MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE MASTER

DIMENSIONAMIENTO, DESARROLLO, DESPLIEGUE Y ANÁLISIS DE RENDIMIENTO DE UNA API GRAPHQL CON UNA ARQUITECTURA DE MICROSERVICIOS Y CLUSTER DE BASE DE DATOS ALTAMENTE ESCALABLE EN AWS PARA PRODUCCIÓN

profesiolan

Empresa apoyada por:

Estudiante: Hospital Gómez, Ander

Director: Prieto Agujeta, Gorka

Curso: 2023-2024

Fecha: Bilbao, a 22 de febrero del 2024

Intencionadamente en blanco.

Abstract

This Final Master's Project consists of the sizing, development, deployment, and subsequent performance analysis of a unified API (Application Programming Interface) for all clients of the Profesiolan product.

The work is framed in a real case of need for a technological Startup that, due to accelerated growth, must migrate all of its MVP (Minimum Viable Product) to a new product that implements scalability, high availability, excellent quality of service, better resilience to failures, security, the possibility of providing service in different geographical areas and, above all, a new paradigm of modular architecture that allows CI/CD.

As it is a challenge that solves a real problem, special attention is paid to the analysis of alternatives, with two critical points being the decision of the technological stack, the architecture paradigm, and the use of an IaaS or Cloud provider compared to a traditional alternative deployment. For the aforementioned analysis of alternatives, in addition to purely technical decision and weighting parameters, economic and financial parameters have also been included, preceded by a series of specifications directly imposed by the company's management.

Due to the qualitative leap in technological level proposed by Cloud Computing providers in the context of the continuous and accelerated development of cloud technologies, this Master's Thesis uses many of the architectural paradigms that make up the current state of the art in the SaaS sector, including microservices, Lambda functions, GraphQL, and multi A-Z deployment among others.

Laburpena

Master Amaierako Proiektu hau Profesiolan produktuaren bezero guztientzat API bateratuaren (Aplikazioen Programazio Interfazea) baten dimentsionatzean, garapenean, hedatzean eta ondorengo errendimenduaren analisian datza.

Lana, hazkunde bizkortuaren ondorioz, bere MVP (Gutxieneko Produktu Bideragarria) eskalagarritasuna, erabilgarritasun handia, zerbitzu-kalitate bikaina, hutsegiteen aurrean erresilientzia hobea, segurtasuna, eremu geografiko desberdinetan zerbitzua emateko aukera eta, batez ere, CI/CD arkitektura modulararen paradigma berri batera bideratzen dituen produktu berri batera migratu behar duen Startup teknologiko baten benetako kasu batean kokatzen da lana.

Benetako arazo bat konpontzen duen erronka denez, arreta berezia jartzen zaio alternatiben analisiari, bi puntu kritiko izanik pila teknologikoaren erabakia, arkitektura paradigma eta IaaS edo Cloud hornitzaile baten erabilera ohiko inplementazio alternatibo batekin alderatuta. Aipatutako alternatiben azterketarako, erabaki eta ponderazio-parametro tekniko hutsez gain, parametro ekonomiko eta finantzarioak ere sartu dira, enpresaren zuzendaritzak zuzenean ezarritako zehaztapenen aurretik.

Cloud Computing-en hornitzaileek hodeiko teknologien etengabeko eta azkarreko garapenaren testuinguruan proposatzen duten maila teknologikoko jauzi kualitatiboa dela eta, Master Amaierako Lan honek SaaS sektorearen egungo egoera osatzen duten paradigma arkitektoniko asko erabiltzen ditu. Testuinguru honetan, mikrozerbitzuez, Lambda funtzioez, GraphQL eta A-Z anitzeko inplementazioaz hitz egin behar da, besteak beste.

Resumen

Este Trabajo Fin de Máster consiste en el dimensionamiento, desarrollo, despliegue y posterior análisis de rendimiento de una API (Application Programming Interface) unificada para todos los clientes del producto Profesiolan.

El trabajo está enmarcado en un caso real de necesidad de una Startup tecnológica que, por un crecimiento acelerado, debe migrar la totalidad de su MVP (Producto Mínimo Viable) a un nuevo producto que implemente escalabilidad, alta disponibilidad, una excelente calidad de servicio, mejor resiliencia ante fallos, seguridad, posibilidad de dar servicio en diferentes zonas geográficas y, sobre todo, un nuevo paradigma de arquitectura modular que permita integración y despliegue continuo (CI/CD).

Al tratarse de un reto que resuelve una problemática real, se presta especial atención al análisis de alternativas, siendo sendos puntos críticos la decisión de la pila tecnológica, el paradigma de arquitectura, y la valoración de uso de un sistema de computación en la nube basado en un sistema de infraestructura como servicio frente a un despliegue alternativo tradicional. Para el mencionado análisis de alternativas, además de parámetros de decisión y ponderación de carácter púramente técnicos, también se han incluido de carácter económico y financiero, precedidos por una serie de especificaciones directamente impuestas por la dirección de la empresa.

Debido al salto cualitativo de nivel tecnológico que proponen los proveedores de computación en la nube o *Cloud Computing* en el contexto del continuo y acelerado desarrollo de las tecnologías en la nube, este Trabajo Fin de Máster emplea muchos de los paradigmas de arquitectura que conforman, a día de hoy, el estado del arte en el sector del Software Como Servicio o *SaaS*. En este contexto, se debe hablar de microservicios, funciones Lambda, GraphQL y multi A-Z deployment, entre otros.

Palabras clave

Cloud Computing, SaaS B2B, AWS, API, Java, GraphQL, MySQL.

Contenido

Lista de tablas

Lista de ilustraciones

Lista de acrónimos

API	Application Programming Interface
AWS	Amazon Web Services
B2B	Business to Business
B2C	Business to Consumer
CI/CD	Continuous Integration & Continuous Development
ERP	Enterprise Resource Planning, o Planificación de Recursos Empresariales
IA	Inteligencia Artificial
IaaS	Infraestructure as a Service
IPO	Initial Public Offering, o Oferta Pública Inicial
MVP	Minimum Viable Product o Producto Mínimo Viable
PaaS	Platform as a Service
PoC	Proof of Concept o Prueba de Concepto
QoS	Quality of Service o Calidad de Servicio
RDS	Relational Database Service
ROI	Return of Investment o Retorno de Inversión
SaaS	Software as a Service
TFM	Trabajo Fin de Máster
VC	Venture Capital, o Capital Riesgo

1. Memoria

1.1. Introducción

El presente Trabajo Fin de Máster (TFM) se ha desarrollado en el marco de desarrollo tecnológico de producto de la Startup tecnológica Profesiolan, empresa de la cual el alumno Ander Hospital Gómez es fundador, socio, y trabajador.

El alumno Ander Hospital es el encargado y propietario de toda la tecnología realizada por la empresa, que ya cuenta con 2.200 empresas clientes, y más de 200.000 visitas mensuales. En el año 2020, el alumno diseñó y desplegó el primer producto de la compañía. Un producto mínimo viable (MVP) basado en Wordpress que complementó con código hasta conseguir el producto que a día de hoy cuenta con miles de sesiones diarias.

No obstante, debido al crecimiento de la tasa de clientes, el número de sesiones servidas simultáneamente y el aumento sin precedentes de las tablas de la base de datos (sobrepasando un stock que acumula las 120.000 imágenes), la empresa Profesiolan planteó la actualización total del producto en septiembre del año 2022 como un proyecto de alta prioridad.

Un refactoring absoluto del producto cuyo propósito fuese la creación de una arquitectura modular, desarrollada con mejores prácticas, con un stack de servicios y tecnología en la vanguardia del *SaaS* (Software as a Service), que fuese capaz de escalar a la velocidad de las necesidades de la empresa, y que no requiriese de otro refactoring total durante toda la vida de la misma. Al contrario, el producto busca la evolución continua mediante la implementación de nuevas funcionalidades o el reemplazo de módulos (o microservicios), en un canal de integración y despliegue continuo (*CI/CD)*.

Este Trabajo Fin de Máster presenta todo el trabajo realizado por el alumno en materia de dimensionamiento, desarrollo, despliegue y posterior análisis de rendimiento de una API (*Application Programming Interface*) unificada para todos los clientes del producto Profesiolan.

A lo largo del presente documento se presentan todas las tecnologías valoradas y en última instancia utilizadas que han contribuido, durante las más de 1.400 horas que han conformado el trabajo, a la implementación de un producto escalable y de alta disponibilidad, con una calidad de servicio reconocida como estándar de mercado, resiliente ante fallos, seguro, modular y replicable para dar servicio en otras zonas geográficas.

1.2. Contexto

Profesiolan es un SaaS B2B (*Business-to-business*) implementable en el departamento de compras de cualquier empresa [1]. El proyecto nace para que la segunda mano sea una alternativa real en las empresas. Ahora mismo, Profesiolan es el mercado transversal exclusivamente profesional de activos fijos de segunda mano más grande en España, con una oferta de más de 10.000 activos fijos de ocasión [2].

Profesiolan es una empresa de impacto. Al igual que reciclar y reducir, reusar es una de las claves para el hito de las emisiones globales netas *net-zero* [3]. Sin ir más lejos, tras la rotación de todos los activos en Profesiolan, ahorramos hasta 50.000 toneladas de CO2 en la producción de nuevas unidades.

En lo que respecta al producto, Profesiolan aspira a crear el mejor producto autoservicio y sin fricciones para la compra de activos fijos de ocasión en todo el mundo. Esto se fundamenta en cuatro pilares principales [4].

- La importancia de la máxima seguridad y la aplicación de las buenas prácticas en la operativa entre compradores y vendedores cuyo modelo de negocio no sea el de la compra y venta de activos fijos de ocasión. Esto se realiza mediante doble filtrado, exclusividad profesional, integraciones de fotoverificación por inteligencia artificial (IA), e integración de métodos de peritaje también por IA, entre otros.
- La importancia de la especificación con decenas de campos de metadatos únicos para cada una de las más de 1.000 categorías de activos fijos.
- La importancia de que el proceso de compra sea totalmente autoservicio, que no existan fricciones, y que tanto el comprador como el vendedor puedan terminar las operaciones en minutos, sin preocupaciones de pensar en logística o labores de venta.
- La importancia del entorno visual y funcional. Un sello de limpieza y un entorno intuitivo y moderno.

En lo que respecta al contexto tecnológico, el proyecto se ejecuta en un momento clave de evolución en todos los servicios que rodean el sector del Cloud Computing. Un momento histórico de madurez y consolidación de herramientas que son clave para el desarrollo de nuevas soluciones innovadoras y tecnológicas. Una época de máxima competitividad en el SaaS, donde el canibalismo entre productos

competidores en los mismos mercados se decanta por la calidad del servicio, el abanico de funcionalidades y la velocidad de adecuación a las necesidades del mercado. Por todo ello, es el desarrollo de producto y la inversión en innovación en la arquitectura de cada una de las soluciones la partida principal en la mayoría de empresas SaaS. Como se puede apreciar en la *Figura 1: Comparación de la inversión en producto de una muestra relevante de empresas SaaS en los dos años previos al IPO*, la relación entre el gasto en producto respecto del gasto en ventas a dos años antes de la ejecución de Oferta Pública de Venta, supera en la mayoría de casos el 30%.

Company	Ticker	Revenue for IPO Year	Revenue 1 year prior	Revenue 2 years prior	Growth for IPO year	Product spend for IPO year	Product spend 1 year prior	Product spend 2 years prior	Product/ Sales for IPO year	Product/ Sales 1 year prior	Product/ Sales 2 years prior
SaaS											
Dropbox	DBX	$1,106,800	$844,800	$603,800	31%	$380,300	$289,700	$201,600	34%	34%	33%
Nutanix	NTNX	$444,928	$241,432	$127,127	84%	$116,400	$73,510	$38,037	26%	30%	30%
Docusign	DOCU	$381,459	$250,481	---	52%	$89,652	$62,225	---	24%	25%	---
Secureworks	SCWX	$339,522	$262,130	$205,830	30%	$49,747	$32,053	$26,993	15%	12%	13%
Atlassian Corporation	TEAM	$319,521	$215,109	$148,512	49%	$140,853	$78,640	$57,301	44%	37%	39%
Cloudera	CLDR	$261,026	$166,048	---	57%	$102,309	$99,314	---	39%	60%	---
Lifelock	LOCK	$193,949	$162,279	$131,368	20%	$17,749	$21,338	$19,925	9%	13%	15%
Mulesoft	MULE	$187,747	$110,252	$57,617	70%	$32,862	$24,725	$17,046	18%	22%	30%
IntraLinks Holdings, Inc.	IL	$184,332	$140,699	$143,401	31%	$17,593	$14,222	$14,847	10%	10%	10%
Zuora	ZUO	$167,926	$113,008	$92,184	49%	$38,639	$26,355	$20,485	23%	23%	22%
Alarm.com	ALRM	$167,312	$130,222	$96,475	28%	$23,193	$13,085	$8,944	14%	10%	9%
Twilio	TWLO	$166,919	$88,846	$49,920	88%	$42,559	$21,824	$13,959	25%	25%	28%
FireEye	FEYE	$161,552	$83,316	$33,658	94%	$66,036	$16,522	$7,275	41%	20%	22%
RingCentral, Inc.	RNG	$160,505	$114,525	$78,877	40%	$33,399	$24,450	$12,199	21%	21%	15%
Okta	OKTA	$160,326	$85,907	$41,010	87%	$38,659	$28,761	$18,370	24%	33%	45%
Paylocity Holding	PCTY	$152,698	$108,687	$77,294	40%	$19,864	$10,355	$6,825	13%	10%	9%
RealPage, Inc.	RP	$140,902	$112,568	$83,581	25%	$27,446	$28,806	$21,708	19%	26%	26%
WorkDay, Inc.	WDAY	$134,427	$68,055	$25,245	98%	$62,014	$39,175	$30,045	46%	58%	119%
Coupa Software	COUP	$133,775	$83,678	$50,845	60%	$30,262	$22,767	$11,887	23%	27%	23%
Appian Corporation	APPN	$132,923	$111,204	$88,996	20%	$22,994	$16,750	$13,488	17%	15%	15%
Veeva Systems Inc.	VEEV	$129,548	$61,262	$29,129	111%	$14,638	$7,750	$3,367	11%	13%	12%
Apptio	APTI	$129,251	$106,615	$73,768	21%	$30,553	$23,099	$17,804	24%	22%	24%
Zendesk, Inc.	ZEN	$127,049	$72,045	$38,228	76%	$36,403	$15,288	$14,816	29%	21%	39%
Zscaler	ZS	$125,717	$80,325	$53,707	57%	$33,561	$20,940	$15,034	27%	26%	28%
Yext	YEXT	$124,261	$89,724	$60,002	38%	$19,316	$16,201	$11,945	16%	18%	20%
Box, Inc.	BOX	$124,192	$58,797	$21,084	111%	$45,967	$28,996	$14,396	37%	49%	68%
Blackline	BL	$123,123	$83,607	$51,677	47%	$21,125	$18,216	$9,705	17%	22%	19%
Splunk, Inc.	SPLK	$120,960	$66,245	$35,000	83%	$23,561	$14,025	$8,479	19%	21%	24%
Blackbaud	BLKB	$118,093	$105,229	$90,034	12%	$15,516	$14,385	$14,755	13%	14%	16%
HubSpot, Inc.	HUBS	$115,876	$77,634	$51,604	49%	$25,638	$15,018	$10,585	22%	19%	21%
SmartSheet	SMAR	$111,253	$66,964	$40,751	66%	$37,590	$19,640	$12,900	34%	29%	32%
Paycom Software	PAYC	$107,601	$76,810	$57,206	40%	$2,146	$1,632	$1,225	2%	2%	2%
Bazaarvoice, Inc.	BV	$106,136	$64,482	$38,648	65%	$20,789	$10,847	$5,828	20%	17%	15%
MobileIron, Inc.	MOBL	$105,574	$40,890	$13,856	158%	$36,400	$23,773	$8,052	34%	58%	58%
Benefitfocus, Inc.	BNFT	$104,752	$81,739	$68,783	28%	$23,532	$14,621	$9,120	22%	18%	13%

crunchbase

Figura 1: Comparación de la inversión en producto de una muestra relevante de empresas SaaS en los dos años previos al IPO [5].

1.3. Objetivos y alcance del trabajo

Al tratarse de un proyecto ligado al lanzamiento de un producto integral, el listado de objetivos es extenso, incluyendo los hitos impuestos por cada una de las partes interesadas (stakeholders), además de aquellos impuestos desde la propia dirección de la empresa.

En este apartado se recogen y definen todos ellos, sin ningún motivo de prioridad en la ordenación.

1.3.1. Objetivos de la empresa

1.3.1.1. Mejora del tiempo de servicio o uptime

El Uptime se define como el porcentaje del tiempo (en una ventana anual) en el cual un cliente tiene disponibilidad del servicio o los servicios ofrecidos por la empresa [6].

Debido a una falta de dimensionamiento en el diseño de arquitectura de la solución inicial, el servicio principal de la empresa ha llegado a verse comprometido en varias ocasiones, interrumpiendo el servicio nominal en lapsos menores de 5 minutos, pero en una decena de ocasiones en el transcurso de un año. Esto ha dejado un tiempo de Uptime no mayor del 99,3% en el primer semestre del año 2023.

En el mercado del Cloud Computing, es precisamente el SaaS la vertical en la que menos afecta este parámetro; la mayoría de empresas del sector afirman una tasa de servicio de entre un 99,00% y un 99,9% [7], frente al 99,95% y superiores reflejados en los SLAs de algunos servicios IaaS y PaaS de AWS [8].

Si bien no se trata de una cifra alarmante debido a la baja densidad de sesiones, la mejora de este parámetro hasta un valor no menor del 99,9% debe ser una necesidad en el corto plazo. Para determinar la prioridad de la mejora de este parámetro, se debe analizar la densidad de sesiones mencionada, y relacionarla con el tiempo en el que se asume no existe servicio en el tiempo de un día. Para este estudio, se toman en cuenta los datos analizados por la herramienta Google Analytics, y se agrega el tiempo consumido en el servicio de las 250.000 páginas vistas por cerca de 30.000 sesiones en el último mes, con una duración media de 3 minutos por sesión. Analizando el caso peor en el que no existe solapamiento de las sesiones, esto da un total de cerca de 3.000 minutos de

servicio diario, o 50 horas. Normalizando el parámetro de solapamiento a las horas de un día, resulta en 2,08 sesiones por minuto.

Por la otra parte, el análisis del tiempo de Downtime (parámetro contrario del Uptime), muestra que existe interrupción en el 0,70% del tiempo, o cerca de 10 minutos diarios.

Este par de métricas debe analizarse desde negocio, correlacionándolo con otros parámetros de interés como el coste de una sesión, el valor de una sesión, o incluso el CAC.

Tras este análisis, se ha establecido que la mejora del Uptime debe ser una prioridad debido a que, por escala, existe un crecimiento intermensual nominal en el número de las visitas del servicio web.

1.3.1.2. Mejora del QoS

En este producto, el QoS o Quality Of Service (Calidad de Servicio), es más propenso a depender de un *frontend* rápido, que renderiza y carga velózmente, que propone una navegación intuitiva y sencilla, que de una respuesta rápida del servidor en las tareas de la API.

Para el continuo estudio de la salud del producto, la totalidad de los parámetros que engloban la Calidad del Servicio, seguidos muy de cerca desde la dirección de la empresa son los siguientes.

1. Largest Contentful Paint (LCP)
2. First Input Delay (FID)
3. Cumulative Layout Shift (CLS)
4. First Contentful Paint (FCP)
5. Interaction to Next Paint (INP)
6. Time to First Byte (TTFB)
7. Tiempo de Respuesta del servidor para la resolución de una consulta tipo búsqueda

En lo que respecta a los parámetros dependientes de la API, tan solo se incluyen el LCP y el tiempo de respuesta del servidor en consultas de tipo búsqueda. No obstante, el LCP tan solo es dependiente de la API en determinadas ocasiones en las que la imagen a renderizar tiene que ser descargada desde la misma, y el tiempo de respuesta del servidor en consultas de búsqueda es

muchas veces compensado con el tiempo de renderizado de la página, cuando la respuesta no es directamente cacheada.

Se concluye por tanto que, mientras que el QoS es siempre un parámetro crítico en un SaaS y que debe ser optimizado para ofrecer una buena percepción del servicio a los clientes, en este caso concreto de producto la optimización de la API no tiene un impacto tan relevante en esa percepción de calidad.

1.3.1.3. Mejora de Features

La mejora de Features se refiere a la capacidad que tiene el equipo de producto y desarrollo de una empresa de desplegar nuevas funcionalidades o actualizaciones de producto con asiduidad, que resuelvan nuevas necesidades de mercado en el tiempo.

Desde la agilidad en el desarrollo y despliegue, hasta la capacidad de evolucionar y pivotar, este punto es esencial en el producto de cualquier Startup.

Por tanto, una de las necesidades principales que motiva el desarrollo y ejecución de este proyecto es la capacidad de establecer un nuevo canal de desarrollo, en un entorno de CI/CD hosteado en AWS, que optimiza la capacidad de la empresa de desplegar nuevas funcionalidades.

1.3.1.4. Inversión Mínima y tiempo de desarrollo mínimo

La motivación de cualquier empresa está basada en la reducción u optimización de costes y en la mejora de la facturación. Asímismo, la ejecución de este proyecto debe conseguir el éxito con una inversión reducida.

Además, el proyecto debe alcanzar el éxito en el tiempo estipulado para que la empresa no sufra el desgaste asociado al alargamiento de un proyecto crítico.

1.3.2. Objetivos técnicos de la API

El objetivo principal de la API es poder resolver todas las funcionalidades y servicios ofertados en el producto MVP, desde una nueva arquitectura totalmente renovada, modular, de alta disponibilidad, y totalmente escalable, con unos tiempos de respuesta que no superen los 200 ms en valor promedio por 1.000 API calls.

Tanto la arquitectura como todos los componentes que permiten esto son definidos en el *Apartado 1.7: Descripción de la solución realizada*, y extendidos y profundizados en el *Anexo 1: Solución tecnológica realizada*. También se subraya que en este apartado no se profundiza en las motivaciones tecnológicas que han impulsado cada una de las decisiones como objetivos, temas desarrollados en el *Apartado 1.5: Estado del arte* y el *Apartado 1.6: Análisis de alternativas*.

A continuación, se desglosan los objetivos de menor nivel de abstracción que posibilitan y contribuyen al mencionado objetivo principal.

1.3.2.1. Objetivos de la Base de Datos

1. Multi AZ Deployment: De cara a ofrecer una escalabilidad a nivel de expansión territorial y poder extender el servicio a diferentes paises, la base de datos debe tener la capacidad de réplica en diferentes regiones.
2. Escalabilidad Horizontal: Definido como la capacidad de disponer de un número determinado y escalable de nodos operando como uno solo, sindicados en lo que se conoce como cluster, la escalabilidad horizontal es necesaria para dar agnosticismo a la solución tecnológica. De este modo, se consigue el mejor diseño de arquitectura en capas, en la que la capa de negocio, donde están situados los módulos o servicios, solo tiene un único punto de acceso a toda la capacidad computacional del cluster de la Base de Datos.
3. Escalabilidad Vertical: De cara a ofrecer una escalabilidad para responder a un tráfico cada vez mayor, se debe tener en todo momento la capacidad de migrar todos los nodos de la arquitectura o del cluster a equipos hardware más potentes cuando se desee, incrementando no solo la RAM Virtual disponible, si no también otros parámetros como la tecnología, y el ancho de banda real disponible.

4. Dynamic Allocation: De cara a un plan estratégico de reducción de costes en el producto, en la medida de lo posible, la base de datos debería proveer algún tipo de asignación dinámica de recursos. Este concepto está totalmente ligado al concepto de escalabilidad horizontal, y a su vez extiende su funcionalidad, ya que propone controlar el número de Nodos participantes o sindicados en el Cluster en función del número de conexiones establecidas con la base de datos en un mismo momento y otros parámetros clave de gestión de redes como hora y fecha.
5. Mínimo 50.000 SELECTs por segundo: Asumiendo un buen diseño de la Base de Datos, con el uso de Índices, la correcta elección de los tipos de datos, y realizando una buena política de dependencias entre las tablas, el parámetro diferencial para maximizar la calidad del servicio es la capacidad del motor. En este caso, se ha establecido una capacidad máxima no menor de 50.000 SELECTs por segundo.
6. Mínimo 5.000 UPDATEs por segundo: Debido a la naturaleza del servicio, la capacidad de servir tareas de tipo UPDATE no es tan crítica como la de servir solicitudes de búsqueda. No obstante, es necesario tener una potencia que ofrezca resiliencia y confiabilidad.

1.3.2.2. Objetivos de la API

1. GraphQL sobre Rest: No se valora en este apartado las motivaciones tecnológicas que han impulsado esta decisión, dado que este análisis se ha realizado en los apartados 1.5: Estado del arte y 1.6: Análisis de alternativas. No obstante, la decisión de tomar GraphQL como el punto de partida y base para la construcción de la API es un objetivo principal.
2. Escalabilidad absoluta para volúmenes de crecimiento prudentes: Un término aparentemente no objetivo, pero sin duda alcanzable mediante el uso de determinadas tecnologías, y dependiendo en proveedores PaaS como AWS. Se le denomina escalabilidad absoluta a la capacidad de resolver tantas solicitudes como se desee, en un rango que se considera prudente considerando el crecimiento esperado y unos márgenes lógicos, siempre limitado a un parámetro por decisiones de seguridad y ajustado en función de la naturaleza del negocio, sin interrupción del servicio. Para ello, se hace uso de conceptos tecnológicos avanzados de menor nivel de abstracción desarrollados en el *Apartado 1.3.2.1: Objetivos de la Base de Datos* como la escalabilidad horizontal, escalabilidad vertical, y la asignación dinámica de recursos.

3. Implementación de seguridad en 3 fases: La que es, por regla general, la principal preocupación de cualquier producto API. De forma resumida, una API, entendida en el contexto de proveedor de lógica de negocio en productos SaaS, es un punto de acceso público en la red, que resuelve un número determinado de consultas y solicitudes con un formato determinado [9]. Mientras que existen solicitudes protegidas por mecanismos de autenticación y autorización que pueden ser rápidamente rechazadas si no cumplen con la determinada Key o clave (en el caso de este producto JWT), existen otras muchas que no requieren de tal clave, como el registro o la propia búsqueda de productos. Una API confiable es una API que, entre otras cosas:
    a. Identifica, impide, loguea y alerta de los ataques de fuerza bruta.
    b. Realiza siempre una correcta y optimizada gestión de las sesiones.
    c. Previene de potenciales ataques de inyección.
    d. En general, trabaja activamente para defenderse de los denominados Security Top 10 de la OWASP (Open Web Application Security Project) [10].
4. Implementación de Caché: Definida como una pequeña memoria volatil y que actúa como una capa de acceso a datos adyacente a la base de datos [11], es una capacidad fundamental, capaz de reducir en más de la mitad del tiempo operaciones de búsqueda recurrentes en un periodo de tiempo.

1.3.2.3. Objetivos del Core o capa de Negocio

1. Modularidad en el diseño: Un diseño modular permite la definición, agrupación y separación de diferentes bloques que completan la totalidad del producto. Esto permite el desarrollo individual y posterior fase de testeo individual de cada uno de los bloques por separado, reduciendo el tiempo de desarrollo y la probabilidad de generar incompatibilidades o fallos [12].
2. Separación de dependencias por usos de negocio: La realización de un diseño modular maximiza la posibilidad de agrupar conjuntos de funciones o métodos que conforman un mismo grupo lógico de capacidades de uso de negocio. Esto es positivo por razones de separación de las dependencias de los datos, pero también afecta positivamente a la optimización de los tiempos de respuesta de la API, normalmente al poder disponer de

aplicaciones más pequeñas que requieren de un acceso localizado y particular de la Base de Datos.

3. Tiempo de respuesta bajo: En esencia, no contribuir en exceso a incrementar el tiempo de respuesta es siempre uno de los objetivos clave de un sistema en cascada o encadenado. Además, entre la capa de la API (entendiéndose únicamente como la labor de parseo de las peticiones y no contabilizando tiempos de transporte por la red), la capa CORE y la capa de la Base de Datos, es la segunda la que contribuye en mayor medida al tiempo de respuesta global del sistema.
4. Transaccionalidad de operaciones: Un objetivo obligatorio de cualquier aplicación o sistema es la transaccionalidad de las operaciones y, aunque se puede englobar dentro de las denominadas buenas prácticas de desarrollo, es importante reflejarlo como un reto y un objetivo a perseguir cuando se trata de arquitecturas modulares o microservicios [13]. De esta forma, se evita que se ejecuten partes de una transacción sin haber sido esta completada en su totalidad. La transaccionalidad, mientras es una funcionalidad aparentemente no complicada en entornos Java con Java Persistence API - Hibernate, y en una arquitectura monolítica, en la migración al entorno de los microservicios, esto se vuelve un reto tecnológico en mayúsculas. Un ejemplo clásico y utilizado ampliamente es el de una aplicación bancaria en la que el crédito y el débito conforman, per sé, microservicios distintos. Sin embargo, en caso de ocurrir un problema en la orquestación del microservicio del débito, en un diseño en el que no se puede garantizar siempre la transaccionalidad, podría darse el caso de ejecución única del microservicio de crédito, originando un fallo crítico en el producto.
5. Granularidad en las excepciones: Para facilitar el desarrollo de los productos clientes, ya sean las aplicaciones móviles como la aplicación web, como la intranet de administración, la API debe siempre proveer de una gran granularidad en la devolución de las excepciones. Un nivel de resolución que extienda por completo los códigos de error HTTP y que, gracias a la arquitectura GraphQL, es una realidad [14].
6. Optimización con las capas adyacentes: En una arquitectura compleja como la desarrollada en el presente proyecto, donde la capa de negocio está dispuesta de forma modular por diferentes grupos de servicios, todos ellos con una única entrada y una única salida identificada, la optimización en el diseño de estas interfaces es crítico. En este subapartado se pueden definir políticas ante error, políticas ante no respuesta en determinadas ventanas de tiempo, y políticas de minimización de la información compartida.

1.4. Beneficios

En este apartado se valoran los diferentes beneficios aportados por la realización del proyecto. Desde los beneficios púramente técnicos hasta los beneficios económicos, pasando por los beneficios para los trabajadores de la empresa y los clientes del servicio.

1.4.1. Beneficios técnicos

1.4.1.1. Creación de un MVP de la API de producto

El principal beneficio propuesto por el proyecto es el de la realización de un MVP de la API del producto Profesiolan. De esta forma, la empresa incluye un activo de valor a su hacienda. Un valor que, más allá de la importancia a nivel negocio, es un hito clave en el proceso de mejora del producto actual.

1.4.2. Beneficios para trabajadores

1.4.2.1. Mejora de la infraestructura de desarrollo centralizada y con buenas prácticas

Como se ha mencionado en los objetivos, el actual flujo de desarrollo de producto no está optimizado, es desagregado, y debe ser actualizado en el corto plazo. El presente Trabajo Fin de Máster ha supuesto un beneficio claro, al presentar un diseño optimizado del entorno de trabajo sobre producto. En concreto, se ha realizado un extensivo trabajo en la preparación y puesta a punto de un entorno privado en AWS, con un estricto protocolo a seguir para contribuir al desarrollo de nuevas funcionalidades, con sistemas de versiones, con posibilidades de rollbacks, con la capacidad de firmar cada una de las actualizaciones de producto y un largo etcétera.

1.4.2.2. Mejora de la gestión de roles y credenciales en el entorno de desarrollo de producto con buenas prácticas AWS

Al hilo del *Beneficio 1.4.2.1: Mejora de la infraestructura de desarrollo centralizada y con buenas prácticas,* se ha realizado una intensiva labor en el diseño de un reparto de credenciales y roles para una correcta separación de los ámbitos de desarrollo, en pro de proteger y securizar al máximo la salud del producto en producción.

De esta forma, es el Administrador de la tecnología (CTO en la jerarquía), la persona encargada de generar roles, distribuir capacidades, y limitar accesos y publicaciones.

Dado que la totalidad del entorno está contemplada en AWS, los trabajadores pueden hacer uso de todos los beneficios implícitos. Por ejemplo, el acceso a la gestión de la totalidad de servicios mediante conexión segura basada en SSH, la capacidad de generación de tickets internos, etc.

1.4.3. Beneficios para clientes

En este caso, el listado de beneficios para los clientes son los siguientes:

1. Menor latencia en el servicio.
2. Menor espera ante nuevas funcionalidades.
3. Menor downtime, ligado al objetivo del 99,9% SLA.

1.4.4. Beneficios para el alumno

Las más de 1.800 horas dedicadas en exclusividad al desarrollo de este proyecto han generado una serie de beneficios al desarrollador, entre otros:

1. Consolidación de conocimientos en Java, AWS, GraphQL, SQL y entornos de producción, entre otros.
2. Preparación para la certificación AWS Cloud Architect Professional.
3. Consolidación de conocimientos de las labores de CTO.

1.4.5. Beneficios económicos

1.4.5.1. Analizado como el ahorro en costes

Previa ejecución del proyecto, y tal y como refleja el Gantt anexo a este Trabajo Fin de Máster, la empresa realiza una fuerte labor de análisis de alternativas, donde se estudian diferentes presupuestos exigidos a diferentes empresas para acometer el proyecto. Presupuestos que oscilan entre los 58.000 € y los 145.000 €.

Si bien la totalidad de los presupuestos hacían referencia a la realización de un trabajo cuantitativamente más amplio que el trabajo recogido en el presente Trabajo Fin de Máster, la parte equivalente ahorrada para la empresa está valorada en 13.348,49 € para el presupuesto de menor valor hasta 100.348,49 € para el más caro.

1.4.5.2. Analizado como una plusvalía

Más allá del ahorro de costes, la realización del presente trabajo tiene un impacto alto en la valoración de la empresa. En Startups SaaS B2B, el disponer de una infraestructura tecnológica sólida, optimizada y actualizada, es un aspecto muy seguido de cerca por el colectivo VC, el stakeholder de mayor importancia en esta clase de empresas.

1.4.5.3. Analizado por el impacto en ventas

Beneficio obtenido a consecuencia de los *Beneficios 1.4.3: Beneficios para clientes*. Con un mejor producto que incluye una mayor oferta de servicios, que presenta una menor latencia y una menor tasa de downtime, las ventas de la compañía tienen un impacto positivo. De cualquier forma, el análisis objetivo y cuantificado de este impacto no se incluye en el presente Trabajo Fin de Máster por no ser el principal motivo de estudio.

1.5. Estado del arte

Este Trabajo Fin de Máster se fundamenta en un profundo análisis del estado del arte de las tecnologías necesarias para el desarrollo y despliegue de un producto de estas características. De este modo, se ha estudiado el grado de madurez tecnológica, así como el nivel de idoneidad de las siguientes tecnologías, campos del estudio, o productos propietarios: Cloud Computing, Java (esencialmente Java en su versión 11), el proyecto Spring, Bases de Datos, la tecnología GraphQL, productos basados en API y cartera de servicios basados en agregación de API, un vistazo al stack tecnológico del principal percentil de las Startups, y un análisis comparativo de las ventajas del desarrollo nativo frente al desarrollo híbrido de aplicaciones móviles.

El presente análisis del estado del arte tiene, por lo tanto, un sentido de precedente del *Apartado 1.6: Análisis de alternativas*. Sin embargo, el estudio pretende además servir de hoja de ruta para el diseño de productos digitales masivos erigidos sobre la vertical de SaaS B2B.

1.5.1. Cloud Computing

La definición formal del Cloud Computing por Red Hat, creadores, mantenedores y contribuyentes de un gran número de proyectos de software libre [15], con productos como Red Hat Enterprise Linux, JBoss e incluso el analizado en este Trabajo Fin de Máster Hibernate, es la ejecución de las cargas de trabajo en las nubes, las cuales son entornos de TI que extraen, agrupan y comparten recursos flexibles en una red [16].

Google por su lado define el Cloud Computing como la disponibilidad bajo demanda de recursos de computación como servicios a través de Internet. Una tecnología que evita que las empresas tengan que encargarse de aprovisionar, configurar o gestionar los recursos y permite que paguen únicamente por los que usen [17].

En realidad, el Cloud Computing es la tecnología sobre la que se ha fundamentado el impacto de Internet en la sociedad, motivado por un crecimiento de los servicios en la nube, la calidad de los mismos, la mayor competencia, la reducción de las barreras de entrada, la facilidad de acceso al mercado por medio de lanzamiento de productos 100% digitales, y un largo etcétera.

En este estudio se van a fundamentar las claves del Cloud Computing, analizadas tanto desde el aspecto tecnológico como desde el aspecto de negocio, entendiendo los diferentes productos, la perspectiva del consumidor y del proveedor, la propuesta de valor, y en qué momentos y escenarios es conveniente adoptar o migrar a una solución basada en Cloud.

Para finalizar con cualquier duda sobre el impacto del Cloud Computing en el día a día de los ciudadanos en cualquier parte del mundo, se propone la siguiente reflexión. Tik-Tok es un SaaS B2C, hosteado de forma distribuida en el IaaS de Alibaba y en soluciones IaaS + PaaS de AWS, que a su vez utiliza SalesForce (un SaaS B2B) como CRM, decenas de herramientas de análiticas como Google Analytics (SaaS B2B), con una red CDN Akamai Connected Cloud (SaaS B2B), Google Ads (SaaS B2B) para captar oferta y demanda, tecnologías de análisis del comportamiento SaaS basadas en API, y una interminable lista de servicios en la nube que permiten que la red social china escale hasta los más de 1.000 millones de usuarios activos mensuales [18] con un crecimiento interanual que casi supera el 100% [19]. Por no mencionar el probable uso de herramientas Cloud como Notion en las reuniones ejecutivas, Figma para el diseño UI/UX en las reuniones de producto, o Mailchimp en las campañas de Email Marketing (o alternativas).

En resumen, el Cloud Computing es la herramienta transversal, horizontal y vertical definitiva que está dinamizando la presente revolución tecnológica.

Para entender las diferentes dimensiones de la tecnología, se presenta la *Figura 2: Desglose en capas del propietario de la gestión para los diferentes modelos Cloud Computing y el modelo on-premises.* Se trata de un gráfico propiedad de Red Hat que ilustra el reparto de responsabilidades en capas.

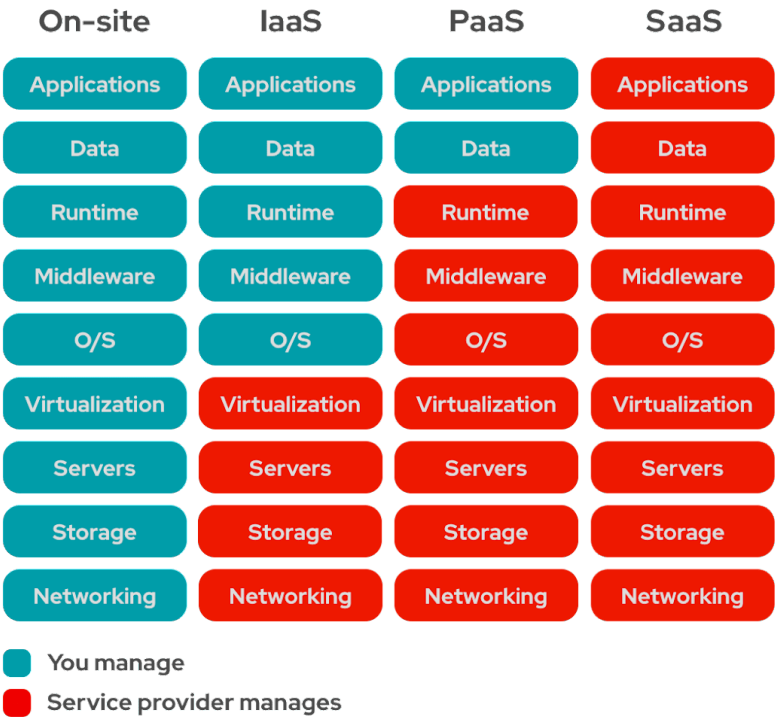

Figura 2: Desglose en capas del propietario de la gestión para los diferentes modelos Cloud Computing y el modelo on-premises [20].

1.5.1.1. Cloud Computing vs Self-Hosted o Self-deployed

El Cloud Computing nace como alternativa al despliegue de servicios o recursos (entornos según la definición Red Hat) por parte de los equipos de TI de forma local o propietaria, de forma denominada on-premises.

Entendiendo un producto de internet como un servicio corriendo en un determinado puerto de una determinada dirección IP y accesible, la evolución natural dió pie a un paradigma que contemplaba la compra y manutección de servidores como condición sine qua non para la gestión de estos productos.

Con el avance del Cloud Computing, el análisis comparativo se puede desarrollar en una serie de claves:

1. CAPEX: La compra y el mantenimiento de equipos en las propias instalaciones o *on-premises* es altamente costoso.
2. La previsión de la necesidad: La inversión depende de un estudio a años vista del servicio, incluyendo en ocasiones error humano.
3. Inversión, estrés de caja: Más allá del coste per sé, el equipamiento debe ser pagado con anterioridad al despliegue del producto a facturar.
4. Incertidumbre del ROI: Al ser la facturación desconocida, el ROI siempre es una incógnita.
5. Seguridad, el segundo gran presupuesto: La gestión on-premises requiere de un equipo de ciberseguridad del que depende la salud del producto de la empresa, con el coste y riesgo asociado.
6. Escalabilidad vertical y horizontal: A medida que el servicio escala, la escalabilidad se plantea como un reto en mayúsculas, en la mejor de las ocasiones tan solo implicando interrupciones en el servicio.

Por su lado, el Cloud Computing promete a las empresas proveedoras de servicios una mejora cualitativa en las seis claves mencionadas, de la siguiente forma:

1. Eliminación del CAPEX en favor de modelos de negocio Pay-per-use
2. No se realizan previsiones de uso y no se puede cometer el error de sobreestimar o subestimar el consumo de tu servicio o producto.
3. La caja no sufre en momentos iniciales, por lo que la empresa puede invertir una mayor partida en talento y producto.
4. La dirección siempre tiene opción de ajustar el gasto en el Cloud para adecuarse a la facturación, tanto positiva como negativamente.
5. El ticket por la gestión, manutección, instalación de actualizaciones y, en general de todos los areas de gestión que supone la seguridad están cubiertos por la empresa proveedora de Cloud, ahorrando en gran medida la factura al realizarse esta en escala. Reflejo burocrático fuerte en SLAs, seguros de Responsabilidad, y protocolos avanzados ante ciberataques son también puntos que se contemplan.
6. La escalabilidad puede llegar a ser transparente para la empresa, y puede ejecutar una asignación dinámica de recursos para atender a tendencias de compra o uso.

Para finalizar con el estudio, es necesario recalcar que existen situaciones concretas en las que soluciones on-premises son tomadas como válidas, en concreto se han descubierto dos en este

estudio. En primer lugar, empresas con un grado de madurez en los mejores percentiles. Empresas con productos líderes que se sitúan a la vanguardia de la tecnología ofertada en Cloud, con necesidades más fuertes en QoS, que son capaces de ofertar un mejor producto gracias a la gestión del stack completo, tanto HW como SW. En segundo y último lugar, empresas donde la protección de la información es crítica. Empresas desarrolladoras de patentes, intranets de gobiernos y lógicamente empresas que expiden certificados, CAs de cualquier nivel, empresas de gestión de red, ICANN…

1.5.1.2. IaaS: Infraestructure as a Service

Según Google, la infraestructura como servicio es un modelo de Cloud Computing que ofrece recursos de infraestructura bajo demanda, como computación, almacenamiento, redes y virtualización [19].

La Infraestructura en la nube propone mejorar la eficiencia operativa, haciendo priorizar el desarrollo de soluciones frente a la gestión de la propia infraestructura. Algunos de los productos más relevantes son los servicios de hosting, servicios de entornos de desarrollo, o de almacenamiento de datos. Otro de los productos principales del IaaS es la computación avanzada ante necesidades de análisis númerico, simulaciones complejas o renderizados, en los cuales puede abaratar el coste y ahorrar el tiempo respecto de la alternativa tradicional on-premises.

Como se refleja en las responsabilidades definidas en la *Figura 2: Desglose en capas del propietario de la gestión para los diferentes modelos Cloud Computing y el modelo on-premises,* el proveedor de IaaS se hace cargo de las capas de Red, Almacenamiento, Servidor (capa física) y virtualización. Por lo tanto, un IaaS dispone de un entorno preparado para la configuración directa de un sistema operativo. Previamente, suele ser común la elección de los parámetros básicos físicos del entorno.

En lo que respecta a los proveedores, al igual que en cualquiera de los productos del Cloud Computing, sobresalen Azure, Google y AWS. No obstante, la carrera por cuota de mercado como proveedores de cloud también es seguida por IBM, Oracle, e incluso por Alibaba. En concreto, el servicio IaaS más consumido en el proveedor líder por cuota de mercado, AWS, es AWS Elastic Compute.

1.5.1.3. PaaS: Platform as a Service

Por su lado, Google define PaaS como la oferta de una plataforma flexible y escalable en la nube para desarrollar, desplegar, ejecutar y gestionar aplicaciones [21].

En esencia, el PaaS presenta una reducción de las responsabilidades del consumidor o cliente, respecto del IaaS. En concreto, permiten el acceso a entornos de computación sin la necesidad de gestión del OS, nivel ahora gestionado por el proveedor. De igual forma, los clientes de los productos PaaS delegan también la capacidad de operar sobre el Runtime. Esto propone una serie de ventajas e inconvenientes que deben ser cuidadosamente analizadas para la correcta toma de decisión en cada caso.

En realidad, mientras que el IaaS supone un nivel de delegación objetivamente alto a la empresa, eliminando las fricciones de las capas de bajo nivel como la física, la de red, y la de virtualización, deja total libertad al cliente en la instalación del sistema operativo, y todo aquello instalado sobre él. La gestión del sistema operativo implica una cuidadosa manutección en materia de recursos, actualizaciones, parches, incompatibilidades, y buenas práxis en aspectos de seguridad. Además, la correcta y óptima gestión de la capa de runtime es igual o más crítica en entornos de producción.

Para entender la dimensión de la capa lógica de runtime, se debe definir cómo se ejecutan las aplicaciones, presentar el rol fundamental que juegan los contenedores, y definir algunas buenas prácticas entorno al despliegue y mantenimiento.

En primer lugar, las aplicaciones deben ser embebidas en contenedores para poder ser ejecutadas. Los contenedores, de forma resumida, ofrecen un entorno dedicado, funcional, fácilmente migrable, y con todo lo necesario en materia de recursos para que una aplicación pueda ser ejecutada. La abstracción lógica del runtime es responsable de especificar y definir el contenedor, además de gestionar los diferentes estados del mismo.

La gestión de los contenedores y por ende, de la capa de runtime, es compleja y es a menudo motivo de caídas en el servicio, cuellos de botella e inconsistencia en entornos de producción. Cuando se presenta este tema, se hace referencia a que los contenedores deben ser ejecutados de forma estandarizada, de forma que existe un buen aislamiento de las capas inferiores y la aplicación es

migrable, segura, e isolada, de forma que existe un buen aislamiento respecto de aplicaciones colindantes o coalojadas.

Para finalizar con el análisis de esta tecnología, se ha desarrollado una tabla comparativa con los pasos a seguir para lanzar una instancia o una función respectivamente de los productos más representativos de cada una de las verticales, siendo ambos además dos de las soluciones con más penetración en el mercado del ecosistema AWS: AWS EC2 y AWS Lambda.

1. Configuración requerida para lanzamiento de instancia AWS EC2 (IaaS):
    a. Arquitectura.
    b. Dimensionado físico de la instancia (RAM y vCPU).
    c. Opciones de almacenamiento.
    d. Ajustes de red, seguridad, *firewall*, y claves.
    e. Elección opcional de una plantilla de Sistema Operativo preconfigurada, con opciones como Amazon Linux, Red Hat, MacOs, Ubuntu y Windows, entre otras.
2. Configuración requerida para lanzamiento de función AWS Lambda (PaaS):
    a. Arquitectura.
    b. Versión del lenguaje de programación en la que estará el código de la aplicación.
    c. Ajustes para la integración con otros servicios complementarios de menor nivel como aspectos de rutado, gestión de la red, entre otros.

Como se puede comprobar, la dimensión de gestión que elimina un producto como AWS Lambda frente a AWS EC2 es mayúsculo y, como todo, la decisión de cada enfoque debe ser cuidadosamente argumentada desde negocio y desde ingeniería. A continuación, se presentan las claves a analizar en cada caso:

1. AWS Lambda recorta el tiempo asociado al lanzamiento de soluciones o aplicaciones en entornos de producción en un orden significativo.
2. AWS Lambda gestiona el OS y el Runtime, por lo que reduce, además reforzado con una política de SLA fuerte y altamente valorada, el espectro potencial de fallo de la aplicación.
3. AWS Lambda gestiona el Runtime de forma inteligente y optimizada, pero esto no tiene por qué ser siempre la mejor opción. AWS Lambda inicia y termina las aplicaciones, denominadas funciones, en función de la necesidad. Si la función no es requerida en una ventana de tiempo, pasará a un estado de hibernación y, en el momento en el que es

nuevamente requerida, se inicia. El tiempo asociado al levantamiento de la aplicación es denominado cold start, y es un parámetro a seguir de cerca para entornos de producción con clientes.

4. El Cold Start varía con el tamaño de la aplicación y con el runtime pasando del orden de las décimas de segundo para una pequeña aplicación Node.js 18.x a más de 15 segundos para una aplicación Java EE con un jar uber de 30 MB. En otros productos PaaS de otros proveedores también existen conceptos similares para referenciar al tiempo de interrupción de servicio relativo a la puesta en marcha de una aplicación.
5. AWS Lambda permite la construcción de un pipe CI/CD más ligero, basado en la publicación de la imagen del contenedor, o del jar/war en su caso.
6. AWS Lambda escala. De forma no transparente, pero escala. Escala hasta el denominado límite de concurrencia, un concepto visual y recurrente en el sector de las soluciones stateless.
7. Y lo más importante, AWS Lambda tiene un sistema de pago por uso, que consigue granularizar la factura, con un grado de resolución ceñido al microsegundo. Este concepto es capaz de decantar en la mayoría de casos la decisión en su favor y en detrimento de AWS EC2, cuyo modelo de negocio está sujeto a un precio fijo por tipo de sistema operativo y ponderado por horas de uso. Salvo en escenarios muy concretos, las aplicaciones en producción están activas las 24 horas del día, lo que hace que la factura de este servicio, sea varios ordenes de magnitud superiores a las de su competidor.
8. El precio a pagar por la delegación de la gestión del OS y el Runtime es alto en AWS Lambda, e influye en la forma en la que se desarrolla el código. Para poder garantizar la escalabilidad horizontal de Lambda, la función no puede ser stateful (la sesión no puede ser mantenida y servida a través de diferentes nodos a los que se accede mediante un balanceador de carga).
9. Además de stateless, la función debe ser ligera, a menudo sirviendo un número de propósitos reducido (o único) por función. AWS Lambda es ideal para el lanzamiento de microservicios, o pequeñas aplicaciones. Al contrario de EC2, Lambda no permite, mediante penalización en forma de retardo, la ejecución de aplicaciones pesadas, enterprise, o monolitos.
10. Si la situación de la empresa es la migración de una arquitectura monolítica on-premises al Cloud, la decisión dependerá únicamente de si se está dispuesto a modificar la arquitectura en un refactoring del producto. En caso positivo, es un momento ideal para preparar la aplicación ante la escala con buenas prácticas del estado del arte. En caso negativo, la

solución será adoptar EC2 o EKS (Servicio de Kubernetes Administrado) en el supuesto de ser una aplicación orquestada basada en Kubernetes y desear la flexibilidad de EC2.

1.5.1.4. SaaS: Software as a Service

Según Amazon, el SaaS es un modelo de software basado en la nube que ofrece aplicaciones a los usuarios finales a través de un navegador de Internet. Los proveedores alojan servicios y aplicaciones para que los clientes puedan acceder a ellos bajo demanda [22].

Mientras que los anteriores productos tienen un cliente concreto, relacionado a ofrecer servicios a equipos de desarrollo, entornos de ejecución de aplicaciones, gestores de redes… El SaaS es el vértice visual que tiene la capacidad de adoptar como consumidores a todo el conjunto de la población. Desde redes sociales, hasta softwares de gestión de recursos humanos bajo demanda, hasta servicios de contabilidad y finanzas para pequeñas empresas, pasando por los servicios de mensajería instantánea o incluso videollamada.

Es aquí donde deben ser definidos los conceptos B2B (Business to Business), B2C (Business to Consumer), y donde debe ser desarrollado el concepto de cómo el modelo de negocio es capaz de decidir la arquitectura de un producto.

Como su propio nombre indica, B2B son soluciones donde el cliente final es otra empresa, mientras que B2C hace referencia a aquellas soluciones donde el cliente final es una persona física. Sendos ejemplos de las dos tecnologías son Zapier y Spotify, respectivamente.

Ahora bien, existe una gran correlación entre el apellido de la tipología del SaaS con el paradigma de la arquitectura de la solución. Esto responde a (1) la tendencia de uso y (2) el tamaño de mercado de las soluciones. Las aplicaciones B2C suelen priorizar los KPI asociados a la tendencia de uso y a la recurrencia de consumo antes que la propia monetización. Además, se dirigen a mercados varios órdenes de magnitud mayores que las soluciones B2B, se puede analizar comparando el número mensual de usuarios activos de Netflix frente al de Salesforce Enterprise. Además, el producto B2C, en la mayoría de los casos, suele ser muy acotado, con un número de funcionalidades limitadas, que se ejecutan de forma masiva continuamente.

Lo que muestra esto, es que la tecnología debe depender del negocio siempre. Las arquitecturas deben depender de las especificaciones impuestas por los casos de uso, utilizando siempre las herramientas disponibles para priorizar aquello que es mejor para vender más. En este sentido, grandes arquitecturas monolíticas están permitidas si están justificadas, y complejas orquestaciones de microservicios mediante Kubernetes no son siempre la solución a adoptar, aunque el presupuesto sea infinito.

Para terminar, existe una coherencia jerárquica y matricial en la forma en la que se propaga la provisión de servicios. Un ERP, que ofrece sus servicios bajo un plan mensual a decenas de miles de empresas, puede estar construído bajo una arquitectura de microservicios en un PaaS como AWS Lambda, mientras que delega el motor de la base de datos a una instancia IaaS EC2 en la que se han provisionado de manera estática una serie de recursos. Lo que está claro, es que la evolución tecnológica tiende a separar cada vez más cada una de las capas lógicas debido a que la modularidad cada vez encuentra más casos de uso y más clientes que están dispuestos a adoptar nuevos y más complejos diseños, en favor de mejorar sus productos.

Esto ofrece una amplia, motivadora, pero amarga visión del estado del arte del Cloud Computing. Amplia debido a la multitud de arquitecturas y combinaciones posibles. Motivadora debido al brillante presente y futuro de la tecnología, donde la tecnología no se estanca, y el mercado es siempre capaz de verter nuevas soluciones que mejoran. Pero amarga, debido a que no existe una única solución, a que los productos quedan fácilmente obsoletos en obsolescencias programadas cada vez más cortas, y a que es complicado encontrar talento experto a la particular elección del stack de una empresa, por lo que los periodos de aprendizaje tienden a crecer.

1.5.2. Java 11

El índice TIOBE marca Java como el tercer lenguaje de programación más utilizado en el desarrollo de software a nivel Enterprise, cerrando mayo del 2023 con un market share del 12.22%. Java se sitúa por tanto justo detrás de Python y C, que cuentan con un 13.45% y un 13.35% respectivamente [23].

De hecho, teniendo en cuenta que la presencia del lenguaje de programación C es debida a su dominancia en otros mercados como el de deep-tech, videojuegos e ingeniería, Java es realmente la segunda fuerza a día de hoy en el mercado analizado en este Trabajo Fin de Máster.

En este caso, es necesario echar un vistazo a las grandes tecnológicas que confían en la comunidad de Java para desarrollar la práctica totalidad de sus productos. Entre otros, Microsoft, Paypal, Netflix, Uber, Salesforce y Airbnb.

Java es elegida debido a su estabilidad, historia, seguridad, comunidad e idoneidad para el desarrollo de soluciones enterprise. Según Oracle, Java es usada por más de 6 millones de desarrolladores, y corre en más de 5.500 millones de dispositivos [24].

Desde su inicio en enero del 1996, Java ha presentado nuevas versiones de sus ediciones SE (Standard Edition) y EE (Enterprise Edition) prácticamente cada año. No obstante, el ecosistema Java tiene una extensa lista de conceptos que pueden ser erráticamente confundidos y llevar a incompatibilidades, como Java Development Kit. Si bien no es el motivo de este trabajo definir y recoger las particularidades de cada uno de los productos, se cree necesario subrayar brevemente cada uno de los conceptos.

Java SE hace referencia a las clases que forman prácticamente el núcleo de los programas escritos en Java; como la API Collections, las clases I/O, la API JDBC, las clases en java.long, y un largo etcétera. Estos "paquetes" de código están contenidos en el directorio del JDK.

El JDK, es la denominación que se le da a la colección de herramientas necesarias para desarrollar aplicaciones Java. En este caso, se incluyen el Compilador Java (encargado de traducir las clases Java en ficheros bytecode .class), la JVM o Máquina Virtual de Java (necesaria para la ejecución de los .class), la herramienta JAR para la compilación, etc.

Por su lado, Java EE es una especificación con sub-especificaciones basado en Java SE. Según Sun, es el standard de la industria para desarrollar aplicaciones Java portables, robustas, escalables y seguras en el server-side. En resumen, proporciona APIs para diferentes servicios, entre otros, web. Informalmente se trata Java EE como un requisito o un signo de buena práxis en soluciones enterprise, aunque no signifique ningún certificado per sé.

Una de las cuestiones principales tras el primer vistazo sobre la comunidad y el abanico de productos, es la decisión de la versión a utilizar. Se trata de un paso crítico y que requiere un compromiso entre funcionalidades, penetración en el mercado, comunidad, fecha de caducidad de soporte, e integraciones o más bien incompatibilidades.

El árbol de decisión más común es rechazar todas aquellas versiones no LTS, lo que tan solo deja Java SE 8, 11 y 17. Tal y como dice Microsoft [25], la cuestión no es si migrar de Java 8 a Java 11, sino cuándo, y es que el soporte de Java 8 está muy próximo a su fin. Por tanto, tan solo queda Java 11 frente a Java 17. Soluciones lanzadas en septiembre del 2018 y Septiembre del 2021, respectivamente y que presentan argumentos a favor y en contra. Es en esta situación cuando la balanza se suele inclinar hacia su versión 11. Con más penetración en el mercado, mayor comunidad y, sobre todo, un grado de compatibilidad casi absoluta, Java 11 suele ser la decisión más coherente. Sin ir más lejos, AWS no disponía de soporte para Java 17 cuando comenzó este proyecto, y no ha sido hasta el 27 de abril del 2023 que la empresa americana ha integrado el soporte finalmente [26].

1.5.3. Spring y Spring Boot

Spring es un Framework para el desarrollo de aplicaciones en Java y de código abierto que nace para simplificar la arquitectura Java EE, ofreciendo un entorno para el desarrollo de código de alto rendimiento, liviano y reutilizable.

En pocas palabras, Spring crea un entorno production-ready totalmente configurable con herramientas propietarias (Spring Security, Spring Data JPA, Spring ElasticSearch…), servidores de aplicaciones embebidos, orientado al despliegue en la nube mediante Spring Cloud, y optimizado para el desarrollo de microservicios.

Según el informe de productividad del desarrollo Java de 2021 [27], el 62% de los desarrolladores Java entrevistados de la muestra del estudio utilizan Spring Boot como su principal framework de desarrollo.

De cualquier forma, Spring y Spring Boot no son lo mismo. Spring Boot es una extensión de Spring que acelera el proceso de desarrollo aún más con una serie de librerías llamadas *starter [28]*. La utilidad de Spring Boot es mayúscula en términos de aceleración de desarrollo, asegurando el uso de

buenas prácticas seguidas por una fuerte y actualizada comunidad. Un ejemplo de esa utilidad propuesta por Spring Boot con las librerías mencionadas puede encontrarse en el proyecto spring-boot-starter-thymeleaf. Con Spring Boot la inclusión de la librería en el directorio Maven [29] del proyecto es suficiente mientras que, con Spring, sería necesaria la inclusión de la librería Thymeleaf, sus dependientes, configuración del template resolver, template engine, y mapear los nombres de la vista con el view resolver.

Dentro de Spring, existen proyectos de gran interés como Spring Security, Spring Cloud, y Spring JPA. Durante el desarrollo de este Trabajo Fin de Máster se ha trabajado extensivamente con cada uno de estos subproyectos, obteniendo así una imagen completa del arbol de beneficios.

Spring Security es el estándar de facto para la gestión del control de acceso en aplicaciones Spring. Provee de los procesos de autenticación y autorización de una forma flexible y con la máxima seguridad ante ataques del tipo fijación de sesión, clickjacking, CSRF, etc.

Spring Cloud por su parte hace posible la rápida adopción de patrones comunes en los entornos distribuidos como la gestión de la configuración, descubrimiento de servicios, rutado inteligente, tokens, elección por voto, distribución de sesiones, gestión del estado del cluster e incluso balanceador de carga. La migración de un proyecto Spring a la nube mediante el uso de esta librería es sencilla, pero sin renunciar a la robustez de la solución o sin echar en falta otras integraciones adicionales.

Finalmente, Spring Data JPA o Spring Data Java Persistence API, es un subproyecto del subproyecto Spring Data. Una librería que convive con otras como Spring Data JDBC, Sring Data MongoDB o incluso Spring Data LDAP. En pocas palabras, ofrece la implementación de repositorios JPA, para el acceso a la capa de datos. Según el propio Spring, la implementación de una capa de acceso de datos implica el desarrollo de extenso *boilerplate code* para la construcción de consultas sencillas, sin mencionar conceptos más avanzados que incluyan paginación y ordenación en los datos. Spring Data JPA provee de una extensa lista de implementaciones, las cuales deben ser identificadas mediante la construcción, como desarrolladores, de una interfaz de repositorio.

1.5.4. Bases de Datos en entornos Cloud, ¿Cómo decidir?

Este apartado no define el estado del arte de las bases de datos per sé. Tampoco define las bases de datos basadas en SQL ni NoSQL, ni las diferencias entre unas y otras. Al contrario, se centra en definir el set de opciones disponibles a la hora de decidir una arquitectura de base de datos para la construcción de un producto SaaS desplegado en la nube.

Al igual que se introducía en el *Apartado 1.5.1.4: SaaS: Software as a Service*, para la correcta decisión de la arquitectura de una de las piezas fundamentales del producto, es necesario un previo análisis del producto a desarrollar y que, normalmente, podría responder al siguiente guión de cuestiones.

1. ¿Cuántas lecturas se van a realizar por segundo?
2. ¿Cómo se realizan las lecturas y a qué obligaciones responde en términos de velocidad de servicio?
3. ¿Cuál es la flexibilidad que necesita el producto en términos de estructura de datos?
4. ¿Es la escalabilidad horizontal una necesidad?
5. ¿Es necesaria integridad y normalización de los datos?
6. ¿Dispones de un data warehouse como fuente de la verdad?
7. ¿Está tu producto relacionado con el procesamiento de big data, streaming, o aplicaciones en tiempo real?
8. ¿Se trata de un producto de captación de sensórica o mediciones rápidas?
9. ¿Es tu aplicación un CRM, ERP, o está relacionada con servicios financieros?
10. ¿Es prioritaria la ejecución de consultas complejas a la velocidad de consultas sencillas?

En la mayoría de los casos en los que la empresa se siente más atraída por los puntos 1 a 8 incluídos, la arquitectura debería ser NoSQL, y el siguiente paso debería ser la profundización de las opciones NoSQL. Por el contrario, en los casos de negocio que responden más a las cuestiones 9 y 10, el enfoque debería ser SQL indudablemente.

Mientras que NoSQL antepone la velocidad de escritura y lectura de operaciones sencillas mediante la redundancia, duplicidad y falta de estructura en los datos, las buenas prácticas de SQL obligan a la normalización y a la rígida gestión de la integridad de los datos, siendo así perfecta para aplicaciones transaccionales sin requisitos de masividad en la escala.

En el presente Trabajo Fin de Máster se ha trabajado con 3 bases de datos, desarrollando sendas aplicaciones funcionales con cada una de las tres: DynamoDB, MySQL, MsSQL. Adicionalmente, se ha realizado otra aplicación de prueba con la versión MySQL de AWS Aurora, un servicio de clusterización, sin servidores, escalable y con un SLA imbatible de bases de datos SQL. Las conclusiones obtenidas han motivado el desarrollo de las reflexiones reflejadas en este apartado.

1.5.5. GraphQL

GraphQL es un lenguaje de consultas para APIs desarrollado por el equipo tecnológico de Facebook [30] como alternativa al paradigma Restful. GraphQL provee de una única y unificada descripción de los recursos de la API, lo cual da al cliente la capacidad de exigir exactamente lo que necesita, y nada más, mejorando en órdenes de magnitud la eficiencia por la reducción de la información transportada.

Desde la propia fundación GraphQL se subraya la predictibilidad de la solución: *consulta lo que necesitas, y obtén exactamente eso, nada más* [31].

Como se ha mencionado, GraphQL nace de la necesidad por una solución actualizada para gestionar las APIs en entornos data-driven. De hecho, la mejor forma de entender los beneficios que presenta es compararla frente al todavía hoy estándar de facto, Rest.

1. Unidad y fuente única de la verdad: Frente al paradigma basado en *endpoints* de Rest, GraphQL propone una única interfaz de entrada a todos los productos de una empresa, ocultando por completo la arquitectura por detrás al cliente. Detrás de una API GraphQL puede existir un *backend*, un cluster de microservicios, una arquitectura modular localizada físicamente en entornos remotos, otras APIs de GraphQL, o incluso una combinación de todas ellas.
2. Un solo recorrido de ida y vuelta: Los clientes realizan una petición y reciben una respuesta. La anidación es iniciada y gestionada por la capa GraphQL, en caso de ser necesaria. Esto es esencial para proporcionar transaccionalidad. Una solicitud GraphQL puede requerir una subconsulta del servidor a una API tercera para un proceso de autenticación/autorización, o para realizar una comprobación, o incluso para descargar una propiedad de datos de

dirección más completa de la API de Google Maps, tras haber solicitado un par Longitud-Latitud del cliente…

3. Los datos y los tipos de datos se definen públicamente, lo que ofrece total transparencia para los clientes, a la par que reduce el espacio para el error.
4. Gestión de versiones: Mientras que en Rest las nuevas versiones suelen representar diferentes directorios, la gestión de versiones en GraphQL no tiene por qué ser binaria. De hecho, al ser una herramienta introspectiva, los desarrolladores o mantenedores de la API pueden indicar textualmente el estado o el plan de versiones (o el plan de dejar obsoleto un recurso), y los clientes podrán solicitar el esquema y actualizarse consecuentemente.
5. Absoluta flexibilidad: Para una solicitud de consulta (GraphQL *Query*) o una petición de actualización o escritura (GraphQL Mutation), los datos parámetros de entrada y datos exigibles de respuesta pueden ser obligatorios o no obligatorios, ofreciendo flexibilidad para los mantenedores de la API, y ofreciendo flexibilidad para los clientes. Cuantos menos propiedades de datos de respuesta sean obligatorias para una consulta, más flexibilidad de tendrá el cliente para obtener únicamente los datos que necesita, que pueden variar con el tiempo.
6. Ligereza: Existe conciencia sobre el overhead asociado en consultas Rest, donde, aunque únicamente sea necesaria la recepción de una propiedad de datos para una consulta, es siempre necesario la devolución de todo el objeto definido como respuesta para la consulta.

Por su lado, las desventajas atribuibles a GraphQL están asociadas a la complejidad en la nueva gestión de la Caché, la complejidad en el diseño de la arquitectura para la resolución de las consultas, la complejidad desplazada a los desarrolladores de los servidores, y el reto asociado a la definición de un esquema GraphQL fácil de entender, inequívoco, y fácil de mantener.

Una vez entendidas las ventajas de la tecnología, son muchas las empresas que están empezando a formar parte de la comunidad GraphQL. Entre otras, Meta (equipo fundador), Github (además inversora [32]), Paypal, Shopify, Twitter e incluso AWS. El market cap de esta comunidad es ya superior a los 7B € [33].

1.5.6. Mejora continua de producto mediante la agregación de APIs

Sin duda uno de los grandes cambios en la forma en la que se construyen aplicaciones es el uso protagonista que ha adoptado el desarrollo mediante la agregación de APIs. Se trata de un concepto definido brevemente en el *Apartado 1.5.1.4: SaaS: Software as a Service*, pero que requiere un desarrollo aparte.

El *sweet point* de la maduración de Internet ha propiciado un círculo de productividad tremendamente positivo, donde continuamente nuevas herramientas surgen al mercado para solucionar retos existentes y abrir un nuevo abanico de opciones. A su vez, estas opciones facilitan directamente el desarrollo de nuevas soluciones innovadoras y tecnológicas que, debido a que aceleran su proceso de escala, reducen las barreras de entrada de caracter tecnológico (pero también monetario) y dan como resultado un mercado mayor.

A este ecosistema se le suma un interés desmedido por parte del sector del capital riesgo (VC) que, traducido en asignaciones de múltiplos desmedidamente altos en las valoraciones (llegando a x64 ARR en octubre del 2020), invierten la práctica totalidad de la capacidad del fondo conquistados por productos híper escalables de márgenes cercanos al 100%. Este fenómeno contribuye a dar como resultado el boom tecnológico del Cloud Computing.

Lo realmente interesante es que la forma en la que se adquieren esas herramientas innovadoras en forma de soluciones empresariales es unificada de forma global, y es mediante APIs. Por tanto, se está viendo un crecimiento en el porcentaje del producto que proviene de una integración por parte de otra empresa, respecto del producto desarrollado por la propia empresa.

Pero estas herramientas no sólamente contribuyen a la agregaciónd de oferta de servicios. Como ya se planteaba a lo largo de este Trabajo Fin de Máster, el producto de una empresa está directamente coordinado con decenas de productos, mediante APIs, para gestiones como la generación de métricas de usabilidad, el seguimiento de la traza de cada uno de los clientes (de dónde viene, seguimiento de sus tendencias, etiquetado y clusterizado…), la generación de eventos de conversión y desencadenamiento de "hooks", la gestión de las pasarelas de pago, etc.

Profesiolan, en su versión MVP, cuenta con más de 20 integraciones clave, mediante APIs, que han hecho del producto inicial un servicio que vende más, que mide más, que ofrece más funcionalidades, y que conoce mejor el comportamiento de sus clientes.

1.5.7. Arquitecturas, paradigmas, workers… ¿Qué hacen los líderes?

Para entender la coyuntura contractual de la tecnología relacionada con este Trabajo, también se hace necesario estudiar los patrones de diseño, las arquitecturas, y en general las tecnologías que se utilizan en las empresas líderes.

Asimismo, se desglosa el presente apartado en la definición de cinco grandes grupos de trabajo que representan fidedignamente los segmentos de trabajo en cualquier proyecto basado en Cloud: Seguridad, fiabilidad, gestión del data, diseño e implementación, y mensajería.

1.5.7.1. Enfoque en la seguridad: Federated Identity vs Gatekeeper

La decisión más relevante a la hora de realizar el diseño tecnológico de la seguridad de una aplicación es la de delegar las tareas de autenticación y autorización a un proveedor de identidades tercero, o abordarla "in-house".

El primero de los escenarios es el abordado por Federated Identity, mediante soluciones como Open-ID, por ejemplo. Se trata de un escenario que evita el overhead de la gestión de las identidades y de las sesiones, además de minimizar el alcance del error [34]. El concepto Identity Federation hace referencia a un sistema de confianza entre dos partes con el objetivo de autenticar usuarios y compartir alcances de información para poder autorizar accesos a recursos.

Las dos partes de un sistema de identidad federada son el IdP (proveedor de identidad) y el SP (proveedor de servicio), y la relación de confianza se presenta como una dependencia del SP a la decisión que realice sobre un usuario o identidad el IdP. Tras una autenticación satisfactoria, el IdP envía al SP la información del usuario necesaria para que el propio SP pueda establecer la sesión más apropiada, con los roles y el alcance necesario. [35, 36, 37]

Falsamente se ha generado una imagen negativa sobre este enfoque, centrada en la crítica de la inflexibilidad. Uno de los aspectos negativos más mediáticos es la imposibilidad de generar soluciones customizadas que se ajusten a las necesidades de cada negocio. En pocas palabras, son soluciones muy genéricas, limitadas, y poco flexibles, en las que a menudo es el propio stack tecnológico el que se ve afectado para poder realizar la integración.

Realmente, este enfoque es totalmente válido y utilizado en toda clase de productos y empresas que desean tener un sistema IdP central para el gobierno seguro de las credenciales de acceso, con múltiples servicios ejerciendo de SP.

Dentro del ecosistema AWS, tras la puesta a punto de IAM (la herramienta general de gestión de políticas de accesos a recursos), existen varios enfoques posibles para la integración del acceso a los SP con un IdP externo. Por ejemplo, mediante el IAM Identity Center, se puede utilizar Okta Universal Directory o incluso Azure Active Directory para autenticar usuarios sobre el protocolo SAML 2.0. De igual forma, AWS también ofrece un servicio de más alto nivel que extiende a IAM Identity Center como servicio, e incorpora más funcionalidades generalmente requeridas (como login, registro, gestión y recuperación de contraseñas, e inicio de sesión social), llamado AWS Cognito.

Finalmente, el enfoque Gatekeeper es en pocas palabras un *firewall* de nivel aplicación. Como tal, el compromiso del sistema Gatekeeper no supone ninguna brecha de seguridad para el producto o la empresa, debido a que en él no se expone ningún tipo de información sensible, ni credenciales.

Un sistema Gatekeeper se coloca entre el cliente y la aplicación o el servicio, y debe validar todas las peticiones y denegar aquellas que no cumplen con la política establecida. Además, es el encargado de exponer el *endpoint* al cliente, permitiendo añadir una capa de seguridad y esconder información de la aplicación.

Sin embargo, esta clase de sistemas también presentan una serie de inconvenientes. En primer lugar, se trata de un punto de fallo que asume la empresa (en comparación a la arquitectura federada). El desarrollo y la correcta gestión de un sistema así, supone una fuerte inversión, un gran volúmen de trabajo, y puede llegar a representar un problema si no se diseña con buenas prácticas. De hecho, el principal problema de los sistemas Gatekeeper es que suponen un único punto de fallo o cuello de

botella, que puede comprometer el sistema entero en caso de caída. Por supuesto, la adición de un sistema en paralelo supone un impacto negativo en el rendimiento total de cada petición.

1.5.7.2. Los monolitos modulares como alternativa eficiente en coste

No es hasta junio del 2023 cuando la propuesta de los monolitos modulares entran en la ecuación de las posibilidades de arquitectura cloud eficiente en coste. En concreto, en este apartado se repasa el paper publicado en junio del 2023 por varios de los directores de tecnología de Google, en HosOS 23 [39].

El estudio establece que el enfoque basado en microservicios, si bien está basado en la premisa de la división modular de una aplicación distribuida, no debería ser la opción principal. Una arquitectura basada en microservicios promete poner límites físicos (la forma en la que se despliega el código) a los límites abstractos (la forma en la que el código está escrito teniendo en cuenta la lógica de negocio).

Es indiscutible que los microservicios afectan al rendimiento debido a su construcción modular y despliegue distribuido, sin embargo, esta no es la única concesión. Estas arquitecturas multiplican el tiempo de desarrollo y el tiempo de gestión; no es lo mismo la gestión de versiones de una única aplicación que la de *n*.

Los autores también destacan la complejidad asociada a los puntos de conexión entre microservicios. Esto es aún más complejo cuando la vida del producto se alarga y el equipo de desarrollo debe lidiar con actualizaciones particulares.

Es por todo ello que se propone la arquitectura basada en monolitos modulares como alternativa eficiente en coste, para esas empresas que deben desarrollar un producto, y que no disponen (1) el presupuesto de las grandes tecnológicas ni (2) la necesidad de volumen de servicio.

La arquitectura presentada se basa en tres premisas. La creación de aplicaciones monolíticas que están modularizadas en componentes lógicos, la capacidad de asignación dinámica y automática por

parte de los componentes lógicos a procesos físicos, y la orquestación para el despliegue atómico de la aplicación para prevenir el coste de la gestión de las diferentes versiones.

Para entender este concepto, se debe presentar el componente. Este es un agente pensado a largo plazo teniendo en cuenta la lógica de negocio, que implementa una interfaz, y que la única forma para interactuar con él es mediante la llamada a métodos de su interfaz. Una vez arquitecturado, los componentes podrían estar desplegados en diferentes entornos (entornos distribuidos), como en el enfoque de los microservicios, pero evitando sus complicaciones.

1.5.7.3. Data Streaming

Con la escala del volumen de datos, el enfoque de la ingesta y la estructuración de los mismos por grupos (o *batches*) ha quedado obsoleta en muchos casos. Antiguamente, se disponía de una fuente de información, que puede ser un almacén de datos (o *data warehouse*), y servir ese contenido era sencillo por medio de la agrupación y obtención sincronizada de los paquetes.

No obstante, con la velocidad a la que crecen los volumenes de información y, sobre todo, aparecen nuevas necesidades de servicio como analíticas o aplicaciones basadas en la alta disponibilidad de tráficos prácticamente infinitos (aplicaciones como Netflix o en otro contexto Snowflake), el mencionado enfoque queda obsoleto.

Es por todo ello que nacen y se desarrollan los diferentes enfoques y conceptos asociados al *data streaming.* A continuación se presentan las principales ventajas del procesamiento en línea sobre el procesamiento en lotes.

- Las consultas no son realizadas sobre la totalidad o la mayoría del conjunto de datos, sino que se realiza sobre el conjunto de información existente en una ventana deslizante de tiempo.
- El tamaño de los conjuntos de datos sobre los que operar pueden llegar a convertirse en órdenes de magnitud menores.
- Como consecuencia directa, la latencia se ve reducida en gran medida.

En el mundo de la computación en la nube, existen soluciones para esta gestión como Amazon Kinesis o Azure Stream Analytics.

Para la correcta consecución de un sistema basado en esto existen dos agentes necesarios; el productor y el consumidor. El productor es el sistema que genera la información, pudiendo variar desde un sensor de IoT, hasta un componente software como una API. El consumidor, por su parte, es el encargado de procesar y analizar ese flujo de información que se encuentra en el *buffer de procesamiento*.

Por supuesto, la correcta puesta a punto de un sistema de consumo de flujos de datos es altamente compleja y con fuertes retos tecnológicos como la consecución de la alta disponibilidad y la escalabilidad para no hallarse en un caso de congestión debida a la alta generación de información por parte del productor y la imposibilidad de gestión por parte de la ventana deslizante y del consumidor.

1.6. Análisis de alternativas

En este apartado del proyecto, se recogen las problemáticas de ingeniería del proyecto. Se exponen en detalle los principales problemas inherentes a la naturaleza de la tecnología, a continuación, se listan y argumentan las alternativas ante la problemática en cuestión. En tercer lugar, se resumen los criterios de selección de las alternativas, ofreciendo campos objetivos de evaluación. En cuarto y último lugar se ponderan y evalúan cada alternativa, obteniendo un cuadro matriz y la decisión tecnológica finalmente seleccionada.

1.6.1. Estudio del paradigma tecnológico de la capa API

El producto es un servicio de consumo masivo por profesionales y empresas, con funcionalidades de búsqueda y filtrado, gestión de stock, y gestión de compra. Es por ello que el servicio debe ser totalmente funcional y rápido. Además, la API debe poder ser reutilizable, escalable, y orientada al alto rendimiento, ofreciendo alta disponibilidad.

1.6.1.1. Exposición de alternativas ante la problemática I

Alternativa I: Desarrollo de una API Restful.

A la hora de realizar un producto basado en API, la primera opción es siempre el tradicional enfoque Rest. En 2023, esta tecnología sigue teniendo una cuota mayoritaria, suponiendo un 93.4% de adopción en el mercado de productos basados en API y que, según Akamai en su informe anual sobre el estado del arte de la seguridad en internet [39], supone más de un 83% de todo el tráfico consumido en internet. Entre otros, gran parte del producto de Amazon, Google, LinkedIn o Twitter se sirve de forma RestFul. [40]

Alternativa II: Desarrollo de una API GraphQL.

Tal y como se desarrolla en el Apartado *1.5.5: Estado del Arte, GraphQL*, GraphQL es una tecnología novedosa, que nace como solución a las carencias de Rest, y que está creciendo de forma muy rápida entre los grandes proveedores de servicios en todo el mundo.

1.6.1.2. Criterios de selección de alternativas ante la problemática I

Criterio de selección I: Tendencia de mercado.
El producto en desarrollo debe tener una vida útil no menor de 10 años, por tanto es indispensable la apuesta por una solución robusta y que se prevea que siga teniendo una posición preferente en el largo plazo. Se trata de un criterio de alta relevancia, por ello tiene una ponderación de 0,4 (el 40% del peso total de los criterios de selección).

Se valorará con la mayor puntuación (5 puntos) aquella solución que tenga un ratio de crecimiento mayor en los últimos 10 años. Si bien esto no es un indicador absoluto de su usabilidad en un futuro, una tendencia de crecimiento ganadora durante una ventana de una década se considera una tendencia a tener en cuenta. De igual forma, se valorará con la mitad de la valoración total disponible más 1 (3) puntos a las otras soluciones.

Criterio de selección II: Adecuación al producto.

La adecuación de una tecnología al producto en desarrollo se ha demostrado que es de alta relevancia, y que sin duda son campos correlados. Es por todo ello que este criterio se valora con un 0,6.

En concreto, se puntuarán positivamente los siguientes aspectos:

- 1 puntos sobre 5 para la solución que pueda ofrecer una única e integrada API para todo el servicio Profesiolan.
- 1 puntos sobre 5 para la solución que permita una mayor velocidad de desarrollo entre los grupos de trabajo *backend* y *frontend*.
- 2 puntos sobre 5 para la solución más eficiente.
- 1 puntos sobre 5 para la solución que más fácil permita integrar la seguridad.

1.6.1.3. Evaluación de las alternativas ante la problemática I

En la *Tabla 1: Evaluación de las alternativas ante la problemática I,* se muestra la evaluación de las diferentes alternativas a la problemática principal en base a los criterios definidos. Para ello, se ha puntuado cada alternativa, siendo el 1 la menor puntuación y el 5 la mayor, para posteriormente calcular una puntuación total.

CRITERIO / ALTERNATIVA	VALORACIÓN (1-5)		
	Alternativa I: Desarrollo de una API Restful.	Alternativa II: Desarrollo de una API GraphQL.	PONDERACIÓN (0 - 1)
Tendencia de mercado	3	5	0,4
Adecuación al producto	1	4	0,6
PUNTUACIÓN	1,8	**4,4**	-

Tabla 1: Evaluación de las alternativas ante la problemática I. (Fuente: Elaboración propia).

Como se puede observar en la Tabla 1, la alternativa II es la que ha obtenido mayor puntuación y, por tanto, es la seleccionada para realizar el diseño de la capa API del producto estudiado en el presente Trabajo Fin de Máster.

1.6.2. Estudio del framework tecnológico del proyecto

Debido a la condición principal impuesta por la dirección de la empresa de utilizar Java como lenguaje y entorno de programación para el desarrollo del producto, la segunda problemática a abordar es la decisión del framework a utilizar, dentro del ecosistema Java.

1.6.2.1. Exposición de alternativas ante la problemática II

Alternativa I: Desarrollo en Java EE.

Tal y como se resume Java EE en el *Apartado 1.5.2 Estado del arte de Java 11*, la versión Enterprise de Java es siempre una decisión acertada y una alternativa obligada para el desarrollo de una aplicación basada en Java. Más aún cuando se busca estabilidad, seguridad, comunidad y apuesta por el futuro.

Alternativa II: Desarrollo en Java Spring Boot.

Entre todos los posibles Frameworks pertenecientes a la familia Java, se ha incluído Spring Boot como alternativa real por su rápido crecimiento e implantación en el mercado. Spring Boot se ha convertido en una opción segura, que apuesta por la modularidad para el desarrollo.

1.6.2.2. Criterios de selección de alternativas ante la problemática II

Criterio de selección I: Tendencia de mercado y adecuación al producto.

Tal y como se ha desarrollado a lo largo del *Apartado 1.5: Estado del arte*, para la correcta decisión en una elección entre alternativas tecnológicas, es siempre necesario hacer un fuerte estudio del

arte de la tecnología, analizar sus usos, y listar las empresas que apuestan por la alternativa y aquellas que no lo hacen.

Este criterio de selección tiene una ponderación de 0,3. En concreto, se valorará con la totalidad de la puntuación aquella solución por la que hayan apostado más startups SaaS basadas en Java en los últimos años y con facturaciones menores a 1 Millón de dólares. La otra solución, recibirá la mitad de la puntuación máxima.

Criterio de selección II: Reutilización y orientación a la escalabilidad.

De igual forma, se valora la escalabilidad del Framework y la capacidad de generar código robusto con una larga vida útil. Este aspecto tiene una ponderación de 0,2.

Criterio de selección IV: Métricas y KPIs.

En una clara apuesta por el buen rendimiento, el criterio de selección cuarto pondera con 0,3 la ligereza, y las métricas y KPIs benchmark de cada una de las alternativas.

Criterio de selección V: Conocimiento y experiencia del entorno por el equipo desarrollador.

Finalmente, también es necesario dar valor a la experiencia del equipo desarrollador, y apostar por la alternativa en la que el equipo pueda aportar mayor valor añadido. Este criterio tiene una ponderación de 0,2.

1.6.2.3. Evaluación de las alternativas ante la problemática II

En la *Tabla 2: Evaluación de las alternativas ante la problemática II* se muestra la evaluación de las diferentes alternativas a la problemática número 2 en base a los criterios definidos. Para ello, se ha puntuado cada alternativa, siendo el 1 la menor puntuación y el 5 la mayor, para posteriormente calcular una puntuación total.

CRITERIO / ALTERNATIVA	VALORACIÓN (1-5)		
	Alternativa I: Desarrollo en Java EE.	Alternativa II: Desarrollo en Java Spring Boot.	PONDERACIÓN (0 - 1)
Tendencia de mercado y adecuación al producto	4	4	0,3
Reutilización y orientación a la escalabilidad	5	4	0,2
Métricas y KPIs	4	4	0,3
Conocimiento y experiencia del entorno por el equipo desarrollador	3	5	0,2
PUNTUACIÓN	4	**4,2**	-

Tabla 2: Evaluación de las alternativas ante la problemática II. (Fuente: Elaboración propia).

Como se puede observar en la Tabla 2, la alternativa II es la que ha obtenido mayor puntuación y, por tanto, es la seleccionada para realizar el core del producto estudiado en el presente Trabajo Fin de Máster.

1.6.3. Estudio de la arquitectura de la base de datos

En Junio del 2023, la base de datos del modelo mínimo viable de Profesiolan, en una arquitectura MySQL supera los 1.5 Millones de entradas, ha recibido 318.3 TiB de tráfico de red, de los cuales 9.0 TiB eran referentes a contenido recibido y 309.2 TiB de contenido entregado. Además, ha llegado a alcanzar un máximo de 182 conexiones concurrentes, y dispone de una base de datos con más de 90.000 fotografías.

Con este precedente, y teniendo en cuenta que el desarrollo de un nuevo producto viene motivado por la necesidad de escalar las limitaciones del producto actual, la exigencia en la ejecución de la arquitectura de la base de datos es mayúscula. En este apartado, se analizarán las tres opciones más valoradas por el equipo de trabajo.

1.6.3.1. Exposición de alternativas ante la problemática III

Alternativa I: Desarrollo en MySQL.

MySQL es uno de los sistemas de gestión de bases de datos relacionales más utilizados en todo el mundo. Además, se trata del motor utilizado en el MVP de Profesiolan.

Alternativa II: Desarrollo en Microsoft SQL Server.

Siguiendo la lista como sistema RDBMS más popular, se encuentra SQL Server de Microsoft. A priori otra solución de SQL, pero propietaria de una de las tecnológicas más grandes del mundo. Esta naturaleza tiene muchas implicaciones positivas, ya que Microsoft dispone de herramientas adicionales para SQL Server y que vienen integradas con el motor, sin la necesidad de terceros. Además, el soporte es exclusivo, lo cual puede llegar a ser una imposición desde dirección en determinados empresas e industrias.

Alternativa III: Desarrollo en DynamoDB (NoSQL).

El ecosistema de las bases de datos sobrepasa el alcance SQL. De hecho, las tecnologías NoSQL están creciendo a un ritmo trepidante en sectores donde prima la velocidad, el rápido almacenamiento de datos, el real-time, y el big data. Entre todas las soluciones, se ha elegido DynamoDB.

DynamoDB, es una base de datos fully-managed, serverless, NoSQL, y de clave-valor, diseñada para ejecutar aplicaciones de muy alto rendimiento en cualquier escala. Su rendimiento para escritura y lectura, a costa de su estructura, es muy superior al rendimiento potencial de cualquiera de las soluciones SQL.

1.6.3.2. Criterios de selección de alternativas ante la problemática III

Criterio de selección I: Adecuación al producto.

Como ya se expone en el *Apartado 1.5.4: Bases de Datos en entornos Cloud, ¿Cómo decidir?*, en el nivel Base de Datos, existe un fit entre el servicio y la naturaleza del motor de base de datos. En él, se desarrolla el trabajo y las preguntas que debe hacer un diseñador de *backend* a la hora de tomar la mejor decisión. Este criterio tiene una ponderación de 0,5.

Criterio de selección II: Conocimiento y experiencia del entorno por el equipo desarrollador.

También es necesario dar valor a la experiencia del equipo desarrollador, y apostar por la alternativa en la que el equipo pueda aportar mayor valor añadido. Este criterio tiene una ponderación de 0,2.

Criterio de selección III: Coste.

Dado el alto coste asociado al despliegue de una base de datos de alto rendimiento en producción, las métricas de costes deben ser analizadas y ponderadas con 0,2.

Criterio de selección IV: Facilidad en la migración.

Debido a que se trata de un diseño que debe reemplazar un diseño anterior, se valora positivamente con 0,1 la facilidad asociada a la migración de toda la base de datos.

1.6.3.3. Evaluación de las alternativas ante la problemática III

En la *Tabla 3: Evaluación de las alternativas ante la problemática III* se muestra la evaluación de las diferentes alternativas a la problemática número 3 en base a los criterios definidos. Para ello, se ha puntuado cada alternativa, siendo el 1 la menor puntuación y el 5 la mayor, para posteriormente calcular una puntuación total.

CRITERIO / ALTERNATIVA	VALORACIÓN (1-5)			
	Alternativa I: Desarrollo en MySQL.	Alternativa II: Desarrollo en Microsoft SQL Server.	Alternativa II: Desarrollo en DynamoDB (NoSQL)	PONDERACIÓN (0 - 1)
Adecuación al producto	5	5	2	0,5
Conocimiento y experiencia del entorno por el equipo desarrollador	5	4	5	0,2
Coste	4	3	3	0,2
Facilidad en la migración	4	3	2	0,1

PUNTUACIÓN	**4,7**	4,2	2,8	-

Tabla 3: Evaluación de las alternativas ante la problemática III. (Fuente: Elaboración propia).

Como se puede observar en la Tabla 3, la alternativa I es la que ha obtenido mayor puntuación y, por tanto, es la seleccionada para ejecutar la arquitectura de la base de datos del producto estudiado en el presente Trabajo Fin de Máster.

1.6.4. Enfoque de la seguridad

Profesiolan es una aplicación de uso masivo que concentra decenas de miles de sesiones de forma mensual. Además, la elección del patrón arquitectónico basado en un *endpoint* público que expone una API GraphQL unificada y que potencialmente puede servir la totalidad del producto Profesiolan, la necesidad de una arquitectura de seguridad robusta, estable, ágil y escalable es también mayúscula.

1.6.4.1. Exposición de alternativas ante la problemática IV

Alternativa I: Enfoque Gatekeeper.

Tal y como se ha resumido en el *Apartado 1.5.7.1 Estudio del arte del enfoque de la seguridad: Federated Identity vs Gatekeeper*, el paradigma basado en un *firewall* de aplicación con la finalidad de esconder el servicio del cliente final es siempre una alternativa válida e interesante a valorar.

Alternativa II: Desarrollo Federated Identity mediante AWS.

Entre los dos enfoques posibles de la alternativa propuesta por AWS para la integración de autenticación mediante Federated Identity, se ha elegido AWS Cognito como servicio a implementar.

La tendencia a su implementación para la gestión de la autenticación en servicios con clientes en forma de aplicaciones móviles está haciendo de AWS Cognito el práctico estándar de facto en la comunidad AWS.

Alternativa III: Desarrollo Federated Identity mediante Java Spring Security.

Dentro de la modularidad propuesta por el Framework Spring Boot, la seguridad se plantea como una integración del proyecto Spring Security. Spring Security es el estándar de facto para securizar aplicaciones basadas en Spring, y se presenta como un paquete potente y totalmente customizable.

Spring Security es un proyecto mantenido por la comunidad Spring, a la orden del día, y muy utilizado en toda clase de servicios de uso masivo en todo el mundo.

1.6.4.2. Criterios de selección de alternativas ante la problemática IV

Criterio de selección I: Coste.

Teniendo en cuenta el número de sesiones diariamente generadas en el producto, se estima que el servicio asociado a autenticación y autorización va a ser ejecutado prácticamente de forma ininterrumpida, y es por ello necesario que la solución adoptada sea barata. Por ello, este criterio tiene una ponderación de 0,5.

Dentro del coste, se debe incluir el coste de desarrollo de la solución, además del gasto promediado por gestión de sesión.

Criterio de selección II: Conocimiento y experiencia del entorno por el equipo desarrollador.

También es necesario dar valor a la experiencia del equipo desarrollador, y apostar por la alternativa en la que el equipo pueda aportar mayor valor añadido. Este criterio tiene una ponderación de 0,2.

Criterio de selección III: Adecuación al producto.

El producto Profesiolan, si bien es un producto con multitud de funcionalidades e integraciones, no presenta un sistema de gestión de usuarios y sesiones complejo. De hecho, no es necesario extender las funcionalidades básicas y por defecto del proceso registro e inicio de sesión de cualquier proveedor de estas funcionalidades. Este criterio tiene una ponderación de 0,3.

1.6.4.3. Evaluación de las alternativas ante la problemática IV

En la *Tabla 4: Evaluación de las alternativas ante la problemática IV* se muestra la evaluación de las diferentes alternativas a la problemática número 4 en base a los criterios definidos. Para ello, se ha puntuado cada alternativa, siendo el 1 la menor puntuación y el 5 la mayor, para posteriormente calcular una puntuación total.

CRITERIO / ALTERNATIVA	**VALORACIÓN (1-5)**			
	Alternativa I: Enfoque Gatekeeper.	Alternativa II: Desarrollo Federated Identity mediante AWS.	Alternativa II: Desarrollo Federated Identity mediante Java Spring Security.	**PONDERACIÓN (0 - 1)**
Coste	2	5	3	0,5
Conocimiento y experiencia del entorno por el equipo desarrollador	3	4	5	0,2
Adecuación al producto	2	5	4	0,2
PUNTUACIÓN	2	**4,3**	3,3	-

Tabla 4: Evaluación de las alternativas ante la problemática IV. (Fuente: Elaboración propia).

Como se puede observar en la Tabla 4, la alternativa II es la que ha obtenido mayor puntuación y, por tanto, es la seleccionada para ejecutar el enfoque de la seguridad del producto estudiado en el presente Trabajo Fin de Máster.

1.6.5. Despliegue de la solución

Tal y como se ha antipicado a lo largo del presente documento, el despliegue y el entorno de despliegue de un producto es uno de los factores más relevantes y condicionantes de la estabilidad y el rendimiento en el largo plazo y ante situaciones de estrés.

El enfoque tomado en el despliegue condiciona absolutamente el CI/CD, el rendimiento debido a la configuración del servicio, e incluso a la velocidad en la que el equipo será capaz de reanimar y volver a activar el producto en caso de sufrir caídas.

Además, con el auge sin precedentes de los proveedores de Cloud, la opción por el despliegue on-premises está dejando de recibir interés por el CAPEX y los riesgos asociados, entre otros motivos. Sin embargo, la decisión por el despliegue mediante un proveedor de soluciones en la nube no elimina opciones, sino que las incrementa, existiendo en el mercado 3 opciones lo suficientemente maduras como para ser planteadas y con un brochure de soluciones muy parecidas sobre el papel.

1.6.5.1. Exposición de alternativas ante la problemática V

Alternativa I: Enfoque PaaS: Google Cloud.

Dentro del enfoque de despliegue basado en un proveedor de Cloud, Google es un referente y la segunda empresa con mayor cuota de mercado. Su completa lista de servicios alcanzan a cubrir la práctica totalidad de los servicios necesarios para el desarrollo, despliegue y gestión de cualquier solución SaaS.

Alternativa II: Enfoque PaaS: AWS.

Con el mismo enfoque en mente, la segunda alternativa es Amazon Web Services (AWS). El líder indiscutible en cuota de mercado también dispone de una extensa lista de servicios, llamados a cubrir todas las necesidades para el desarrollo de aplicaciones de cualquier sector en cualquier stack tecnológico.

Alternativa III: Enfoque en las instalaciones (on-premises).

Finalmente, también se valora como alternativa menor el despliegue de la totalidad del producto de forma on-premises.

1.6.5.2. Criterios de selección de alternativas ante la problemática V

Criterio de selección I: Adecuación al producto y a su estado de maduración.

Como se introduce en el *Apartado 1.5.1.1 Cloud Computing vs Self-Hosted o Self-deployed*, existen determinadas situaciones en las que un enfoque on-premises puede ser una buena idea para el despliegue de un producto software como servicio (SaaS). Sin embargo, debido a su coste, la complejidad y el punto de fallo que añade, y el jóven de la empresa, este enfoque se debe valorar con un 0,2.

En lo que respecta a la solución propuesta por AWS y Google respectivamente, el reparto de puntos debe ser el mismo y el máximo. La ponderación asociada a este criterio de selección es 0,2.

Criterio de selección II: Inversión inicial.

Para realizar un correcto análisis de costes del enfoque en el despliegue, es necesario calcular el diseño lógico y el stack de la solución para cada uno de los enfoques, y realizar un proyecto de cálculo. Este criterio tiene una ponderación de 0,3.

Finalmente, el enfoque que presente una solución más barata tendrá la totalidad de puntos. La siguiente la mitad, y la última no tendrá ningún punto asociado.

Criterio de selección III: Escalabilidad.

Una de las decisiones de mayor relevancia que motivan la migración a un proveedor de Cloud es la facilidad de escalabilidad; tanto horizontal como vertical. Este criterio tiene una ponderación de 0,1.

Criterio de selección IV: Cuota de mercado y tendencia.

El enfoque del despliegue es una decisión absoluta, y que no permite cambios en el largo plazo. De hecho, de todas las problemáticas de ingeniería en este documento desarrolladas y analizadas, este análisis de alternativas es el que presenta un mayor coste en caso de cambio de futuro, ya que supone un cambio de la base sobre la que se fundamenta el producto, el stack, la base de datos, y la seguridad.

Además, en productos con propuestas de valor y oferta de servicios tan similares, con precios similares, y con comunidades fuertes y activas, la decisión es todavía más complicada.

Es por ello que en este momento, la dirección de la empresa, tras un estudio de las estadísticas y de tendencias de las alternativas, debe hacer una apuesta por la empresa que más crecimiento prometa en el largo plazo. Este criterio tiene una ponderación de 0,2.

Criterio de selección V: Conocimiento y experiencia del entorno por el equipo desarrollador.
También es necesario dar valor a la experiencia del equipo desarrollador, y apostar por la alternativa en la que el equipo pueda aportar mayor valor añadido. Este criterio tiene una ponderación de 0,2.

1.6.5.3. Evaluación de las alternativas ante la problemática V

En la *Tabla 5: Evaluación de las alternativas ante la problemática V* se muestra la evaluación de las diferentes alternativas a la problemática número 5 en base a los criterios definidos. Para ello, se ha puntuado cada alternativa, siendo el 1 la menor puntuación y el 5 la mayor, para posteriormente calcular una puntuación total.

CRITERIO / ALTERNATIVA	VALORACIÓN (1-5)			
	Alternativa I: Enfoque PaaS: Google Cloud.	Alternativa II: Enfoque PaaS: AWS.	Alternativa II: Enfoque On-premises.	PONDERACIÓN (0 - 1)
Adecuación al producto y a su estado de maduración	5	5	2	0,2
Inversión inicial	5	5	2	0,3
Escalabilidad	5	5	3	0,1
Cuota de mercado y tendencia	4	5	3	0,2
Conocimiento y experiencia del entorno por el equipo desarrollador	2	5	5	0,2
PUNTUACIÓN	4,2	**5**	2,9	-

Tabla 5: Evaluación de las alternativas ante la problemática V. (Fuente: Elaboración propia).

Como se puede observar en la Tabla 5, la alternativa II es la que ha obtenido mayor puntuación y, por tanto, es la seleccionada para ejecutar el enfoque del despliegue del producto estudiado en el presente Trabajo Fin de Máster.

1.7. Descripción de la solución realizada

La descripción detallada de la solución tecnológica realizada se divide en los tres primeros anexos; *Anexo I. Descripción de la solución realizada: Diseño de alto nivel*, *Anexo II. Descripción de la solución realizada: Diseño de bajo nivel*, y *Anexo III. Análisis de rendimiento*.

No obstante, en esta memoria se incluye un breve resumen de la solución realizada relativa a la arquitectura de alto nivel, y un resumen relativo al profundo análisis de rendimiento operado.

1.7.1. Resumen de la arquitectura de alto nivel

Previa definición de los componentes que componen el *backend*, se hace especial mención a que el alcance del desarrollo tecnológico realizado está limitado a la construcción de una dirección *endpoint* unificada que pretende satisfacer la totalidad de las funcionalidades del producto Profesiolan.

Por tanto, la arquitectura asume la existencia de un cliente que tiene el conocimiento de las funcionalidades expuestas por la API, y tiene como alcance toda la tecnología desarrollada para servir cualquier consulta realizada por ese mismo cliente.

Así, el producto desarrollado en el presente Trabajo Fin de Máster es, de forma lógica, una caja negra con una única interfaz de entrada hacia internet, pública, y con la cual se puede interactuar siguiendo la documentación presentada a desarrolladores para el consumo de la misma.

A continuación, se presenta en la Figura 3: Resumen de la arquitectura de alto nivel, la arquitectura modular del diseño. En ella, se puede observar el bloque grisáceo identificado con el logo de la empresa, referente a la totalidad del producto desarrollado.

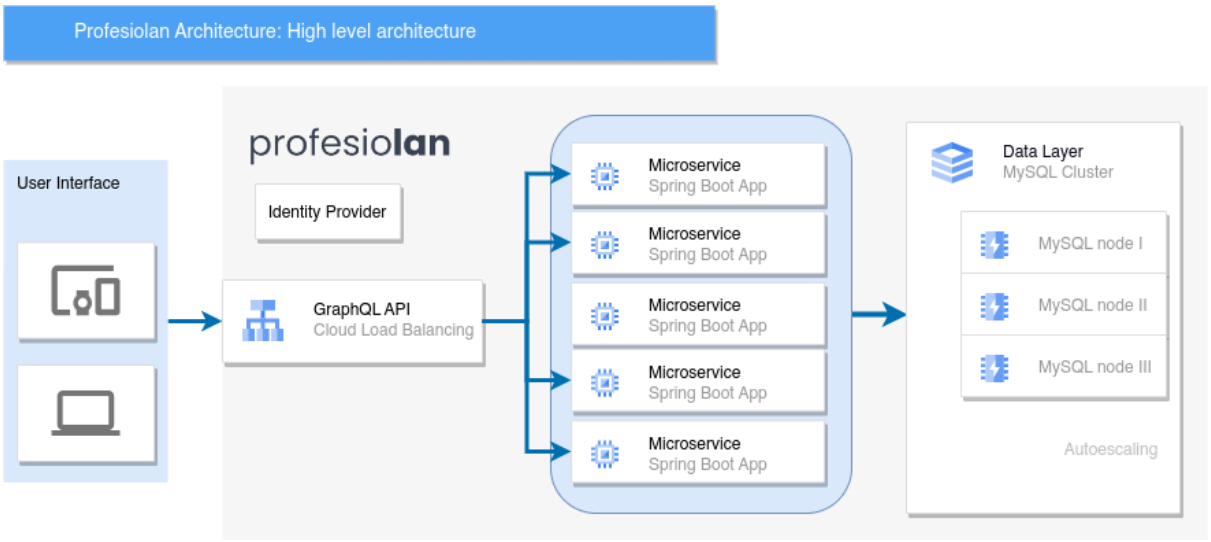

Figura 3: Resumen de la arquitectura de alto nivel. (Fuente: Elaboración propia).

A su izquierda, el módulo User Interface representa las aplicaciones web y móvil llamadas aplicaciones cliente, que interactúan con la API mediante llamadas HTTP a la dirección pública y única de la API.

Gran parte del reto tecnológico ha sido la convergencia de toda la cadena de servicio en un solo *endpoint*, que implementa un cortafuegos a nivel de aplicación y a nivel de red, realiza balanceo de cargas, ejecuta la redirección de la consulta parseada al microservicio correspondiente en cada caso, sirve la respuesta del microservicio en alta disponibilidad y con una latencia mínima, y recoge métricas y registros de todas las acciones ocurridas.

1.7.2. Modularidad y escalabilidad a través de una arquitectura en tres capas

El *backend* se compone a su vez de tres capas de abstracción. La capa de datos o *Data Layer*, la capa de negocio o *Core Layer*, y la capa API o *API Layer*. El diseño, desarrollo, despliegue y gestión de cada una de las capas supone per se tres proyectos diferentes, con tecnologías, lenguajes de programación, entornos, y riesgos o estándares de seguridad a aplicar diferentes en cada caso.

El propósito que ha motivado la mencionada arquitectura es el de garantizar una escalabilidad en el largo plazo mediante la modularización. De esta forma, queda obsoleta la arquitectura monolítica ejecutada en el producto mínimo viable de la empresa, para dar paso a un enfoque más adecuado al

estado de la misma, ofreciendo un producto que va a poder asumir una escala a nivel de volumen de servicio en el corto, medio e incluso el largo plazo.

Como se ha desarrollado a lo largo del *Apartado 1.6 Análisis de alternativas*, para la capa de base de datos se ha utilizado un cluster basado en SQL provisto por MySQL, en la capa de negocio, se han desarrollado aplicaciones autocontenidas Java Spring Boot como stack principal, y en la capa API, se ha diseñado y desplegado un servidor bajo el paradigma GraphQL. Asimismo, el entorno común a todo el proyecto es Amazon Web Services.

1.7.3. Amazon Web Services como habilitador y plataforma de despliegue

Resulta de interés mencionar que, si bien se presenta la tecnología desarrollada como una arquitectura basada en tres capas de abstracción, estas en realidad se traducen en el desarrollo y despliegue de servicios autocontenidos del propio AWS.

De este modo, la capa de negocio o *Core Layer* pasa a ser un grupo de aplicaciones Lambda desplegadas en entornos distribuidos. La API basada en GraphQL se ha desplegado sobre el servicio AppSync y, el cluster de base de datos sobre el servicio Aurora para bases de datos relacionales.

Además de estos, se utilizan otros tantos servicios indirectos necesarios para alcanzar la versión final del producto. Algunos de estos son los siguientes: VPC para la creación de redes virtuales privadas donde desplegar recursos sensibles, Cognito para la oferta de autenticación y autorización mediante el despliegue y aprovisionamiento de un servidor de autorización Oauth 2.0, Cloudwatch para la creación granular de métricas de salud, e IAM para la gestión integral de los roles y del alcance de todos los servicios distribuidos de la solución. Todos ellos están presentados en el *Anexo I: Descripción de la solución realizada: Diseño de alto nivel*.

Para finalizar, se presenta en la F*igura 4: Resumen de la arquitectura a nivel servicios PaaS de AWS*, la arquitectura desarrollada, esta vez en términos de los servicios utilizados en el ecosistema de computación en la nube de Amazon.

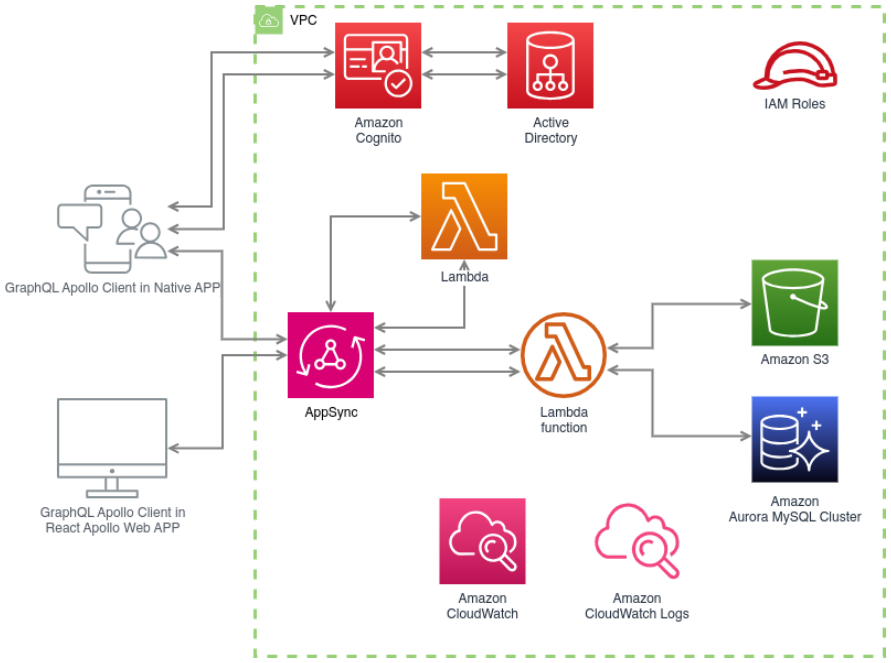

Figura 4: Resumen de la arquitectura a nivel servicios PaaS de AWS. (Fuente: Elaboración propia).

Para una más detallada descripción de todos y cada uno de los componentes del sistema se recomienda la lectura del *Anexo I. Descripción de la solución realizada: Diseño de alto nivel* y del *Anexo II. Descripción de la solución realizada: Diseño de bajo nivel.*

1.7.4. Introducción a los componentes

En este apartado se procede a resumir el alcance de cada uno de los tres componentes de abstracción desarrollados en la solución.

1.7.4.1. Data Layer

La capa de datos está basada en un cluster MySQL, desplegado sobre el servicio AWS Aurora. Durante este proyecto, se ha diseñado la base de datos, se ha configurado la naturaleza del cluster (incluyendo aspectos de alta relevancia como es el autoescalado), se ha realizado una población inicial de datos *dummy* y se ha testeado el rendimiento de la solución mediante técnicas avanzadas de auditoría de rendimiento.

En concreto, se han desarrollado y desplegado nodos con una capacidad mínima de 2GiB de RAM en una Londres, con la versión 5.7.2.11.3 de MySQL, que autoescalan en situación de congestión, que

únicamente son accesibles por la capa de negocio (inaccesibles públicamente de forma absoluta) y que generan una copia de seguridad cada 24 horas.

A continuación se presenta en la Figura 5 el diagrama entidad-relación (ER) diseñado.

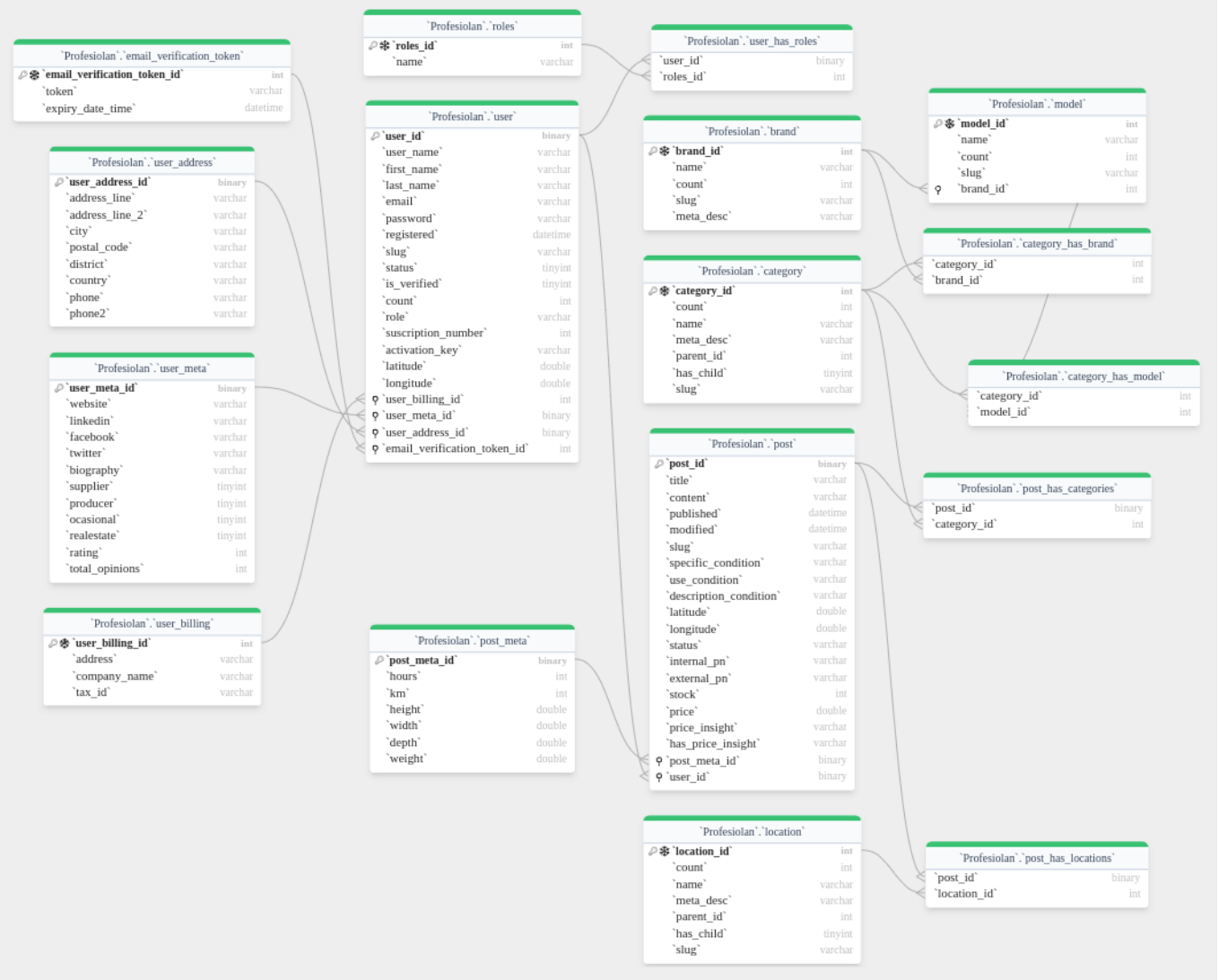

Figura 5: Diagrama ER. (Fuente: Elaboración propia).

1.7.4.2. Core Layer

La totalidad de la lógica de negocio del proyecto Profesiolan está basada en una arquitectura distribuida y modular de microservicios basados en aplicaciones Java Spring Boot, desplegadas en contenedores AWS Lambda.

En concreto, se tratan de aplicaciones autocontenidas desarrolladas en Java 11, con una memoria RAM de 512 MiB (potencialmente expandibles hasta los 10 GiB), y que una vez más son inaccesibles de forma pública; el único acceso habilitado es para la API.

Una vez definido el alcance y modo, se procede a listar y resumir brevemente cada una de las aplicaciones desplegadas. Los cuatro microservicios son el servicio Search, User, Stock y Mgmt.

El microservicio Search es el módulo encargado de la garantización de todas las tareas de búsqueda, filtrado y navegación principal de usuario. Se trata del microservicio con mayor uso. Se presenta en la *Tabla 6* las funcionalidades a cubrir por este módulo.

	Llamada	Uso
SearchService	getBrandById	MGMT
	getBrandsByCategoryId	MGMT
	getModelById	MGMT
	getModelsByBrandAndCategory	Producto
	getCategoryById	Producto
	getCategoriesByPostId	Producto
	adminGetPostById	Producto
	adminGetPostsByUser	MGMT
	getMyPosts	Producto
	getPostsByFilters	Producto
	getPostById	Producto

Tabla 6: Funcionalidades cubiertas por el microservicio de búsqueda. (Fuente: Elaboración propia).

En segundo lugar, el microservicio User unifica la totalidad de la gestión de usuarios. Desde actualización de información de las cuentas de los clientes, hasta el borrado de cuentas y toda la información relativa, hasta la modificación del plan contratado en Profesiolan. Se presenta en la *Tabla 7* las funcionalidades a cubrir por este módulo.

	Llamada	Uso
UserService	adminCreateUserBilling	MGMT
	adminUpdateUserBilling	MGMT
	adminDeleteUserBilling	MGMT

	adminCreateUserMeta	MGMT
	adminUpdateUserMeta	MGMT
	adminDeleteUserMeta	MGMT
	adminCreateUserAddress	MGMT
	adminUpdateUserAddress	MGMT
	adminDeleteUserAddress	MGMT
	adminUpdateUser	MGMT
	updateUser	Producto
	adminGetUser	MGMT
	getUserMetaByUserId	MGMT
	getUserAddressByUserId	MGMT
	getUserBillingByUserId	MGMT
	hasUserMeta	Producto
	hasUserAddress	Producto
	hasUserBilling	Producto
	getUserByUserId	Producto

Tabla 7: Funcionalidades cubiertas por el microservicio de gestión de usuarios. (Fuente: Elaboración propia).

El microservicio Stock, por su parte, es el encargado de la gestión de todo el stock publicado en Profesiolan. Se trata del módulo que permite la publicación, actualización y el borrado de los activos publicados por las empresas registradas. Se presenta en la *Tabla 8* las funcionalidades a cubrir por este módulo.

	Llamada	Uso
StockService	adminCreatePost	MGMT
	adminUpdatePost	MGMT
	adminDeletePost	MGMT
	adminApprovePost	MGMT
	publishPost	Producto
	updatePost	Producto
	deletePost	Producto

Tabla 8: Funcionalidades cubiertas por el microservicio de gestión del stock. (Fuente: Elaboración propia).

Finalmente, el microservicio Mgmt, como su propio nombre indica, es un microservicio de administración que ofrece métodos de gestión a usuarios con el rol de administradores o editores. Estas gestiones suelen ser relativas al mantenimiento de las categorías, taxonomías y filtros. Se presenta en la *Tabla 9* las funcionalidades a cubrir por este módulo.

	Llamada	Uso
MgmtService	adminCreateBrand	MGMT
	adminUpdateBrand	MGMT
	adminDeleteBrand	MGMT
	adminCreateCategory	MGMT
	adminUpdateCategory	MGMT
	adminDeleteCategory	MGMT
	adminCreateLocation	MGMT
	adminUpdateLocation	MGMT
	adminDeleteLocation	MGMT
	adminCreateModel	MGMT
	adminUpdateModel	MGMT
	adminDeleteModel	MGMT
	getCategoryById	MGMT
	getLocationById	MGMT
	getBrandById	MGMT

Tabla 9: Funcionalidades cubiertas por el microservicio de gestión general. (Fuente: Elaboración propia).

1.7.4.3. API Layer

La capa API del proyecto Profesiolan está basada en tecnología GraphQL, y ha sido desplegada sobre el servicio AWS AppSync.

La API es accesible mediante el siguiente *endpoint* público: https://lg2uz3spjjczzpalaaanio7ndi.appsync-api.eu-west-2.amazonaws.com/graphql, tiene habilitada autenticación y autorización mediante Oauth2 y tokens JWT, también ofrece métodos de uso público accesibles mediante *API-Keys*, se garantiza la alta disponibilidad e incluso implementa un *firewall* de

nivel de red y de aplicación. Además, permite la implementación de cachés para la reducción del tiempo de servicio.

La razón subyacente relacionada al desarrollo de diferentes políticas de acceso se basa en la necesidad de poder ofrecer servicios de navegación limitada por el producto para usuarios no registrados, y la necesidad de ofrecer servicios avanzados para usuarios autenticados. Para ello, en el primer caso se hace uso de un sistema de autorización de peticiones a través de las mencionadas *API-Keys*, ofrecidas a la API por el propio cliente web Profesiolan, y que serán consultas relacionadas con peticiones de filtrado y navegación por los diferentes productos. En contra, se utilizan los sistemas Oauth2 y los tokens JWT para poder ofrecer métodos de cliente como la actualización de los datos de perfil del usuario, o la publicación, edición, o borrado de productos.

En lo que respecta a la seguridad implementada, tanto mediante el *firewall* de nivel de red como el de aplicación, el objetivo es siempre el de ofrecer la máxima seguridad y la máxima protección de la API. Es por ello necesario crear mecanismos que combatan los ataques de fuerza bruta, de denegación de servicio, de inyección SQL, y un largo etcétera. En esta dimensión, el desarrollo en el ecosistema AWS ha sido clave para poder implementar estas medidas directamente, ya que vienen incorporadas en las capas inferiores de la pila de protocolos de los IaaS.

La implementación de caché también tiene su particular capítulo en el Anexo relativo al análisis del rendimiento de la solución desarrollada. De forma resumida, se ha demostrado la capacidad de activación de una infraestructura de caché que permite reducir los tiempos relacionados con las consultas de forma considerable y perceptible para el usuario final. Esto ha sido posible mediante el diseño de una política de caché a nivel API y con granularidad de petición; pudiendo reducir al máximo los retardos asociados al filtrado de productos (tarea crucial en un marketplace), mientras que se limita para tareas como el listado de información de una empresa vendedora concreta. Esta capacidad de decisión ofrece al arquitecto de la solución abaratar costes en una de las materias más costosas de la solución.

Debido al diseño desarrollado, el mencionado *endpoint* permite, a un usuario totalmente nuevo, entender a la perfección todas las características, funcionalidades, y las diferentes posibilidades de entrada y salida. Esto es así debido a la utilización de GraphQL como tecnología habilitadora, que ofrece una plantilla de versiones y anotaciones (accesible mediante cualquier entorno de desarrollo de APIs que consuma GraphQL como Postman) donde se listan todas las características.

2. Metodología

En este segundo bloque se desarrolla la metodología de planificación seguida, exponiendo el desglose de tareas, la agrupación de las mismas en fases y los recursos asignados en cada caso.

2.1. Descripción de los recursos

A continuación se describe la relación de los recursos destinados al proyecto, tanto los recursos humanos de los que dispone el proyecto, como de los recursos hardware y software explotados.

2.1.1. Recursos Humanos

El grupo de trabajo es común para todas las tareas definidas para completar la totalidad del proyecto, y se define en la *Tabla 10: Definición de recursos humanos para el desarrollo del proyecto.*

ID recurso	Nombre y Apellidos	Responsabilidad	Horas del contrato	Jornadas completas contrato	Coste horario bruto
R.H.1	Ander Hospital	Ingeniero de Telecomunicaciones	1856	232	23,50 €

Tabla 10: Definición de recursos humanos para el desarrollo del proyecto. (Fuente: Elaboración propia).

El recurso humano dispone de un sueldo bruto en la empresa por valor de 23.50 € hora. Tras las 232 jornadas de trabajo completadas, el recurso ha costado a la empresa 43.616,00 € brutos.

2.1.2. Recursos Materiales

En los recursos materiales se han incluído todos los programas informáticos de licencia no libre, así como los ordenadores y hasta el lugar de trabajo donde se ha llevado a cabo la actividad.

ID recurso	Tipo	Descripción
R.M.1	Licencias Software	Amazon Web Services
R.M.2	Licencias Software	Google Cloud
R.M.3	Licencias Software	Okta
R.M.4	Software de Gestión	Slack
R.M.5	Software de Gestión	Trello
R.M.6	Licencias Software	IntelliJ
R.M.7	Equipos Informáticos	MSI Prestige 15 (1 Unidad)
R.M.8	Mobiliario y medios auxiliares	Mobiliario de oficina misc.
R.M.9	Activos fijos inmobiliarios	Oficina de empresa

Tabla 11: Definición de recursos humanos para el desarrollo del proyecto. (Fuente: Elaboración propia).

El resumen de costes se define a lo largo del Apartado 4.1.2: Recursos materiales.

2.2. Descripción de las tareas, fases y procedimientos

El proyecto tiene inicio el 20 de septiembre del 2022, y final el 26 de agosto del 2023. En total, se han diseñado, planificado, llevado a cabo y terminado con éxito 109 tareas, que se han ejecutado correctamente en 1856 horas, por el único recurso disponible.

A continuación, se lista en *Tabla 12: Definición de tareas y paquetes de trabajo. Fuente: Elaboración Propia*, el resumen de todas las tareas y grupos de tareas ejecutadas durante el proyecto de ingeniería relacionado con el presente Trabajo Fin de Máster.

En la misma, las tareas identificadas como hitos, anotadas con el prefijo *H*, no tienen asociada una duración en tiempo, ya que conceptualmente deben darse por cumplidas y correspondientemente emitidas dentro del plazo del grupo de tareas que la incluye.

ID	Nombre de la tarea	Fecha Inicio	Fecha Final	Duración (días laborables)
G.T.1	**Primera revisión de las especificaciones**	**9/20/22**	**10/12/22**	**17**

T.1.1	Estudio de las necesidades a partir del MVC	9/20/22	9/27/22	6
T.1.2	Definición del alcance de las necesidades no cubiertas	9/28/22	10/4/22	5
T.1.3	Constatación con clientes	10/5/22	10/6/22	2
T.1.4	Revisión y clasificación de las necesidades	10/7/22	10/12/22	4
H1	Documento de Especificaciones en Versión 1	10/13/22	10/13/22	S/T
G.T.2	**Análisis de Viabilidad del Proyecto**	**10/14/22**	**10/14/22**	**1**
T2.1	Revisión del Documento de Especificaciones a nivel Negocio	10/14/22	10/14/22	1
H2	El proyecto es en primer lugar invertible	10/17/22	10/17/22	0
G.T.3	**Análisis de alternativas**	**10/19/22**	**2/23/23**	**92**
G.T.3.1	**Estudio del Stack**	**10/19/22**	**2/23/23**	**92**
G.T.3.1.1	**API: Estudio práctico de la mejor alternativa**	**11/29/22**	**1/23/23**	**40**
T.3.1.1.1	Rest	11/29/22	1/23/23	40
T3.1.1.2	GraphQL	11/29/22	1/23/23	40
G.T.3.1.2	**Framework: Estudio práctico de la mejor alternativa**	**10/19/22**	**11/28/22**	**29**
T.3.1.2.1	Java EE	10/19/22	10/26/22	6
T.3.1.2.2	Java Spring Boot	10/27/22	11/28/22	23
G.T.3.1.3	**DDBB: Estudio práctico de la mejor alternativa**	**11/29/22**	**12/14/22**	**12**
T.3.1.3.1	MySQL	11/29/22	12/2/22	4
T.3.1.3.2	MsSQL	12/5/22	12/8/22	4
T.3.1.3.3	DynamoDB (NoDB approach)	12/9/22	12/14/22	4
G.T.3.1.4	**Seguridad: Estudio práctico de la mejor alternativa**	**12/27/22**	**1/19/23**	**18**
T.3.1.4.1	Spring Security	12/27/22	1/5/23	8
T.3.1.4.2	Provista por Cloud	1/6/23	1/19/23	10
G.T.3.1.5	**Autenticación y Autorización: Estudio práctico de la mejor alternativa**	**1/24/23**	**2/23/23**	**23**
T.3.1.5.1	Spring Security	1/24/23	2/6/23	10
T.3.1.5.2	Provista por Cloud	2/7/23	2/21/23	11
T.3.1.5.3	Okta	2/22/23	2/23/23	2
H.3.1	Estudio del stack	2/24/23	2/24/23	0
G.T.3.2	**Deployment**	**11/11/22**	**2/22/23**	**74**
G.T.3.2.1	**Cloud PaaS**	**11/17/22**	**2/22/23**	**70**
G.T.3.2.1.1	**Google Cloud como entorno de despliegue end-to-end**	**11/17/22**	**12/12/22**	**18**
T.3.2.1.1.1	Análisis de la solución concreta	11/17/22	11/28/22	8

T.3.2.1.1.2	Análisis del pricing	11/29/22	12/5/22	5
T.3.2.1.1.3	Análisis de benchmarks, idoneidad y proyección	12/6/22	12/6/22	1
T.3.2.1.1.4	Valoración de la solución	12/7/22	12/12/22	4
G.T.3.2.1.2	**AWS como entorno de despliegue end-to-end**	**12/14/22**	**2/22/23**	**51**
T.3.2.1.2.1	Análisis de EC2 y prueba de concepto	12/14/22	12/27/22	10
T.3.2.1.2.2	Análisis de Beanstalk y prueba de concepto	12/28/22	12/30/22	3
T.3.2.1.2.3	Análisis de Lambda + Appsync y prueba de concepto	1/2/23	2/9/23	29
T.3.2.1.2.4	Análisis agregado del pricing	2/10/23	2/10/23	1
T.3.2.1.2.5	Análisis agregado de los benchmarks de cada solución	2/13/23	2/15/23	3
T.3.2.1.2.6	Análisis agregado de la proyección de AWS como PaaS en largo plazo	2/16/23	2/16/23	1
T.3.2.1.2.7	Clasificación de la mejor arquitectura y ballpark de la solución	2/17/23	2/22/23	4
G.T.3.2.2	**Hosteado in-house**	**11/11/22**	**1/23/23**	**52**
T.3.2.2.1	Análisis de necesidades	11/11/22	11/21/22	7
T.3.2.2.2	Definición lógica de la arquitectura	11/22/22	12/19/22	20
T.3.2.2.3	Análisis preliminar de rendimiento potencial	12/20/22	1/2/23	10
T.3.2.2.4	Análisis de escalabilidad	1/3/23	1/10/23	6
T.3.2.2.5	Análisis de seguridad	1/11/23	1/23/23	9
H.3.2	Estudio del deployment	2/23/23	2/23/23	0
G.T.4	**Diseño de la solución**	**1/24/23**	**2/28/23**	**26**
T.4.1	Diseño lógico de alto nivel	1/24/23	2/28/23	26
T.4.2	Definición de los microservicios	2/14/23	2/27/23	10
T.4.3	Diseño finalizado con una primera revisión del alcance	3/1/23	3/1/23	0
G.T.5	**PoC AWS**	**11/29/22**	**4/27/23**	**108**
T.5.1	Preparación entorno: Creación de roles de desarrollo	3/3/23	3/9/23	5
T.5.2	Preparación entorno: Creación de políticas de acceso	3/10/23	3/13/23	2
T.5.3	Preparación entorno: Definición de las zonas de seguridad	3/14/23	3/20/23	5
T.5.4	Preparación entorno: Diseño operativa DevOps	3/21/23	3/21/23	1
T.5.5	Desarrollo de la definición del esquema GraphQL	12/23/22	3/27/23	67
T.5.6	Publicación de primera versión API GraphQL en AppSync	3/28/23	3/28/23	0
T.5.7	Integración AppSync - Microservicio Lambda	3/28/23	3/28/23	1
T.5.8	Desarrollo de la base de datos MySQL 5.7	11/29/22	4/4/23	91
T.5.9	Publicación de primera versión DDBB en Cluster MySQL	4/5/23	4/5/23	0

T.5.10	Integración Microservicio Lambda - Cluster MySQL	4/5/23	4/24/23	14
T.5.11	Estudio preliminar de latencias, cuellos de botellas y obtención de KPIs	4/25/23	4/27/23	3
H.5	PoC Finalizada	4/28/23	4/28/23	0
G.T.6	**Sprint 1: Microservicio Search**	**4/28/23**	**5/30/23**	**23**
T.6.1	Planificación del Sprint	4/28/23	5/1/23	2
T.6.2	Desarrollo de la aplicación Spring Boot	5/2/23	5/16/23	11
T.6.3	Despliegue e integración con AppSync y MySQL	5/17/23	5/22/23	4
T.6.4	Check-in y revisión por negocio	5/23/23	5/26/23	4
T.6.5	Revisiones y cambios	5/29/23	5/30/23	2
H.6	Despliegue válido	5/31/23	5/31/23	0
G.T.7	**Sprint 2: Microservicio Post**	**5/31/23**	**6/20/23**	**15**
T.7.1	Planificación del Sprint	5/31/23	5/31/23	1
T.7.2	Desarrollo de la aplicación Spring Boot	6/1/23	6/8/23	6
T.7.3	Despliegue e integración con AppSync y MySQL	6/9/23	6/9/23	1
T.7.4	Check-in y revisión por negocio	6/12/23	6/13/23	2
T.7.5	Revisiones y cambios	6/14/23	6/20/23	5
H.7	Despliegue válido	6/21/23	6/21/23	0
G.T.8	**Sprint 3: Microservicio User**	**6/21/23**	**7/25/23**	**25**
T.8.1	Planificación del Sprint	6/21/23	6/21/23	1
T.8.2	Desarrollo de la aplicación Spring Boot	6/22/23	7/5/23	10
T.8.3	Desarrollo del User Pool para autenticación y autorización	6/22/23	7/11/23	14
T.8.4	Integración AWS Cognito (proveedor de Auth. de AWS)	7/12/23	7/17/23	4
T.8.5	Creación de dummy data e inserción en la base de datos	7/18/23	7/18/23	1
T.8.6	Despliegue e integración de la aplicación con AppSync y MySQL	7/19/23	7/19/23	1
T.8.7	Check-in y revisión por negocio	7/20/23	7/21/23	2
T.8.8	Revisiones y cambios	7/24/23	7/24/23	1
H.8	Despliegue válido	7/25/23	7/25/23	1
G.T.9	**Estudio del Rendimiento**	**7/26/23**	**8/16/23**	**16**
T.9.1	Desarrollo de tests en PostMan	7/26/23	7/31/23	4
T.9.2	Desarrollo de AWS X-Ray para obtención de métricas avanzadas	8/1/23	8/1/23	1
G.T.9.1	**Análisis AppSync**	**8/2/23**	**8/8/23**	**5**
T.9.1.1	Análiticas per-resolver vs batch-resolver	8/2/23	8/3/23	2

T.9.1.2	Impacto de la implementación de Cache a nivel resolver	8/4/23	8/4/23	1
T.9.1.3	Impacto de la implementación de Cache a nivel usuario	8/7/23	8/7/23	1
T.9.1.4	Estudio del rendimiento en la escalabilidad, ¿Infinito Ancho de Banda?	8/8/23	8/8/23	1
G.T.9.2	**Análisis Microservicios**	**7/26/23**	**8/3/23**	**7**
T.9.2.1	Análiticas en función de la RAM dedicada	7/26/23	8/1/23	5
T.9.2.2	Estudio del Cold Start y la implicación de Snapshots	8/2/23	8/2/23	1
T.9.2.3	Estudio del rendimiento en la escalabilidad, creación nuevos clusteres y multiregión	8/3/23	8/3/23	1
G.T.9.3	**Análisis Cluster MySQL**	**7/26/23**	**8/16/23**	**16**
T.9.3.1	Analíticas en función de la RAM dedicada	7/26/23	7/28/23	3
T.9.3.2	Estudio de la DATA API, pros y contras vs JDBC, JPA, Hibernate...	7/31/23	8/8/23	7
T.9.3.3	Estudio del rendimiento en la escalabilidad, creación nuevos clusteres y multiregión	8/9/23	8/16/23	6
T.10	Redacción del Documento Memoria	1/2/23	8/25/23	170

Tabla 12: Definición de tareas y paquetes de trabajo. (Fuente: Elaboración propia).

2.3. Diagrama de Gantt

El diagrama de Gantt está basado en los paquetes de paquetes de tareas definidos en *2.2: Descripción de las tareas, fases y procedimientos*. El documento del diagrama de Gantt se adjunta como *Anexo IV. Diagrama Gantt*.

No obstante, también se incluye en la *Figura 6: Resumen del diagrama Gantt*, un resumen visual por grupos de tareas. En cualquier caso, para ver el diagrama completo con total definición se recomienda la lectura del Anexo IV. En él, se presentan con detalle todas las tareas, las fechas de inicio y fin, y se puede comprobar el estado de finalización del proyecto.

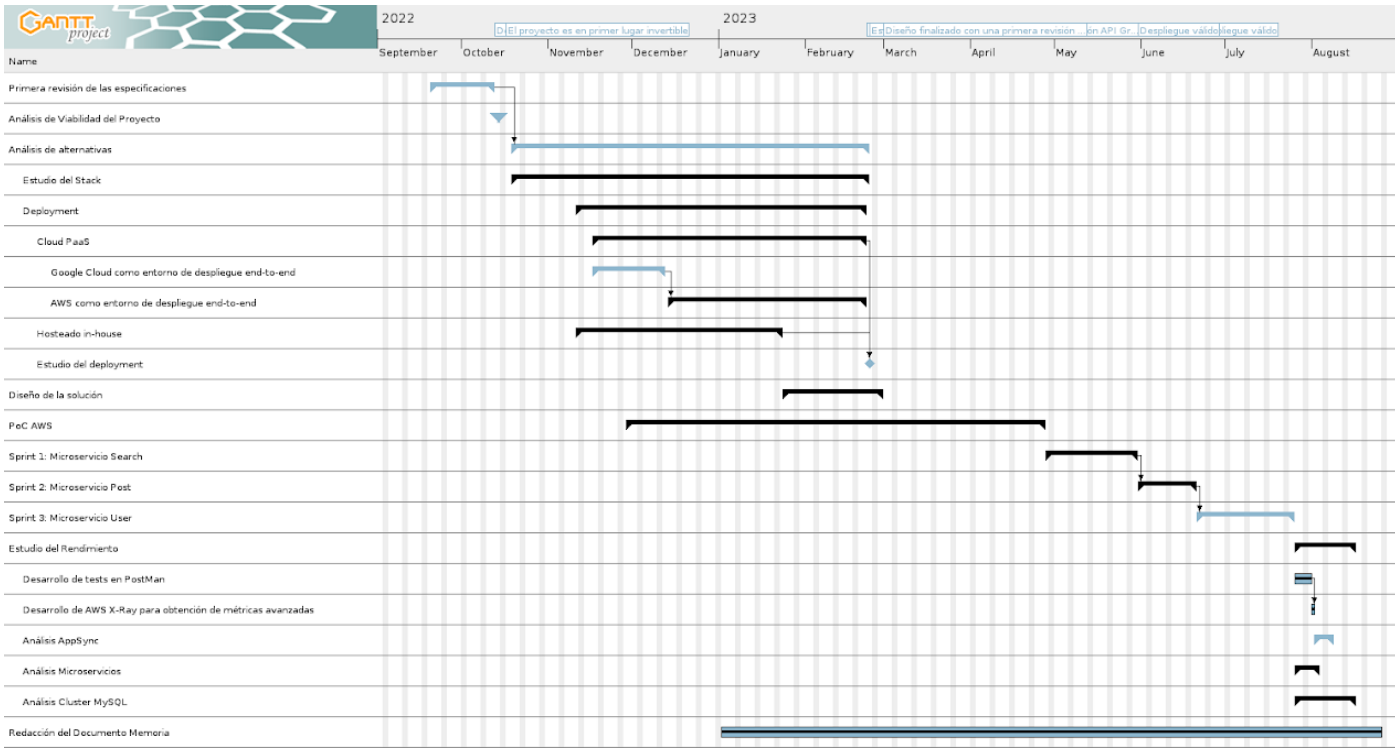

Figura 6: Resumen del diagrama Gantt. (Fuente: Elaboración propia).

2.4. Presentación de los resultados: Análisis de rendimiento

La API bajo la cual se fundamente un producto software de uso masivo y que pretende escalar a las millones de sesiones de forma mensual orgánicamente, debe ser robusta, resiliente, rápida y segura. Es precisamente para asegurar que el sistema desarrollado es resiliente, rápido y robusto, que se ha diseñado un plan de análisis de rendimiento exhaustivo.

El plan de rendimiento operado pretende varios objetivos. Primeramente, se propone encontrar, listar y definir todos aquellos parámetros a seguir durante la vida útil del producto. Estos deben ser parámetros que resuman la salúd del producto en todo momento, y que permitan la rápida extracción de conclusiones mediante su consumo.

En segundo lugar, propone poner números a cada una de las métricas. De modo que, simulando diferentes entornos de carga y repitiendo el experimento un número suficientemente alto de veces, se cuantifican todos y cada uno de los identificadores de salud.

Consecuentemente, propone exponer los cuellos de botella que existen en todo sistema, para su correspondiente evaluación y estudio de la posibilidad de mejora.

Finalmente, pretende simular entornos de congestión severos para el estudio del comportamiento de la arquitectura. Escenarios con 100 sesiones simultáneas realizando una navegación fidedigna o incluso la simulación de 100 publicaciones de activos en el marketplace de forma simultánea también.

Para poder ofrecer un mayor contexto antes de la presentación de los resultados, se lista la planificación del análisis de rendimiento desarrollado. Una vez más, cabe destacar que la totalidad del análisis de rendimiento ejecutado se presenta en el *Anexo III*.

2.4.1. Presentación de los flujos

Para una correcta auditoría del rendimiento de la solución realizada, las pruebas a realizar deben ser las mismas a las que se enfrente el producto en producción: usuarios navegando, creando y actualizando cuentas de usuario, gestionando sus stocks, e iniciando transacciones de compra.

Para todo ello se han desarrollado tres tipos de flujos: uno relativo a la navegación nominal de un usuario por la página de activos (llamado Flujo 1), uno relativo a la gestión de su información de un usuario registrado y autenticado (Flujo 2) y, por último, una publicación de activo por parte de otro usuario registrado, autenticado, y con un plan comercial acorde (Flujo 3).

Cada uno de estos flujos son, en realidad, una síntesis de llamadas API en el entorno JMeter de Apache. Por ejemplo, el primer flujo está compuesto de las siguientes acciones sintetizadas en sendas llamadas API:

- getCategoryById: Obtención de la información de una categoría durante el filtrado dado su identificador.
- getBrandsByCategoryId: Obtención de la información de todas las marcas asociadas a activos pertenecientes a una determinada categoría.
- getModelsByBrandAndCategory: Obtención de la información de todos los modelos asociados a una marca perteneciente a una categoría determinada. No es lo mismo, obtener

todos los modelos de furgonetas isotermas de la marca Mercedes-Benz a las cabezas tractoras de la misma marca.

- getPostsByFilters: Obtención del listado resumido y paginado de todos los activos que cumplen con la determinada consulta de filtro. Método que soporta además de paginación, ordenación y una fuerte carga de filtrado por taxonomías.
- getPostById: Obtención de toda la información pública relativa a un activo para el la generación de la página de compra de ese mismo activo.

A su vez, estos flujos han sido ejecutados en diferentes casos de test (a partir de ahora CTs). Estos CTs tienen como objetivo el de simular el comportamiento de los flujos ante determinados umbrales de volumen de uso. Estos flujos son los siguientes:

- CT1: Estrés leve en navegación: Ejecución del F1 por parte de 1 usuario cada 10 segundos. Estudio realizado en un bucle de 100 veces.
- CT2: Estrés nominal esperado en navegación: Ejecución del F1 por parte de 5 usuarios cada 10 segundos. Estudio realizado en un bucle de 20 veces.
- CT3: Estrés bruto en navegación: Ejecución del F1 por parte de 100 usuarios en un mismo instante basado en una ventana de 10 segundos.
- CT4: Estrés nominal en gestión del stock: Ejecución del F2 por parte de 5 usuarios cada 10 segundos.
- CT5: Estrés nominal en publicación: Ejecución del F3 por parte de un usuario cada 10 segundos. Estudio realizado en un bucle de 100 veces.
- CT6: Estrés bruto en publicación: Ejecución del F3 por parte de un 10 usuarios cada 10 segundos.

2.4.2. Visión resumida de los resultados

Tras la ejecución de los seis casos de test, los resultados obtenidos son prometedores y se encuentran por debajo de los 200 ms en el percentil 75, valores que objetivamente son positivos. Prueba de ello, se presenta un resumen de las latencias asociadas a las principales consultas de navegación mediante filtrado en el marketplace (caso de test 1) en la Figura 7: Resumen de latencias medias por *query* para navegación nominal, y la dispersión de las mismas en la *Figura 8: Presentación avanzada de resultados para navegación nominal.*

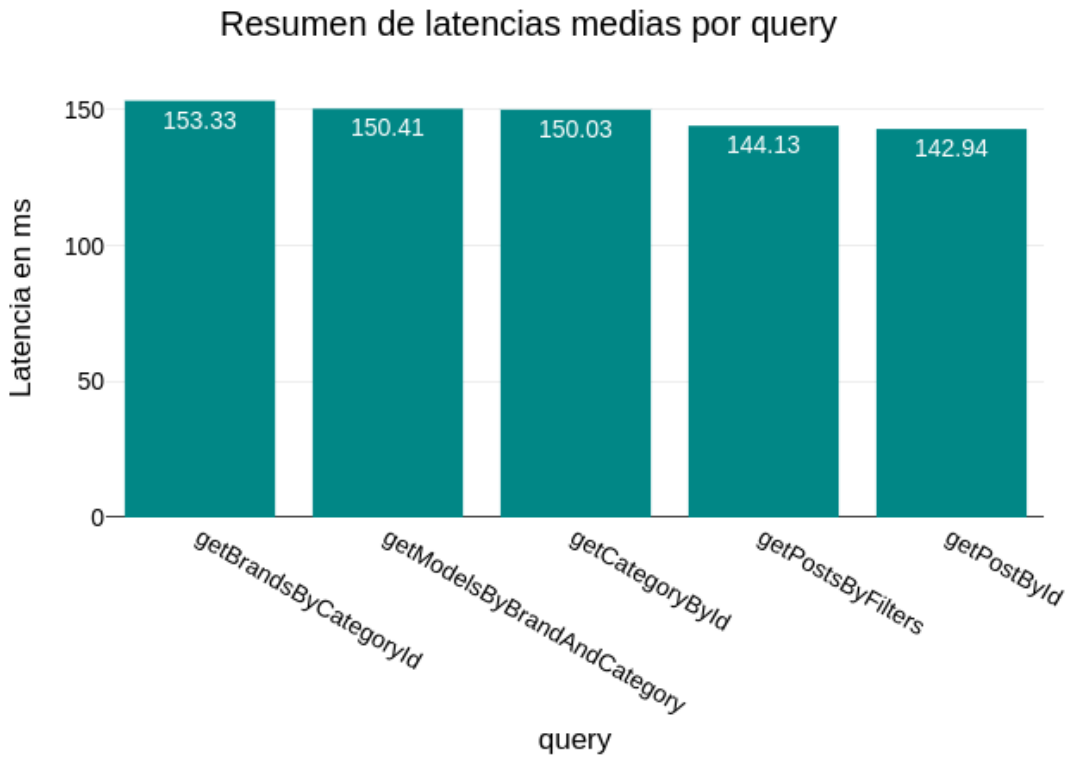

Figura 7: Resumen de latencias medias por query para navegación nominal. (Fuente: Elaboración propia).

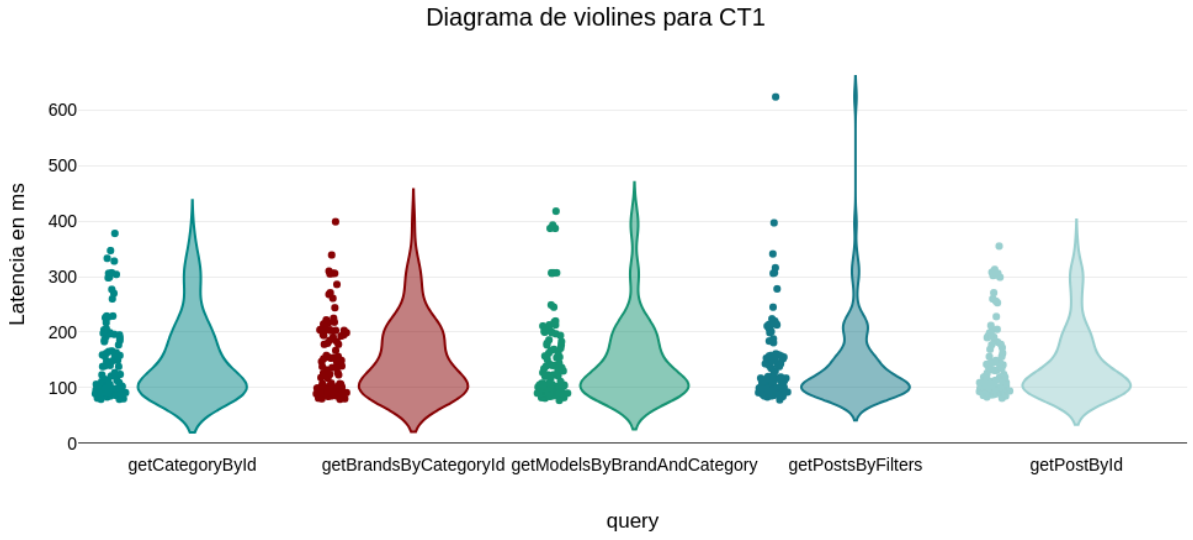

Figura 8: Presentación avanzada de resultados para navegación nominal. (Fuente: Elaboración propia).

En general, las solicitudes tienen tiempos de respuesta muy positivos y consistentes. La operación más rápida presenta un tiempo promedio de 142,94 ms, mientras que la operación más lenta 153,33 ms.

También se puede observar una distribución uniforme y positiva, con un centro de masas bajo y cercano al mínimo, y una pequeña variabilidad. Resulta interesante ver que tan solo existe, de 500 valores testeados, un único valor que supera los 600 ms, y menos de 8 han superado el límite de 400 ms.

Para dimensionar estos niveles de calidad de servicio, estos resultados tienen que ser analizados en el contexto de los requisitos del proyecto, que a su vez deben ser derivados de las tendencias del mercado. Se concluye la viabilidad del proyecto desarrollado.

Además, en el análisis presentado en el *Anexo III: Análisis de rendimiento* también se pueden observar otros dos estudios de alto interés; los casos de congestión simulados, y los resultados tras la configuración y activación de una memoria caché a nivel API. Los resultados se presentan en la *Figura 9: Presentación de resultados para simulación de congestión* y *Figura 10: Presentación de resultados para simulación de congestión con caché.*

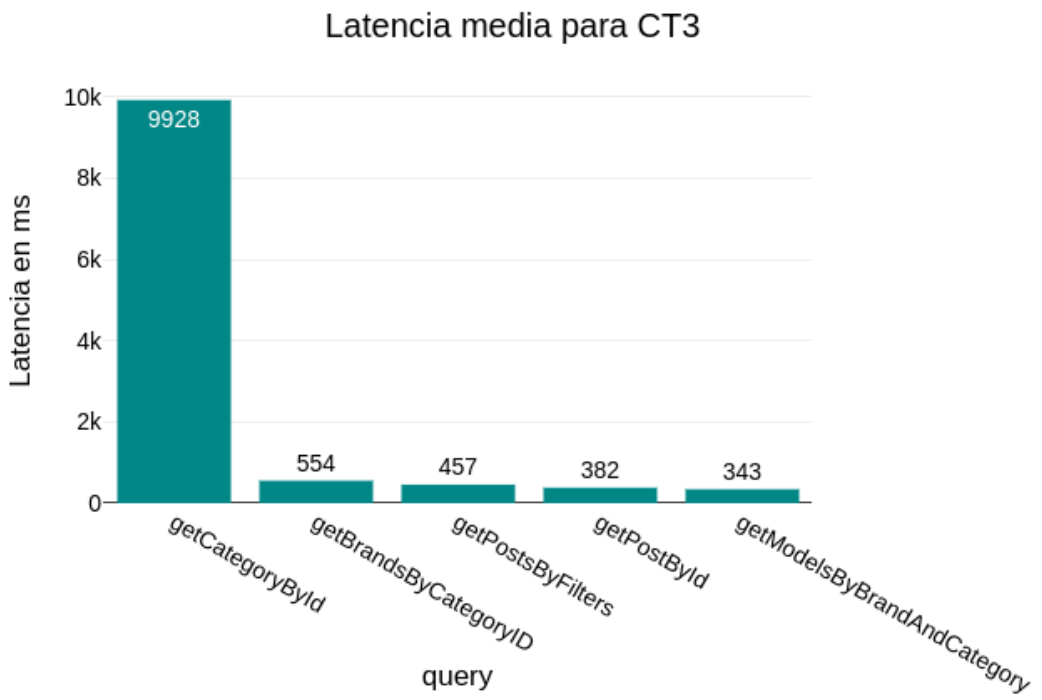

Figura 9: Presentación de resultados para simulación de congestión. (Fuente: Elaboración propia).

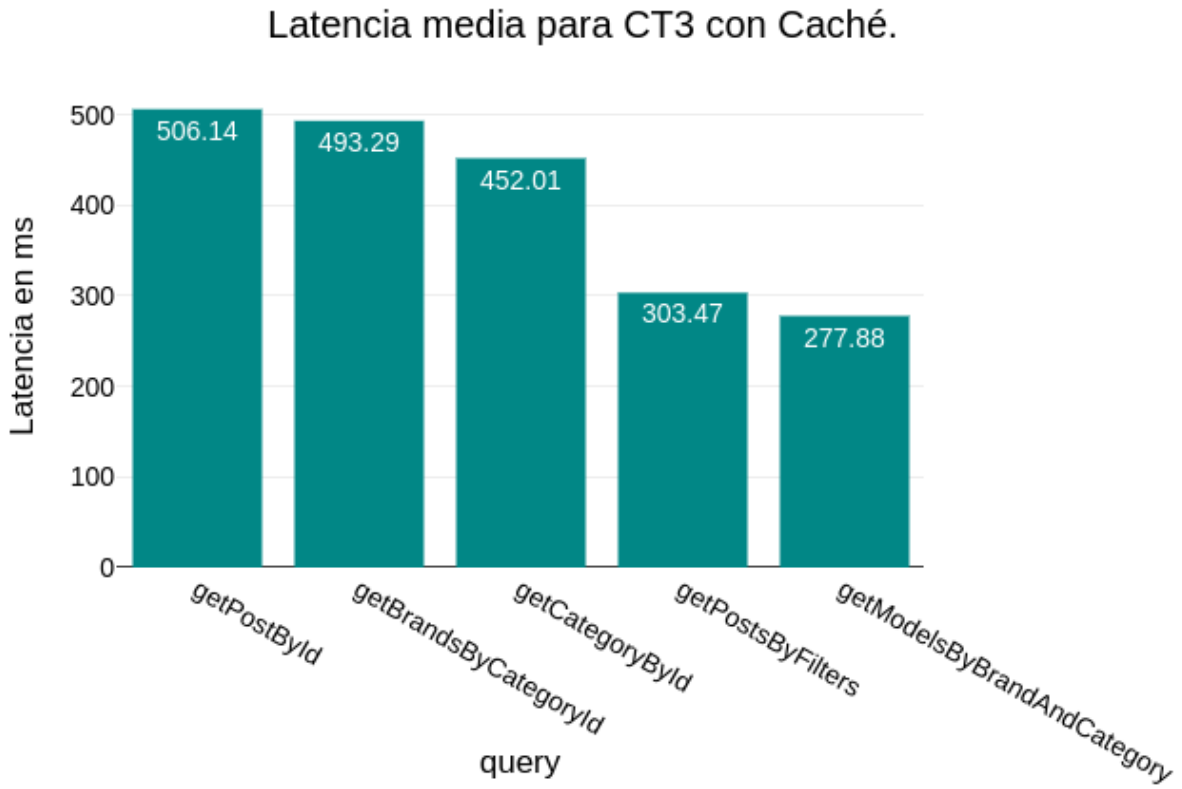

Figura 10: Presentación de resultados para simulación de congestión con caché. (Fuente: Elaboración propia).

Lo que se puede observar a simple vista es que, mientras sigue existiendo entradas en congestión como es lógico tras la entrada de 100 sesiones instantáneas que además realizan una navegación de forma simulada, la percepción media de tiempo de tiempo de latencia se ve reducida.

No obstante, la latencia media no es un indicador completo, ya que el valor medio se ve altamente influenciado por las entradas en congestión. Sin ir más lejos, una solicitud satisfecha tras entrada en congestión supera los 10 segundos, mientras que una sin congestión (y con el uso de caché) se satisface en menos de 150 ms.

Para solucionar este problema, se presenta en la *Figura 11: Presentación de resultados para simulación de congestión con caché (diagrama de violín)*, los diagramas de comportamiento. En ellos, se puede observar que la práctica totalidad de las consultas fueron satisfechas en menos de 200 ms (llegando a mínimos de 40 ms como se puede observar en el Anexo).

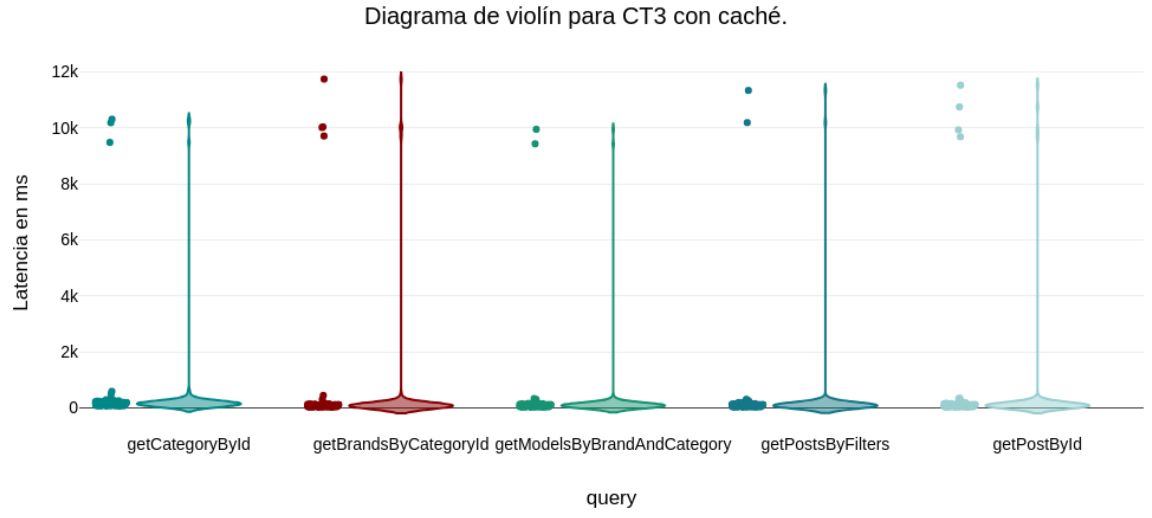

Figura 11: Presentación de resultados para simulación de congestión con caché (diagrama de violín). (Fuente: Elaboración propia).

2.4.3. Gestión de la salud de la solución durante la vida del producto

Para terminar con el presente resumen de la presentación de los resultados, se hace especial mención al desarrollo realizado para garantizar un correcto seguimiento de la salud de la solución en tiempo real durante la vida del producto.

En concreto, se han desarrollado paneles de administrador de muy alta utilidad y que sintetizan las métricas de seguimiento más relevantes en todo momento. En estos paneles se pueden observar métricas como el estado de la CPU, el número de llamadas a la API desglosadas en diferentes ventanas de tiempo, o incluso el número de congestiones que se han dado en el último día.

Dada la naturaleza modular de la solución, se han desarrollado tres paneles similares, correspondientes a las tres capas del sistema (API, Core, Data). Se presenta, en la *Figura 12: Panel de salúd de la capa API*, uno de estos paneles, correspondiente a la monitorización de la capa API.

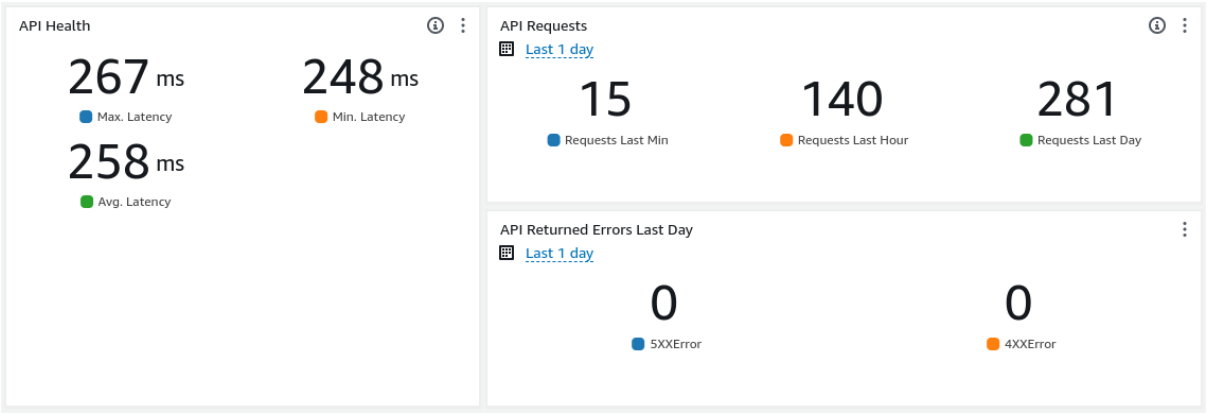

Figura 12: Panel de salúd de la capa API. (Fuente: Elaboración propia).

3. Aspectos económicos

A continuación, se presentan los aspectos económicos, incluyendo el presupuesto disponible, el ejecutado, un análisis de los costes de operación del producto mínimo viable, un análisis de los costes de operación del nuevo producto, y el análisis del retorno de la inversión (ROI) del proyecto.

3.1. Descripción del presupuesto disponible

La empresa disponía de 60.000 € para el desarrollo de la primera versión del refactoring del *backend*, o en otras palabras, la ejecución del alcance tecnológico desarrollado en el presente Trabajo Fin de Máster.

A continuación, se presenta la relación presupuestada aprobada a fecha de inicio del segundo semestre del año 2022.

3.1.1. Horas internas

Tal y como se define en el *Apartado 2.1.1: Recursos Humanos*, el grupo de horas internas comprende el trabajo de un trabajador, a jornada completa, desde el 20 de septiembre del 2022 hasta el 26 de agosto del 2023, ambos incluídos.

A un coste horario bruto de 23,50 €, la partida presupuestada para horas internas era de 43.616,00 € brutos.

3.1.2. Recursos materiales

Para el estudio del presupuesto asociado a los recursos materiales para la ejecución de la solución propuesta, se tuvo en cuenta de forma diferenciada aquellos gastos en activos fijos amortizables y no amortizables.

3.1.2.1. Gastos amortizables

A continuación, se presenta desglosado el estudio de la amortización del proyecto firmado en el presupuesto. En primer lugar, se define la duración del proyecto en el cuál se van a amortizar los activos fijos señalados.

Duración del Proyecto	
Duración total en días	**Duración total en horas**
232	1856

Tabla 13: Duración del proyecto. (Fuente: Elaboración propia).

Posteriormente, se procede a estimar la vida útil de estos activos fijos, en horas.

Vida útil			
Identificador del recurso	**Descripción del recurso**	**Duración en años**	**Duración total en horas**
R.M.7	Equipos Informáticos	4	8096
R.M.8	Mobiliario de oficina misc.	10	20240

Tabla 14: Estimación de la vida útil de los activos amortizables en el Proyecto. (Fuente: Elaboración propia).

Para terminar, se obtiene de forma sencilla el valor amortizado de cada uno de los activos fijos a estudiar, ponderando las horas del proyecto frente a la vida útil definida, tal y como se presenta en la *Tabla 15: Estudio de la amortización de los activos amortizables en el Proyecto.*

Amortizaciones					
Identificador del recurso	**Unidades**	**Precio unitario de compra (€)**	**Horas de uso**	**Coste unitario ponderado**	**Total amortización**
R.M.7	Equipos Informáticos	2.350,00 €	1856	538,73 €	676,28 €
R.M.8	Mobiliario de oficina misc.	1.500,00 €	1856	137,55 €	

Tabla 15: Estudio de la amortización de los activos amortizables en el Proyecto. (Fuente: Elaboración propia).

3.1.2.2. Gastos no amortizables

La relación de gastos no amortizables se define en la siguiente *Tabla 16: Gastos no amortizables*.

Gastos no amortizables		
Identificador del recurso	**Descripción**	**Coste total (€)**
R.M.1	Amazon Web Services	2.500,00 €
R.M.2	Google Cloud	350,00 €
R.M.3	Okta	100,00 €
R.M.4	Slack	50,00 €
R.M.5	Trello	50,00 €
R.M.6	IntelliJ	500,00 €
R.M.9	Oficina	0,00 €

Tabla 16: Gastos no amortizables. (Fuente: Elaboración propia).

3.1.2.3. Subcontrataciones

En el presupuesto no estaba incluída ninguna partida asociada a subcontrataciones.

3.2. Descripción del presupuesto ejecutado

En primer lugar, se procede a definir, en la *Tabla 17: Resumen del presupuesto económico total*, el resumen del presupuesto firmado previo inicio del proyecto.

Presupuesto total	
Concepto	**Resumen de costes (€)**
Horas internas	43.616,00 €
Amortizaciones	676,28 €
Gastos	3.550,00 €
Subcontrataciones	0,00 €
Subtotal 1	**47.842,28 €**
Indirectos (7,00%)	3.348,96 €
Subtotal 2	**51.191,24 €**
Imprevistos (5,00%)	2.559,56 €
Total	**53.750,80 €**

Tabla 17: Resumen del presupuesto económico total. (Fuente: Elaboración propia).

A continuación, se procede a definir, una vez concluído el proyecto, el presupuesto ejecutado.

3.2.1. Horas internas

El cumplimiento exitoso del proyecto se ha traducido en un cumplimiento estricto del presupuesto en materia de horas internas.

3.2.2. Recursos materiales

A continuación, se hace un listado del coste real asociado a cada uno de los recursos.

- **R.M.1:** El gasto ejecutado finalmente en AWS ha sido de 359,23 €. El ahorro sustancial frente al previsto en el presupuesto ha sido debido a un uso consciente y una gestión prudente de los recursos disponibles para cada servicio. AWS ofrece niveles de servicio gratuitos bajo unos determinados umbrales de uso y siempre asociados a unas asignaciones con menores prestaciones.

- **R.M.2:** El gasto ejecutado finalmente en Google Cloud ha sido de 0,00 €, ya que no se han superado los umbrales de consumo en ninguno de los servicios utilizados.
- **R.M.3, R.M.4, R.M.5:** El gasto ejecutado finalmente en Okta, Slack y Trello ha sido de 0,00 €, ya que se han utilizado en todo momento los planes freemium en cada caso.
- **R.M.6:** El gasto ejecutado finalmente en el IDE, IntelliJ Idea, ha sido de 0,00 €, ya que se disponía de una licencia de estudiante gratuita.
- **R.M.7, R.M.8:** El gasto ejecutado ha sido el mismo al previsto en el presupuesto debido a que el proyecto no se ha alargado en tiempo.
- **R.M.9**: El gasto en oficinas ha sido 0,00 € debido a subvenciones clave y al trabajo remoto.

3.2.3. Resumen del coste total del proyecto

En la *Tabla 18: Resumen del coste total del proyecto*, se resume el presupuesto actualizado una vez concluido con éxito el proyecto.

Presupuesto total	
Concepto	**Resumen de costes (€)**
Horas internas	43.616,00 €
Amortizaciones	676,28 €
Gastos	359,23 €
Subcontrataciones	0,00 €
Subtotal 1	**44.651,51 €**
Indirectos	0,00 €
Subtotal 2	**44.651,51 €**
Imprevistos	0,00 €
Total	**44.651,51 €**

Tabla 18: Resumen del presupuesto económico total. (Fuente: Elaboración propia).

Como se puede observar, el coste real del proyecto pudo ser reducido considerablemente gracias al consumo inteligente de los recursos disponibles y a una ejecución que no ha incluído costes indirectos ni imprevistos.

3.3. Estudio de costes de operación del producto mínimo viable

Para finalizar el estudio económico del proyecto, se debe analizar los costes fijos asociados a la explotación del producto mínimo viable, de cara a poder compararlos con aquellos previstos por el nuevo producto y estudiar la diferencia que le supone a la empresa la explotación de una y otra arquitectura, y el coste-valor real del refactoring.

La distribución de costes asociada al producto mínimo viable actual se representa en la *Tabla 15: Distribución de costes asociada al MVP*. En este estudio, no se incluye la inversión en ningún término asociada al diseño y puesta en marcha, así como costes variables imputados como comisiones de pasarelas de pago como Stripe.

Costes totales asociados a la explotación del MVP		
Identificador del Coste	**Descripción**	**Coste anual (€)**
C.P.1	Dominio	49,99 €
C.P.2	Hosting	1.999,00 €
C.P.3	AWS	410,00 €
C.P.4	SEO	189,99 €
C.P.5	Zapier	599,99 €
C.P.6	Docusign	389,99 €
C.P.7	Lectura Specs	1.999,99 €
C.P.8	Valora Peritación	699,99 €
C.P.9	Proveedor Mail	299,99 €
Total		**6.638,93 €**

Tabla 19: Distribución de costes asociada al MVP. (Fuente: Elaboración propia).

3.4. Estudio de costes de operación del nuevo producto

En segundo lugar, se presentan en la Tabla 20: *Distribución de costes asociada al producto Profesiolan actualizado*, los costes estimados asociados a la explotación del producto tras refactoring.

Estudio de costes asociados al producto actualizado		
Identificador del Coste	**Descripción**	**Coste anual (€)**
C.PW.1	Capa Base de Datos	10,20 €
C.PW.2	Capa Core	535,68 €
C.PW.3	Capa API	4.020,96 €
C.PW.4	Gestión de Identidades	372,00 €
C.PW.5	Gestión de Salud	191,92 €
C.PW.6	Otros Costes	4.039,94 €
Total		9.170,70 €

Tabla 20: Distribución de costes asociada al producto Profesiolan actualizado. (Fuente: Elaboración propia).

La descripción detallada de cada una de las partidas se realiza a continuación de forma individualizada.

3.4.1. Estudio de costes de la capa Base de Datos

En este caso, el alcance del cluster MySQL, así como el de la gestión de la HTTP API de acceso, y toda la seguridad aplicable, es gestionado por AWS Aurora, en su versión v2 Serverless. Las características del mismo, que serán las últimas responsables de la configuración del ticket, son resumidas a continuación:

- Engine Version: 5.7.mysql_aurora.2.11.3.
- Unidades de capacidad mínimas: 1.
- Unidades de capacidad máximas: 4 unidades.
- Tiempo de autoescalado: 5 minutos.
- Tiempo para el pausado de una únidad de capacidad en inactividad: 5 minutos.
- Autenticación por IAM: Habilitada.
- Encriptación: Habilitada.

La base de costes asociada a la versión v2 Serverless es fácilmente calculable mediante la tarificación en dos tramos basada en 0,10 € por GB almacenado por mes, sumado a los 0,20 € por millón de peticiones. Partiendo de la base de que el tamaño de la base de datos en el MVP es conocida, y es

MySQL versión 5.7, la misma, es posible aplicar estos números para una primera estimación de costes.

Con una base de datos actual de 4GB, y una previsión de 1 Millón de peticiones a la base de datos de forma mensual, el ticket mínimo estimado es de 0,60 € mensuales. No obstante, a esta cuantía se le debe añadir sobrecostes debidos al uso de la HTTP API, un autoescalado basado en hasta 4 nodos o Unidades de capacidad, y unos tiempos asociados a la escalabilidad horizontal bajos.

En resumen, se espera que el coste asociado a la capa de la base de datos no sea mayor a 0,85 € de forma mensual.

3.4.2. Estudio de costes de la capa Core

El estudio de costes asociado a la totalidad del core está unificado mediante el servicio AWS Lambda. Para realizar un analisis acertado, es necesario analizar los siguientes aspectos:

- Arquitectura: x86_64.
- Runtime: Java 11.
- Peso del código: Entorno a los 25 MB por microservicio.
- Memoria: 512 MB.
- Ephemeral storage: 512 MB.
- Timeout: 15 s.
- SnapStart: Sí.
- Url pública de función: No.
- Embebido en VPC: No.
- Invocación asíncrona: Establecida en 6 horas, con 2 intentos de recuperación.
- Versionado: Activado.
- Proxies: No.
- Orquestramiento: No configurado.

Dada (1) la complejidad asociada en realizar una matriz con el desglose de cada una de las partidas de costes en función de la característica de los microservicios, (2) la naturaleza de pago por tiempo de servicio levantado con una granularidad de milisegundos, ponderado a unas ventanas de peticiones ejecutadas, y (3) que se asume una distribución del consumo equitativa y equilibrada

entre todos los microservicios, el análisis del coste se basa en ofrecer el coste real incurrido por un mes de operación de uno de los microservicios desarrollados (3,72 €). A continuación, se asume un multiplicador de 2 debido a las particularidades mencionadas, y otro multiplicador de 1,5 debido a la posibilidad de entrada en escalabilidad ante un potencial pico de usabilidad semanal. Este análisis sitúa el coste mensual estimado por microservicio en operación en menos de 11,16 € (44,64 € en total).

3.4.3. Estudio de costes de la capa API

La forma de facturación de AWS Appsync es sencilla, con una única tarifa de 4,00 € por cada millón de operaciones GraphQL ejecutadas. No están incluidas, sin embargo, operaciones de tipo GraphQL subscription, que tienen un precio de 2,00 € por cada millón de actualizaciones en real-time. De cualquier forma, la versión de la API desarrollada en este Trabajo Fin de Máster no incluye ningún tipo de operación del tipo GraphQL subscription. En este caso, el enfoque del cálculo del precio orientativo pasa por estimar 50.000 usuarios activos mensualmente, cada uno realizando una media de 20 búsquedas, generando un millón de operaciones GraphQL. Obviando el precio de la transferencia de información, ya que es varios órdenes de magnitud menores al coste de la propia operación y porque el tráfico es texto, el coste mensual estimado es de 4,00 € mensuales.

En cualquier caso, el grueso del coste asociado a la explotación de la capa API es el incurrido por la memoria Caché. AWS Appsync ofrece hasta 8 niveles de Caché diferentes, listados en la *Figura 13: Oferta de memorias Caché aplicables al servicio AWS Appsync*.

Instance type	vCPU	Memory	Network Performance	Pricing
cache.small	1	1.55	Low to Moderate	$0.044
cache.medium	2	3.22	Low to Moderate	$0.089
cache.large	2	12.3	Up to 10 Gigabit	$0.298
cache.xlarge	4	25.05	Up to 10 Gigabit	$0.595
cache.2xlarge	8	50.47	Up to 10 Gigabit	$1.189
cache.4xlarge	16	101.38	Up to 10 Gigabit	$2.379
cache.8xlarge	32	203.26	10 Gigabit	$4.758
cache.12xlarge	48	317.77	10 Gigabit	$6.775

Figura 13: Oferta de memorias Caché aplicables al servicio AWS Appsync. (Fuente: AWS Appsync)

Se han realizado distintas pruebas, cuyos procedimientos y resultados se incluyen en el *Anexo 1: Descripción de la solución desarrollada*, para valorar la mejor opción, teniendo en consideración el precio, el valor aportado en materia de métricas de retardo de respuesta, y el tamaño.

Tras dicho estudio, se ha determinado que la Caché elegida debe ser la denominada como cache.medium, con un coste de 0,089 € por hora. Además, teniendo en cuenta que esta caché debe ser activada por cada uno de los resolvers GraphQL (existen 5), la activación de la caché tendría un precio de 331,08 € mensuales.

3.4.4. Estudio de costes asociados a la gestión de identidades y autenticación y autorización

En este caso, el alcance de la gestión de identidades, así como el de la autenticación y la autorización, es gestionado por AWS Cognito. El precio asociado a este servicio es totalmente gratuito hasta 50.000 identidades, saltando a 0,0055 € por identidad gestionada a partir de la 50.001 (y hasta la identidad número 900.000).

En este punto, se debe hacer especial hincapié en la naturaleza B2B de la solución y que, si bien se trata de un marketplace público, aquellas tareas relacionadas con la necesidad de almacenar una identidad son la de publicación (requiere un fuerte proceso de verificación y fotovalidación), y la de generación de ticket de interés o compra (con su consiguiente proceso de verificación), por lo que el número de clientes o identidades a gestionar por Profesiolan no se espera que supere los 50.000 en los proximos 18-24 meses.

Si que se debería incluir en este apartado el servicio AWS SNS, relacionado con el envío de notificaciones en diferentes medios para complementar los procesos de autenticación y autorización con buenas prácticas. En otras palabras, AWS SNS permite integrar funcionalidades de interés como el inicio de sesión en dos dispositivos, la renovación de la contraseña mediante el envío de diferentes correos y mensajes telefónicos, la posibilidad de verificar un teléfono mediante un SMS, y un largo etcétera.

A su vez, SNS aplica diferentes tarifas en función del tipo de *endpoint* al que se notifica, aplicando 0,50 € por millón de notificaciones tipo Push en teléfono para aplicaciones móviles, 2,00 € por cada 100.000 correos electrónicos, y 0,60 € por cada millón de notificaciones HHTP/s. También aplica tarifas a SMS, con una base imponible por el Carrier elegido, y un formato de contabilizado en el que cada 64KB de data se factura como 1 envío, pudiendo llegar a incurrir en unos 0,03 € por SMS con alguna operadora nacional.

Una estimación de 50.000 correos electrónicos mensuales, y 1.000 SMS, el coste asociado a esta partida se situaría en torno a los 31,00 € mensuales.

3.4.5. Estudio de costes asociados a la gestión de la salud del producto

El estudio de costes asociado a la gestión de logs, monitorización, gestión de alarmas y eventos, y obtención de métricas de salud está unificado mediante el servicio AWS Cloudwatch, y a través de sus subservicios.

Por todo esto, a continuación se realiza un desglose de todas las subpartidas asociadas a la gestión de la salud del *backend*.

3.4.5.1. Logging

El coste asociado al logging se resume en las siguientes subpartidas:

- Data Ingestion: 0,50 € por GB. Se estima una generación de 1 GB de datos de log de forma quincenal, que suman un total de 1,00 € de forma acumulativa cada mes. Teniendo en cuenta que se plantea un congelamiento de los logs con anterioridad mayor a 12 meses mediante una exportación, borrado de la memoria de Cloudwatch, e importación a S3 Glacier, el coste anual siempre es el mismo: 78,00 €.
- Data Archival: Aplicando el mismo umbral de generación de eventos, y a un precio de 0,03 € por GB, el archivo de los logs tiene un coste anual previsto de 0,72 €.

- Analyze: Herramienta clave que permite ejecutar consultas avanzadas, por valor de 0,005 € por GB escaneado, y que en total se estima un consumo no mayor a 10,00 € anuales.

3.4.5.2. Metrics

Metrics es una herramienta consumible vía API mediante comandos del tipo GetMetricData, GetInsightReport… que devuelven KPIs estádisticos clave, por un valor de 0,30 € estadístico a estudiar por mes.

En este contexto, se definen hasta 10 métricas estadísticas a estudiar, relativas a las conexiones establecidas al cluster de base de datos, nivel de congestión en la API, y latencias clave en todas las etapas de la arquitectura.

Por todo ello, se estima un gasto mensual de 3,00 €.

3.4.5.3. Dashboards

AWS Cloudwatch permite la creación de paneles de administrador altamente customizables. En concreto, la granularidad ofrecida es absoluta, pudiendo incluir estadísticos primarios ofrecidos por cada uno de los servicios AWS, e incluso la creación de paneles totalmente personalizados que se ajustan a la perfección a las necesidades de cualquier producto.

Honestamente, el trabajo de diseño de una interfaz de monitorización y gestión basada en Cloudwatch Dashboards ha podido realizarse sin la necesidad de ningún panel personalizado. En cualquier caso, cada panel tiene un coste mensual de 3,00 €.

3.4.5.4. Alarms

De igual forma, AWS Cloudwatch permite la creación de alarmas, de altísimo valor para gestionar caídas de servicio, fallos puntuales, caídas en el rendimiento, picos de usabilidad, y un largo etcétera.

Durante el desarrollo de los sistemas relacionados con la gestión de la salud, se han establecido hasta 20 alarmas, pero se estima extender convenientemente e iterativamente el número de alarmas hasta 50 durante el primer año de operación. El precio propuesto por AWS es de 0,10 € por alarma por mes, que suma 5,00 € de forma acumulada mensual, o 60,00 € anuales.

3.4.6. Estudio de costes asociados al almacenamiento de recursos

La elección elegida para el almacenamiento de los recursos estáticos como imágenes y vídeos de activos, ficheros de stock o incluso imágenes de proveedores es AWS S3.

Debido a la naturaleza asíncrona de la necesidad de obtener los recursos y debido a que no son recursos que se puedan almacenar en formato "glacier" por su necesidad de lectura constante, el plan elegido es S3 Intelligent -Tiering: Frequent Access Tier, 50 TB / Month, con un coste asociado de 0.023 por GB por mes.

Todos los recursos de tipo imagen asociados al stock actualmente en la base de datos de Profesiolan supera los 25 GB. Por ello, se estima un coste mensual de 0,60 € de forma mensual.

3.4.7. Estudio de otros costes

En esta partida, *Tabla 21: Estudio de otros costes asociados a la explotación del producto*, se deben incluir servicios y costes que ya estaban incluídos en la explotación del MVP. Entre otros, integraciones clave para proveer de fotoverificación, firma electrónica vía API, integraciones de especificaciones técnicas, Zapier para realizar integraciones puntuales y realizar testeos y, por supuesto, el coste del dominio.

Estudio de otros costes asociados a la explotación del producto		
Identificador del Coste	**Descripción**	**Coste anual (€)**
C.PA.1	Dominio	49,99 €
C.PA.2	Zapier	599,99 €
C.PA.3	Docusign	389,99 €
C.PA.4	Lectura Specs	1.999,99 €
C.PA.5	Valora Peritación	699,99 €
C.PA.6	Proveedor Mail	299,99 €
Total		4.039,94 €

Tabla 21: Estudio de otros costes asociados a la explotación del producto. (Fuente: Elaboración propia).

3.5. Análisis del ROI del proyecto

El retorno de la inversión de un proyecto asociado al refactoring del producto principal de una empresa es una métrica confusa en líneas generales. De cualquier forma, en este apartado se arroja un procedimiento de maduración propia que, si bien podría ser válido, la complejidad del mismo y la consiguiente facilidad para cometer errores en la exposición de los antecedentes, lo hace de poca utilidad.

1. Elección de un determinado número de KPIs de servicio que serán las variables estadísticas a estudiar.
2. Cuantificación, de manera objetiva y total, de las diferencias en la calidad de servicio entre el servicio previo y el actualizado, entendiendo como calidad de servicio el dataset definido en 1.
3. Valoración objetiva, por medio de estudio de la analítica de las conversiones, de la calidad de servicio. Se debe ser preciso y, por medio de la correlación entre la calidad de servicio y las ventas realizadas, obtener un desglose de percentiles de tiempos y ventas perdidas.
4. Estimar las ventas agregadas debidas a la mejora del servicio calculada en 2.
5. Ejecutar el cálculo del retorno de la inversión, tratando de cruzar el surplus de ventas con, no solo la inversión de desarrollo, si no también con el nuevo coste de mantenimiento de producto.

Otra metodología válida puede desarrollarse por medio del campo de la valoración de riesgos. En este caso, es la propia dirección junto a los técnicos de la empresa quienes deben valorar el riesgo de crecer sin invertir en una arquitectura de producto robusta y que atiende a buenas prácticas.

En la experiencia del autor de este Trabajo Fin de Máster, se ha comprobado que es precisamente este segundo método el más dado para la toma de la decisión sobre la ejecución de un refactoring total de producto.

Por todo ello, a menudo resulta de poco valor el cálculo del retorno de la inversión asociado a un proyecto que tiene como fin mejorar el producto base de una empresa, y más cuando se trata de un SaaS, donde la práctica totalidad de la cadena de valor proporcionada por la empresa se sirve de forma self-service y sin necesidad de un equipo humano, y por tanto la totalidad del valor de la empresa reside en la calidad, robustez, y resiliencia de su producto.

4. Conclusiones

Tras más de 230 días de ejecución del proyecto, se realiza un crítico y meticuloso análisis del trabajo realizado. En este apartado no se juzgan únicamente los resultados del producto entregado, sino que también se evalúan aspectos relativos a la dirección de proyectos de ingeniería como son la gestión del tiempo, la exactitud en la previsión de la planificación, la planificación de los recursos, o incluso la conformidad de los objetivos marcados.

El proyecto ha concluido completando, de manera exitosa, la totalidad del alcance previsto en una primera revisión de los objetivos. Esto incluye la finalización de la totalidad de las funcionalidades, pero también incluye otros activos de valor como son la documentación sobre la arquitectura, la definición, medida y gestión en vida de KPIs y otros indicadores de rendimiento del producto.

En otro plano, se ha realizado una excelente previsión del alcance del proyecto, que se ha traducido en una correcta y exacta planificación de las tareas, los plazos, y los recursos asociados. El proyecto, que constaba de más de 100 tareas divididos en más de una docena de grupos de tareas correctamente paquetizadas y definidas en alcance ha finalizado en tiempo.

A continuación, se hace una valoración cualitativa del trabajo realizado, sin intervenir en la propia presentación de la tecnología.

4.1. Cumplimiento de los objetivos de la empresa

- ☑ **Mejora del uptime:** La solución desarrollada, una vez finalizada con todas las funcionalidades extra y tras un extensivo proceso de revisión del código previo al lanzamiento del refactoring, cumplirá con una mejora del uptime gracias a las decisiones tomadas en el contexto del stack tecnológico y el trabajo realizado en relación al desarrollo de la infraestructura con buenas prácticas.
- ☑ **Mejora del QoS:** De igual forma, y aunque tal y como se adelantaba en el *Apartado 1.3.1.2: Mejora del QoS,* las limitaciones más críticas que motivaban un refactoring absoluto del producto no tenían relación con el QoS, una mejora general del uptime, y un tiempo de respuesta del servidor más rápida tendrá unas consecuencias positivas y presentará una mejora sin duda también en este campo.

- ☑ **Mejora de Features:** La nueva infraestructura unificada de desarrollo en AWS, y la nueva arquitectura modular de las funcionalidades de la lógica de negocio, promete una rapidez mayúscula a la hora de desarrollar, testear, e incorporar nuevas funcionalidades a la API. De igual forma, esta tendencia no compromete el sistema de versiones de la API, ofreciendo el producto desarrollado las mismas herramientas de etiquetado de versiones a nivel API mediante GraphQL que grandes compañías de Software como Facebook o Instagram.
- ☑ **Inversión Mínima y tiempo de desarrollo mínimo:** También se considera alcanzado el objetivo máximo de la empresa en relación al recorte de costes y optimización en el tiempo. Este aspecto se desarrolla con más calidad durante el *Apartado 4: Aspectos económicos*, donde se hace referencia al ahorro supuesto frente a la alternativa de la subcontratación del proyecto.

4.2. Cumplimiento de los objetivos técnicos de la Base de Datos

- ☑ **Multi AZ Deployment:** La solución desarrollada cumple con el objetivo de poder ser escalable territorialmente en un futuro de forma sencilla y sin fricciones.
- ☑ **Escalabilidad horizontal y vertical:** De igual forma, la solución desarrollada cumple con la necesidad de escalado de forma automática para dar servicio a picos de consumo.
- ☑ **Mejora de las prestaciones del MVP:** La nueva Base de Datos desplegada cumple con la obligación de mejorar en prestaciones la solución actual diseñada en el contexto del producto mínimo viable.
- ☑ **Inversión Mínima y tiempo de desarrollo mínimo:** También se considera alcanzado el objetivo básico.

4.3. Cumplimiento de los objetivos técnicos de la API

- ☑ **Solución basada en GraphQL:** La solución desarrollada cumple con el interés estratégico de uso de GraphQL, propuesto desde la dirección de la empresa, y que supone una apuesta tecnológica en el largo plazo.
- ☑ **Escalabilidad práctica infinita:** Tal y como se definía en el *Apartado 1.3.2.2 Objetivos de la API*, la escalabilidad infinita es un concepto que atiende más a un contexto de venta que de

ingeniería. No obstante, los proveedores de servicios en la nube son capaces de proporcionar una sensación de escalabilidad infinita, a cambio de un incremento en el ticket. Además, los tiempos relativos a un incremento de la capacidad (levantamiento de una réplica del servicio por parte de AWS en este caso para dar servicio cuando el servicio principal entra en congestión) son realmente bajos. Esto mejora en gran medida los estándares de servicio ofrecidos actualmente en el MVP.

- ☑ **Seguridad multinivel:** La API desplegada realiza, mediante el servicio Cognito, tareas de autenticación y autorización. Además, previene ataques de inyección, impide ataques de fuerza bruta, y permite el mapeo de tokens JWT con credenciales de lógica de negocio.
- ☑ **Implementación de Caché:** El acceso a la Caché de AWS es uno de los aspectos de mayor relevancia y valor añadido incorporados, las pruebas realizadas dan una muestra de la diferencia en tiempos de servicio. Por supuesto, tal y como se ha visto, la activación de esta encarece en gran medida el ticket.

4.4. Cumplimiento de los objetivos técnicos de la capa de Negocio

- ☑ **Modularidad:** La solución desarrollada tiene un diseño modular, que permite el refactoring parcial de cada uno de los servicios de forma individual, sin afectar a los demas componentes. Esto también contribuye positivamente al objetivo segundo, separación de dependencias por usos de negocio.
- ☑ **Tiempos de respuesta bajos:** A pesar de haber realizado una migración desde un sistema monolítico a un sistema distribuído, modular, y basado en diferentes servicios, se ha conseguido que el tiempo de respuesta relativo al trabajo de la capa core, ponderado al potencialmente equivalente en el sistema monolítico del MVP, no sea mayor.
- ☑ **Transaccionalidad:** En la presente versión del trabajo, se ha conseguido configurar la transaccionalidad para todas las funcionalidades. Sin duda ha sido todo un reto debido al caracter modular de la solución.
- ☑ **Granularidad en las excepciones:** Como se describe en profundidad en el *Anexo I: Descripción de la solución realizada*, se han diseñado diferentes excepciones descriptivas que aportan valor para encontrar lo más rápido posible la raíz de las mismas.

5. Bibliografía

[1]. Profesiolan. La segunda mano, para tu empresa, [https://profesiolan.com/]. [Consulta: 01-08-2023].

[2]. El Referente: Profesiolan, [https://elreferente.es/startup/profesiolan/]. [Consulta: 01-08-2023]

[3]. Circular, For Resource and Waste Professionals (29 de noviembre del 2022): Making the case for reuse: achieving net zero by 2050, [https://www.circularonline.co.uk/opinions/making-the-case-for-reuse-achieving-net-zero-by-2050/]. [Consulta: 03-08-2023]

[4]. Profesiolan. Presentación reducida, [https://drive.google.com/file/d/1oUcJUG0OwBKF07u2ct7XDMLLo28UGyoU/view?usp=drive_link]. [Consulta: 03-08-2023]

[5]. Sammy Abdullah (Chrunchbase.com, publicado en noviembre del 2018 y actualizado en abril del 2023). What Should Your Startup be Spending on Product?, [https://about.crunchbase.com/blog/startup-product-spend/]. [Consulta: 05-08-2023]

[6]. Wikipedia: Uptime, [https://es.wikipedia.org/wiki/Uptime]. [Consulta: 05-08-2023]

[7]. StatusCast (2022): SaaS Application Uptime Stats Benchmark, [https://statuscast.com/saas-application-uptime/]. [Consulta: 06-08-2023]

[8]. AWS Lambda SLA, [https://aws.amazon.com/lambda/sla]. [Consulta: 06-08-2023]

[9]. AWS: ¿Qué es una API?, [https://aws.amazon.com/es/what-is/api/]. [Consulta: 06-08-2023]

[10]. The OWASP Top 10 (Informe anual del 2017), [https://www.cloudflare.com/learning/security/threats/owasp-top-10/]. [Consulta: 07-08-2023]

[11]. AWS: Información General Sobre el Almacenamiento en Caché, [https://aws.amazon.com/es/caching/]. [Consulta: 15-09-2023]

[12]. Knoernschild, Kirk. Java design: objects, UML, and process. Addison-Wesley Professional, 2002. [Consulta: 15-09-2023]

[13]. Distributed transaction patterns for microservices compared, [https://developers.redhat.com/articles/2021/09/21/distributed-transaction-patterns-microservices-compared]. [Consulta: 16-09-2023]

[14]. Apollo GraphQL: Error Handling, [https://www.apollographql.com/docs/apollo-server/data/errors/]. [Consulta: 10-10-2023]

[15]. Wikipedia: Red Hat, [https://es.wikipedia.org/wiki/Red_Hat]. [Consulta: 10-10-2023]

[16]. Red Hat: Cloud Computing, [https://www.redhat.com/es/topics/cloud]. [Consulta: 10-10-2023]

[17]. Google: Cloud Computing, [https://cloud.google.com/learn/what-is-cloud-computing]. [Consulta: 01-09-2023]

[18]. The 2023 Global Social Media Trends Report, by Hubspot, [https://offers.hubspot.com/social-media-trends-report]. [Consulta: 24-09-2023]

[19]. State of Mobile 2021, by App Annie (former Data.ai), [https://www.data.ai/en/go/state-of-mobile-2021]. [Consulta: 24-09-2023]

[20]. IaaS vs PaaS vs SaaS? Google Cloud Learn, [https://www.redhat.com/es/topics/cloud-computing/iaas-vs-paas-vs-saas]. [Consulta: 24-09-2023]

[21]. What is IaaS? Google Cloud Learn, [https://cloud.google.com/learn/what-is-iaas]. [Consulta: 24-09-2023]

[22]. What is PaaS? Google Cloud Learn, [https://cloud.google.com/learn/what-is-paas]. [Consulta: 24-09-2023]

[23]. What is SaaS? AWS, [https://aws.amazon.com/es/what-is/saas/]. [Consulta: 24-09-2023]

[24]. TIOBE Index, [https://www.tiobe.com/tiobe-index/]. [Consulta: 01-01-2024]. [Consulta: 24-09-2023]

[25]. Moved By Java, Oracle, [https://www.oracle.com/java/moved-by-java/timeline]. [Consulta: 24-09-2023]

[26]. Borges B., Erickson K., Adams, G. (4 de abril del 2023). Reasons to move to Java 11, by Microsoft, [https://learn.microsoft.com/en-us/java/openjdk/reasons-to-move-to-java-11]. [Consulta: 01-10-2023]

[27]. AWS Lambda adds support for Java 17, [https://aws.amazon.com/about-aws/whats-new/2023/04/aws-lambda-java-17/]. [Consulta: 01-10-2023]

[28]. JRebel: 2021 Java Developer Productivity Report, [https://www.jrebel.com/resources/java-developer-productivity-report-2021]. [Consulta: 01-10-2023]

[29]. Maven Repository of Spring Boot Starter Library, [https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter]. [Consulta: 07-10-2023]

[30]. Red Hat: GraphQL, [https://www.redhat.com/es/topics/api/what-is-graphql]. [Consulta: 09-09-2023]

[31]. GraphQL, a query language for your API, [https://graphql.org/]. [Consulta: 01-08-2023]

[32]. About the GraphQL API, Github, [https://docs.github.com/en/graphql/overview/about-the-graphql-api]. [Consulta: 01-08-2023]

[33]. GraphQL Landscape, [https://landscape.graphql.org/borderless-mode?grouping=category]. [Consulta: 01-08-2023]

[34]. Chadwick, David. (2009). Federated Identity Management. 10.1007/978-3-642-03829-7_3. [Consulta: 04-01-2024]

[35]. Identity Management Institute, Center for Identity Governance: Federated Identity Management Challenges, [https://identitymanagementinstitute.org/federated-identity-management-challenges/] [Consulta: 04-01-2024]

[36]. Common Federated Identity Protocols: Open ID Connect vs. OAuth vs. SAML 2, [https://www.securityjourney.com/post/analysis-of-common-federated-identity-protocols-openid-connect-vs-oauth-2.0-vs-saml-2.0]. [Consulta: 04-01-2024]

[37]. Eleanor Birrell and Fred B. Schneider. (Cornell University, 2013). Federated Identity Management Systems: A Privacy-Based Characterization. [Consulta: 05-01-2024]

[38]. Sanjay Ghemawat, Robert Grandl, Srdjan Petrovic, Michael Whittaker, Parveen Patel, Ivan Posva, and Amin Vahdat. 2023. Towards Modern Development of Cloud Applications. In Proceedings of the 19th Workshop on Hot Topics in Operating Systems (HOTOS '23). Association for Computing Machinery, New York, NY, USA, 110–117. https://doi.org/10.1145/3593856.3595909. [Consulta: 05-01-2024]

[39]. McKeay, M., Ragan, S., Goedde, A., Tuttle, C.. State of the Internet: A Year in Review (Akamai, 2020). [Consulta: 05-01-2024]

[40]. Patni, S. Pro RESTful APIs: Design, Build and Integrate with REST, JSON, XML and JAX-RS (2017). [Consulta: 12-01-2024]