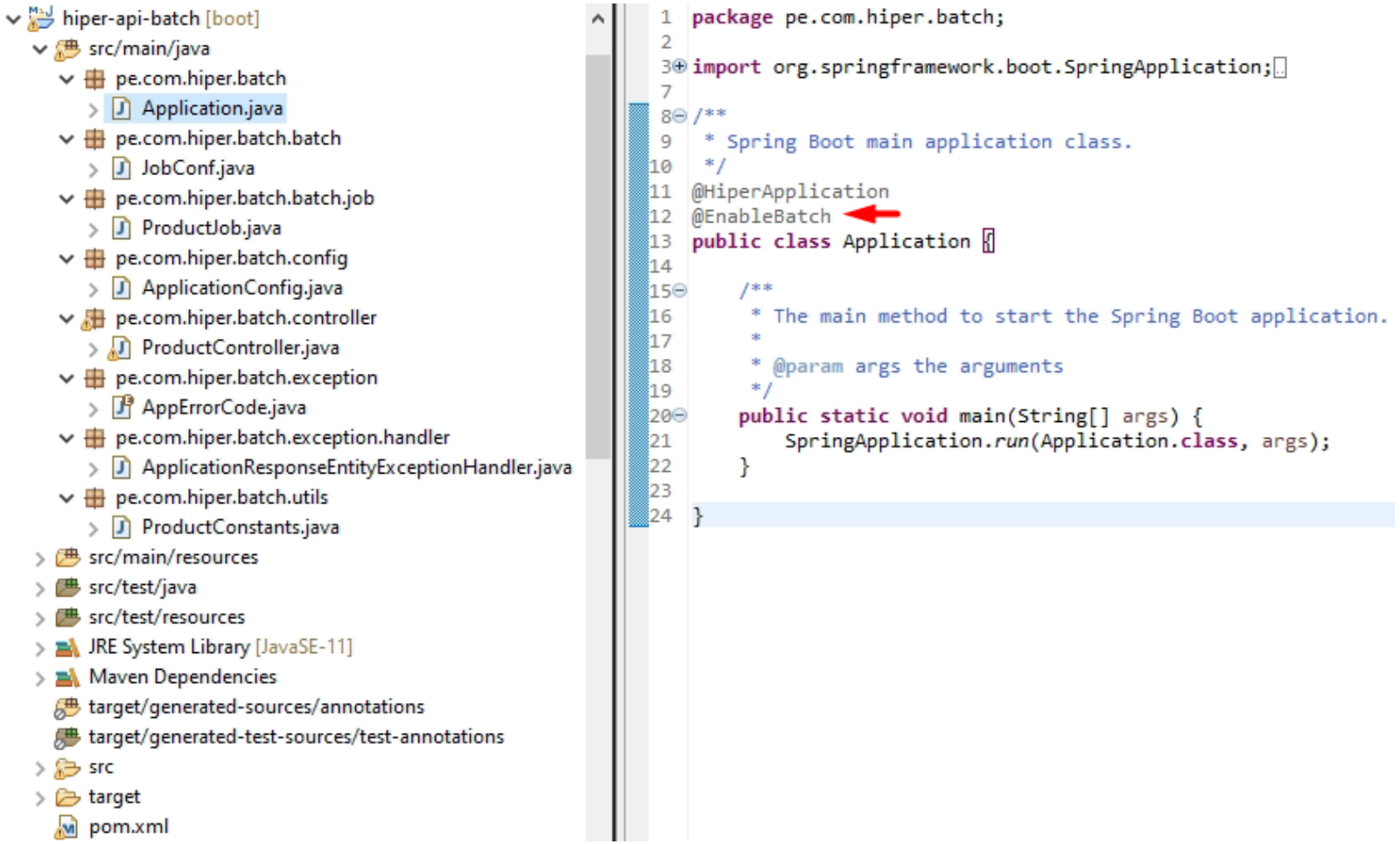
Ingresamos el identificador del arquetipo Batch para generar el microservicio con sus respectivas dependencias.

Gráfico 84: *Generar API Batch*



Fuente: Elaboración propia

Gráfico 85: *Estructura del microservicio batch*



Fuente: Elaboración propia

4.2.6. Monitoreo y evaluación de la solución

Se realizó el monitoreo y la respectiva evaluación para la conformidad de los usuarios.

Elementos de monitoreo y evaluación

A continuación, se describe los elementos para el monitoreo y evaluación de la arquitectura APH:

- Recolección de datos de las fuentes determinadas y el registro en los instrumentos respectivos.
- Verificación del cumplimiento de los objetivos planteados.
- Decisión respecto a las acciones correctivas o de retroalimentación necesarias de acuerdo con la información obtenida.
- Validación de la implementación.

Los elementos planteados son imprescindibles para el monitoreo de la solución, es así, que nos permitirá comprobar que la solución cumpla con los objetivos, si fuere lo contrario se ejecuta acciones correctivas y que estas sean validadas en base a las pruebas de control de calidad.

Políticas y reglas de procedimiento

A continuación, se describe las políticas y reglas de procedimiento:

- Las acciones correctivas realizadas deben pasar el control de calidad de acuerdo con los objetivos.
- El acceso a los microservicios basados a la arquitectura APH, se encuentra restringido por perfiles de cada cliente de acuerdo con el contrato de compartimiento de fuente de datos.

- El despliegue de los cambios en el entorno de producción se debe realizar siempre y cuando cumpla con los objetivos y debidamente aprobadas por los involucrados.
- Para el uso de la arquitectura APH, la solicitud de las credenciales de acceso mediante token debe ser solicitada formalmente al departamento de arquitectura.
- La arquitectura APH se encuentra empaquetada y disponible en el repositorio de GitHub pudiendo acceder a ella desde cualquier lugar y tiempo.
- El departamento de desarrollo está en la obligación de crear los nuevos microservicios en base a la arquitectura desarrollada.

Plan de monitoreo y evaluación

En el presente plan se tiene como objetivo la implementación de la solución, a continuación, se detalla:

Tabla 5: *Plan de monitoreo y evaluación*

Monitoreo	Evaluación
Verificación del repositorio (GitHub) para la implementación	Cumple con todos los niveles de seguridad y requisitos para la implementación
Verificación del despliegue en el entorno de producción	El proceso de despliegue cuenta con la documentación y apuntes de guía
Verificación de los módulos de la arquitectura APH	Los microservicios modularizados tienen la funcionalidad con 0 errores
Verificación de la interoperabilidad con servicios de terceros	Nivel de acuerdo en respuesta y soporta la comunicación de tipo REST y Web Service
Equipo de TI con conocimientos en el desarrollo de microservicios	Cantidad que sobrepasa el 50% con conocimientos en el desarrollo de microservicio

Pruebas piloto en 3 equipos del departamento de desarrollo y 1 en arquitectura	Funcionamiento correcto de los arquetipos de API básico, API CRUD y API Batch
Capacitación a los departamentos involucrados de la funcionalidad de la arquitectura	Cantidad mínima con desconocimiento respecto al desarrollo e implementación de microservicios
Comprobación de objetivos en la arquitectura con la implementación de microservicios con funcionalidades de casos reales	La arquitectura satisface las funcionalidades planificadas y cumple con los objetivos

Fuente: Elaboración propia

La tabla descrita se realizó el monitoreo con indicadores, la cuales tienen su evaluación respectiva, dichas evaluaciones se reflejan en el plan de monitoreo y se aplica con el fin de mejorar su ejecución y eficacia.

4.2.7. Bitácora y puesta a punto

Bitácora

Las etapas del proyecto con sus respectivas actividades y observaciones se detallan en el siguiente cuadro de bitácora.

Tabla 6: *Bitácora del proyecto*

Fecha	Etapa	Actividad	Observación
05/07/2022	Evaluación preliminar	Acta de constitución del proyecto de software	Se contó con el apoyo del jefe de proyecto de la organización
		Reunión con el jefe del proyecto y lideres técnicos	Se contó de la colaboración del jefe y lideres del proyecto
		Recopilación de información	Se recopiló información con formatos establecidos

		Modelado de diagramas de la situación actual	Se elaboró un modelo en representación de la situación actual
18/07/2022	Análisis	Análisis de los procesos identificados	Se facilitó la información de los procesos
		Definición de requerimientos del negocio	Se analizó y se describió los requerimientos
01/08/2022	Diseño	Diseño del prototipo de solución	En base al análisis se diseñó el prototipo
		Modelar los componentes de la arquitectura	Se construyó un modelo de los componentes de la arquitectura
15/08/2022	Desarrollo	Desarrollo de la arquitectura APH	En base al diseño de solución se desarrolló la arquitectura APH
16/11/2022	Pruebas	Pruebas funcionales del desarrollador	Se realizó las pruebas necesarias y se levantó las observaciones
		Pruebas funcionales por el departamento de calidad	Se revalidó las funcionalidades con el apoyo del departamento de calidad
19/11/2022	Implementación	Capacitación a los equipos de los departamentos involucrados	Se brindó las capacitaciones a los desarrolladores interesados de los departamentos
		Arquitectura puesta en producción	Se desplegó la arquitectura al entorno de producción sin errores

Fuente: Elaboración propia

Aprobación de la solución tecnológica

Implementado la arquitectura APH luego de las pruebas integrales se desplegó en el entorno de producción del repositorio GitHub para su operatividad en los nuevos microservicios que se irán creando por los equipos de desarrollo.

4.2.8. Presentación de análisis descriptivo

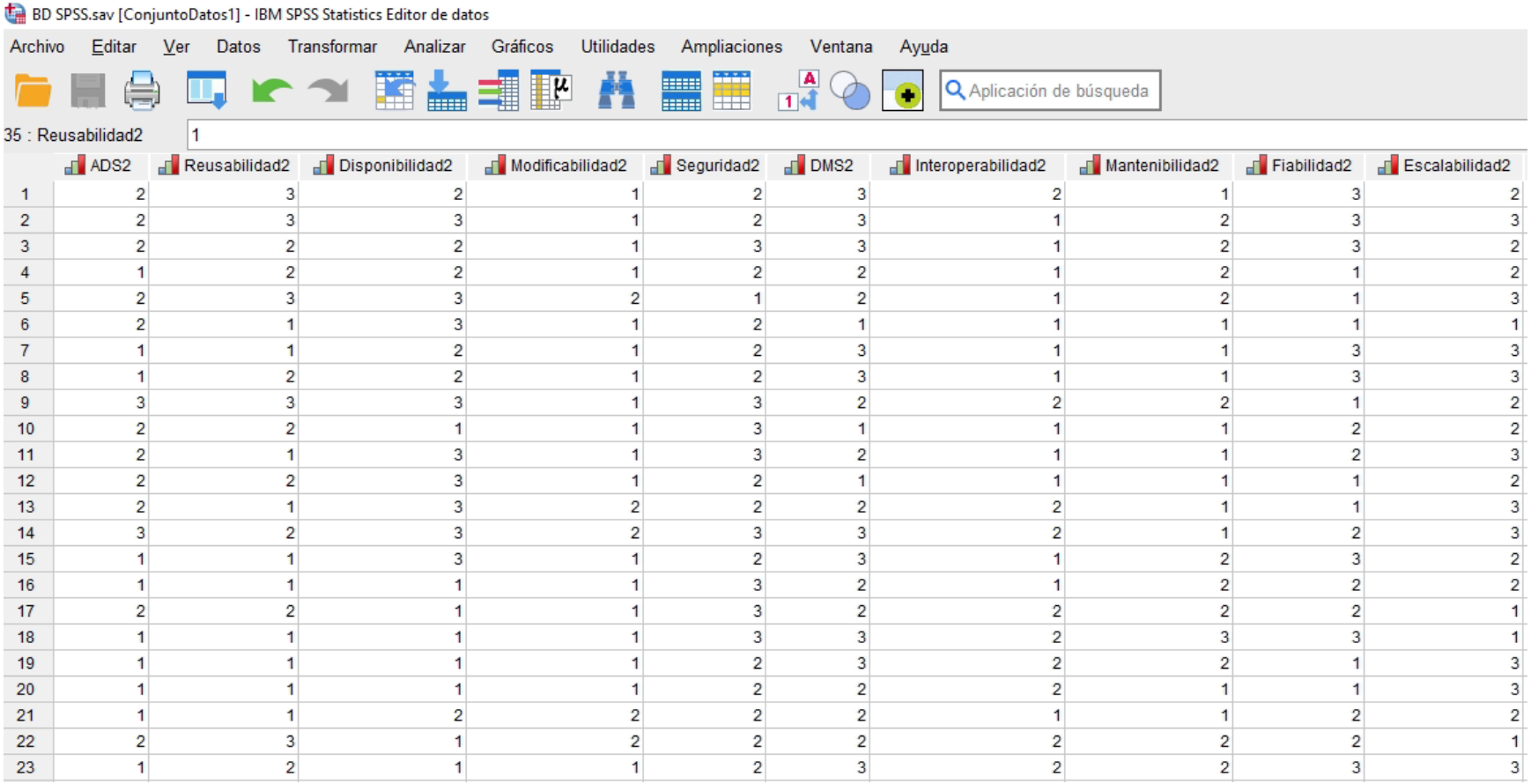
Se proporciona resultados del análisis descriptivo de la investigación, para facilitar una comparación entre la prueba previa y posterior. Para evaluar los indicadores del **desarrollo de microservicios** de acuerdo con la encuesta aplicada (**Anexo 05 y 06**) en el cuadro N° 29 se muestra la ponderación planteada.

Tabla 7: *Ponderación de niveles de calificación de los indicadores*

Indicadores	Ponderación (Máx.)	Intervalo	Nivel	Valor
Interoperabilidad	15	1 – 5	Deficiente	1
Fiabilidad		6 – 10	Regular	2
Escalabilidad		11 – 15	Eficiente	3
Mantenibilidad	10	1 – 3	Deficiente	1
		4 – 7	Regular	2
		8 – 10	Eficiente	3

Fuente: Elaboración propia

Gráfico 86: *Matriz de datos del nivel de calificación de los indicadores*


BD SPSS.sav [ConjuntoDatos1] - IBM SPSS Statistics Editor de datos

Archivo Editar Ver Datos Transformar Analizar Gráficos Utilidades Ampliaciones Ventana Ayuda

Aplicación de búsqueda

35 : Reusabilidad2 1

	ADS2	Reusabilidad2	Disponibilidad2	Modificabilidad2	Seguridad2	DMS2	Interoperabilidad2	Mantenibilidad2	Fiabilidad2	Escalabilidad2
1	2	3	2	1	2	3	2	1	3	2
2	2	3	3	1	2	3	1	2	3	3
3	2	2	2	1	3	3	1	2	3	2
4	1	2	2	1	2	2	1	2	1	2
5	2	3	3	2	1	2	1	2	1	3
6	2	1	3	1	2	1	1	1	1	1
7	1	1	2	1	2	3	1	1	3	3
8	1	2	2	1	2	3	1	1	3	3
9	3	3	3	1	3	2	2	2	1	2
10	2	2	1	1	3	1	1	1	2	2
11	2	1	3	1	3	2	1	1	2	3
12	2	2	3	1	2	1	1	1	1	2
13	2	1	3	2	2	2	2	1	1	3
14	3	2	3	2	3	3	2	1	2	3
15	1	1	3	1	2	3	1	2	3	2
16	1	1	1	1	3	2	1	2	2	2
17	2	2	1	1	3	2	2	2	2	1
18	1	1	1	1	3	3	2	3	3	1
19	1	1	1	1	2	3	2	2	1	3
20	1	1	1	1	2	2	2	1	1	3
21	1	1	2	2	2	2	1	1	2	2
22	2	3	1	2	2	2	2	2	2	1
23	1	2	1	1	2	3	2	2	3	3

Fuente: Elaboración propia (Base de datos SPSS)

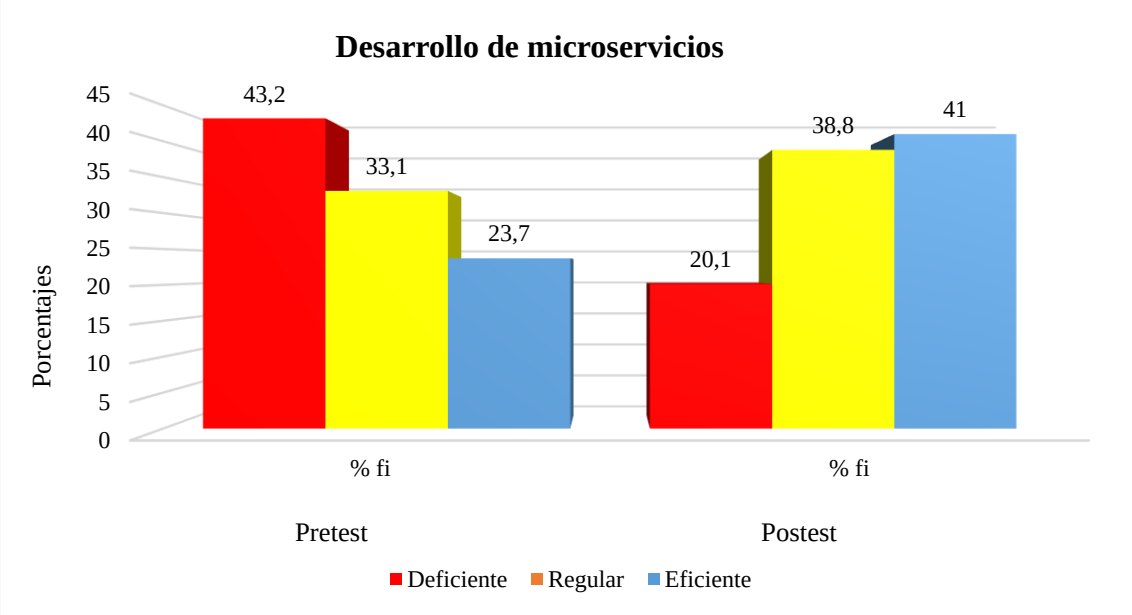
Los resultados descriptivos se proporcionan en forma de tablas y gráficos.

Tabla 8: *Niveles y frecuencias del desarrollo de microservicios*

Niveles	Pretest Desarrollo de microservicios		Postest Desarrollo de microservicios	
	f_i	%	f_i	%
Deficiente	60	43,2	28	20,1
Regular	46	33,1	54	38,8
Eficiente	33	23,7	57	41,0
Total	139	100,0	139	100,0

Fuente: Elaboración propia

Gráfico 87: *Nivel del desarrollo de microservicios*



Fuente: Elaboración propia

Conforme con los resultados de la tabla N° 6 y Gráfico 90, se aprecian los resultados de la variable desarrollo de microservicios. Para lo cual se llevó a cabo una encuesta a los empleados de la organización Hiper. En se sentido los resultados para el pretest muestran que el desarrollo de microservicios para 60 empleados se encontró en un nivel deficiente con 43,2%, para 46 empleados en

nivel regular con 33,1% y para 33 empleados en nivel eficiente con 23,7%. En ese sentido, el nivel preponderante en el pretest fue el deficiente.

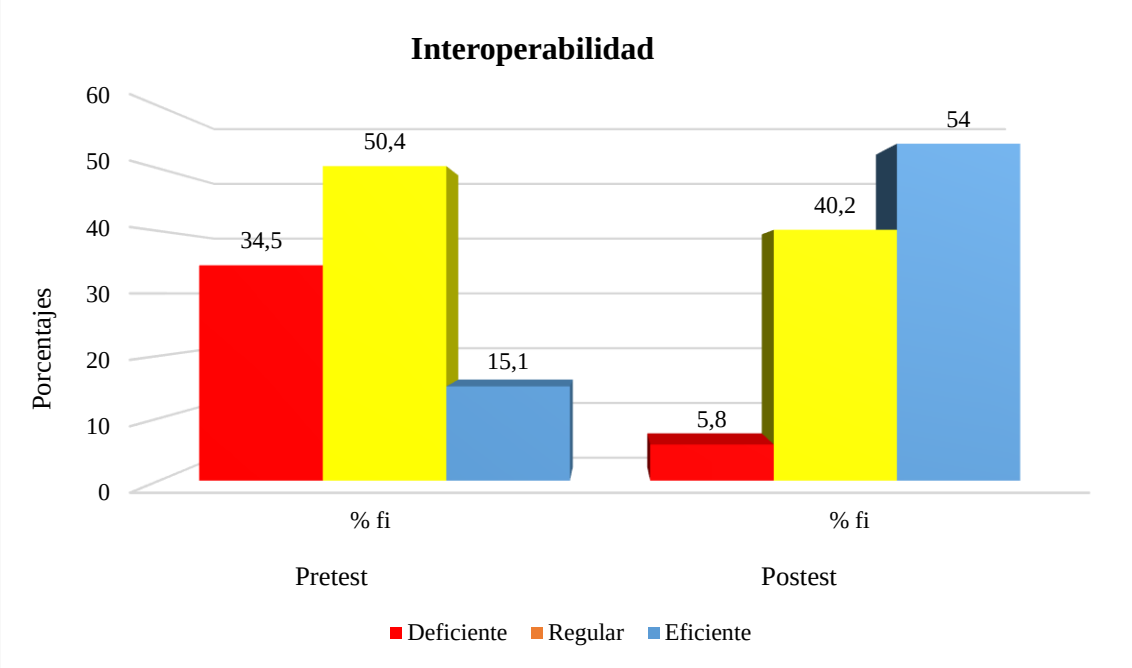
Por otro lado, en el postest los resultados mostraron que, el desarrollo de microservicios para 28 empleados se encontró en un nivel deficiente con 20,1%, para 54 empleados en nivel regular con 38,8% y para 57 empleados en nivel eficiente con 41%. En ese sentido, el nivel preponderante en el postest fue el eficiente. Lo cual demuestra la efectividad de aplicar la arquitectura de software APH.

Tabla 9: *Niveles y frecuencias de la interoperabilidad*

Niveles	Pretest Interoperabilidad		Postest Interoperabilidad	
	f_i	%	f_i	%
Deficiente	48	34,5	8	5,8
Regular	70	50,4	56	40,2
Eficiente	21	15,1	75	54,0
Total	139	100,0	139	100,0

Fuente: Elaboración propia

Gráfico 88: *Nivel de la interoperabilidad*



Fuente: Elaboración propia

Conforme con los resultados de la tabla N° 7 y gráfico 91, se aprecian los resultados de la dimensión interoperabilidad. Para lo cual se llevó a cabo una encuesta a los empleados de la organización Hiper. En se sentido, para la dimensión interoperabilidad los resultados para el pretest muestran que para 48 empleados se encontró en un nivel deficiente con 34,5%, para 70 empleados en nivel regular con 50,4% y para 21 empleados en nivel eficiente con 15,1%. En ese sentido, el nivel preponderante en el pretest fue el regular.

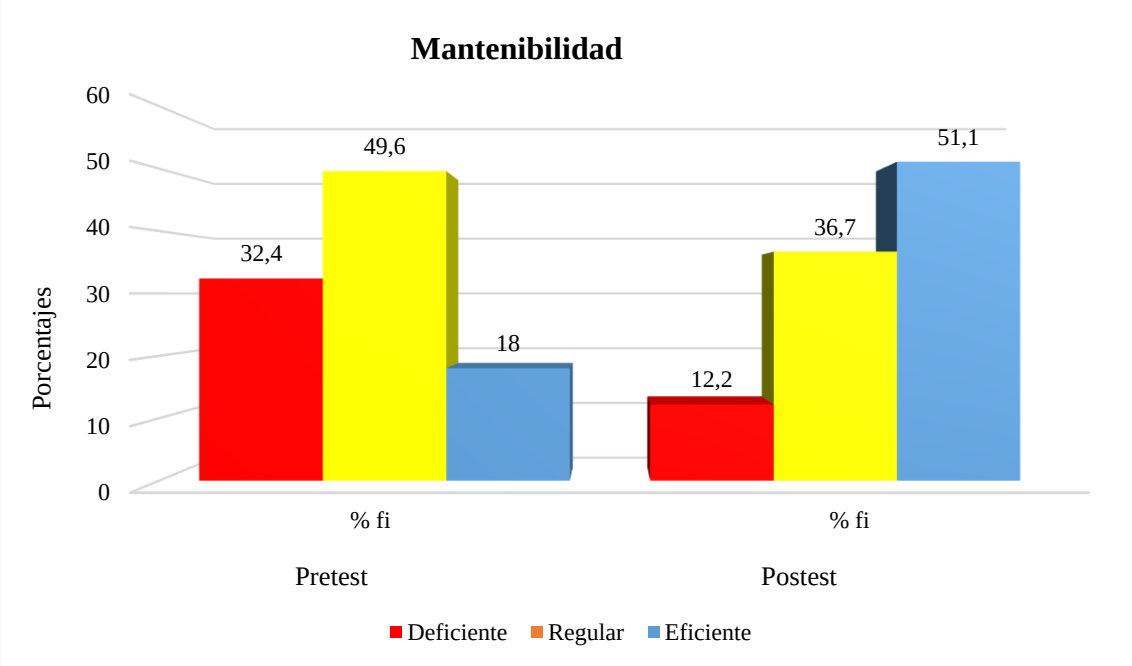
Por otro lado, en el postest para la dimensión interoperabilidad los resultados mostraron que para 8 empleados se encontró en un nivel deficiente con 5,8%, para 56 empleados en nivel regular con 40,2% y para 75 empleados en nivel eficiente con 54%. En ese sentido, el nivel preponderante en el postest fue el eficiente. Por lo cual demuestra la efectividad de aplicar la Arquitectura de software APH.

Tabla 10: *Niveles y frecuencias de la mantenibilidad*

Niveles	Pretest Mantenibilidad		Postest Mantenibilidad	
	f_i	%	f_i	%
Deficiente	45	32,4	17	12,2
Regular	69	49,6	51	36,7
Eficiente	25	18,0	71	51,1
Total	139	100,0	139	100,0

Fuente: Elaboración propia

Gráfico 89: *Nivel de la mantenibilidad*



Fuente: Elaboración propia

Conforme con los resultados de la tabla N° 8 y gráfico 92, se aprecian los resultados de la dimensión mantenibilidad. Para lo cual se llevó a cabo una encuesta a los empleados de la organización Hiper. En se sentido, para la dimensión mantenibilidad los resultados para el pretest muestran que para 45

empleados se encontró en un nivel deficiente con 32,4%, para 69 empleados en nivel regular con 49,46% y para 25 empleados en nivel eficiente con 18%. En ese sentido, el nivel preponderante en el pretest fue el regular.

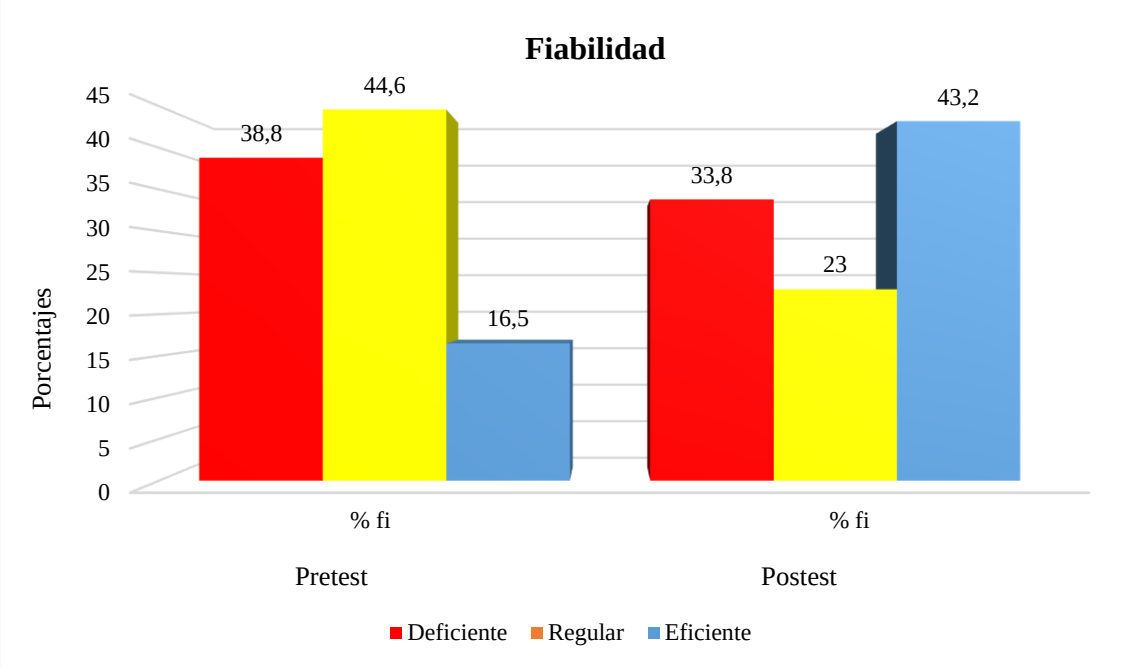
Por otro lado, en el postest para la dimensión mantenibilidad los resultados mostraron que para 17 empleados se encontró en un nivel deficiente con 12,2%, para 51 empleados en nivel regular con 36,7% y para 71 empleados en nivel eficiente con 51,1%. En ese sentido, el nivel preponderante en el postest fue el eficiente. Por lo cual demuestra la efectividad de aplicar la arquitectura de software APH.

Tabla 11: *Niveles y frecuencias de la fiabilidad*

Niveles	Pretest Fiabilidad		Postest Fiabilidad	
	f_i	%	f_i	%
Deficiente	54	38,8	47	33,8
Regular	62	44,6	32	23,0
Eficiente	23	16,5	60	43,2
Total	139	100,0	139	100,0

Fuente: Elaboración propia

Gráfico 90: *Nivel de la fiabilidad*



Fuente: Elaboración propia

Conforme con los resultados de la tabla N° 9 y gráfico 93, se aprecian los resultados de la dimensión fiabilidad. Para lo cual se llevó a cabo una encuesta a los empleados de la organización Hiper. En se sentido, para la dimensión fiabilidad los resultados para el pretest muestran que para 54 empleados se encontró en un nivel deficiente con 38,8%, para 62 empleados en nivel regular con 44,6% y para 23 empleados en nivel eficiente con 16,5%. En ese sentido, el nivel preponderante en el pretest fue el regular.

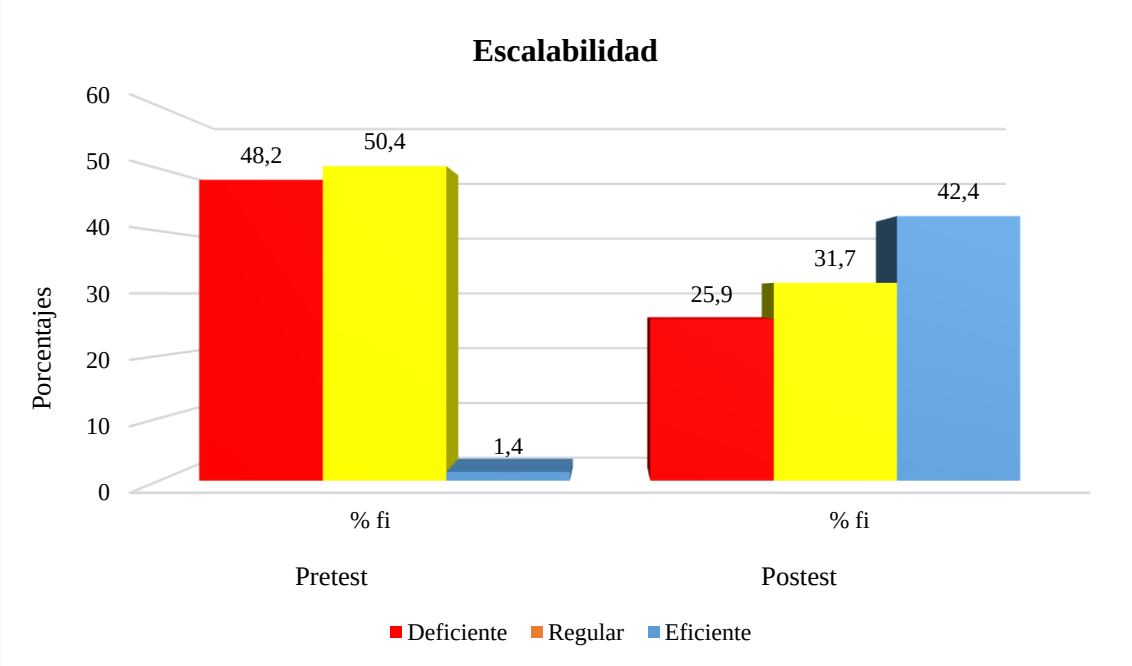
Por otro lado, en el postest para la dimensión fiabilidad los resultados mostraron que para 47 empleados se encontró en un nivel deficiente con 33,8%, para 32 empleados en nivel regular con 23% y para 60 empleados en nivel eficiente con 43,2%. En ese sentido, el nivel preponderante en el postest fue el eficiente. Por lo cual demuestra la efectividad de aplicar la arquitectura de software APH.

Tabla 12: *Niveles y frecuencias de la escalabilidad*

Niveles	Pretest Escalabilidad		Postest Escalabilidad	
	f_i	%	f_i	%
Deficiente	67	48,2	36	25,9
Regular	70	50,4	44	31,7
Eficiente	2	1,4	59	42,4
Total	139	100,0	139	100,0

Fuente: Elaboración propia

Gráfico 91: *Nivel de la escalabilidad*



Fuente: Elaboración propia

Conforme con los resultados de la tabla N° 10 y gráfico 94, se aprecian los resultados de la dimensión escalabilidad. Para lo cual se llevó a cabo una encuesta a los empleados de la organización Hiper. En se sentido, para la dimensión escalabilidad los resultados para el pretest muestran que para 57 empleados se

encontró en un nivel deficiente con 48,2%, para 70 empleados en nivel regular con 50,4% y para 2 empleados en nivel eficiente con 1,4%. En ese sentido, el nivel preponderante en el pretest fue el regular.

Por otro lado, en el postest para la dimensión escalabilidad los resultados mostraron que para 36 empleados se encontró en un nivel deficiente con 25,9%, para 44 empleados en nivel regular con 31,7% y para 59 empleados en nivel eficiente con 42,4%. En ese sentido, el nivel preponderante en el postest fue el eficiente. Por lo cual demuestra la efectividad de aplicar la Arquitectura de software APH.

Análisis descriptivo de la arquitectura de software APH

Los resultados de la encuesta se muestran en el **Anexo 07**, donde se realizó a partir de las pruebas previas. Es importante resaltar que no se cuenta con información pretest debido a que la solución tecnológica no se contaba anteriormente.

Tabla 13: *Resultados de aceptación de la arquitectura APH*

	Arquitectura de software APH			
Nivel\Dimensión	**Reusabilidad**	**Disponibilidad**	**Modificabilidad**	**Seguridad**
Totalmente desacuerdo	0%	0%	0%	0%
En desacuerdo	0%	0%	0%	0%
Neutral	0%	0%	0%	3%
De acuerdo	42%	33%	54%	53%
Totalmente acuerdo	58%	67%	46%	44%

Fuente: Elaboración propia

Conforme a los resultados de la tabla N° 11, se aprecia los resultados sobre la arquitectura de software APH y sus respectivas dimensiones. En ese sentido se

observa para las dimensiones de reusabilidad, disponibilidad modificabilidad y seguridad, tienen una aceptación de un nivel “De acuerdo” y “Totalmente acuerdo”. Por lo cual se demuestra que la arquitectura de software APH cumple con los objetivos planificados.

4.2.9. Presentación de análisis inferencial

El análisis inferencial se llevó a cabo después de la finalización del análisis descriptivo. Dado que había más de 50 componentes en la muestra, se sometió a la prueba de Kolmogórov-Smirnov. De manera similar, se usó un umbral de certeza del 95 %.

Regla de decisión:

Si:

Sig. < 0.05 adopta una distribución no normal.

Sig. ≥ 0.05 adopta una distribución normal.

Dónde:

Sig.: P-valor o nivel crítico del contraste.

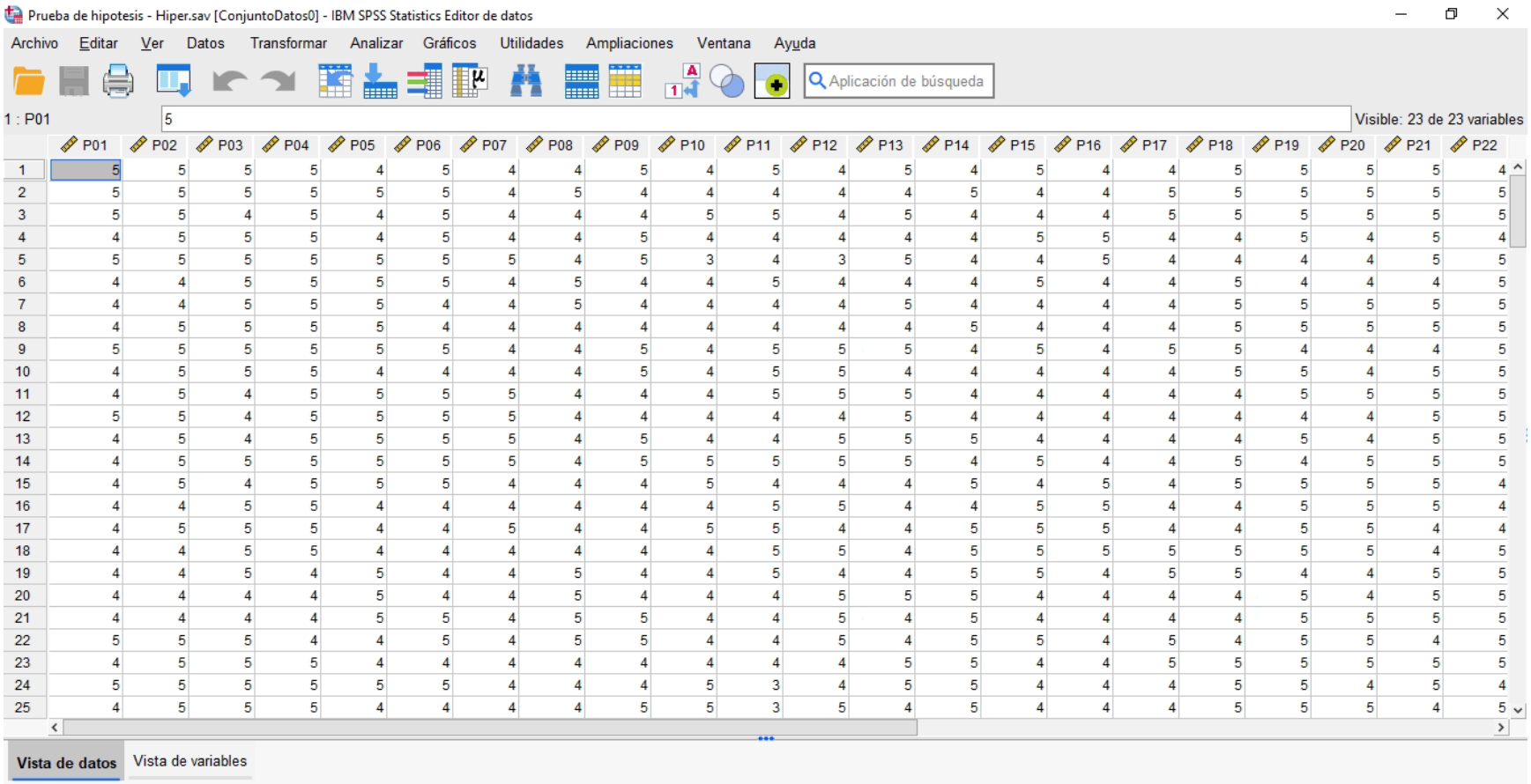
Se llevó a cabo la prueba de normalidad para la variable desarrollo de microservicios.

H_0: Los datos de la variable desarrollo de microservicios siguen una distribución normal.

H_a: Los datos de la variable desarrollo de microservicios no siguen una distribución normal.

En el siguiente gráfico se muestra la matriz de datos trabajado en el programa SPSS.

Gráfico 92: *Matriz de datos*



Fuente: Elaboración propia

Tabla 14: *Prueba de normalidad de la variable desarrollo de microservicios*

	Kolmogórov-Smirnov		
	Estadístico	gl	Sig.
Desarrollo de microservicios _Pretest	,276	139	,000
Desarrollo de microservicios _Postest	,250	139	,000

Fuente: Elaboración propia (Corrección de significación de Lilliefors)

Conforme con la tabla N° 11, se aprecia que los resultados del pretest y postest de la variable desarrollo de microservicios obtuvieron cantidades menores a 0.05. Por lo cual, se rechaza la hipótesis nula y se acepta la hipótesis alterna y se concluye que la variable no se distribuye normalmente.

Asimismo, se realizó la prueba de normalidad para la dimensión **interoperabilidad**.

H_0: Los datos de la dimensión interoperabilidad sigue una distribución normal.

H_a: Los datos de la dimensión interoperabilidad no sigue una distribución normal.

Tabla 15: *Prueba de normalidad de la dimensión interoperabilidad*

	Kolmogórov-Smirnov		
	Estadístico	gl	Sig.
Interoperabilidad _Pretest	,267	139	,000
Interoperabilidad _Postest	,343	139	,000

Fuente: Elaboración propia (Corrección de significación de Lilliefors)

Conforme con la tabla N° 12, se aprecia que los resultados del pretest y postest de la dimensión **interoperabilidad** obtuvieron cantidades menores a 0.05. Por lo cual, se rechaza la hipótesis nula y se acepta la hipótesis alterna y se concluye que la dimensión interoperabilidad no se distribuye normalmente.

Se realizó la prueba de normalidad para la dimensión **mantenibilidad**.

H_0: Los datos de la dimensión mantenibilidad sigue una distribución normal.

H_a: Los datos de la dimensión mantenibilidad no sigue una distribución normal.

Tabla 16: *Prueba de normalidad de la dimensión mantenibilidad*

	Kolmogórov-Smirnov		
	Estadístico	gl	Sig.
Mantenibilidad _Pretest	,258	139	,000
Mantenibilidad _Postest	,321	139	,000

Fuente: Elaboración propia (Corrección de significación de Lilliefors)

Conforme con la tabla N° 13, se aprecia que los resultados del pretest y postest de la dimensión **mantenibilidad** obtuvieron cantidades menores a 0.05. Por lo cual, se rechaza la hipótesis nula y se acepta la hipótesis alterna y se concluye que la dimensión mantenibilidad no se distribuye normalmente.

Se realizó la prueba de normalidad para la dimensión **fiabilidad**.

H_0: Los datos de la dimensión fiabilidad sigue una distribución normal.

H_a: Los datos de la dimensión fiabilidad no sigue una distribución normal.

Tabla 17: *Prueba de normalidad de la dimensión fiabilidad*

	Kolmogórov-Smirnov		
	Estadístico	Gl	Sig.
Fiabilidad _Pretest	,251	139	,000
Fiabilidad _Postest	,221	139	,000

Fuente: Elaboración propia (Corrección de significación de Lilliefors)

Conforme con la tabla N° 14, se aprecia que los resultados del pretest y postest de la dimensión **fiabilidad** obtuvieron cantidades menores a 0.05. Por lo cual, se

rechaza la hipótesis nula y se acepta la hipótesis alterna y se concluye que la dimensión fiabilidad no se distribuye normalmente.

Se realizó la prueba de normalidad para la dimensión **escalabilidad**.

H_0: Los datos de la dimensión escalabilidad sigue una distribución normal.

H_a: Los datos de la dimensión escalabilidad no sigue una distribución normal.

Tabla 18: *Prueba de normalidad de la dimensión escalabilidad*

	Kolmogórov-Smirnov		
	Estadístico	gl	Sig.
Escalabilidad _Pretest	,330	139	,000
Escalabilidad _Postest	,214	139	,000

Fuente: Elaboración propia (Corrección de significación de Lilliefors)

Conforme con la tabla N° 15, se aprecia que los resultados del pretest y postest de la dimensión **escalabilidad** obtuvieron cantidades menores a 0.05. Por lo cual, se rechaza la hipótesis nula y se acepta la hipótesis alterna y se concluye que la dimensión escalabilidad no se distribuye normalmente.

4.2.10. Prueba de hipótesis

El desarrollo de los microservicios y sus dimensiones asociadas no tienen una distribución normal, como lo demuestra el hecho de que su valor de p es inferior a 0,05. Por eso se usó la prueba de Wilcoxon. Durante la prueba de hipótesis se utilizaron los siguientes criterios para tomar una determinación:

a) Hipótesis

H_0: La arquitectura de software APH no mejora favorablemente el desarrollo de microservicios en la empresa Hiper.

H_a: La arquitectura de software APH mejora favorablemente el desarrollo de microservicios en la empresa Hiper.

b) Nivel de significancia

α = 0.05 el margen de error de la prueba de hipótesis

Nivel de confianza = 95%

c) Elección de la prueba estadística

Wilcoxon

d) Estimación del P-Valor

Si p-valor <= α se rechaza la hipótesis nula.

Si p-valor > α no se rechaza la hipótesis nula.

Tabla 19: *Prueba de Wilcoxon para el desarrollo de microservicios*

	Pruebas de rangos con signos de Wilcoxon	
	Z	Sig. Asint (bilateral)
Desarrollo de microservicios Pretest y Postest	-,137	,000

Fuente: Elaboración propia (Base de datos SPSS)

Se aprecia en la tabla N° 16, la prueba de hipótesis, lo cual permite ver el valor Sig., del **desarrollo de microservicios** es 0,00, siendo menor a α= 0.05, se rechaza la hipótesis nula y se acepta la hipótesis alterna con una confianza de 95%. En ese sentido, la arquitectura de software APH mejora favorablemente el desarrollo de microservicios en la empresa Hiper.

Igualmente, se realizó la prueba de hipótesis para la dimensión **interoperabilidad**.

H_0: La arquitectura de software APH no incrementa la interoperabilidad de los microservicios.

H_a: La arquitectura de software APH incrementa la interoperabilidad de los microservicios.

Tabla 20: *Prueba de Wilcoxon para la dimensión interoperabilidad*

	Pruebas de rangos con signos de Wilcoxon	
	Z	Sig. Asint (bilateral)
Interoperabilidad de Pretest y Postest	-3,830	,000

Fuente: Elaboración propia (Base de datos SPSS)

Se aprecia en la tabla N° 17, la prueba de hipótesis, lo cual permite ver el valor Sig., de la dimensión **interoperabilidad** es 0,00, siendo menor a α= 0.05, se rechaza la hipótesis nula y se acepta la hipótesis alterna con una confianza de 95%. En ese sentido, la arquitectura de software APH si incrementa la interoperabilidad de los microservicios.

Se realizó la prueba de hipótesis para la dimensión **mantenibilidad**.

H_0: La arquitectura de software APH no mejora la mantenibilidad de los microservicios.

H_a: La arquitectura de software APH mejora la mantenibilidad de los microservicios.

Tabla 21: *Prueba de Wilcoxon para la dimensión mantenibilidad*

	Pruebas de rangos con signos de Wilcoxon	
	Z	Sig. Asint (bilateral)
Mantenibilidad de Pretest y Postest	-3,157	,002

Fuente: Elaboración propia (Base de datos SPSS)

Se aprecia en la tabla N° 18, la prueba de hipótesis, lo cual permite ver el valor Sig., de la dimensión **interoperabilidad** es 0,02, siendo menor a α= 0.05, se rechaza la hipótesis nula y se acepta la hipótesis alterna con una confianza de 95%. En ese sentido, la arquitectura de software APH mejora la mantenibilidad de los microservicios.

Se realizó la prueba de hipótesis para la dimensión **Fiabilidad**.

H_0: La arquitectura de software APH no aumenta la fiabilidad de los microservicios.

H_a: La arquitectura de software APH aumenta la fiabilidad de los microservicios.

Tabla 22: *Prueba de Wilcoxon para la dimensión fiabilidad*

	Pruebas de rangos con signos de Wilcoxon	
	Z	Sig. Asint (bilateral)
Fiabilidad de Pretest y Postest	-1,462	,001

Fuente: Elaboración propia (Base de datos SPSS)

Se aprecia en la tabla N° 19, la prueba de hipótesis, lo cual permite ver el valor Sig., de la dimensión interoperabilidad es 0,01, siendo menor a α= 0.05, se rechaza la hipótesis nula y se acepta la hipótesis alterna con una confianza de 95%. En ese sentido, la arquitectura de software APH aumenta la fiabilidad de los microservicios.

Se realizó la prueba de hipótesis para la dimensión **escalabilidad**.

H_0: La arquitectura de software APH no maximiza la escalabilidad de los microservicios.

H_a: La arquitectura de software APH maximiza la escalabilidad de los microservicios.

Tabla 23: *Prueba de Wilcoxon para la dimensión escalabilidad*

	Pruebas de rangos con signos de Wilcoxon	
	Z	Sig. Asint (bilateral)
Escalabilidad de Pretest y Postest	-5,830	,000

Fuente: Elaboración propia (Base de datos SPSS)

Se aprecia en la tabla N° 20, la prueba de hipótesis, lo cual permite ver el valor Sig., de la dimensión escalabilidad es 0,00, siendo menor a α= 0.05, se rechaza la hipótesis nula y se acepta la hipótesis alterna con una confianza de 95%. En ese sentido, la arquitectura de software APH maximiza la escalabilidad de los microservicios.

Toma de decisiones

Se ha obtenido p< 0.05 entonces rechazamos la hipótesis nula y nos quedamos con la hipótesis alterna.

Se concluye con un 95% de confiabilidad, se mejora en el desarrollo de microservicios basado en la arquitectura APH.

4.3. Discusión de resultados

4.3.1. Sobre el desarrollo de la solución tecnológica

Debido a que la arquitectura tiene un enfoque modularizado, fue necesario implementar cada módulo en base a pequeños microservicios, por lo que su desarrollo se hizo en varias iteraciones.

La construcción de la solución responde a la arquitectura de solución planteada, para ello se siguió de rigor los seis flujos de trabajo por la interacción, iniciándose con la identificación de requerimientos, para luego en el análisis refinarlos y definirlos, en el diseño consolidar el prototipo de la arquitectura, durante el desarrollo construir todos los componentes que integra la arquitectura, pruebas de la solución donde se verificó su funcionamiento y finalmente la implementación de la arquitectura APH en el repositorio de GitHub.

A partir de las pruebas realizadas sobre la arquitectura, se obtuvo un nivel de aceptación “Acuerdo” y “Totalmente acuerdo” respecto a la solución tecnológica. Ver tabla N° 11.

4.3.2. Sobre la cuantificación de indicadores de la matriz de operacionalización de variables

El 41% de los empleados afirma que se mejoró el desarrollo de microservicios a un nivel eficiente y el 38.8% a un nivel regular, mientras 20.1% calificó en un nivel deficiente, comparando con la primera encuesta sin la arquitectura APH fue del 43.2% de los encuestados calificaba a un nivel deficiente, el 33.1% a un nivel regular y el 23.7% eficiente.

Con ello queda demostrado que la arquitectura de software APH mejoró el desarrollo de microservicios en la organización Hiper.

4.3.3. Sobre la arquitectura APH en la mejora del desarrollo de microservicios en la organización Hiper

a) Interoperabilidad

La dimensión de interoperabilidad ha sido calificada en un nivel deficiente con 5.8%, a un nivel regular 40.2% y el 54% a un nivel eficiente. Por lo cual se demuestra la efectividad de aplicar la arquitectura APH en el desarrollo de los microservicios.

b) Mantenibilidad

A partir de la prueba post se obtuvo que el 12.2% calificaron a un nivel deficiente, el 36.7% a un nivel regular y el 51.1% a un nivel eficiente. Por lo cual se demuestra la efectividad de aplicar la arquitectura de software APH.

c) Fiabilidad

Respecto la fiabilidad los resultados se muestran que el 33.8% calificaron deficiente, el 23% regular y el 43.2% a un nivel eficiente. Por lo cual se demuestra que el nivel preponderante es el nivel eficiente y es efectivo aplicar la arquitectura APH en el desarrollo de los microservicios.

d) Escalabilidad

Los resultados respecto la dimensión de escalabilidad mostraron que el 25.9% calificaron un nivel deficiente, el 31.7% a regular y el 42.4% a un nivel eficiente. Por lo cual se demuestra la efectividad de aplicar la arquitectura de software APH.

5. CONCLUSIONES

a) Se logro validar que la recopilación de los requerimientos funcionales de los usuarios pertenecientes a los diferentes departamentos a través de encuestas, entrevistas y observaciones son sumamente importantes para lograr un desarrollo optimo cubriendo todas las necesidades con la solución tecnológica propuesta.

b) Con la implementación de la arquitectura APH se logró cubrir los objetivos planificados, para mejorar el desarrollo de los microservicios, que permitirá a la organización como tal brindar mejores servicios y dar confianza a sus clientes.

c) La tecnología y la arquitectura propuesta permite seguir contribuyendo más funcionalidades a la arquitectura con el objetivo de que esta se convierta en una herramienta robusta que cobija desde el ámbito de desarrollo hasta los despliegues automatizados.

d) Se concluye con un 95% de confiabilidad, nivel de calificación de 41% eficiente y 38.8% regular la mejora en el desarrollo de microservicios basados en la arquitectura de software APH.

e) La solución tecnológica implementada tuvo una aceptación “Acuerdo” y “Totalmente acuerdo” en base a las dimensiones planteadas en reusabilidad, disponibilidad, modificabilidad y seguridad.

6. RECOMENDACIONES

- En el desarrollo de proyectos de tesis se debe cumplir con los flujos de trabajo acorde en el cronograma previamente establecido, a fin de evitar retrasos e inconvenientes a último momento.
- Se recomienda ir documentando toda la información recibida en todo el proceso del desarrollo de un proyecto de software.
- La toma de información debe realizarse con el soporte de soluciones tecnológicas basadas en plataforma web y uso de dispositivos móviles, para la optimización de los recursos tiempo y costo.
- Se recomienda dar la capacitación correspondiente a los departamentos involucrados sobre la funcionalidad de la arquitectura, así lo mismo los detalles técnicos orientados al equipo de arquitectura quienes estarán administrando la solución tecnológica implementada.

REFERENCIAS BIBLIOGRÁFICAS

Refactoring Guru. (s.f.). *¿Qué es un patrón de diseño?* Patrones de diseño: https://refactoring.guru/es/design-patterns/what-is-pattern

Albertos, E. (2018). Arquitecturas de software para microservicios: Una revisión sistemática de la literatura. *(Tesis maestría)*. Universidad Politécnica de Madrid, Madrid, España.

Alvarez Caules, C. (18 de setiembre de 2018). *¿Qué es un Java Maven Artifact?* Arquitectura Java: https://www.arquitecturajava.com/que-es-un-java-maven-artifact/

Alvarez Villanueva, E. (2017). *Introducción a JMS (Java Message Service)*. https://docplayer.es/47732479-Introduccion-a-jms-java-message-service.html

Apache. (2022). *Apache Tomcat*. https://tomcat.apache.org/

Aprender Big Data. (2022). *Arquitectura de microservicios: Introducción*. https://aprenderbigdata.com/microservicios/.

AWS. (s.f.). *¿Qué es Java?* https://aws.amazon.com/es/what-is/java/

AWS. (s.f.). *Elastic Load Balancing*. https://docs.aws.amazon.com/es_es/elasticloadbalancing/latest/application/introduction.html

Baeldung. (2021). *The DAO Pattern in Java*. https://www.baeldung.com/java-dao-pattern

Baeldung. (s.f.). *The DTO Pattern (Data Transfer Object)*. https://www.baeldung.com/java-dto-pattern

Bamboo Agile. (2022). *Pros and Cons of Using Spring Boot*. https://bambooagile.eu/insights/pros-and-cons-of-using-spring-boot/

Blancarte Iturralde, O. J. (2017). *Webhook una alternativa al Polling*. https://www.oscarblancarteblog.com/2017/09/11/webhook-una-alternativa-al-polling/

Blancarte Iturralde, O. J. (2020). *Introducción a la arquitectura de software - Un enfoque práctico.*

Blancarte Iturralde, O. J. (2020). *Log Aggregation*. https://reactiveprogramming.io/blog/es/patrones-arquitectonicos/log-aggregation

Blancarte Iturralde, O. J. (2020). *Polling*. https://reactiveprogramming.io/blog/es/patrones-arquitectonicos/polling

C. V. GUNTUR. (05 de mayo de 2020). *UNDERSTANDING APACHE MAVEN – PART 1 – BASICS*. https://cguntur.me/2020/05/22/understanding-apache-maven-part-1-basics/

Covarrubias, G. (s.f.). *Escalamiento Vertical Vs Horizontal – Aquitectura de Sistemas*. https://c4-technologies.com/escalamiento-vertical-vs-horizontal/

Das, T. (31 de octubre de 2022). *Los 14 mejores repositorios de alojamiento de paquetes para sus proyectos DevOps*. GEEKFLARE: https://geekflare.com/es/best-package-hosting-repo/

Fugaro, L., & Vocale, M. (2019). Hands-On Cloud-Native Microservices with Jakarta EE. Birmingham.

Garcia Puebla, I. (9 de junio de 2008). *Arquetipos de maven: cómo crear, distribuir y generar proyectos con JSF e ICEfaces, JBoss y EJB3*. https://www.adictosaltrabajo.com/2008/06/09/creararquetiposmaven/

Git. (2022). *Git Description*. https://git-scm.com/docs/git

GitHub. (s.f.). *Introduction to GitHub Packages*. https://docs.github.com/es/packages/learn-github-packages/introduction-to-github-packages.

Google Cloud. (s.f.). *¿Qué es la arquitectura de microservicios?* https://cloud.google.com/learn/what-is-microservices-architecture

Gustavo, B. (14 de noviembre de 2022). *Qué es GitHub y cómo empezar a usarlo*. https://www.hostinger.es/tutoriales/que-es-github

Halterman, J. (s.f.). *Why ModelMapper?* Modelmapper: http://modelmapper.org/

Hernandez, R. (2021). *El patrón modelo-vista-controlador: Arquitectura y frameworks explicados*. https://www.freecodecamp.org/espanol/news/el-modelo-de-arquitectura-view-controller-pattern/

Indrasiri, K. (2018). Microservices for the Enterprise. USA: Apress.

JavabyKiran. (s.f.). *Key Components and Internals of Spring Boot Framework*. https://www.jbktutorials.com/spring-boot/key-components-of-spring-boot-framework.php#gsc.tab=0

JWT. (s.f.). *Introduction to JSON Web Tokens*. https://jwt.io/introduction

KeepCoding. (2022). *¿Qué es una arquitectura de software?* KeepCoding: https://keepcoding.io/blog/que-es-arquitectura-software/

Kinsta. (2022). *What is GitHub? A Beginner's introduction to GitHub*. https://kinsta.com/knowledgebase/what-is-github/

Lewis, J., & Fowler, M. (2014). *Microservice Resource Guide.* Chicago.

Lopez, D., & Maya, E. (2017). Arquitectura de software basada en microservicios para el desarrollo de aplicaciones web de la asamblea nacional. *(Tesis de maestría).* Universidad Técnica del Norte, Ibarra, Ecuador.

López, J. (2022). *Patrones de arquitectura de software*. https://www.dreams.es/transformacion-digital/desarrolladores-paginas-web/patrones-de-arquitectura-de-software

Martinez Canelo, M. (24 de junio de 2020). *¿Qué son los patrones de diseño de software?* Desarrollo web: https://profile.es/blog/patrones-de-diseno-de-software/

Microsoft. (2022). *Autenticación y autorización*. Microsoft: https://docs.microsoft.com/es-es/dotnet/architecture/maui/authentication-and-authorization

Microsoft. (s.f.). *Microservice architecture style*. https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices

Middleware. (2021). *What are microservices? How microservices architecture works*. https://middleware.io/blog/microservices-architecture/

Navarro, H. (5 de agosto de 2020). *Escalando aplicaciones digitales*. https://elementi.me/escalando-aplicaciones-digitales/

NewsMDirector. (2020). *¿Qué es un webhook y para qué sirve?* DIRECTOR: https://www.mdirector.com/blog/que-es-un-webhook/

Okta. (s.f.). *Access Tokens*. https://www.oauth.com/oauth2-servers/access-tokens/

Posa, R. (3 de agosto de 2022). *Key Components and Internals of Spring Boot Framework*. DigitalOcean: https://www.digitalocean.com/community/tutorials/key-components-and-internals-of-spring-boot-framework

Reclu IT. (2022). *¿Qué es Spring?* https://recluit.com/que-es-spring/#.YzCAIXbMJPZ

Richardson, C. (2016). Microservices form Design to Development. USA: Nginx.

Richardson, C. (2016). *What are microservices ?* https://microservices.io/

Richardson, C. (s.f.). *Pattern: Monolithic Architecture*. https://microservices.io/patterns/monolithic.html.

Rootstack. (s.f.). *¿Qué es Spring Boot y que ofrece?* https://rootstack.com/es/learning/que-es-spring-boot-y-que-ofrece

Shaw, M., & Garlan, D. (1996). *Software architecture: perspectives on an emerging discipline.* Prentice Hall.

Shubham. (2022). *DAO Design Pattern*. https://www.digitalocean.com/community/tutorials/dao-design-pattern

Spring boot – DTO vs DAO. (2020). https://blog.hillpig.top/java-spring-dto-dao/

Spring. (s.f.). *Overview of Spring Framework*. https://docs.spring.io/spring-framework/docs/5.0.0.RC2/spring-framework-reference/overview.html

Sumo Logic. (s.f.). *Log aggregation - definition & overview*. https://www.sumologic.com/glossary/log-aggregation/

Taylor, R., Medvidović, N., & Dashofy, E. (2009). *Software architecture: Foundations, Theory and Practice.* Wiley.

Tech Target. (2016). *Definition Git*. https://www.techtarget.com/searchitoperations/definition/Git

Tupac, M. (21 de diciembre de 2022). *Arquitectura de Software: ¿Qué es, y cómo funciona?* INGENIERÍA DE SISTEMAS E INFORMÁTICA: https://blog.continental.edu.pe/sistemas-informatica/2013/01/10/arquitectura-de-software-que-es-y-como-funciona/

Universidad de Alicante. (s.f.). *Modelo Vista Controlador (MVC)*. Servicio de Informática ASP.NET MVC 3 Framework: https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html

Webmaster. (s.f.). *Cluster de servidores, ¿qué es y cómo funciona?* https://www.solingest.com/cluster-de-servidores-que-es-y-como-funciona

Yagüe, C. (29 de abril de 2019). *Qué es Apache Maven*. OpenWebinars: https://openwebinars.net/blog/que-es-apache-maven/

ANEXOS

ANEXO 01: MATRIZ DE CONSISTENCIA DEL PROYECTO

Arquitectura de software APH para mejorar el desarrollo de microservicios en la empresa Hiper, año 2022.

Problemas	Objetivos	Hipótesis	Variables y dimensiones	Metodología
Problema general ¿En qué medida mejora el desarrollo de microservicios en la empresa Hiper con la arquitectura de software APH? **Problema específico 1** ¿Cómo implementar la arquitectura de software APH que satisfaga los requerimientos de construcción de los microservicios? **Problema específico 2** ¿Determinar en qué medida se incrementa la interoperabilidad de los microservicios construidos con la arquitectura de software APH? **Problema específico 3** ¿Determinar en qué medida mejora la mantenibilidad de los microservicios construidos con la arquitectura de software APH? **Problema específico 4** ¿Determinar en qué medida se aumenta la fiabilidad de los microservicios construidos con la arquitectura de software APH? **Problema específico 5** ¿Determinar en qué medida se maximiza la escalabilidad de los microservicios construidos con la arquitectura de software APH?	**Objetivo general** Mejorar el desarrollo de microservicios mediante la arquitectura de software APH en la empresa Hiper. **Objetivo específico 1** Implementar la arquitectura de software APH que satisfaga los requerimientos de construcción de los microservicios. **Objetivo específico 2** Incrementar la interoperabilidad en la construcción de los microservicios en base a la arquitectura de software APH. **Objetivo específico 3** Mejorar la mantenibilidad de los microservicios en base a la arquitectura de software APH. **Objetivo específico 4** Aumentar la fiabilidad de los microservicios en base a la arquitectura software APH. **Objetivo específico 5** Maximizar la escalabilidad de los microservicios con la arquitectura de software APH.	**Hipótesis general** La arquitectura de software APH mejora favorablemente el desarrollo de microservicios en la empresa Hiper. **Hipótesis especifico 1** La implementación de la arquitectura de software APH satisface los requerimientos de construcción de los microservicios. **Hipótesis especifico 2** La arquitectura de software APH incrementa la interoperabilidad de los microservicios. **Hipótesis especifico 3** La arquitectura de software APH mejora la mantenibilidad de los microservicios. **Hipótesis especifico 4** La arquitectura de software APH aumenta la fiabilidad de los microservicios. **Hipótesis especifico 5** La arquitectura de software APH maximiza la escalabilidad de los microservicios.	**Variable independiente:** Arquitectura de software APH **Dimensiones:** ▪ Reusabilidad ▪ Disponibilidad ▪ Modificabilidad ▪ Seguridad **Variable dependiente:** Desarrollo de microservicios **Dimensiones:** ▪ Interoperabilidad ▪ Mantenibilidad ▪ Fiabilidad ▪ Escalabilidad	**Tipo de investigación** La investigación realizada corresponde a los siguientes tipos: Según la intervención del investigador: Experimental Según la planificación de la toma de decisiones: Prospectivo Según en que mide la variable de estudio: Longitudinal Según variables de interés: Analítico **Nivel de investigación** La investigación que se realizará corresponde al nivel correlacional casual. **Método de investigación** La presente investigación por la naturaleza de las variables en estudio, se utilizará el método aplicado, de análisis y correlacional. **Diseño de la investigación** Descriptiva **Población de estudio:** 218 personales. **Muestra necesaria:** De acuerdo con el cálculo realizado haciendo uso de la formula estadística se tomará a 139 empleados.

Fuente: Elaboración propia

ANEXO 02: INSTRUMENTO DE RECOLECCIÓN DE DATOS

El presente cuestionario tiene por objetivo recolectar información del personal que integra los departamentos de desarrollo, arquitectura, seguridad y calidad de la empresa Hiper. Se solicita de forma objetiva poder completar con cada una de las preguntas, así podamos determinar como la arquitectura de software incide en la mejora del desarrollo de microservicios.

Se le agradece su participación y por ser parte de esta innovación, la información que nos proporciona nos permitirá medir los resultados de satisfacción del desarrollo de microservicios en base a la arquitectura de software APH.

Nota: La presente encuesta tendrá carácter secreta y anónima; no tendrá preguntas malas o buenas, marcar la opción que usted crea conveniente.

Totalmente desacuerdo	En desacuerdo	Neutral	De acuerdo	Totalmente acuerdo
1	2	3	4	5

N°	Criterios	1	2	3	4	5
Arquitectura de software APH						
Reusabilidad						
1	¿Considera que la arquitectura incluye los componentes comunes?	○	○	○	◉	○
2	¿Cree que la arquitectura nos brinda flexibilidad en el uso de los componentes?	○	○	○	○	◉
3	¿Considera que la arquitectura es modular porque podemos ir agregando otros componentes en el futuro?	○	○	○	◉	○
Disponibilidad						
4	¿Considera que la arquitectura está siempre a la disposición para su uso?	○	○	○	○	◉
5	¿Cree que la arquitectura se adapta a las integraciones con servicios externos?	○	○	○	◉	○
6	¿Cree que las nuevas versiones de la arquitectura estarán disponibles de forma abstracta e inmediata?	○	○	○	○	◉
Modificabilidad						
7	¿Considera que la arquitectura sea modificable para incrementar su robustez con nuevas características?	○	○	○	○	◉

8	¿Cree que los cambios que surjan en la arquitectura son transparentes para el equipo de desarrollo?	○	○	○	○	◉
9	¿Considera que en base a la arquitectura sería óptimo la actualización de los proyectos a las nuevas tecnologías?	○	○	○	◉	○
Seguridad						
10	¿Considera que la arquitectura brinda seguridad y confianza a los clientes?	○	○	○	○	◉
11	¿Cree que es la mejor opción que la arquitectura se encargue de la seguridad de los microservicios?	○	○	○	◉	○
12	¿Considera que la arquitectura soporta las tecnologías actuales para la seguridad?	○	○	○	○	◉
Desarrollo de microservicios						
Interoperabilidad						
1	¿Cree que los microservicios están en la capacidad de integrarse con servicios externos y/o internos?	○	○	○	○	◉
2	¿Considera que la integración con servicios externos incrementará dado al soporte a diferentes tecnologías?	○	○	○	◉	○
3	¿Considera que es eficiente y seguro la comunicación con servicios externos?	○	○	○	○	◉
Mantenibilidad						
4	¿Considera que la inclusión de nuevos componentes en la arquitectura sea adaptable a los microservicios?	○	○	○	○	◉
5	¿Considera flexible la inserción de cambios a los microservicios?	○	○	○	◉	○
Fiabilidad						
6	¿Considera que los microservicios están en la capacidad de tolerancia a fallos?	○	○	○	○	◉
7	¿Considera que los tiempos de respuesta de los microservicios son óptimos?	○	○	○	○	◉
8	¿Considera más eficiente tener el control de errores técnicos en la arquitectura más no en los microservicios?	○	○	○	◉	○
Escalabilidad						
9	¿Cree que los microservicios alcancen su robustez en base a nuevas versiones de la arquitectura?	○	○	○	○	◉
10	¿Considera que los microservicios están modularizados?	○	○	○	◉	○
11	¿Considera que el desarrollo de microservicios sigue un estándar y que su compresión es mucho más eficiente?	○	○	○	○	◉

Fuente: Elaboración propia

ANEXO 03: MATRIZ DE JUICIO DEL EXPERTO

JUICIO DE EXPERTO SOBRE LA MATRIZ DE VALIDACIÓN DEL INSTRUMENTO PARA EL MODELO DE CALIDAD QUE SERÁ APLICADA A LOS ELEMENTOS DE LA MUESTRA.

Matriz de juicio de experto sobre el instrumento de medición documental.

INFORME DE OPINION DE EXPERTOS DEL INSTRUMENTO DE RECOLECCIÓN DATOS GENERALES

- **Apellidos y nombres del experto:** Alvarado Tolentino Joseph Darwin
- **Grado académico:** Magister
- **Profesión:** Ingeniero de Sistemas e Informática
- **Autor del instrumento:** Aldo Omar Morales Carlos

INSTRUCCIONES:

Seleccione la opción correspondiente al aspecto cualitativo de cada ítem y alternativa de respuesta, según los criterios que a continuación se detallan.

Totalmente desacuerdo	En desacuerdo	Neutral	De acuerdo	Totalmente acuerdo
1	2	3	4	5

Titulo	**Arquitectura de software APH para mejorar el desarrollo de microservicios en la empresa Hiper, año 2022**					
Objetivo	Mejorar el desarrollo de microservicios mediante la arquitectura de software APH en la empresa Hiper					
Indicadores	**Criterios**	**1**	**2**	**3**	**4**	**5**
Claridad	Está formulado en un lenguaje apropiado.	○	○	○	○	◉
Objetividad	Está expresado en conductas observables.	○	○	○	○	◉
Organización	Existe orden lógico de ideas.	○	○	○	○	◉

Suficiencia	Comprende las dimensiones de la investigación en cantidad y calidad.	○	○	○	○	◉
Intencionalidad	Adecuado para valorar la variable seleccionada.	○	○	○	○	◉
Consistencia	Basado en el aspecto teórico científico y del tema de estudio.	○	○	○	○	◉
Coherencia	Hay relación entre variables, dimensiones e indicadores.	○	○	○	○	◉
Metodología	El instrumento se relaciona con el método planteado en el proyecto.	○	○	○	○	◉

Firma del evaluador

INFORME DE OPINION DE EXPERTOS DEL INSTRUMENTO DE RECOLECCIÓN DATOS GENERALES

- **Apellidos y nombres del experto:** Jamanca Ramirez Marco Antonio
- **Grado académico:** Ingeniero
- **Profesión:** Ingeniero de Sistemas e Informática
- **Autor del instrumento:** Aldo Omar Morales Carlos

INSTRUCCIONES:

Seleccione la opción correspondiente al aspecto cualitativo de cada ítem y alternativa de respuesta, según los criterios que a continuación se detallan.

Totalmente desacuerdo	En desacuerdo	Neutral	De acuerdo	Totalmente acuerdo

1	2	3	4	5

Titulo	**Arquitectura de software APH para mejorar el desarrollo de microservicios en la empresa Hiper, año 2022**					
Objetivo	Mejorar el desarrollo de microservicios mediante la arquitectura de software APH en la empresa Hiper					
Indicadores	**Criterios**	**1**	**2**	**3**	**4**	**5**
Claridad	Está formulado en un lenguaje apropiado.	○	○	○	◉	○
Objetividad	Está expresado en conductas observables.	○	○	○	◉	○
Organización	Existe orden lógico de ideas.	○	○	○	◉	○
Suficiencia	Comprende las dimensiones de la investigación en cantidad y calidad.	○	○	○	◉	○
Intencionalidad	Adecuado para valorar la variable seleccionada.	○	○	○	◉	○
Consistencia	Basado en el aspecto teórico científico y del tema de estudio.	○	○	○	◉	○
Coherencia	Hay relación entre variables, dimensiones e indicadores.	○	○	○	◉	○
Metodología	El instrumento se relaciona con el método planteado en el proyecto.	○	○	○	◉	○

INFORME DE OPINION DE EXPERTOS DEL INSTRUMENTO DE RECOLECCIÓN DATOS GENERALES

- **Apellidos y nombres del experto:** Rosales Maguiña Lilian Rocío
- **Grado académico:** Maestría en Administración, con mención en Administración de Negocios, MBA.
- **Profesión:** Ingeniero en Informática y Sistemas
- **Autor del instrumento:** Aldo Omar Morales Carlos

INSTRUCCIONES:

Seleccione la opción correspondiente al aspecto cualitativo de cada ítem y alternativa de respuesta, según los criterios que a continuación se detallan.

Totalmente desacuerdo	En desacuerdo	Neutral	De acuerdo	Totalmente acuerdo
1	2	3	4	5

Titulo	**Arquitectura de software APH para mejorar el desarrollo de microservicios en la empresa Hiper, año 2022**					
Objetivo	Mejorar el desarrollo de microservicios mediante la arquitectura de software APH en la empresa Hiper					
Indicadores	**Criterios**	**1**	**2**	**3**	**4**	**5**
Claridad	Está formulado en un lenguaje apropiado.	○	○	○	○	◉
Objetividad	Está expresado en conductas observables.	○	○	○	◉	○
Organización	Existe orden lógico de ideas.	○	○	○	○	◉
Suficiencia	Comprende las dimensiones de la investigación en cantidad y calidad.	○	○	○	○	◉
Intencionalidad	Adecuado para valorar la variable seleccionada.	○	○	○	◉	○
Consistencia	Basado en el aspecto teórico científico y del tema de estudio.	○	○	○	○	◉

Coherencia	Hay relación entre variables, dimensiones e indicadores.	○	○	○	◉	○
Metodología	El instrumento se relaciona con el método planteado en el proyecto.	○	○	○	◉	○

ANEXO 04: INTEGRACIÓN DE MICROSERVICIOS ATR

La tabla siguiente muestra las integraciones o sincronizaciones que se tienen que realizar y el método que se debe usar para realizar estas integraciones.

Gráfico X: *Integraciones con servicios de terceros*

Módulo ATR	Integración	Funcionalidad que lo requiere	Tipo Integración	¿Cómo realizar la integración?	Periodicidad	Mantenimiento integración	Observaciones
DREF	Flag Habilitación MSC (IARMC)	Acceso MSC Creación de ofertas - validación	Topic		Inmediata		Se requiere inmediata ya que tiene consecuencias operativas (cancelar las ofertas)
	Habilitación (FCMA)	Acceso MSC Creación de ofertas - validación Casación ofertas	Topic		Inmediata		Se requiere inmediata ya que tiene consecuencias operativas (cancelar las ofertas)
	Servicios ATR	Producto Panel de ofertas Creación de ofertas Consulta ofertas Administración de filtros	Réplica	Carga inicial + actualización periódica	Diaria	Permitir lanzarla manualmente	
	Instalación (Agrupación infr, infr y puntos)	Producto Panel de ofertas Creación de ofertas Creación de ofertas - validación (PCI VIP) Consulta ofertas Administración de filtros	Manual/Réplica	Carga inicial + actualización periódica	Diaria	Permitir lanzarla manualmente	
	Sujetos	Panel de ofertas Creación de ofertas (GTS) Agredir ofertas (GTS) Consulta ofertas Administración de Filtros	Réplica/Topic				
	Unidades capacidad del servicio	Creación de ofertas Consulta ofertas Administración de filtros	Manual			¿Cómo hacer si cambian las unidades de un servicio y tengo histórico=	
CONT	Contratos	Creación de ofertas Creación de ofertas - validación Agredir ofertas Consulta ofertas Casación ofertas	API				
	Capacidad contratada y primas	Creación de ofertas - validación Modificación oferta Casación ofertas	API				
	Transacciones (crear)	Casación de ofertas	API				
	Ofertas (CONT)	Otros Procesos de CONT - Cesiones - Subarriendos - Cambios de titularidad - Renuncia	API				¿Qué ocurre en caso de fallo?
GAR	Bloquear garantías ofertas	Creación de ofertas - validación Modificación oferta Casación ofertas	API				
	Bloquear garantías transacciones (CONT)	Casación de ofertas	Interna				
	Liberar garantías	Cancelar oferta Casación	API				
	Unidades de precio	Creación de ofertas Consulta ofertas Administración de filtros	Manual				
PREFACT	Transacciones con Prima		Topic (ya existente)				Ver si lo cogemos desde MSC o desde CONT
SAP	Transacciones		WS				
DWH	Ofertas		ETL				
	Transacciones con Prima		ETL				
EXTERNO	Envio de ofertas XML	Creación de ofertas Modificación ofertas Cancelación ofertas					

Fuente: Integración de microservicios ATR (Intranet Hiper, 2021)

ANEXO 05: PRETEST – DESARROLLO DE MICROSERVICIOS

Tabla 24: *Encuesta Pre-Test de desarrollo de microservicios*

VARIABLES	V2: Desarrollo de microservicios															
DIMENSIONES	Interoperabilidad			SUBTOTAL	Mantenibilidad		SUBTOTAL	Fiabilidad			SUBTOTAL	Escalabilidad			SUBTOTAL	SUMA TOTAL
PREGUNTAS	P1	P2	P3		P4	P5		P6	P7	P8		P9	P10	P11		
E1	3	4	1	8	3	3	6	3	2	3	8	2	4	3	9	31
E2	2	3	2	7	4	3	7	4	3	2	9	3	3	2	8	31
E3	2	4	3	9	2	2	4	3	3	2	8	3	4	2	9	30
E4	3	3	1	7	3	3	6	4	3	2	9	3	3	3	9	31
E5	3	3	3	9	3	3	6	2	3	3	8	4	2	3	9	32
E6	4	2	4	10	1	4	5	3	4	2	9	4	4	4	12	36
E7	2	2	2	6	3	2	5	2	2	2	6	2	4	1	7	24
E8	2	4	2	8	2	2	4	2	2	2	6	3	1	2	6	24
E9	3	4	3	10	3	3	6	4	3	3	10	3	3	3	9	35
E10	3	2	3	8	3	3	6	4	3	3	10	3	3	1	7	31
E11	2	2	2	6	2	2	4	2	2	3	7	2	2	2	6	23
E12	2	1	3	6	4	1	5	1	4	2	7	1	1	1	3	21
E13	1	2	2	5	2	2	4	4	2	1	7	2	2	2	6	22
E14	2	4	2	8	3	2	5	2	2	2	6	2	2	2	6	25
E15	3	3	3	9	3	3	6	4	3	3	10	3	3	1	7	32
E16	2	2	3	7	2	2	4	2	3	2	7	2	2	2	6	24
E17	3	3	2	8	3	3	6	2	3	3	8	3	3	3	9	31
E18	3	3	3	9	3	3	6	4	3	3	10	3	3	1	7	32
E19	4	2	4	10	3	4	7	3	4	4	11	4	4	4	12	40
E20	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22
E21	3	3	3	9	3	3	6	3	4	3	10	3	3	3	9	34
E22	3	3	3	9	3	3	6	3	4	3	10	3	3	1	7	32
E23	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22
E24	1	1	1	3	1	1	2	3	1	1	5	1	1	1	3	13
E25	3	2	3	8	3	3	6	3	3	3	9	3	3	2	8	31
E26	2	3	3	8	3	3	6	3	4	3	10	3	2	3	8	32
E27	3	4	3	10	3	3	6	4	3	3	10	3	1	2	6	32
E28	4	2	3	9	2	3	5	3	3	3	9	3	3	3	9	32

E29	3	3	3	9	3	3	6	3	4	3	10	3	2	3	8	33
E30	3	3	3	9	3	3	6	3	3	3	9	3	3	1	7	31
E31	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22
E32	1	1	1	3	4	1	5	1	1	1	3	1	1	1	3	14
E33	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22
E34	2	2	2	6	3	2	5	2	3	2	7	2	1	2	5	23
E35	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22
E36	2	4	2	8	1	2	3	2	2	2	6	2	2	2	6	23
E37	2	3	2	7	4	2	6	2	2	2	6	2	2	2	6	25
E38	2	3	2	7	2	2	4	2	2	2	6	2	2	2	6	23
E39	3	2	3	8	3	3	6	3	3	3	9	3	3	3	9	32
E40	3	3	3	9	3	3	6	3	3	3	9	3	3	3	9	33
E41	1	1	1	3	1	1	2	3	1	1	5	1	1	1	3	13
E42	2	3	2	7	2	2	4	2	2	2	6	2	2	2	6	23
E43	3	2	3	8	3	3	6	3	3	3	9	3	3	3	9	32
E44	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22
E45	3	4	3	10	4	3	7	3	3	3	9	3	3	3	9	35
E46	3	3	3	9	4	3	7	3	3	3	9	3	3	3	9	34
E47	3	3	3	9	3	3	6	3	3	3	9	3	3	2	8	32
E48	3	1	1	5	1	1	2	1	1	2	4	1	1	1	3	14
E49	2	2	2	6	3	2	5	2	2	2	6	2	2	2	6	23
E50	2	4	2	8	2	2	4	2	2	2	6	2	2	2	6	24
E51	2	2	2	6	3	2	5	2	2	2	6	2	2	2	6	23
E52	2	3	2	7	2	2	4	2	2	2	6	2	2	2	6	23
E53	3	3	3	9	3	3	6	3	2	3	8	3	3	1	7	30
E54	2	4	3	9	1	3	4	3	3	3	9	3	3	2	8	30
E55	3	3	3	9	3	3	6	3	2	3	8	3	3	3	9	32
E56	1	2	3	6	4	3	7	3	3	3	9	3	3	3	9	31
E57	3	1	3	7	3	3	6	3	3	3	9	3	3	3	9	31
E58	3	2	2	7	2	2	4	2	2	2	6	2	2	2	6	23
E59	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22
E60	3	3	3	9	4	3	7	3	3	3	9	3	3	2	8	33
E61	3	4	3	10	3	3	6	3	3	3	9	3	3	3	9	34
E62	2	3	2	7	2	2	4	2	2	2	6	2	2	2	6	23
E63	1	1	1	3	1	1	2	1	3	1	5	1	1	1	3	13
E64	2	4	2	8	3	2	5	2	2	2	6	2	2	2	6	25
E65	2	3	2	7	2	2	4	2	2	2	6	2	2	2	6	23
E66	1	1	1	3	4	1	5	3	1	1	5	1	1	1	3	16
E67	3	3	3	9	3	3	6	3	3	3	9	3	3	3	9	33
E68	1	1	1	3	4	1	5	1	1	2	4	1	1	1	3	15
E69	3	4	1	8	1	1	2	1	1	1	3	1	1	1	3	16
E70	2	3	2	7	3	2	5	2	2	2	6	2	2	2	6	24

E72	3	3	3	9	3	3	6	3	3	3	9	3	3	3	9	33
E73	3	3	3	9	3	3	6	3	3	3	9	3	3	1	7	31
E74	2	2	2	6	4	2	6	2	2	2	6	2	2	2	6	24
E75	2	2	2	6	3	2	5	2	2	2	6	2	2	2	6	23
E76	3	3	3	9	3	3	6	3	3	3	9	3	3	3	9	33
E77	4	2	4	10	4	4	8	4	2	4	10	4	2	2	8	36
E78	3	4	4	11	4	4	8	4	3	4	11	4	1	1	6	36
E79	3	3	3	9	3	3	6	3	3	3	9	3	3	3	9	33
E80	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22
E81	3	3	3	9	3	3	6	3	3	3	9	3	3	3	9	33
E82	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22
E83	3	3	3	9	3	3	6	3	3	3	9	3	3	3	9	33
E84	3	4	3	10	4	3	7	3	3	3	9	3	3	3	9	35
E85	3	3	3	9	3	3	6	3	3	3	9	3	3	3	9	33
E86	3	3	3	9	4	3	7	3	3	3	9	3	3	3	9	34
E87	3	4	4	11	3	4	7	4	4	4	12	4	1	2	7	37
E88	3	3	3	9	2	3	5	3	3	3	9	3	3	3	9	32
E89	4	4	4	12	4	4	8	4	2	4	10	4	2	1	7	37
E90	2	2	2	6	2	2	4	2	2	2	6	2	1	2	5	21
E91	3	4	4	11	3	4	7	4	3	4	11	4	1	2	7	36
E92	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22
E93	1	1	1	3	1	1	2	1	1	2	4	1	2	1	4	13
E94	3	3	3	9	3	3	6	3	2	3	8	3	3	3	9	32
E95	1	1	1	3	1	1	2	3	1	1	5	1	1	1	3	13
E96	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22
E97	3	3	3	9	4	3	7	3	3	3	9	3	3	3	9	34
E98	3	3	3	9	4	3	7	3	3	3	9	3	3	3	9	34
E99	3	4	4	11	3	4	7	4	4	4	12	4	2	2	8	38
E100	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22
E101	4	4	4	12	3	4	7	4	4	4	12	4	1	1	6	37
E102	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22
E103	2	4	2	8	4	2	6	2	2	2	6	2	2	2	6	26
E104	2	2	2	6	3	2	5	2	2	2	6	2	2	2	6	23
E105	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22
E106	3	1	1	5	1	1	2	2	4	2	8	1	1	1	3	18
E107	3	3	3	9	3	3	6	3	3	3	9	3	3	3	9	33
E108	3	3	3	9	4	3	7	3	4	2	9	3	3	3	9	34
E109	3	3	3	9	3	3	6	3	3	3	9	3	1	3	7	31
E110	3	4	3	10	3	3	6	3	3	4	10	3	3	2	8	34
E111	3	3	3	9	4	3	7	3	3	3	9	3	2	3	8	33
E112	3	3	3	9	3	3	6	3	3	3	9	3	3	2	8	32
E113	5	3	3	11	3	3	6	3	4	3	10	3	3	3	9	36
E114	3	3	3	9	4	3	7	3	3	3	9	3	2	3	8	33

E115	3	3	3	9	3	3	6	3	3	3	9	3	3	3	9	33
E116	4	4	4	12	3	4	7	4	4	4	12	4	2	2	8	39
E117	2	2	2	6	4	2	6	2	2	2	6	2	2	1	5	23
E118	1	3	1	5	1	2	3	1	1	1	3	1	1	1	3	14
E119	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22
E120	2	2	2	6	2	2	4	2	2	2	6	2	1	2	5	21
E121	3	3	3	9	3	3	6	3	3	3	9	3	3	3	9	33
E122	3	4	3	10	3	3	6	3	3	3	9	3	3	2	8	33
E123	3	3	3	9	3	3	6	3	3	3	9	3	3	3	9	33
E124	2	2	2	6	3	2	5	2	2	2	6	2	2	2	6	23
E125	2	2	2	6	2	2	4	2	4	2	8	2	2	2	6	24
E126	5	3	3	11	3	4	7	3	3	3	9	3	3	3	9	36
E127	2	2	2	6	4	2	6	2	2	2	6	2	2	2	6	24
E128	2	4	2	8	4	3	7	3	2	2	7	2	2	2	6	28
E129	2	2	2	6	2	2	4	3	2	2	7	2	2	2	6	23
E130	3	3	5	11	3	4	7	3	3	3	9	3	3	3	9	36
E131	1	1	1	3	1	1	2	4	1	1	6	1	1	1	3	14
E132	3	3	3	9	3	3	6	3	3	3	9	3	3	3	9	33
E133	2	2	2	6	4	2	6	4	3	2	9	2	2	2	6	27
E134	2	2	2	6	2	3	5	2	4	2	8	2	1	2	5	24
E135	3	3	3	9	3	3	6	5	3	3	11	3	2	3	8	34
E136	3	3	3	9	4	2	6	3	3	3	9	3	3	3	9	33
E137	3	4	4	11	4	3	7	4	4	4	12	4	1	2	7	37
E138	3	3	3	9	3	3	6	5	3	3	11	3	3	3	9	35
E139	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22

Fuente: Elaboración propia

ANEXO 06: POSTTEST – DESARROLLO DE MICROSERVICIOS

Tabla 25: *Encuesta Post-Test de desarrollo de microservicios*

VARIABLES	V2: Desarrollo de microservicios															
DIMENSIONES	Interoperabilidad			SUBTOTAL	Mantenibilidad		SUBTOTAL	Fiabilidad			SUBTOTAL	Escalabilidad			SUBTOTAL	SUMA TOTAL
PREGUNTAS	P1	P2	P3		P4	P5		P6	P7	P8		P9	P10	P11		
E1	3	4	1	8	3	3	6	3	2	3	8	2	4	3	9	31
E2	2	3	2	7	4	3	7	4	3	2	9	3	3	2	8	31
E3	2	4	3	9	2	2	4	3	3	2	8	3	4	2	9	30
E4	3	3	1	7	3	3	6	4	3	2	9	3	3	3	9	31
E5	3	3	3	9	3	3	6	2	3	3	8	4	2	3	9	32
E6	4	2	4	10	1	4	5	3	4	2	9	4	4	4	12	36
E7	2	2	2	6	3	2	5	2	2	2	6	2	4	1	7	24
E8	2	4	2	8	2	2	4	2	2	2	6	3	1	2	6	24
E9	3	4	3	10	3	3	6	4	3	3	10	3	3	3	9	35
E10	3	2	3	8	3	3	6	4	3	3	10	3	3	1	7	31
E11	2	2	2	6	2	2	4	2	2	3	7	2	2	2	6	23
E12	2	1	3	6	4	1	5	1	4	2	7	1	1	1	3	21
E13	1	2	2	5	2	2	4	4	2	1	7	2	2	2	6	22
E14	2	4	2	8	3	2	5	2	2	2	6	2	2	2	6	25
E15	3	3	3	9	3	3	6	4	3	3	10	3	3	1	7	32
E16	2	2	3	7	2	2	4	2	3	2	7	2	2	2	6	24
E17	3	3	2	8	3	3	6	2	3	3	8	3	3	3	9	31
E18	3	3	3	9	3	3	6	4	3	3	10	3	3	1	7	32
E19	4	2	4	10	3	4	7	3	4	4	11	4	4	4	12	40
E20	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22
E21	3	3	3	9	3	3	6	3	4	3	10	3	3	3	9	34
E22	3	3	3	9	3	3	6	3	4	3	10	3	3	1	7	32
E23	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22
E24	1	1	1	3	1	1	2	3	1	1	5	1	1	1	3	13

E25	3	2	3	8	3	3	6	3	3	3	9	3	3	2	8	31
E26	2	3	3	8	3	3	6	3	4	3	10	3	2	3	8	32
E27	3	4	3	10	3	3	6	4	3	3	10	3	1	2	6	32
E28	4	2	3	9	2	3	5	3	3	3	9	3	3	3	9	32
E29	3	3	3	9	3	3	6	3	4	3	10	3	2	3	8	33
E30	3	3	3	9	3	3	6	3	3	3	9	3	3	1	7	31
E31	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22
E32	1	1	1	3	4	1	5	1	1	1	3	1	1	1	3	14
E33	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22
E34	2	2	2	6	3	2	5	2	3	2	7	2	1	2	5	23
E35	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22
E36	2	4	2	8	1	2	3	2	2	2	6	2	2	2	6	23
E37	2	3	2	7	4	2	6	2	2	2	6	2	2	2	6	25
E38	2	3	2	7	2	2	4	2	2	2	6	2	2	2	6	23
E39	3	2	3	8	3	3	6	3	3	3	9	3	3	3	9	32
E40	3	3	3	9	3	3	6	3	3	3	9	3	3	3	9	33
E41	1	1	1	3	1	1	2	3	1	1	5	1	1	1	3	13
E42	2	3	2	7	2	2	4	2	2	2	6	2	2	2	6	23
E43	3	2	3	8	3	3	6	3	3	3	9	3	3	3	9	32
E44	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22
E45	3	4	3	10	4	3	7	3	3	3	9	3	3	3	9	35
E46	3	3	3	9	4	3	7	3	3	3	9	3	3	3	9	34
E47	3	3	3	9	3	3	6	3	3	3	9	3	3	2	8	32
E48	3	1	1	5	1	1	2	1	1	2	4	1	1	1	3	14
E49	2	2	2	6	3	2	5	2	2	2	6	2	2	2	6	23
E50	2	4	2	8	2	2	4	2	2	2	6	2	2	2	6	24
E51	2	2	2	6	3	2	5	2	2	2	6	2	2	2	6	23
E52	2	3	2	7	2	2	4	2	2	2	6	2	2	2	6	23
E53	3	3	3	9	3	3	6	3	2	3	8	3	3	1	7	30
E54	2	4	3	9	1	3	4	3	3	3	9	3	3	2	8	30
E55	3	3	3	9	3	3	6	3	2	3	8	3	3	3	9	32
E56	1	2	3	6	4	3	7	3	3	3	9	3	3	3	9	31
E57	3	1	3	7	3	3	6	3	3	3	9	3	3	3	9	31
E58	3	2	2	7	2	2	4	2	2	2	6	2	2	2	6	23
E59	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22
E60	3	3	3	9	4	3	7	3	3	3	9	3	3	2	8	33
E61	3	4	3	10	3	3	6	3	3	3	9	3	3	3	9	34
E62	2	3	2	7	2	2	4	2	2	2	6	2	2	2	6	23
E63	1	1	1	3	1	1	2	1	3	1	5	1	1	1	3	13
E64	2	4	2	8	3	2	5	2	2	2	6	2	2	2	6	25
E65	2	3	2	7	2	2	4	2	2	2	6	2	2	2	6	23
E66	1	1	1	3	4	1	5	3	1	1	5	1	1	1	3	16
E67	3	3	3	9	3	3	6	3	3	3	9	3	3	3	9	33

E68	1	1	1	3	4	1	5	1	1	2	4	1	1	1	3	15
E69	3	4	1	8	1	1	2	1	1	1	3	1	1	1	3	16
E70	2	3	2	7	3	2	5	2	2	2	6	2	2	2	6	24
E71	2	4	2	8	2	2	4	2	2	2	6	2	2	2	6	24
E72	3	3	3	9	3	3	6	3	3	3	9	3	3	3	9	33
E73	3	3	3	9	3	3	6	3	3	3	9	3	3	1	7	31
E74	2	2	2	6	4	2	6	2	2	2	6	2	2	2	6	24
E75	2	2	2	6	3	2	5	2	2	2	6	2	2	2	6	23
E76	3	3	3	9	3	3	6	3	3	3	9	3	3	3	9	33
E77	4	2	4	10	4	4	8	4	2	4	10	4	2	2	8	36
E78	3	4	4	11	4	4	8	4	3	4	11	4	1	1	6	36
E79	3	3	3	9	3	3	6	3	3	3	9	3	3	3	9	33
E80	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22
E81	3	3	3	9	3	3	6	3	3	3	9	3	3	3	9	33
E82	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22
E83	3	3	3	9	3	3	6	3	3	3	9	3	3	3	9	33
E84	3	4	3	10	4	3	7	3	3	3	9	3	3	3	9	35
E85	3	3	3	9	3	3	6	3	3	3	9	3	3	3	9	33
E86	3	3	3	9	4	3	7	3	3	3	9	3	3	3	9	34
E87	3	4	4	11	3	4	7	4	4	4	12	4	1	2	7	37
E88	3	3	3	9	2	3	5	3	3	3	9	3	3	3	9	32
E89	4	4	4	12	4	4	8	4	2	4	10	4	2	1	7	37
E90	2	2	2	6	2	2	4	2	2	2	6	2	1	2	5	21
E91	3	4	4	11	3	4	7	4	3	4	11	4	1	2	7	36
E92	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22
E93	1	1	1	3	1	1	2	1	1	2	4	1	2	1	4	13
E94	3	3	3	9	3	3	6	3	2	3	8	3	3	3	9	32
E95	1	1	1	3	1	1	2	3	1	1	5	1	1	1	3	13
E96	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22
E97	3	3	3	9	4	3	7	3	3	3	9	3	3	3	9	34
E98	3	3	3	9	4	3	7	3	3	3	9	3	3	3	9	34
E99	3	4	4	11	3	4	7	4	4	4	12	4	2	2	8	38
E100	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22
E101	4	4	4	12	3	4	7	4	4	4	12	4	1	1	6	37
E102	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22
E103	2	4	2	8	4	2	6	2	2	2	6	2	2	2	6	26
E104	2	2	2	6	3	2	5	2	2	2	6	2	2	2	6	23
E105	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22
E106	3	1	1	5	1	1	2	2	4	2	8	1	1	1	3	18
E107	3	3	3	9	3	3	6	3	3	3	9	3	3	3	9	33
E108	3	3	3	9	4	3	7	3	4	2	9	3	3	3	9	34
E109	3	3	3	9	3	3	6	3	3	3	9	3	1	3	7	31
E110	3	4	3	10	3	3	6	3	3	4	10	3	3	2	8	34

E111	3	3	3	9	4	3	7	3	3	3	9	3	2	3	8	33
E112	3	3	3	9	3	3	6	3	3	3	9	3	3	2	8	32
E113	5	3	3	11	3	3	6	3	4	3	10	3	3	3	9	36
E114	3	3	3	9	4	3	7	3	3	3	9	3	2	3	8	33
E115	3	3	3	9	3	3	6	3	3	3	9	3	3	3	9	33
E116	4	4	4	12	3	4	7	4	4	4	12	4	2	2	8	39
E117	2	2	2	6	4	2	6	2	2	2	6	2	2	1	5	23
E118	1	3	1	5	1	2	3	1	1	1	3	1	1	1	3	14
E119	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22
E120	2	2	2	6	2	2	4	2	2	2	6	2	1	2	5	21
E121	3	3	3	9	3	3	6	3	3	3	9	3	3	3	9	33
E122	3	4	3	10	3	3	6	3	3	3	9	3	3	2	8	33
E123	3	3	3	9	3	3	6	3	3	3	9	3	3	3	9	33
E124	2	2	2	6	3	2	5	2	2	2	6	2	2	2	6	23
E125	2	2	2	6	2	2	4	2	4	2	8	2	2	2	6	24
E126	5	3	3	11	3	4	7	3	3	3	9	3	3	3	9	36
E127	2	2	2	6	4	2	6	2	2	2	6	2	2	2	6	24
E128	2	4	2	8	4	3	7	3	2	2	7	2	2	2	6	28
E129	2	2	2	6	2	2	4	3	2	2	7	2	2	2	6	23
E130	3	3	5	11	3	4	7	3	3	3	9	3	3	3	9	36
E131	1	1	1	3	1	1	2	4	1	1	6	1	1	1	3	14
E132	3	3	3	9	3	3	6	3	3	3	9	3	3	3	9	33
E133	2	2	2	6	4	2	6	4	3	2	9	2	2	2	6	27
E134	2	2	2	6	2	3	5	2	4	2	8	2	1	2	5	24
E135	3	3	3	9	3	3	6	5	3	3	11	3	2	3	8	34
E136	3	3	3	9	4	2	6	3	3	3	9	3	3	3	9	33
E137	3	4	4	11	4	3	7	4	4	4	12	4	1	2	7	37
E138	3	3	3	9	3	3	6	5	3	3	11	3	3	3	9	35
E139	2	2	2	6	2	2	4	2	2	2	6	2	2	2	6	22

Fuente: Elaboración propia

ANEXO 07: POSTTEST – ARQUITECTURA DE SOFTWARE APH

Tabla 26: *Encuesta Post-Test Arquitectura de software APH*

VARIABLES	V1: Arquitectura de software APH																
DIMENSIONES	Reusabilidad			SUBTOTAL	Disponibilidad			SUBTOTAL	Modificabilidad			SUBTOTAL	Seguridad			SUBTOTAL	SUMA TOTAL
PREGUNTAS	P1	P2	P3		P4	P5	P6		P7	P8	P9		P10	P11	P12		
E1	5	5	5	15	5	4	5	14	4	4	5	13	4	5	4	13	55
E2	5	5	5	15	5	5	5	15	4	5	4	13	4	4	4	12	55
E3	5	5	4	14	5	4	5	14	4	4	4	12	5	5	4	14	54
E4	4	5	5	14	5	4	5	14	4	4	5	13	4	4	4	12	53
E5	5	5	5	15	5	5	5	15	5	4	5	14	3	4	3	10	54
E6	4	4	5	13	5	5	5	15	4	5	4	13	4	5	4	13	54
E7	4	4	5	13	5	5	4	14	4	5	4	13	4	4	4	12	52
E8	4	5	5	14	5	5	4	14	4	4	4	12	4	4	4	12	52
E9	5	5	5	15	5	5	5	15	4	4	5	13	4	5	5	14	57
E10	4	5	5	14	5	4	4	13	4	4	5	13	4	5	5	14	54
E11	4	5	4	13	5	5	5	15	5	4	4	13	4	5	5	14	55
E12	5	5	4	14	5	5	5	15	5	4	4	13	4	4	4	12	54
E13	4	5	4	13	5	5	5	15	5	4	5	14	4	4	5	13	55
E14	4	5	5	14	5	5	5	15	5	4	5	14	5	5	5	15	58
E15	4	5	4	13	5	5	5	15	4	4	4	12	5	4	4	13	53
E16	4	4	5	13	5	4	4	13	4	4	4	12	4	5	5	14	52
E17	4	5	5	14	5	4	4	13	5	4	4	13	5	5	4	14	54
E18	4	4	5	13	5	4	4	13	4	4	4	12	4	5	5	14	52
E19	4	4	5	13	4	5	4	13	4	5	4	13	4	5	4	13	52
E20	4	4	4	12	4	5	4	13	4	5	4	13	4	4	5	13	51
E21	4	4	4	12	4	5	5	14	4	5	5	14	4	4	5	13	53
E22	5	5	5	15	4	4	5	13	4	5	5	14	4	4	5	13	55
E23	4	5	5	14	5	4	4	13	4	4	4	12	4	4	4	12	51
E24	5	5	5	15	5	5	5	15	4	4	4	12	5	3	4	12	54

E25	4	5	5	14	5	4	4	13	4	4	5	13	5	3	5	13	53
E26	4	5	5	14	5	4	4	13	4	4	5	13	5	5	4	14	54
E27	4	5	5	14	5	5	5	15	5	4	5	14	5	5	4	14	57
E28	4	5	5	14	5	4	4	13	5	5	4	14	5	5	4	14	55
E29	5	5	5	15	5	5	5	15	5	5	4	14	4	5	5	14	58
E30	5	5	5	15	5	4	5	14	5	5	4	14	4	5	5	14	57
E31	4	5	5	14	5	5	4	14	5	5	4	14	4	4	5	13	55
E32	4	5	5	14	4	5	5	14	5	5	5	15	5	4	4	13	56
E33	4	4	4	12	4	5	5	14	5	5	4	14	5	5	4	14	54
E34	5	5	4	14	5	5	5	15	5	5	5	15	5	5	5	15	59
E35	4	5	4	13	5	5	5	15	5	4	5	14	5	4	5	14	56
E36	4	5	4	13	4	5	5	14	4	4	5	13	5	4	4	13	53
E37	4	5	5	14	5	5	4	14	5	4	5	14	5	5	4	14	56
E38	4	5	4	13	5	5	5	15	4	4	5	13	4	5	3	12	53
E39	4	5	4	13	5	5	5	15	4	4	4	12	4	4	5	13	53
E40	4	5	4	13	5	5	5	15	4	4	4	12	4	4	4	12	52
E41	4	5	4	13	5	4	5	14	4	4	4	12	3	5	4	12	51
E42	4	5	5	14	5	5	5	15	4	5	4	13	5	4	5	14	56
E43	4	5	5	14	5	5	5	15	4	5	4	13	5	4	4	13	55
E44	4	5	5	14	5	4	5	14	4	5	4	13	5	5	4	14	55
E45	4	5	5	14	5	5	4	14	4	5	4	13	5	4	4	13	54
E46	5	4	5	14	5	5	5	15	4	5	4	13	5	4	4	13	55
E47	5	4	5	14	4	5	5	14	4	5	4	13	4	5	4	13	54
E48	5	5	5	15	4	4	4	12	4	5	4	13	4	5	4	13	53
E49	5	5	5	15	5	5	5	15	5	4	4	13	4	4	5	13	56
E50	4	4	4	12	5	5	5	15	5	4	4	13	4	4	5	13	53
E51	5	5	4	14	4	5	4	13	4	4	4	12	4	5	4	13	52
E52	4	5	5	14	5	5	5	15	5	4	4	13	5	4	4	13	55
E53	5	5	4	14	5	5	5	15	5	5	4	14	5	4	5	14	57
E54	5	5	4	14	5	5	4	14	5	5	5	15	5	4	5	14	57
E55	4	4	4	12	4	5	4	13	4	5	5	14	5	4	4	13	52
E56	5	4	5	14	4	4	4	12	4	4	5	13	4	4	4	12	51
E57	4	4	5	13	5	5	5	15	4	4	5	13	4	4	4	12	53
E58	4	4	4	12	5	4	5	14	4	4	4	12	4	5	5	14	52
E59	4	4	4	12	5	5	5	15	4	4	5	13	5	5	4	14	54
E60	4	4	4	12	5	5	5	15	4	4	4	12	5	4	4	13	52
E61	4	4	4	12	5	5	4	14	5	4	5	14	4	4	4	12	52
E62	4	4	5	13	5	5	5	15	4	5	4	13	4	3	4	11	52
E63	5	4	5	14	5	5	5	15	4	5	4	13	4	4	4	12	54
E64	5	4	4	13	5	5	5	15	4	5	4	13	4	5	4	13	54
E65	4	5	4	13	4	4	5	13	4	4	4	12	4	4	5	13	51
E66	5	5	4	14	4	5	5	14	4	4	4	12	5	4	5	14	54
E67	5	5	4	14	4	5	5	14	5	5	4	14	4	4	4	12	54

E68	5	4	4	13	4	5	5	14	5	5	4	14	4	4	4	12	53
E69	4	5	5	14	5	5	5	15	5	5	5	15	4	4	4	12	56
E70	4	5	5	14	5	5	5	15	4	5	5	14	4	5	4	13	56
E71	5	5	5	15	5	5	5	15	4	5	5	14	5	5	5	15	59
E72	5	5	4	14	5	4	5	14	4	5	5	14	4	4	5	13	55
E73	4	5	4	13	4	4	5	13	5	5	4	14	4	4	5	13	53
E74	4	5	4	13	4	4	5	13	5	5	4	14	4	5	5	14	54
E75	4	5	4	13	4	5	5	14	5	4	5	14	5	4	5	14	55
E76	4	5	5	14	4	4	5	13	4	4	5	13	3	5	4	12	52
E77	4	5	4	13	5	4	5	14	4	4	4	12	5	5	4	14	53
E78	4	5	4	13	5	5	5	15	5	4	5	14	5	5	4	14	56
E79	5	5	4	14	5	5	5	15	5	4	5	14	5	5	5	15	58
E80	5	5	4	14	5	5	5	15	5	4	5	14	5	5	5	15	58
E81	5	5	4	14	5	4	4	13	4	5	4	13	5	5	4	14	54
E82	5	5	4	14	5	4	4	13	4	4	4	12	5	5	4	14	53
E83	5	5	4	14	5	4	5	14	4	5	4	13	5	5	4	14	55
E84	5	5	4	14	5	5	5	15	5	5	4	14	5	5	5	15	58
E85	5	5	4	14	5	4	5	14	5	5	4	14	5	5	5	15	57
E86	5	4	5	14	4	4	5	13	5	5	4	14	5	4	5	14	55
E87	5	4	5	14	4	4	5	13	5	5	5	15	5	4	4	13	55
E88	5	4	5	14	5	4	5	14	4	5	5	14	5	4	5	14	56
E89	5	4	4	13	5	4	4	13	4	4	5	13	5	5	4	14	53
E90	5	4	4	13	5	4	5	14	4	4	5	13	5	5	4	14	54
E91	5	4	4	13	5	5	5	15	4	5	5	14	4	4	5	13	55
E92	5	4	4	13	5	5	5	15	4	5	5	14	4	4	4	12	54
E93	4	5	5	14	5	5	4	14	4	5	5	14	4	5	5	14	56
E94	5	5	5	15	5	5	5	15	4	5	4	13	4	5	5	14	57
E95	5	4	5	14	5	4	4	13	4	5	5	14	4	5	4	13	54
E96	5	5	5	15	5	4	5	14	4	5	5	14	4	4	4	12	55
E97	5	5	5	15	5	5	5	15	5	4	5	14	4	4	4	12	56
E98	5	5	5	15	5	5	4	14	5	4	5	14	4	5	4	13	56
E99	4	5	5	14	4	5	4	13	5	5	5	15	5	5	4	14	56
E100	4	5	4	13	4	4	4	12	5	5	4	14	5	4	4	13	52
E101	4	5	4	13	4	4	5	13	4	5	4	13	5	5	5	15	54
E102	4	5	4	13	4	5	5	14	4	4	4	12	5	5	5	15	54
E103	4	5	4	13	4	5	5	14	4	4	5	13	5	4	5	14	54
E104	4	5	4	13	4	4	5	13	4	5	4	13	5	5	5	15	54
E105	4	5	4	13	4	4	5	13	4	5	5	14	5	5	4	14	54
E106	4	5	5	14	4	4	5	13	4	4	5	13	5	5	4	14	54
E107	4	5	5	14	4	4	4	12	4	4	4	12	4	4	4	12	50
E108	5	4	5	14	5	4	4	13	4	4	4	12	4	4	4	12	51
E109	5	4	5	14	4	4	5	13	5	4	4	13	4	4	4	12	52
E110	5	4	5	14	5	4	5	14	4	5	4	13	4	5	4	13	54

E111	5	5	5	15	4	4	5	13	5	5	4	14	4	4	4	12	54
E112	4	5	4	13	4	5	4	13	5	5	4	14	4	4	4	12	52
E113	5	4	4	13	4	5	5	14	5	5	4	14	4	3	4	11	52
E114	5	5	4	14	4	4	5	13	5	5	4	14	4	5	5	14	55
E115	5	5	4	14	4	4	5	13	5	5	5	15	4	4	5	13	55
E116	5	4	5	14	4	5	5	14	5	4	4	13	4	4	4	12	53
E117	5	4	5	14	4	5	5	14	5	4	4	13	4	3	4	11	52
E118	5	4	5	14	4	5	5	14	5	4	4	13	4	4	5	13	54
E119	5	4	5	14	4	5	5	14	4	5	4	13	4	5	4	13	54
E120	5	5	5	15	5	5	4	14	4	4	4	12	4	5	4	13	54
E121	5	5	4	14	5	4	5	14	5	4	4	13	5	4	4	13	54
E122	5	5	4	14	5	5	4	14	5	4	4	13	5	4	4	13	54
E123	5	5	4	14	5	4	4	13	5	4	4	13	5	4	4	13	53
E124	4	5	5	14	5	5	4	14	5	4	4	13	5	5	4	14	55
E125	5	5	4	14	5	5	5	15	5	4	5	14	5	5	5	15	58
E126	5	5	4	14	4	5	4	13	5	5	4	14	5	5	5	15	56
E127	5	5	4	14	4	5	4	13	5	5	5	15	5	4	3	12	54
E128	5	5	4	14	4	4	5	13	5	5	5	15	5	4	5	14	56
E129	4	5	5	14	4	4	5	13	5	5	4	14	4	5	5	14	55
E130	5	4	5	14	4	5	5	14	4	5	5	14	4	5	4	13	55
E131	5	4	5	14	4	5	5	14	4	5	4	13	5	4	4	13	54
E132	5	4	5	14	5	4	5	14	5	5	4	14	4	5	5	14	56
E133	5	5	5	15	5	5	5	15	5	5	4	14	5	5	5	15	59
E134	5	4	4	13	5	5	4	14	4	4	5	13	4	4	5	13	53
E135	5	4	4	13	5	5	5	15	4	4	5	13	4	4	4	12	53
E136	5	5	5	15	5	5	5	15	5	5	4	14	4	5	4	13	57
E137	5	4	4	13	5	5	5	15	5	5	5	15	4	5	5	14	57
E138	5	4	5	14	5	5	5	15	5	5	5	15	4	5	5	14	58
E139	5	4	5	14	5	5	5	15	5	4	4	13	5	5	4	14	56

Fuente: Elaboración propia