MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE MASTER

Anexo III: Análisis de rendimiento de la solución desarrollada.

profesiolan

Empresa apoyada por:

Estudiante: Hospital Gómez, Ander

Director: Prieto Agujeta, Gorka

Curso: 2023-2024

Fecha: Bilbao, a 23 de enero del 2024

Intencionadamente en blanco.

Introducción

El presente anexo está englobado en el Trabajo Fin de Máster 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.

Este apartado analiza, en profundidad y de forma integral, el desempeño de la API desarrollada. Para ello, se van a analizar los tiempos de respuesta absolutos para consultas concretas así como los tiempos asociados a cada una de las tres capas, de cara a encontrar los cuellos de botella en materia de latencia de la solución. Además, se realiza un estudio del impacto en rendimiento debido a la activación y del correcto diseño de la caché a nivel API. Finalmente, también se realizan análisis de estrés para flujos de navegación nominales, estudiando el impacto en tiempo de forma absoluta, y ofreciendo unas métricas reales y objetivas del desempeño de la API desarrollada.

Contenido

Lista de tablas

Lista de ilustraciones

1. Contexto y alcance del análisis

La API bajo la cuál se fundamenta la totalidad de un Marketplace (SaaS) debe ser robusta, resiliente, rápida, y segura. Dejando a un lado la seguridad, cuyo análisis no es objetivo del presente Anexo, tanto la robustez como la resiliencia y la velocidad son conceptos a analizar y mejorar.

En concreto, son conceptos que deben ser cuantificados y cualificados. Para ello, deben ser convertidos a métricas, enumerados, medidos y analizados. Después, deberán sacarse las conclusiones oportunas, atendiendo a los benchmarks de mercado, necesidad real de aplicación, y caso de uso.

El presente Anexo también se presenta como análisis de estrés. Esto es gracias al desarrollo de casos de test desarrollados en base a flujos habituales de navegación de usuario, que han sido sintetizados en eventos, y ejecutados en masa contra la API, obteniendo decenas de métricas de interés y obteniendo una visión fidedigna del comportamiento de la API ante un estrés varios ordenes de magnitud mayores a los existentes a día de hoy.

Además, el presente Anexo se presenta como demostrador de pruebas unitarias por la misma razón anteriormente mencionada. Realmente, la ejecución de tests en masa visibilizan el funcionamiento nominal individualizado de la solución.

Para todo ello, se ha hecho uso del entorno Postman, además de la herramienta JMeter de Apache. Durante el presente Anexo también se comparte una guía de "setup" de esta segunda herramienta, con la idea de garantizar la replicabilidad para un agente externo.

2. Entorno JMeter

JMeter es una herramienta de código abierto desarrollada por la Apache Software Foundation para la realización de análisis de rendimiento en aplicaciones. Su versatilidad permite analizar decenas de métricas de valor en diferentes protocolos. Además, es ideal para realizar Análisis de Estés en APIs tanto basadas en paradigmas RestFul como GraphQL. Algunas de las funcionalidades clave se listan a continuación.

- Pruebas de Carga Realistas: JMeter permite simular múltiples usuarios concurrentes, emulando así condiciones de carga realista, que sirven para evaluar el comportamiento de una API bajo presión.

- Escenarios Complejos: La herramienta propone la generación de escenarios complejos mediante la configuración de threads, y la definición de peticiones HTTP concretas a ejecutar en masa, permitiendo la reproducción de situaciones de uso real.

- Monitoring y Reporting Avanzado: JMeter ofrece herramientas avanzadas de monitoring y reporting, proporcionando información detallada sobre tiempos de respuesta, errores, tasas de interés, etc.

2.1. Presentación de los flujos

En este apartado se enumeran y presentan los casos de rendimiento diseñados.

2.1.1. F1: Unauthenticated navigation flow

El primero de los flujos representa una simulación realista de una navegación de compra de un usuario no autorizado (no login, únicamente API Keys). Para ello, se ha seguido un flujo común basado en un prefiltro, la aplicación de filtros, la búsqueda de activos, y la visualización de un activo concreto.

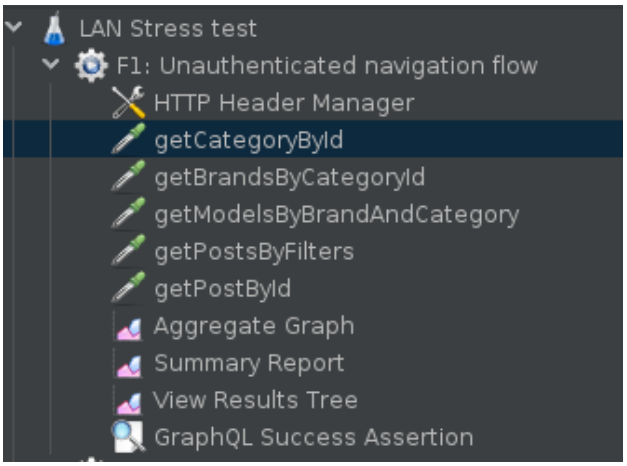

Figura 1: Definición de F1 en la herramienta JMeter.

A continuación se presentan las consultas concretas hardcodeadas del hilo. En primer lugar, se presenta el método getCategoryById, de utilidad tras el formulario presentado en la *Figura 2: Uso de getCategoryById en el producto profesiolan*.

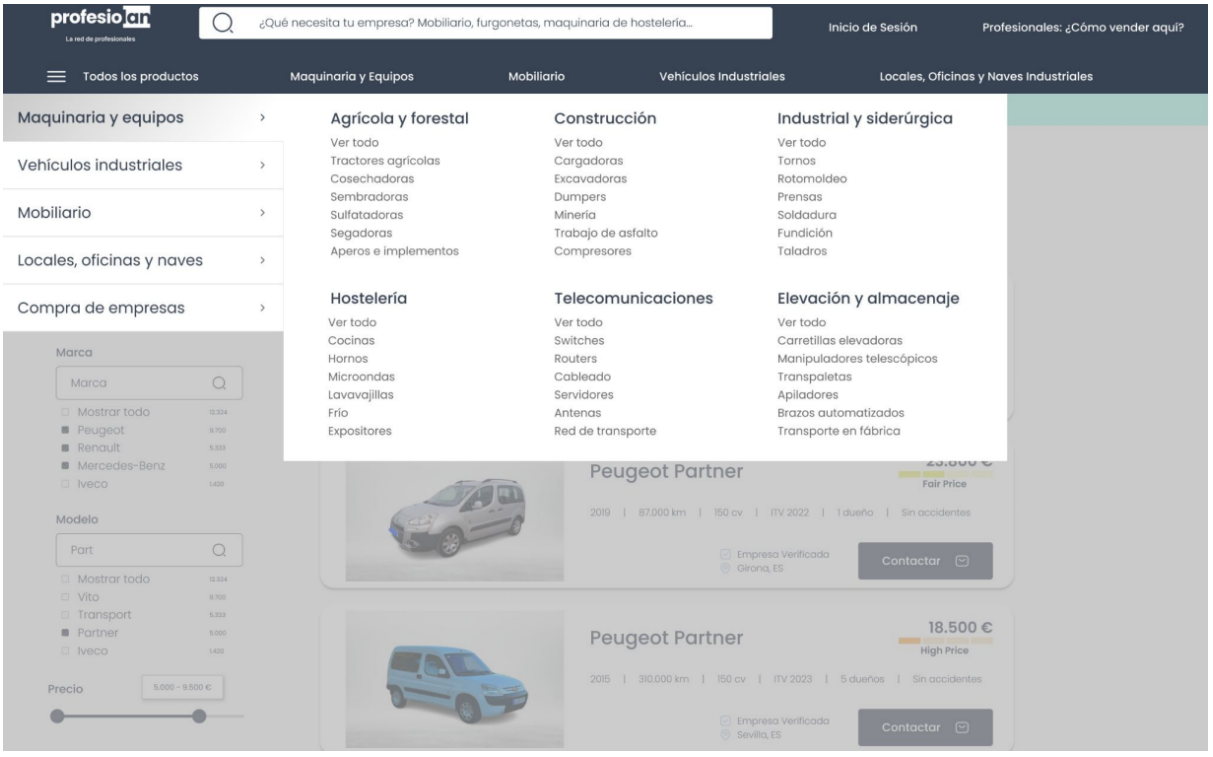

Figura 2: Uso de getCategoryById en el producto profesiolan.

De igual forma, se presenta de forma simplificada la entradas y metadata asociada a la query en *Tabla 1: Consulta getCategoryById perteneciente al Thread F1.*

F1: getCategoryById	
Nombre	getCategoryById
Server	lg2uz3spjjczzpalaaanio7ndi.appsync-api.eu-west-2.amazonaws.com
Protocol	https
Path	/graphql/
Http Request	POST
Auth.	API Key

Query	query MyQuery {
	getCategoryById(category_id: 437) {
	category_id
	name
	count
	meta_desc
	slug
	}
	}

Tabla 1: Consulta getCategoryById perteneciente al Thread F1.

De igual forma, se presenta el uso de la query getBrandsByCategoryId por parte del producto en la *Figura 3: Uso de getBrandsByCategoryId en el producto profesiolan*, y la descripción de la query en la *Tabla 2: Consulta getBrandsByCategoryId perteneciente al Thread F1.*

Ahora, obtenido el identificador de la categoría, se trata de presentar, en el filtro de navegación principal, las marcas con más volúmen para su selección.

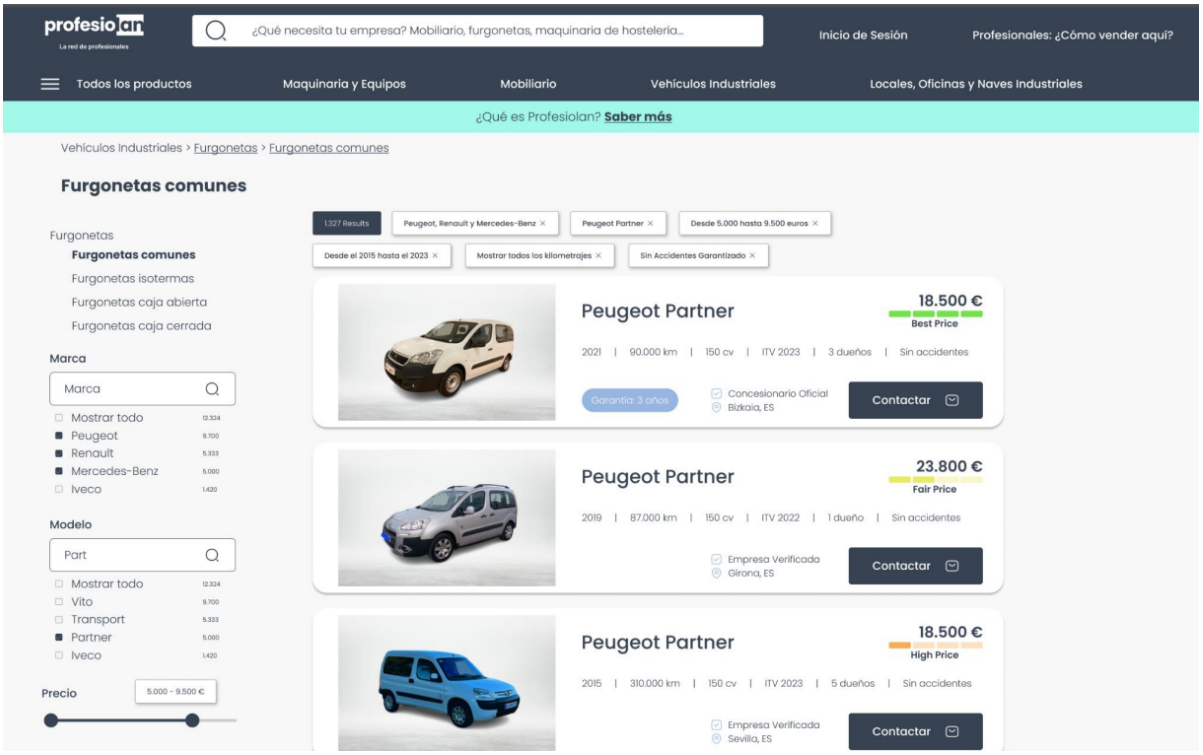

Figura 3: Uso de getBrandsByCategoryId en el producto profesiolan.

F1: getBrandsByCategoryId	
Nombre	getBrandsByCategoryId
Server	lg2uz3spjjczzpalaaanio7ndi.appsync-api.eu-west-2.amazonaws.com
Protocol	https
Path	/graphql/
Http Request	POST
Auth.	API Key

Query	query MyQuery {
	getBrandsByCategoryId(category_id: 437){
	brand_id
	name
	slug
	meta_desc
	count
	}
	}

Tabla 2: Consulta getBrandsByCategoryId perteneciente al Thread F1.

De igual forma, el uso de la query getModelsByBrandAndByCategory por parte del producto se presenta en la *Figura 3: Uso de getBrandsByCategoryId*, ya que forma parte del mismo formulario de búsqueda, y la descripción de la query en la *Tabla 2: Consulta getModelsByBrandAndByCategory perteneciente al Thread F1.*

La idea principal es la obtención de los principales modelos de maquinaria o vehículos asociados a una determinada marca y por supuesto a una determinada categoría.

F1: getModelsByBrandAndCategory	
Nombre	getModelsByBrandAndCategory
Server	lg2uz3spjjczzpalaaanio7ndi.appsync-api.eu-west-2.amazonaws.com
Protocol	https
Path	/graphql/
Http Request	POST
Auth.	API Key

Query	query MyQuery {
	getModelsByBrandAndCategory(brand_id: 1, category_id: 437) {
	model_id
	name
	slug
	count
	}
	}

Tabla 3: Consulta getModelsByBrandAndCategory perteneciente al Thread F1.

Finalmente, se ejecuta el formulario y se presentan los activos que cumplen con las determinadas características filtradas, tal y como se puede comprobar en la misma *Figura 3: Uso de getBrandsByCategoryId en el producto profesiolan.* También se presenta la descripción resumida de la query para su replicabilidad.

El resultado, tal y como se presenta en los Anexos I y II del presente Trabajo Fin de Máster, es devuelto paginado y con la opción de ordenación en función de varios parámetros.

F1: getPostsByFilters	
Nombre	getPostsByFilters
Server	lg2uz3spjjczzpalaaanio7ndi.appsync-api.eu-west-2.amazonaws.com
Protocol	https
Path	/graphql/
Http Request	POST
Auth.	API Key

Query	query MyQuery {
	getPostsByFilters(category_id: 66, pageNumber: 1, pageSize: 10, professional: false, sortBy: "published", sortDirection: "DESC") {
	post_id
	price
	professional
	slug
	status
	title
	}
	}

Tabla 4: Consulta getPostsByFilters perteneciente al Thread F1.

Finalmente, una vez el cliente descubre el activo que desea comprar o reservar, se abrirá la página de producto, como se presenta en la *Figura 4: Página de producto.*

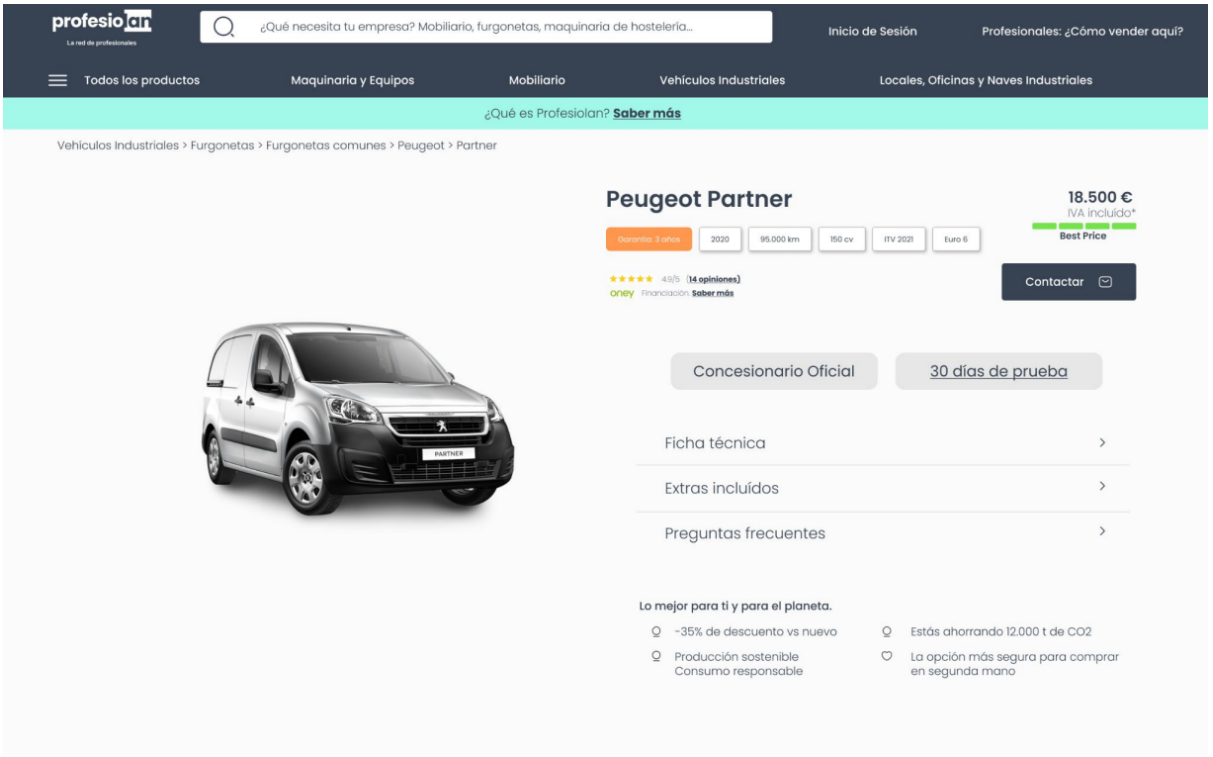

Figura 4: Página de producto.

Por último, se presenta la tabla resumen de la consulta para replicabilidad en *Tabla 5: Consulta getPostById perteneciente al Thread F1.*

F1: getPostById	
Nombre	getPostById

Server	lg2uz3spjjczzpalaaanio7ndi.appsync-api.eu-west-2.amazonaws.com
Protocol	https
Path	/graphql/
Http Request	POST
Auth.	API Key

Query	query MyQuery {
	getPostById(post_id:
	"a4761f72-0518-11ee-b68c-2eb5a363657c") {
	post_id
	title
	price
	slug
	has_price_insight
	user_id
	use_condition
	description_condition
	stock
	}
	}

Tabla 5: Consulta getPostById perteneciente al Thread F1.

2.1.2. F2: Authenticated post mgmt flow

El segundo de los flujos representa una simulación del flujo de gestión de activo publicado por parte de un usuario registrado y autorizado. Para ello, se ha seguido un flujo común basado en una búsqueda, la visualización de un activo concreto y la actualización de este.

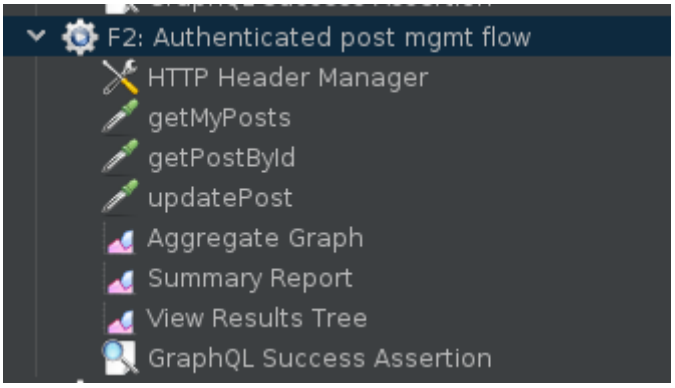

Figura 5: Definición de F1 en la herramienta JMeter.

En concreto, a continuación se presentan las consultas concretas hardcodeadas del hilo.

En primer lugar, se presenta la consulta getMyPosts, que tiene como fin el de mostrar, de forma resumida, la lista de activos asociados a un cliente, en su escritorio. Esta consulta se presenta de forma resumida en la *Tabla 6: Consulta getMyPosts perteneciente al Thread F1.*

F2: getMyPosts	
Nombre	getMyPosts
Server	lg2uz3spjjczzpalaaanio7ndi.appsync-api.eu-west-2.amazonaws.com
Protocol	https
Path	/graphql/
Http Request	POST
Auth.	API Key

Query	query MyQuery {
	  getMyPosts(user_id: "921f2bd6-d04a-4ffc-af85-03b90963574b") {
	  post_id
	  title
	  price
	  professional
	  slug
	  status
	  }
	}

Tabla 6: Consulta getMyPosts perteneciente al Thread F2.

Una vez encontrado el activo a modificar o borrar por parte del cliente, se hará click en el escritorio del cliente, para abrir la vista de editor del activo, mediante la consulta descrita en la Tabla 7: Consulta getPostById perteneciente al Thread F1.

F2: getPostById	
Nombre	getPostById
Server	lg2uz3spjjczzpalaaanio7ndi.appsync-api.eu-west-2.amazonaws.com
Protocol	https
Path	/graphql/
Http Request	POST
Auth.	API Key

Query	query MyQuery {
	  getPostById(post_id: "430d4b39-fdd6-4b70-8ee2-fa0ce0fd1eb1") {
	  post_id
	  title
	  price
	  slug
	  has_price_insight
	  user_id
	  use_condition
	  description_condition
	  stock
	  }
	}

Tabla 7: Consulta getPostById perteneciente al Thread F2.

Finalmente, el usuario realizará cambios en el activo, y los publicará para su actualización, mediante la Mutation descrita en *Tabla 8: Consulta updatePost perteneciente al Thread F1.*

F2: updatePost	
Nombre	updatePost
Server	lg2uz3spjjczzpalaaanio7ndi.appsync-api.eu-west-2.amazonaws.com
Protocol	https
Path	/graphql/
Http Request	POST
Auth.	API Key

Query	mutation MyMutation {
	  updatePost(input: {category_id: [437, 438, 439], content: "Updated Post", description_condition: "Prácticamente nuevo", latitude: 1.5, longitude: 1.5, post_id: "430d4b39-fdd6-4b70-8ee2-fa0ce0fd1eb1", price: 1.5, title: "Updated Post", use_condition: "Prácticamente nuevo", user_id: "921f2bd6-d04a-4ffc-af85-03b90963574b"}) {
	  categories {
	  category_id
	  name
	  }
	  content
	  description_condition
	  latitude
	  longitude
	  use_condition
	  title
	  stock
	  status
	  slug
	  published
	  professional

```
    post_id
    price
  }
}
```

Tabla 8: Consulta updatePost perteneciente al Thread F2.

2.1.3. F3: Post publish stress flow

El tercero de los flujos representa la publicación de un activo por parte de un usuario registrado y autorizado. La explicación sencilla de esto es poder realizar un Caso de Test en el que se simule la publicación en masa simultánea de publicaciones por parte de usuarios.

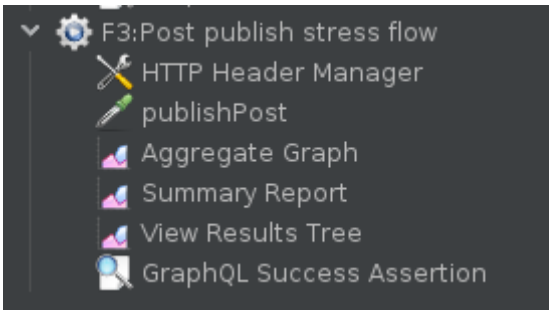

Figura 6: Definición de F3 en la herramienta JMeter.

Se presenta, en la *Tabla 9: Consulta publishPost perteneciente al Thread F3*, la naturaleza de la Mutation publishPosts, que tiene como objetivo la creación de un nuevo activo y la asociación del mismo a un usuario creado, con los permisos suficientes, y que además debe ser el mismo que realiza la operación.

F2: publishPosts	
Nombre	publishPosts
Server	lg2uz3spjjczzpalaaanio7ndi.appsync-api.eu-west-2.amazonaws.com
Protocol	https
Path	/graphql/
Http Request	POST
Auth.	API Key

Query	mutation MyMutation {
	publishPost(input: {category_id: [437, 438, 439], content: "Updated Post", description_condition: "Prácticamente nuevo", latitude: 1.5, longitude: 1.5, price: 1.5, title: "Updated Post", use_condition: "Prácticamente nuevo", user_id: "921f2bd6-d04a-4ffc-af85-03b90963574b"}) {
	categories {
	category_id
	name
	}
	content
	description_condition
	latitude
	longitude
	use_condition
	title
	stock
	status
	slug
	published
	professional
	post_id
	price
	}
	}

Tabla 9: Consulta publishPost perteneciente al Thread F3.

2.2. Presentación de los Casos de Test

En este apartado se desarrollan los diferentes enfoques tomados para la ejecución de cada uno de los diferentes casos de rendimiento identificados en el *Apartado 4.1: Presentación de los casos de rendimiento.*

- 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.

3. Análisis de rendimiento end-to-end

En este apartado se enumeran y presentan los resultados obtenidos para cada uno de los casos de test realizados y presentados en el *Apartado 2.2: Presentación de los Casos de Test.*

3.1.1. Presentación de resultados para CT1

En este apartado se presentan los resultados asociados al primer caso de test, en el que se testea el flujo principal de navegación no autorizada, con un usuario cada 10 segundos. Para otorgar de validez a este estudio, el flujo descrito se ha ejecutado 100 veces, obteniendo 500 resultados de latencias (100 para cada una de las consultas). En primer lugar, se presentarán los tiempos medios asociados por cada query para, a continuación, presentar los diagramas de velas y de violín del caso de test probado.

A continuación, se introducen los valores de la latencia media en la Figura 7: Resumen de latencias medias por query para CT1.

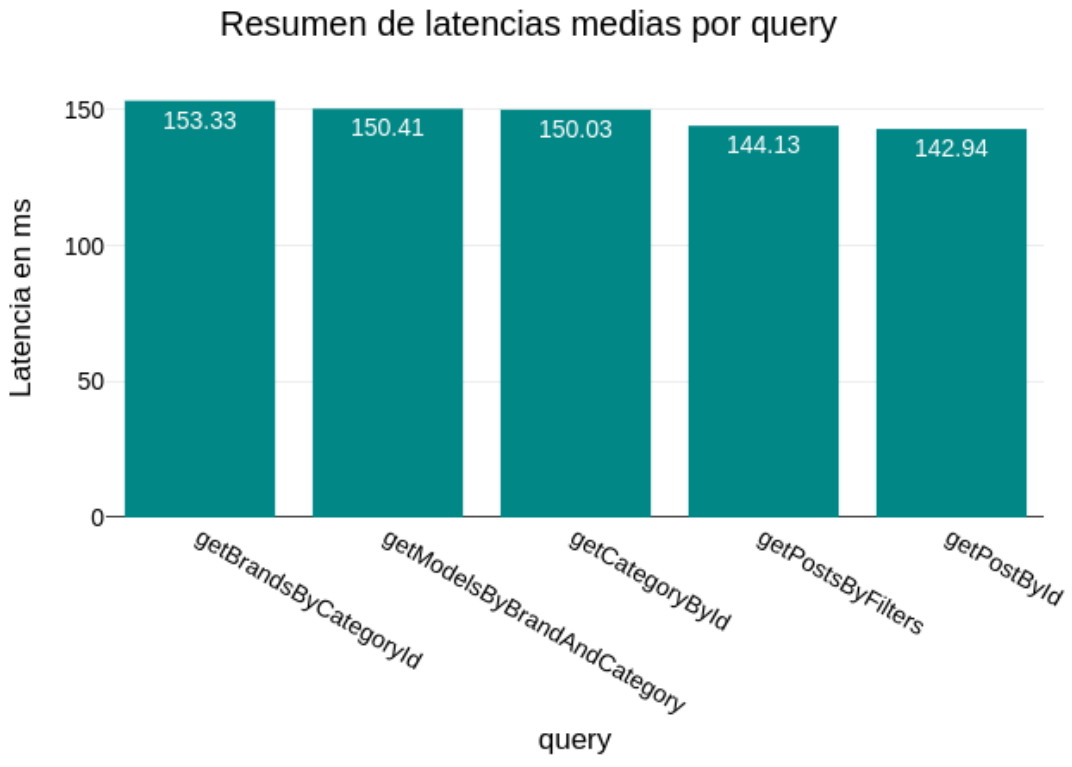

Figura 7: Resumen de latencias medias por query para CT1.

En general, las solicitudes tienen tiempos de respuesta muy positivos y consistentes. La operación más rápida es getPostById, con un tiempo promedio de 142,94 ms. La operación más lenta es getBrandsByCategoryId, con un tiempo promedio de 153,33 ms.

No obstante, resulta interesante presentar en la *Figura 8: Presentación avanzada de resultados para CT1 I* y la *Figura 9: Presentación avanzada de resultados para CT1 II* los resultados del estudio tanto en un diagrama de velas como en uno de violín, de cara a visualizar el comportamiento real de la solución desarrollada.

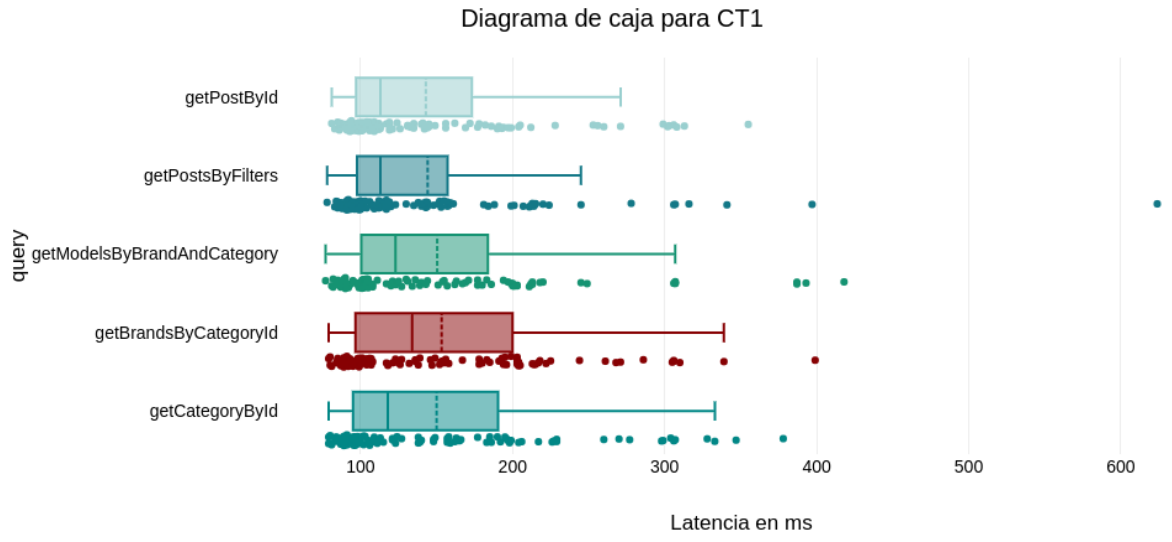

Figura 8: Presentación avanzada de resultados para CT1 I.

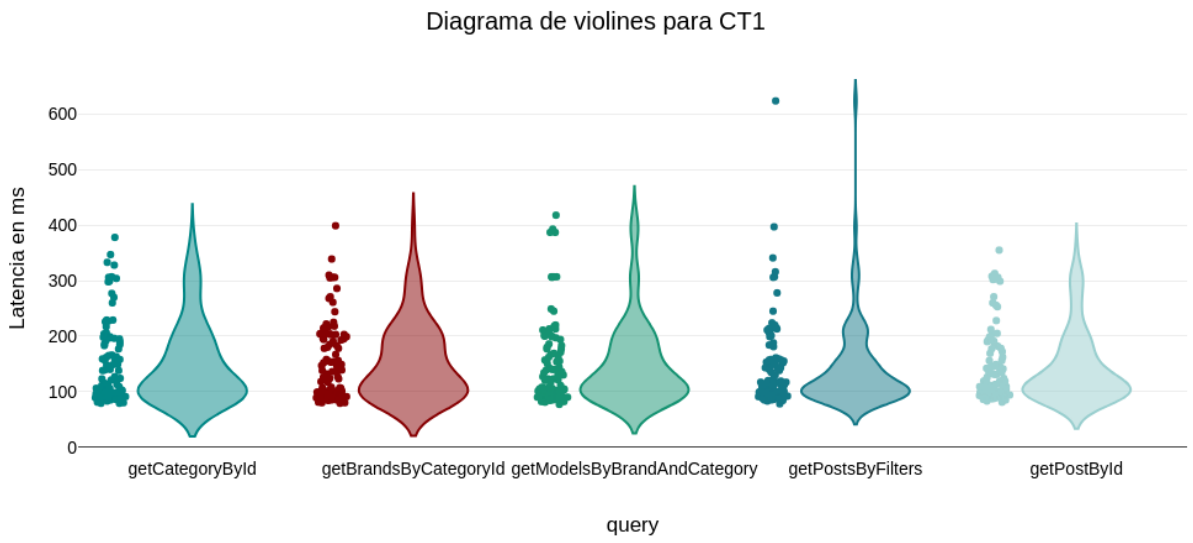

Figura 9: Presentación avanzada de resultados para CT1 II.

En este caso, 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, 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. Teniendo en cuenta que unas latencias menores a la quinta parte de un segundo supone un rendimiento excepcional, se concluye la viabilidad del proyecto desarrollado.

3.1.2. Presentación de resultados para CT2

En este apartado se presentan los resultados asociados al segundo caso de test, en el que se testea el flujo principal de navegación no autorizada, con 5 usuarios cada 10 segundos. Para otorgar de validez a este estudio, el flujo descrito se ha ejecutado 20 veces, obteniendo 500 resultados de latencias (100 para cada uno de los usuarios). En primer lugar, se presentarán los tiempos medios asociados por cada query para, a continuación, presentar los diagramas de velas y de violín del caso de test probado.

A continuación, se introducen los valores de la latencia media en la *Figura 10: Resumen de latencias medias por query para CT2.*

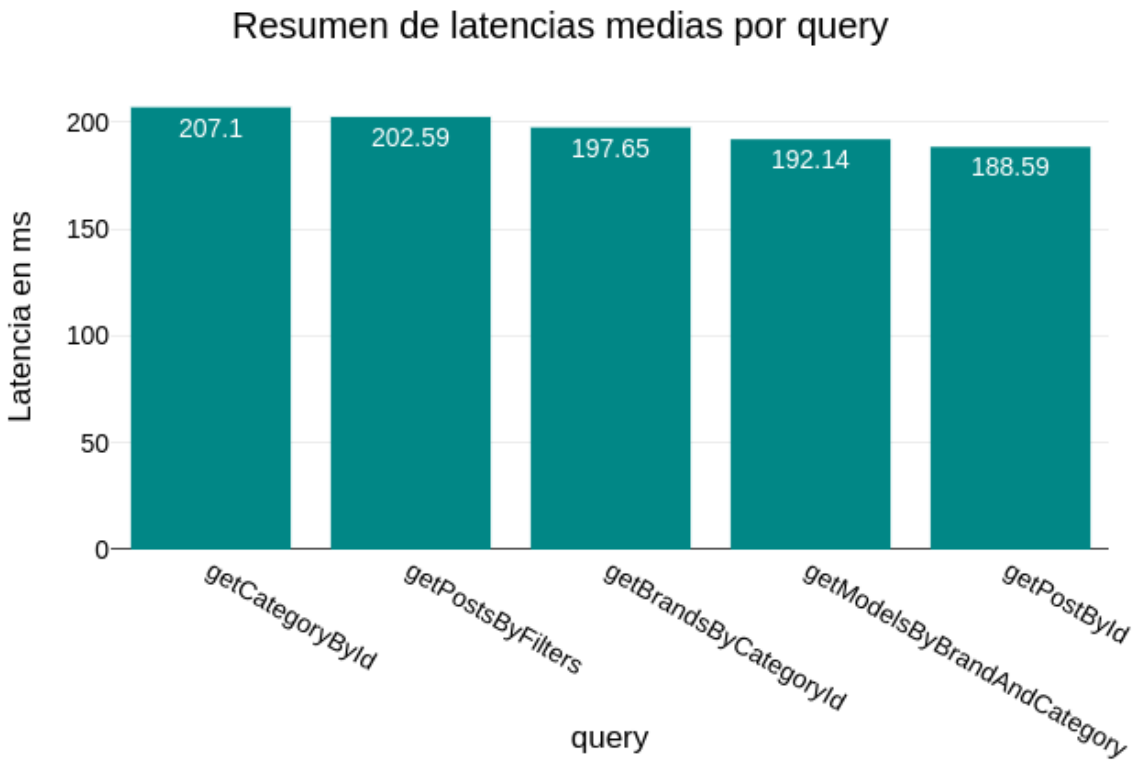

Figura 10: Presentación de resultados para CT2.

En general, las solicitudes tienen tiempos de respuesta bajos y consistentes. La operación más rápida es getPostById una vez más, con un tiempo promedio de 188,59 ms. La operación más lenta es getCategoryById, con un tiempo promedio de 207 ms. También resulta interesante observar que la diferencia en tiempo entre la media de la query más lenta y la más rápida no supera los 20 ms.

Al igual que se ha realizado en el anterior caso, a continuación se presenta, a través de la *Figura 11: Presentación avanzada de resultados para CT2 I* y la *Figura 12: Presentación avanzada de resultados para CT2 II* los diagramas de velas y de violín de los resultados obtenidos del test, respectivamente.

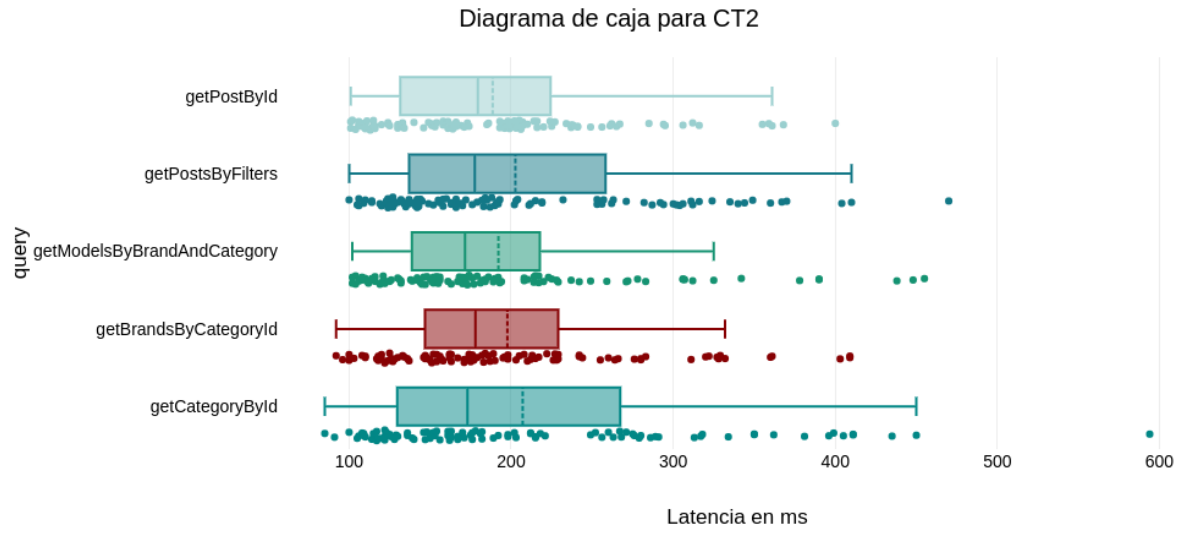

Figura 11: Presentación avanzada de resultados para CT2 I.

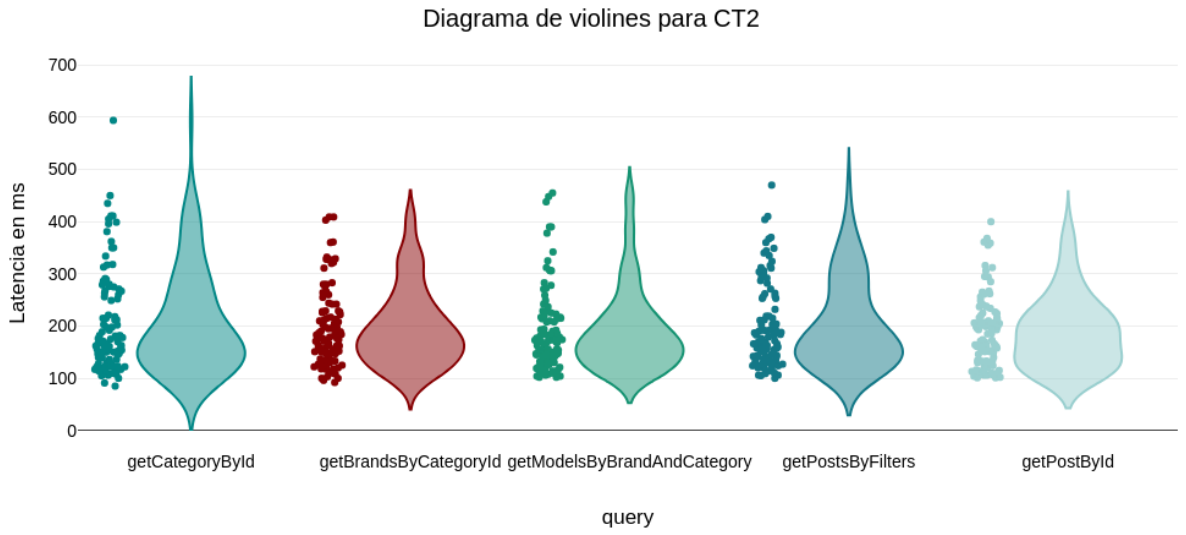

Figura 12: Presentación avanzada de resultados para CT2 II.

Observando el mapa de máximos, no se observan valores indeseables, más que en un valor de la consulta getCategoryById. De hecho, la conclusión obtenida es una vez más el excelente rendimiento de la aplicación realizada, incluso con un un volúmen de tráfico considerable.

Además, cabe destacar que estos números no reflejan el comportamiento final en despliegue ni la experiencia de usuario al consumir la aplicación, ya que el usuario siempre recibirá esta información cacheada, viéndose los tiempos de respuesta reducidos en enorme medida.

En el de mínimos, se observan unos tiempos muy positivos y alentadores, por debajo de los 100 ms. Se debe recordar que en estos tiempos también está incluido el tiempo de trayecto de ida y vuelta entre la localización de los clientes y el servidor en Londres.

Cabe destacar que no ha habido ningún error durante la operación.

3.1.3. Presentación de resultados para CT3

En este caso, se procede a analizar el comportamiento de la solución para un caso de congestión severo, en el que existen 100 usuarios recurrentes en un mismo momento consumiendo la aplicación. En la *Figura 13: Presentación de resultados para CT3,* se presentan los resultados para el tercer caso de test, ofreciendo cuatro resultados por cada query, relativos al valor medio, mediano, mínimo y máximo, obtenido por parte de los 100 usuarios concurrentes del test.

En el presente estudio, se analiza la capacidad de la API desarrollada para reponerse de una situación de congestión por medio del autoescalado, y no tanto la calidad del servicio (debido a que es un volúmen órdenes de magnitud mayor del esperado).

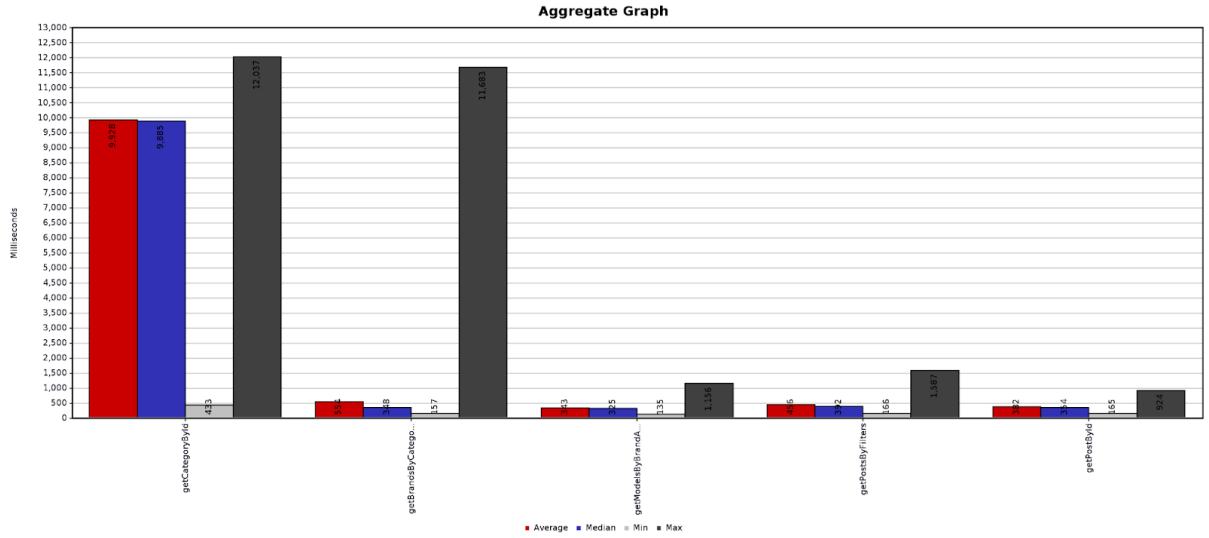

Figura 13: Presentación de resultados para CT3.

En este caso, lo primero que se puede observar en la *Figura 13: Presentación de resultados para CT3,* es que ha existido un caso de saturación en la API. No obstante, lejos de ser un mal indicador, trae consigo la genial noticia que significa la resiliencia del backend desarrollado.

La explicación radica en que, al inicio del test, Jmeter ha encolado la consulta primera, relativa a getCategoryById, en forma masiva. Siendo esta repetida en 10 de las primeras 12 consultas ejecutadas, tal y como se puede observar en la *Figura 14: Encolación masiva de la consulta getCategoryById en el CT3*. Esto ha perjudicado la salud de la API, que se ha visto obligada a encolar y autoescalar para poder servir a tantas "ConcurrentRequests", métrica del propio servidor GraphQL AppSync. No obstante, la API ha podido recuperar su nivel de rendimiento nominal al final del test, tal y como se puede observar en las métricas generales en la *Figura 13: Presentación de resultados para CT3.*

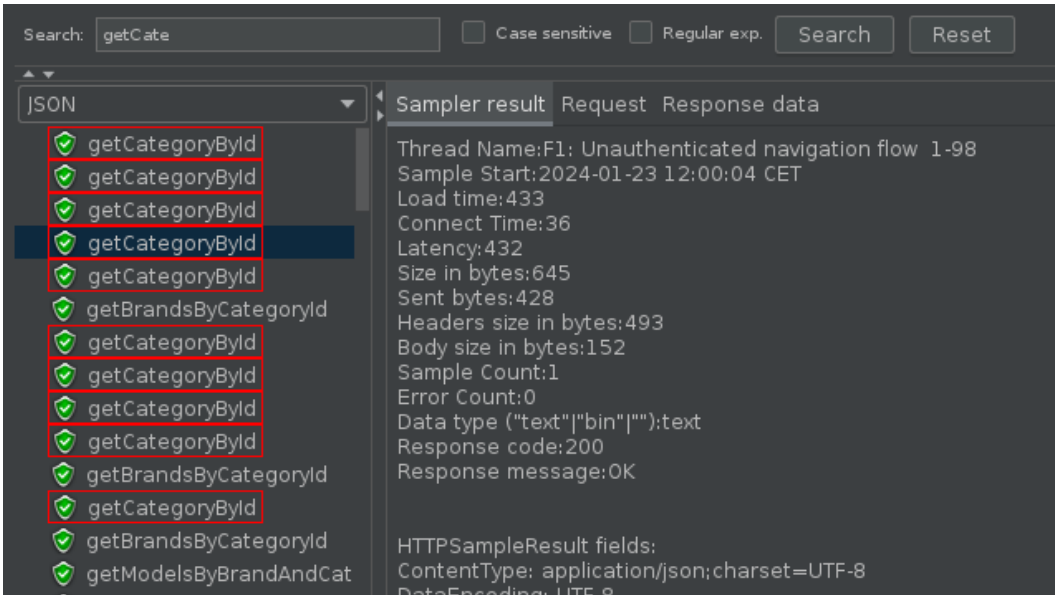

Figura 14: Encolación masiva de la consulta getCategoryById en el CT3.

Además, se puede observar que, a pesar de que el tiempo de respuesta para esta consulta concreta haya sido superior al esperado, y obviando el comentario recurrente del uso de la Caché, la totalidad de las consultas han finalizado con un código 200 OK, indicando una tasa de error del 0.00 %.

Comentado esto, las solicitudes tienen tiempos de respuesta razonables y consistentes. La operación más rápida es getModelsByBrandAndCategory una vez más, con un tiempo promedio de 343 ms. Los valores de mediana, obviando el caso de saturación, son de 348 ms, 325 ms, 392 ms, y 354 ms.

En el de mínimos, una vez más, se observan unos tiempos muy positivos y alentadores, por debajo de los 100 ms. Se debe recordar que en estos tiempos también está incluido el tiempo de trayecto de ida y vuelta entre la localización de los clientes y el servidor en Londres. Respectivamente, 433 ms

(caso de saturación), 157 ms, 135 ms, 166 ms, y 165 ms. La realidad es que estos valores reflejan un nivel de rendimiento excepcional.

El rendimiento total de la prueba es de 22.07 solicitudes por segundo, indicando la capacidad de la API para manejar la carga simulada por un usuario durante el período de prueba. En resumen, estos resultados indican un rendimiento razonable de la API bajo la carga simulada de un usuario durante la prueba. Cabe destacar que no ha habido ningún error durante la operación.

3.1.4. Presentación de resultados para CT4

Antes de presentar los resultados del cuarto caso de test, se presenta en la *Figura 15: Resultados para CT4 con autorización exigida* los valores de latencia obtenidos en la realización del test por un usuario únicamente autenticado con API Keys, y no con el JWT concreto de un usuario registrado, tras login, ejecutando operaciones sobre sus propios activos publicados.

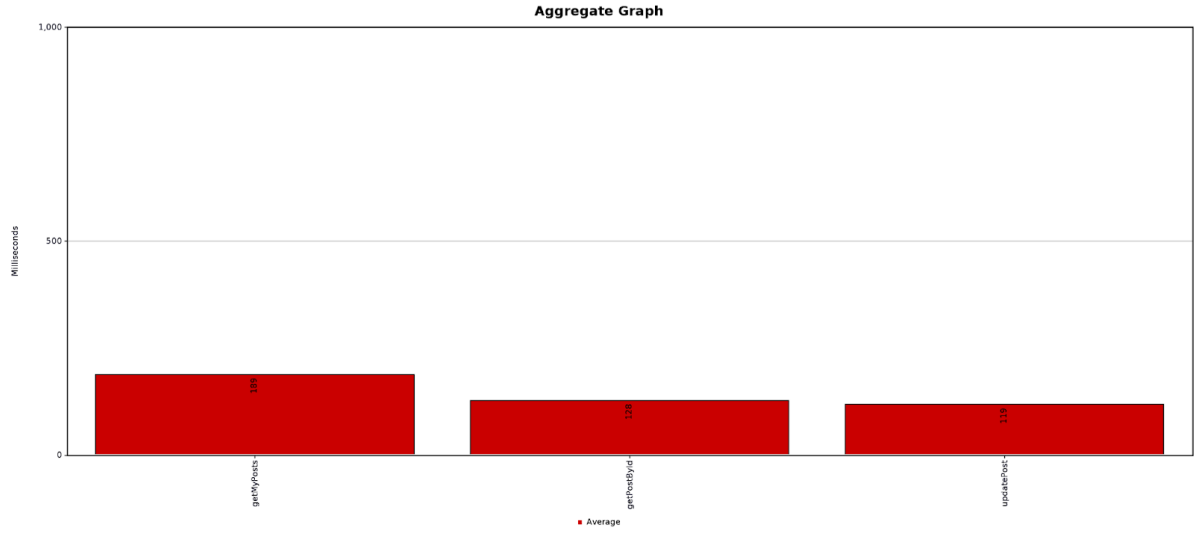

Figura 15: Resultados para CT4 con autorización exigida.

Tal y como se puede observar, se obtiene una respuesta código 401: Unauthorized, en 145 ms de media entre los 15 los requests realizados (5 usuarios por tres operaciones cada uno). Esto también refleja una buena salud en materia de excepciones; la API es capaz de identificar, gestionar y expedir excepciones al cliente entorno a los 145 ms.

A continuación, se realiza un login en la API, se obtiene el JWT, se procede a introducir en un HTTP Header Manager del propio JMeter y se vuelve a ejecutar el test, obteniendo los resultados presentados en la *Figura 16: Presentación de resultados para CT4.*

Se recuerda que este caso de test presenta una navegación realizada concurrentemente por 5 usuarios cada 10 segundos, y se ha repetido en un bucle de 20 ejecuciones, obteniendo 300 valores del análisis.

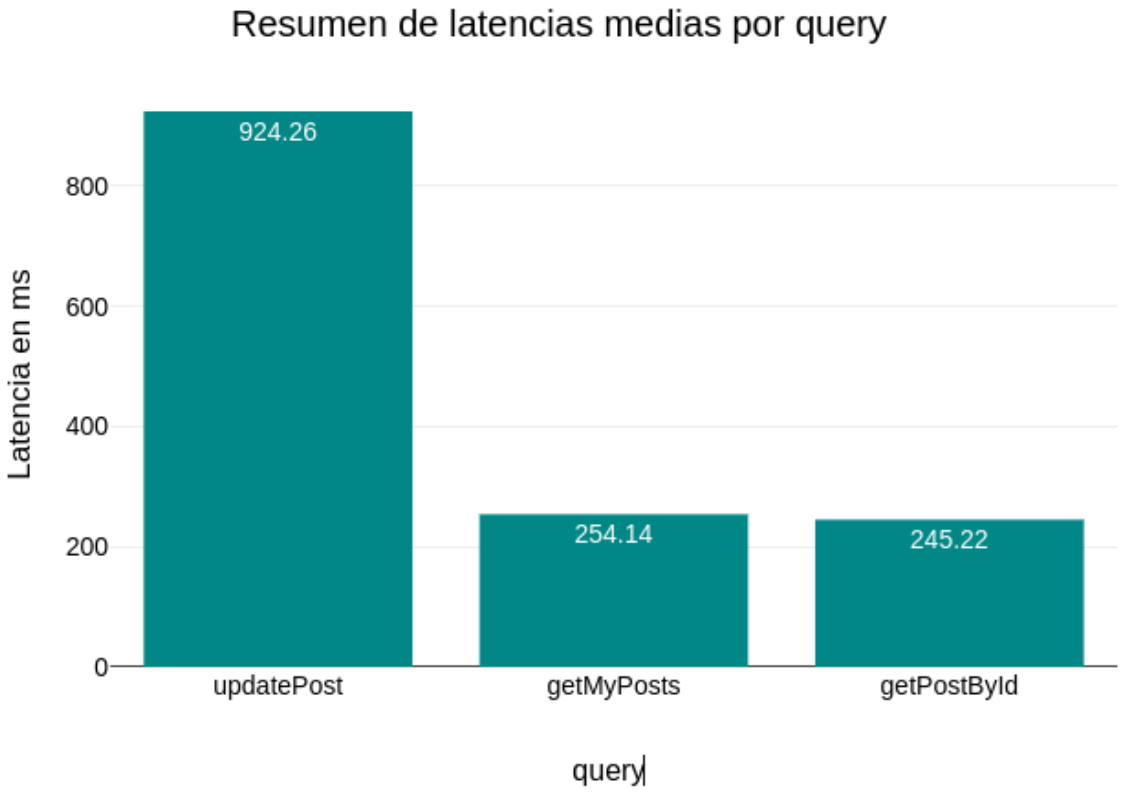

Figura 16: Presentación de resultados para CT4.

Una vez más, queda comprobado un excelente rendimiento en materia de la gestión de su información y navegación por el dashboard de cliente.

En concreto, se observan tiempos medios de actualización de un Post de 924,26 ms, valor que aunque pudiera parecer alto comparado con las operaciones de tipo SELECT (las Queries), está perfectamente dentro del benchmark. La realidad es que se trata de operaciones que garantizan la transaccionalidad, y que debido a la complejidad de la API, una actualización de una publicación provoca una modificación en las tablas de Localizaciones, Categorías y Post Meta, además de en la

propia tabla de Post. Una vez más, se presenta en la *Figura 17: Presentación de resultados para CT4 (diagrama de velas).*

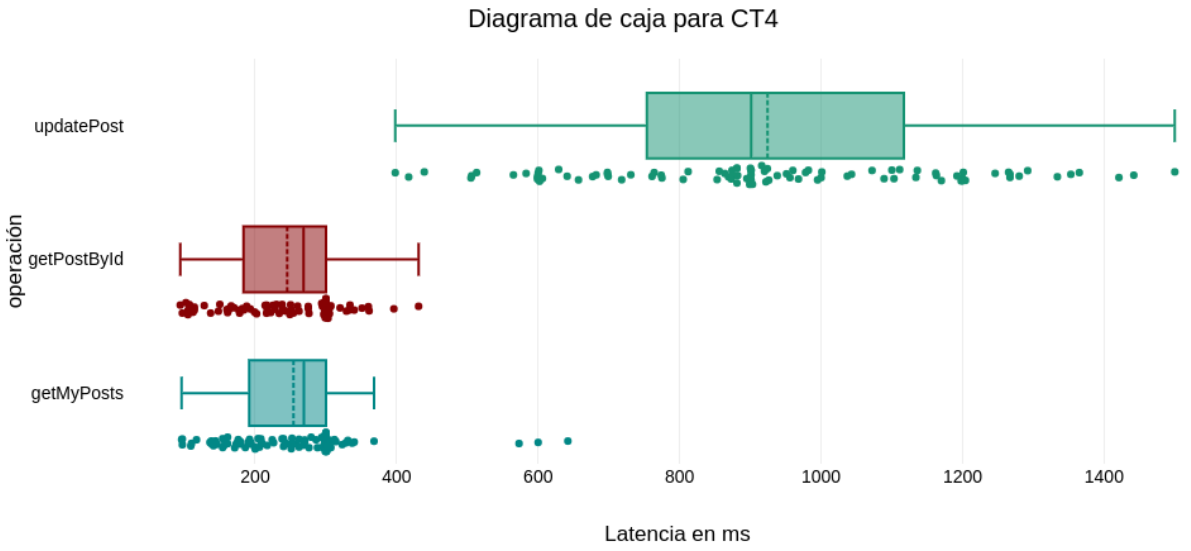

Figura 17: Presentación de resultados para CT4 (diagrama de velas).

Una vez más, se observan valores dentro de lo esperado y exigido en las especificaciones del proyecto. La autenticación por medio del JWT no supone un aditivo a la latencia.

3.1.5. Presentación de resultados para CT5

En este caso, se realiza una prueba de publicación de un Post por parte de un usuario autenticado, y se repite en un bucle de 100 para obtener estadísticos de calidad. Para ello, se realiza un login y se recoge el JWT correspondiente. El resultado obtenido del caso de test quinto se presenta en la *Figura 18: Presentación de resultados para CT5.*

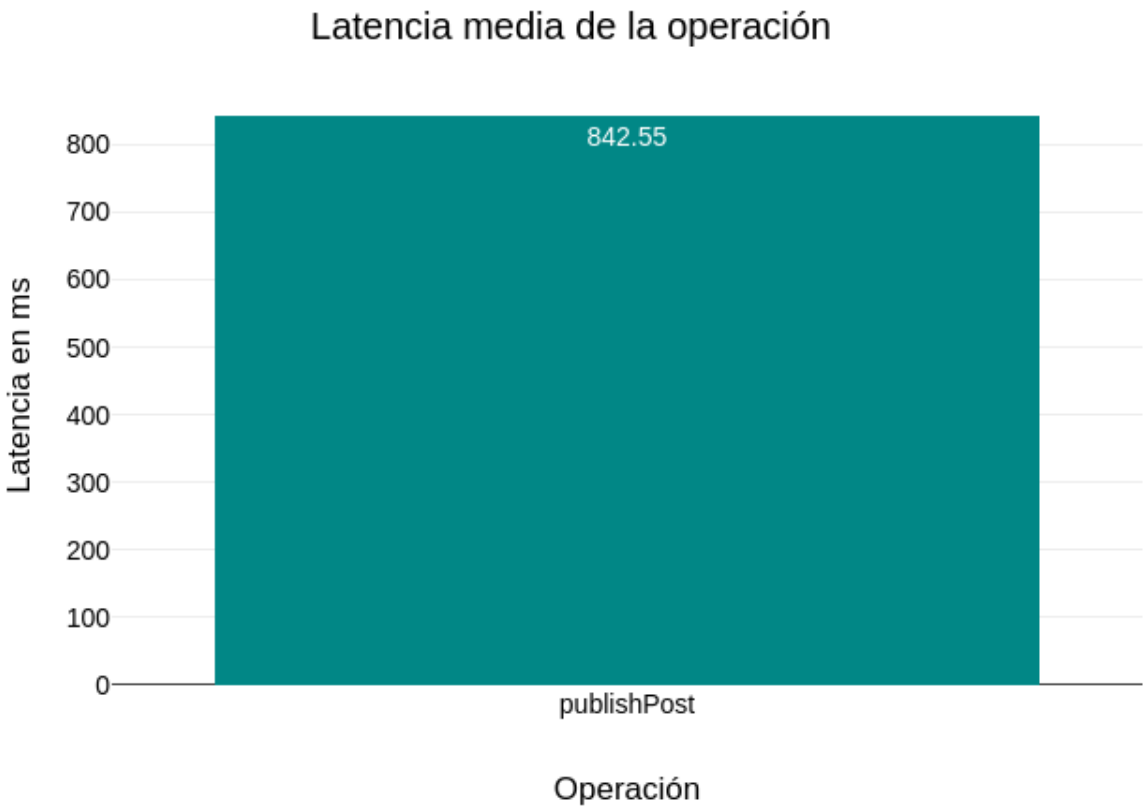

Figura 18: Presentación de resultados para CT5.

En este caso, se puede observar que el tiempo de publicación de un activo por parte de un usuario autenticado y con los roles adecuados para la publicación es de 842,55 ms. Se trata de un tiempo alto, pero dentro de las especificaciones. Esto es debido a que, tal y como ocurría en el caso de la actualización, la garantía de la transaccionalidad supone un aditivo en la latencia, además del hecho de ser una base de datos relacional, y que la publicación de un activo lleva implícitas muchas operaciones de lógica de negocio.

Finalmente, también se incluye el diagrama de violín obtenido con los resultados del presente estudio en la *Figura 19: Presentación de resultados para CT5 (diagrama de violín).*

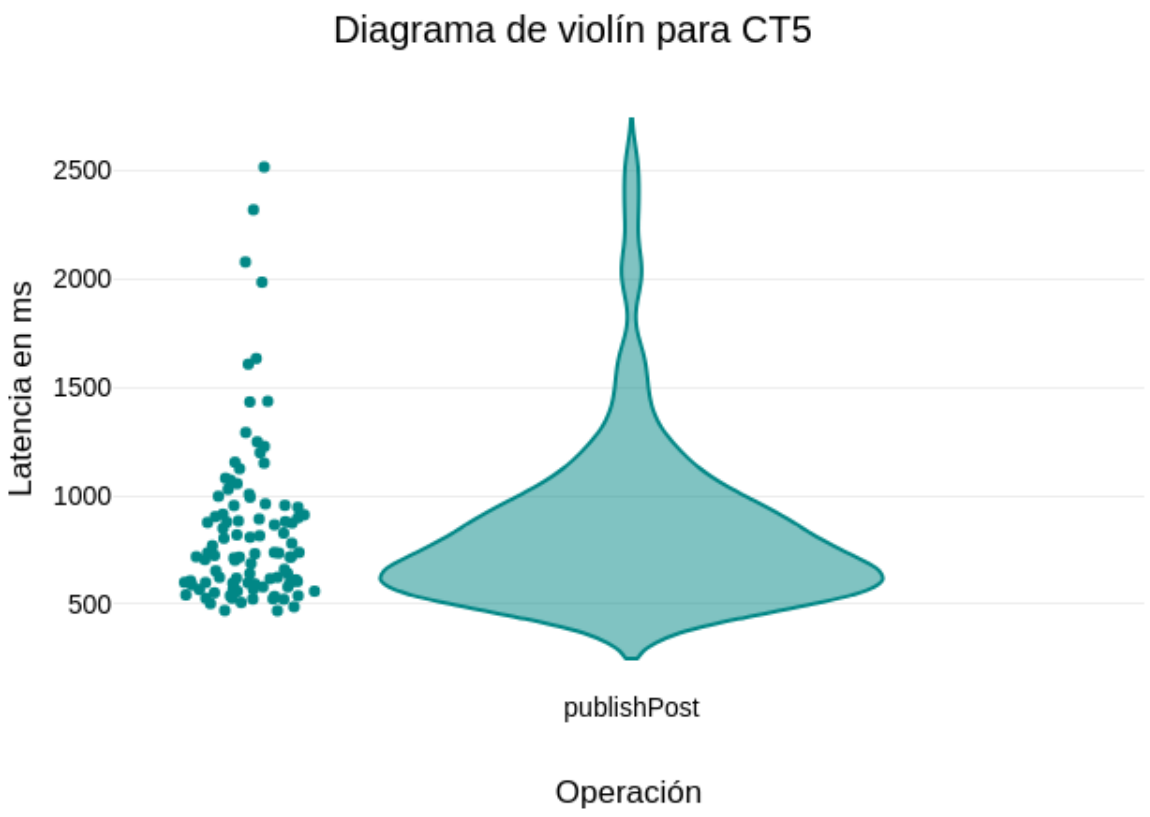

Figura 19: Presentación de resultados para CT5 (diagrama de violín).

3.1.6. Presentación de resultados para CT6

Por último, se realiza una prueba de estrés para el análisis de rendimiento asociado a la publicación de un Post por parte de un usuario autenticado. Para ello, se recupera el JWT obtenido para los casos anteriores de test. El resultado obtenido del caso de test quinto se presenta en la *Figura 20: Presentación de resultados para CT6.* Se recuerda que el test consta de la ejecución simultánea de 100 operaciones de publicación de activo de forma instantanea en una ventana de 10 segundos de tiempo.

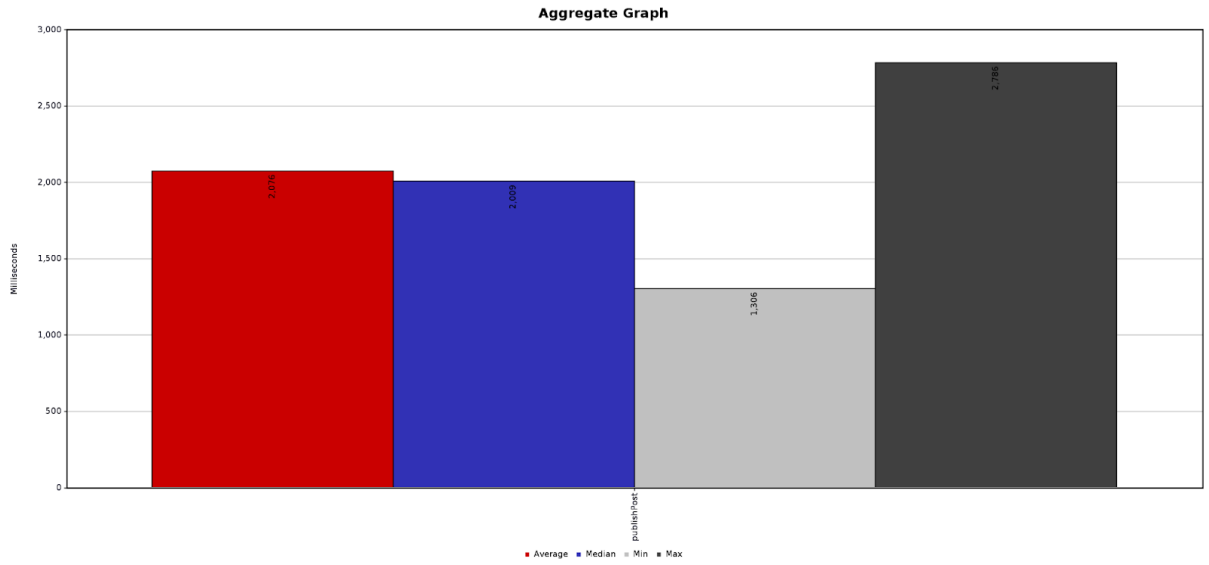

Figura 20: Presentación de resultados para CT6.

Una vez más y en último lugar, se observa la resiliencia de la API desarrollada, frente a un test de estrés basado en la publicación masiva. En concreto, se observa un tiempo medio de 2.0 segundos, un mínimo de 1.3 segundos, y un máximo de 2.7 segundos.

Por supuesto, la tasa de error ha sido del 0,00 %, y el throughput de 58.06 por minuto (coherente con los 10 usuarios en 10 segundos simulados).

4. Análisis de rendimiento por capas

Este apartado presenta un análisis detallado y pormenorizado del rendimiento de la API desarrollada. En concreto, se realiza un estudio detallado del comportamiento de cada una de las tres capas de abstracción que forman la totalidad del backend. Este estudio pretende ser de ayuda para identificar potenciales cuellos de botella además de enumerar métricas clave a seguir de cara a la gestión de la salúd por parte de cada uno de los módulos.

4.1. Rendimiento de la Data Layer

4.1.1. Comentarios previos

Por motivos de abaratamiento de costes, el cluster de base de datos desarrollado está programado para que quede en hibernación tras 10 minutos de inactividad. Por supuesto esto será actualizado en el momento del levantamiento en producción.

En segundo lugar, la activación del cluster tras una entrada en hibernación lleva asociado un "cold-start" de aproximadamente 30 segundos.

Finalmente, si bien se ha desarrollado una política de autoescalado asociada a una tendencia de congestión, la configuración por defecto es suficiente para servir el volúmen de sesiones nominal servido en el MVP.

La API implementa métodos para evitar ataques de fuerza bruta, por lo que la entrada en congestión de la base de datos únicamente es posible en entornos reales de uso.

Tras meses de desarrollo de la solución, nunca se ha llegado al autoescalado, ya que la memoria asignada está altamente sobredimensionada. Además, los tiempos de respuesta asociados a operaciones DDL se sitúa siempre en las pocas unidades de milisegundos.

4.1.2. Métricas establecidas

Para poder cuantificar y cualificar el rendimiento del cluster desplegado en la capa de datos, se generan las siguientes métricas:

- ReadLatency: El tiempo medio tomado por el disco para operaciones I/O.
- SelectLatency: El tiempo medio tomado para realizar operaciones SELECT.
- DDLThroughput: El número medio de peticiones DDL por segundo..
- Queries: Número de Queries realizadas por segundo de media.
- CPUUtilization: El porcentaje de utilización de la CPU. El valor es relativo a la memoria CPU asignada independientemente del autoescalado.

Dada la baja influencia, no se incluyen métricas de seguimiento relacionadas con latencias asociadas a operaciones INSERT o DELETE.

4.1.3. Estudio elaborado

En primer lugar, se ha realizado una prueba de cargas de tipo SELECT en busca de la saturación del disco y la entrada en autoescalado, obteniendo los resultados presentados en la *Figura 21: Resultados del estudio de rendimiento de la capa de datos.* En concreto, se puede observar las más de 200 operaciones de tipo SELECT realizadas, mediadas en ventanas de 10 segundos.

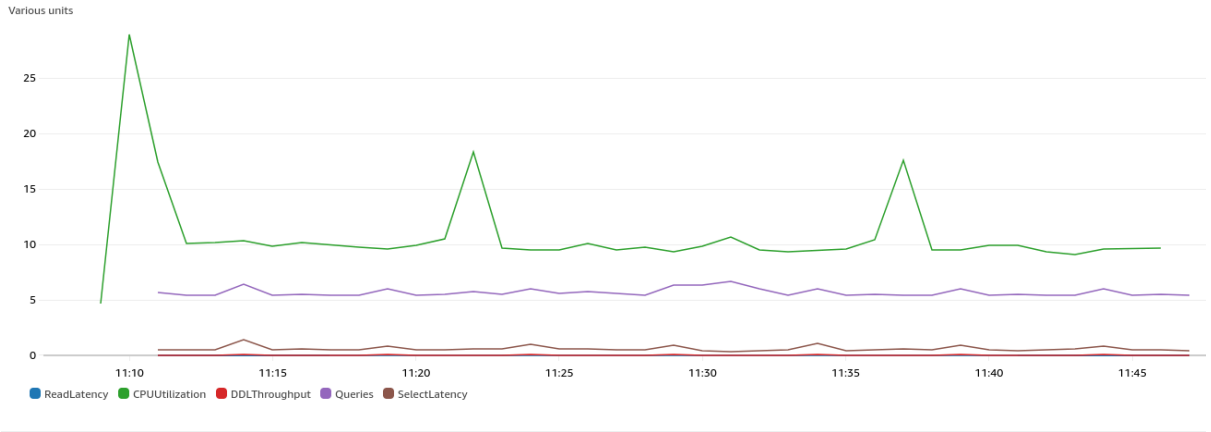

Figura 21: Resultados del estudio de rendimiento de la capa de datos.

Como se puede observar en la leyenda, los parámetros a estudiar son los definidos en el *Apartado 3.1.2: Métricas establecidas.* A simple vista, se pueden observar las siguientes reflexiones:

- El "cold-time" dura hasta 35 segundos.
- Durante este periodo, el consumo de la CPU se dispara hasta un valor dificilmente alcanzable con el volúmen nominal de sesiones en el MVP.
- La utilización del CPU ronda siempre el 10%, viéndose este superado cuando se concentran los picos del estrés del test.
- En ningún momento el estrés ha causado un nivel de utilización mayor al 20%. Y se debe reflejar que, el test suponía un orden de magnitud superior al tráfico nominal esperado. Por tanto, no se prevé la entrada en autoescalado en entornos reales en el medio plazo.
- El tiempo de respuesta de operaciones de tipo SELECT no supera en ningún caso los 10 ms.
- El tiempo asociado a la latencia de lectura se sitúa por debajo de 1 ms.

A continuación, también se ha ejecutado el Caso de Test de estrés con JMeter, con 20 usuarios concurrentes, obteniendo los resultados presentados en la *Figura 22: Resultados del Caso de Test 1 en el Panel de salúd de la Data Layer.*

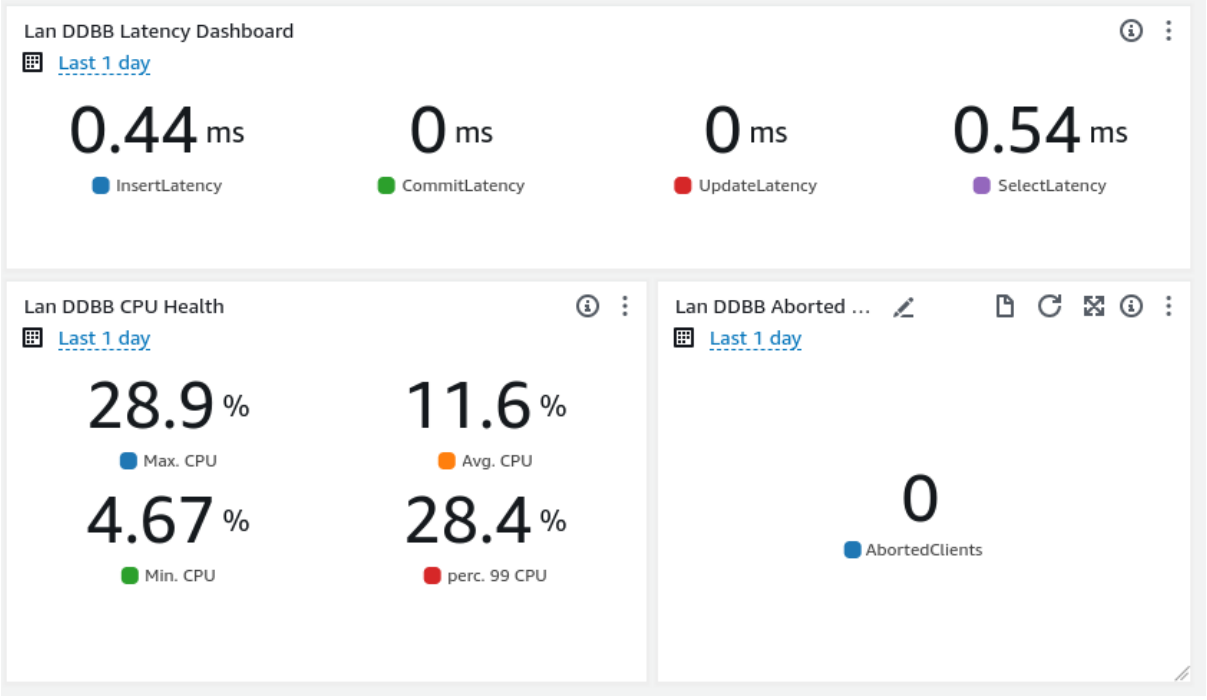

Figura 22: Resultados del Caso de Test 1 en el Panel de salúd de la Data Layer.

En general, se observa un excelente desempeño de esta capa "Data Layer". Tras el estudio, no se observan puntos de mejora a implementar en el corto ni en el medio plazo, más que el cambio de la configuración para evitar la entrada en hibernación y el consecuente "cold-start".

Se concluye que este componente va a sufrir pocos refactorings y va a acompañar la vida del proyecto durante muchos años.

4.2. Rendimiento de la Core Layer

4.2.1. Comentarios previos

El "Core Layer" está compuesto por microservicios Java, que contienen la totalidad de la lógica de negocio de la aplicación. Este componente recibe las solicitudes debidamente paradas por la API, y se comunica con la "Data Layer" para devolver el resultado esperado en cada caso.

Como puede resultar obvio, de existir, es este el módulo que impone una mayor latencia en el desglose pormenorizado de las consultas realizadas.

Este apartado tratará de identificar y cuantificar esa latencia, para que pueda ser comparada y valorada, junto a las latencias inducidas por los otros módulos o componentes en cascada de la solución desarrollada.

De igual manera que ocurría para el "Data Layer", cada uno de los microservicios tiene la capacidad de entrar en hibernación cuando no es utilizado en un periodo de tiempo de 15 minutos. Al recibir una nueva petición estando en el estado de hibernación, este sufrirá un "cold-start" de aproximadamente 10 segundos.

Por supuesto, esta naturaleza también debe ser actualizada previo el despliegue de la solución a producción.

El presente análisis de rendimiento no repara en el "cold-start", y procede directamente a analizar el comportamiento en estado nominal de cada uno de los microservicios.

4.2.2. Métricas establecidas

Para poder cuantificar y cualificar el rendimiento de los microservicios desplegados en la capa de lógica de negocio, se generan las siguientes métricas:

- Invocations: El número de veces que la función es invocada, incluyendo consultas exitosas y aquellas que resuelven en error. Es el número de requests por las que factura AWS.
- Duration: El tiempo que necesita la función para procesar una petición.
- Throttles: El número de peticiones que no pueden ser encoladas y por tanto son desechadas.
- Errors: El número de peticiones que resuelve en error.
- ConcurrentExecutions: El número de instancias del microservicio activas procesando eventos detrás de load balancer.

4.2.3. Estudio elaborado

Una vez más, se ha ejecutado el Caso de Test de estrés con JMeter, con 20 usuarios concurrentes, obteniendo los resultados presentados en la *Figura 23: Resultados del Caso de Test 1 en el Panel de salúd de microservicios, Search Service.*

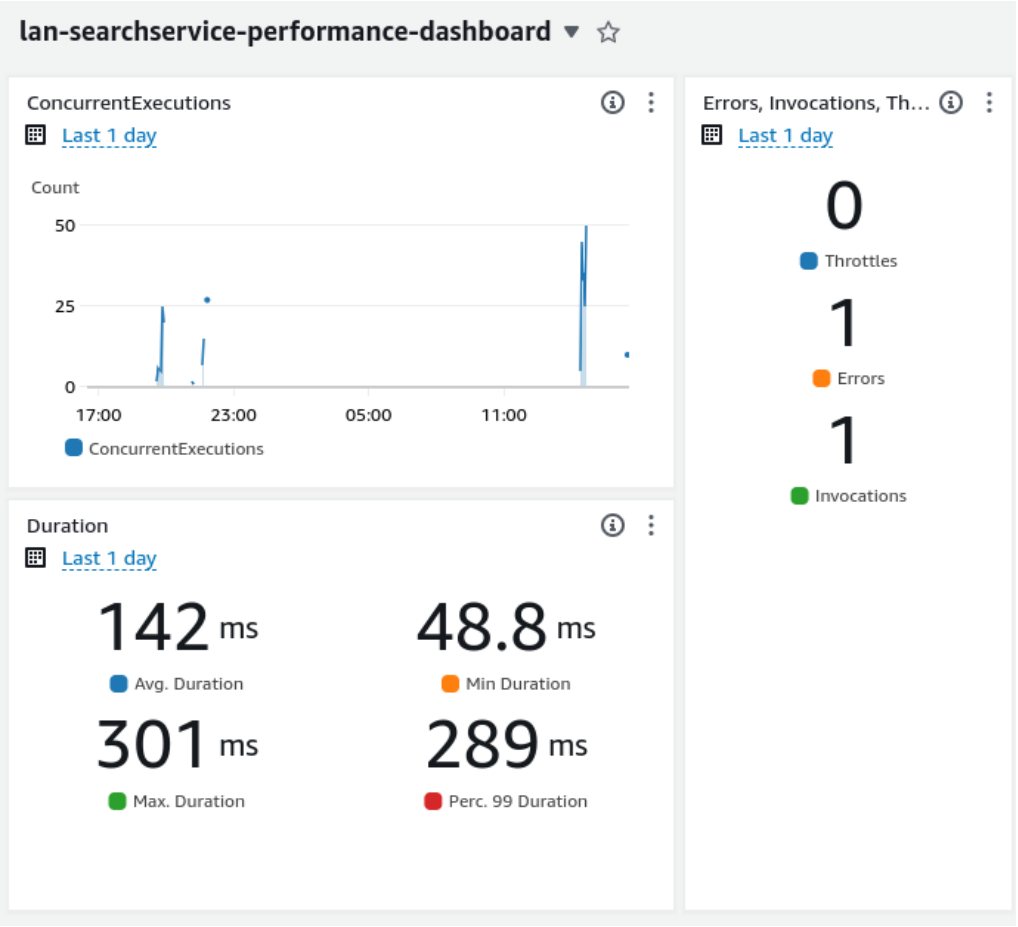

Figura 23: Resultados del Caso de Test 1 en el Panel de salúd de microservicios, Search Service.

En este caso, se extráen las siguientes conclusiones:

- El número de sesiones concurrentes escala hasta el número de consultas simultáneas. En caso de consultas espaciadas, el número de sesiones permanece en uno.
- Se ha dado un error, relacionado con la primera petición, que como se ha explicado cae en el "cold-start".
- Si bien se aprecia la diferencia en materia de tiempos respecto a la "Data Layer", las latencias están dentro de benchmark. Hay que tener en cuenta que en estos resultados no aplica la caché. Las latencias a nivel cliente deberían ser mucho menores tras la aplicación de esta.
- En general, se aprecia un excelente rendimiento a nivel core.

4.3. Rendimiento de la API Layer

4.3.1. Métricas establecidas

En este caso, debido a que la API es la parte expuesta del producto, las métricas serán más sencillas que en los casos anteriores, siendo estas:

- Latency: Latencia, en ms., de cada petición.
- API Requests: El número de peticiones a la API.
- Errores: El número de peticiones que resuelve en un error.

4.3.2. Estudio elaborado

Una vez más, se ha ejecutado el Caso de Test de estrés con JMeter, con 20 usuarios concurrentes, obteniendo los resultados presentados en la *Figura 24: Resultados del Caso de Test 1 en el Panel de salúd de API.*

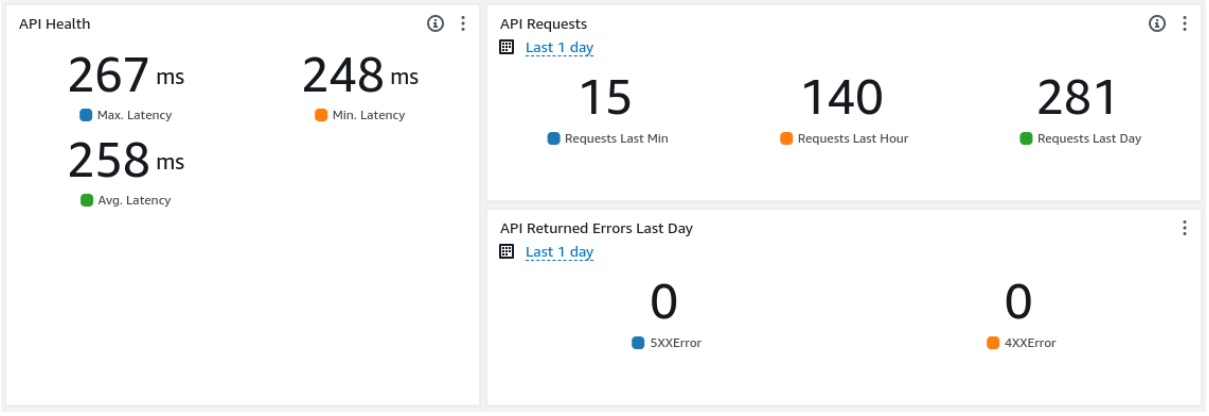

Figura 24: Resultados del Caso de Test 1 en el Panel de salúd de API.

Una vez más, la única conclusión que cabe destacar es que el rendimiento es excepcional, aún en pruebas de estrés exigentes.

5. Impacto en materia de rendimiento debido a la caché

Tal y como se explica en el *Documento Principal*, el uso de una Caché es fundamental, y permite reducir en gran medida las latencias asociadas a las peticiones de consultas.

Además, tal y como se ha desarrollado en el *Anexo II: Descripción de la solución realizada: Diseño de bajo nivel,* el servicio AWS AppSync (servicio donde está desplegado el Servidor GraphQL diseñado) dispone de múltiples configuraciones posibles para brindar la máxima flexibilidad y granularidad a la hora de configurar una memoria Caché.

Para poder hacer una demostración de la diferencia en materia de latencias, se procede a replicar el experimento CT3 (análisis de congestión), tras configurar una Caché a nivel API. A continuación se presentan los resultados en la *Figura 25: Presentación de resultados para CT3 con Caché.*

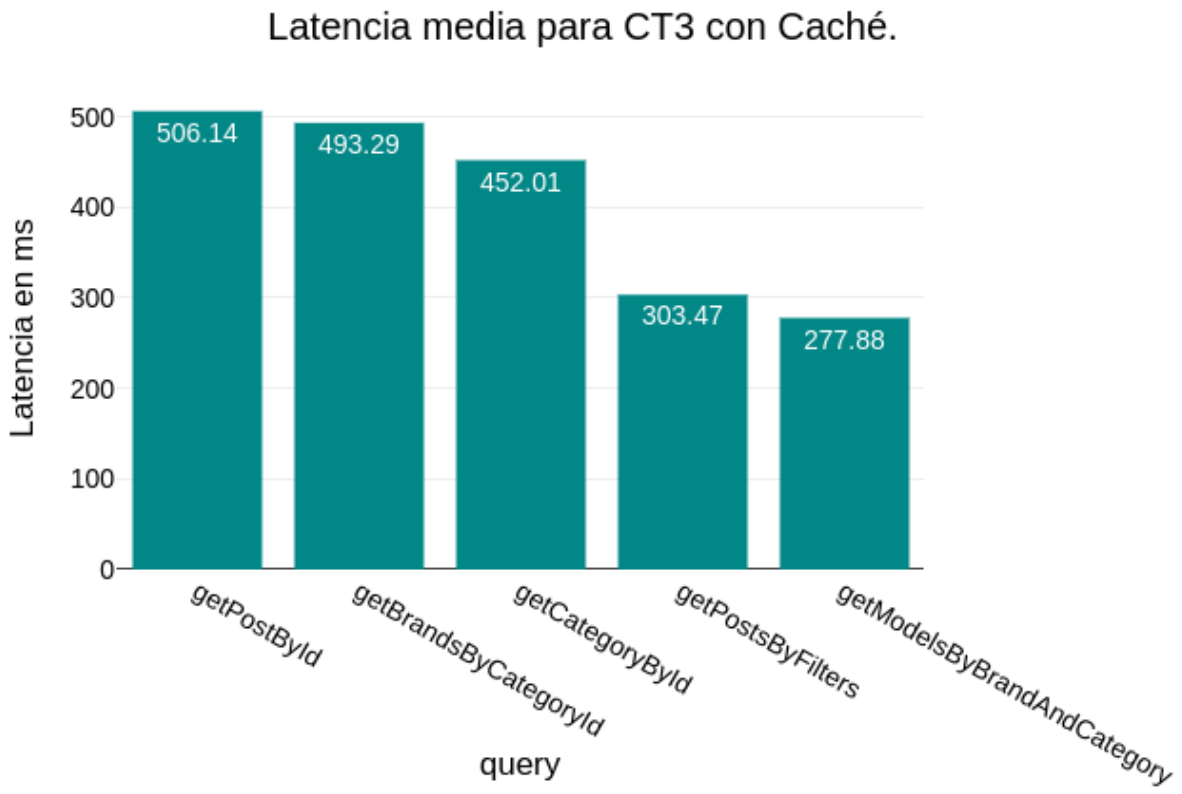

Figura 25: Presentación de resultados para CT3 con Caché.

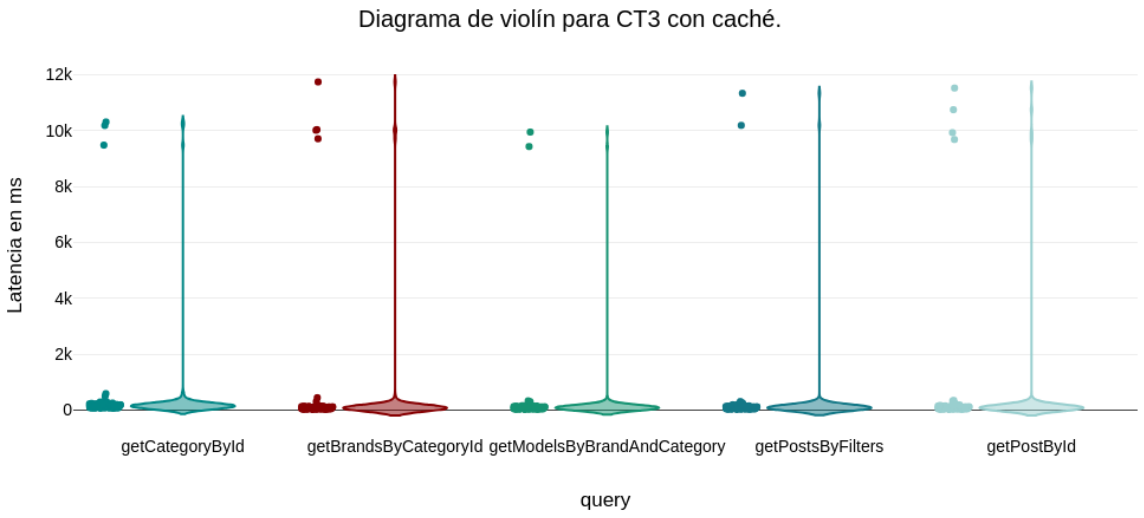

Figura 26: Presentación de resultados para CT3 con Caché.

A simple vista, se pueden observar dos fenómenos. El primero y más visual es la entrada en saturación puntual de las cinco queries, con tiempos de hasta 10 e incluso 11 segundos. Esto es algo completamente normal dado el escenario de estrés buscado, simulando 100 usuarios navegando en un mismo instante.

Sin embargo, el fenómeno de mayor importancia es la significativa reducción de los tiempos de forma generalizada. Se pueden observar unos tiempos mínimos de 70 ms, 41 ms, 40 ms, 42 ms y 38 ms, y unos tiempos medianos de 111 ms, 123 ms, 82 ms, 269 ms y 290 ms respectivamente. Ni siquiera los valores que han sufrido un encolamiento más agresivo han perjudicado significativamente los valores medios del estudio.

Para terminar, se ha querido hacer un último test de performance del F1, con Caché, con el caso de test segundo. En este caso, se trata de presentar una situación de un volúmen de tráfico significativo, pero sin llegar a valores no realistas, utilizando 20 usuarios en vez de 100.

Los resultados obtenidos son presentados en la Figura 26: *Presentación de resultados para CT2 con Caché.* En este caso, se puede observar cómo la congestión se ha extinguido, la buena salud general asociada a los tiempos de latencia, y el buen rendimiento que ofrece la Caché configurada. En este caso, los valores medios de latencia obtenidos han sido de 197,45 ms, 186 ms, 178,90 ms, 144,65 ms y 152,75 ms respectivamente.

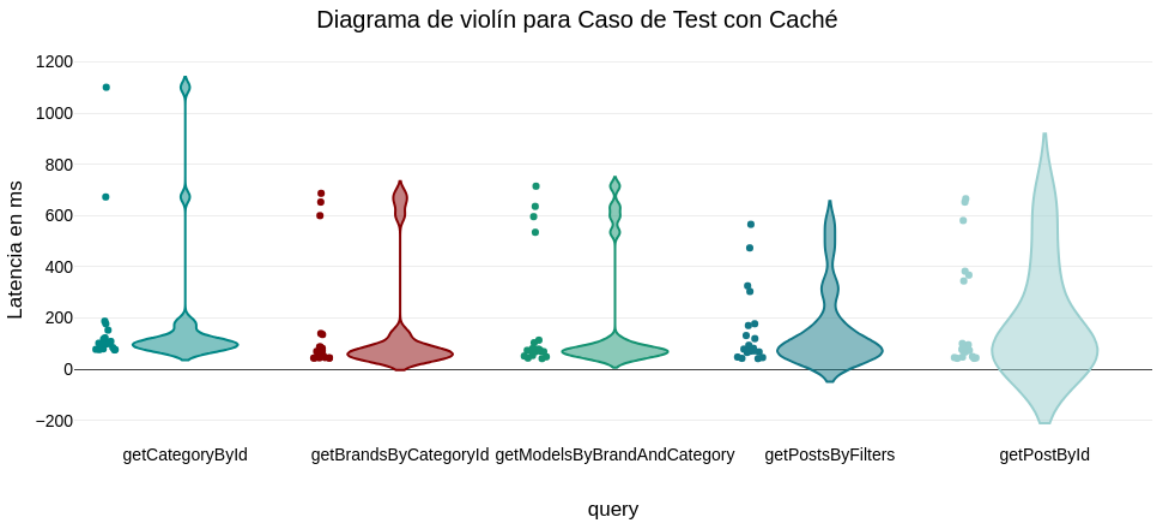

Figura 27: Presentación de resultados para CT2 con Caché.