UNIVERSIDAD PONTIFICIA DE SALAMANCA
FACULTAD DE INFORMÁTICA
Grado en Informática

Trabajo Fin de Grado

Estudio y análisis de los beneficios de la programación reactiva.

Iván Castillo Villanueva

Dr. Roberto Berjón Gallinas
Director

Salamanca, junio de 2023

Resumen

En base al aumento de la necesidad de aumentar la capacidad de manejo de los datos, de la velocidad y la disminución del uso de recursos, nace la programación reactiva. Esta metodología se centra en un nuevo enfoque de programación que se basa en las secuencias de eventos y los flujos de datos para lograr estos beneficios.

Se explicará en que consiste este nuevo método, sus ventajas y cómo se puede codificar, abordando las entrañas de este para explicar cómo se dan estos flujos de datos. Además, se tratarán diferentes principios que se deben cumplir y opciones para poder cumplir con ellos.

Ahora bien, para ilustrar las diferencias entre la programación tradicional y la programación reactiva, podemos desarrollar dos aplicaciones prácticas. En una de ellas utilizaremos el enfoque tradicional y secuencial, mientras que en la otra aplicaremos el enfoque reactivo. Ambas aplicaciones deberán cumplir con los mismos requisitos y se analizará el rendimiento de cada una en particular.

Al comparar los resultados, se podrán apreciar las ventajas de la programación reactiva, en términos de eficiencia, escalabilidad y capacidad de respuesta ante eventos en tiempo real. El ahorro de tiempo y recursos que ofrece este nuevo enfoque puede marcar una gran diferencia en el desarrollo de aplicaciones que requieren un alto rendimiento y una gestión óptima de los datos.

Abstract

Based on the increasing need for increased data handling capacity, speed and decreased resource usage, reactive scheduling was born. This methodology focuses on a new programming approach that relies on event sequences and data flows to achieve these benefits.

It will be explained what this new method consists of, its advantages and how it can be coded, addressing the guts of it to explain how these data flows occur. In addition, different principles to be complied with and options to be able to comply with them will be discussed.

Now, to illustrate the differences between traditional programming and reactive programming, we can develop two practical applications. In one of them we will use the traditional, sequential approach, while in the other we will apply the reactive approach. Both applications will have to meet the same requirements and the performance of each will be analyzed in particular.

By comparing the results, it will be possible to appreciate the advantages of reactive scheduling, in terms of efficiency, scalability and responsiveness to real-time events. The time and resource savings offered by this new approach can make a big difference in the development of applications that require high performance and optimal data management.

Descriptores

Secuencia de eventos, flujo de datos, ahorro de recursos y manejo de gran cantidad de datos.

Índice

Índice de Figuras

Índice de Tablas

1. Introducción.

La programación reactiva se ha convertido en un enfoque fundamental para desarrollar aplicaciones modernas en diversos campos, que van desde el desarrollo web hasta las aplicaciones empresariales de alto rendimiento. En este trabajo, se explorará la programación reactiva en el contexto del lenguaje Java, uno de los lenguajes de programación más populares y ampliamente utilizados en la industria.

La programación reactiva en Java se basa en una serie de principios y conceptos clave, como la programación basada en eventos, la gestión de flujos de datos y la capacidad de respuesta. Al utilizar estas técnicas, los desarrolladores pueden crear sistemas altamente escalables y resilientes, capaces de manejar grandes cantidades de datos y de ofrecer experiencias de usuario fluidas y receptivas.

En este trabajo, se explorará cómo implementar la programación reactiva en Java, desde el uso de bibliotecas y marcos de trabajo específicos, hasta el diseño de patrones y mejores prácticas. También se discutirán las ventajas de la programación reactiva en Java.

Al comprender los fundamentos de la programación reactiva en Java, los desarrolladores podrán aprovechar al máximo las capacidades del lenguaje y sus herramientas para crear aplicaciones robustas y altamente eficientes en términos de uso de recursos y rendimiento.

1.1 Objetivos.

El objetivo principal de este trabajo es proporcionar una comprensión clara de los conceptos fundamentales de la programación reactiva y cómo se aplican en el entorno de desarrollo Java. Exploraremos los conceptos clave, como la reactividad, la programación basada en eventos y la gestión de flujos de datos.

De este, derivarán otros considerados como secundarios. Pero por ello también será importante tenerlos en cuenta.

Se examinarán las ventajas asociadas con la programación reactiva en Java. Esto incluirá discusiones sobre el rendimiento, la escalabilidad, la gestión de errores y la complejidad del código, entre otros aspectos relevantes. Al comprender tanto los beneficios como las posibles dificultades, los desarrolladores podrán tomar decisiones informadas al implementar la programación reactiva en sus proyectos.

A lo largo del trabajo, se presentarán ejemplos prácticos de implementación de programación reactiva en Java, junto con mejores prácticas para diseñar y desarrollar aplicaciones reactivas eficientes. Estos ejemplos ayudarán a los desarrolladores a comprender

cómo aplicar los conceptos teóricos en situaciones reales y mejorar su habilidad en la programación reactiva en Java.

Al alcanzar estos objetivos primarios y secundarios, este trabajo proporcionará una visión integral de la programación reactiva en Java, permitiendo a los lectores adquirir los conocimientos y habilidades necesarios para implementar sistemas reactivos eficientes y escalables en el entorno de desarrollo Java.

1.2 ¿Qué es la programación reactiva?

La programación reactiva es un tipo de programación novedoso, enfocado desde un punto de vista totalmente distinto que cada vez se utiliza con mayor frecuencia ya que introduce un cambio notable en los requisitos necesarios dentro del entorno de las solicitudes, y, a través de un conjunto de principios, posibilita la creación de diferentes sistemas y aplicaciones logrando un resultado con un mayor grado de eficiencia, escalabilidad, resiliencia y simultaneidad. Pero, esto no es todo, si no que, además, se logra el soporte de una mayor carga si la comparamos con el sistema que se ha utilizado tradicionalmente, mediante el uso de los recursos que se encuentran disponibles de forma bastante más eficiente, como pueden ser la CPU y la memoria. Esto se puede traducir como la manejabilidad de una cantidad de datos superior, que se realiza, principalmente, mediante flujos.

1.3 Características de la programación reactiva.

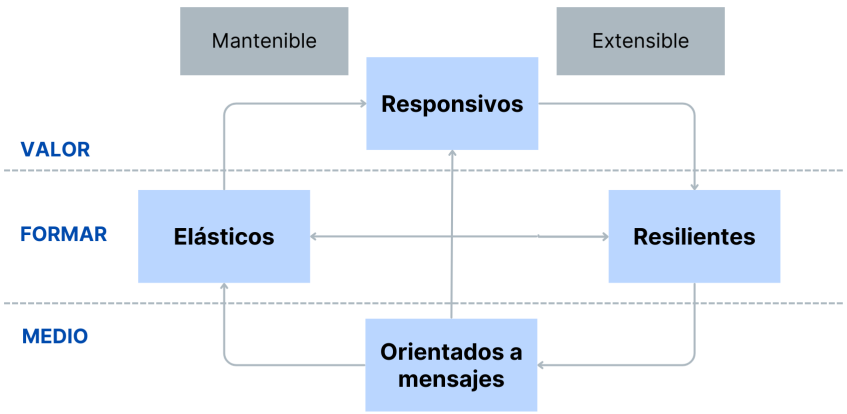

Figura 1-1 Características de los sistemas reactivos

Basándonos en lo anterior podemos decir que un sistema que se considere reactivo deberá cumplir en todo momento las siguientes especificaciones.

1.3.1 Respuesta.

Una de las características a tener más en cuenta dentro de las principales es la capacidad de respuesta ya que es muy importante en lo que entromete a usabilidad y utilidad.

Pero esto no es todo ya que tener la capacidad de encontrar los problemas de manera simple y abordarlos eficazmente está implícito en ofrecer una capacidad de respuesta que se considere

adecuada. Es por esto por lo que los sistemas reactivos ponen el foco en ofrecer consistencia y rapidez en lo que a los tiempos de respuesta se refiere.

Además, lo que se logra gracias a la simplificación en el manejo de los errores es mejorar la confianza del usuario considerado como final. Y, de este modo, se promueve una mayor interacción con el sistema.

1.3.2 Resistente.

El sistema seguirá respondiendo a pesar de que se produzca un error. Esto no sucede en cualquier sistema que esté considerado como no resiliente ya que no tendrá esta capacidad frente la aparición de un error.

Gracias a los conceptos de replicación, contención, aislamiento y delegación podemos conseguir esto. Concretando más, el objetivo es que el sistema se pueda recuperare de los fallos que se produzcan en partes concretas de este, sin que influya en el resto del sistema. Esto se puede conseguir gracias al tratamiento de los fallos dentro del propio componente en el que se produzcan. Mientras que, a su vez, todos los componentes se encuentran aislados unos de otros.

Además, para garantizar una alta disponibilidad, la recuperación individual de un componente después de un fallo se delega en otro componente externo distinto. Para esto se puede hacer uso de la replicación si es necesario.

De esta forma, es prácticamente imposible que un cliente se vea sobrecargado por el manejo de fallos que se han producido en el componente que actúa como proveedor de eventos.

1.3.3 Elástico.

El sistema seguirá respondiendo independientemente de que la carga de trabajo que reciba sea variable. Los sistemas reactivos son capaces de modificar la cantidad de recursos asignada a una solicitud. Además, pueden reaccionar ajustando dicha cantidad según la carga de trabajo que reciban.

Con diseños de este tipo se evita de forma total la problemática relacionada con los cuellos de botella centrales y los puntos de contención. Esto supone que se tenga la capacidad de distribuir entradas entre distintos componentes y de fragmentar o replicar estos mismos.

Los sistemas distribuidos permiten algoritmos de escala predictivos y reactivos, proporcionando medidas relevantes de rendimiento en vivo.

1.3.4 Impulsado por mensajes.

El uso de la mensajería asíncrona es lo que permite que unos componentes no dependan de otros. Así, a la hora de colaborar unos con otros no existirá una gran dependencia entre ellos. De este modo se logra un mayor aislamiento debido a la transparencia de ubicación, que se resume en que cada componente no necesita conocer la ubicación del resto para realizar las funciones que le sean asignadas.

Además, mediante esta separación entre componentes es como se proporcionan los medios para poder emitir los fallos que se han producido como si fueran mensajes, o lo que posteriormente veremos como emitir un evento de fallo. Comunicar los fallos mediante mensajes que contienen la información de estos permite la gestión de carga, la elasticidad y el control de flujo. Todo esto a partir del seguimiento y la configuración de las colas de mensajes en el sistema y la aplicación de back-pressure, siempre y cuando sea necesario. Esto consiste en comunicar al componente que se encuentra emitiendo eventos que no se dispone de la capacidad de procesar tal cantidad, por lo que deberá disminuirla.

Que la gestión de los fallos funcione con las mismas construcciones y semántica a través de un clúster o dentro de un solo host se hace posible gracias a que el medio de comunicación es la mensajería asíncrona en un entorno de transparencia de ubicación. Dicha comunicación sin bloqueo permite consumir recursos a los destinatarios, es decir, suscribirse a un objeto emisor de eventos y reaccionar ante ellos una vez que los ha recibido. Pero solo lo podrán hacer mientras se encuentren activos, lo que supone que el sistema se encuentre menos sobrecargado.

2. Estado del arte.

2.1 Los principios de la programación reactiva.

Cuando nos adentramos en el desarrollo de una aplicación nativa, ya sea en entornos en la nube, Edge o IoT, nos enfrentamos al reto de crear y ejecutar un sistema que se va a encontrar distribuido en un hardware y redes que no inspiren confianza. Para que una aplicación sea considerada reactiva, es esencial que se adhiera a una serie de principios tanto en su diseño como en su arquitectura y, en muchas ocasiones, también en su modelo de programación.

La distinción entre una aplicación reactiva y una no reactiva radica en su enfoque para afrontar los problemas. Una aplicación reactiva aborda los problemas que surgen debido a algún fallo y los que puede presentar cada solicitud en sus abstracciones, modelos de programación, protocolos, interacciones y en el manejo de errores, además de otros aspectos relacionados con el diseño y la arquitectura. Por otro lado, otras aplicaciones afrontan estos problemas de forma improvisada, centrando su atención en cuestiones operativas u otras relacionadas con la infraestructura. Por lo que, para resolver estos problemas, recurren a pilas tecnológicas cada vez más complejas, envolturas, soluciones y abstracciones problemáticas.

2.1.1 Mantenerse receptivo.

Consiste en tener la capacidad de poder responder siempre de la mejor manera. Esta es la parte más importante para el usuario ya que este no prestará atención a toda la parte que conforma el sistema, sino a que este consiga realizar la función para la que fue diseñado en el menor tiempo posible.

Aunque esto es lo principal, no lo es todo, ya que, además de responder rápidamente, cosa que es sencilla cuando no se produce ningún fallo, para que un sistema se considere receptivo es necesario que también lo sea cuando surgen los problemas, como puede ser demasiada carga de trabajo o que se produzca un fallo de manera inesperada, y ahí gestionar los cambios que sean necesarios y reflejarlos posteriormente.

Lo primordial en una aplicación reactiva será tratar de responder siempre de la manera más adecuada, pero hay ciertas situaciones en las que esto no es posible, incluso cuando dicha respuesta simplemente sea la propagación de un error o la devolución de parte del servicio que se estaba completado antes de que se produjera el fallo. Es decir, si la solicitud corresponde a una lista de apartamentos, devolver los apartamentos que se consultaron hasta que se produjo el error. Pero aun así hay veces en las que esto no es posible.

Con este nivel de respuesta se ayuda a una detección de los errores más sencilla y se consigue evitar en todo momento unos tiempos de respuesta elevados, creando así un sistema compacto en cuanto a las respuestas, ya que une todos los elementos del sistema y los comunica entre sí, logrando, por tanto, una mayor confianza del usuario en la aplicación.

2.1.2 Aceptar la incertidumbre.

Consiste en que a pesar de que la aplicación esté desarrollada sobre unos recursos que no transmiten confianza conseguir que esta sea fiable.

El principal problema a la hora de diseñar un sistema es conocer lo que se sucede en los diferentes nodos de este al mismo tiempo, ya que no disponemos de una memoria de almacenamiento mediante la cual podamos guardar todos los datos necesarios para poderlo conseguir. Además, no se considera eficiente estar esperando la información que necesitamos hasta que esta esté completa si esto supone un tiempo excesivo. Esto conlleva que en el momento en el que los componentes de la aplicación tengan que tomar decisiones lo harán basándose en una información y en unos datos que no estarán totalmente actualizados o completos.

Existen algoritmos para lidiar con este problema, pero en un sistema de carácter reactivo no son útiles, ya que, aunque solucionan esto, no son capaces de ofrecer una respuesta rápida, por lo que no cumplen el principio anterior.

Sin embargo, la falta de una información segura y actualizada se puede solucionar teniendo en cuenta desde el momento inicial en el que diseñamos el sistema la incertidumbre de no disponer de unos datos en perfecto estado. Se debe conseguir que el sistema esté compuesto por componentes autónomos, es decir, que no dependan de otras partes del sistema y no las necesiten para operar, sino que esté extremadamente claro cuáles serán sus funciones y cuál será el proceso que hay que seguir para acceder a ellas, conociendo el efecto que estas producirán sobre los datos de los que disponemos en la aplicación.

De esta manera se podrá determinar el grado de fiabilidad que está ofreciendo el sistema en el momento actual.

2.1.3 Aceptar el fracaso.

Consiste en diseñar teniendo en cuenta que, seguramente, en algún momento suceda algo que derive en un fallo de manera inesperada y, por tanto, habrá que reponerse ante eso.

Al desarrollar una aplicación es necesario la creación de métodos para la gestión de fallos, ya que es un suceso que seguramente ocurra. Esta gestión se puede realizar en el mismo componente, lo que se denomina redundancia interna, o mediante la implementación de un componente externo que se encargue de supervisar la gestión de los fallos una vez que se hayan producido.

Hay que recordar la importancia de dar una respuesta a la solicitud siempre que el sistema esté capacitado para ello. Debido a esto, es por lo que el componente, al realizar la gestión del fallo que se ha producido, tiene la capacidad de enviar parte del servicio que debía de procesar. Por ejemplo, supongamos que se le ha solicitado consultar una lista de localidades, y que en mitad del proceso se produce un fallo, en vez de no enviar nada, emitirá la lista de localidades que le dio tiempo a consultar y, posteriormente, informará del fallo. De este modo en vez de no enviar nada cuando se produce un fallo, podrá emitir los eventos que tuviera disponibles en lugar de no informar al usuario y no devolver nada útil.

Es por esto por lo que el modelo de arquitectura ideal sería el que fuese capaz de una vez que se ha producido el fallo, detectarlo, tratarlo, notificarlo y repararse. Aunque otros tipos de arquitecturas proponen parar de inmediato la ejecución, matar el subproceso de trabajo y volver a empezar desde el principio.

También existe la posibilidad de que el fallo que se ha producido sea indetectable, pero, en principio, debido a todo lo anterior no debería afectar en gran medida al funcionamiento general de la aplicación.

2.1.4 Conseguir el mayor grado de autonomía posible.

Consiste en diseñar el sistema de manera que absolutamente todos los componentes que lo forman sean autónomos, pero trabajen colaborando entre sí unos con otros.

Esta autonomía se consigue aislando los componentes unos de otros y comunicándolos solo para lo estrictamente necesario. Además, es importante especificar donde se encontrarán los datos y cómo se puede acceder a ellos. Unificando ambos procedimientos se pueden obtener componentes que tengan la capacidad de toma de decisiones, sin la necesidad de que tener que recurrir a otros debido a la falta de información o a que se encuentran trabajando de manera conjunta. Así, este mismo componente podrá decidir si responder o no en situaciones en las que esté sobrepasado por la carga de trabajo a la que está sometido ya que esta decisión solo depende de él. Debido a esto es imprescindible que el protocolo que se utilice esté basado en eventos y sea asíncrono.

También es importante que la colaboración entre los distintos componentes se realice solo y exclusivamente por los caminos que han sido diseñados para ello, excluyendo totalmente la posibilidad de puertas laterales que puedan comunicar unos con otros sin que esto este contemplado. De esta manera se puede verificar que se respeta la autonomía de los distintos componentes y que pueden colaborar entre sí de manera correcta.

Esta forma de comunicación de los diferentes eventos entre componentes permite que el componente que los recibe tenga la capacidad de toma de decisiones sin tener la necesidad de tener que recurrir a la ayuda de terceros.

2.1.5 Consistencia a nivel de componente.

Consiste en conseguir un equilibrio entre el mejor rendimiento que pueda ofrecer la aplicación y mantenerse disponible el mayor tiempo posible. Esto se puede lograr gracias a cambiar el foco de atención del sistema global a cada componente, de manera que se consiga que estos sean lo suficientemente consistentes.

La consistencia trata de asegurar que la aplicación y los datos del usuario que se emplean sean correctos y no estén corrompidos. Partiendo de esto es importante entender que añadir más garantías de consistencia de las que son necesarias simplemente producirá un retardo en la disponibilidad de la aplicación y esto será perjudicial para el rendimiento y la disponibilidad, por lo que el usuario final se verá afectado y esto no será ventajoso.

Podemos ver los sistemas reactivos como grupos de componentes en los que, cuando es necesario, varias partes colaboran para lograr dar respuesta a una solicitud de servicio, pero sin que estos lleguen a unirse por completo. Es por esto por lo que cada componente debe ser consistente a nivel individual, de manera que, gracias a esto, el resultado de las colaboraciones entre componentes sea prácticamente predecible.

Generalmente, no es necesario desarrollar los componentes con una fuerte consistencia, ya que el procesamiento asíncrono permite retrasos en la información y falta de disponibilidad de

algún componente en ciertos casos. Esto es perfecto ya que cuando el sistema esté disponible conseguirá colaborar y proporcionar el servicio solicitado por el cliente, y, si no, tendrá la capacidad de recuperarse.

Aunque, en el caso de necesitar una consistencia fuerte, debe intentarse que el tamaño de las unidades en las que se implementa sea lo más pequeño posible.

2.1.6 Desacoplamiento temporal.

Consiste en procesar de forma asíncrona la mayor cantidad de solicitudes posibles. De esta manera se evita la coordinación entre diferentes partes del sistema, y, por tanto, se reduce la espera que produce la comunicación necesaria para dicha coordinación.

A medida que evitamos la coordinación, y, como consecuencia, la comunicación entre componentes conseguimos un sistema más escalable y limitamos la demora a la hora de dar respuesta a las solicitudes.

Mediante el desacoplamiento temporal eliminamos los tiempos de espera a la hora de interactuar con servicios remotos. Esto se debe a que como hay momentos en los que la otra parte no se encuentra disponible no se asume que lo está. Así se consigue que los componentes sean más independientes y, de nuevo, más autónomos, logrando que el sistema en general sea más fiable y rápido.

Además, gracias a esto, el componente que realiza la llamada tiene la posibilidad de no bloquear el hilo de ejecución. Por el contrario, podrá continuar ejecutando otro proceso de manera asíncrona, pero será necesario que deje registrada una devolución de llamada para que en el momento en el que se complete la primera pueda volver a ella.

Con todo esto se consigue que cada componente pueda procesar la información cuando él considere oportuno, teniendo la capacidad de ejecutar según sus prioridades.

2.1.7 Desacoplamiento espacial.

Consiste en distribuir los componentes del sistema de una manera flexible según el hardware del que se disponga.

Es necesario distribuir los componentes entre todo el hardware del que dispongamos. Esto se debe a que de este modo el sistema seguirá funcionando aun cuando algún elemento del hardware no lo haga. Así se planteará un sistema dividido por partes pero que se acoplará, siempre y cuando sea necesario, para ofrecer un servicio lo más rápido posible. Esta comunicación entre componentes que se encuentran ubicados en zonas diferentes se consigue gracias al paso de mensajes asíncrono.

Este uso de la mensajería asíncrona obliga a diseñar el sistema pensando en que seguramente en algún momento falle. La ventaja es que ayudará a la autonomía de la aplicación, ya que gracias a ella no es necesario que un componente conozca en qué lugar se encuentra el resto, consiguiendo que estén todavía más aislados unos de otros. Esto se denomina transparencia de ubicación.

Otro modo de aumentar la disponibilidad y la resistencia del sistema frente a posibles fallos es el uso de la replicación. Esto se trata de ejecutar diferentes instancias del mismo componente para poder dividir la carga de trabajo entre ellas, sucediendo lo mismo si una de ellas directamente no se implementa o se bloquea. Esto es vital a la hora de lograr que el servicio nunca sea interrumpido.

2.1.8 Manejar la dinámica.

Consiste en que el sistema se adapte constantemente a las diferentes variaciones producidas tanto en la demanda de trabajo, como en los recursos de los que se dispone.

Una aplicación reactiva debe ser elástica, es decir, debe de ser capaz de satisfacer la demanda en todo momento y, además, debe regular la asignación de recursos en función de esta. También, en el caso de que los recursos de los que dispone el sistema sean fijos, la cantidad de datos recibidos se debe ajustar correctamente con respecto a estos y, por supuesto, informar de dicho ajuste para que ningún emisor la sobrepase.

Todo esto se ve reflejado en que puede que haya momentos en los que se vea reducida la eficiencia de la aplicación debido a que la cantidad de datos con los que se opere sea menor. Pero, por el contrario, aumentaremos la disponibilidad de los componentes ya que nos aseguraremos de que en ningún momento estén sobrecargados gracias a estos ajustes en la tasa de entrada.

Pero para que esto sea posible es necesario que el componente sea autónomo. De este modo no dependerá de otros a la hora de repartir adecuadamente la carga de trabajo entre los recursos que tenga disponibles y conocerá en todo momento el estado en el que se encuentra la aplicación. De esta manera se podrá entender cómo se está haciendo frente a la carga actual y, por tanto, cómo hacer unas mejores estimaciones sobre dicha carga de trabajo.

Los sistemas reactivos están bajo diferentes dinámicas dependiendo del tipo de aplicación, por lo que también será necesaria la correcta elección entre desacoplamiento temporal o espacial dependiendo de la evolución del sistema.

2.2 Los patrones de la programación reactiva.

Es posible que los patrones expuestos a continuación ayuden a aplicar y codificar los principios reactivos en los sistemas y aplicaciones. Estos patrones no son aplicables a todos los problemas, pero exponen opciones interesantes para tener en cuenta que pueden ser usadas y aplicadas de manera selectiva cuando hagan falta.

No está pensado como una serie de patrones fijos que supongan la totalidad de estos, sino como una muestra de algunos de ellos.

2.2.1 Dividir el proceso entre diferentes nodos.

Consiste en aprovechar el paralelismo de los sistemas reactivos para poder brindar un servicio mediante una carga computacional menor. Esto se consigue al dividir el proceso que se está ejecutando actualmente en varios y, así, repartir la carga de trabajo.

Del mismo modo que distribuir partes del sistema entre el hardware ayuda a la escalabilidad, distribuir la ejecución actual entre diferentes nodos conlleva que se puedan resolver problemas computacionales de un tamaño mayor. Esto se debe a que un nodo tiene una capacidad de procesamiento de datos finita, sin embargo, al combinar varios esta aumenta de manera significativa.

Estas particiones se realizan con la intención de que sean paralelas ya que se están ejecutando más o menos al mismo tiempo, pero con la diferencia de que sean lo suficientemente independientes como para que no necesiten unas de otras. Para ello hay datos que hay que dividir con cuidado ya que no es tan sencillo lograr esa independencia a la hora de ejecutar.

Este estilo conlleva que se produzca una reducción de la consistencia y de la sencillez del sistema al tener que emplear varios componentes para un solo proceso. Pero, por otro lado, aumenta tanto el rendimiento, como la escalabilidad y mejora las reacciones ante fallos.

2.2.2 Comunicar datos que hayan sido contrastados.

Consiste en la elección en todo momento de conjuntos de eventos que no se vayan a modificar frente a otros que se estén utilizando en la ejecución y, por tanto, se están actualizando.

Parte de los datos que se utilizan en una ejecución no son estables, ya que están representando continuamente el último valor. Esta modificación continua supone que sobrescriben el nuevo valor sobre el anterior. Esto conlleva en gran parte a una corrupción o, incluso, a una pérdida de los datos debido a la continua actualización de estos.

Igual que con la mayoría de los problemas, para este ya existen soluciones, pero los algoritmos dedicados a esto son muy complejos. Además, exigen un alto grado de coordinación, lo que supone una pérdida de escalabilidad y de rendimiento debido a la cantidad de comunicaciones que esto necesita.

Entonces, en vez de esto, la mejor solución es confiar y basarse en los datos que tenemos de momentos pasados. Datos que son consistentes y que se pueden compartir sin preocupación. Esto se debe a que como ocurrieron en el pasado ya no se pueden cambiar y, por tanto, para refutarlos harían falta otros datos contrarios. De todos modos, aunque esto sucediera no se eliminarían, si no que dejarían de ser relevantes para la fase en la que se encuentra el sistema actualmente.

Estos conjuntos de datos seguros se pueden compartir mejor como secuencias de eventos. Por tanto, pueden ser suscritos y consumidos, lo que significa ser utilizados, por otros componentes que necesiten conocer los diferentes cambios a los que ha sido sometido un determinado componente.

2.2.3 Aislar los componentes que no ejecuten datos seguros.

Consiste en utilizar divisiones para contener y aislar la zona siempre que se estén usando datos que se van modificando en la ejecución.

Siempre que se esté haciendo uso de este tipo de datos en un componente será necesario el aislamiento de este con respecto a los demás. Hay que asegurarse de que no haya nada que se comparta entre este y el resto. Con esto lo que se consigue será evitar problemas con los fallos ya que se gestionarán directamente en dicho componente y no se propagarán hacia el resto. De este modo se evitarán posibles fallos en cascada.

Con este método se aislará el componente tanto de manera temporal como en espacio, haciendo uso de la mensajería asíncrona para comunicarle con los demás. Este tipo de mensajería también tiene la capacidad de evitar que dicho componente se sobrecargue de trabajo mediante el uso de la replicación.

En definitiva, se deben usar los mínimos datos variables posibles, y, si los usamos, hacerlo en una zona del sistema lo más pequeña posible. Una vez que estos datos han sido procesados, lo mejor será comunicar los resultados como un valor estable que no vaya a ser actualizado.

Gracias a esto los demás componentes pueden hacer estimaciones basándose en los datos seguros que han sido producidos. Y, al mismo tiempo, internamente puede seguir beneficiándose del procesamiento de algunos datos que son actualizados en tiempo de ejecución.

2.2.4 Coordinar el flujo de datos.

Consiste en crear y organizar un flujo de información que sea constante y, por tanto, no tenga interrupciones.

A la hora de diseñar un sistema es necesario tener en cuenta el tipo de procesos que se ejecutarán en cada componente, aparte, de la estructura general del sistema. Con esto se podrá permitir que unos componentes se puedan suscribir de manera asíncrona a los flujos de eventos que emitan otros y así los puedan recibir y reaccionar ante ellos.

Además de esto, hay que conseguir que los componentes sean capaces de controlar la tasa de consumo y la de producción. De esta manera será prácticamente imposible que un componente se sobrecargue debido a que en todo momento tiene bajo control lo que está sucediendo y la carga que él mismo puede soportar.

Sin embargo, hay zonas donde es necesario tener especial cuidado con los flujos de datos. Estas zonas suelen estas situadas donde interactúan unos componentes con otros o con algún servicio externo. En estas habrá que desarrollar funciones que tengan la capacidad de reducir los flujos de información o simplemente mantenerlos bajo control para que el componente no se vea sobrepasado.

De esta manera se evitan una gran cantidad de fallos y, en caso de que surja alguno, también es útil para gestionarlos.

2.2.5 Localizar en un mismo lugar los datos y el procesamiento de estos.

Consiste en mantener en el mismo componente el procesamiento de los datos y los propios datos durante el tiempo que dure la ejecución.

Mantener en la misma ubicación estos dos conceptos mejora el rendimiento y los tiempos de espera. Esto se debe a que consigue que las cargas de trabajo estén mejor repartidas y, por tanto, sea más sencillo procesarlas.

Hay dos maneras de lograr esto. La primera es trasladar la ejecución al lugar en el que se encuentran los datos necesarios para que esta se pueda realizar. De este modo se considera que los datos que se van a utilizar, y que, por tanto, están almacenados en memoria, son totalmente verídicos y no están alterados. Este proceso podría tener cierta similitud con la memoria caché, pero, sin embargo, este tipo de almacenamiento se mantiene conectado al origen de los datos y esto disminuiría la consistencia del sistema.

La segunda es trasladar los datos al lugar en el que se deben procesar. Esto se puede conseguir mediante la replicación de los datos a todos los lugares en los que podría darse dicha ejecución. Esto produciría un aumento de la disponibilidad, además de aumentar también la consistencia de los propios datos. Esto se debe a que se encontrarían disponibles en varios componentes, que sería algo similar a una copia de seguridad.

2.2.6 Observar las comunicaciones.

Consiste en observar la mayor cantidad de métricas posibles con el objetivo de entender el sistema por completo.

Observar un sistema internamente es más que necesario, ya que es la única forma de conocer cómo se encuentra actualmente. Pero no solo esto, sino que también ofrece información que indica cómo se ha encontrado anteriormente.

Esto se consigue mediante el seguimiento de métricas y en las aplicaciones reactivas es esencial. Pero, además, en estas es necesario hacer un seguimiento también de las

comunicaciones que se han producido. Observar esto puede brindar una gran cantidad de conocimiento acerca de cómo se han ido distribuyendo los flujos de datos y qué componentes han sido receptores y productores de estos.

Mediante esto se puede lograr ajustar mucho mejor las distribuciones. Lo que conlleva satisfacer de manera más eficiente la demanda, además de tener mucha más información a la hora de decidir sobre la elasticidad del sistema. Que hay que recordar, es la manera en la que se distribuye dicho sistema sobre los recursos hardware disponibles.

2.3 Beneficios de la programación reactiva.

Desarrollar aplicaciones y sistema siguiendo el modelo de programación reactiva produce una serie de beneficios que mejoran notablemente el modelo tradicional.

Estos beneficios son los siguientes.

2.3.1 Uso de los recursos más eficiente.

En el modelo de programación tradicional un hilo de ejecución emplea una gran parte del tiempo en esperar a que le respondan, principalmente en operaciones en las que está involucrada la E/S. Esto se evita mediante el modelo reactivo ya que una vez que llega a ese punto continúa ejecutando otra petición. Esto desemboca en que en todo momento se asegure de que cada subproceso se encuentra trabajando en resolver alguna petición y no ocupando recursos para, simplemente, esperar respuestas de servicios externos.

Utilizar de manera eficiente los recursos deriva en un menor gasto económico. Pero no solo esto, sino que también aumenta la capacidad de soportar una carga de trabajo mayor con los recursos disponibles actualmente.

2.3.2 Mantenibilidad.

En el modelo reactivo, el código necesario para resolver cualquier problema, por muy complejo que sea, siempre será muy autodescriptivo. Esto se debe a la potencia de la que disponen los operadores y a que la base de este modelo son los flujos de eventos.

Esto posibilita que, aunque ya no sea el desarrollador original, se entienda a la perfección la lógica del sistema de manera sencilla y relativamente rápida, posibilitando mejoras o mantenimiento del propio código.

2.3.3 Escalabilidad.

El modelo reactivo se basa en el mínimo acoplamiento posible y el aislamiento de fallos. Esto junto con que el centro de su funcionamiento sea el manejo de flujos de eventos, hace que sea posible gestionar cantidades de eventos prácticamente impensables. Lo que es muy útil para cualquier trabajo en el que se necesite la gestión de muchos datos, como puede ser el Big Data.

2.3.4 Manejo sencillo de la cantidad de envíos del emisor.

Hay situaciones en las que se puede llegar a dar que el emisor esté emitiendo un número de eventos mayor del que el consumidor puede procesar, esto reduce la velocidad de respuesta y, por tanto, ralentiza el sistema.

Esto se gestiona en el modelo reactivo ya que el suscriptor puede indicar el número de elementos máximos a recibir, por lo que el emisor nunca enviará más y, por tanto, no saturará al consumidor.

2.3.5 Implementación sencilla.

Este último apartado está ligado al de mantenimiento ya que las razones son similares. Resolver problemas que serían complejos con el modelo tradicional se vuelve mucho más sencillo con el reactivo, lo que conlleva que su adopción, en cuanto a uso, crezca de manera exponencial.

2.4 Arquitectura reactiva de Quarkus.

Quarkus ofrece la posibilidad de crear aplicaciones reactivas. Pero no solo esto, ya que también da la posibilidad de poder desarrollar estas por partes, es decir, es posible implementar zonas de manera reactiva y otras de manera tradicional. Una vez construida la aplicación funcionarán de manera conjunta. Gracias a esta posible combinación de ambos tipos no hay necesidad de utilizar diferentes herramientas, pilas o APIs.

2.4.1 Aplicaciones Reactivas.

Las aplicaciones reactivas son capaces de continuar gestionando las solicitudes aun encontrándose ante una carga de demanda variable y con fallos.

Quarkus facilita en gran medida el proceso de creación de un sistema reactivo. Pero también se asegura de que los principios reactivos sean cumplidos adecuadamente por cada elemento del sistema que los quiera utilizar. Contando, además, con que tenga un grado de eficiencia elevado.

La eficiencia es lo más importante de un sistema que es capaz de proporcionar un servicio, sobre todo si se trata de entornos cloud o de contenedores. Es por esto, que las aplicaciones desarrolladas de manera tradicional consumen tanta memoria y CPU. El modelo tradicional supone una eficiencia menor, además de que, como los recursos son compartidos entre varias aplicaicones, un uso elevado de una sola aplicación perjudicará la ejecución del resto. Lo que se verá resumido en una disminución de la eficiencia o, por el contrario, de la necesidad de aumentar los recursos disponibles. Ninguna de las dos es una solución óptima.

Actualmente la E/S es una de las partes más utilizadas de prácticamente cualquier aplicación, de hecho, hay toda clase de operaciones basadas en ella. Es por esto por lo que no gestionarla de manera adecuada puede llevar a un escenario muy similar al anterior, en el que la aplicación se vuelva ineficiente y ofrezca unos tiempos de respuesta extremadamente lentos.

Frente a todo esto Quarkus utiliza una E/S no bloqueante. Lo que supone que solo sean necesarios un número de hilos de ejecución ínfimos en comparación con los necesarios de forma tradicional para gestionar un número de E/S todavía mayor al que se gestionarían con el anterior modelo. Pero, además, empleando menos recursos, tiempo y, por tanto, aumentando la eficiencia.

Netty y Eclipse Vert.x son los encargados del motor de carácter reactivo que utiliza Quarkus detrás de todo el código, anotaciones, etc. y es mediante este como se gestionan las E/S sin bloqueo.

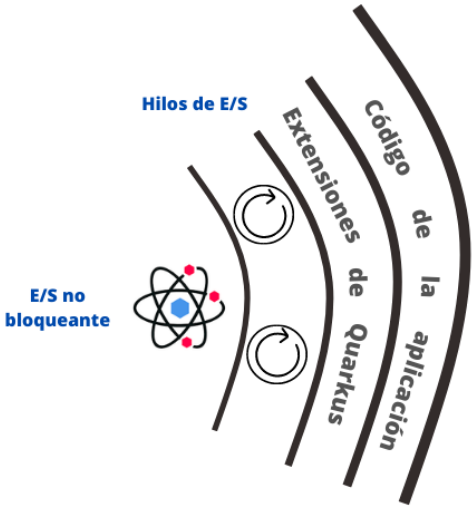

Figura 2-1 Esquema del modelo reactivo de Quarkus

Tanto el código desarrollado de la aplicación, como las propias extensiones de Quarkus, son las que tienen la capacidad de hacer uso de este motor para comunicarse con bases de datos, gestionar operaciones del tipo E/S, comunicarse mediante mensajería asíncrona, etc.

2.4.2 Modelo de ejecución reactiva.

El uso de la programación reactiva y su característica más importante, que es la E/S sin bloqueo, aportan una gran cantidad de beneficios. Pero, por el contrario, implica desarrollar basándose un modelo de ejecución muy diferente al utilizado tradicionalmente.

Los sistemas y aplicaciones desarrollados hasta ahora mediante el método tradicional están basados en un modelo de ejecución secuencial y en una E/S bloqueante. Eso significa que se irán ejecutando las peticiones en orden y solicitando operaciones de E/S esperando a que se finalice la anterior.

Explicado desde el punto de vista de una aplicación cuyo final es mostrar un HTTP, cada vez que se produzca una solicitud a esta aplicación primero deberá esperar a que se finalice con la que se estuviera ejecutando anteriormente. Una vez que esto ha sucedido, empleará un hilo de manera única y exclusiva para esta solicitud, de manera que la procesará al completo y solo una vez que termine será liberado. En resumen, mientras este ejecutando una petición ese subproceso de ejecución no tendrá la capacidad de atender nada más. Si encima le añadimos a esa solicitud una necesidad de utilizar un proceso que conlleva una E/S, ese hilo quedará bloqueado hasta que llegue el resultado de dicha operación y luego continuará con la misma solicitud.

Todo esto supondrá un tiempo perdido que solo está dedicado a la espera y que ralentiza la aplicación. Este modelo es sencillo de desarrollar al tratarse de un modelo secuencial, pero no es el óptimo. Esto se debe a que si se desea poder gestionar varias solicitudes al mismo tiempo será necesario crear un subproceso para cada solicitud. Lo que se ve derivado en un grupo de hilos de ejecución que solo aumenta el uso de los recursos y ralentiza el sistema.

Todo esto resulta en aplicaciones ineficientes.

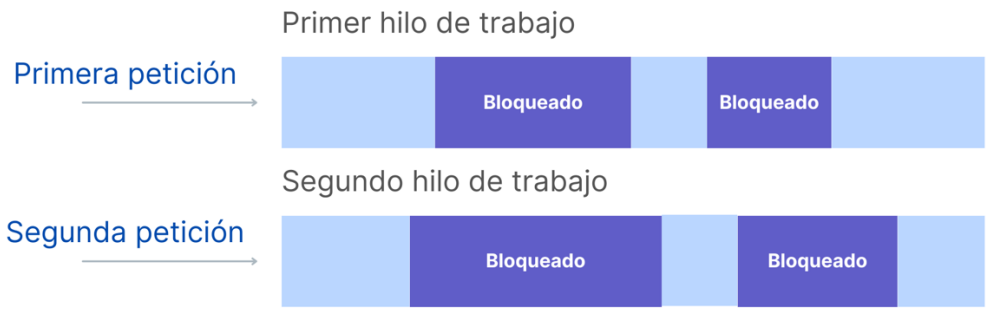

Figura 2-2 Esquema representativo de hilos de ejecución en el modelo bloqueante

Esto se soluciona gracias al uso de la E/S sin bloqueo. Esta permite la ejecución y gestión de múltiples operaciones relacionadas con la E/S al mismo tiempo y mediante un mismo hilo de ejecución.

La solicitud se ejecutará en un subproceso, pero en el momento que necesite comunicarse con un servicio remoto ya no bloqueará el hilo. Para esto requerirá que una vez programada la E/S se le administre una continuación mediante la que seguirá ejecutando. Una continuación es una indicación que señala sobre que fragmento de código debe de continuar. Una vez que el proceso de E/S se haya completado satisfactoriamente el hilo volverá a dicha solicitud y la finalizará.

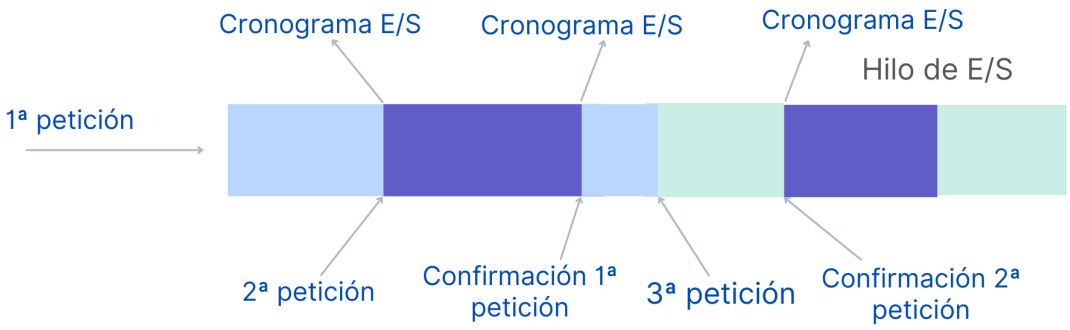

Figura 2-3 Esquema representativo de hilos de ejecución en el modelo reactivo

Por lo tanto, el modelo reactivo es mucho más eficiente. El único pero es que hace falta una forma concreta de implementar las continuaciones que permitan seguir ejecutando mientras se está realizando la operación de E/S sin bloqueo mediante código.

2.4.3 Unificación del modelo secuencial y el reactivo.

Quarkus es reactivo gracias al motor reactivo que posee. Pero desde el punto de vista de la implementación de aplicaciones, al tener la posibilidad de crear un conjunto formado por la parte del código secuencial y la parte del código reactivo, no es necesario crear código de carácter estrictamente reactivo.

Esto conlleva la posibilidad de desarrollar aplicaciones mediante el código bloqueante tradicional. Entonces, para evitar bloquear subprocesos mediante la E/S, Quarkus cambia de subproceso de trabajo siempre que sea necesario, lo que se conoce como proactor pattern.

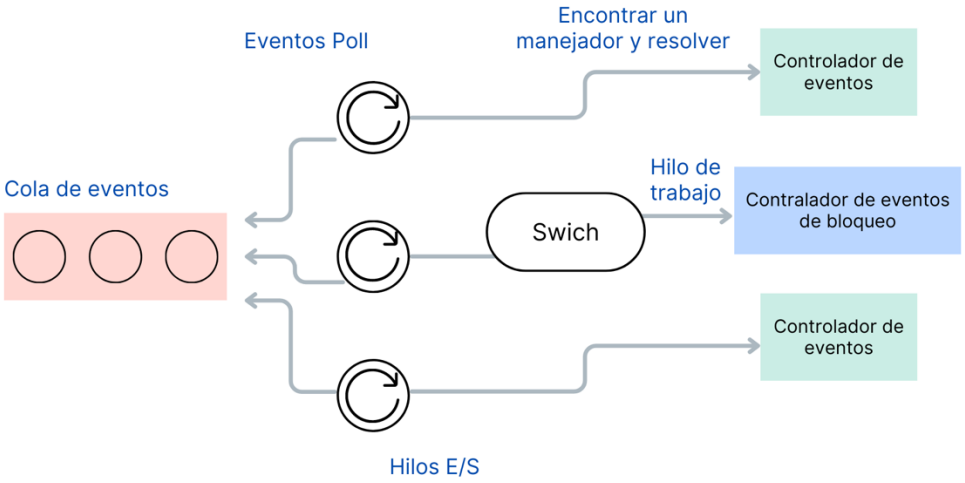

Figura 2-4 Esquema de la lógica de decisión de una aplicación en Quarkus

Las extensiones de las que dispone Quarkus mediante las que el desarrollador puede añadir sugerencias a lo largo del código son las que tienen todo el poder en este aspecto. Un ejemplo de esto son las anotaciones `@Blocking` y `@NonBlocking`, es decir, son las que decidirán si la aplicación avanza hacia un modelo reactivo o hacia el tradicional.

Esto se resume en que la petición necesitará un hilo de E/S y una vez que llegue a la zona de bloqueo será redirigida a otro hilo. Pero a qué es lo que decidirá la extensión, que tendrá varias opciones. Puede llamarlo en el hilo de trabajo de E/S actual, derivarlo a otro subproceso de trabajo distinto o no prestar atención a las modificaciones de subprocesos. Pero esto ya dependerá exclusivamente de la extensión.

Juntando todo lo anterior se encuentra un modelo de programación bastante versátil. Según el cual el desarrollador tiene toda la capacidad de decidir absolutamente todo en cuanto a la implementación y la ejecución del programa y sus respectivas solicitudes. El desarrollador dispone de todas las opciones posibles, la reactiva, la tradicional o, incluso, un conjunto de ambas que puede guiar mediante extensiones añadidas al código.

2.4.4 Extensiones de Quarkus que permiten Reactive.

Quarkus pone a disposición del desarrollador un gran conjunto de características y APIs reactivas, aunque no están todas enumeradas. Además, Quarkus va añadiendo diferentes características novedosas en cada versión.

2.4.4.1 HTTP.

RESTEasy Reactive. Un tipo de ejecución de JAX-RS amoldada a la arquitectura usada por Quarkus. Esta extensión sigue una orientación reactiva, pero eso no impide que admita también código pensado de manera secuencial gracias a la anotación `@Blocking`.

Reactive Routes. Una manera asertiva de registrar rutas de carácter HTTP de primera mano en el enrutador que Vert.x proporciona empleado para enrutar solicitudes del tipo HTTP a los métodos por Quarkus.

Reactive Rest Client. Gracias a esta extensión tienes la capacidad de consumir puntos finales de carácter HTTP. Por debajo de lo observable a simple vista, emplea las funciones de E/S no bloqueantes que ofrece la arquitectura Quarkus.

Qute. El propulsor de plantillas de esta extensión propone una API de carácter reactivo con la finalidad de convertir plantillas mediante un estilo de programación no bloqueante.

2.4.4.2 Datos.

Hibernate Reactive. Una versión de Hibernate ORM que emplea clientes no bloqueantes y asíncronos para comunicarse con una base de datos externa.

Hibernate Reactive con Panache. Además de lo añadido por Hibernate Reactive, facilita repositorio y soporte de registro audaz.

Reactive PostgreSQL Client. Un cliente no bloqueante y asíncrono que se comunica con diferentes bases de datos externas que son PostgreSQL. Esto consigue que se pueda soportar una alta concurrencia.

Reactive MySQL Client. Un cliente no bloqueante y asíncrono que se comunica con diferentes bases de datos externas que son MySQL.

MongoDB extension. Con la finalidad de comunicarse con MongoDB propone una API de carácter reactivo, denominada Mutiny, e imperativa.

Mongo con Panache. De un modo similar a Hibernate Reactive con Panache, facilita soporte de registro audaz para las APIs reactivas y, también, para las APIs imperativas.

Cassandra extension. Propone una API, denominada Mutiny, de carácter reactivo e imperativa para comunicarse con Cassandra.

Redis extension. Propone una API, denominada Mutiny, de carácter reactivo e imperativa para consultar y guardar datos de un almacén de Redis el cual funciona mediante clave-valor.

2.4.4.3 Arquitectura basada en eventos.

Reactive Messaging. Da la posibilidad de, partiendo de código de carácter reactivo e imperativo, desarrollar aplicaciones basadas en eventos.

Kafka Connector for Reactive Messaging. Da la posibilidad de desarrollar sistemas que reciben y guardan registros Kafka.

AMQP 1.0 Connector for Reactive Message. Da la posibilidad de desarrollar aplicaciones que envían y escuchan mensajes AMQP.

2.4.4.4 Protocolos de red y unidades.

gPRC. Desarrollar y utilizar servicios del tipo gPRC, junto con brindar interfaces de desarrollo imperativas y de carácter reactivo.

GraphQL. Facilita como trasmisiones de eventos suscripciones y APIs de Mutiny. Además, se puede consumir y desarrollar el datasource, como un cliente por medio de GraphQL.

Tolerancia a los fallos. También es posible usarla con tipos de Mutiny. Ofrece capacidades de reserva, de reintento e interruptores al sistema.

2.4.4.5 Motor.

Vert.x es el propulsor de carácter reactivo que se utiliza de base en la arquitectura de Quarkus. La extensión da la posibilidad de acceder a la instancia administrativa de Vert.x, pero no solo a esta, sino también a su variante Mutiny, mostrando la API de Vert.x mediante la utilización de tipos de Mutiny.

Context Propagation. Mediante el uso de una canalización de carácter reactivo, recoger y difundir objetos contextuales.

2.5 Mutiny, desarrollo asíncrono de manera sencilla.

Se trata de la colección de programación reactiva más sencilla de utilizar y más popular, habiéndose convertido en la principal opción a la hora crear aplicaciones siguiendo el modelo reactivo.

2.5.1 Una API de programación reactiva impulsada por eventos.

Al implementar una aplicación con Mutiny es necesario diseñarla enfocada a los eventos y como se reacciona ante ellos. Todos los sistemas reactivos se basan en eventos y es lo que facilita que las continuaciones necesarias para los subprocesos que se encuentran realizando operaciones de E/S sean mucho más concretas

Estos eventos se pueden gestionar mediante el uso de dos tipos que, precisamente, son impulsados por eventos. Mediante estos se puede representar cualquier tipo de interacción y, para ello, simplemente será necesario suscribirse a ellos. Una vez que estamos suscritos solo habrá que esperar a que emitan uno o varios elementos o un fallo. Y cuando recibes dicho evento se podrá transformar siguiendo la semántica de onX().action(), leyéndose como en el elemento X realiza la acción action.

2.5.1.1 Uni.

Es un tipo de flujo pensado para representar acciones de tipo asíncrono mediante la emisión de un solo evento. Cabe la posibilidad de que sea un elemento si todo salió correctamente, o un fallo si se produjo algún error. En definitiva, devuelven como resultado 0 o 1.

Ejemplos claros de representaciones que realiza un objeto del tipo Uni a la perfección son solicitudes HTTP, una llamada a un procedimiento externo, una operación que produce un único resultado o el envío de mensaje a una cola de intermediario de mensajes.

Pero no queda ahí, aunque solo pueda emitir un evento, este puede ser un elemento que sea igual a null. Para esto la API del tipo Uni tiene unos métodos concretos, ya que `Uni<T>` ofrece una gran cantidad de operadores que producen, modifican y gestionan secuencias del tipo Uni.

Además, cuando trabajamos con un Uni<Void> emitirá o bien null, en el lugar de emitir un elemento, o el fallo, en el caso de que no se haya podido llevar a cabo la operación. En este caso, dicho valor null tendrá el significado de haber finalizado correctamente la operación.

Una vez que el evento ha sido emitido, se puede procesar o, incluso, transformar mediante operadores que intervienen en las canalizaciones creadas para dicho evento. Independientemente de que sea el elemento o el fallo. Pero esto no ocurre solo, sino que es necesario que haya un suscriptor de dicho elemento Uni, ya que hasta que esto no ocurra no comenzará a operar.

A continuación, se muestra un fragmento de código que ofrece un ejemplo sencillo de canalización usando el tipo Uni:

```
Uni.createFrom().item(1)
        .onItem().transform(item -> "hola-" + item)
        .onItem().delayIt().by(Duration.ofMillis(200))
        .subscribe().with(System.out::println);
```

2.5.1.1.1 Suscribirse a un Uni para que haya transmisión de eventos.

Como se ha dicho anteriormente es necesario suscribirse. Esto se debe a que, además de que es necesario para que comience a operar, las canalizaciones de eventos se llevan a cabo para cada suscripción.

Una vez hecho esto está la posibilidad de incluir hasta dos devoluciones de llamada que serán invocadas una vez que el elemento ha sido emitido. La primera servirá para recibir el elemento y la segunda recibirá el fallo, ambas en caso de producirse.

Además, al ser un objeto del tipo cancellable ofrece la posibilidad de cancelar la transmisión en caso de que se considere necesario.

```
Cancellable cancellable = uni
        .subscribe().with(
                item -> System.out.println(item),
                failure -> System.out.println("Ha fallado con " + failure));
```

2.5.1.1.2 Crear Unis a partir de elementos conocidos.

A la hora de crear instancias del tipo Uni hay una gran variedad de posibilidades, que se pueden observar con `Uni.createFrom()`.

Un ejemplo de cómo crear una instancia de tipo Uni a partir de un valor que se conoce sería el siguiente. En él, cada suscriptor del tipo Uni recibirá el artículo 8 en el momento preciso que se suscriba.

```
Uni<Integer> valorConocido = Uni.createFrom().item(8);
```

También existe la posibilidad de pasar un Supplier, que será invocado para cada suscriptor, reflejándose en que cada uno recibirá un valor diferente al resto.

```
AtomicInteger contador = new AtomicInteger();
Uni<Integer> uni = Uni.createFrom().item(() -> counter.getAndIncrement());
```

2.5.1.1.3 Crear Unis que simulen fallos.

Hay que recordar que el tipo Uni también puede emitir un evento de fallo. Para indicar que la operación no se ha podido llevar a cabo se emitirá dicho evento de fallo.

Se pueden crear instancias de Uni que simulan que un error ocurrió mediante el uso de una excepción que describa el fallo. Pero también se puede pasar un Supplier, que generará una excepción distinta para cada suscriptor.

```
// Pasar directamente una excepción:
Uni<Integer> fallo1 = Uni.createFrom().failure(new Exception("explotó"));

// Pasar una llamada Supplier para cada suscriptor:
Uni<Integer> fallo2 = Uni.createFrom().failure(() -> new Exception("explotó"));
```

2.5.1.1.4 Crear instancias del tipo Uni<Void>.

Hay operaciones que no devuelven ningún valor. En este tipo de operaciones se devolverá un elemento que sea igual a null para indicar que se ha terminado la operación con éxito. En el caso de que fallara emitiría un evento de fallo igual que en otros casos.

Esto se hace con la finalidad de mantener informado al receptor.

```
Uni<Void> uniSinResultado = Uni.createFrom().nullItem();
```

2.5.1.1.5 Crear Unis a partir de un emisor de eventos.

También existe la posibilidad de crear objetos del tipo Uni partiendo de un emisor de eventos. Esto puede ser útil es situaciones en las que pretendamos incorporar APIs que se caractericen porque necesitan recibir una devolución de llamada de parte de dicho emisor. Es decir, esperarán a recibir un evento para emitir el suyo.

Pero este tipo tampoco garantiza que no haya un error. Por lo que también tiene la posibilidad de emitir un fallo y de que la transmisión sea cancelada.

```
Uni<String> uniEmisor = Uni.createFrom().emitter(emisor -> {
    // Cuando el resultado esté disponible, emitirlo
    emisor.complete(result);
});
```

2.5.1.1.6 Crear Unis desde un CompletionStage.

También se pueden crear objetos Unis desde CompletionStage/ CompletableFuture, lo que es bastante práctico cuando se acopla con APIs que se basan en estos tipos.

```
Uni<String> uniStage = Uni.createFrom().completionStage(stage);
```

2.5.1.2 Multi.

Es un tipo de flujo pensado para representar acciones de tipo asíncrono mediante la emisión de diversos eventos. Estos pueden ser n elementos, un fallo o una finalización. Estos elementos están pensados para emitir flujos de datos de una longitud indeterminada, pudiendo ser ilimitados en determinados momentos. Del mismo modo que con el tipo Uni, un fallo significa la finalización de la transmisión y, por tanto, no se recibirá ningún otro elemento después de este.

A medida que los eventos son emitidos, se pueden procesar o, incluso, transformar mediante operadores que intervienen en las canalizaciones creadas para dicho evento. Independientemente de que sean los elementos o el fallo. Pero esto no ocurre solo, sino que es necesario que haya un suscriptor de dicho elemento Multi, ya que hasta que esto no ocurra no comenzará a operar.

Mediante el siguiente fragmento de código se muestra una manera sencilla de canalización usando este tipo.

```
Multi.createFrom().items(6, 7, 8, 9, 10)
        .onItem().transform(item -> item * 2)
        .select().first(4)
        .onFailure().recoverWithItem(0)
        .subscribe().with(System.out::println);
```

En este ejemplo se crea un objeto del tipo Multi con los elementos conocidos {6, 7, 8, 9, 10}. Para cada elemento se produce una transformación que consiste en multiplicarlo por 2. Una vez hecho se seleccionarán los cuatro primeros y se imprimirán en consola.

Sin embargo, en caso de fallo se recuperará con un 0.

2.5.1.2.1 Suscripción a un Multi.

Como se ha dicho anteriormente es necesario suscribirse. Esto imprescindible para que el objeto Multi comience a operar y porque las canalizaciones se llevan a cabo para cada suscripción.

Una vez hecho esto está la posibilidad de incluir hasta tres devoluciones de llamada que serán invocadas a medida que los elementos son emitidos. La primera servirá para recibir el elemento, la segunda recibirá el fallo y, por último, la tercera las respectivas finalizaciones. Estas

finalizaciones de trasmisiones se producirán porque ya no quedan elementos que emitir o porque una vez que se produce un fallo no se emiten más elementos.

Además, al ser un objeto del tipo cancellable, como su propio nombre indica, ofrece la posibilidad de cancelar la transmisión en caso de que se considere necesario.

```
Cancellable multiCancellable = multi
        .subscribe().with(
                i -> System.out.println(i),
                fallo -> System.out.println("Falló con " + fallo),
                () -> System.out.println("Completado"));
```

2.5.1.2.2 Crear Multis a partir de elementos conocidos.

A la hora de crear instancias del tipo Multi hay una gran variedad de posibilidades, que se pueden observar con `Multi.createFrom()`.

Un ejemplo de la creación de objetos del tipo Multi partiendo de elementos que conocemos o de un Iterable es el siguiente. Según este cada suscriptor del tipo Multi recibirá un conjunto de artículos similar (6, 7, …, 9) justo en el momento preciso que se suscriba.

```
Multi<Integer> multiConocido = Multi.createFrom().items(5, 6, 7, 8);
Multi<Integer> multiIterable = Multi.createFrom().iterable(Arrays.asList(5, 6,
7, 8, 9));
```

También existe la posibilidad de pasar un Supplier, que será invocado para cada suscriptor. Reflejándose en que cada uno recibirá un conjunto de valores diferente al resto.

```
AtomicInteger contador = new AtomicInteger();
Multi<Integer> multiContador = Multi.createFrom().items(() ->
        IntStream.range(contador.getAndIncrement(),contador.get()*2).boxed());
```

2.5.1.2.3 Creación de Multis que simulen fallos.

No todo funciona siempre de manera correcta. En ocasiones, es posible que en las trasmisiones se produzcan fallos. Estos se emplean para avisar a los suscriptores finales que en la transmisión se encontró un error importante y, por lo tanto, tiene que parar de inmediato la emisión de elementos. Es por todo esto que se necesita poder crear instancias fallidas del tipo Multi con fragmentos de código como el siguiente.

Es posible pasarle directamente una excepción que describa el fallo. Pero también está la posibilidad de crear una excepción para cada suscriptor mediante un Supplier.

```
// Pasar directamente una excepción:
Multi<Integer> fallo1 = Multi.createFrom().failure(new Exception("explotó"));

// Pasar una llamada Supplier para cada suscriptor:
Multi<Integer>     fallo2     =     Multi.createFrom().failure(()     ->     new
Exception("explotó"));
```

2.5.1.2.4 Crear Multis vacíos.

Las trasmisiones del tipo Multi tienen terminantemente prohibido la transmisión de elementos null ya que son trasmisiones reactivas y en estas está prohibido. Por tanto, no envían dichos elementos. Por el contrario, lo que sucede es que envían eventos de finalización que informan de que no hay más elementos disponibles para emitir.

También es posible que desde un primer momento no haya elementos para emitir. Esto supondrá que desde un principio se emita el evento de finalización, creando, por tanto, una secuencia vacía.

Usando lo siguiente es posible crear una trasmisión de este tipo.

```
Multi<String> multiVacio = Multi.createFrom().empty();
```

2.5.1.2.5 Crear Multis a partir de un emisor de eventos.

También existe la posibilidad de crear objetos del tipo Multi partiendo de un emisor de eventos. Esto puede ser útil es situaciones en las que se pretenda incorporar APIs que se caractericen por necesitar recibir una devolución de llamada de parte de dicho emisor. Es decir, esperarán a recibir un evento para empezar a emitir los suyos.

Pero esto tampoco garantiza que no haya un error. Por lo que también tiene la posibilidad de emitir un fallo y de que la transmisión sea cancelada.

```
Multi<Integer> multiEmitter = Multi.createFrom().emitter(emitter -> {
    emitter.emit(5);
    emitter.emit(6);
    emitter.emit(7);
    emitter.complete();
});
```

2.5.1.2.6 Creación de Multis a partir de ticks.

Crear una trasmisión que emita ticks de forma periódica es posible tal y como se muestra en el siguiente ejemplo. En este, el suscriptor recibe un contador que es un objeto de tipo long. El valor en el primer tick es 0, posteriormente 1, 2 y así sucesivamente.

```
Multi<Long> multi = Multi.createFrom().ticks().every(Duration.ofMillis(80));
```

2.5.1.2.7 Creación de Multis desde un generador.

Partiendo de un estado inicial y con una función generadora también se puede crear una secuencia. Por medio de un proveedor, en este ejemplo () -> 1, se da el estado inicial. La función generadora necesita dos argumentos. El primero es el estado actual. y el segundo será un emisor con la capacidad de emitir un evento de fallo, un nuevo elemento o una finalización.

La función del generador devolverá el siguiente estado actual como valor de retorno. En el caso del ejemplo, al ejecutarlo, el resultado será un conjunto de números como el siguiente: {2, 4, 7, 11, 17, 26, 40, 61}.

```
Multi<Object> multiSequence = Multi.createFrom().generator(() -> 1, (numero,
emitter) -> {
    int siguiente = numero + (numero / 2) + 1;
    if (numero < 50) {
        emitter.emit(siguiente);

    } else {
        emitter.complete();
    }
    return siguiente;
});
```

2.5.2 Observar los eventos emitidos por los tipos Uni y Multi.

Tanto el tipo Uni como el Multi basan su funcionamiento en la emisión de eventos. Es por esto por lo que es imprescindible que el código de la aplicación los observe y reaccione ante ellos para poder procesarlos.

En la mayor parte de los casos, solo interesarán los elementos y eventos de fallos que emitan. Pero en algunas ocasiones hará falta prestar atención a otros tipos para entender que es lo que está sucediendo en la aplicación o reaccionar ante ciertos elementos. Algún ejemplo de esto podría ser la necesidad de registrar un mensaje de cancelación o de fallo, o, incluso, el cierre de un recurso que estemos utilizando una vez que ha recibido un evento de cancelación.

Dentro de los tipos de eventos que se pueden emitir cada uno tiene un método asociado a él. `onItem()` para los elementos, `onFailure()` para los fallos y `onCompletion()` para las finalizaciones de transmisión. Pero en todos se pueden encontrar y utilizar los métodos `invoke()` y `call()`. Estos se encargarán de avisar de que algo ha pasado para tener la oportunidad de reaccionar a ello, pero en ningún caso realizará una modificación del evento detectado. Es más, una vez que se reaccione ante el evento en cuestión se continuará propagando del mismo modo que antes de que uno de estos métodos reaccionara ante él.

2.5.2.1 El método invoke().

Mediante este método, que no devuelve nada y es síncrono, se observa la emisión de eventos. A medida que estos son detectados, se reacciona ante ellos con la devolución de llamada que se haya configurado. Pero se recalca que solo se reacciona ante él, una vez que esto ha sucedido se deja que el evento avance.

```
Uni<String> uni = uni.onItem()
    .invoke(item -> System.out.println("Item recibido: " + item));
Multi<String> multi = multi.onItem()
    .invoke(item -> System.out.println("Item recibido: " + item));
```

Al ser un método síncrono produce un bloqueo. Este se producirá en el momento en que se está reaccionando al evento, antes de ser enviado hacia el consumidor del evento.

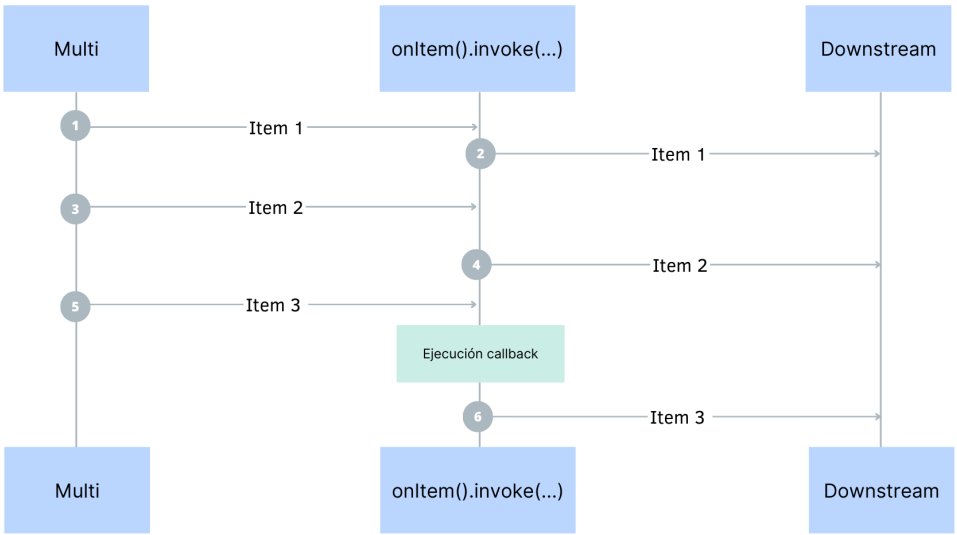

Figura 2-5 Esquema de la lógica del método invoke()

Pero en programación reactiva lo principal es el desarrollo no bloqueante, por lo que bloquear nunca es una buena opción.

Para configurar diferentes reacciones según el tipo de evento se puede hacer de varias maneras.

```
multi
    .onSubscription()
        .invoke(() -> System.out.println("(Down) Suscrito"))
    .onItem()
        .invoke(item -> System.out.println("(Down) Ttem recivido: " + item))
    .onFailure()
        .invoke(fallo -> System.out.println("(Down) Falló con " + fallo))
    .onCompletion()
        .invoke(() -> System.out.println("(Down)Completado"))
```

```
    .onCancellation()
        .invoke(() -> System.out.println("(Up) Cancelado"))
    .onRequest()
        .invoke(sol -> System.out.println("(Up) Solicitado: " + sol));
```

Los paréntesis añadidos al fragmento de código indican si el suceso proviene de upstream o downstream, es decir, de la ejecución o del consumidor.

Solo hay una excepción en la que este método modifica el evento. Es cuando la devolución de llamada devuelve una excepción. En este caso, el flujo que llega desde la ejecución no recoge el valor real, sino que por el contrario recoge un evento de fallo como devolución.

Al observar este suceso de fallo, Mutiny divulga una CompositeException añadiendo tanto el fallo que proviene de la devolución de llamada como el fallo original.

2.5.2.2 El método call().

Este método es muy utilizado a la hora de crear reacciones asíncronas a la recepción del evento, aunque no son prioritarias. Un ejemplo de esto puede ser cerrar recursos. Se puede realizar ya que devuelve un objeto del tipo Uni y es completamente asíncrono, a diferencia del anterior.

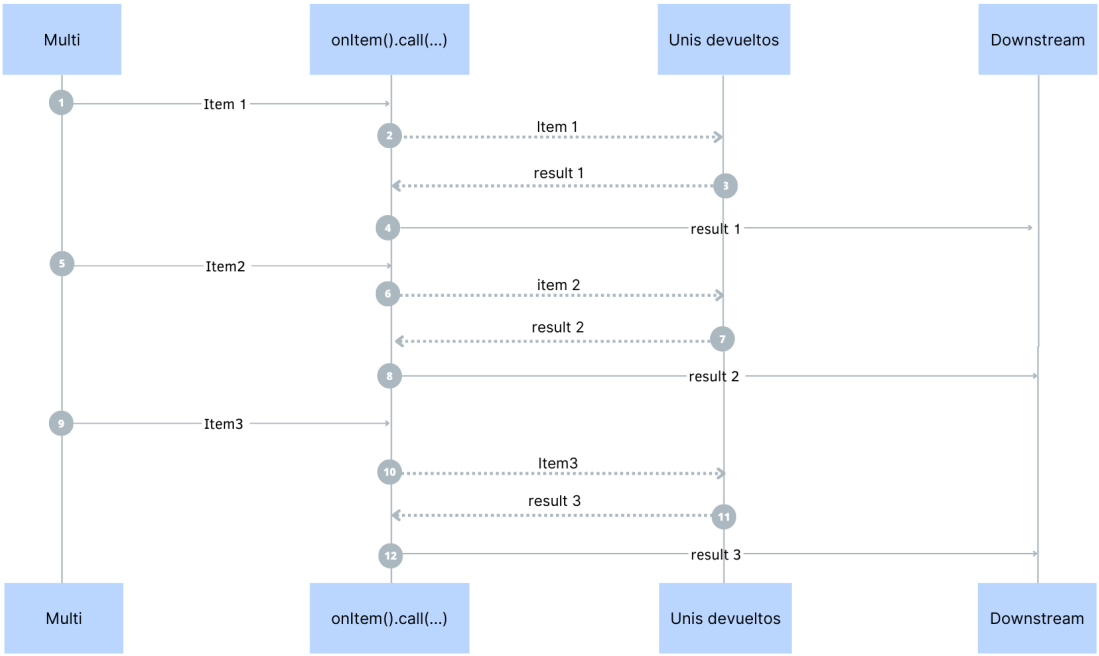

Figura 2-6 Esquema de la lógica del método call()

Una vez que el objeto de tipo Uni configurado en la devolución de llamada emite un evento, se envía el otro evento al que se reaccionó hacia el suscriptor, pero en ningún caso en un momento anterior a la recepción de este.

```
multi
    .onItem().call(item ->
        Uni.createFrom().voidItem()
            .onItem().delayIt().by(Duration.ofSeconds(2))
    )
);
```

En el ejemplo anterior la única finalidad de uso de este método es la de retrasar el envío del evento y esto es totalmente lícito. Pero normalmente es usado para finalizar otras acciones

asíncronas que ya se habían empezado, como puede ser la de invocar al método asíncrono `close()` en un recurso.

```
multi.onCompletion().call(() -> resource.close());
```

En realidad, lo que está sucediendo es que no se tiene en cuenta el objeto Uni como tal. Solamente se utiliza el evento que emite y para esto se suscribe a él, se observa la devolución y después se descarta.

En el caso de que la devolución de llamada configurada emita una excepción o el objeto Uni emita un fallo, simplemente se enviará el fallo o una CompositeException hacia el consumidor. Se sustituye el evento producido en un primer momento por dicha excepción.

2.5.3 Transformar elementos.

Las operaciones más comunes con relación a los tipos Uni y Multi son las de transformación de elementos ya que ambos tipos basan su funcionamiento en la emisión de estos. Estas transformaciones se realizarán mediante la función `onItem().transform(Function<T, U>)`. Esta función es síncrona, por lo que producirá bloqueo, y se hará del modo 1 a 1. Es decir, el elemento lo transformará en lo que hayamos configurado, pero siempre uno por uno.

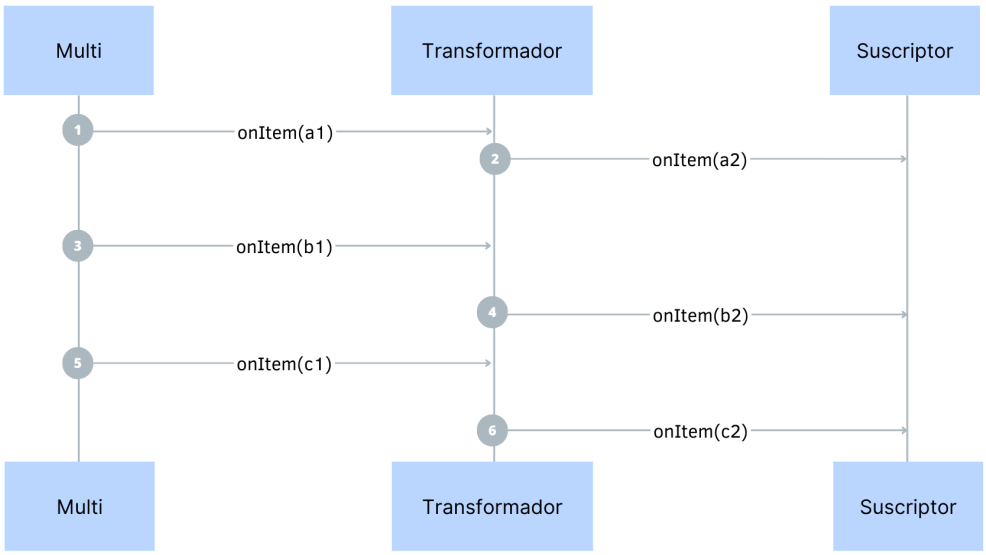

Figura 2-7 Esquema de la lógica de los métodos onItem().transform()

2.5.3.1 Transformar elementos que han sido producidos por objetos del tipo Uni.

Suponiendo que se dispone de un objeto `Uni<String>` y que lo que se quiere es transformar a minúsculas dicha cadena, se realizaría dicha modificación de la siguiente forma.

```
Uni<String> uni = Uni.createFrom().item("HOLA MUNDO");
uni
    .onItem().transform(item -> item.toLowerCase())
    .subscribe().with(item2 -> System.out.println(item2)); // Print hola mundo
```

2.5.3.2 Transformar elementos que han sido producidos por objetos del tipo Multi.

Se realiza de la misma manera que el anterior, salvo por una diferencia. En Multi la función se invoca para cada elemento y posteriormente son emitidos de manera descendente hacia el suscriptor.

```
Multi<String> multi = Multi.createFrom().items("i", "c", "v");
multi
     .onItem().transform(item -> item.toUpperCase())
     .subscribe().with(
                Item2 -> System.out.println(item2)); // Print I C V
```

2.5.3.3 *Si se produce un fallo en la transformación se finaliza.*

En caso de que la trasformación falle se tratará de manera muy similar a como se hace con ambos tipos. Se genera una excepción que es enviada al suscriptor como un evento de fallo, lo que repercutirá en que no recibirá más elementos.

2.5.3.4 *Encadenar varias transformaciones.*

También existe la posibilidad de concatenar varias transformaciones.

```
Uni<String> uni = uni
        .onItem().transform(item -> item.toUpperCase())
        .onItem().transform(item -> item + "!");
```

2.5.4 Transformar elementos de forma asíncrona.

Hay ocasiones en las que no es suficiente con trasformar un elemento de un flujo en otro y se necesita transformarlo en un objeto del tipo Uni o en un objeto del tipo Multi. Esto se hará mediante las funciones `onItem().transformToUni(Function<T, Uni<0>>)` y `onItem().transformToMulti(Function<T, Multi<0>>)`, que al permitir realizar estas transformaciones abren un mundo de oportunidades.

2.5.4.1 *Uni – Transformar un elemento en un objeto del tipo Uni.*

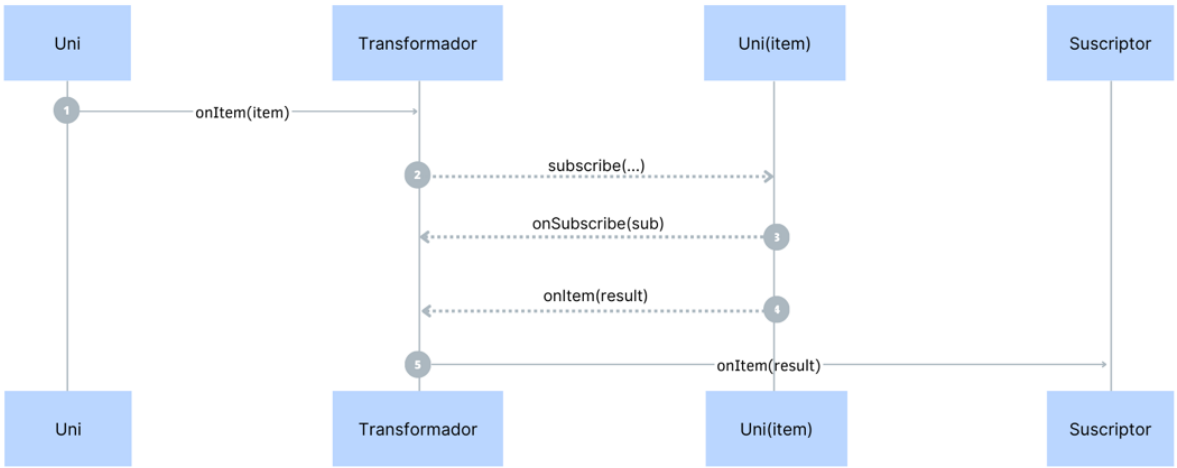

Figura 2-8 Esquema de la lógica de transformar un elemento en un Uni

Suponiendo que se dispone de un `Uni<String>` y se desea invocar a un servicio externo que conlleva una acción asíncrona representada por un Uni. Se necesitaría transformar el elemento que se recoge del primer Uni en otro Uni que haya sido enviado por el servicio. Además, al encadenar dos unis se dispone de la opción de emitir el elemento si la operación se realizó con éxito o el fallo si no se pudo completar.

```
Uni<String> uniString = Uni.createFrom().item("Ivan");
uniString
    .onItem().transformToUni(nombre -> invokeRemoteGreetingService(nombre))
    .subscribe().with(
             item -> System.out.println(item), // Print "Hola Ivan",
             fallo -> fallo.printStackTrace()); // Print el seguimiento de la
pila de fallos
```

2.5.4.2 Uni - Transformar un elemento en un objeto del tipo Multi.

También está la posibilidad de querer transformar el elemento en un flujo, o llamado de otra manera, un objeto Multi.

En el siguiente fragmento se crea un flujo del tipo Multi mediante la duplicación del elemento que se disponía.

Aunque en este ejemplo se realiza de forma sencilla, en casos reales lo más seguro es que el objeto Multi sea más complejo e, incluso, tenga la capacidad de emitir elementos de manera asíncrona.

```
Multi<String> resultado = uni
    .onItem().transformToMulti(item -> Multi.createFrom().items(item, item))
    .subscribe().with(
             item -> System.out.println(item)); // Llamado dos veces
```

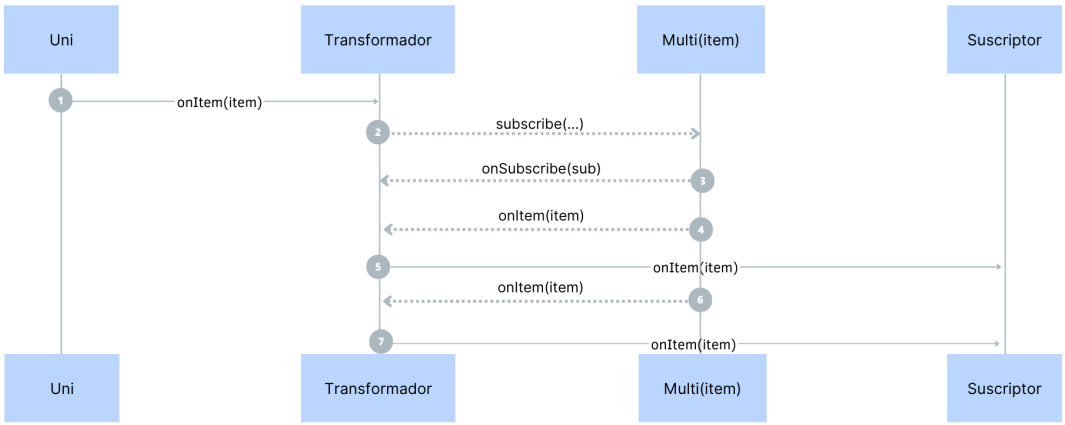

Figura 2-9 Esquema de la lógica de transformación de un elemento en un Multi

2.5.4.3 Transformar elementos Multi: fusionar vs concatenar.

Al convertir los elementos que han sido enviados por medio de un flujo ascendente del tipo Multi, es necesario que se agrupen los artículos creados.

Es aquí donde aparece el dilema de la preferencia. Es decir, se prefiere rapidez u orden. Esto depende de la elección que se haga en cuanto a fusionar o concatenar. La primera opción simplemente emite elementos conforme los va recibiendo, independientemente de que sea el orden adecuado o no. Mientras que la segunda guarda los elementos y los concatena manteniendo el orden inicial.

Suponiendo que Multi envía los artículos “Ivan” y “Castillo”, en ese orden, y se llama a `invokeRemoteGreetingService` desde la parte superior. Para hacer esto se utilizaría `invokeRemoteGreetingService("Ivan")` y del mismo modo con “Castillo”. Una vez hecho esto, a la hora de recibir los elementos está la posibilidad de que se haga en distinto orden y es por esto que se debe especificar cual será la preferencia que se debe seguir.

2.5.4.4 Mulit – Transformar un elemento en un objeto del tipo Uni.

Dependiendo de la preferencia se utilizará una opción u otra.

```
Multi<String> fusionado = multi
    .onItem().transformToUniAndMerge(nombre                                    ->
invokeRemoteGreetingService(nombre));

Multi<String> concatenado = multi
    .onItem().transformToUniAndConcatenate(nombre                              ->
invokeRemoteGreetingService(nombre));
```

2.5.4.5 Multi – Transformar un elemento a un objeto del tipo Multi.

Del mismo modo que con los Uni, dependiendo de la preferencia se convertirán en Multi siguiendo un patrón u otro.

```
Multi<String> fusionado = multi
    .onItem().transformToMultiAndMerge(item -> someMulti(item));

Multi<String> concatenado = multi
    .onItem().transformToMultiAndConcatenate(item -> someMulti(item));
```

2.5.5 Manejar los fallos.

Cuando algo falla en la transmisión que se está emitiendo se envía un evento de fallo. Esto no es más que un aviso de que algo no salió como se esperaba y que, por tanto, no se pueden emitir más eventos.

Al recibirlo se puede continuar propagando el fallo hasta que en algún componente se decida a gestionarlo o este llegue al suscriptor final. Pero en el momento de gestionarlo tendremos la posibilidad de transformarlo en otro tipo de fallo del que podamos extraer más información, intentar recuperarse o simplemente volver a intentar la misma operación.

2.5.5.1 Observar fallos.

En el momento en el que se produce un fallo hay ocasiones en las que es interesante ejecutar alguna acción que haya sido personalizada previamente, como puede ser informar del fallo por consola al usuario.

```
Uni<String> uni = uni
        .onFailure().invoke(fallo -> log(fallo));

Multi<String> multi = multi
        .onFailure().invoke(fallo -> log(fallo));
```

2.5.5.2 Transformar fallos.

Otra opción respecto a los fallos es la de convertirlo en otro que brinde más información. Normalmente cuando se quiere transformar los fallos en otros es con la intención de obtener más información de este y esto es posible al envolver un fallo de bajo nivel, como lo es una IOException en otro fallo, es decir, una ServiceUnabiableException.

```
Uni<String> uni = uni
        .onFailure().transform(fallo ->
                new ServiceUnavailableException(fallo));
```

2.5.5.3 Recuperarse mediante el empleo de elementos alternativos al recibido.

En el caso de que la opción elegida sea intentar recuperarse, hay varias maneras de intentarlo. La primera sería la sustitución del fallo por otro elemento distinto. Esto se puede hacer de varias formas. Reemplazarlo directamente o incluir un Supplier a través de cual calcular el elemento. De ambas formas los consumidores de este flujo no recibirán un evento de fallo, sino que recibirán un elemento distinto.

```
Uni<String> uni1 = uni
        .onFailure().recoverWithItem("elemento sustituto");

Uni<String> uni2 = uni
        .onFailure().recoverWithItem(fallo -> getFallback(fallo));
```

Pero es importante recordar que una vez que se produce un fallo no se puede continuar emitiendo más artículos. Por lo que recogerán el artículo de respaldo y, por último, el aviso de finalización.

2.5.5.4 Sustituir el fallo por una finalización.

Otra opción en el intento de recuperar es el de completar cuando se trabaja con un objeto del tipo Multi. En este se puede sustituir el fallo por un aviso de finalización, así los consumidores que se encuentren por debajo no recibirán el fallo, sino dicho evento.

```
Multi<String> multi = multi
        .onFailure().recoverWithCompletion();
```

2.5.5.5 Cambiar a otro subproceso.

La última opción de recuperación es el cambio de hilo de ejecución. Hay situaciones en las que es interesante, en caso de fallo, variar a otro flujo distinto. Una vez que se recoge el fallo, es posible suscribirle a este otra transmisión diferente, y, mediante esto, emitir los artículos de dicha transmisión. Pero es esencial que la transmisión original ascendente contenga los mismos artículos que los que han creado las transmisiones alternativas.

```
Uni<String> uni = uni
        .onFailure().recoverWithUni(fallo -> getFallbackUni(fallo));

Multi<String> multi = multi
        .onFailure().recoverWithMulti(fallo -> getFallbackMulti(fallo));
```

2.5.6 Reintentar cuando se produce un fallo.

Una vez que se ha producido un fallo, es normal volver a intentar procesar de nuevo el trabajo que se estaba realizando, y esto es posible a través de varios métodos.

2.5.6.1 Reintentar en varias ocasiones.

Mediante `onFailure().retry()` es posible reintentar retomar el proceso. El parámetro que se está pasando significa el número de veces que se quiere volver a intentarlo.

También existe el método `onFailure().retry().indefinitely()`. Pero este, probablemente nunca terminará, por lo que en un principio no debería utilizarse o, en caso de usarse, hacerlo siendo sumamente cuidadoso.

```
Uni<String> uni = uni
        .onFailure().retry().atMost(2);
Multi<String> multi = multi
        .onFailure().retry().atMost(2);
```

2.5.6.2 Añadir retrasos.

El método `retry()` reintentará de manera inmediata a no ser que se añada alguna modificación. Pero al hacer uso de servicios externos, hay veces en las que es conveniente esperar a la hora de volver a intentar.

La espera es personalizable gracias a que se puede añadir un tiempo de espera máxima y mínima. Además de tener la posibilidad de personalizar un jitter para conseguir que la espera tenga un cierto carácter aleatorio.

Si no se quiere personalizar un número determinado de reintentos, es decir no se quiere utilizar el método atMost, si no que se desea establecer una fecha límite mientras se utiliza la espera exponencial, se debe utilizar el método expireIn o expireAt.

```
Uni<String> uni = uni
        .onFailure().retry()
        .withBackOff(Duration.ofMillis(200), Duration.ofSeconds(2))
        .atMost(2);
```

2.5.6.3 Reintentar cuando se cumpla una condición.

Se puede emplear until como opción a atMost, ya que admite un predicado después de cada fallo. En caso de que este retorne el valor false, no reintentará y divulgará el último fallo hacia los consumidores. Pero si retorna el valor contrario volverá a intentarlo.

```
Uni<String> uni = uni
        .onFailure().retry()
        .until(fallo -> shouldWeRetry(fallo));
```

2.5.7 Características de Mutiny.

Quarkus creó Mutiny aun existiendo otras APIs de programación reactiva, ya que esta aborda un punto totalmente diferente debido a sus características.

2.5.7.1 Impulsado por eventos.

Mutiny trata los eventos como el ámbito principal de su diseño. Se trata de observarlos, reaccionar ante ellos y construir canalizaciones en las que procesar los datos de manera entendible y sin una cantidad de código extremadamente larga.

2.5.7.2 Navegable.

Mutiny ofrece una API clara y navegable en la que es relativamente sencillo llegar al operador que estamos buscando, esto soluciona que, pese a la finalización inteligente del código, las clases con demasiados métodos sean dudosas.

2.5.7.3 E/S sin bloqueo.

Mutiny es perfecto para manejar la característica asíncrona de los sistemas con E/S no bloqueante que promueve Quarkus ya que se puede modificar datos, crear operaciones de forma declarativa, forzar que se cumpla el proceso establecido y, además, tiene la capacidad de recuperase de fallos.

2.5.7.4 Perfecto para un sistema asíncrono.

Mutiny se puede utilizar en cualquier aplicación asíncrona, desde las mismas aplicaciones reactivas hasta aplicaciones basadas en mensajes, microservicios basados en eventos, procesamiento de flujos de datos o utilidades de red.

2.5.7.5 Reactive Streams y Converter Built-In.

Al basarse en la especificación Reactive Streams, Mutiny puede integrarse a la perfección con la biblioteca de programación reactiva que desee. Además, ofrece una serie de convertidores

para comunicarse con otras bibliotecas que usa una gran parte de los desarrolladores de este entorno.

2.5.8 Integración de Mutiny en Quarkus.

La adaptación de Mutiny en Quarkus no se queda solo en una librería, sino que propone uniones que consiguen que se integre a la perfección con Quarkus.

Llamar a la función `await` o `toIterable` produciría un fallo en el caso de que se estuviera ejecutando un subproceso de E/S para que no se bloquee dicho hilo de ejecución.

El grupo de hilos de trabajo de Quarkus es el mismo que el conjunto de subprocesos por defecto de Mutiny.

En el momento en el que se utiliza los de tipos de Mutiny, que hay que recordar que son Uni y Multi, se habilita de manera predeterminada la propagación de eventos.

2.6 Cliente SQL reactivo de Quarkus.

El cliente reactivo de Quarkus se basa en Vert.x, que es un framework de aplicaciones reactivas diseñado para construir sistemas eficientes y escalables en Java. Vert.x se está consolidando notablemente en el ámbito de las aplicaciones reactivas debido a su capacidad para ofrecer un gran rendimiento al aprovechar lo máximo posible el modelo de programación asincrónica y no bloqueante.

Quarkus, mediante la utilización del enfoque reactivo de Vert.x, ha desarrollado el módulo quarkus-reactive-pg-client que se basa en el cliente PostgreSQL que proporciona por Vert.x. Este cliente reactivo se integra de manera nativa en el ecosistema de Quarkus. Esto que permite a los desarrolladores poder realizar operaciones de E/S de manera mucho más eficiente con la base de datos PostgreSQL utilizando programación reactiva.

El cliente reactivo de Quarkus, que recordemos, está respaldado por Vert.x, ofrece una serie de ventajas que son notables. En primer lugar, permite realizar operaciones de consulta y modificación de datos de manera areactiva. Que hay que recordar que esto significa que las solicitudes de base de datos no bloquean la ejecución del hilo principal de la aplicación, lo que resulta en una mejor utilización de los recursos y una mayor capacidad de respuesta en comparación con los enfoques tradicionales basados en operaciones bloqueantes y sincrónicas.

Además, el cliente reactivo de Quarkus se basa en el modelo de programación reactiva, que se centra en el intercambio eficiente de mensajes y la propagación de secuencias de eventos. Esto permite construir aplicaciones reactivas que pueden manejar grandes cargas de trabajo y responder de una manera rápida y consistente, incluso en escenarios de alta concurrencia.

Otro aspecto importante es la escalabilidad. El cliente reactivo de Quarkus puede manejar de manera eficiente un gran número de solicitudes concurrentes, ya que puede aprovechar al máximo los recursos del sistema y distribuir las operaciones de manera eficiente entre los hilos disponibles.

Además, la integración del cliente reactivo de Quarkus con la base de datos PostgreSQL es sencilla y se realiza a través de una configuración simple en el archivo application.properties o application.yaml, donde se especifican los parámetros de conexión, como el host, puerto, nombre de usuario y contraseña.

En resumen, el cliente reactivo de Quarkus, basado en Vert.x, ofrece a los desarrolladores un enfoque novedoso y eficiente para interactuar con la base de datos PostgreSQL. Al aprovechar la programación reactiva y el modelo asincrónico y no bloqueante, permite construir aplicaciones altamente eficientes, escalables y capaces de manejar cargas de trabajo intensivas, ofreciendo una gran experiencia de usuario.

3. Especificación de requisitos.

Tras haber realizado el estudio y exposición de las características, beneficios y métodos de la programación reactiva, se han realizado dos aplicaciones. La primera se implementó siguiendo el modelo de programación tradicional, en el que la ejecución es secuencial y la E/S es bloqueante. Respecto a la segunda, se ha realizado siguiendo el método de programación reactivo, en el que la ejecución no es secuencial y, además, las operaciones de E/S son no bloqueantes.

3.1 Descripción general de las aplicaciones.

De manera general, se desarrollarán dos aplicaciones mediante las cuales poder analizar diferencias para demostrar los beneficios del método reactivo frente al tradicional.

Se simulará el sistema del que dispone una inmobiliaria, en la que se almacenan los apartamentos que tiene registrados, las localidades en las que actúa y, por último, las puntuaciones que se le han asignado a los apartamentos comprendidos en una localidad. Respecto a esto, se podrán realizar operaciones de consulta, alta, baja y modificación de los datos que se encuentra en los diferentes módulos.

Ambas aplicaciones estarán basadas en microservicios y, además, la desarrollada mediante el método reactivo estará orientada a eventos. Estarán desarrolladas en Java y expondrán un punto final HTTP mediante el cual acceder a diferentes recursos.

En concreto las aplicaciones estarán formadas por cuatro microservicios, que serán localidades, apartamentos, puntuaciones y un gateway. Mediante esto, a la hora de completar una operación será necesaria la colaboración entre dos de estos microservicios, entre los que siempre estará el gateway, que actuará como receptor de la solicitud del usuario por medio de los principios Rest utilizados en HTTP y la dirigirá al módulo correspondiente.

Dentro de este ecosistema los datos se enviarán a través de la serialización de objetos Json y la comunicación entre el módulo gateway y los otros tres restantes se realizará mediante Rest Client, que, en el caso de la aplicación reactiva este también lo será, es decir, emplearemos Rest Client Reactive. Esto se resume en que el Gateway actuará como cliente de los demás módulos, que tendrán la función de actuar como servidores a la hora de comunicarle a este los datos en formato Json mediante el envío de solicitudes HTTP a los puntos finales que estos proporcionen.

Además, se implementa caché para aumentar el rendimiento y la velocidad de respuesta de la aplicación mediante Redis, que, del mismo modo que con lo anterior, en la aplicación reactiva también será caché reactivo. Esto se consigue gracias a la estructura de almacenamiento de datos en memoria de Redis, que se basa en el almacenamiento mediante clave-valor. Con la utilización de caché en el repositorio de la totalidad de los módulos se consigue una recuperación de datos de gran rendimiento y, además, una menor latencia.

Por último, el despliegue de la aplicación se realizará mediante Docker-compose, que automáticamente creará una red que unirá los diferentes microservicios entre sí, teniendo la posibilidad de añadir el número de instancias que se desea de cada uno de ellos. Además, a la hora de crear las diferentes bases de datos de cada módulo, basadas en PostgreSQL, se utilizará el servicio Flyway, que automáticamente creará las tablas, secuencias y restricciones que sean necesarias, contando también con la inclusión de los datos correspondientes.

También incluirán Swagger-ui, una herramienta de Open API que proporciona una interfaz sencilla para facilitar la exploración y prueba de las diferentes funcionalidades que se implementen.

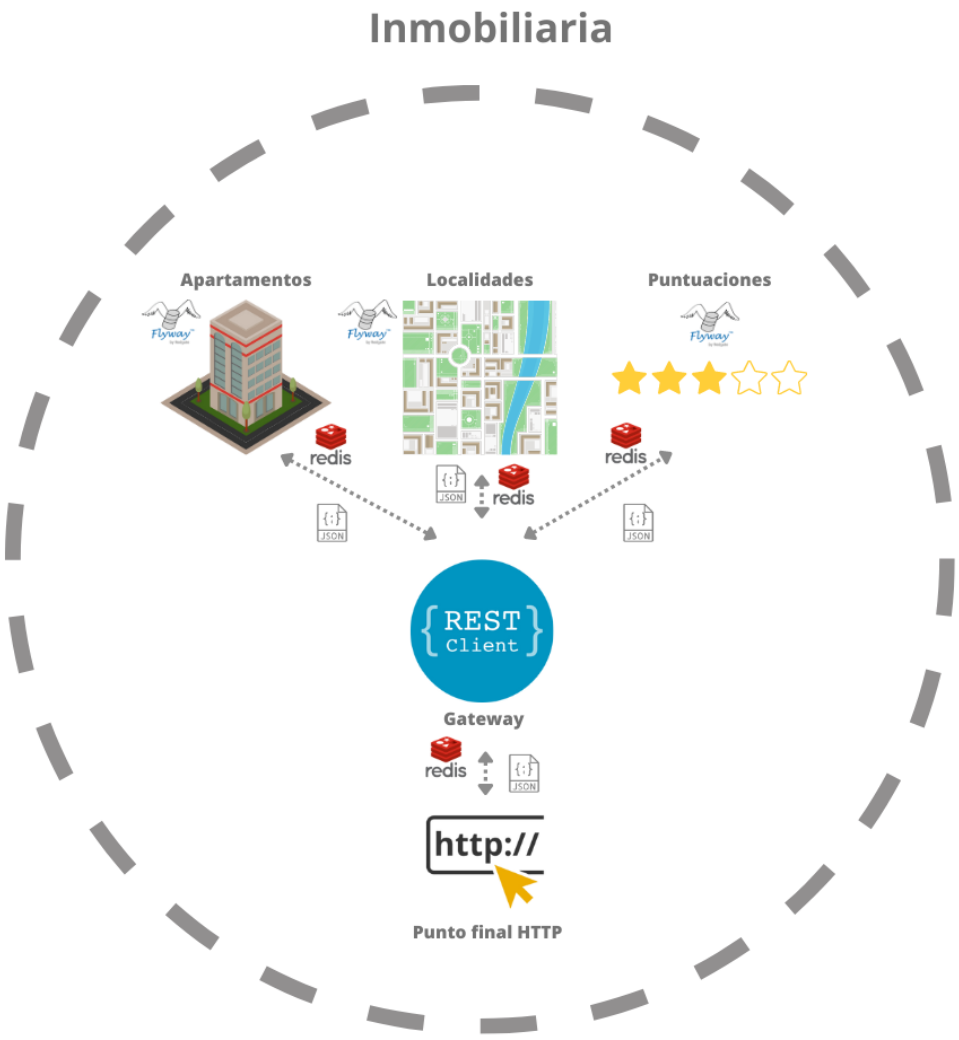

Figura 3-1 Esquema de la lógica de las aplicaciones

3.2 Casos de uso.

En este apartado, se definirán los casos de uso creados mediante UML (Unified Modeling Language) que se darán en las aplicaciones.

Los casos de uso son una técnica utilizada para capturar y representar las interacciones entre los actores (usuarios, sistemas externos u otros componentes) y el sistema en desarrollo. Estas representaciones gráficas permiten comprender y documentar de manera efectiva los diferentes escenarios y funcionalidades que deben ser considerados en el diseño y desarrollo del software.

Los casos de uso proporcionan una visión detallada de cómo interactúan los usuarios y otros elementos con el sistema, identificando las acciones que realizan y las respuestas que esperan del software.

Por tanto, los de las aplicaciones desarrolladas son los siguientes.

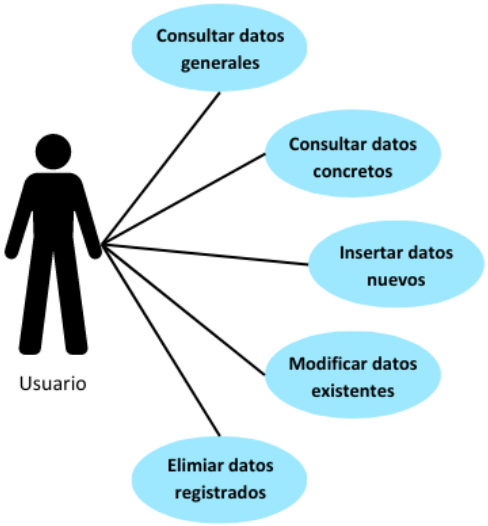

Figura 3-2 Diagramas de casos de uso 1, 2, 3, 4 y 5

Tabla 3-1 Caso de uso 1

Caso de uso 1

Actor	Usuario
Descripción	Consulta de datos general. Este caso de uso se utilizará para permitir que el usuario consulte todos los datos que hay registrados en el sistema de cada tipo. El sistema mostrará todos los campos ordenados por bloques. En el caso de que sean localidades, se mostrará el listado completo de localidades registradas en el sistema. En el caso de que sean apartamentos, se mostrará el listado completo de apartamentos registrados en el sistema. En el caso de que sean puntuaciones, se mostrará el listado completo de apartamentos registrados dentro de una localidad concreta que se encuentran en el sistema.

Tabla 3-2 Caso de uso 2

Caso de uso 2

Actor	Usuario
Descripción	Consulta de datos concretos. Este caso de uso se utilizará para permitir que el usuario consulte los datos referentes a un apartado concreto de los datos registrados. El sistema mostrará todos los campos del objeto que pertenecen a dicha consulta. En el caso de que sean localidades, se mostrará la localidad que corresponda con la búsqueda del usuario. En el caso de que sean apartamentos, se mostrará el apartamento que corresponda con la búsqueda del usuario. En el caso de que sean puntuaciones, se mostrará la localidad en la que se encuentra registrada el apartamento solicitado por el usuario.

Tabla 3-3 Caso de uso 3

Caso de uso 3

Actor	Usuario
Descripción	Insertar nuevos datos. Este caso de uso se utilizará para permitir que el usuario registre nuevos datos en la base de datos. El usuario debe cumplimentar los campos necesarios para la inserción. En el caso de que sean localidades, se añadirá una nueva localidad con todos sus campos que deberán ser cumplimentados por el usuario, a excepción del codloc. En el caso de que sean apartamentos, se añadirá un nuevo apartamento con todos sus campos que deberán ser cumplimentados por el usuario, a excepción del codapt.

Tabla 3-4 Caso de uso 4

Caso de uso 4

Actor	Usuario
Descripción	Modificar datos existentes. Este caso de uso se utilizará para permitir que el usuario modifique parte de los campos de la base de datos. El usuario deberá cumplimentar todos los datos, incluidos los que no desea modificar. En el caso de que sean localidades, se modificarán los datos de la localidad que el usuario desee en concreto. Es por esto por lo que deberá cumplimentar todos sus campos.

	En el caso de que sean apartamentos, se modificarán los datos del apartamento que el usuario desee en concreto. Es por esto por lo que deberá cumplimentar todos sus campos.

Tabla 3-5 Caso de uso 5

Caso de uso 5

Actor	Usuario
Descripción	Eliminar datos registrados. Este caso de uso se utilizará para permitir que el usuario elimine datos que ya se habían registrado en el sistema previamente. El sistema eliminará los datos basándose en el identificador. En el caso de que sean localidades, se eliminará la localidad que corresponda con la solicitud del usuario. En el caso de que sean apartamentos, se eliminará el apartamento que corresponda con la solicitud del usuario. En el caso de que sean puntuaciones, se eliminará tanto la puntuación del apartamento como la relación entre este y la localidad en la que se encuentra registrada.

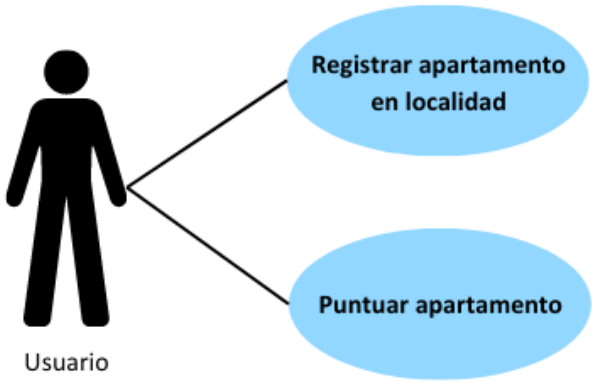

Figura 3-3 Diagramas de casos de uso 6 y 7

Tabla 3-6 Caso de uso 6

Caso de uso 6

Actor	Usuario
Descripción	Registrar apartamento en localidad. Este caso de uso se utilizará para permitir que el usuario registre un apartamento dentro de una localidad concreta. Esta será una operación de inserción en la base de datos por lo que será necesario proporcionar el codloc de la localidad y el codapt del apartamento.

Tabla 3-7 Caso de uso 7

Caso de uso 7	
Actor	Usuario
Descripción	Puntuar apartamento. Este caso de uso se utilizará para permitir que el usuario puntúe un apartamento que previamente ha sido registrado en una localidad. Esto será una modificación, por lo que el usuario debe indicar que apartamento concreto dentro de que localidad y, lógicamente, las estrellas que se le asignan.

3.3 Requisitos funcionales.

En este apartado, definiremos los requisitos funcionales que deberán cumplir las aplicaciones.

Estos requisitos son esenciales para determinar y especificar las funcionalidades específicas que deben ser implementadas en una aplicación con el fin de cumplir con los objetivos y necesidades del proyecto. Los requisitos funcionales describen de manera detallada las acciones, operaciones y comportamientos que se esperan del software, proporcionando una guía clara y precisa para el diseño, desarrollo y prueba de este.

Además, los requisitos funcionales también ayudan a priorizar las tareas y establecer una hoja de ruta clara para el desarrollo del software.

Por tanto, los de las aplicaciones desarrolladas son los siguientes:

Tabla 3-8 Requisito funcional 1

Requisito funcional 1	
Nombre	Base de datos.
Descripción	Cada vez que se añada o realice algún cambio en los datos debe quedar almacenado en un contenedor de la base de datos. Esta será PostgreSQL y se complementará con la utilización de Flyway.

Tabla 3-9 Requisito funcional 2

Requisito funcional 2	
Nombre	Operación de solicitud de datos.
Descripción	La aplicación contará con una opción en la que el usuario pueda consultar rápidamente los datos del listado general. Esta operación proporcionará unos datos u otros según la url que el usuario introduzca. En caso de solicitar el recurso /localidades, se proporcionará un listado completo de las diferentes localidades que hay registradas en el sistema, en el que cada localidad estará compuesta por un codloc,

comunidad autónoma a la que pertenece, cp general de la localidad, los habitantes de esta y su nombre. En caso de solicitar el recurso /apartamentos, se proporcionará un listado completo de los diferentes apartamentos que hay registrados en el sistema, en el que cada apartamento estará compuesto por un codapt, la dirección del apartamento, las vistas de las que dispone, el número de habitaciones, el de baños y, por último, su precio. En caso de solicitar el recurso /localidades/{codloc}/apartamentos, se proporcionará un listado completo de los diferentes apartamentos que se encuentren registrados en esa localidad y su respectiva puntuación, cada uno estará compuesto por el codapt, el codloc y, además, las estrellas que se le han asignado.

Tabla 3-10 Requisito funcional 3

Requisito funcional 3

Nombre	Operación de solicitud de datos concretos.
Descripción	La aplicación contará con una opción en la que el usuario pueda consultar rápidamente los datos de una localidad, apartamento o puntuación que se encuentre registrada en el sistema. Esta operación proporcionará unos datos u otros según la url que el usuario introduzca. En caso de solicitar el recurso /localidades/{codloc}, se proporcionará la localidad al completo, que estará compuesta por un codloc, comunidad autónoma a la que pertenece, cp general de la localidad, los habitantes de esta y su nombre. En caso de solicitar el recurso /apartamentos/{codapt}, se proporcionará el apartamento al completo, que estará compuesto por un codapt, la dirección del apartamento, las vistas de las que dispone, el número de habitaciones, el de baños y, por último, su precio. En caso de solicitar el recurso /apartamentos/{codapt}/localidades/{codloc}, se proporcionará la localidad en la que se encuentra registrada dicho apartamento, mostrando el codapt del apartamento, el codloc de la localidad y las estrellas en el caso de que estuviera puntuado. En caso de solicitar el recurso /localidades/{codloc}/apartamentos/{codapt}, se proporcionará la puntuación del apartamento con el codapt que se encuentra registrado en la localidad con el código codloc, que estará compuesta por un codloc de la localidad, un codapt del apartamento y las estrellas con las que está valorado.

Tabla 3-11 Requisito funcional 4

Requisito funcional 4

Nombre	Operación de registro de datos.
Descripción	La aplicación contará con una opción en la que el usuario pueda registrar nuevos datos rápidamente. Esta operación registrará unos datos u otros según la url que el usuario introduzca. En caso de querer añadir datos mediante la url /localidades, será necesario proporcionar el valor de los campos comunidad autónoma, cp, número de habitantes y nombre. Una vez proporcionado se registrará en el sistema, que creará automáticamente el codloc de esta para asegurarse de que es único. En caso de querer añadir datos mediante la url /apartamentos, será necesario proporcionar el valor de los campos dirección, vistas, número de habitaciones y baños y el precio. Una vez proporcionado se registrará en el sistema, que creará automáticamente el codapt de este para asegurarse de que es único. En caso de querer añadir datos mediante la url /localidades/{codloc}/apartamentos/{codapt}, no será necesario proporcionar el valor de ningún campo, ya que se registrará el apartamento con dicho código dentro de la localidad a la que pertenezca el código de la url.

Tabla 3-12 Requisito funcional 5

Requisito funcional 5

Nombre	Operación de modificación de datos.
Descripción	La aplicación contará con una opción en la que el usuario pueda modificar rápidamente los datos de algún campo que se encuentre previamente registrado. Esta operación modificará unos datos u otros según la url que el usuario introduzca. En caso de querer modificar datos mediante la url /localidades, será necesario proporcionar el valor de los campos codloc, comunidad autónoma, cp, número de habitantes y nombre. Será necesario proporcionar todos los datos, pero los que se deseen modificar deberán proporcionarse con el nuevo valor. En caso de querer modificar datos mediante la url /apartmaentos, será necesario proporcionar el valor de los campos codapt, dirección, vistas, número de habitaciones y baños y el precio. Será necesario proporcionar todos los datos, pero los que se deseen modificar deberán proporcionarse con el nuevo valor. En caso de querer modificar datos mediante la url /localidades/{codloc}/apartamentos/{codapt}, será necesario proporcionar el valor del campo estrellas. Esto se debe a que la

puntuación se añadirá teniendo en cuenta el codloc y el codapt de la url.

Tabla 3-13 Requisito funcional 6

Requisito funcional 6	
Nombre	Operación de eliminación de datos.
Descripción	La aplicación contará con una opción en la que el usuario pueda eliminar rápidamente los datos que se encuentren previamente registrados. Esta operación eliminará unos datos u otros según la url que el usuario introduzca. En el caso de querer eliminar datos mediante la url /localidades/{codloc}, se elminarán los datos correspondientes a la localidad que corresponda al codloc proporcionado en la url. En el caso de querer eliminar datos mediante la url /apartamentos/{codapt}, se elminarán los datos correspondientes al apartamento que corresponda al codpat proporcionado en la url. En el caso de querer eliminar datos mediante la url /localidades/{codloc}/apartamentos/{codapt}, se elminarán los datos que relacionan al apartamento añadido dentro de una localidad, y en caso de que existiera, su puntuación.

3.4 Requisitos no funcionales.

En este apartado, se definirán los requisitos no funcionales que deberán cumplir las aplicaciones.

Estos requisitos son igualmente importantes que los funcionales, ya que definen atributos de calidad, restricciones y características técnicas que debe cumplir el software. A diferencia de los requisitos funcionales que se centran en las funcionalidades, los requisitos no funcionales se enfocan en aspectos como el rendimiento, la seguridad, la usabilidad, la escalabilidad y otros aspectos no directamente relacionados con las acciones del software.

Es importante destacar que los requisitos no funcionales a menudo se entrelazan con los funcionales y pueden influir en su diseño e implementación. Por lo tanto, es fundamental considerar tanto los requisitos funcionales como los no funcionales para lograr un software exitoso y de calidad.

En el caso de las aplicaciones desarrolladas son los siguientes.

Tabla 3-14 Requisito no funcional 1

Requisito no funcional 1

Nombre	Rendimiento
Descripción	La aplicación deberá funcionar de la manera más fluida y rápida posible, sin que se produzcan largas esperas que hagan que el usuario desista. Esto se realizará mediante la implementación de caché, además de por medio de la utilización de los tipos Uni y Multi propuestos por Mutiny en el modelo reactivo.

4. Desarrollo.

4.1 Diseño de la arquitectura del sistema.

En este apartado, se mostrará el diseño de la arquitectura empleada en las diferentes aplicaciones, un aspecto fundamental en el desarrollo de sistemas robustos y escalables.

El diseño de la arquitectura de software es esencial para garantizar que el sistema cumpla con los requisitos funcionales y no funcionales establecidos, y que se pueda adaptar y evolucionar de manera eficiente a medida que los requerimientos cambien. Una arquitectura bien diseñada permite una mayor flexibilidad, mantenibilidad y reutilización de componentes, lo que a su vez facilita la construcción y el mantenimiento del software.

Al tratarse de dos aplicaciones similares en cuanto a las funciones que ofrecen, pero diferentes en la manera de desarrollarse, se expondrán dos diseños diferentes.

4.1.1 Presentación de las tecnologías utilizadas.

Lo primero será definir brevemente las diferentes tecnologías utilizadas en dicha arquitectura para poder comprenderla mejor. Posteriormente se explicará la conexión entre estas.

(1) PostgreSQL. Es un sistema de gestión de bases de datos relacionales de código abierto publicado con una licencia de PostgreSQL. Es compatible con herramientas relacionales (SQL) y no relacionales (JSON) y ofrece funciones de SQL avanzadas, como claves externas, subconsultas y activadores. Además, PostgreSQL tiene una gran capacidad de ampliación, lo que te permite definir tipos de datos y generar funciones personalizadas.

Flyway. Es una herramienta de migración de bases de datos de código abierto. Permite a los desarrolladores controlar y automatizar los cambios en la estructura y contenido de las bases de datos a lo largo del tiempo. Flyway utiliza scripts SQL para definir y ejecutar las migraciones, lo que facilita el versionado y la colaboración en proyectos de desarrollo de software.

Java. Es un lenguaje de programación de alto nivel, portátil y orientado a objetos. Es conocido por su seguridad, robustez y capacidad de ejecutarse en diferentes plataformas a través de la máquina virtual Java (JVM). Tiene una amplia biblioteca estándar y es ampliamente utilizado en el desarrollo de aplicaciones empresariales, móviles y web.

Quarkus. Es un framework de desarrollo de aplicaciones Java que se destaca por su enfoque en la creación de aplicaciones nativas en la nube. Proporciona un arranque rápido y un bajo consumo de memoria, lo que lo hace ideal para entornos de contenedores y microservicios.

Quarkus ofrece herramientas y bibliotecas eficientes para desarrollar aplicaciones escalables y de alto rendimiento.

Smallrye Mutiny. Es una biblioteca de programación reactiva diseñada para facilitar la creación de aplicaciones asincrónicas y reactivas en Java. Proporciona un conjunto de constructores y operadores que permiten trabajar con flujos de datos de manera eficiente y concisa.

Vert.x. Es un framework de aplicaciones reactivas y orientado a eventos, diseñado para construir aplicaciones de alto rendimiento y escalables. Proporciona un modelo de programación basado en el manejo eficiente de eventos y permite desarrollar aplicaciones en varios lenguajes, como Java, Kotlin, Groovy y JavaScript. Vert.x es conocido por su capacidad para manejar grandes volúmenes de conexiones concurrentes y su integración con tecnologías como HTTP, TCP, WebSocket y bases de datos.

Redis Caché. Es una solución de almacenamiento en memoria de alto rendimiento que se utiliza para acelerar el acceso a datos en aplicaciones web y sistemas distribuidos. Funciona como una base de datos en memoria que almacena datos en pares clave-valor, permitiendo un acceso rápido y eficiente a la información. Redis Caché se caracteriza por su capacidad de manejar grandes volúmenes de datos y por ofrecer funciones avanzadas como la expiración de datos, la persistencia en disco y la capacidad de almacenar y manipular estructuras de datos complejas.

HTTP. Las siglas significan Protocolo de Transferencia de Hipertexto. Este es un protocolo de comunicación empleado en la World Wide Web (WWW) con la intención de que la transferencia de datos entre un servidor y un cliente sea más sencilla. HTTP permite solicitar recursos como páginas web, imágenes y archivos, mediante el uso de URL (Uniform Resource Locators) y proporciona un conjunto de métodos estándar como GET, POST, PUT y DELETE para realizar diferentes tipos de solicitudes. Además, HTTP utiliza códigos de estado para indicar el resultado de una operación.

4.1.2 Aplicación reactiva.

La arquitectura de cada módulo con acceso a la base de datos de la aplicación desarrollada mediante el modelo reactivo constará de tres partes. La base de datos, que se encargará de todo el almacenamiento de estos; el backend que se encargará de la gestión de las solicitudes y de las modificaciones que haya que realizar en la base de datos; y, por último, los puntos de acceso HTTP al módulo.

Respecto a la base de datos se ha elegido PostgreSQL ya que es una base de datos relacional que está perfectamente adaptada al modelo reactivo. Esto, junto con Flyway crea un entorno prácticamente perfecto para que a la hora de desplegarse mediante Docker-compose se realicen todas las migraciones necesarias para disponer de los datos en todo momento.

El backend se ha desarrollado mediante el lenguaje de programación Java, dentro del entorno del framework de Quarkus. Además, se ha hecho uso de los tipos que proporciona Mutiny para la comunicación de eventos ya que como se ha comentado el modelo reactivo se basa en la gestión de estos. Junto a esto, Vert.x ha sido el que ha brindado las opciones de comunicarse con la base de datos a modo de cliente reactivo SQL.

Además, se ha añadido Redis caché reactivo en el repositorio de cada uno de los módulos, implementado también mediante la utilización de los tipos propuestos por Mutiny, logrando un acceso a los datos todavía más rápido. Es decir, el método reactivo ya mejora la rapidez de

respuesta de la aplicación, pero si, también se añade memoria caché reactiva, mejora exponencialmente.

Para los puntos de acceso a la aplicación se ha utilizado HTTP, con sus respectivos métodos GET, PUT, POST y DELETE, mediante los cuales se puede acceder a las funciones que estos posibilitan. Esto es posible gracias a los principios de Rest Response que también se integran en el lenguaje java

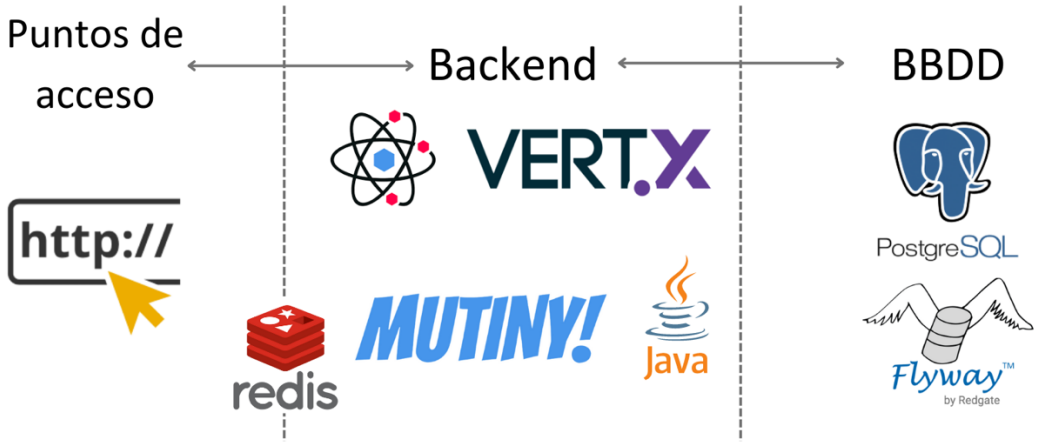

Figura 4-1 Esquema de la arquitectura de la aplicación reactiva

4.1.3 Aplicación tradicional.

La arquitectura de cada módulo con acceso a la base de datos de la aplicación desarrollada mediante el modelo tradicional constará, del mismo modo que la anterior, de tres partes. La base de datos, que se encargará de todo el almacenamiento de estos; el backend que se encargará de la gestión de las solicitudes y de las modificaciones que haya que realizar en la base de datos; y, por último, los puntos de acceso HTTP al módulo.

Respecto a la base de datos se ha elegido PostgreSQL ya que es una base de datos relacional que está perfectamente adaptada al modelo reactivo. Esto, junto con Flyway crea un entorno prácticamente perfecto para que a la hora de desplegarse mediante Docker-compose se realicen todas las migraciones necesarias para disponer de los datos en todo momento.

El backend se ha desarrollado mediante el lenguaje de programación Java, dentro del entorno del framework de Quarkus. Mediante estos se accede a la base de datos como cliente y se atienden las solicitudes.

Además, se ha añadido Redis caché en el repositorio de cada uno de los módulos, logrando un acceso a los datos más rápido ya que se quedan almacenados y no es necesario que se vuelva a realizar la operación de E/S a la base de datos. Con esto se logra mejorar notablemente el rendimiento de la aplicación, supliendo en muchos casos la espera que pueden producir algunas de estas operaciones.

Para los puntos de acceso a la aplicación se ha utilizado HTTP, con sus respectivos métodos GET, PUT, POST y DELETE, mediante los cuales se puede acceder a las funciones que estos posibilitan.

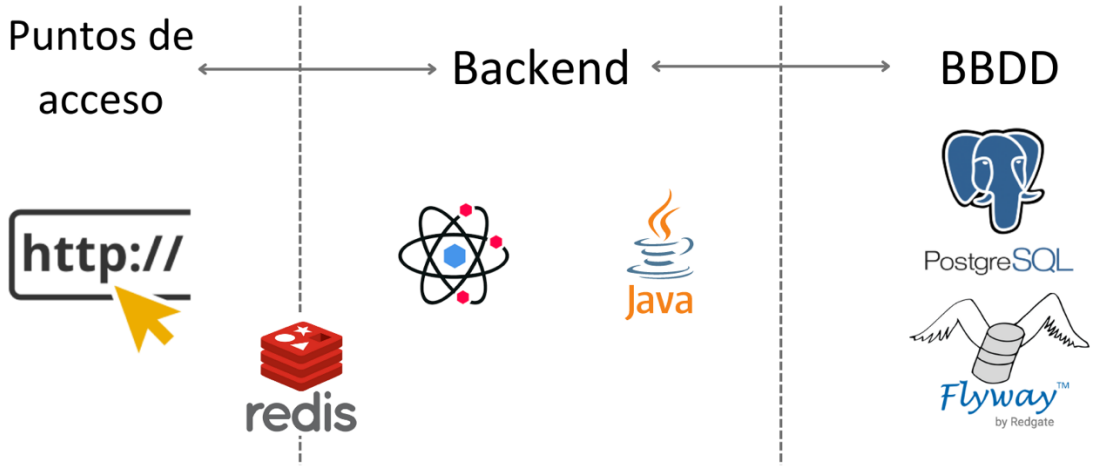

Figura 4-2 Esquema de la arquitectura de la aplicación tradicional

4.2 Esquema de la base de datos.

El esquema de datos de una base de datos se refiere a la estructura lógica y organización de los datos dentro de la base de datos. Define la manera en que los datos serán almacenados, relacionados y accedidos.

El esquema de datos también especifica los tipos de datos que se pueden almacenar en cada columna de una tabla, las claves primarias y foráneas que definen las relaciones entre las tablas, y las restricciones que se aplican para garantizar la validez y consistencia de los datos.

En las aplicaciones se ha utilizado el tipo VARCHAR en todos los campos que no fueran estrictamente numéricos debido a que solo almacena los caracteres que se introduzcan, y no la cantidad determinada inicialmente. Esto supone un gran ahorro de memoria en sistemas de gran tamaño y añade un grado más de escalabilidad a la aplicación.

4.2.1 Módulo localidades.

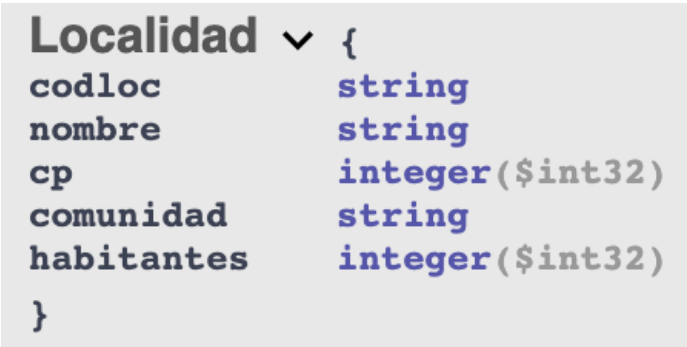

Figura 4-3 Esquema de los datos de la tabla localidades de la bbdd

Estará comprendido por cinco campos. El primero será codloc, que será del tipo VARCHAR. Este campo corresponderá con la calve primaria de la tabla y se añadirá de manera automática mediante una secuencia, de manera que el usuario no podrá influir en él. De este modo se

disminuyen notablemente los errores en relación con la clave primaria. Además, tendrá una limitación de once caracteres.

El segundo será la comunidad a la que pertenece la localidad. Este también es del tipo VARCHAR y está restringido con un máximo de veinte caracteres. Se comprobará que el campo no es igual a null.

El tercero será el cp general de la localidad, es decir, el código postal. Este será del tipo NUMERIC y se comprobará que como máximo tenga una extensión de cinco cifras. Además, tampoco podrá ser igual a null.

El siguiente será el número de habitantes de la localidad. Este también será del tipo NUMERIC y se comprobará que como máximo tenga siete cifras. También se comprobará que el valor introducido sea mayor que cero.

El último es el nombre de la localidad. Este será del tipo VARCHAR y se comprobará que no se superen los cien caracteres. Además, tampoco podrá ser null.

4.2.2 Módulo apartamentos.

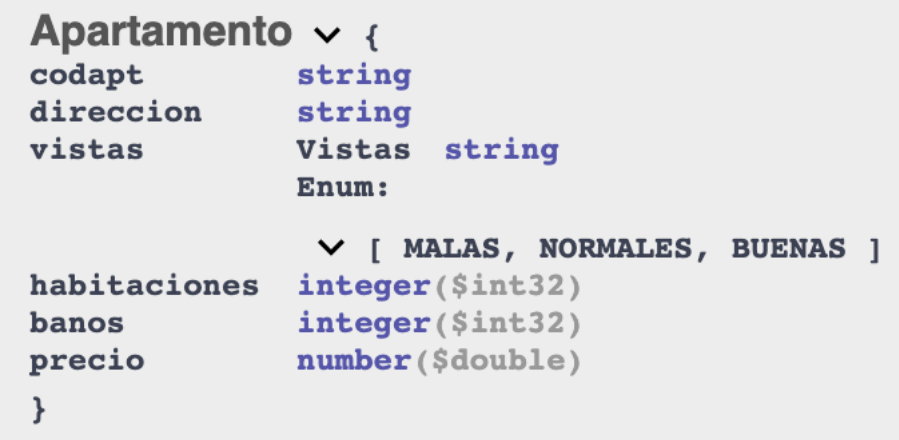

Figura 4-4 Esquema de los datos de la tabla apartamentos de la bbdd

Estará comprendido por seis campos. El primero será codapt, que será del tipo VARCHAR. Este campo corresponderá con la calve primaria de la tabla y se añadirá de manera automática mediante una secuencia, de manera que el usuario no podrá influir en él. De este modo se disminuyen notablemente los errores en relación con la clave primaria. Además, tendrá una limitación de once caracteres.

El segundo será la dirección en la que se encuentra dicho apartamento. Este también es del tipo VARCHAR y está restringido con un máximo de cien caracteres. Se comprobará que el campo no es igual a null.

El tercero serán las vistas de las que se dispone desde el apartamento. Este será del tipo NUMERIC y se comprobará que se el valor de estas se encuentre comprendido entre 1 y 3. Esto se debe a que en la aplicación estos valores estarán asignados a una enumeración. En esta tendremos las opciones de MALAS, NORMALES y BUENAS respectivamente.

El siguiente será el número de habitaciones que se encuentran en el apartamento. Este también será del tipo NUMERIC y se comprobará que el valor introducido sea mayor que cero.

El penúltimo será el número de baños que se encuentran en el apartamento. Este también será del tipo NUMERIC y se comprobará que el valor introducido sea mayor que cero.

El último es el precio al que se encuentra el apartamento. Este será del tipo NUMERIC y solamente se guardarán dos decimales. Además, se comprobará que el valor introducido sea mayor que cero.

4.2.3 Módulo puntuaciones.

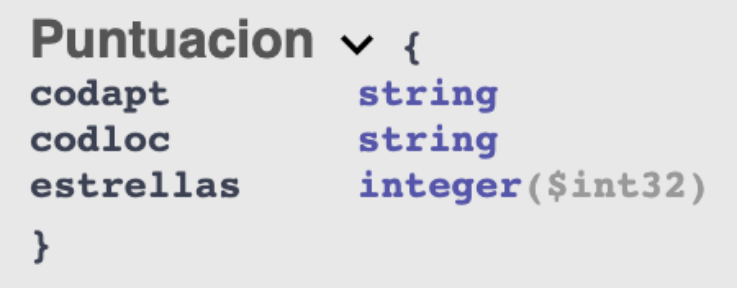

Figura 4-5 Esquema de los datos de la tabla puntuaciones de la bbdd

Estará comprendido por tres campos. El primero será codapt, que será de tipo VARCHAR. Este campo corresponderá con la clave primaria de la tabla de apartamentos.

El primero será codloc, que será de tipo VARCHAR. Del mismo modo que el anterior, corresponderá con la clave primaria de la tabla de localidades. De este modo queda relacionado cada apartamento con la localidad en la que se encuentre.

El último será las estrellas que se le han asignado como valoración al apartamento. Este campo será del tipo NUMERIC y se comprobará que se encuentre entre 0 y 5.

4.3 Recursos de las aplicaciones.

En este apartado, se presentarán los recursos web habilitados como parte del diseño del sistema. Los recursos web juegan un papel fundamental en el desarrollo de aplicaciones modernas, ya que permiten la exposición y acceso a datos y funcionalidades a través de la arquitectura REST (Representational State Transfer).

Los recursos web habilitados son elementos clave en la implementación de servicios web RESTful, que siguen los principios y estándares establecidos por la arquitectura REST. Estos recursos representan entidades o conjuntos de datos específicos que pueden ser accedidos, creados, actualizados o eliminados mediante las operaciones RESTful como GET, POST, PUT y DELETE.

Una vez que se han presentado, los recursos que se han implementado en las aplicaciones son los siguientes. Es importante destacar que la comunicación de objetos se realizará mediante JSON.

4.3.1 GET/localidades

El propósito de esta funcionalidad es la de listar todas las localidades que se encuentran almacenadas en la base de datos.

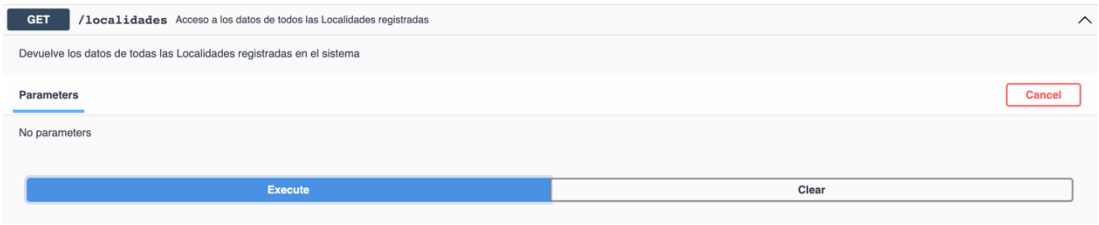

Figura 4-6 GET/localidades

El método HTTP empleado para esto es el GET y no necesitará parámetros ya que se está realizando una búsqueda general.

Para acceder a este recurso hemos utilizado el resource path http://localhost:80/localidades

Puede haber dos tipos de respuesta, que son el Status code 200 OK en el caso de que se encuentren localidades que mostrar. Pero en el caso opuesto se responderá con un 404 Not Found, que indicará que no se encontraron objetos.

Por tanto, si se encuentran localidades obtendremos una lista de objetos json de aspecto similar al siguiente.

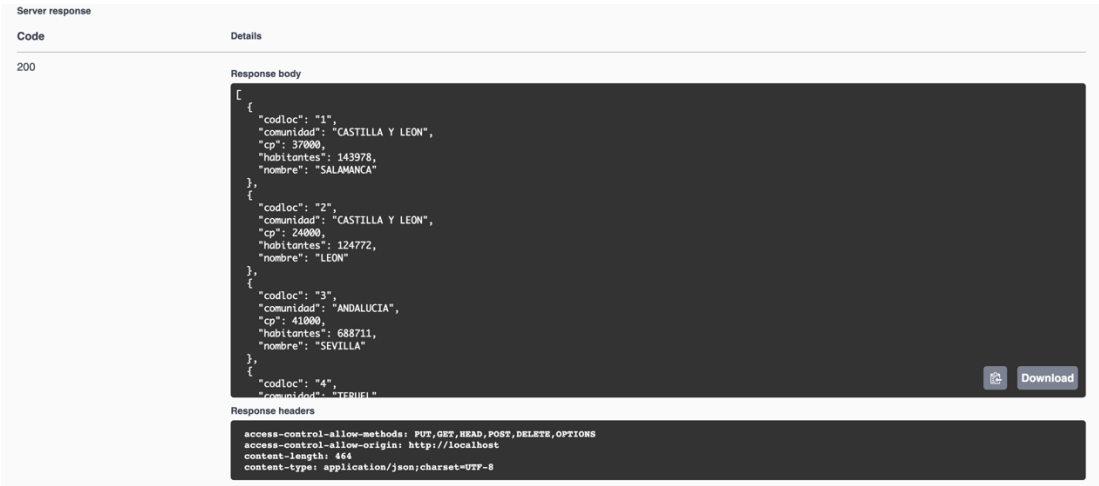

Figura 4-7 Resultado obtenido GET/localidades

4.3.2 GET/localidades/{codloc}

El propósito de esta funcionalidad es la de mostrar una localidad que ha sido buscada en la base de datos mediante su codloc.

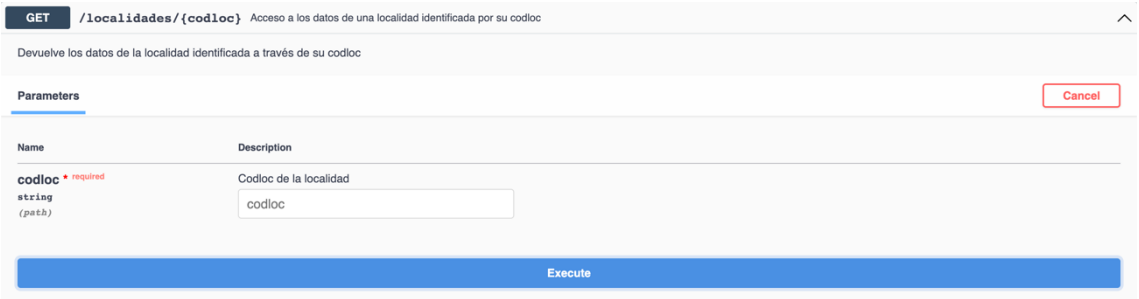

Figura 4-8 GET/localidades/{codloc}

El método HTTP empleado para esto es el GET y necesitará un parámetro del tipo PathParam que nos indique el codloc de la localidad que se desea consultar.

Para acceder a este recurso hemos utilizado el resource path http://localhost:80/localidades/{codloc}

Puede haber dos tipos de respuesta, que son el Status code 200 OK en el caso de que se encuentre la localidad que se ha de mostrar. Pero en el caso opuesto se responderá con un 404 Not Found, que indicará que no se encontraron objetos.

Por tanto, si se encuentra la localidad obtendremos un objeto json de aspecto similar al siguiente.

Figura 4-9 Resultado obtenido GET/localidades/{codloc}

4.3.3 POST/localidades

El propósito de esta funcionalidad es la de añadir una localidad con todos sus campos, incluido el codloc que se añadirá automáticamente mediante una secuencia. Además, nos devolverá todos los datos de la localidad al completo.

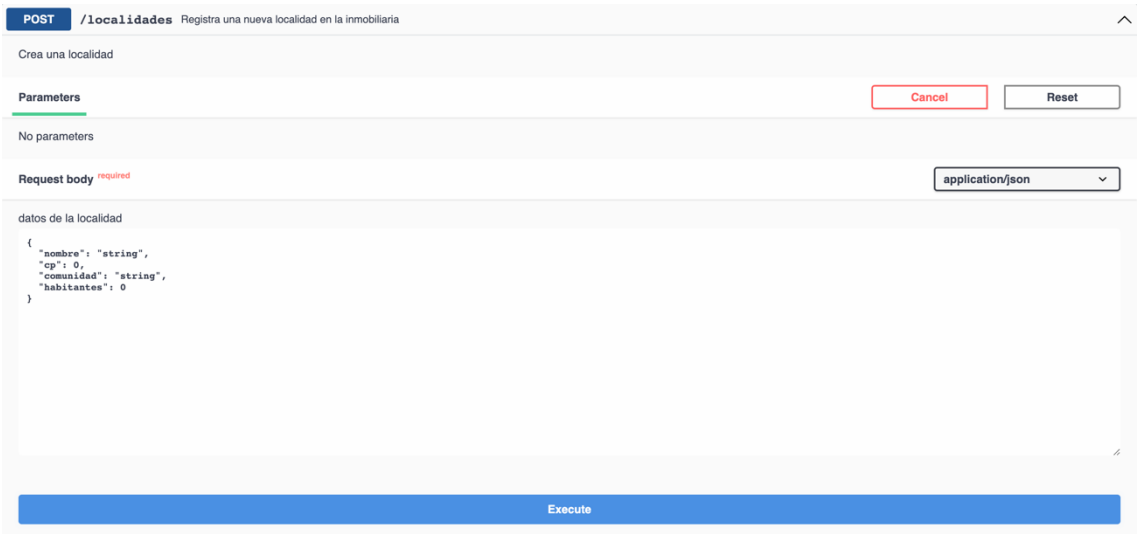

Figura 4-10 POST/localidades

El método HTTP empleado para esto es el POST y necesitará consumir como Media Type Entity un objeto Json con todos los datos de la localidad salvo el codloc.

Para acceder a este recurso hemos utilizado el resource path http://localhost:80/localidades

En este caso puede haber dos tipos de respuesta. Estos son el Status code 201 Created en el caso de que se añada la localidad correctamente. Pero, también existirá la posibilidad de responder con un 500 Server Error. Esto se debe a que el usuario está definiendo el valor de los campos y esto puede conllevar algún error.

Por tanto, si se añade correctamente la localidad obtendremos un objeto json de aspecto similar al siguiente.

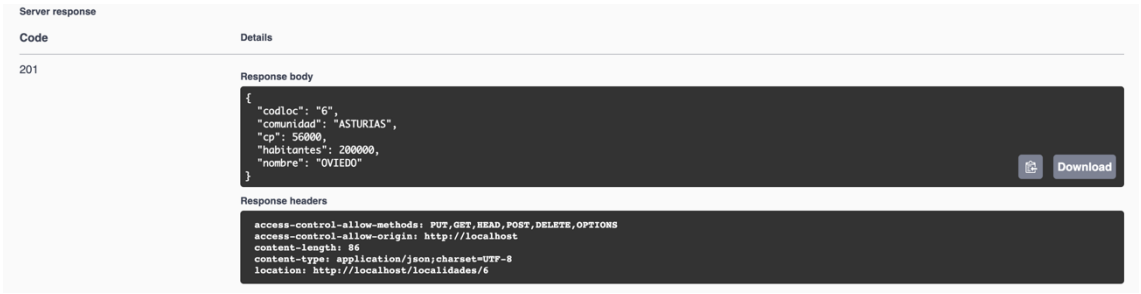

Figura 4-11 Resultado obtenido POST/localidades

4.3.4 PUT/localidades

El propósito de esta funcionalidad es la de actualizar los campos de una localidad sin incluir el codloc ya que este no puede ser modificado.

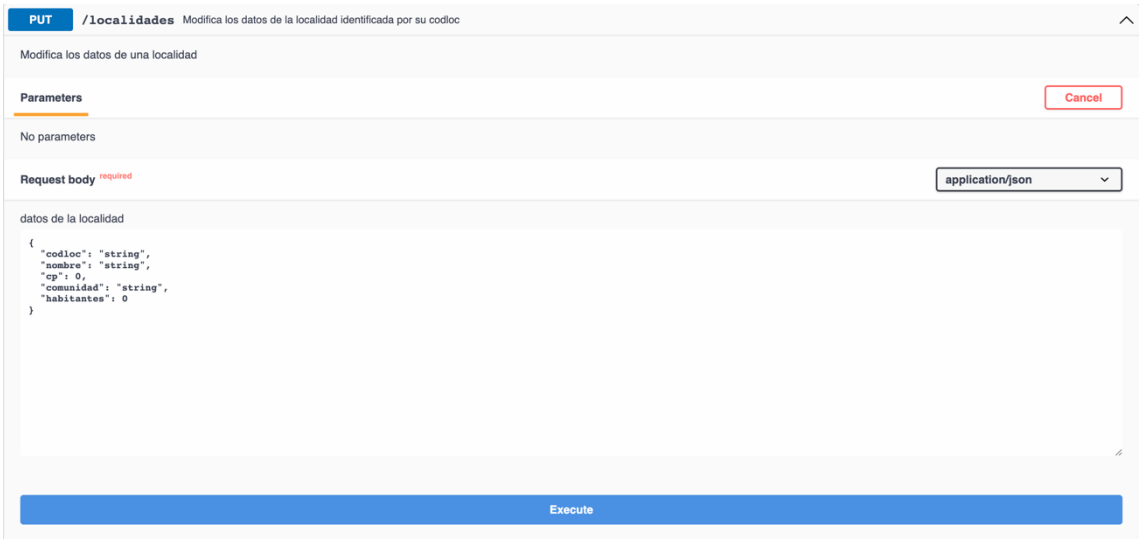

Figura 4-12 PUT/localidades

El método HTTP empleado para esto es el PUT y necesitará consumir como Media Type Entity un objeto Json con todos los datos de la localidad con los nuevos valores.

Para acceder a este recurso hemos utilizado el resource path http://localhost:80/localidades

Puede haber tres tipos de respuesta, que son el Status code 200 Ok en el caso de que se encuentre la localidad que se ha de actualizar y se actualice. Pero en el caso opuesto se responderá con un 404 Not Found, que indicará que no se encontraron objetos. Pero, también existirá la posibilidad de responder con un 500 Server Error. Esto se debe a que el usuario está definiendo el valor de los campos y esto puede conllevar algún error.

Por tanto, si se actualiza correctamente la localidad obtendremos un objeto json de aspecto similar al siguiente.

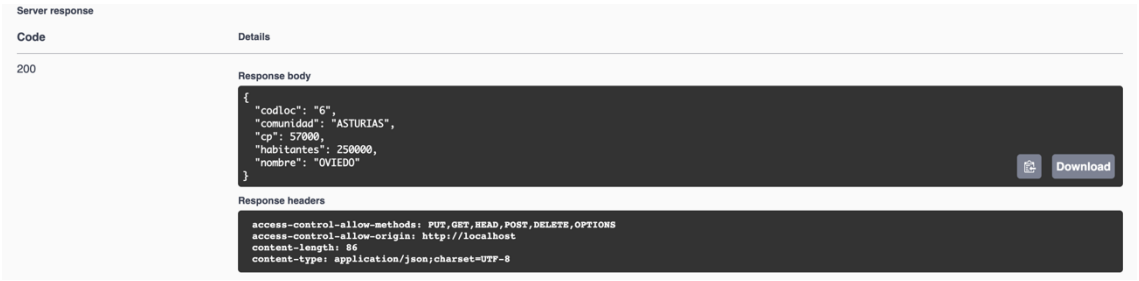

Figura 4-13 Resultado obtenido PUT/localidades

4.3.5 DELETE/localidades/{codloc}

El propósito de esta funcionalidad es la de eliminar una localidad que ha sido buscada en la base de datos mediante su codloc.

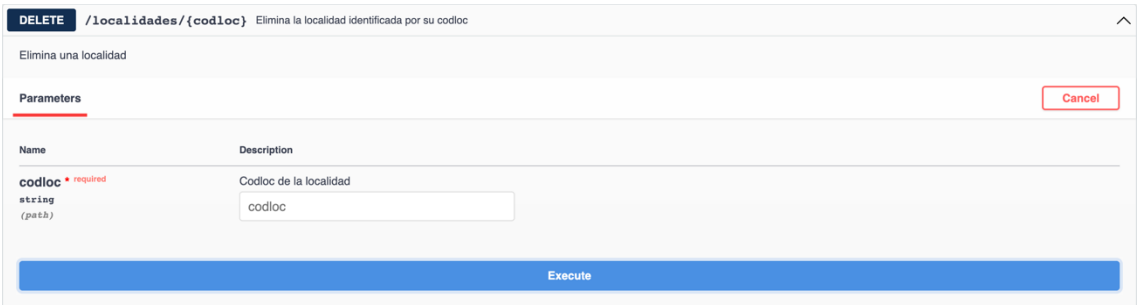

Figura 4-14 DELETE/localidades/{codloc}

El método HTTP empleado para esto es el DELETE y necesitará un parámetro del tipo PathParam que nos indique el codloc de la localidad que se desea eliminar.

Para acceder a este recurso hemos utilizado el resource path http://localhost:80/localidades/{codloc}

Puede haber dos tipos de respuesta, que son el Status code 204 No Content en el caso de que se encuentre la localidad que se ha de eliminar y se haga correctamente. Pero en el caso opuesto se responderá con un 404 Not Found, que indicará que no se encontraron objetos.

4.3.6 GET/apartamentos

El propósito de esta funcionalidad es la de listar todos los apartamentos que se encuentren almacenados en la base de datos.

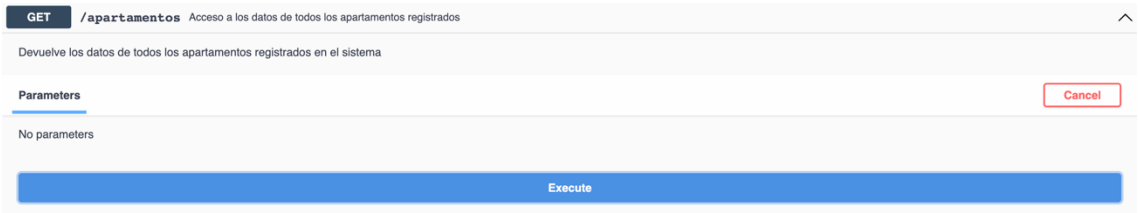

Figura 4-15 GET/apartamentos

El método HTTP empleado para esto es el GET y no necesitará parámetros ya que se está realizando una búsqueda general.

Para acceder a este recurso hemos utilizado el resource path http://localhost:80/apartamentos

Puede haber dos tipos de respuesta, que son el Status code 200 OK en el caso de que se encuentren apartamentos que mostrar. Pero en el caso opuesto se responderá con un 404 Not Found, que indicará que no se encontraron objetos.

Por tanto, si se encuentran apartamentos obtendremos una lista de objetos json de aspecto similar al siguiente.

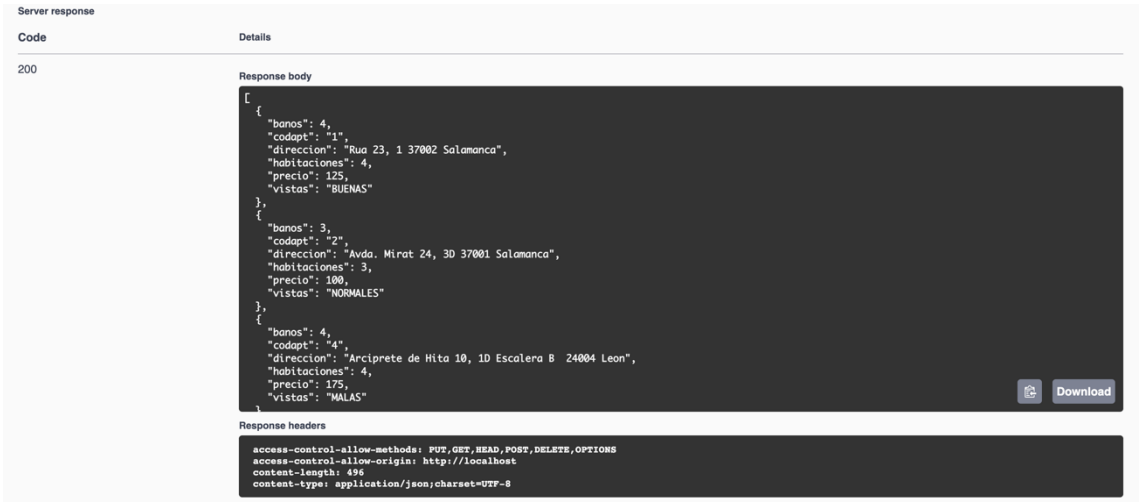

Figura 4-16 Resultado obtenido GET/apartamentos

4.3.7 GET/apartamentos/{codapt}

El propósito de esta funcionalidad es la de mostrar un apartamento que ha sido buscado en la base de datos mediante su codapt.

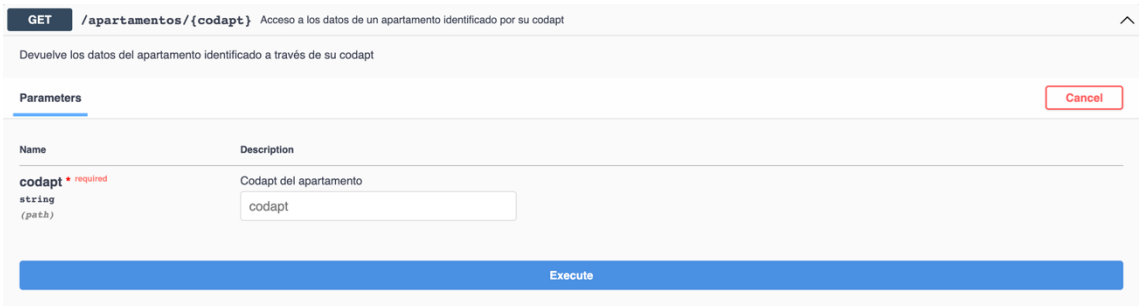

Figura 4-17 GET/apartamentos/{codapt}

El método HTTP empleado para esto es el GET y necesitará un parámetro del tipo PathParam que nos indique el codapt del apartamento que se desea consultar.

Para acceder a este recurso hemos utilizado el resource path http://localhost:80/apartamentos/{codapt}

Puede haber dos tipos de respuesta, que son el Status code 200 OK en el caso de que se encuentre el apartamento que se ha de mostrar. Pero en el caso opuesto se responderá con un 404 Not Found, que indicará que no se encontraron objetos.

Por tanto, si se encuentra el apartamento obtendremos un objeto json de aspecto similar al siguiente.

Figura 4-18 Resultado obtenido GET/apartamentos/{codapt}

4.3.8 POST/apartamentos

El propósito de esta funcionalidad es la de añadir un apartamento con todos sus campos, incluido el codapt que se añadirá automáticamente mediante una secuencia. Además, nos devolverá todos los datos del apartamento al completo.

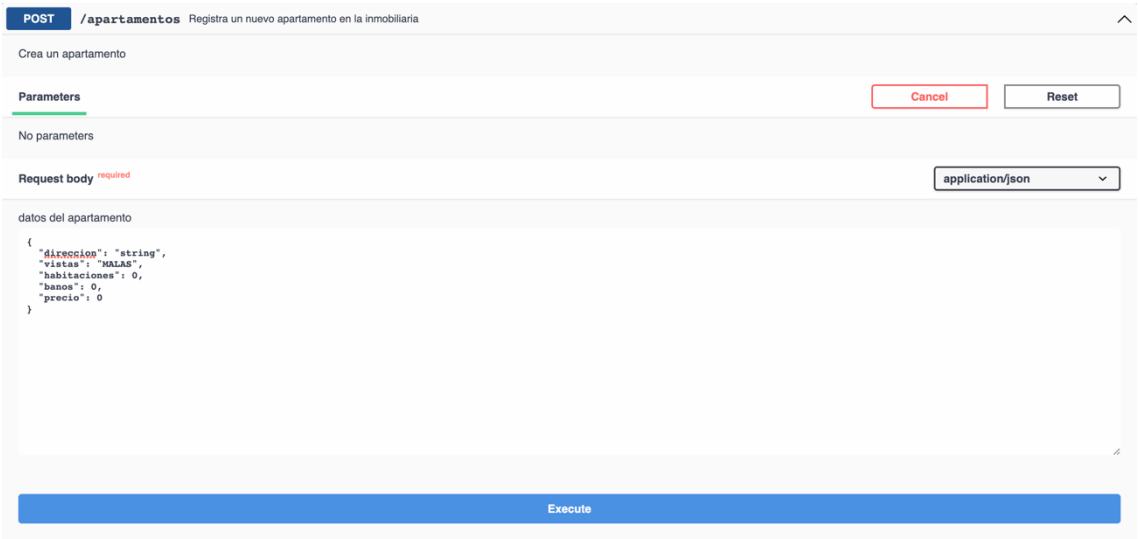

Figura 4-19 POST/apartamentos

El método HTTP empleado para esto es el POST y necesitará consumir como Media Type Entity un objeto Json con todos los datos del apartamento salvo el codapt.

Para acceder a este recurso hemos utilizado el resource path http://localhost:80/apartamentos

En este caso puede haber dos tipos de respuesta. Estos son el Status code 201 Created en el caso de que se añada el apartamento correctamente. Pero, también existirá la posibilidad de responder con un 500 Server Error. Esto se debe a que el usuario está definiendo el valor de los campos y esto puede conllevar algún error.

Por tanto, si se añade correctamente el apartamento obtendremos un objeto json de aspecto similar al siguiente.

Figura 4-20 Resultado POST/apartamentos

4.3.9 PUT/apartamentos

El propósito de esta funcionalidad es la de actualizar los campos de un apartamento sin incluir el codapt ya que este no puede ser modificado.

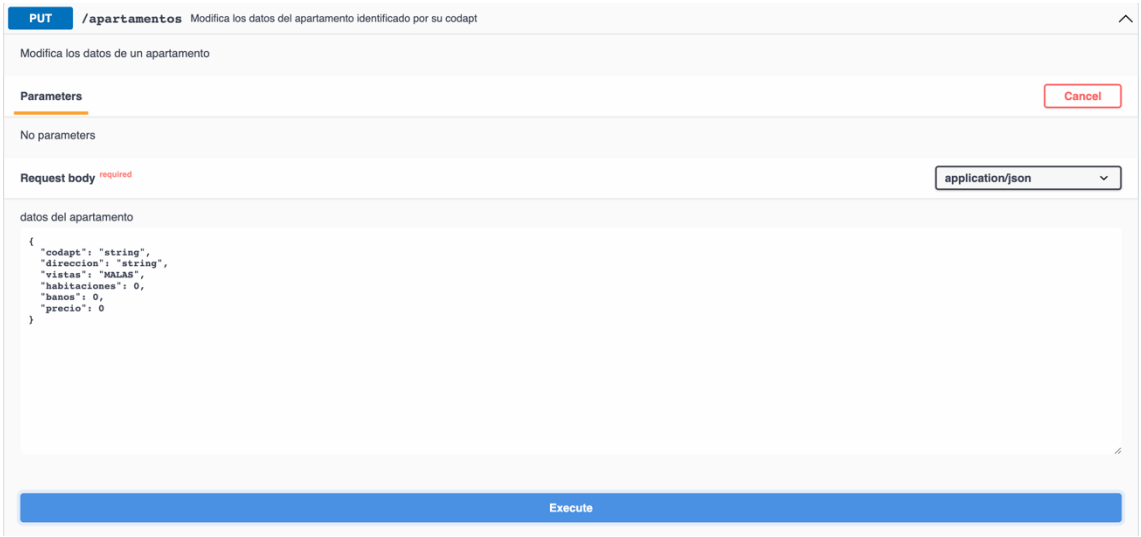

Figura 4-21 PUT/apartamentos

El método HTTP empleado para esto es el PUT y necesitará consumir como Media Type Entity un objeto Json con todos los datos del apartamento con los nuevos valores.

Para acceder a este recurso hemos utilizado el resource path http://localhost:80/apartamentos

Puede haber tres tipos de respuesta, que son el Status code 200 Ok en el caso de que se encuentre el apartamento que se ha de actualizar y se actualice. Pero en el caso opuesto se responderá con un 404 Not Found, que indicará que no se encontraron objetos. Pero, también existirá la posibilidad de responder con un 500 Server Error. Esto se debe a que el usuario está definiendo el valor de los campos y esto puede conllevar algún error.

Por tanto, si se actualiza correctamente el apartamento obtendremos un objeto json de aspecto similar al siguiente.

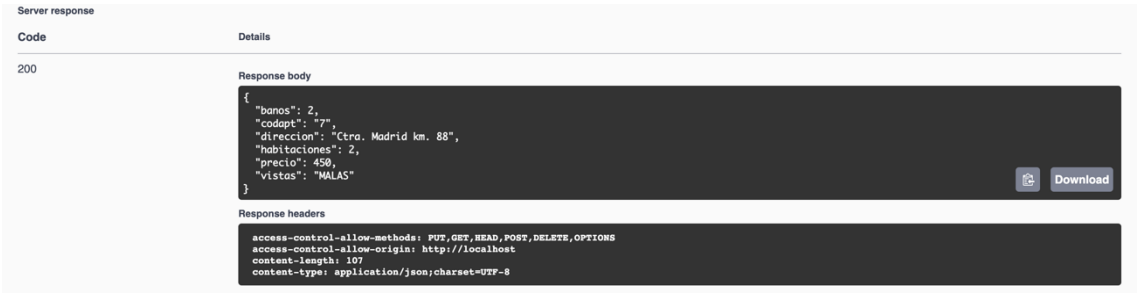

Figura 4-22 Resultado obtenido PUT/apartamentos

4.3.10 DELETE/apartamentos

El propósito de esta funcionalidad es la de eliminar un apartamento que ha sido buscado en la base de datos mediante su codapt.

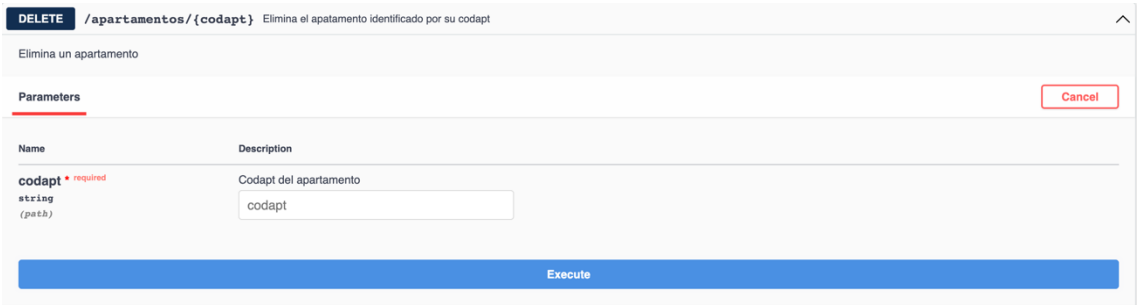

Figura 4-23 DELETE/aparatamentos/{codapt}

El método HTTP empleado para esto es el DELETE y necesitará un parámetro del tipo PathParam que nos indique el codapt del apartamento que se desea eliminar.

Para acceder a este recurso hemos utilizado el resource path http://localhost:80/apartamentos/{codapt}

Puede haber dos tipos de respuesta, que son el Status code 204 No Content en el caso de que se encuentre el apartamento que se ha de eliminar y se haga correctamente. Pero en el caso opuesto se responderá con un 404 Not Found, que indicará que no se encontraron objetos.

4.3.11 GET/apartamentos/{codapt}/localidades

El propósito de esta funcionalidad es la de mostrar la localidad en la que se encuentra registrado un apartamento que ha sido buscado en la base de datos mediante su codapt.

Figura 4-24 GET/apartamentos/{codapt}/localidades

El método HTTP empleado para esto es el GET y necesitará un parámetro del tipo PathParam que nos indique el codapt del apartamento que se desea consultar.

Para acceder a este recurso hemos utilizado el resource path http://localhost:80/apartamentos/{codapt}/localidades

Puede haber dos tipos de respuesta, que son el Status code 200 OK en el caso de que se encuentre la relación entre el apartamento y su localidad que se ha de mostrar. Pero en el caso opuesto se responderá con un 404 Not Found, que indicará que no se encontraron objetos.

Por tanto, si se encuentra el apartamento obtendremos un objeto json de aspecto similar al siguiente.

Figura 4-25 Resultado GET/apartamentos/{codapt}/localidades

4.3.12 GET/localidades/{codloc}/apartamentos

El propósito de esta funcionalidad es la de listar todos los apartamentos que se encuentren registrados dentro de una localidad que será buscada mediante su codloc.

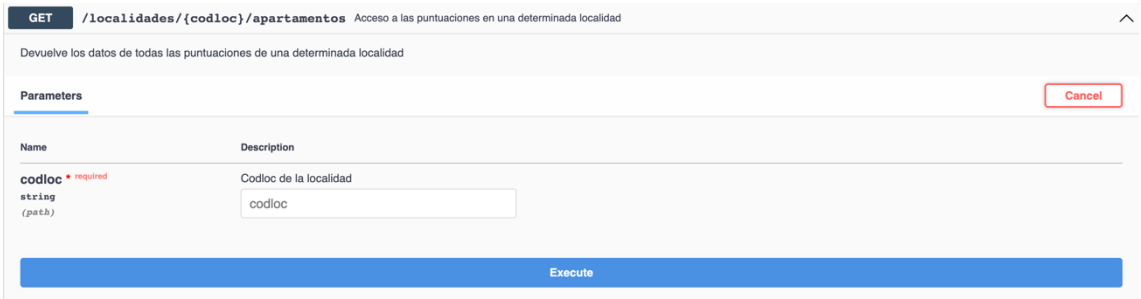

Figura 4-26 GET/localidades/{codloc}/apartamentos

El método HTTP empleado para esto es el GET y necesitará un parámetro del tipo PathParam que nos indique el codloc de la localidad que se desea consultar.

Para acceder a este recurso hemos utilizado el resource path http://localhost:80/localidades/{codloc}/apartamentos

Puede haber dos tipos de respuesta, que son el Status code 200 OK en el caso de que se encuentren relaciones entre apartamentos y la localidad que mostrar. Pero en el caso opuesto se responderá con un 404 Not Found, que indicará que no se encontraron objetos.

Por tanto, si se encuentran apartamentos obtendremos una lista de objetos json de aspecto similar al siguiente.

Figura 4-27 Resultado obtenido GET/localidades/{codloc}/apartamentos

4.3.13 GET/localidades/{codloc}/apartamentos/{codapt}

El propósito de esta funcionalidad es la de mostrar la puntuación que se le ha asignado a un apartamento concreto registrado en una localidad, que serán consultados mediante su codloc y su codapt.

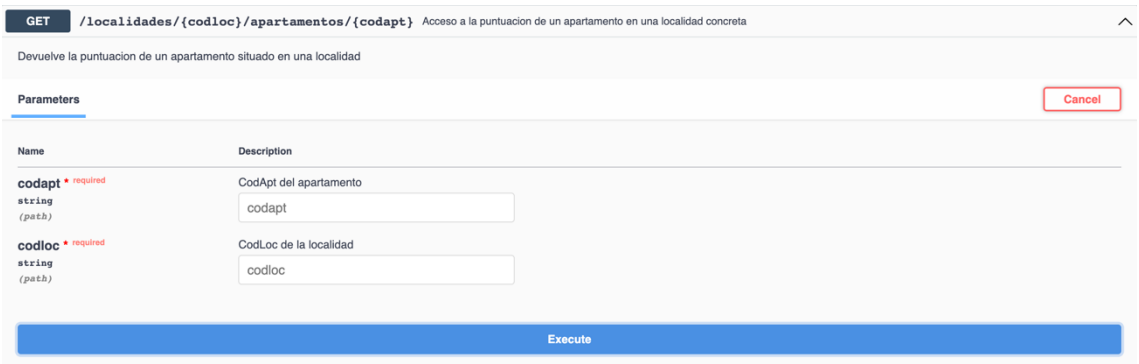

Figura 4-28 GET/localidades/{codloc}/apartamentos/{codapt}

El método HTTP empleado para esto es el GET y necesitará dos parámetros del tipo PathParam que nos indique el codapt del apartamento y el codloc de la localidad que se desea consultar.

Para acceder a este recurso hemos utilizado el resource path http://localhost:80/localidades/{codloc}/apartamentos/{codapt}

Puede haber dos tipos de respuesta, que son el Status code 200 OK en el caso de que se encuentre la puntuación asignada al apartamento y su localidad que se ha de mostrar. Pero en el caso opuesto se responderá con un 404 Not Found, que indicará que no se encontraron objetos.

Por tanto, si se encuentra el apartamento obtendremos un objeto json de aspecto similar al siguiente.

Figura 4-29 Resultado obtenido GET/localidades/{codloc}/apartamentos/{codapt}

4.3.14 POST/localidades/{codloc}/apartamentos/{codapt}

El propósito de esta funcionalidad es la de registrar un apartamento en una localidad, registrando ambos mediante su clave primaria, es decir, el codloc y el codapt. Además, nos devolverá todos los datos de la relación al completo.

Figura 4-30 POST/localidades/{codloc}/apartamentos/{codapt}

El método HTTP empleado para esto es el POST y necesitará dos parámetros del tipo PathParam que nos indique el codapt del apartamento y el codloc de la localidad en la que se desea registrar. Al ser el solamente el registro no se le asignarán estrellas.

Para acceder a este recurso hemos utilizado el resource path http://localhost:80/localidades/{codloc}/apartamentos/{codapt}

En este caso puede haber dos tipos de respuesta. Estos son el Status code 201 Created en el caso de que se registre el apartamento en la localidad correctamente. Pero, también existirá la posibilidad de responder con un 500 Server Error. Esto se debe a que el usuario está definiendo el valor de los campos y esto puede conllevar algún error.

Por tanto, si se añade correctamente el apartamento obtendremos un objeto json de aspecto similar al siguiente.

Figura 4-31 Resultado obtenido POST/localidades/{codloc}/apartamentos/{codapt}

4.3.15 PUT/localidades/{codloc}/apartamentos/{codapt}

El propósito de esta funcionalidad será la de puntuar un apartamento que ya se ha registrado dentro de una localidad previamente.

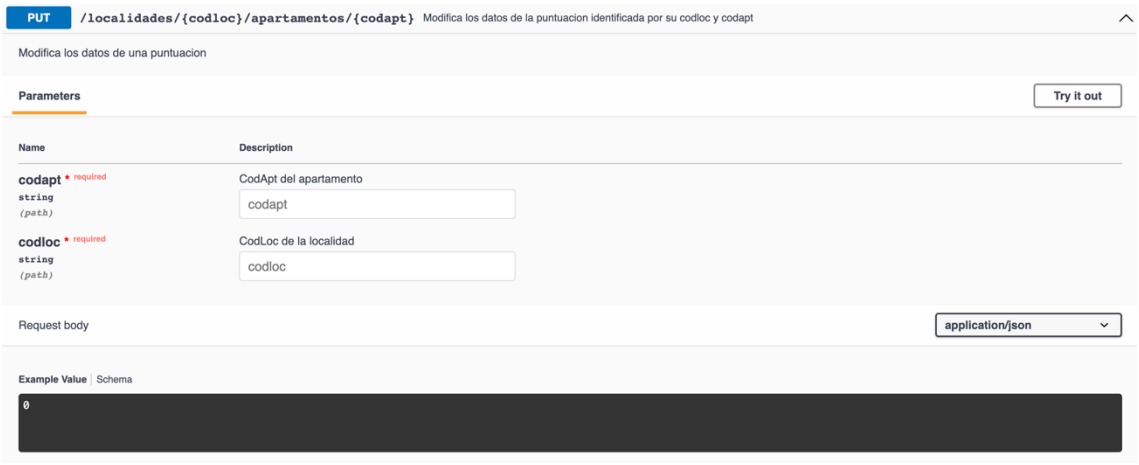

Figura 4-32 PUT/localidades/{codloc}/apartamentos/{codapt}

El método HTTP empleado para esto es el PUT y necesitará dos parámetros del tipo PathParam que nos indique el codapt del apartamento y el codloc de la localidad que se desea actualizar. Además, habrá que introducir las estrellas que se le asignarán al apartamento.

Para acceder a este recurso hemos utilizado el resource path http://localhost:80/localidades/{codloc}/apartamentos/{codapt}

Puede haber tres tipos de respuesta, que son el Status code 200 Ok en el caso de que se encuentre la puntuación que se ha de actualizar y se actualice. Pero en el caso opuesto se responderá con un 404 Not Found, que indicará que no se encontraron objetos. Pero, también existirá la posibilidad de responder con un 500 Server Error. Esto se debe a que el usuario está definiendo el valor de los campos y esto puede conllevar algún error.

Por tanto, si se actualiza correctamente la puntuación obtendremos un objeto json de aspecto similar al siguiente.

Figura 4-33 Resultado obtenido PUT/localidades/{codloc}/apartamentos/{codapt}

4.3.16 DELETE/localidades/{codloc}/apartamentos/{codapt}

El propósito de esta funcionalidad es la de eliminar una puntuación y el registro de un apartamento en una localidad que ha sido buscado en la base de datos mediante su codapt y su codloc.

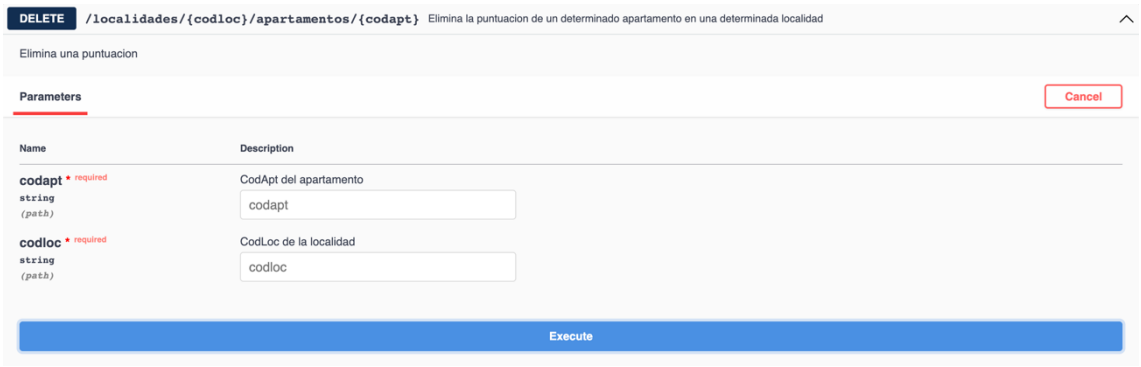

Figura 4-34 DELETE/localidades/{codloc}/apartamentos/{codapt}

El método HTTP empleado para esto es el DELETE y se necesitarán dos parámetros del tipo PathParam que nos indique el codapt del apartamento y el codloc de la localidad que se desea eliminar.

Para acceder a este recurso hemos utilizado el resource path http://localhost:80/localidades/{codloc}/apartamentos/{codloc}

Puede haber dos tipos de respuesta, que son el Status code 204 No Content en el caso de que se encuentre el apartamento que se ha de eliminar y se haga correctamente. Pero en el caso opuesto se responderá con un 404 Not Found, que indicará que no se encontraron objetos.

5. Resultados obtenidos.

En este apartado, se llevará a cabo una comparación de rendimiento y eficacia entre las dos aplicaciones creadas mediante las dos metodologías, una desarrollada utilizando el método reactivo y otra mediante un enfoque tradicional.

El objetivo principal de esta comparativa es evaluar cómo se comportan estas dos aplicaciones al manejar una carga de trabajo intensiva, donde se lanzan múltiples hilos y se realizan numerosas peticiones. Para ello, se creará un programa que simulará clientes que generen este tipo de carga de trabajo.

Este programa también estará desarrollado en Java y se encargará de crear una gran cantidad de hilos de ejecución, dentro de cada cual habrá muchas solicitudes. Para ello, se conectará a la url del recurso que se quiera medir y se indicará el tipo de solicitud a enviar, es decir, GET, POST, PUT o DELETE. Además, en caso de que se necesite el paso de parámetros estos serán proporcionados mediante Json, ya que las aplicaciones están diseñadas para recibir este tipo de parámetros a la hora de realizar las operaciones. Una vez hecho esto, se medirán los tiempos de respuesta de los diferentes métodos, proporcionando también el tiempo mínimo y máximo del conjunto de solicitudes de cada cliente.

Se analizará la comparativa entre los diferentes recursos habilitados en las aplicaciones, pero no solo se comparará una con otra para ver las diferencias entre modelos, sino que también se tendrá en cuenta el caché implementado. Porque, hay que recordar, que también mejora la eficiencia y la velocidad. Es por esto por lo que se tendrán en cuenta ambas aplicaciones y, además, el tiempo sin caché.

Para poder realizar dichas comparaciones las aplicaciones serán desplegadas mediante docker-compose siguiendo las especificaciones mencionadas en anteriores apartados. Por otro lado, el programa destinado a la medición de los recursos será desplegado en un portátil MacBook Air de 13,3 pulgadas, con un almacenamiento de 128 GB y 8 GB de memoria RAM del tipo SSD. En lo que al procesador se refiere, el portátil cuenta con un Intel Core i5-8210Y con 2 núcleos y una velocidad de 1,6 GHz. Respecto a la tarjeta gráfica cuenta con una Intel UHD Graphics 617.

Por tanto, como se podrá observar en las siguientes figuras la línea general que se sigue, salvo en algún caso muy específico, será la de una mayor velocidad en la implementación reactiva frente a la tradicional. Además, en los recursos GET es todavía más notable si se comparan en las que se utiliza caché, ya que se ahorra la consulta a la base de datos y mejora todavía más el rendimiento.

Respecto a los demás recursos en el que se utilicen otros métodos, como lo son el POST, PUT y DELETE, hay veces que la velocidad de respuesta se verá un poco afectada por el caché, ya que

en este caso no nos aporta la información, sino que tiene que registrarla. Esto supone una pequeña pérdida de tiempo, aunque no sea tanta como un registro con operaciones de E/S. Aun así, sigue siendo conveniente su uso, ya que el porcentaje de solicitudes mediante el método GET suele ser mayor.

En resumen, el desarrollo de aplicaciones mediante el método reactivo es más eficiente y rápido frente al tradicional en todos los recursos. Pero se puede mejorar todavía más mediante la implementación de caché.

5.1 Comparación GET/localidades

```
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=166.86666666666667, Tiempo mínimo=11, Tiempo maximo=432
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=166.4, Tiempo mínimo=38, Tiempo maximo=344
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=163.53333333333333, Tiempo mínimo=38, Tiempo maximo=350
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=172.0, Tiempo mínimo=33, Tiempo maximo=336
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=173.6, Tiempo mínimo=71, Tiempo maximo=417
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=177.33333333333334, Tiempo mínimo=29, Tiempo maximo=421
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=185.2, Tiempo mínimo=25, Tiempo maximo=345
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=185.46666666666667, Tiempo mínimo=23, Tiempo maximo=381
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=186.0, Tiempo mínimo=32, Tiempo maximo=387
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=189.26666666666668, Tiempo mínimo=29, Tiempo maximo=403
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=191.46666666666667, Tiempo mínimo=65, Tiempo maximo=353
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=193.73333333333332, Tiempo mínimo=21, Tiempo maximo=456
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=194.13333333333333, Tiempo mínimo=17, Tiempo maximo=390
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=195.26666666666668, Tiempo mínimo=32, Tiempo maximo=308
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=195.53333333333333, Tiempo mínimo=47, Tiempo maximo=359
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=196.73333333333332, Tiempo mínimo=50, Tiempo maximo=390
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=196.8, Tiempo mínimo=67, Tiempo maximo=445
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=200.53333333333333, Tiempo mínimo=40, Tiempo maximo=405
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=201.46666666666667, Tiempo mínimo=45, Tiempo maximo=444
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=202.4, Tiempo mínimo=22, Tiempo maximo=391
```

Figura 5-1 Tiempos GET/localidades con implementación reactiva y caché

```
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=237.86666666666667, Tiempo mínimo=45, Tiempo maximo=716
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=248.13333333333333, Tiempo mínimo=21, Tiempo maximo=593
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=266.73333333333335, Tiempo mínimo=39, Tiempo maximo=569
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=277.8666666666667, Tiempo mínimo=59, Tiempo maximo=581
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=284.3333333333333, Tiempo mínimo=101, Tiempo maximo=505
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=286.73333333333335, Tiempo mínimo=123, Tiempo maximo=594
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=288.1333333333333, Tiempo mínimo=32, Tiempo maximo=601
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=294.06666666666666, Tiempo mínimo=32, Tiempo maximo=555
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=296.3333333333333, Tiempo mínimo=23, Tiempo maximo=727
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=302.1333333333333, Tiempo mínimo=19, Tiempo maximo=696
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=306.53333333333336, Tiempo mínimo=21, Tiempo maximo=789
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=306.3333333333333, Tiempo mínimo=50, Tiempo maximo=692
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=309.2, Tiempo mínimo=15, Tiempo maximo=605
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=318.2, Tiempo mínimo=16, Tiempo maximo=632
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=318.6666666666667, Tiempo mínimo=22, Tiempo maximo=636
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=320.4, Tiempo mínimo=23, Tiempo maximo=540
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=323.46666666666664, Tiempo mínimo=29, Tiempo maximo=818
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=324.4, Tiempo mínimo=20, Tiempo maximo=550
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=327.53333333333336, Tiempo mínimo=19, Tiempo maximo=894
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=327.53333333333336, Tiempo mínimo=33, Tiempo maximo=674
```

Figura 5-2 Tiempos GET/localidades con implementación tradicional y caché

```
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=250.86666666666667, Tiempo mínimo=43, Tiempo máximo=504
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=284.6666666666667, Tiempo mínimo=28, Tiempo máximo=570
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=288.53333333333336, Tiempo mínimo=60, Tiempo máximo=559
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=300.0, Tiempo mínimo=19, Tiempo máximo=616
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=306.73333333333335, Tiempo mínimo=34, Tiempo máximo=626
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=309.8666666666667, Tiempo mínimo=22, Tiempo máximo=662
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=320.4, Tiempo mínimo=31, Tiempo máximo=578
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=322.73333333333335, Tiempo mínimo=45, Tiempo máximo=598
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=326.46666666666664, Tiempo mínimo=76, Tiempo máximo=546
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=326.6, Tiempo mínimo=113, Tiempo máximo=621
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=331.06666666666666, Tiempo mínimo=59, Tiempo máximo=621
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=331.0, Tiempo mínimo=26, Tiempo máximo=817
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=328.73333333333335, Tiempo mínimo=57, Tiempo máximo=685
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=333.93333333333334, Tiempo mínimo=99, Tiempo máximo=683
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=333.93333333333334, Tiempo mínimo=142, Tiempo máximo=605
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=334.1333333333333, Tiempo mínimo=96, Tiempo máximo=733
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=334.6666666666667, Tiempo mínimo=85, Tiempo máximo=683
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=333.8, Tiempo mínimo=51, Tiempo máximo=660
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=339.26666666666665, Tiempo mínimo=67, Tiempo máximo=746
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=337.46666666666664, Tiempo mínimo=16, Tiempo máximo=743
```

Figura 5-3 Tiempos GET/localidades con implementación reactiva y sin caché

```
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=519.8666666666667, Tiempo mínimo=108, Tiempo máximo=3713
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=533.0666666666667, Tiempo mínimo=86, Tiempo máximo=3699
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=568.0666666666667, Tiempo mínimo=94, Tiempo máximo=3728
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=567.8666666666667, Tiempo mínimo=157, Tiempo máximo=3612
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=569.6, Tiempo mínimo=110, Tiempo máximo=3645
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=571.0, Tiempo mínimo=174, Tiempo máximo=3636
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=574.6666666666666, Tiempo mínimo=105, Tiempo máximo=3654
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=576.6666666666666, Tiempo mínimo=132, Tiempo máximo=3750
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=582.8666666666667, Tiempo mínimo=154, Tiempo máximo=3711
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=583.8666666666667, Tiempo mínimo=103, Tiempo máximo=3680
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=586.0666666666667, Tiempo mínimo=153, Tiempo máximo=3672
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=591.7333333333333, Tiempo mínimo=127, Tiempo máximo=3640
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=592.1333333333333, Tiempo mínimo=120, Tiempo máximo=3641
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=592.1333333333333, Tiempo mínimo=151, Tiempo máximo=3735
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=593.0666666666667, Tiempo mínimo=166, Tiempo máximo=3690
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=593.4, Tiempo mínimo=126, Tiempo máximo=3706
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=594.4666666666667, Tiempo mínimo=173, Tiempo máximo=3626
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=596.9333333333333, Tiempo mínimo=166, Tiempo máximo=3695
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=597.1333333333333, Tiempo mínimo=102, Tiempo máximo=3680
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=597.8666666666667, Tiempo mínimo=128, Tiempo máximo=3647
```

Figura 5-4 Tiempos GET/localidades con implementación tradicional y sin caché

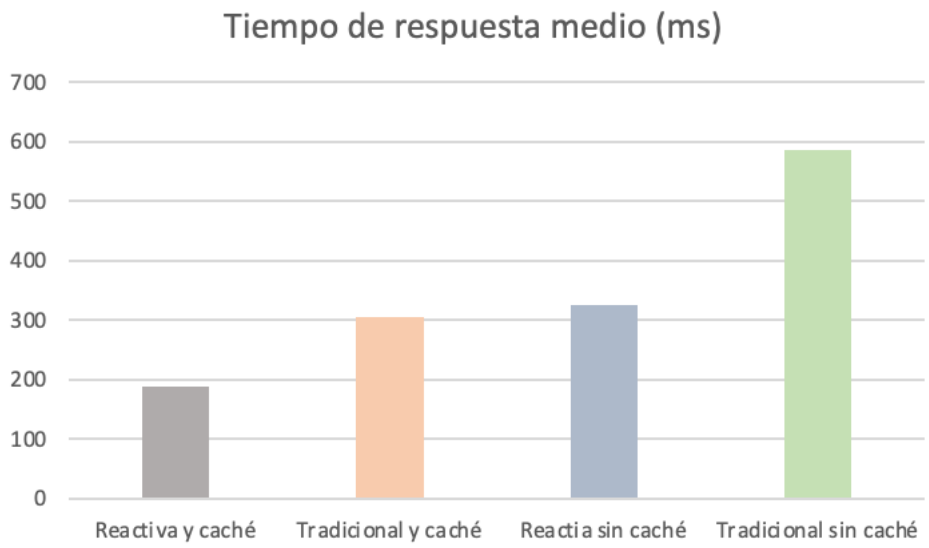

Figura 5-5 Gráfico de los tiempos de respuesta medios GET/localidades

Como se puede observar en el gráfico creado con los tiempos medios de las respuestas que otorgó el sistema en los diferentes formatos que se estudian, se puede concluir que el modelo reactivo es más rápido que el tradicional siempre que se compare en igualdad de condiciones. Esto significa que en las dos implementaciones con caché muestra un tiempo de respuesta menor el modelo reactivo, y en las dos implementaciones sin caché también. Además, la diferencia entra la implementación tradicional con caché y la reactiva sin caché es prácticamente insignificante, lo que nos muestra la potencia de la programación reactiva.

5.2 Comparación GET/localidades/{codloc}

```
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=189.46666666666667, Tiempo mínimo=75, Tiempo máximo=323
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=187.86666666666667, Tiempo mínimo=103, Tiempo máximo=284
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=197.2, Tiempo mínimo=59, Tiempo máximo=282
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=199.13333333333333, Tiempo mínimo=104, Tiempo máximo=355
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=203.4, Tiempo mínimo=84, Tiempo máximo=320
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=209.2, Tiempo mínimo=88, Tiempo máximo=387
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=211.26666666666668, Tiempo mínimo=102, Tiempo máximo=378
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=211.66666666666666, Tiempo mínimo=142, Tiempo máximo=303
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=213.13333333333333, Tiempo mínimo=103, Tiempo máximo=431
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=213.33333333333334, Tiempo mínimo=163, Tiempo máximo=292
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=215.8, Tiempo mínimo=65, Tiempo máximo=340
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=215.73333333333332, Tiempo mínimo=119, Tiempo máximo=305
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=218.0, Tiempo mínimo=17, Tiempo máximo=463
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=220.06666666666666, Tiempo mínimo=109, Tiempo máximo=336
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=220.46666666666667, Tiempo mínimo=151, Tiempo máximo=311
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=221.26666666666668, Tiempo mínimo=123, Tiempo máximo=442
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=221.93333333333334, Tiempo mínimo=92, Tiempo máximo=364
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=222.13333333333333, Tiempo mínimo=25, Tiempo máximo=386
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=222.33333333333334, Tiempo mínimo=31, Tiempo máximo=350
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=223.66666666666666, Tiempo mínimo=131, Tiempo máximo=405
```

Figura 5-6 Tiempos GET/localidades/{codloc} con implementación reactiva y caché

```
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=279.53333333333336, Tiempo mínimo=115, Tiempo máximo=585
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=294.6, Tiempo mínimo=85, Tiempo máximo=629
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=300.26666666666665, Tiempo mínimo=130, Tiempo máximo=582
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=301.73333333333335, Tiempo mínimo=124, Tiempo máximo=646
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=310.1333333333333, Tiempo mínimo=126, Tiempo máximo=569
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=312.06666666666666, Tiempo mínimo=83, Tiempo máximo=664
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=317.2, Tiempo mínimo=100, Tiempo máximo=700
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=317.6, Tiempo mínimo=91, Tiempo máximo=710
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=323.4, Tiempo mínimo=137, Tiempo máximo=699
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=323.6666666666667, Tiempo mínimo=110, Tiempo máximo=1015
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=324.6666666666667, Tiempo mínimo=114, Tiempo máximo=604
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=326.1333333333333, Tiempo mínimo=156, Tiempo máximo=682
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=328.2, Tiempo mínimo=138, Tiempo máximo=638
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=329.06666666666666, Tiempo mínimo=130, Tiempo máximo=551
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=330.73333333333335, Tiempo mínimo=91, Tiempo máximo=677
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=332.73333333333335, Tiempo mínimo=84, Tiempo máximo=662
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=332.8, Tiempo mínimo=124, Tiempo máximo=595
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=333.2, Tiempo mínimo=198, Tiempo máximo=899
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=334.6, Tiempo mínimo=65, Tiempo máximo=727
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=337.6666666666667, Tiempo mínimo=32, Tiempo máximo=633
```

Figura 5-7 Tiempos GET/localidades/{codloc} con implementación tradicional y caché

```
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=326.6666666666667, Tiempo mínimo=35, Tiempo máximo=1035
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=361.1333333333333, Tiempo mínimo=23, Tiempo máximo=1262
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=361.53333333333336, Tiempo mínimo=66, Tiempo máximo=1054
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=386.06666666666666, Tiempo mínimo=26, Tiempo máximo=1031
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=386.6666666666667, Tiempo mínimo=50, Tiempo máximo=960
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=403.6, Tiempo mínimo=34, Tiempo máximo=968
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=417.46666666666664, Tiempo mínimo=31, Tiempo máximo=971
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=421.93333333333334, Tiempo mínimo=22, Tiempo máximo=1336
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=423.06666666666666, Tiempo mínimo=46, Tiempo máximo=1054
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=427.4, Tiempo mínimo=51, Tiempo máximo=928
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=430.26666666666665, Tiempo mínimo=23, Tiempo máximo=1119
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=438.8, Tiempo mínimo=83, Tiempo máximo=1080
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=441.0, Tiempo mínimo=31, Tiempo máximo=1099
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=446.6666666666667, Tiempo mínimo=69, Tiempo máximo=1124
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=448.2, Tiempo mínimo=53, Tiempo máximo=1168
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=456.8, Tiempo mínimo=99, Tiempo máximo=1127
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=458.6, Tiempo mínimo=125, Tiempo máximo=1277
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=460.53333333333336, Tiempo mínimo=120, Tiempo máximo=1133
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=464.0, Tiempo mínimo=79, Tiempo máximo=1187
Cliente finalizado: URL=http://localhost:80/localidades/1, Peticiones=15, Tiempo promedio=464.06666666666666, Tiempo mínimo=93, Tiempo máximo=1138
```

Figura 5-8 Tiempos GET/localidades/{codloc} con implementación reactiva y sin caché

```
Cliente finalizado: URL=http://localhost:80/localidades/SA, Peticiones=15, Tiempo promedio=511.26666666666665, Tiempo mínimo=129, Tiempo máximo=1188
Cliente finalizado: URL=http://localhost:80/localidades/SA, Peticiones=15, Tiempo promedio=523.1333333333333, Tiempo mínimo=125, Tiempo máximo=1336
Cliente finalizado: URL=http://localhost:80/localidades/SA, Peticiones=15, Tiempo promedio=549.0, Tiempo mínimo=121, Tiempo máximo=1285
Cliente finalizado: URL=http://localhost:80/localidades/SA, Peticiones=15, Tiempo promedio=551.4, Tiempo mínimo=137, Tiempo máximo=1642
Cliente finalizado: URL=http://localhost:80/localidades/SA, Peticiones=15, Tiempo promedio=608.4, Tiempo mínimo=62, Tiempo máximo=1220
Cliente finalizado: URL=http://localhost:80/localidades/SA, Peticiones=15, Tiempo promedio=626.2666666666667, Tiempo mínimo=142, Tiempo máximo=1089
Cliente finalizado: URL=http://localhost:80/localidades/SA, Peticiones=15, Tiempo promedio=628.8666666666667, Tiempo mínimo=240, Tiempo máximo=1858
Cliente finalizado: URL=http://localhost:80/localidades/SA, Peticiones=15, Tiempo promedio=643.4666666666667, Tiempo mínimo=142, Tiempo máximo=1720
Cliente finalizado: URL=http://localhost:80/localidades/SA, Peticiones=15, Tiempo promedio=644.3333333333334, Tiempo mínimo=220, Tiempo máximo=1873
Cliente finalizado: URL=http://localhost:80/localidades/SA, Peticiones=15, Tiempo promedio=645.2666666666667, Tiempo mínimo=137, Tiempo máximo=1668
Cliente finalizado: URL=http://localhost:80/localidades/SA, Peticiones=15, Tiempo promedio=646.8, Tiempo mínimo=111, Tiempo máximo=1460
Cliente finalizado: URL=http://localhost:80/localidades/SA, Peticiones=15, Tiempo promedio=648.6666666666666, Tiempo mínimo=57, Tiempo máximo=1633
Cliente finalizado: URL=http://localhost:80/localidades/SA, Peticiones=15, Tiempo promedio=653.2666666666667, Tiempo mínimo=105, Tiempo máximo=1344
Cliente finalizado: URL=http://localhost:80/localidades/SA, Peticiones=15, Tiempo promedio=654.8666666666667, Tiempo mínimo=136, Tiempo máximo=1732
Cliente finalizado: URL=http://localhost:80/localidades/SA, Peticiones=15, Tiempo promedio=655.8, Tiempo mínimo=122, Tiempo máximo=1683
Cliente finalizado: URL=http://localhost:80/localidades/SA, Peticiones=15, Tiempo promedio=661.8666666666667, Tiempo mínimo=232, Tiempo máximo=1362
Cliente finalizado: URL=http://localhost:80/localidades/SA, Peticiones=15, Tiempo promedio=664.5333333333333, Tiempo mínimo=116, Tiempo máximo=1461
Cliente finalizado: URL=http://localhost:80/localidades/SA, Peticiones=15, Tiempo promedio=667.6666666666666, Tiempo mínimo=81, Tiempo máximo=1433
Cliente finalizado: URL=http://localhost:80/localidades/SA, Peticiones=15, Tiempo promedio=668.6666666666666, Tiempo mínimo=73, Tiempo máximo=1843
Cliente finalizado: URL=http://localhost:80/localidades/SA, Peticiones=15, Tiempo promedio=672.0666666666667, Tiempo mínimo=34, Tiempo máximo=1814
```

Figura 5-9 Tiempos GET/localidades/{codloc} con implementación tradicional y sin caché

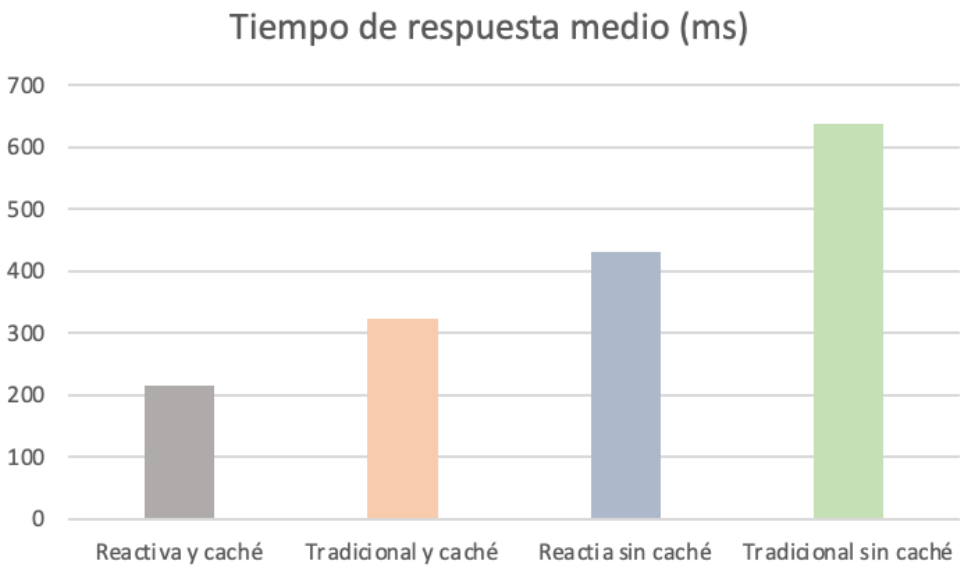

Figura 5-10 Gráfico de los tiempos de respuesta medios GET/localidades/{codloc}

Como se puede observar en el gráfico creado con los tiempos medios de las respuestas que otorgó el sistema en los diferentes formatos que se estudian, se puede concluir que el modelo reactivo es más rápido que el tradicional siempre que se compare en igualdad de condiciones. Esto significa que en las dos implementaciones con caché muestra un tiempo de respuesta menor el modelo reactivo, y en las dos implementaciones sin caché también. Además, se puede ver cómo, en este caso, los tiempos van aumentando de manera proporcional a medida que dejamos de utilizar elementos que ayudan a mejorar los tiempos de respuesta del sistema.

5.3 Comparación POST/localidades

```
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=410.3333333333333, Tiempo mínimo=53, Tiempo máximo=1731
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=423.73333333333335, Tiempo mínimo=141, Tiempo máximo=1211
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=428.06666666666666, Tiempo mínimo=184, Tiempo máximo=1666
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=437.8666666666667, Tiempo mínimo=77, Tiempo máximo=1806
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=443.6, Tiempo mínimo=82, Tiempo máximo=1650
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=443.3333333333333, Tiempo mínimo=20, Tiempo máximo=2513
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=463.0, Tiempo mínimo=83, Tiempo máximo=2276
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=463.53333333333336, Tiempo mínimo=125, Tiempo máximo=2447
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=463.26666666666665, Tiempo mínimo=64, Tiempo máximo=2824
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=463.8, Tiempo mínimo=151, Tiempo máximo=1631
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=465.1333333333333, Tiempo mínimo=94, Tiempo máximo=2501
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=465.73333333333335, Tiempo mínimo=68, Tiempo máximo=2778
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=466.6, Tiempo mínimo=40, Tiempo máximo=2805
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=466.6, Tiempo mínimo=102, Tiempo máximo=2629
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=470.3333333333333, Tiempo mínimo=60, Tiempo máximo=2811
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=470.3333333333333, Tiempo mínimo=74, Tiempo máximo=2644
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=471.26666666666665, Tiempo mínimo=37, Tiempo máximo=2648
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=471.26666666666665, Tiempo mínimo=59, Tiempo máximo=2347
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=473.1333333333333, Tiempo mínimo=53, Tiempo máximo=2823
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=473.1333333333333, Tiempo mínimo=49, Tiempo máximo=2348
```

Figura 5-11 Tiempos POST/localidades con implementación reactiva y caché

```
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=995.0666666666667, Tiempo mínimo=353, Tiempo máximo=4204
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=1014.1333333333333, Tiempo mínimo=381, Tiempo máximo=4135
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=1025.7333333333333, Tiempo mínimo=172, Tiempo máximo=4233
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=1026.0, Tiempo mínimo=245, Tiempo máximo=4201
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=1028.5333333333333, Tiempo mínimo=492, Tiempo máximo=4252
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=1029.0, Tiempo mínimo=336, Tiempo máximo=4116
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=1043.2666666666667, Tiempo mínimo=445, Tiempo máximo=4145
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=1043.3333333333333, Tiempo mínimo=600, Tiempo máximo=4371
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=1052.4, Tiempo mínimo=250, Tiempo máximo=4195
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=1053.5333333333333, Tiempo mínimo=435, Tiempo máximo=4360
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=1054.1333333333334, Tiempo mínimo=283, Tiempo máximo=4394
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=1055.0, Tiempo mínimo=303, Tiempo máximo=4459
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=1055.2, Tiempo mínimo=388, Tiempo máximo=4153
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=1055.3333333333333, Tiempo mínimo=292, Tiempo máximo=4125
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=1056.5333333333333, Tiempo mínimo=237, Tiempo máximo=4161
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=1058.8, Tiempo mínimo=473, Tiempo máximo=4334
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=1058.9333333333334, Tiempo mínimo=461, Tiempo máximo=4197
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=1059.8, Tiempo mínimo=347, Tiempo máximo=4153
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=1070.5333333333333, Tiempo mínimo=163, Tiempo máximo=4483
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=1074.0, Tiempo mínimo=50, Tiempo máximo=4435
```

Figura 5-12 Tiempos POST/localidades con implementación tradicional y caché

```
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=261.53333333333336, Tiempo mínimo=61, Tiempo máximo=556
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=278.2, Tiempo mínimo=59, Tiempo máximo=430
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=289.2, Tiempo mínimo=107, Tiempo máximo=783
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=289.4, Tiempo mínimo=108, Tiempo máximo=735
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=293.1333333333333, Tiempo mínimo=59, Tiempo máximo=959
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=294.2, Tiempo mínimo=141, Tiempo máximo=503
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=296.0, Tiempo mínimo=53, Tiempo máximo=871
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=296.53333333333336, Tiempo mínimo=83, Tiempo máximo=696
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=299.8666666666667, Tiempo mínimo=66, Tiempo máximo=633
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=299.53333333333336, Tiempo mínimo=45, Tiempo máximo=1011
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=299.9333333333334, Tiempo mínimo=165, Tiempo máximo=563
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=300.6, Tiempo mínimo=93, Tiempo máximo=575
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=300.53333333333336, Tiempo mínimo=71, Tiempo máximo=736
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=301.73333333333335, Tiempo mínimo=148, Tiempo máximo=718
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=302.06666666666666, Tiempo mínimo=60, Tiempo máximo=747
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=304.2, Tiempo mínimo=58, Tiempo máximo=784
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=304.6, Tiempo mínimo=88, Tiempo máximo=486
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=305.6666666666667, Tiempo mínimo=36, Tiempo máximo=1065
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=306.3333333333333, Tiempo mínimo=60, Tiempo máximo=574
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=306.0, Tiempo mínimo=33, Tiempo máximo=621
```

Figura 5-13 Tiempos POST/localidades con implementación reactiva y sin caché

```
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=494.3333333333333, Tiempo mínimo=160, Tiempo máximo=1082
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=487.26666666666665, Tiempo mínimo=108, Tiempo máximo=967
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=499.8, Tiempo mínimo=105, Tiempo máximo=1246
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=501.26666666666665, Tiempo mínimo=146, Tiempo máximo=1292
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=501.53333333333336, Tiempo mínimo=159, Tiempo máximo=1443
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=502.3333333333333, Tiempo mínimo=152, Tiempo máximo=1225
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=505.3333333333333, Tiempo mínimo=112, Tiempo máximo=1156
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=509.1333333333333, Tiempo mínimo=130, Tiempo máximo=1182
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=511.0, Tiempo mínimo=205, Tiempo máximo=1010
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=511.4, Tiempo mínimo=151, Tiempo máximo=1037
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=512.5333333333333, Tiempo mínimo=103, Tiempo máximo=1006
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=514.7333333333333, Tiempo mínimo=89, Tiempo máximo=1571
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=516.4, Tiempo mínimo=170, Tiempo máximo=1150
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=516.6, Tiempo mínimo=164, Tiempo máximo=1126
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=518.1333333333333, Tiempo mínimo=177, Tiempo máximo=1107
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=519.7333333333333, Tiempo mínimo=81, Tiempo máximo=1105
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=520.6, Tiempo mínimo=174, Tiempo máximo=1102
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=522.1333333333333, Tiempo mínimo=95, Tiempo máximo=1089
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=522.6666666666666, Tiempo mínimo=66, Tiempo máximo=1469
Cliente finalizado: URL=http://localhost:80/localidades, Peticiones=15, Tiempo promedio=522.7333333333333, Tiempo mínimo=37, Tiempo máximo=1337
```

Figura 5-14 Tiempos POST/localidades con implementación tradicional y sin caché

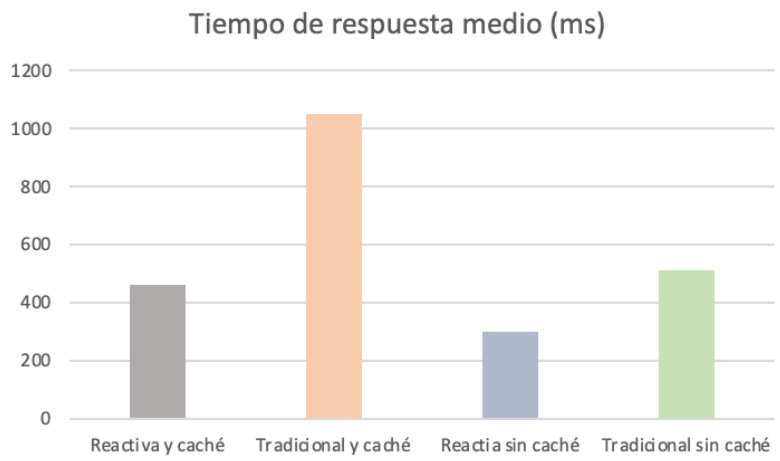

Figura 5-15 Gráfico tiempos de respuesta medios POST/localidades

Como se puede observar en el gráfico creado con los tiempos medios de las respuestas que otorgó el sistema en los diferentes formatos que se estudian, se puede concluir que el modelo reactivo es más rápido que el tradicional siempre que se compare en igualdad de condiciones. Esto significa que en las dos implementaciones con caché muestra un tiempo de respuesta menor el modelo reactivo, y en las dos implementaciones sin caché también. Además, se puede ver cómo, al ser un método POST, se obtiene un tiempo de respuesta menor en los modelos sin caché ya que con caché no se brinda un acceso más rápido a los datos, sino que hay que almacenarlos, por tanto, se pierde un poco más de tiempo.

Aun así, el porcentaje de solicitudes de este estilo es bastante menor con el de solicitudes GET, por lo que en el sistema en general sigue siendo beneficioso el uso de caché.

5.4 Comparación PUT/localidades

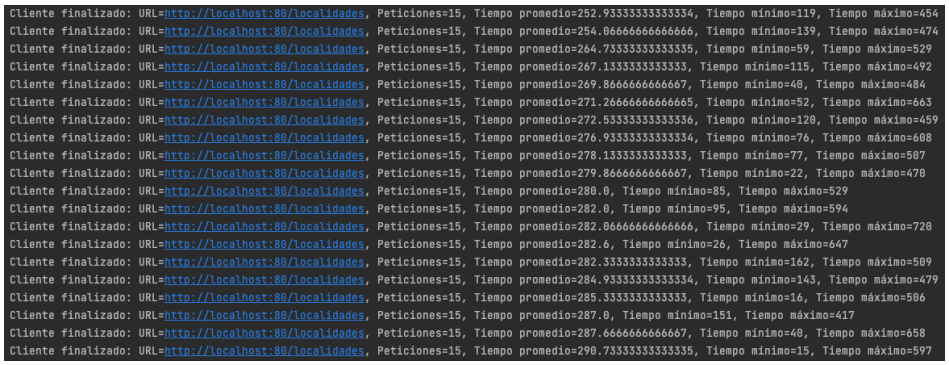

Figura 5-16 Tiempos PUT/localidades con implementación reactiva y caché

Figura 5-17 Tiempos PUT/localidades con implementación tradicional y caché

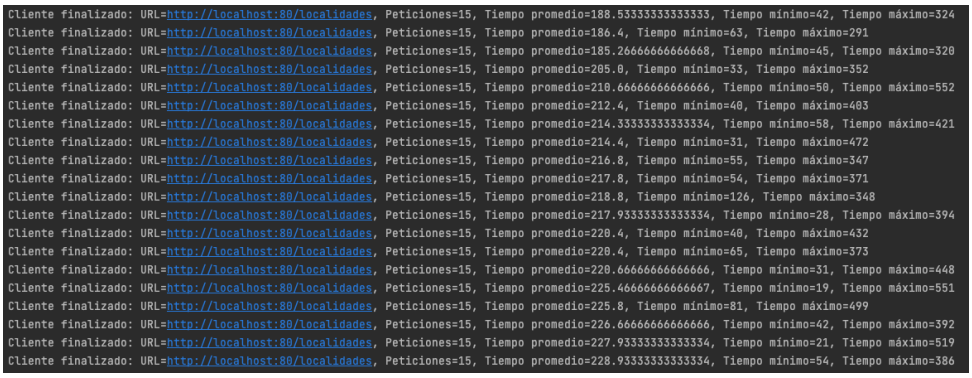

Figura 5-18 Tiempos PUT/localidades con implementación reactiva y sin caché

```
Cliente finalizado: URL=http://localhost:80/localidades/87, Peticiones=15, Tiempo promedio=205.13333333333333, Tiempo mínimo=79, Tiempo máximo=440
Cliente finalizado: URL=http://localhost:80/localidades/17, Peticiones=15, Tiempo promedio=283.1333333333333, Tiempo mínimo=62, Tiempo máximo=648
Cliente finalizado: URL=http://localhost:80/localidades/27, Peticiones=15, Tiempo promedio=310.6666666666667, Tiempo mínimo=89, Tiempo máximo=619
Cliente finalizado: URL=http://localhost:80/localidades/147, Peticiones=15, Tiempo promedio=339.73333333333335, Tiempo mínimo=109, Tiempo máximo=928
Cliente finalizado: URL=http://localhost:80/localidades/87, Peticiones=15, Tiempo promedio=365.26666666666665, Tiempo mínimo=26, Tiempo máximo=2730
Cliente finalizado: URL=http://localhost:80/localidades/177, Peticiones=15, Tiempo promedio=378.0, Tiempo mínimo=177, Tiempo máximo=685
Cliente finalizado: URL=http://localhost:80/localidades/167, Peticiones=15, Tiempo promedio=378.0, Tiempo mínimo=142, Tiempo máximo=1515
Cliente finalizado: URL=http://localhost:80/localidades/37, Peticiones=15, Tiempo promedio=387.3333333333333, Tiempo mínimo=81, Tiempo máximo=1897
Cliente finalizado: URL=http://localhost:80/localidades/157, Peticiones=15, Tiempo promedio=402.53333333333336, Tiempo mínimo=84, Tiempo máximo=2103
Cliente finalizado: URL=http://localhost:80/localidades/107, Peticiones=15, Tiempo promedio=409.93333333333334, Tiempo mínimo=134, Tiempo máximo=1932
Cliente finalizado: URL=http://localhost:80/localidades/127, Peticiones=15, Tiempo promedio=425.6666666666667, Tiempo mínimo=70, Tiempo máximo=2906
Cliente finalizado: URL=http://localhost:80/localidades/77, Peticiones=15, Tiempo promedio=448.3333333333333, Tiempo mínimo=94, Tiempo máximo=3888
Cliente finalizado: URL=http://localhost:80/localidades/187, Peticiones=15, Tiempo promedio=452.46666666666664, Tiempo mínimo=91, Tiempo máximo=3397
Cliente finalizado: URL=http://localhost:80/localidades/57, Peticiones=15, Tiempo promedio=457.4, Tiempo mínimo=61, Tiempo máximo=4182
Cliente finalizado: URL=http://localhost:80/localidades/197, Peticiones=15, Tiempo promedio=461.6, Tiempo mínimo=71, Tiempo máximo=4764
Cliente finalizado: URL=http://localhost:80/localidades/67, Peticiones=15, Tiempo promedio=466.2, Tiempo mínimo=30, Tiempo máximo=4769
Cliente finalizado: URL=http://localhost:80/localidades/47, Peticiones=15, Tiempo promedio=470.4, Tiempo mínimo=54, Tiempo máximo=4465
Cliente finalizado: URL=http://localhost:80/localidades/137, Peticiones=15, Tiempo promedio=470.46666666666664, Tiempo mínimo=38, Tiempo máximo=5170
Cliente finalizado: URL=http://localhost:80/localidades/117, Peticiones=15, Tiempo promedio=473.26666666666665, Tiempo mínimo=42, Tiempo máximo=4936
Cliente finalizado: URL=http://localhost:80/localidades/97, Peticiones=15, Tiempo promedio=476.53333333333336, Tiempo mínimo=19, Tiempo máximo=5416
```

Figura 5-19 Tiempos PUT/localidades con implementación tradicional y sin caché

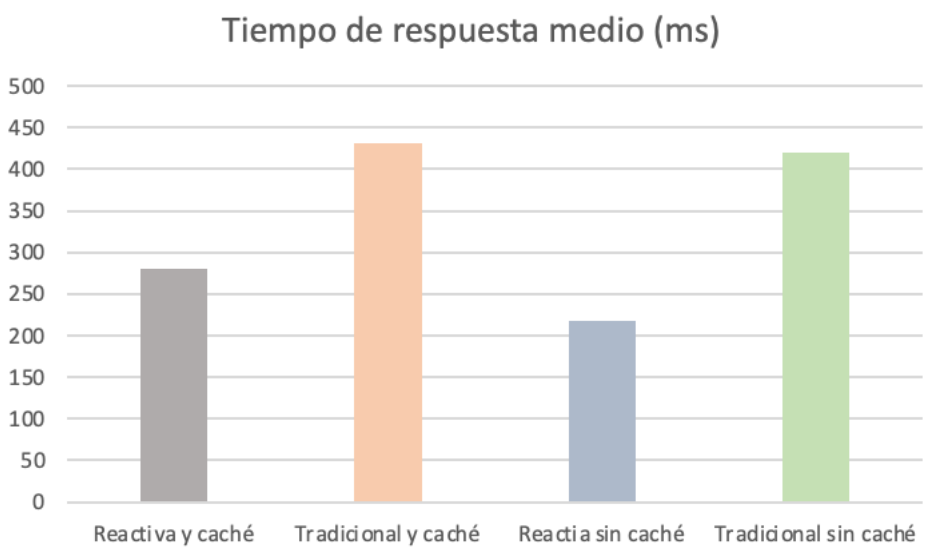

Figura 5-20 Gráfico tiempos medios PUT/localidades

Como se puede observar en el gráfico creado con los tiempos medios de las respuestas que otorgó el sistema en los diferentes formatos que se estudian, se puede concluir que el modelo reactivo es más rápido que el tradicional siempre que se compare en igualdad de condiciones. Esto significa que en las dos implementaciones con caché muestra un tiempo de respuesta menor el modelo reactivo, y en las dos implementaciones sin caché también. Además, se puede ver cómo, al ser un método PUT, se obtiene un tiempo de respuesta menor en los modelos sin caché ya que con caché no se brinda un acceso más rápido a los datos, sino que hay que modificar los que ya se encuentran disponibles, por tanto, se pierde un poco más de tiempo.

Aun así, el porcentaje de solicitudes de este estilo es bastante menor con el de solicitudes GET, por lo que en el sistema en general sigue siendo beneficioso el uso de caché.

5.5 Comparación DELETE/localidades/{codloc}

```
Cliente finalizado: URL=http://localhost:80/localidades/117, Peticiones=15, Tiempo promedio=230.2, Tiempo mínimo=57, Tiempo máximo=502
Cliente finalizado: URL=http://localhost:80/localidades/147, Peticiones=15, Tiempo promedio=236.13333333333333, Tiempo mínimo=45, Tiempo máximo=598
Cliente finalizado: URL=http://localhost:80/localidades/77, Peticiones=15, Tiempo promedio=252.8, Tiempo mínimo=36, Tiempo máximo=511
Cliente finalizado: URL=http://localhost:80/localidades/157, Peticiones=15, Tiempo promedio=264.6, Tiempo mínimo=31, Tiempo máximo=611
Cliente finalizado: URL=http://localhost:80/localidades/47, Peticiones=15, Tiempo promedio=276.46666666666664, Tiempo mínimo=36, Tiempo máximo=577
Cliente finalizado: URL=http://localhost:80/localidades/17, Peticiones=15, Tiempo promedio=282.46666666666664, Tiempo mínimo=117, Tiempo máximo=550
Cliente finalizado: URL=http://localhost:80/localidades/97, Peticiones=15, Tiempo promedio=285.2, Tiempo mínimo=70, Tiempo máximo=530
Cliente finalizado: URL=http://localhost:80/localidades/137, Peticiones=15, Tiempo promedio=292.3333333333333, Tiempo mínimo=80, Tiempo máximo=516
Cliente finalizado: URL=http://localhost:80/localidades/37, Peticiones=15, Tiempo promedio=300.1333333333333, Tiempo mínimo=70, Tiempo máximo=489
Cliente finalizado: URL=http://localhost:80/localidades/87, Peticiones=15, Tiempo promedio=311.06666666666666, Tiempo mínimo=46, Tiempo máximo=563
Cliente finalizado: URL=http://localhost:80/localidades/67, Peticiones=15, Tiempo promedio=310.26666666666665, Tiempo mínimo=48, Tiempo máximo=522
Cliente finalizado: URL=http://localhost:80/localidades/27, Peticiones=15, Tiempo promedio=318.06666666666666, Tiempo mínimo=53, Tiempo máximo=586
Cliente finalizado: URL=http://localhost:80/localidades/87, Peticiones=15, Tiempo promedio=322.6, Tiempo mínimo=63, Tiempo máximo=560
Cliente finalizado: URL=http://localhost:80/localidades/127, Peticiones=15, Tiempo promedio=326.6666666666667, Tiempo mínimo=30, Tiempo máximo=624
Cliente finalizado: URL=http://localhost:80/localidades/197, Peticiones=15, Tiempo promedio=328.26666666666665, Tiempo mínimo=78, Tiempo máximo=627
Cliente finalizado: URL=http://localhost:80/localidades/107, Peticiones=15, Tiempo promedio=336.2, Tiempo mínimo=32, Tiempo máximo=591
Cliente finalizado: URL=http://localhost:80/localidades/177, Peticiones=15, Tiempo promedio=337.46666666666664, Tiempo mínimo=59, Tiempo máximo=679
Cliente finalizado: URL=http://localhost:80/localidades/57, Peticiones=15, Tiempo promedio=339.06666666666666, Tiempo mínimo=63, Tiempo máximo=632
Cliente finalizado: URL=http://localhost:80/localidades/187, Peticiones=15, Tiempo promedio=340.6666666666667, Tiempo mínimo=23, Tiempo máximo=686
Cliente finalizado: URL=http://localhost:80/localidades/167, Peticiones=15, Tiempo promedio=342.1333333333333, Tiempo mínimo=17, Tiempo máximo=636
```

Figura 5-21 Tiempos DELETE/localidades/{codloc} con implementación reactiva y caché

```
Cliente finalizado: URL=http://localhost:80/localidades/41, Peticiones=15, Tiempo promedio=269.06666666666666, Tiempo mínimo=50, Tiempo máximo=1133
Cliente finalizado: URL=http://localhost:80/localidades/81, Peticiones=15, Tiempo promedio=288.0, Tiempo mínimo=53, Tiempo máximo=1727
Cliente finalizado: URL=http://localhost:80/localidades/11, Peticiones=15, Tiempo promedio=311.26666666666665, Tiempo mínimo=75, Tiempo máximo=1154
Cliente finalizado: URL=http://localhost:80/localidades/151, Peticiones=15, Tiempo promedio=314.93333333333334, Tiempo mínimo=147, Tiempo máximo=1030
Cliente finalizado: URL=http://localhost:80/localidades/191, Peticiones=15, Tiempo promedio=333.3333333333333, Tiempo mínimo=37, Tiempo máximo=1442
Cliente finalizado: URL=http://localhost:80/localidades/161, Peticiones=15, Tiempo promedio=335.3333333333333, Tiempo mínimo=116, Tiempo máximo=1793
Cliente finalizado: URL=http://localhost:80/localidades/31, Peticiones=15, Tiempo promedio=338.73333333333335, Tiempo mínimo=87, Tiempo máximo=1263
Cliente finalizado: URL=http://localhost:80/localidades/21, Peticiones=15, Tiempo promedio=343.93333333333334, Tiempo mínimo=65, Tiempo máximo=1158
Cliente finalizado: URL=http://localhost:80/localidades/71, Peticiones=15, Tiempo promedio=342.53333333333336, Tiempo mínimo=46, Tiempo máximo=2049
Cliente finalizado: URL=http://localhost:80/localidades/131, Peticiones=15, Tiempo promedio=342.73333333333335, Tiempo mínimo=93, Tiempo máximo=2261
Cliente finalizado: URL=http://localhost:80/localidades/181, Peticiones=15, Tiempo promedio=342.8, Tiempo mínimo=60, Tiempo máximo=1497
Cliente finalizado: URL=http://localhost:80/localidades/121, Peticiones=15, Tiempo promedio=346.53333333333336, Tiempo mínimo=46, Tiempo máximo=1772
Cliente finalizado: URL=http://localhost:80/localidades/141, Peticiones=15, Tiempo promedio=350.2, Tiempo mínimo=42, Tiempo máximo=2048
Cliente finalizado: URL=http://localhost:80/localidades/101, Peticiones=15, Tiempo promedio=354.8666666666667, Tiempo mínimo=42, Tiempo máximo=1426
Cliente finalizado: URL=http://localhost:80/localidades/51, Peticiones=15, Tiempo promedio=356.06666666666666, Tiempo mínimo=67, Tiempo máximo=2289
Cliente finalizado: URL=http://localhost:80/localidades/171, Peticiones=15, Tiempo promedio=355.6666666666667, Tiempo mínimo=79, Tiempo máximo=2278
Cliente finalizado: URL=http://localhost:80/localidades/111, Peticiones=15, Tiempo promedio=357.6, Tiempo mínimo=34, Tiempo máximo=2017
Cliente finalizado: URL=http://localhost:80/localidades/81, Peticiones=15, Tiempo promedio=358.1333333333333, Tiempo mínimo=25, Tiempo máximo=2337
Cliente finalizado: URL=http://localhost:80/localidades/91, Peticiones=15, Tiempo promedio=358.4, Tiempo mínimo=67, Tiempo máximo=1712
Cliente finalizado: URL=http://localhost:80/localidades/61, Peticiones=15, Tiempo promedio=358.8666666666667, Tiempo mínimo=47, Tiempo máximo=2323
```

Figura 5-22 Tiempos DELETE/localidades/{codloc} con implementación tradicional y caché

```
Cliente finalizado: URL=http://localhost:80/localidades/167, Peticiones=15, Tiempo promedio=207.0, Tiempo mínimo=50, Tiempo máximo=372
Cliente finalizado: URL=http://localhost:80/localidades/17, Peticiones=15, Tiempo promedio=222.06666666666666, Tiempo mínimo=23, Tiempo máximo=685
Cliente finalizado: URL=http://localhost:80/localidades/77, Peticiones=15, Tiempo promedio=273.26666666666665, Tiempo mínimo=114, Tiempo máximo=604
Cliente finalizado: URL=http://localhost:80/localidades/187, Peticiones=15, Tiempo promedio=270.4, Tiempo mínimo=34, Tiempo máximo=700
Cliente finalizado: URL=http://localhost:80/localidades/157, Peticiones=15, Tiempo promedio=273.26666666666665, Tiempo mínimo=56, Tiempo máximo=826
Cliente finalizado: URL=http://localhost:80/localidades/127, Peticiones=15, Tiempo promedio=276.93333333333334, Tiempo mínimo=92, Tiempo máximo=742
Cliente finalizado: URL=http://localhost:80/localidades/197, Peticiones=15, Tiempo promedio=278.8, Tiempo mínimo=38, Tiempo máximo=728
Cliente finalizado: URL=http://localhost:80/localidades/47, Peticiones=15, Tiempo promedio=290.3333333333333, Tiempo mínimo=76, Tiempo máximo=730
Cliente finalizado: URL=http://localhost:80/localidades/87, Peticiones=15, Tiempo promedio=299.73333333333335, Tiempo mínimo=41, Tiempo máximo=956
Cliente finalizado: URL=http://localhost:80/localidades/107, Peticiones=15, Tiempo promedio=311.6, Tiempo mínimo=79, Tiempo máximo=881
Cliente finalizado: URL=http://localhost:80/localidades/97, Peticiones=15, Tiempo promedio=316.4, Tiempo mínimo=47, Tiempo máximo=803
Cliente finalizado: URL=http://localhost:80/localidades/137, Peticiones=15, Tiempo promedio=319.53333333333336, Tiempo mínimo=65, Tiempo máximo=566
Cliente finalizado: URL=http://localhost:80/localidades/87, Peticiones=15, Tiempo promedio=320.3333333333333, Tiempo mínimo=119, Tiempo máximo=630
Cliente finalizado: URL=http://localhost:80/localidades/27, Peticiones=15, Tiempo promedio=322.26666666666665, Tiempo mínimo=38, Tiempo máximo=935
Cliente finalizado: URL=http://localhost:80/localidades/67, Peticiones=15, Tiempo promedio=322.73333333333335, Tiempo mínimo=78, Tiempo máximo=936
Cliente finalizado: URL=http://localhost:80/localidades/117, Peticiones=15, Tiempo promedio=322.2, Tiempo mínimo=59, Tiempo máximo=943
Cliente finalizado: URL=http://localhost:80/localidades/57, Peticiones=15, Tiempo promedio=323.53333333333336, Tiempo mínimo=109, Tiempo máximo=826
Cliente finalizado: URL=http://localhost:80/localidades/147, Peticiones=15, Tiempo promedio=323.6666666666667, Tiempo mínimo=48, Tiempo máximo=891
Cliente finalizado: URL=http://localhost:80/localidades/177, Peticiones=15, Tiempo promedio=324.06666666666666, Tiempo mínimo=25, Tiempo máximo=969
Cliente finalizado: URL=http://localhost:80/localidades/37, Peticiones=15, Tiempo promedio=326.26666666666665, Tiempo mínimo=24, Tiempo máximo=676
```

Figura 5-23 Tiempos DELETE/localidades/{codloc} con implementación reactiva y sin caché

```
Cliente finalizado: URL=http://localhost:80/localidades/111, Peticiones=15, Tiempo promedio=106.4, Tiempo mínimo=34, Tiempo máximo=349
Cliente finalizado: URL=http://localhost:80/localidades/141, Peticiones=15, Tiempo promedio=117.26666666666667, Tiempo mínimo=29, Tiempo máximo=409
Cliente finalizado: URL=http://localhost:80/localidades/81, Peticiones=15, Tiempo promedio=120.06666666666666, Tiempo mínimo=40, Tiempo máximo=357
Cliente finalizado: URL=http://localhost:80/localidades/31, Peticiones=15, Tiempo promedio=132.6, Tiempo mínimo=35, Tiempo máximo=344
Cliente finalizado: URL=http://localhost:80/localidades/91, Peticiones=15, Tiempo promedio=137.6, Tiempo mínimo=25, Tiempo máximo=767
Cliente finalizado: URL=http://localhost:80/localidades/101, Peticiones=15, Tiempo promedio=144.93333333333334, Tiempo mínimo=21, Tiempo máximo=383
Cliente finalizado: URL=http://localhost:80/localidades/191, Peticiones=15, Tiempo promedio=182.46666666666667, Tiempo mínimo=46, Tiempo máximo=1313
Cliente finalizado: URL=http://localhost:80/localidades/171, Peticiones=15, Tiempo promedio=222.53333333333333, Tiempo mínimo=44, Tiempo máximo=1062
Cliente finalizado: URL=http://localhost:80/localidades/51, Peticiones=15, Tiempo promedio=239.86666666666667, Tiempo mínimo=45, Tiempo máximo=1608
Cliente finalizado: URL=http://localhost:80/localidades/41, Peticiones=15, Tiempo promedio=245.6, Tiempo mínimo=61, Tiempo máximo=1950
Cliente finalizado: URL=http://localhost:80/localidades/151, Peticiones=15, Tiempo promedio=251.4, Tiempo mínimo=41, Tiempo máximo=2259
Cliente finalizado: URL=http://localhost:80/localidades/21, Peticiones=15, Tiempo promedio=254.66666666666666, Tiempo mínimo=22, Tiempo máximo=2250
Cliente finalizado: URL=http://localhost:80/localidades/161, Peticiones=15, Tiempo promedio=258.4, Tiempo mínimo=21, Tiempo máximo=2183
Cliente finalizado: URL=http://localhost:80/localidades/81, Peticiones=15, Tiempo promedio=264.73333333333335, Tiempo mínimo=34, Tiempo máximo=2408
Cliente finalizado: URL=http://localhost:80/localidades/121, Peticiones=15, Tiempo promedio=269.93333333333334, Tiempo mínimo=35, Tiempo máximo=2954
Cliente finalizado: URL=http://localhost:80/localidades/61, Peticiones=15, Tiempo promedio=269.6666666666667, Tiempo mínimo=32, Tiempo máximo=2634
Cliente finalizado: URL=http://localhost:80/localidades/181, Peticiones=15, Tiempo promedio=276.1333333333333, Tiempo mínimo=17, Tiempo máximo=2895
Cliente finalizado: URL=http://localhost:80/localidades/131, Peticiones=15, Tiempo promedio=288.73333333333335, Tiempo mínimo=33, Tiempo máximo=3097
Cliente finalizado: URL=http://localhost:80/localidades/71, Peticiones=15, Tiempo promedio=291.46666666666664, Tiempo mínimo=16, Tiempo máximo=3409
Cliente finalizado: URL=http://localhost:80/localidades/11, Peticiones=15, Tiempo promedio=295.26666666666665, Tiempo mínimo=9, Tiempo máximo=3593
```

Figura 5-24 Tiempos DELETE/localidades/{codloc} con implementación tradicional y sin caché

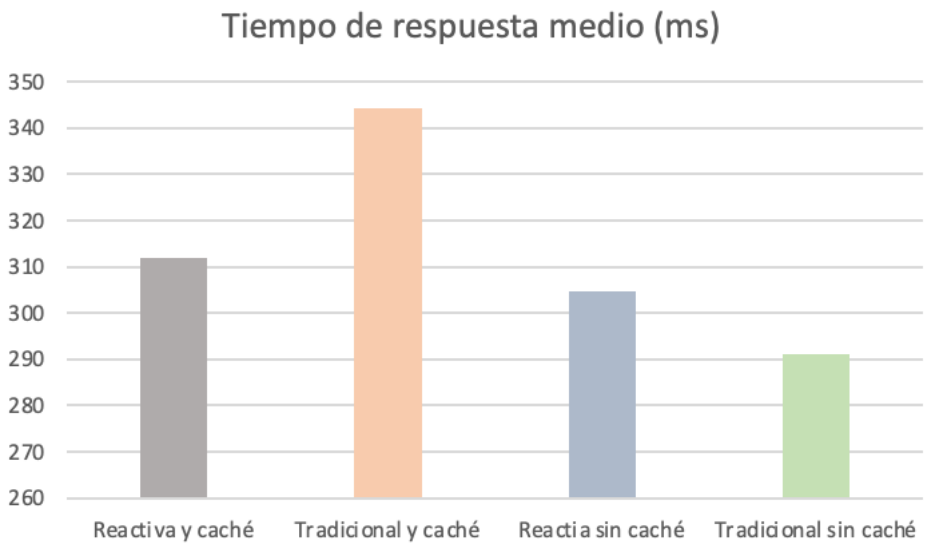

Figura 5-25 Gráfico tiempos de respuesta medios DELETE/localidades/{codloc}

Como se puede observar en el gráfico creado con los tiempos medios de las respuestas que otorgó el sistema en los diferentes formatos que se estudian, se puede concluir que el modelo reactivo es más rápido que el tradicional en la implementación con caché, sin embargo, en este caso concreto, en la implementación sin caché es un poco más rápido el tradicional. Además, se puede ver cómo, al ser un método DELETE, se obtiene un tiempo de respuesta menor en los modelos sin caché ya que con caché no se brinda un acceso más rápido a los datos, sino que hay que eliminar los que ya se encuentran disponibles, por tanto, se pierde un poco más de tiempo.

Aun así, el porcentaje de solicitudes de este estilo es bastante menor con el de solicitudes GET, por lo que en el sistema en general sigue siendo beneficioso el uso de caché.

5.6 Comparación GET/apartamentos

```
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=164.46666666666667, Tiempo minimo=49, Tiempo maximo=236
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=167.0, Tiempo minimo=72, Tiempo maximo=252
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=173.8, Tiempo minimo=60, Tiempo maximo=292
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=173.8, Tiempo minimo=74, Tiempo maximo=289
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=173.93333333333334, Tiempo minimo=62, Tiempo maximo=274
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=174.26666666666668, Tiempo minimo=57, Tiempo maximo=288
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=174.46666666666667, Tiempo minimo=98, Tiempo maximo=271
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=170.6, Tiempo minimo=107, Tiempo maximo=275
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=176.33333333333334, Tiempo minimo=48, Tiempo maximo=287
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=177.6, Tiempo minimo=98, Tiempo maximo=238
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=178.6, Tiempo minimo=84, Tiempo maximo=295
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=180.33333333333334, Tiempo minimo=91, Tiempo maximo=295
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=182.53333333333333, Tiempo minimo=78, Tiempo maximo=330
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=183.46666666666667, Tiempo minimo=127, Tiempo maximo=261
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=184.8, Tiempo minimo=80, Tiempo maximo=275
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=185.2, Tiempo minimo=36, Tiempo maximo=323
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=185.46666666666667, Tiempo minimo=68, Tiempo maximo=286
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=185.53333333333333, Tiempo minimo=57, Tiempo maximo=326
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=186.33333333333334, Tiempo minimo=20, Tiempo maximo=414
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=187.0, Tiempo minimo=33, Tiempo maximo=322
```

Figura 5-26 Tiempos GET/apartamentos con implementación reactiva y caché

```
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=221.6, Tiempo minimo=60, Tiempo maximo=845
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=234.8, Tiempo minimo=56, Tiempo maximo=805
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=243.93333333333334, Tiempo minimo=64, Tiempo maximo=918
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=249.13333333333333, Tiempo minimo=63, Tiempo maximo=840
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=249.4, Tiempo minimo=54, Tiempo maximo=840
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=249.66666666666666, Tiempo minimo=91, Tiempo maximo=804
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=250.26666666666668, Tiempo minimo=67, Tiempo maximo=830
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=253.2, Tiempo minimo=81, Tiempo maximo=907
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=253.86666666666667, Tiempo minimo=65, Tiempo maximo=824
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=258.2, Tiempo minimo=88, Tiempo maximo=840
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=258.53333333333336, Tiempo minimo=104, Tiempo maximo=842
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=260.8, Tiempo minimo=79, Tiempo maximo=1207
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=260.8666666666667, Tiempo minimo=36, Tiempo maximo=1369
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=265.0, Tiempo minimo=86, Tiempo maximo=842
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=266.6, Tiempo minimo=123, Tiempo maximo=843
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=272.6, Tiempo minimo=27, Tiempo maximo=1891
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=274.06666666666666, Tiempo minimo=14, Tiempo maximo=1629
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=274.8666666666667, Tiempo minimo=47, Tiempo maximo=1424
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=278.3333333333333, Tiempo minimo=26, Tiempo maximo=2065
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=279.73333333333335, Tiempo minimo=18, Tiempo maximo=2357
```

Figura 5-27 Tiempos GET/apartamentos con implementación tradicional y caché

```
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=208.53333333333333, Tiempo mínimo=35, Tiempo máximo=552
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=222.26666666666668, Tiempo mínimo=48, Tiempo máximo=429
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=223.26666666666668, Tiempo mínimo=81, Tiempo máximo=369
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=226.2, Tiempo mínimo=100, Tiempo máximo=435
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=234.46666666666667, Tiempo mínimo=73, Tiempo máximo=434
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=235.8, Tiempo mínimo=114, Tiempo máximo=394
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=236.2, Tiempo mínimo=53, Tiempo máximo=428
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=236.73333333333332, Tiempo mínimo=30, Tiempo máximo=476
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=239.66666666666666, Tiempo mínimo=125, Tiempo máximo=384
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=241.8, Tiempo mínimo=81, Tiempo máximo=397
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=242.8, Tiempo mínimo=60, Tiempo máximo=454
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=243.73333333333332, Tiempo mínimo=151, Tiempo máximo=342
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=243.93333333333334, Tiempo mínimo=76, Tiempo máximo=377
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=244.6, Tiempo mínimo=19, Tiempo máximo=572
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=246.06666666666666, Tiempo mínimo=12, Tiempo máximo=503
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=246.46666666666667, Tiempo mínimo=126, Tiempo máximo=440
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=246.73333333333332, Tiempo mínimo=57, Tiempo máximo=430
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=247.0, Tiempo mínimo=57, Tiempo máximo=411
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=247.73333333333332, Tiempo mínimo=56, Tiempo máximo=492
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=248.93333333333334, Tiempo mínimo=17, Tiempo máximo=460
```

Figura 5-28 Tiempos GET/apartamentos con implementación reactiva y sin caché

```
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=380.4, Tiempo mínimo=47, Tiempo máximo=1721
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=378.46666666666664, Tiempo mínimo=29, Tiempo máximo=1784
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=385.1333333333333, Tiempo mínimo=78, Tiempo máximo=1758
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=386.8666666666667, Tiempo mínimo=25, Tiempo máximo=1705
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=389.93333333333334, Tiempo mínimo=76, Tiempo máximo=1733
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=390.93333333333334, Tiempo mínimo=34, Tiempo máximo=3271
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=392.46666666666664, Tiempo mínimo=42, Tiempo máximo=1714
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=397.0, Tiempo mínimo=76, Tiempo máximo=1754
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=393.2, Tiempo mínimo=46, Tiempo máximo=1687
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=400.46666666666664, Tiempo mínimo=142, Tiempo máximo=1730
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=408.6666666666667, Tiempo mínimo=98, Tiempo máximo=1711
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=409.46666666666664, Tiempo mínimo=64, Tiempo máximo=1725
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=421.26666666666665, Tiempo mínimo=56, Tiempo máximo=2276
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=426.93333333333334, Tiempo mínimo=37, Tiempo máximo=3825
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=431.0, Tiempo mínimo=59, Tiempo máximo=2736
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=432.8666666666667, Tiempo mínimo=29, Tiempo máximo=4825
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=433.0, Tiempo mínimo=27, Tiempo máximo=4554
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=434.0, Tiempo mínimo=57, Tiempo máximo=3582
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=434.46666666666664, Tiempo mínimo=36, Tiempo máximo=4523
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=439.4, Tiempo mínimo=23, Tiempo máximo=4179
```

Figura 5-29 Tiempos GET/apartamentos con implementación tradicional y sin caché

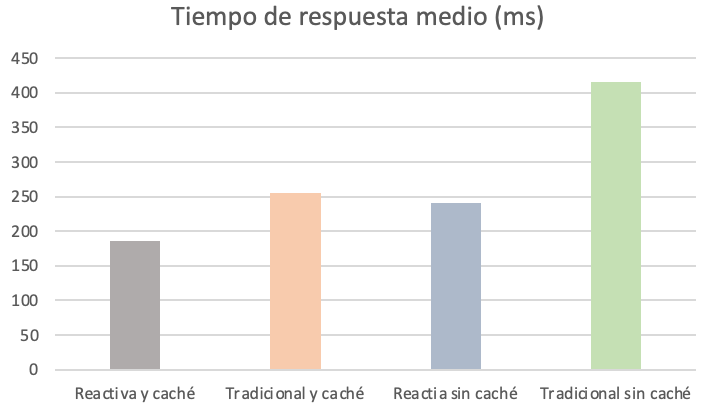

Figura 5-30 Gráfico tiempos de respuesta medios GET/apartamentos

Como se puede observar en el gráfico creado con los tiempos medios de las respuestas que otorgó el sistema en los diferentes formatos que se estudian, se puede concluir que el modelo reactivo es más rápido que el tradicional siempre que se compare en igualdad de condiciones. Esto significa que en las dos implementaciones con caché muestra un tiempo de respuesta menor el modelo reactivo, y en las dos implementaciones sin caché también. Además, implementación reactiva sin caché es más rápida, incluso, que la tradicional con caché.

5.7 Comparación GET/apartamentos/{codapt}

```
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=153.26666666666668, Tiempo minimo=34, Tiempo maximo=244
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=160.86666666666667, Tiempo minimo=42, Tiempo maximo=276
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=161.53333333333333, Tiempo minimo=50, Tiempo maximo=294
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=162.53333333333333, Tiempo minimo=38, Tiempo maximo=261
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=168.73333333333332, Tiempo minimo=42, Tiempo maximo=379
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=175.4, Tiempo minimo=38, Tiempo maximo=263
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=178.33333333333334, Tiempo minimo=39, Tiempo maximo=286
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=178.26666666666668, Tiempo minimo=52, Tiempo maximo=321
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=180.0, Tiempo minimo=84, Tiempo maximo=304
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=179.93333333333334, Tiempo minimo=51, Tiempo maximo=270
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=180.93333333333334, Tiempo minimo=20, Tiempo maximo=296
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=181.06666666666666, Tiempo minimo=95, Tiempo maximo=261
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=182.2, Tiempo minimo=47, Tiempo maximo=277
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=182.2, Tiempo minimo=88, Tiempo maximo=282
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=183.93333333333334, Tiempo minimo=42, Tiempo maximo=397
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=183.8, Tiempo minimo=81, Tiempo maximo=337
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=185.13333333333333, Tiempo minimo=46, Tiempo maximo=266
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=186.06666666666666, Tiempo minimo=30, Tiempo maximo=287
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=186.4, Tiempo minimo=40, Tiempo maximo=329
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=187.0, Tiempo minimo=19, Tiempo maximo=304
```

Figura 5-31 Tiempos GET/apartamentos/{codapt} con implementación reactiva y caché

```
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=190.93333333333334, Tiempo minimo=42, Tiempo maximo=1229
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=186.06666666666666, Tiempo minimo=51, Tiempo maximo=1233
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=195.53333333333333, Tiempo minimo=55, Tiempo maximo=1246
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=196.53333333333333, Tiempo minimo=67, Tiempo maximo=1253
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=196.86666666666667, Tiempo minimo=50, Tiempo maximo=1240
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=198.93333333333334, Tiempo minimo=49, Tiempo maximo=1296
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=200.93333333333334, Tiempo minimo=13, Tiempo maximo=1263
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=201.26666666666668, Tiempo minimo=51, Tiempo maximo=1281
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=202.93333333333334, Tiempo minimo=61, Tiempo maximo=1258
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=203.86666666666667, Tiempo minimo=64, Tiempo maximo=1269
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=204.26666666666668, Tiempo minimo=44, Tiempo maximo=1203
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=204.66666666666666, Tiempo minimo=29, Tiempo maximo=1296
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=205.13333333333333, Tiempo minimo=53, Tiempo maximo=1419
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=206.0, Tiempo minimo=70, Tiempo maximo=1284
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=206.8, Tiempo minimo=37, Tiempo maximo=1274
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=207.2, Tiempo minimo=39, Tiempo maximo=1331
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=207.33333333333334, Tiempo minimo=60, Tiempo maximo=1301
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=207.8, Tiempo minimo=30, Tiempo maximo=1279
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=209.4, Tiempo minimo=34, Tiempo maximo=1201
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=209.73333333333332, Tiempo minimo=16, Tiempo maximo=1425
```

Figura 5-32 Tiempos GET/apartamentos/{codapt} con implementación tradicional y caché

```
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=325.46666666666664, Tiempo mínimo=67, Tiempo máximo=774
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=339.53333333333336, Tiempo mínimo=20, Tiempo máximo=963
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=340.26666666666665, Tiempo mínimo=43, Tiempo máximo=1026
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=340.6666666666667, Tiempo mínimo=38, Tiempo máximo=929
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=342.93333333333334, Tiempo mínimo=48, Tiempo máximo=817
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=356.73333333333335, Tiempo mínimo=60, Tiempo máximo=735
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=357.73333333333335, Tiempo mínimo=20, Tiempo máximo=983
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=363.6666666666667, Tiempo mínimo=49, Tiempo máximo=794
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=369.6, Tiempo mínimo=38, Tiempo máximo=786
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=371.73333333333335, Tiempo mínimo=49, Tiempo máximo=1086
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=376.6666666666667, Tiempo mínimo=41, Tiempo máximo=1431
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=378.1333333333333, Tiempo mínimo=104, Tiempo máximo=723
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=378.93333333333334, Tiempo mínimo=54, Tiempo máximo=1063
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=379.26666666666665, Tiempo mínimo=41, Tiempo máximo=1441
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=382.8, Tiempo mínimo=43, Tiempo máximo=1052
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=386.53333333333336, Tiempo mínimo=112, Tiempo máximo=897
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=387.1333333333333, Tiempo mínimo=44, Tiempo máximo=1103
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=386.8666666666667, Tiempo mínimo=26, Tiempo máximo=824
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=389.6, Tiempo mínimo=49, Tiempo máximo=929
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=390.0, Tiempo mínimo=24, Tiempo máximo=1420
```

Figura 5-33 Tiempos GET/apartamentos/{codapt} con implementación reactiva y sin caché

```
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=520.1333333333333, Tiempo mínimo=98, Tiempo máximo=1240
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=590.6, Tiempo mínimo=416, Tiempo máximo=861
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=615.5333333333333, Tiempo mínimo=108, Tiempo máximo=1848
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=619.1333333333333, Tiempo mínimo=221, Tiempo máximo=1673
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=618.8666666666667, Tiempo mínimo=162, Tiempo máximo=1644
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=652.8666666666667, Tiempo mínimo=229, Tiempo máximo=1643
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=653.4, Tiempo mínimo=164, Tiempo máximo=1605
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=657.0, Tiempo mínimo=96, Tiempo máximo=1697
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=658.2, Tiempo mínimo=169, Tiempo máximo=1649
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=659.2, Tiempo mínimo=138, Tiempo máximo=1652
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=659.8, Tiempo mínimo=226, Tiempo máximo=1657
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=668.3333333333334, Tiempo mínimo=219, Tiempo máximo=1840
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=670.6666666666666, Tiempo mínimo=140, Tiempo máximo=1665
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=671.4, Tiempo mínimo=166, Tiempo máximo=2015
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=677.5333333333333, Tiempo mínimo=131, Tiempo máximo=1749
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=677.0666666666667, Tiempo mínimo=135, Tiempo máximo=1852
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=678.7333333333333, Tiempo mínimo=145, Tiempo máximo=1690
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=681.4666666666667, Tiempo mínimo=75, Tiempo máximo=1858
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=682.5333333333333, Tiempo mínimo=120, Tiempo máximo=1550
Cliente finalizado: URL=http://localhost:80/apartamentos/1, Peticiones=15, Tiempo promedio=682.4666666666667, Tiempo mínimo=70, Tiempo máximo=1592
```

Figura 5-34 Tiempos GET/apartamentos/{codapt} con implementación tradicional y sin caché

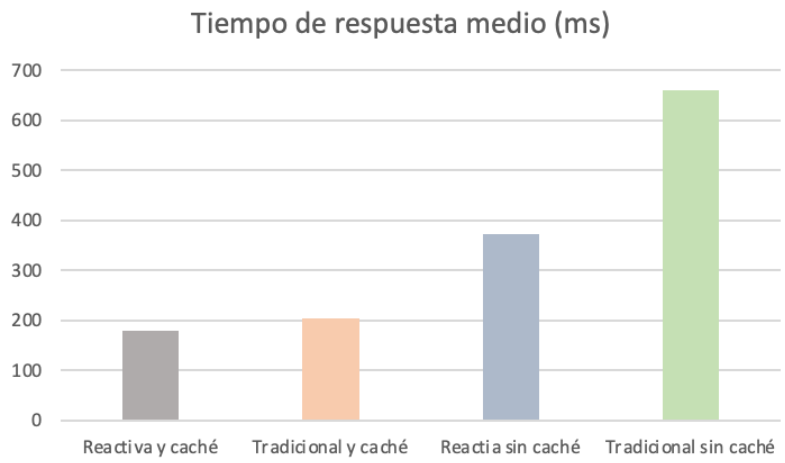

Figura 5-35 Gráfico tiempos de respuesta medios GET/apartamentos/{codapt}

Como se puede observar en el gráfico creado con los tiempos medios de las respuestas que otorgó el sistema en los diferentes formatos que se estudian, se puede concluir que el modelo reactivo es más rápido que el tradicional siempre que se compare en igualdad de condiciones. Esto significa que en las dos implementaciones con caché muestra un tiempo de respuesta menor el modelo reactivo, y en las dos implementaciones sin caché también. Además, se puede ver cómo, en este caso, los tiempos van aumentando de manera proporcional a medida que dejamos de utilizar elementos que ayudan a mejorar los tiempos de respuesta del sistema.

5.8 Comparación POST/apartamentos

```
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=251.33333333333334, Tiempo mínimo=142, Tiempo máximo=530
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=251.06666666666666, Tiempo mínimo=94, Tiempo máximo=465
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=273.1333333333333, Tiempo mínimo=96, Tiempo máximo=813
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=274.8666666666667, Tiempo mínimo=96, Tiempo máximo=584
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=278.8666666666667, Tiempo mínimo=60, Tiempo máximo=529
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=280.8, Tiempo mínimo=41, Tiempo máximo=788
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=284.0, Tiempo mínimo=80, Tiempo máximo=592
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=285.46666666666664, Tiempo mínimo=166, Tiempo máximo=598
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=287.0, Tiempo mínimo=151, Tiempo máximo=466
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=287.73333333333335, Tiempo mínimo=123, Tiempo máximo=783
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=288.2, Tiempo mínimo=124, Tiempo máximo=557
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=289.4, Tiempo mínimo=104, Tiempo máximo=588
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=290.26666666666665, Tiempo mínimo=125, Tiempo máximo=548
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=292.0, Tiempo mínimo=70, Tiempo máximo=508
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=292.8, Tiempo mínimo=145, Tiempo máximo=552
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=294.0, Tiempo mínimo=63, Tiempo máximo=824
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=295.06666666666666, Tiempo mínimo=98, Tiempo máximo=581
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=296.8666666666667, Tiempo mínimo=13, Tiempo máximo=622
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=296.8, Tiempo mínimo=88, Tiempo máximo=497
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=299.46666666666664, Tiempo mínimo=23, Tiempo máximo=809
```

Figura 5-36 Tiempos POST/aparamentos con implementación reactiva y caché

```
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=717.4, Tiempo mínimo=235, Tiempo máximo=3869
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=710.5333333333333, Tiempo mínimo=223, Tiempo máximo=3536
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=735.4666666666667, Tiempo mínimo=264, Tiempo máximo=3410
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=737.7333333333333, Tiempo mínimo=237, Tiempo máximo=3398
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=746.9333333333333, Tiempo mínimo=235, Tiempo máximo=3431
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=747.8666666666667, Tiempo mínimo=262, Tiempo máximo=3448
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=747.0666666666667, Tiempo mínimo=293, Tiempo máximo=3665
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=748.7333333333333, Tiempo mínimo=354, Tiempo máximo=3774
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=751.2666666666667, Tiempo mínimo=269, Tiempo máximo=3592
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=755.8666666666667, Tiempo mínimo=268, Tiempo máximo=3558
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=759.6666666666666, Tiempo mínimo=321, Tiempo máximo=3702
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=765.3333333333334, Tiempo mínimo=201, Tiempo máximo=3715
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=764.8, Tiempo mínimo=334, Tiempo máximo=3357
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=766.3333333333334, Tiempo mínimo=205, Tiempo máximo=3520
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=771.7333333333333, Tiempo mínimo=175, Tiempo máximo=3607
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=772.9333333333333, Tiempo mínimo=142, Tiempo máximo=3422
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=777.2666666666667, Tiempo mínimo=138, Tiempo máximo=3510
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=780.4, Tiempo mínimo=92, Tiempo máximo=3732
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=780.4, Tiempo mínimo=59, Tiempo máximo=3698
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=783.2666666666667, Tiempo mínimo=48, Tiempo máximo=3797
```

Figura 5-37 Tiempos POST/apartamentos con implementación tradicional y caché

```
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=268.2, Tiempo mínimo=49, Tiempo máximo=1211
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=285.2, Tiempo mínimo=132, Tiempo máximo=1123
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=297.73333333333335, Tiempo mínimo=28, Tiempo máximo=1012
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=301.93333333333334, Tiempo mínimo=141, Tiempo máximo=1071
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=302.8, Tiempo mínimo=91, Tiempo máximo=1148
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=308.1333333333333, Tiempo mínimo=67, Tiempo máximo=1109
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=311.6, Tiempo mínimo=174, Tiempo máximo=1019
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=311.73333333333335, Tiempo mínimo=113, Tiempo máximo=1155
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=318.6666666666667, Tiempo mínimo=52, Tiempo máximo=1062
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=316.6, Tiempo mínimo=112, Tiempo máximo=1124
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=317.2, Tiempo mínimo=133, Tiempo máximo=1043
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=318.1333333333333, Tiempo mínimo=106, Tiempo máximo=1166
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=319.46666666666664, Tiempo mínimo=84, Tiempo máximo=1150
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=319.8666666666667, Tiempo mínimo=89, Tiempo máximo=1074
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=320.2, Tiempo mínimo=123, Tiempo máximo=1105
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=321.6666666666667, Tiempo mínimo=83, Tiempo máximo=1081
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=322.0, Tiempo mínimo=123, Tiempo máximo=1188
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=321.46666666666664, Tiempo mínimo=61, Tiempo máximo=1051
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=322.46666666666664, Tiempo mínimo=97, Tiempo máximo=982
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=323.6, Tiempo mínimo=21, Tiempo máximo=1037
```

Figura 5-38 Tiempos POST/apartamentos con implementación reactiva y sin caché

```
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=562.0, Tiempo mínimo=186, Tiempo máximo=2271
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=582.1333333333333, Tiempo mínimo=221, Tiempo máximo=2459
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=584.0666666666667, Tiempo mínimo=267, Tiempo máximo=2669
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=585.0666666666667, Tiempo mínimo=130, Tiempo máximo=2350
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=602.0666666666667, Tiempo mínimo=226, Tiempo máximo=2542
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=605.6, Tiempo mínimo=284, Tiempo máximo=2816
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=605.8666666666667, Tiempo mínimo=253, Tiempo máximo=2638
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=606.5333333333333, Tiempo mínimo=271, Tiempo máximo=2317
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=609.4, Tiempo mínimo=285, Tiempo máximo=2896
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=611.4666666666667, Tiempo mínimo=279, Tiempo máximo=2926
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=614.7333333333333, Tiempo mínimo=231, Tiempo máximo=2282
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=615.4666666666667, Tiempo mínimo=252, Tiempo máximo=2555
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=620.7333333333333, Tiempo mínimo=268, Tiempo máximo=2659
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=624.2, Tiempo mínimo=110, Tiempo máximo=2709
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=626.1333333333333, Tiempo mínimo=225, Tiempo máximo=2373
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=626.6, Tiempo mínimo=181, Tiempo máximo=2632
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=630.1333333333333, Tiempo mínimo=108, Tiempo máximo=2763
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=631.1333333333333, Tiempo mínimo=183, Tiempo máximo=2902
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=632.4, Tiempo mínimo=68, Tiempo máximo=2348
Cliente finalizado: URL=http://localhost:80/apartamentos, Peticiones=15, Tiempo promedio=634.4, Tiempo mínimo=52, Tiempo máximo=2492
```

Figura 5-39 Tiempos POST/apartamentos con implementación tradicional y sin caché

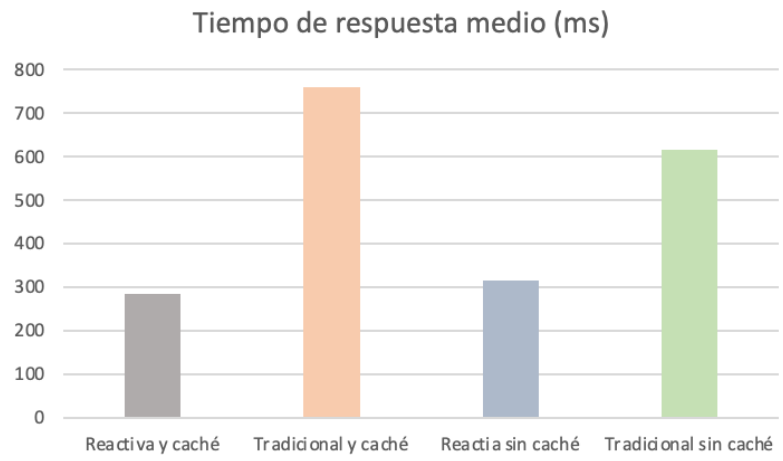

Figura 5-40 Gráfico tiempos de respuesta medios POST/apartamentos

Como se puede observar en el gráfico creado con los tiempos medios de las respuestas que otorgó el sistema en los diferentes formatos que se estudian, se puede concluir que el modelo reactivo es más rápido que el tradicional siempre que se compare en igualdad de condiciones. Esto significa que en las dos implementaciones con caché muestra un tiempo de respuesta menor el modelo reactivo, y en las dos implementaciones sin caché también. Además, en este caso los tiempos del modelo reactivo no varían mucho aun eliminando el caché, lo que indica que sin caché el sistema sigue ofreciendo un tiempo de respuesta muy bueno.

5.9 Comparación PUT/apartamentos

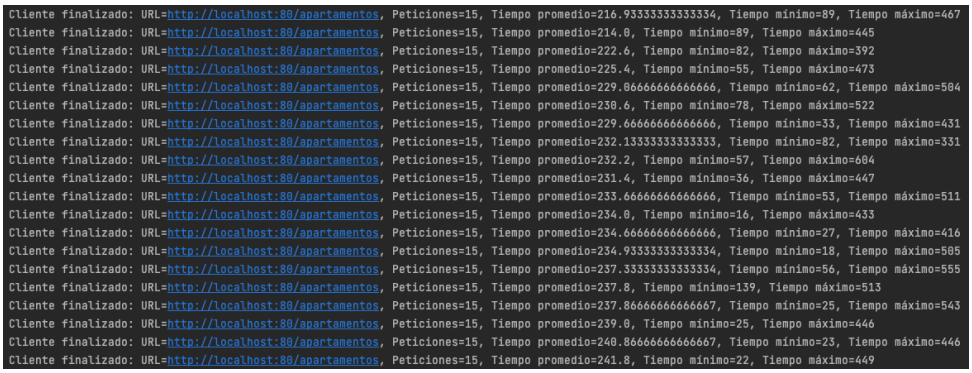

Figura 5-41 Tiempos PUT/apartamentos con implementación reactiva y caché

Figura 5-42 Tiempos PUT/apartamentos con implementación tradicional y caché

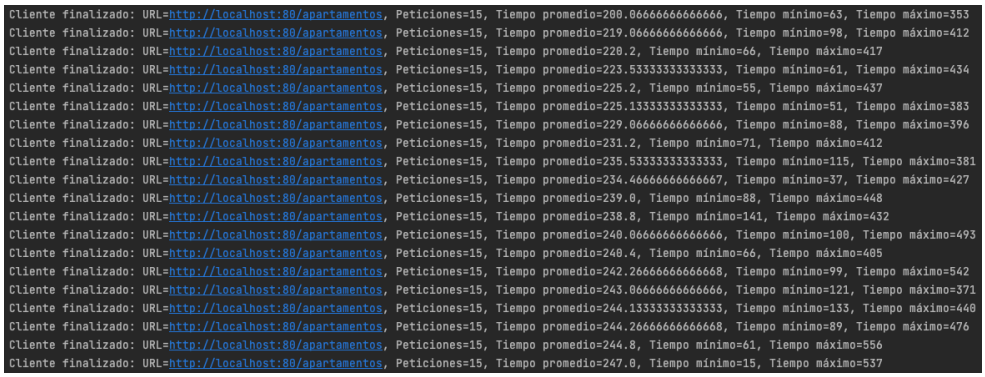

Figura 5-43 Tiempos PUT/apartamentos con implementación reactiva y sin caché

```
Cliente finalizado: URL=http://localhost:80/apartamentos/01, Peticiones=15, Tiempo promedio=281.8, Tiempo mínimo=19, Tiempo máximo=1416
Cliente finalizado: URL=http://localhost:80/apartamentos/61, Peticiones=15, Tiempo promedio=371.1333333333333, Tiempo mínimo=97, Tiempo máximo=684
Cliente finalizado: URL=http://localhost:80/apartamentos/141, Peticiones=15, Tiempo promedio=394.26666666666665, Tiempo mínimo=112, Tiempo máximo=844
Cliente finalizado: URL=http://localhost:80/apartamentos/71, Peticiones=15, Tiempo promedio=397.93333333333334, Tiempo mínimo=130, Tiempo máximo=866
Cliente finalizado: URL=http://localhost:80/apartamentos/181, Peticiones=15, Tiempo promedio=399.1333333333333, Tiempo mínimo=120, Tiempo máximo=1037
Cliente finalizado: URL=http://localhost:80/apartamentos/131, Peticiones=15, Tiempo promedio=401.6666666666667, Tiempo mínimo=82, Tiempo máximo=1124
Cliente finalizado: URL=http://localhost:80/apartamentos/171, Peticiones=15, Tiempo promedio=416.46666666666664, Tiempo mínimo=137, Tiempo máximo=896
Cliente finalizado: URL=http://localhost:80/apartamentos/21, Peticiones=15, Tiempo promedio=419.3333333333333, Tiempo mínimo=185, Tiempo máximo=970
Cliente finalizado: URL=http://localhost:80/apartamentos/101, Peticiones=15, Tiempo promedio=435.46666666666664, Tiempo mínimo=143, Tiempo máximo=1516
Cliente finalizado: URL=http://localhost:80/apartamentos/41, Peticiones=15, Tiempo promedio=455.4, Tiempo mínimo=101, Tiempo máximo=2330
Cliente finalizado: URL=http://localhost:80/apartamentos/151, Peticiones=15, Tiempo promedio=475.6, Tiempo mínimo=73, Tiempo máximo=3386
Cliente finalizado: URL=http://localhost:80/apartamentos/51, Peticiones=15, Tiempo promedio=477.26666666666665, Tiempo mínimo=140, Tiempo máximo=2545
Cliente finalizado: URL=http://localhost:80/apartamentos/81, Peticiones=15, Tiempo promedio=478.3333333333333, Tiempo mínimo=116, Tiempo máximo=1988
Cliente finalizado: URL=http://localhost:80/apartamentos/191, Peticiones=15, Tiempo promedio=486.73333333333335, Tiempo mínimo=100, Tiempo máximo=2996
Cliente finalizado: URL=http://localhost:80/apartamentos/121, Peticiones=15, Tiempo promedio=488.3333333333333, Tiempo mínimo=71, Tiempo máximo=2524
Cliente finalizado: URL=http://localhost:80/apartamentos/31, Peticiones=15, Tiempo promedio=495.93333333333334, Tiempo mínimo=103, Tiempo máximo=2745
Cliente finalizado: URL=http://localhost:80/apartamentos/91, Peticiones=15, Tiempo promedio=498.4, Tiempo mínimo=89, Tiempo máximo=3323
Cliente finalizado: URL=http://localhost:80/apartamentos/161, Peticiones=15, Tiempo promedio=499.6, Tiempo mínimo=50, Tiempo máximo=3832
Cliente finalizado: URL=http://localhost:80/apartamentos/111, Peticiones=15, Tiempo promedio=499.53333333333336, Tiempo mínimo=74, Tiempo máximo=3501
Cliente finalizado: URL=http://localhost:80/apartamentos/11, Peticiones=15, Tiempo promedio=502.6, Tiempo mínimo=64, Tiempo máximo=3696
```

Figura 5-44 Tiempos PUT/apartamentos con implementación tradicional y sin caché

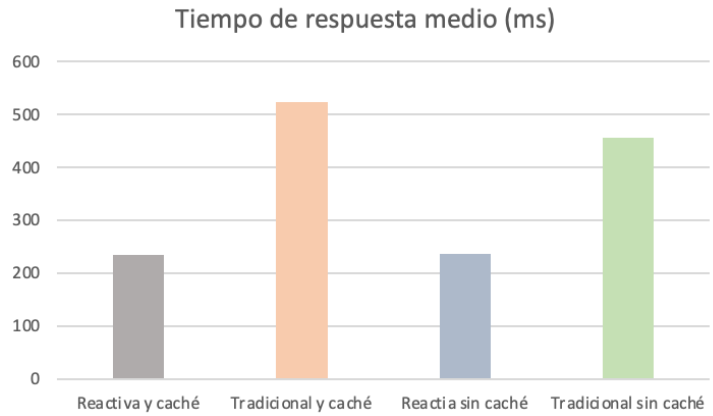

Figura 5-45 Gráfico tiempos de respuesta medios PUT/apartamentos

Como se puede observar en el gráfico creado con los tiempos medios de las respuestas que otorgó el sistema en los diferentes formatos que se estudian, se puede concluir que el modelo reactivo es más rápido que el tradicional siempre que se compare en igualdad de condiciones. Esto significa que en las dos implementaciones con caché muestra un tiempo de respuesta menor el modelo reactivo, y en las dos implementaciones sin caché también. Además, se puede ver cómo, al ser un método PUT, se obtiene un tiempo de respuesta menor en los modelos sin caché ya que con caché no se brinda un acceso más rápido a los datos, sino que hay que modificar los que ya se encuentran disponibles, por tanto, se pierde un poco más de tiempo.

Aun así, el porcentaje de solicitudes de este estilo es bastante menor con el de solicitudes GET, por lo que en el sistema en general sigue siendo beneficioso el uso de caché.

5.10 Comparación DELETE/apartamentos/{codapt}

```
Cliente finalizado: URL=http://localhost:80/apartamentos/27, Peticiones=15, Tiempo promedio=350.26666666666665, Tiempo mínimo=92, Tiempo máximo=801
Cliente finalizado: URL=http://localhost:80/apartamentos/187, Peticiones=15, Tiempo promedio=341.1333333333333, Tiempo mínimo=45, Tiempo máximo=809
Cliente finalizado: URL=http://localhost:80/apartamentos/127, Peticiones=15, Tiempo promedio=429.1333333333333, Tiempo mínimo=104, Tiempo máximo=975
Cliente finalizado: URL=http://localhost:80/apartamentos/147, Peticiones=15, Tiempo promedio=455.4, Tiempo mínimo=179, Tiempo máximo=735
Cliente finalizado: URL=http://localhost:80/apartamentos/87, Peticiones=15, Tiempo promedio=457.3333333333333, Tiempo mínimo=138, Tiempo máximo=1058
Cliente finalizado: URL=http://localhost:80/apartamentos/137, Peticiones=15, Tiempo promedio=467.4, Tiempo mínimo=185, Tiempo máximo=1191
Cliente finalizado: URL=http://localhost:80/apartamentos/07, Peticiones=15, Tiempo promedio=473.53333333333336, Tiempo mínimo=28, Tiempo máximo=1416
Cliente finalizado: URL=http://localhost:80/apartamentos/97, Peticiones=15, Tiempo promedio=476.6666666666667, Tiempo mínimo=116, Tiempo máximo=1048
Cliente finalizado: URL=http://localhost:80/apartamentos/117, Peticiones=15, Tiempo promedio=482.4, Tiempo mínimo=120, Tiempo máximo=1041
Cliente finalizado: URL=http://localhost:80/apartamentos/47, Peticiones=15, Tiempo promedio=485.1333333333333, Tiempo mínimo=108, Tiempo máximo=770
Cliente finalizado: URL=http://localhost:80/apartamentos/177, Peticiones=15, Tiempo promedio=485.6, Tiempo mínimo=128, Tiempo máximo=1285
Cliente finalizado: URL=http://localhost:80/apartamentos/157, Peticiones=15, Tiempo promedio=486.8, Tiempo mínimo=111, Tiempo máximo=918
Cliente finalizado: URL=http://localhost:80/apartamentos/67, Peticiones=15, Tiempo promedio=488.53333333333336, Tiempo mínimo=99, Tiempo máximo=1339
Cliente finalizado: URL=http://localhost:80/apartamentos/197, Peticiones=15, Tiempo promedio=491.8, Tiempo mínimo=84, Tiempo máximo=754
Cliente finalizado: URL=http://localhost:80/apartamentos/57, Peticiones=15, Tiempo promedio=493.93333333333334, Tiempo mínimo=105, Tiempo máximo=888
Cliente finalizado: URL=http://localhost:80/apartamentos/37, Peticiones=15, Tiempo promedio=493.1333333333333, Tiempo mínimo=134, Tiempo máximo=1014
Cliente finalizado: URL=http://localhost:80/apartamentos/107, Peticiones=15, Tiempo promedio=495.06666666666666, Tiempo mínimo=82, Tiempo máximo=1388
Cliente finalizado: URL=http://localhost:80/apartamentos/17, Peticiones=15, Tiempo promedio=496.2, Tiempo mínimo=116, Tiempo máximo=886
Cliente finalizado: URL=http://localhost:80/apartamentos/77, Peticiones=15, Tiempo promedio=496.2, Tiempo mínimo=56, Tiempo máximo=1240
Cliente finalizado: URL=http://localhost:80/apartamentos/167, Peticiones=15, Tiempo promedio=496.6, Tiempo mínimo=18, Tiempo máximo=1151
```

Figura 5-46 Tiempos DELETE/apartamentos/{codapt} con implementación reactiva y caché

```
Cliente finalizado: URL=http://localhost:80/apartamentos/107, Peticiones=15, Tiempo promedio=446.46666666666664, Tiempo mínimo=80, Tiempo máximo=1351
Cliente finalizado: URL=http://localhost:80/apartamentos/127, Peticiones=15, Tiempo promedio=444.06666666666666, Tiempo mínimo=119, Tiempo máximo=1055
Cliente finalizado: URL=http://localhost:80/apartamentos/157, Peticiones=15, Tiempo promedio=441.0, Tiempo mínimo=121, Tiempo máximo=1537
Cliente finalizado: URL=http://localhost:80/apartamentos/167, Peticiones=15, Tiempo promedio=445.3333333333333, Tiempo mínimo=119, Tiempo máximo=969
Cliente finalizado: URL=http://localhost:80/apartamentos/187, Peticiones=15, Tiempo promedio=445.3333333333333, Tiempo mínimo=86, Tiempo máximo=1747
Cliente finalizado: URL=http://localhost:80/apartamentos/197, Peticiones=15, Tiempo promedio=448.06666666666666, Tiempo mínimo=33, Tiempo máximo=1736
Cliente finalizado: URL=http://localhost:80/apartamentos/27, Peticiones=15, Tiempo promedio=459.3333333333333, Tiempo mínimo=53, Tiempo máximo=1000
Cliente finalizado: URL=http://localhost:80/apartamentos/117, Peticiones=15, Tiempo promedio=457.46666666666664, Tiempo mínimo=99, Tiempo máximo=1059
Cliente finalizado: URL=http://localhost:80/apartamentos/57, Peticiones=15, Tiempo promedio=463.73333333333335, Tiempo mínimo=48, Tiempo máximo=2361
Cliente finalizado: URL=http://localhost:80/apartamentos/17, Peticiones=15, Tiempo promedio=470.8, Tiempo mínimo=59, Tiempo máximo=1428
Cliente finalizado: URL=http://localhost:80/apartamentos/177, Peticiones=15, Tiempo promedio=461.93333333333334, Tiempo mínimo=81, Tiempo máximo=1167
Cliente finalizado: URL=http://localhost:80/apartamentos/137, Peticiones=15, Tiempo promedio=473.4, Tiempo mínimo=48, Tiempo máximo=3090
Cliente finalizado: URL=http://localhost:80/apartamentos/97, Peticiones=15, Tiempo promedio=484.8, Tiempo mínimo=33, Tiempo máximo=4198
Cliente finalizado: URL=http://localhost:80/apartamentos/37, Peticiones=15, Tiempo promedio=487.8666666666667, Tiempo mínimo=24, Tiempo máximo=3217
Cliente finalizado: URL=http://localhost:80/apartamentos/77, Peticiones=15, Tiempo promedio=486.53333333333336, Tiempo mínimo=22, Tiempo máximo=2456
Cliente finalizado: URL=http://localhost:80/apartamentos/67, Peticiones=15, Tiempo promedio=486.6, Tiempo mínimo=36, Tiempo máximo=3062
Cliente finalizado: URL=http://localhost:80/apartamentos/07, Peticiones=15, Tiempo promedio=491.2, Tiempo mínimo=32, Tiempo máximo=3616
Cliente finalizado: URL=http://localhost:80/apartamentos/87, Peticiones=15, Tiempo promedio=490.06666666666666, Tiempo mínimo=35, Tiempo máximo=3758
Cliente finalizado: URL=http://localhost:80/apartamentos/47, Peticiones=15, Tiempo promedio=491.3333333333333, Tiempo mínimo=40, Tiempo máximo=4027
Cliente finalizado: URL=http://localhost:80/apartamentos/147, Peticiones=15, Tiempo promedio=483.46666666666664, Tiempo mínimo=31, Tiempo máximo=4497
```

Figura 5-47 Tiempos DELETE/apartamentos/{codapt} con implementación tradicional y caché

```
Cliente finalizado: URL=http://localhost:80/apartamentos/77, Peticiones=15, Tiempo promedio=331.8666666666667, Tiempo mínimo=51, Tiempo máximo=758
Cliente finalizado: URL=http://localhost:80/apartamentos/187, Peticiones=15, Tiempo promedio=385.6666666666667, Tiempo mínimo=99, Tiempo máximo=825
Cliente finalizado: URL=http://localhost:80/apartamentos/27, Peticiones=15, Tiempo promedio=395.73333333333335, Tiempo mínimo=98, Tiempo máximo=570
Cliente finalizado: URL=http://localhost:80/apartamentos/107, Peticiones=15, Tiempo promedio=395.06666666666666, Tiempo mínimo=139, Tiempo máximo=745
Cliente finalizado: URL=http://localhost:80/apartamentos/57, Peticiones=15, Tiempo promedio=407.6, Tiempo mínimo=28, Tiempo máximo=995
Cliente finalizado: URL=http://localhost:80/apartamentos/137, Peticiones=15, Tiempo promedio=427.3333333333333, Tiempo mínimo=127, Tiempo máximo=689
Cliente finalizado: URL=http://localhost:80/apartamentos/07, Peticiones=15, Tiempo promedio=432.3333333333333, Tiempo mínimo=101, Tiempo máximo=748
Cliente finalizado: URL=http://localhost:80/apartamentos/177, Peticiones=15, Tiempo promedio=433.1333333333333, Tiempo mínimo=95, Tiempo máximo=647
Cliente finalizado: URL=http://localhost:80/apartamentos/67, Peticiones=15, Tiempo promedio=433.8, Tiempo mínimo=35, Tiempo máximo=843
Cliente finalizado: URL=http://localhost:80/apartamentos/157, Peticiones=15, Tiempo promedio=437.2, Tiempo mínimo=138, Tiempo máximo=842
Cliente finalizado: URL=http://localhost:80/apartamentos/17, Peticiones=15, Tiempo promedio=438.6666666666667, Tiempo mínimo=179, Tiempo máximo=795
Cliente finalizado: URL=http://localhost:80/apartamentos/147, Peticiones=15, Tiempo promedio=438.6, Tiempo mínimo=122, Tiempo máximo=838
Cliente finalizado: URL=http://localhost:80/apartamentos/37, Peticiones=15, Tiempo promedio=441.53333333333336, Tiempo mínimo=99, Tiempo máximo=793
Cliente finalizado: URL=http://localhost:80/apartamentos/47, Peticiones=15, Tiempo promedio=448.4, Tiempo mínimo=65, Tiempo máximo=844
Cliente finalizado: URL=http://localhost:80/apartamentos/197, Peticiones=15, Tiempo promedio=445.73333333333335, Tiempo mínimo=174, Tiempo máximo=784
Cliente finalizado: URL=http://localhost:80/apartamentos/127, Peticiones=15, Tiempo promedio=450.26666666666665, Tiempo mínimo=27, Tiempo máximo=729
Cliente finalizado: URL=http://localhost:80/apartamentos/87, Peticiones=15, Tiempo promedio=451.6666666666667, Tiempo mínimo=130, Tiempo máximo=696
Cliente finalizado: URL=http://localhost:80/apartamentos/117, Peticiones=15, Tiempo promedio=453.3333333333333, Tiempo mínimo=92, Tiempo máximo=759
Cliente finalizado: URL=http://localhost:80/apartamentos/167, Peticiones=15, Tiempo promedio=453.26666666666665, Tiempo mínimo=99, Tiempo máximo=966
Cliente finalizado: URL=http://localhost:80/apartamentos/97, Peticiones=15, Tiempo promedio=453.73333333333335, Tiempo mínimo=68, Tiempo máximo=1004
```

Figura 5-48 Tiempos DELETE/apartamentos/{codapt} con implementación reactiva y sin caché

```
Cliente finalizado: URL=http://localhost:80/apartamentos/31, Peticiones=15, Tiempo promedio=344.6, Tiempo mínimo=50, Tiempo máximo=736
Cliente finalizado: URL=http://localhost:80/apartamentos/51, Peticiones=15, Tiempo promedio=341.73333333333335, Tiempo mínimo=21, Tiempo máximo=836
Cliente finalizado: URL=http://localhost:80/apartamentos/11, Peticiones=15, Tiempo promedio=360.26666666666665, Tiempo mínimo=107, Tiempo máximo=803
Cliente finalizado: URL=http://localhost:80/apartamentos/71, Peticiones=15, Tiempo promedio=367.73333333333335, Tiempo mínimo=49, Tiempo máximo=664
Cliente finalizado: URL=http://localhost:80/apartamentos/91, Peticiones=15, Tiempo promedio=372.8666666666667, Tiempo mínimo=23, Tiempo máximo=724
Cliente finalizado: URL=http://localhost:80/apartamentos/111, Peticiones=15, Tiempo promedio=391.0, Tiempo mínimo=123, Tiempo máximo=891
Cliente finalizado: URL=http://localhost:80/apartamentos/121, Peticiones=15, Tiempo promedio=395.2, Tiempo mínimo=15, Tiempo máximo=1643
Cliente finalizado: URL=http://localhost:80/apartamentos/191, Peticiones=15, Tiempo promedio=397.4, Tiempo mínimo=62, Tiempo máximo=1368
Cliente finalizado: URL=http://localhost:80/apartamentos/41, Peticiones=15, Tiempo promedio=402.6, Tiempo mínimo=129, Tiempo máximo=1525
Cliente finalizado: URL=http://localhost:80/apartamentos/131, Peticiones=15, Tiempo promedio=403.73333333333335, Tiempo mínimo=68, Tiempo máximo=1256
Cliente finalizado: URL=http://localhost:80/apartamentos/01, Peticiones=15, Tiempo promedio=405.2, Tiempo mínimo=59, Tiempo máximo=1603
Cliente finalizado: URL=http://localhost:80/apartamentos/101, Peticiones=15, Tiempo promedio=407.0, Tiempo mínimo=81, Tiempo máximo=961
Cliente finalizado: URL=http://localhost:80/apartamentos/171, Peticiones=15, Tiempo promedio=406.6, Tiempo mínimo=103, Tiempo máximo=1604
Cliente finalizado: URL=http://localhost:80/apartamentos/151, Peticiones=15, Tiempo promedio=407.2, Tiempo mínimo=118, Tiempo máximo=1600
Cliente finalizado: URL=http://localhost:80/apartamentos/141, Peticiones=15, Tiempo promedio=410.6666666666667, Tiempo mínimo=33, Tiempo máximo=1101
Cliente finalizado: URL=http://localhost:80/apartamentos/21, Peticiones=15, Tiempo promedio=411.6, Tiempo mínimo=57, Tiempo máximo=1141
Cliente finalizado: URL=http://localhost:80/apartamentos/81, Peticiones=15, Tiempo promedio=413.26666666666665, Tiempo mínimo=19, Tiempo máximo=1120
Cliente finalizado: URL=http://localhost:80/apartamentos/61, Peticiones=15, Tiempo promedio=414.46666666666664, Tiempo mínimo=42, Tiempo máximo=1401
Cliente finalizado: URL=http://localhost:80/apartamentos/181, Peticiones=15, Tiempo promedio=414.8, Tiempo mínimo=54, Tiempo máximo=961
Cliente finalizado: URL=http://localhost:80/apartamentos/161, Peticiones=15, Tiempo promedio=415.0, Tiempo mínimo=35, Tiempo máximo=1180
```

Figura 5-49 Tiempos DELETE/apartamentos/{codapt} con implementación tradicional y sin caché

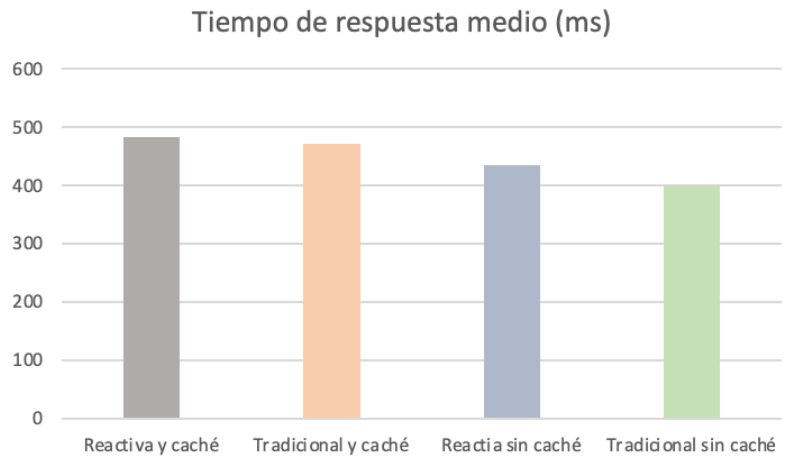

Figura 5-50 Gráfico tiempos de respuesta medios DELETE/apartamentos/{codapt}

Como se puede observar en el gráfico creado con los tiempos medios de las respuestas que otorgó el sistema en los diferentes formatos que se estudian, en este caso el modelo reactivo has sido un poco más lento que el tradicional. Esto puede deberse a otras causas como un sobrecalentamiento del portátil en el momento de la medición, porque es un resultado extraño.

Aun así, se mantiene que al ser un método DELETE, se obtiene un tiempo de respuesta menor en los modelos sin caché ya que con caché no se brinda un acceso más rápido a los datos, sino que hay que eliminar los que ya se encuentran disponibles, por tanto, se pierde un poco más de tiempo.

5.11 Comparación GET/apartamentos/{codapt}/localidades

```
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=32.733333333333334, Tiempo mínimo=15, Tiempo máximo=104
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=34.46666666666667, Tiempo mínimo=10, Tiempo máximo=121
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=27.4, Tiempo mínimo=10, Tiempo máximo=107
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=33.8, Tiempo mínimo=13, Tiempo máximo=120
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=27.066666666666666, Tiempo mínimo=12, Tiempo máximo=118
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=35.2, Tiempo mínimo=7, Tiempo máximo=106
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=35.6, Tiempo mínimo=13, Tiempo máximo=117
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=36.266666666666666, Tiempo mínimo=13, Tiempo máximo=142
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=36.93333333333333, Tiempo mínimo=19, Tiempo máximo=108
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=37.53333333333333, Tiempo mínimo=13, Tiempo máximo=122
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=37.46666666666667, Tiempo mínimo=10, Tiempo máximo=178
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=38.53333333333333, Tiempo mínimo=14, Tiempo máximo=139
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=38.6, Tiempo mínimo=10, Tiempo máximo=149
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=39.2, Tiempo mínimo=3, Tiempo máximo=115
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=39.46666666666667, Tiempo mínimo=14, Tiempo máximo=130
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=40.0, Tiempo mínimo=6, Tiempo máximo=143
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=40.53333333333333, Tiempo mínimo=13, Tiempo máximo=139
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=41.0, Tiempo mínimo=4, Tiempo máximo=115
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=40.86666666666667, Tiempo mínimo=5, Tiempo máximo=129
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=41.06666666666667, Tiempo mínimo=5, Tiempo máximo=121
```

Figura 5-51 Tiempos GET/apartamentos/{codapt}/localidades con implementación reactiva y caché

```
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=38.53333333333333, Tiempo mínimo=9, Tiempo máximo=107
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=46.6, Tiempo mínimo=11, Tiempo máximo=132
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=46.46666666666667, Tiempo mínimo=8, Tiempo máximo=107
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=52.13333333333333, Tiempo mínimo=14, Tiempo máximo=136
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=53.8, Tiempo mínimo=14, Tiempo máximo=137
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=52.86666666666667, Tiempo mínimo=22, Tiempo máximo=105
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=54.666666666666664, Tiempo mínimo=19, Tiempo máximo=105
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=55.53333333333333, Tiempo mínimo=16, Tiempo máximo=123
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=55.46666666666667, Tiempo mínimo=23, Tiempo máximo=134
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=55.733333333333334, Tiempo mínimo=23, Tiempo máximo=121
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=55.666666666666664, Tiempo mínimo=24, Tiempo máximo=119
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=55.733333333333334, Tiempo mínimo=5, Tiempo máximo=129
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=55.86666666666667, Tiempo mínimo=29, Tiempo máximo=106
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=56.46666666666667, Tiempo mínimo=12, Tiempo máximo=135
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=58.0, Tiempo mínimo=15, Tiempo máximo=129
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=57.4, Tiempo mínimo=18, Tiempo máximo=110
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=57.666666666666664, Tiempo mínimo=8, Tiempo máximo=108
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=58.06666666666667, Tiempo mínimo=31, Tiempo máximo=120
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=58.333333333333336, Tiempo mínimo=26, Tiempo máximo=129
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=57.93333333333333, Tiempo mínimo=18, Tiempo máximo=134
```

Figura 5-52 Tiempos GET/apartamentos/{codapt}/localidades con implementación tradicional y caché

```
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=165.4, Tiempo mínimo=56, Tiempo máximo=284
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=188.4, Tiempo mínimo=57, Tiempo máximo=376
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=188.6, Tiempo mínimo=66, Tiempo máximo=419
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=193.53333333333333, Tiempo mínimo=81, Tiempo máximo=300
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=196.53333333333333, Tiempo mínimo=57, Tiempo máximo=316
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=199.0, Tiempo mínimo=76, Tiempo máximo=306
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=199.46666666666667, Tiempo mínimo=81, Tiempo máximo=318
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=200.86666666666667, Tiempo mínimo=119, Tiempo máximo=380
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=202.66666666666666, Tiempo mínimo=108, Tiempo máximo=318
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=203.2, Tiempo mínimo=28, Tiempo máximo=425
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=202.4, Tiempo mínimo=87, Tiempo máximo=330
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=207.8, Tiempo mínimo=53, Tiempo máximo=405
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=208.4, Tiempo mínimo=91, Tiempo máximo=428
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=209.06666666666666, Tiempo mínimo=61, Tiempo máximo=344
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=210.73333333333332, Tiempo mínimo=66, Tiempo máximo=373
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=211.46666666666667, Tiempo mínimo=107, Tiempo máximo=486
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=212.53333333333333, Tiempo mínimo=39, Tiempo máximo=440
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=213.2, Tiempo mínimo=63, Tiempo máximo=455
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=213.46666666666667, Tiempo mínimo=72, Tiempo máximo=361
Cliente finalizado: URL=http://localhost:80/apartamentos/1/localidades, Peticiones=15, Tiempo promedio=214.0, Tiempo mínimo=61, Tiempo máximo=430
```

Figura 5-53 Tiempos GET/apartamentos/{codapt}/localidades con implementación reactiva y sin caché

```
Cliente finalizado: URL=http://localhost:80/apartamentos/6/localidades, Peticiones=15, Tiempo promedio=433.1333333333333, Tiempo mínimo=100, Tiempo máximo=924
Cliente finalizado: URL=http://localhost:80/apartamentos/6/localidades, Peticiones=15, Tiempo promedio=475.73333333333335, Tiempo mínimo=77, Tiempo máximo=1087
Cliente finalizado: URL=http://localhost:80/apartamentos/6/localidades, Peticiones=15, Tiempo promedio=475.6666666666667, Tiempo mínimo=66, Tiempo máximo=1137
Cliente finalizado: URL=http://localhost:80/apartamentos/6/localidades, Peticiones=15, Tiempo promedio=476.3333333333333, Tiempo mínimo=106, Tiempo máximo=997
Cliente finalizado: URL=http://localhost:80/apartamentos/6/localidades, Peticiones=15, Tiempo promedio=477.46666666666664, Tiempo mínimo=110, Tiempo máximo=1142
Cliente finalizado: URL=http://localhost:80/apartamentos/6/localidades, Peticiones=15, Tiempo promedio=481.53333333333336, Tiempo mínimo=117, Tiempo máximo=879
Cliente finalizado: URL=http://localhost:80/apartamentos/6/localidades, Peticiones=15, Tiempo promedio=493.06666666666666, Tiempo mínimo=214, Tiempo máximo=767
Cliente finalizado: URL=http://localhost:80/apartamentos/6/localidades, Peticiones=15, Tiempo promedio=495.6666666666667, Tiempo mínimo=157, Tiempo máximo=1068
Cliente finalizado: URL=http://localhost:80/apartamentos/6/localidades, Peticiones=15, Tiempo promedio=497.8666666666667, Tiempo mínimo=125, Tiempo máximo=883
Cliente finalizado: URL=http://localhost:80/apartamentos/6/localidades, Peticiones=15, Tiempo promedio=497.93333333333334, Tiempo mínimo=100, Tiempo máximo=938
Cliente finalizado: URL=http://localhost:80/apartamentos/6/localidades, Peticiones=15, Tiempo promedio=498.06666666666666, Tiempo mínimo=45, Tiempo máximo=1069
Cliente finalizado: URL=http://localhost:80/apartamentos/6/localidades, Peticiones=15, Tiempo promedio=498.73333333333335, Tiempo mínimo=149, Tiempo máximo=995
Cliente finalizado: URL=http://localhost:80/apartamentos/6/localidades, Peticiones=15, Tiempo promedio=499.2, Tiempo mínimo=57, Tiempo máximo=1069
Cliente finalizado: URL=http://localhost:80/apartamentos/6/localidades, Peticiones=15, Tiempo promedio=503.53333333333336, Tiempo mínimo=90, Tiempo máximo=969
Cliente finalizado: URL=http://localhost:80/apartamentos/6/localidades, Peticiones=15, Tiempo promedio=505.1333333333333, Tiempo mínimo=91, Tiempo máximo=963
Cliente finalizado: URL=http://localhost:80/apartamentos/6/localidades, Peticiones=15, Tiempo promedio=505.93333333333334, Tiempo mínimo=116, Tiempo máximo=1125
Cliente finalizado: URL=http://localhost:80/apartamentos/6/localidades, Peticiones=15, Tiempo promedio=507.46666666666664, Tiempo mínimo=63, Tiempo máximo=1007
Cliente finalizado: URL=http://localhost:80/apartamentos/6/localidades, Peticiones=15, Tiempo promedio=515.3333333333334, Tiempo mínimo=70, Tiempo máximo=1070
Cliente finalizado: URL=http://localhost:80/apartamentos/6/localidades, Peticiones=15, Tiempo promedio=515.4666666666667, Tiempo mínimo=59, Tiempo máximo=1069
Cliente finalizado: URL=http://localhost:80/apartamentos/6/localidades, Peticiones=15, Tiempo promedio=517.8, Tiempo mínimo=28, Tiempo máximo=1036
```

Figura 5-54 Tiempos GET/apartamentos/{codapt}/localidades con implementación tradicional y sin caché

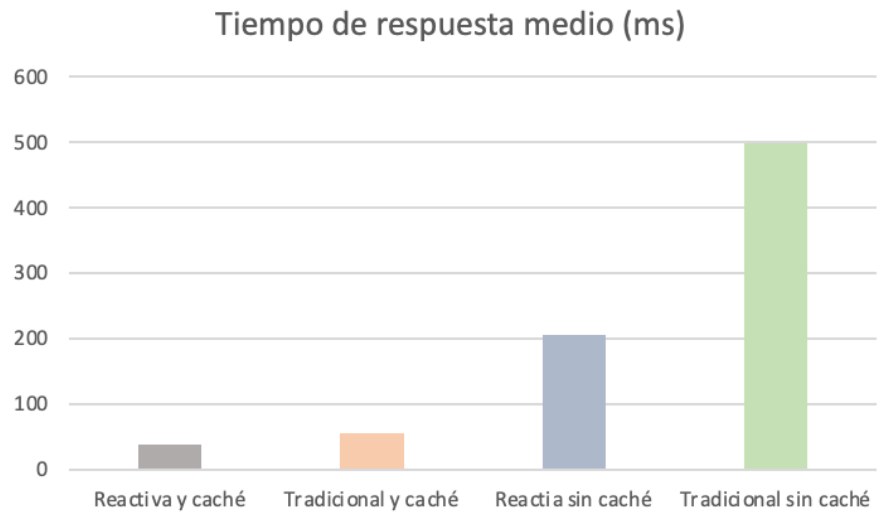

Figura 5-55 Gráfico tiempos de respuesta medios GET/apartamentos/{codapt}/localidades

Como se puede observar en el gráfico creado con los tiempos medios de las respuestas que otorgó el sistema en los diferentes formatos que se estudian, se puede concluir que el modelo reactivo es más rápido que el tradicional siempre que se compare en igualdad de condiciones. Esto significa que en las dos implementaciones con caché muestra un tiempo de respuesta menor el modelo reactivo, y en las dos implementaciones sin caché también. Además, se puede ver cómo, en este caso, los tiempos van aumentando de manera proporcional a medida que dejamos de utilizar elementos que ayudan a mejorar los tiempos de respuesta del sistema.

5.12 Comparación GET/localidades/{codloc}/apartamentos

```
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=38.13333333333333, Tiempo mínimo=5, Tiempo maximo=145
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=55.06666666666667, Tiempo mínimo=11, Tiempo maximo=112
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=50.13333333333333, Tiempo mínimo=14, Tiempo maximo=130
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=62.53333333333333, Tiempo mínimo=8, Tiempo maximo=188
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=62.93333333333333, Tiempo mínimo=17, Tiempo maximo=180
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=63.666666666666664, Tiempo mínimo=9, Tiempo maximo=164
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=64.33333333333333, Tiempo mínimo=18, Tiempo maximo=130
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=67.86666666666666, Tiempo mínimo=20, Tiempo maximo=166
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=68.6, Tiempo mínimo=26, Tiempo maximo=116
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=69.0, Tiempo mínimo=32, Tiempo maximo=160
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=68.86666666666666, Tiempo mínimo=28, Tiempo maximo=187
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=69.33333333333333, Tiempo mínimo=21, Tiempo maximo=138
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=69.6, Tiempo mínimo=33, Tiempo maximo=123
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=70.0, Tiempo mínimo=12, Tiempo maximo=136
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=70.6, Tiempo mínimo=35, Tiempo maximo=172
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=72.0, Tiempo mínimo=9, Tiempo maximo=168
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=72.26666666666667, Tiempo mínimo=18, Tiempo maximo=151
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=72.93333333333334, Tiempo mínimo=13, Tiempo maximo=191
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=73.26666666666667, Tiempo mínimo=15, Tiempo maximo=135
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=74.33333333333333, Tiempo mínimo=14, Tiempo maximo=165
```

Figura 5-56 Tiempos GET/localidades/{codloc}/apartamentos con implementación reactiva y caché

```
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=40.666666666666664, Tiempo mínimo=16, Tiempo maximo=125
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=37.2, Tiempo mínimo=20, Tiempo maximo=111
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=39.93333333333333, Tiempo mínimo=24, Tiempo maximo=126
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=40.4, Tiempo mínimo=25, Tiempo maximo=136
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=43.53333333333333, Tiempo mínimo=17, Tiempo maximo=154
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=43.733333333333334, Tiempo mínimo=17, Tiempo maximo=116
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=40.53333333333333, Tiempo mínimo=15, Tiempo maximo=119
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=37.86666666666667, Tiempo mínimo=15, Tiempo maximo=145
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=46.266666666666666, Tiempo mínimo=27, Tiempo maximo=133
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=46.8, Tiempo mínimo=13, Tiempo maximo=128
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=46.733333333333334, Tiempo mínimo=20, Tiempo maximo=136
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=47.733333333333334, Tiempo mínimo=15, Tiempo maximo=130
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=48.53333333333333, Tiempo mínimo=18, Tiempo maximo=120
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=48.53333333333333, Tiempo mínimo=14, Tiempo maximo=133
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=49.53333333333333, Tiempo mínimo=13, Tiempo maximo=119
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=49.8, Tiempo mínimo=21, Tiempo maximo=146
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=49.93333333333333, Tiempo mínimo=22, Tiempo maximo=153
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=50.266666666666666, Tiempo mínimo=8, Tiempo maximo=152
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=50.666666666666664, Tiempo mínimo=15, Tiempo maximo=137
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=50.8, Tiempo mínimo=5, Tiempo maximo=127
```

Figura 5-57 Tiempos GET/localidades/{codloc}/apartamentos con implementación tradicional y caché

```
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=226.26666666666668, Tiempo mínimo=55, Tiempo máximo=476
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=238.4, Tiempo mínimo=90, Tiempo máximo=466
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=249.0, Tiempo mínimo=114, Tiempo máximo=432
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=253.13333333333333, Tiempo mínimo=74, Tiempo máximo=436
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=253.13333333333333, Tiempo mínimo=96, Tiempo máximo=418
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=257.0, Tiempo mínimo=109, Tiempo máximo=393
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=259.93333333333334, Tiempo mínimo=127, Tiempo máximo=452
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=260.53333333333336, Tiempo mínimo=96, Tiempo máximo=448
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=256.6666666666667, Tiempo mínimo=44, Tiempo máximo=443
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=262.06666666666666, Tiempo mínimo=71, Tiempo máximo=448
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=263.6, Tiempo mínimo=53, Tiempo máximo=485
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=264.0, Tiempo mínimo=103, Tiempo máximo=450
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=265.4, Tiempo mínimo=134, Tiempo máximo=384
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=265.6, Tiempo mínimo=99, Tiempo máximo=441
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=267.8666666666667, Tiempo mínimo=85, Tiempo máximo=473
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=268.53333333333336, Tiempo mínimo=100, Tiempo máximo=444
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=270.26666666666665, Tiempo mínimo=65, Tiempo máximo=410
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=271.2, Tiempo mínimo=35, Tiempo máximo=517
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=271.8666666666667, Tiempo mínimo=104, Tiempo máximo=472
Cliente finalizado: URL=http://localhost:80/localidades/1/apartamentos, Peticiones=15, Tiempo promedio=273.1333333333333, Tiempo mínimo=43, Tiempo máximo=441
```

Figura 5-58 Tiempos GET/localidades/{codloc}/apartamentos con implementación reactiva y sin caché

```
Cliente finalizado: URL=http://localhost:80/localidades/5/apartamentos, Peticiones=15, Tiempo promedio=502.0, Tiempo mínimo=136, Tiempo máximo=1492
Cliente finalizado: URL=http://localhost:80/localidades/5/apartamentos, Peticiones=15, Tiempo promedio=532.2, Tiempo mínimo=81, Tiempo máximo=1485
Cliente finalizado: URL=http://localhost:80/localidades/5/apartamentos, Peticiones=15, Tiempo promedio=545.0666666666667, Tiempo mínimo=193, Tiempo máximo=1782
Cliente finalizado: URL=http://localhost:80/localidades/5/apartamentos, Peticiones=15, Tiempo promedio=557.2666666666667, Tiempo mínimo=133, Tiempo máximo=2514
Cliente finalizado: URL=http://localhost:80/localidades/5/apartamentos, Peticiones=15, Tiempo promedio=559.8, Tiempo mínimo=64, Tiempo máximo=2248
Cliente finalizado: URL=http://localhost:80/localidades/5/apartamentos, Peticiones=15, Tiempo promedio=570.4, Tiempo mínimo=162, Tiempo máximo=1884
Cliente finalizado: URL=http://localhost:80/localidades/5/apartamentos, Peticiones=15, Tiempo promedio=579.4, Tiempo mínimo=210, Tiempo máximo=3588
Cliente finalizado: URL=http://localhost:80/localidades/5/apartamentos, Peticiones=15, Tiempo promedio=594.6666666666666, Tiempo mínimo=120, Tiempo máximo=2285
Cliente finalizado: URL=http://localhost:80/localidades/5/apartamentos, Peticiones=15, Tiempo promedio=598.3333333333334, Tiempo mínimo=88, Tiempo máximo=4321
Cliente finalizado: URL=http://localhost:80/localidades/5/apartamentos, Peticiones=15, Tiempo promedio=600.0666666666667, Tiempo mínimo=140, Tiempo máximo=2786
Cliente finalizado: URL=http://localhost:80/localidades/5/apartamentos, Peticiones=15, Tiempo promedio=602.1333333333333, Tiempo mínimo=232, Tiempo máximo=2719
Cliente finalizado: URL=http://localhost:80/localidades/5/apartamentos, Peticiones=15, Tiempo promedio=607.6666666666666, Tiempo mínimo=203, Tiempo máximo=2765
Cliente finalizado: URL=http://localhost:80/localidades/5/apartamentos, Peticiones=15, Tiempo promedio=611.4, Tiempo mínimo=48, Tiempo máximo=3971
Cliente finalizado: URL=http://localhost:80/localidades/5/apartamentos, Peticiones=15, Tiempo promedio=632.9333333333333, Tiempo mínimo=92, Tiempo máximo=4946
Cliente finalizado: URL=http://localhost:80/localidades/5/apartamentos, Peticiones=15, Tiempo promedio=645.6666666666666, Tiempo mínimo=65, Tiempo máximo=5304
Cliente finalizado: URL=http://localhost:80/localidades/5/apartamentos, Peticiones=15, Tiempo promedio=651.8, Tiempo mínimo=85, Tiempo máximo=4688
Cliente finalizado: URL=http://localhost:80/localidades/5/apartamentos, Peticiones=15, Tiempo promedio=657.2666666666667, Tiempo mínimo=33, Tiempo máximo=7976
Cliente finalizado: URL=http://localhost:80/localidades/5/apartamentos, Peticiones=15, Tiempo promedio=667.6, Tiempo mínimo=55, Tiempo máximo=6803
Cliente finalizado: URL=http://localhost:80/localidades/5/apartamentos, Peticiones=15, Tiempo promedio=669.7333333333333, Tiempo mínimo=29, Tiempo máximo=7009
Cliente finalizado: URL=http://localhost:80/localidades/5/apartamentos, Peticiones=15, Tiempo promedio=670.2, Tiempo mínimo=37, Tiempo máximo=7892
```

Figura 5-59 Tiempos GET/localidades/{codloc}/apartamentos con implementación tradicional y sin caché

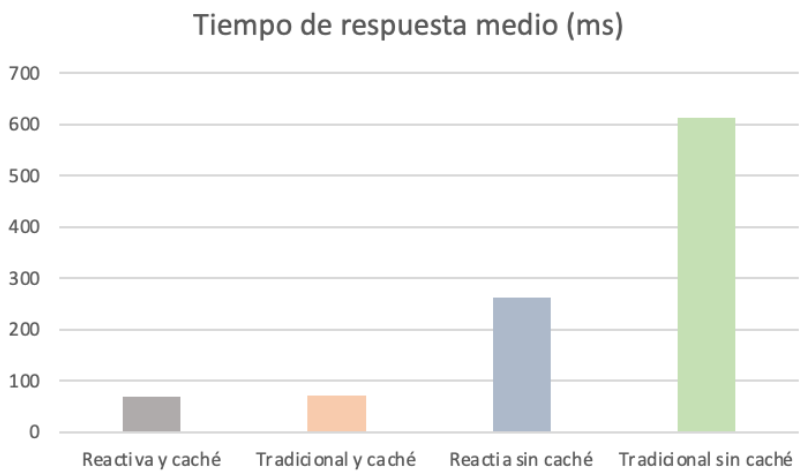

Figura 5-60 Gráfico tiempos de respuesta medios GET/localidades/{codloc}/apartamentos

Como se puede observar en el gráfico creado con los tiempos medios de las respuestas que otorgó el sistema en los diferentes formatos que se estudian, se puede concluir que el modelo reactivo es más rápido que el tradicional siempre que se compare en igualdad de condiciones. Esto significa que en las dos implementaciones con caché muestra un tiempo de respuesta menor el modelo reactivo, y en las dos implementaciones sin caché también. Además, se puede ver cómo, en este caso, los tiempos van aumentando de manera proporcional a medida que dejamos de utilizar elementos que ayudan a mejorar los tiempos de respuesta del sistema.

5.13 Comparación GET/localidades/{codloc}/apartamentos/{codapt}

Figura 5-61 Tiempos GET/localidades/{codloc}/apartamentos/{codloc} con implementación reactiva y caché

Figura 5-62 Tiempos GET/localidades/{codloc}/apartamentos/{codloc} con implementación tradicional y caché

Figura 5-63 Tiempos GET/localidades/{codloc}/apartamentos/{codapt} con implementación reactiva y sin caché

```
Cliente finalizado: URL=http://localhost:80/puntuaciones/157/apartamentos/151, Peticiones=15, Tiempo promedio=338.4, Tiempo mínimo=166, Tiempo máximo=613
Cliente finalizado: URL=http://localhost:80/puntuaciones/197/apartamentos/191, Peticiones=15, Tiempo promedio=347.93333333333334, Tiempo mínimo=188, Tiempo máximo=670
Cliente finalizado: URL=http://localhost:80/puntuaciones/117/apartamentos/111, Peticiones=15, Tiempo promedio=351.6666666666667, Tiempo mínimo=154, Tiempo máximo=676
Cliente finalizado: URL=http://localhost:80/puntuaciones/137/apartamentos/131, Peticiones=15, Tiempo promedio=359.73333333333335, Tiempo mínimo=171, Tiempo máximo=650
Cliente finalizado: URL=http://localhost:80/puntuaciones/97/apartamentos/91, Peticiones=15, Tiempo promedio=359.73333333333335, Tiempo mínimo=110, Tiempo máximo=845
Cliente finalizado: URL=http://localhost:80/puntuaciones/77/apartamentos/71, Peticiones=15, Tiempo promedio=364.06666666666666, Tiempo mínimo=144, Tiempo máximo=898
Cliente finalizado: URL=http://localhost:80/puntuaciones/57/apartamentos/51, Peticiones=15, Tiempo promedio=364.4, Tiempo mínimo=134, Tiempo máximo=613
Cliente finalizado: URL=http://localhost:80/puntuaciones/187/apartamentos/181, Peticiones=15, Tiempo promedio=364.3333333333333, Tiempo mínimo=201, Tiempo máximo=614
Cliente finalizado: URL=http://localhost:80/puntuaciones/167/apartamentos/161, Peticiones=15, Tiempo promedio=369.6666666666667, Tiempo mínimo=153, Tiempo máximo=550
Cliente finalizado: URL=http://localhost:80/puntuaciones/17/apartamentos/11, Peticiones=15, Tiempo promedio=373.1333333333333, Tiempo mínimo=152, Tiempo máximo=736
Cliente finalizado: URL=http://localhost:80/puntuaciones/27/apartamentos/21, Peticiones=15, Tiempo promedio=373.8666666666667, Tiempo mínimo=85, Tiempo máximo=753
Cliente finalizado: URL=http://localhost:80/puntuaciones/47/apartamentos/41, Peticiones=15, Tiempo promedio=373.26666666666665, Tiempo mínimo=105, Tiempo máximo=629
Cliente finalizado: URL=http://localhost:80/puntuaciones/67/apartamentos/61, Peticiones=15, Tiempo promedio=371.93333333333334, Tiempo mínimo=124, Tiempo máximo=746
Cliente finalizado: URL=http://localhost:80/puntuaciones/177/apartamentos/171, Peticiones=15, Tiempo promedio=371.73333333333335, Tiempo mínimo=91, Tiempo máximo=704
Cliente finalizado: URL=http://localhost:80/puntuaciones/127/apartamentos/121, Peticiones=15, Tiempo promedio=376.2, Tiempo mínimo=149, Tiempo máximo=729
Cliente finalizado: URL=http://localhost:80/puntuaciones/87/apartamentos/81, Peticiones=15, Tiempo promedio=376.26666666666665, Tiempo mínimo=139, Tiempo máximo=780
Cliente finalizado: URL=http://localhost:80/puntuaciones/147/apartamentos/141, Peticiones=15, Tiempo promedio=377.46666666666664, Tiempo mínimo=21, Tiempo máximo=898
Cliente finalizado: URL=http://localhost:80/puntuaciones/107/apartamentos/101, Peticiones=15, Tiempo promedio=378.1333333333333, Tiempo mínimo=124, Tiempo máximo=750
Cliente finalizado: URL=http://localhost:80/puntuaciones/97/apartamentos/91, Peticiones=15, Tiempo promedio=380.2, Tiempo mínimo=109, Tiempo máximo=667
Cliente finalizado: URL=http://localhost:80/puntuaciones/37/apartamentos/31, Peticiones=15, Tiempo promedio=381.1333333333333, Tiempo mínimo=121, Tiempo máximo=833
```

Figura 5-64 Tiempos GET/localidades/{codloc}/apartamentos/{codapt} con implementación tradicional y sin caché

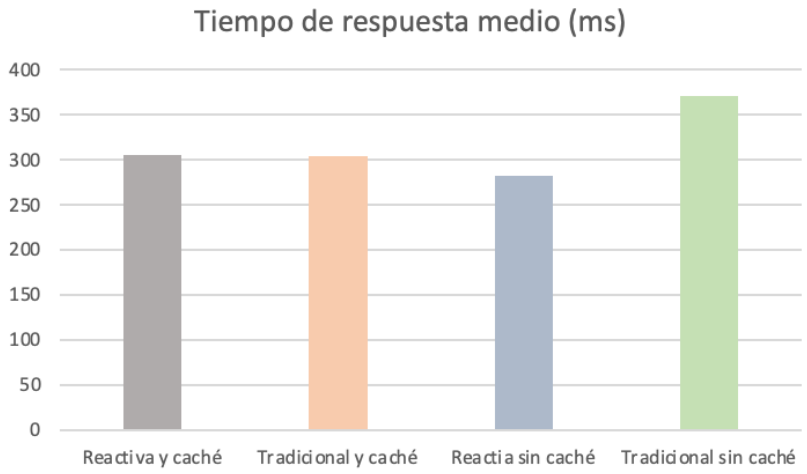

Figura 5-65 Gráfico tiempos de respuesta medios GET/localidades/{codloc}/apartamentos/{codapt}

Como se puede observar en el gráfico creado con los tiempos medios de las respuestas que otorgó el sistema en los diferentes formatos que se estudian, se puede concluir que el modelo reactivo es más rápido que el tradicional siempre que se compare en igualdad de condiciones. Esto significa que en las dos implementaciones con caché muestra un tiempo de respuesta muy similar, sin poder apreciar diferencia prácticamente, y en las dos implementaciones sin caché muestra una gran diferencia del reactivo respecto al tradicional. Como caso extraordinario, es más rápida la implementación sin caché que con caché.

5.14 Comparación POST/localidades/{codloc}/apartamentos/{codapt}

```
Cliente finalizado: URL=http://localhost:80/localidades/37/apartamentos/31, Peticiones=15, Tiempo promedio=242.0, Tiempo mínimo=86, Tiempo máximo=706
Cliente finalizado: URL=http://localhost:80/localidades/87/apartamentos/81, Peticiones=15, Tiempo promedio=322.1333333333333, Tiempo mínimo=94, Tiempo máximo=818
Cliente finalizado: URL=http://localhost:80/localidades/167/apartamentos/161, Peticiones=15, Tiempo promedio=353.0, Tiempo mínimo=27, Tiempo máximo=838
Cliente finalizado: URL=http://localhost:80/localidades/97/apartamentos/91, Peticiones=15, Tiempo promedio=354.06666666666666, Tiempo mínimo=41, Tiempo máximo=752
Cliente finalizado: URL=http://localhost:80/localidades/57/apartamentos/51, Peticiones=15, Tiempo promedio=365.1333333333333, Tiempo mínimo=132, Tiempo máximo=718
Cliente finalizado: URL=http://localhost:80/localidades/197/apartamentos/191, Peticiones=15, Tiempo promedio=367.26666666666665, Tiempo mínimo=96, Tiempo máximo=952
Cliente finalizado: URL=http://localhost:80/localidades/87/apartamentos/81, Peticiones=15, Tiempo promedio=370.6666666666667, Tiempo mínimo=62, Tiempo máximo=1406
Cliente finalizado: URL=http://localhost:80/localidades/187/apartamentos/181, Peticiones=15, Tiempo promedio=373.8666666666667, Tiempo mínimo=76, Tiempo máximo=1107
Cliente finalizado: URL=http://localhost:80/localidades/137/apartamentos/131, Peticiones=15, Tiempo promedio=379.2, Tiempo mínimo=59, Tiempo máximo=1312
Cliente finalizado: URL=http://localhost:80/localidades/47/apartamentos/41, Peticiones=15, Tiempo promedio=385.46666666666664, Tiempo mínimo=95, Tiempo máximo=1122
Cliente finalizado: URL=http://localhost:80/localidades/127/apartamentos/121, Peticiones=15, Tiempo promedio=388.73333333333335, Tiempo mínimo=43, Tiempo máximo=728
Cliente finalizado: URL=http://localhost:80/localidades/117/apartamentos/111, Peticiones=15, Tiempo promedio=388.93333333333334, Tiempo mínimo=27, Tiempo máximo=929
Cliente finalizado: URL=http://localhost:80/localidades/67/apartamentos/61, Peticiones=15, Tiempo promedio=395.46666666666664, Tiempo mínimo=62, Tiempo máximo=1017
Cliente finalizado: URL=http://localhost:80/localidades/177/apartamentos/171, Peticiones=15, Tiempo promedio=395.3333333333333, Tiempo mínimo=91, Tiempo máximo=1109
Cliente finalizado: URL=http://localhost:80/localidades/17/apartamentos/11, Peticiones=15, Tiempo promedio=397.4, Tiempo mínimo=69, Tiempo máximo=1011
Cliente finalizado: URL=http://localhost:80/localidades/27/apartamentos/21, Peticiones=15, Tiempo promedio=401.06666666666666, Tiempo mínimo=48, Tiempo máximo=1588
Cliente finalizado: URL=http://localhost:80/localidades/147/apartamentos/141, Peticiones=15, Tiempo promedio=401.4, Tiempo mínimo=88, Tiempo máximo=1212
Cliente finalizado: URL=http://localhost:80/localidades/77/apartamentos/71, Peticiones=15, Tiempo promedio=402.6, Tiempo mínimo=41, Tiempo máximo=1229
Cliente finalizado: URL=http://localhost:80/localidades/107/apartamentos/101, Peticiones=15, Tiempo promedio=403.1333333333333, Tiempo mínimo=59, Tiempo máximo=1117
Cliente finalizado: URL=http://localhost:80/localidades/157/apartamentos/151, Peticiones=15, Tiempo promedio=408.1333333333333, Tiempo mínimo=17, Tiempo máximo=1066
```

Figura 5-66 Tiempos POST/localidades/{codloc}/apartamentos/{codapt} con implementación reactiva y caché

```
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=580.53333333333336, Tiempo mínimo=126, Tiempo máximo=2425
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=512.2666666666667, Tiempo mínimo=163, Tiempo máximo=2226
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=514.1333333333333, Tiempo mínimo=166, Tiempo máximo=2158
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=516.8, Tiempo mínimo=222, Tiempo máximo=2191
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=517.2, Tiempo mínimo=267, Tiempo máximo=2216
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=521.9333333333333, Tiempo mínimo=229, Tiempo máximo=2139
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=521.6666666666666, Tiempo mínimo=179, Tiempo máximo=2123
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=534.5333333333333, Tiempo mínimo=168, Tiempo máximo=2568
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=535.9333333333333, Tiempo mínimo=198, Tiempo máximo=2088
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=538.4, Tiempo mínimo=164, Tiempo máximo=2410
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=540.2, Tiempo mínimo=196, Tiempo máximo=2477
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=542.0666666666667, Tiempo mínimo=191, Tiempo máximo=2304
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=544.2, Tiempo mínimo=223, Tiempo máximo=2324
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=544.4, Tiempo mínimo=199, Tiempo máximo=2087
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=545.6666666666666, Tiempo mínimo=213, Tiempo máximo=2396
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=545.7333333333333, Tiempo mínimo=132, Tiempo máximo=2479
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=547.2, Tiempo mínimo=208, Tiempo máximo=2164
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=548.9333333333333, Tiempo mínimo=88, Tiempo máximo=2112
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=550.8, Tiempo mínimo=54, Tiempo máximo=2478
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=552.8666666666667, Tiempo mínimo=56, Tiempo máximo=2255
```

Figura 5-67 Tiempos POST/localidades/{codloc}/apartamentos/{codapt} con implementación tradicional y caché

```
Cliente finalizado: URL=http://localhost:80/localidades/137/apartamentos/131, Peticiones=15, Tiempo promedio=311.8, Tiempo mínimo=34, Tiempo máximo=560
Cliente finalizado: URL=http://localhost:80/localidades/47/apartamentos/41, Peticiones=15, Tiempo promedio=338.8, Tiempo mínimo=119, Tiempo máximo=648
Cliente finalizado: URL=http://localhost:80/localidades/177/apartamentos/171, Peticiones=15, Tiempo promedio=352.3333333333333, Tiempo mínimo=55, Tiempo máximo=927
Cliente finalizado: URL=http://localhost:80/localidades/107/apartamentos/101, Peticiones=15, Tiempo promedio=363.2, Tiempo mínimo=36, Tiempo máximo=893
Cliente finalizado: URL=http://localhost:80/localidades/187/apartamentos/181, Peticiones=15, Tiempo promedio=363.2, Tiempo mínimo=27, Tiempo máximo=733
Cliente finalizado: URL=http://localhost:80/localidades/167/apartamentos/161, Peticiones=15, Tiempo promedio=363.6666666666667, Tiempo mínimo=87, Tiempo máximo=632
Cliente finalizado: URL=http://localhost:80/localidades/127/apartamentos/121, Peticiones=15, Tiempo promedio=378.53333333333336, Tiempo mínimo=80, Tiempo máximo=994
Cliente finalizado: URL=http://localhost:80/localidades/67/apartamentos/61, Peticiones=15, Tiempo promedio=381.6666666666667, Tiempo mínimo=30, Tiempo máximo=1102
Cliente finalizado: URL=http://localhost:80/localidades/27/apartamentos/21, Peticiones=15, Tiempo promedio=383.8, Tiempo mínimo=98, Tiempo máximo=725
Cliente finalizado: URL=http://localhost:80/localidades/17/apartamentos/11, Peticiones=15, Tiempo promedio=384.53333333333336, Tiempo mínimo=64, Tiempo máximo=1034
Cliente finalizado: URL=http://localhost:80/localidades/157/apartamentos/151, Peticiones=15, Tiempo promedio=385.26666666666665, Tiempo mínimo=82, Tiempo máximo=865
Cliente finalizado: URL=http://localhost:80/localidades/77/apartamentos/71, Peticiones=15, Tiempo promedio=388.26666666666665, Tiempo mínimo=54, Tiempo máximo=768
Cliente finalizado: URL=http://localhost:80/localidades/37/apartamentos/31, Peticiones=15, Tiempo promedio=390.8, Tiempo mínimo=126, Tiempo máximo=781
Cliente finalizado: URL=http://localhost:80/localidades/87/apartamentos/81, Peticiones=15, Tiempo promedio=390.1333333333333, Tiempo mínimo=102, Tiempo máximo=848
Cliente finalizado: URL=http://localhost:80/localidades/97/apartamentos/91, Peticiones=15, Tiempo promedio=391.8, Tiempo mínimo=77, Tiempo máximo=989
Cliente finalizado: URL=http://localhost:80/localidades/57/apartamentos/51, Peticiones=15, Tiempo promedio=392.8666666666667, Tiempo mínimo=38, Tiempo máximo=856
Cliente finalizado: URL=http://localhost:80/localidades/117/apartamentos/111, Peticiones=15, Tiempo promedio=393.26666666666665, Tiempo mínimo=125, Tiempo máximo=837
Cliente finalizado: URL=http://localhost:80/localidades/197/apartamentos/191, Peticiones=15, Tiempo promedio=394.8666666666667, Tiempo mínimo=43, Tiempo máximo=1087
Cliente finalizado: URL=http://localhost:80/localidades/147/apartamentos/141, Peticiones=15, Tiempo promedio=395.8666666666667, Tiempo mínimo=104, Tiempo máximo=1098
Cliente finalizado: URL=http://localhost:80/localidades/87/apartamentos/81, Peticiones=15, Tiempo promedio=397.6, Tiempo mínimo=60, Tiempo máximo=1189
```

Figura 5-68 Tiempos POST/localidades/{codloc}/apartamentos/{codapt} con implementación reactiva y sin caché

```
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=561.0, Tiempo mínimo=200, Tiempo máximo=1973
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=561.1333333333333, Tiempo mínimo=227, Tiempo máximo=1799
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=552.4666666666667, Tiempo mínimo=229, Tiempo máximo=1641
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=567.2, Tiempo mínimo=286, Tiempo máximo=1658
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=567.6, Tiempo mínimo=313, Tiempo máximo=1618
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=571.5333333333333, Tiempo mínimo=251, Tiempo máximo=2432
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=573.2, Tiempo mínimo=287, Tiempo máximo=2219
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=573.8, Tiempo mínimo=297, Tiempo máximo=2139
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=578.0666666666667, Tiempo mínimo=330, Tiempo máximo=1977
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=579.6666666666666, Tiempo mínimo=212, Tiempo máximo=1873
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=580.9333333333333, Tiempo mínimo=233, Tiempo máximo=1867
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=581.3333333333334, Tiempo mínimo=234, Tiempo máximo=2360
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=584.2, Tiempo mínimo=274, Tiempo máximo=2105
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=585.2, Tiempo mínimo=239, Tiempo máximo=1741
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=588.6, Tiempo mínimo=134, Tiempo máximo=2520
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=590.2, Tiempo mínimo=162, Tiempo máximo=2105
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=592.4, Tiempo mínimo=113, Tiempo máximo=2593
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=594.1333333333333, Tiempo mínimo=110, Tiempo máximo=2577
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=595.4666666666667, Tiempo mínimo=111, Tiempo máximo=2594
Cliente finalizado: URL=http://localhost:80/puntuaciones, Peticiones=15, Tiempo promedio=596.0666666666667, Tiempo mínimo=73, Tiempo máximo=2247
```

Figura 5-69 Tiempos POST/localidades/{codoc}/apartamentos/{codapt} con implementación tradicional y sin caché

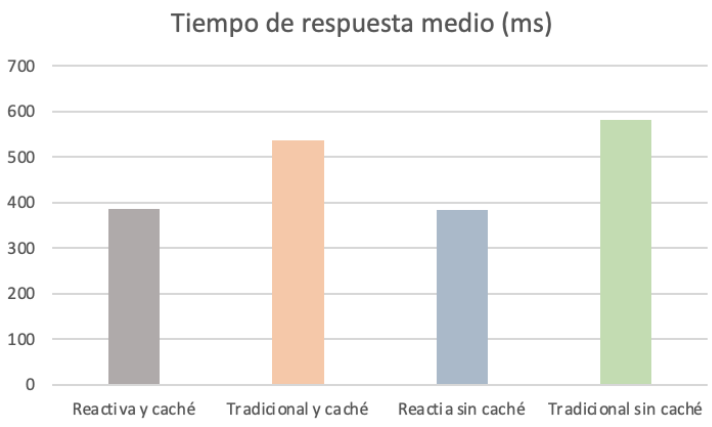

Figura 5-70 Gráfico tiempos de respuesta medios POST/localidades/{codloc}/apartamentos/{codapt}

Como se puede observar en el gráfico creado con los tiempos medios de las respuestas que otorgó el sistema en los diferentes formatos que se estudian, se puede concluir que el modelo reactivo es más rápido que el tradicional siempre que se compare en igualdad de condiciones. Esto significa que en las dos implementaciones con caché muestra un tiempo de respuesta menor el modelo reactivo, y en las dos implementaciones sin caché también. Además, en este caso los tiempos del modelo reactivo no varían mucho aun eliminando el caché, lo que indica que sin caché el sistema sigue ofreciendo un tiempo de respuesta muy bueno.

5.15 Comparación PUT/localidades/{codloc}/apartamentos/{codapt}

Figura 5-71 Tiempos PUT/localidades/{codloc}/apartamentos/{codapt} con implementación reactiva y caché

Figura 5-72 Tiempos PUT/localidades/{codloc}/apartamentos/{codapt} con implementación tradicional y caché

Figura 5-73 Tiempos PUT/localidades/{codloc}/apartamentos/{codapt} con implementación reactiva y sin caché

```
Cliente finalizado: URL=http://localhost:80/localidades/87/apartamentos/81, Peticiones=15, Tiempo promedio=258.53333333333336, Tiempo mínimo=71, Tiempo máximo=479
Cliente finalizado: URL=http://localhost:80/localidades/147/apartamentos/141, Peticiones=15, Tiempo promedio=271.73333333333335, Tiempo mínimo=32, Tiempo máximo=632
Cliente finalizado: URL=http://localhost:80/localidades/187/apartamentos/181, Peticiones=15, Tiempo promedio=285.1333333333333, Tiempo mínimo=120, Tiempo máximo=535
Cliente finalizado: URL=http://localhost:80/localidades/107/apartamentos/101, Peticiones=15, Tiempo promedio=287.8, Tiempo mínimo=104, Tiempo máximo=707
Cliente finalizado: URL=http://localhost:80/localidades/167/apartamentos/161, Peticiones=15, Tiempo promedio=291.26666666666665, Tiempo mínimo=83, Tiempo máximo=585
Cliente finalizado: URL=http://localhost:80/localidades/17/apartamentos/11, Peticiones=15, Tiempo promedio=292.3333333333333, Tiempo mínimo=109, Tiempo máximo=551
Cliente finalizado: URL=http://localhost:80/localidades/77/apartamentos/71, Peticiones=15, Tiempo promedio=295.53333333333336, Tiempo mínimo=50, Tiempo máximo=693
Cliente finalizado: URL=http://localhost:80/localidades/87/apartamentos/81, Peticiones=15, Tiempo promedio=296.8666666666667, Tiempo mínimo=36, Tiempo máximo=714
Cliente finalizado: URL=http://localhost:80/localidades/67/apartamentos/61, Peticiones=15, Tiempo promedio=296.2, Tiempo mínimo=110, Tiempo máximo=578
Cliente finalizado: URL=http://localhost:80/localidades/157/apartamentos/151, Peticiones=15, Tiempo promedio=296.8666666666667, Tiempo mínimo=80, Tiempo máximo=612
Cliente finalizado: URL=http://localhost:80/localidades/117/apartamentos/111, Peticiones=15, Tiempo promedio=297.2, Tiempo mínimo=64, Tiempo máximo=678
Cliente finalizado: URL=http://localhost:80/localidades/57/apartamentos/51, Peticiones=15, Tiempo promedio=300.0, Tiempo mínimo=87, Tiempo máximo=531
Cliente finalizado: URL=http://localhost:80/localidades/137/apartamentos/131, Peticiones=15, Tiempo promedio=300.8, Tiempo mínimo=109, Tiempo máximo=581
Cliente finalizado: URL=http://localhost:80/localidades/97/apartamentos/91, Peticiones=15, Tiempo promedio=303.8666666666667, Tiempo mínimo=87, Tiempo máximo=542
Cliente finalizado: URL=http://localhost:80/localidades/27/apartamentos/21, Peticiones=15, Tiempo promedio=305.3333333333333, Tiempo mínimo=104, Tiempo máximo=589
Cliente finalizado: URL=http://localhost:80/localidades/197/apartamentos/191, Peticiones=15, Tiempo promedio=304.53333333333336, Tiempo mínimo=34, Tiempo máximo=529
Cliente finalizado: URL=http://localhost:80/localidades/47/apartamentos/41, Peticiones=15, Tiempo promedio=307.53333333333336, Tiempo mínimo=83, Tiempo máximo=570
Cliente finalizado: URL=http://localhost:80/localidades/127/apartamentos/121, Peticiones=15, Tiempo promedio=310.26666666666665, Tiempo mínimo=60, Tiempo máximo=620
Cliente finalizado: URL=http://localhost:80/localidades/37/apartamentos/31, Peticiones=15, Tiempo promedio=309.8, Tiempo mínimo=39, Tiempo máximo=592
Cliente finalizado: URL=http://localhost:80/localidades/177/apartamentos/171, Peticiones=15, Tiempo promedio=312.0, Tiempo mínimo=47, Tiempo máximo=718
```

Figura 5-74 Tiempos PUT/localidades/{codloc}/apartamentos/{codapt} con implementación tradicional y sin caché

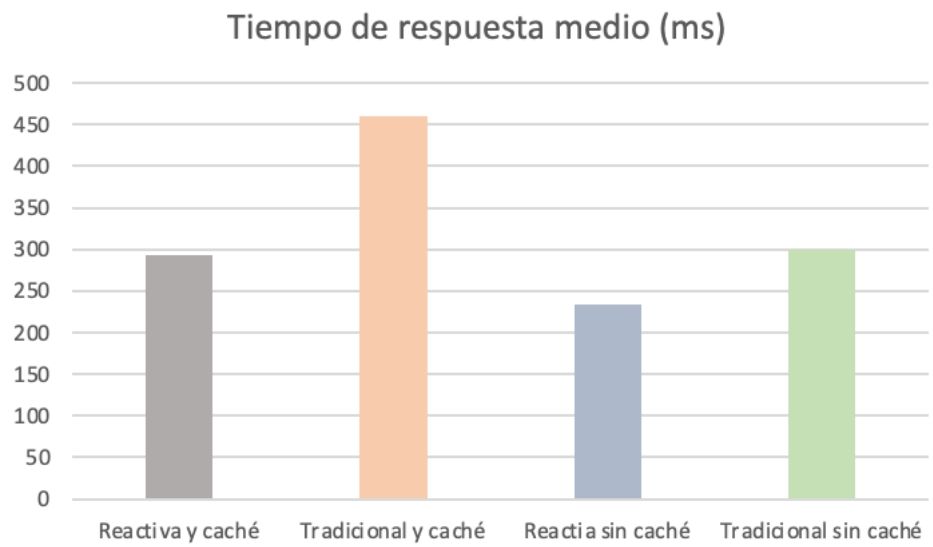

Figura 5-75 Gráfico tiempos de respuesta medios PUT/localidades/{codloc}/apartamentos/{codapt}

Como se puede observar en el gráfico creado con los tiempos medios de las respuestas que otorgó el sistema en los diferentes formatos que se estudian, se puede concluir que el modelo reactivo es más rápido que el tradicional siempre que se compare en igualdad de condiciones. Esto significa que en las dos implementaciones con caché muestra un tiempo de respuesta menor el modelo reactivo, y en las dos implementaciones sin caché también. Además, se puede ver cómo, al ser un método PUT, se obtiene un tiempo de respuesta menor en los modelos sin caché ya que con caché no se brinda un acceso más rápido a los datos, sino que hay que modificar los que ya se encuentran disponibles, por tanto, se pierde un poco más de tiempo.

Aun así, el porcentaje de solicitudes de este estilo es bastante menor con el de solicitudes GET, por lo que en el sistema en general sigue siendo beneficioso el uso de caché.

5.16 Comparación DELETE/localidades/{codloc}/apartamentos/{codapt}

Figura 5-76 Tiempos DELETE/localidades/{codloc}/apartamentos/{codapt} con implementación reactiva y caché

Figura 5-77 Tiempos DELETE/localidades/{codloc}/apartamentos/{codpat} con implementación tradicional y caché

Figura 5-78 Tiempos DELETE/localidades/{codloc}/apartamentos/{codapt} con implementación reactiva y sin caché

Figura 5-79 Tiempos DELETE/localidades/{codloc}/apartamentos/{codapt} con implementación tradicional y sin caché

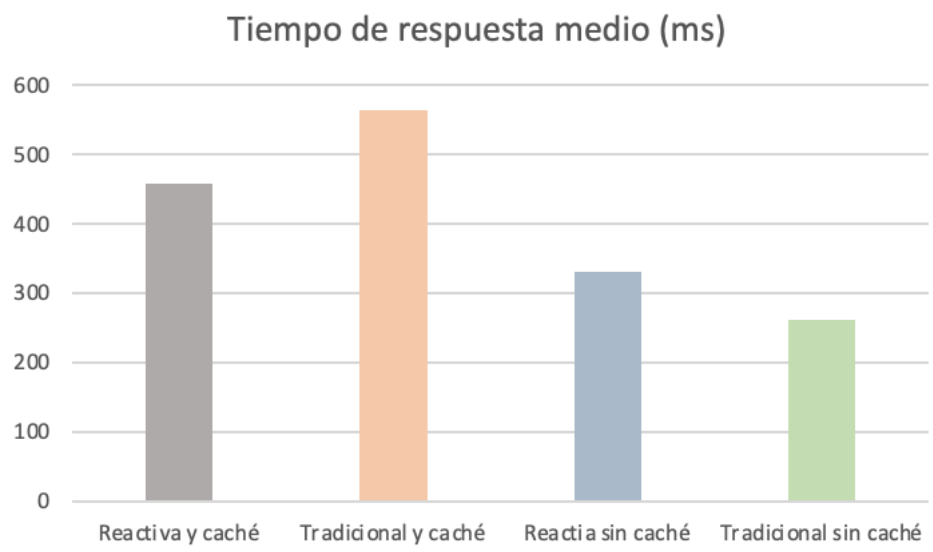

Figura 5-80 Gráfico tiempos de respuesta medios DELETE/localidades/{codloc}/apartamentos/{codapt}

Como se puede observar en el gráfico creado con los tiempos medios de las respuestas que otorgó el sistema en los diferentes formatos que se estudian, se puede concluir que el modelo reactivo es más rápido que el tradicional en la implementación con caché, sin embargo, en este caso concreto, en la implementación sin caché es un poco más rápido el tradicional. Además, se puede ver cómo, al ser un método DELETE, se obtiene un tiempo de respuesta menor en los modelos sin caché ya que con caché no se brinda un acceso más rápido a los datos, sino que hay que eliminar los que ya se encuentran disponibles, por tanto, se pierde un poco más de tiempo.

Aun así, el porcentaje de solicitudes de este estilo es bastante menor con el de solicitudes GET, por lo que en el sistema en general sigue siendo beneficioso el uso de caché.

6. Conclusiones

A lo largo de este trabajo sobre la programación reactiva y tras el desarrollo de dos aplicaciones, una reactiva y otra tradicional, se ha demostrado claramente que la programación reactiva ofrece una mayor eficiencia en comparación con los enfoques tradicionales.

La aplicación reactiva, basada en los principios de programación reactiva, ha mostrado una capacidad excepcional para manejar la complejidad y los cambios constantes en los sistemas modernos. Su enfoque en la propagación de cambios y la gestión asincrónica de eventos ha permitido una mayor escalabilidad y una mejor respuesta en tiempo real.

Por otro lado, la aplicación tradicional ha seguido un enfoque más secuencial y basado en la ejecución de tareas sincrónicas. Si bien puede ser adecuada para aplicaciones más simples y lineales, ha demostrado dificultades al lidiar con la complejidad creciente y los requisitos de tiempo real. La falta de capacidad de respuesta y la necesidad de reevaluar constantemente el estado de la aplicación han llevado a una menor eficiencia y rendimiento.

La programación reactiva, por su parte, ha abordado estos desafíos mediante la introducción de conceptos como flujos de datos y observables, que permiten una gestión más eficiente de los eventos y una mejor coordinación de las tareas en tiempo real. Además, la capacidad de reaccionar automáticamente ante los cambios y propagarlos a través del sistema ha demostrado ser crucial para garantizar una aplicación robusta y adaptable.

En resumen, la programación reactiva ha mostrado una clara superioridad en términos de eficiencia y capacidad de respuesta en comparación con los enfoques tradicionales. Su enfoque en la gestión de eventos y la propagación de cambios ha permitido una mayor escalabilidad y adaptabilidad, lo que la convierte en una opción altamente recomendada para el desarrollo de aplicaciones modernas y complejas.

Bibliografía

1. Google. PostgreSQL y SQL: ¿cuáles son las diferencias clave? [En línea] https://cloud.google.com/learn/postgresql-vs-sql?hl=es.

2. Los principios reactivos. [En línea] https://www.reactiveprinciples.org.

3. Arquitectura reactiva de Quarkus. [En línea] https://quarkus.io/guides/quarkus-reactive-architecture.

4. El manifiesto reactivo. [En línea] 2014. https://www.reactivemanifesto.org.

5. Mutiny - Asíncrono para mortales. [En línea] https://quarkus.io/guides/mutiny-primer.

6. Uni y Multi. [En línea] https://smallrye.io/smallrye-mutiny/2.1.0/reference/uni-and-multi/.

7. Primeros pasos con Mutiny. [En línea] https://smallrye.io/smallrye-mutiny/2.1.0/tutorials/getting-mutiny/.

8. Primeros pasos con reactivo. [En línea] https://quarkus.io/guides/getting-started-reactive.

9. Clientes SQL reactivos. [En línea] https://quarkus.io/guides/reactive-sql-clients.

10. Uso del REST Client. [En línea] https://quarkus.io/guides/rest-client-reactive.

11. Uso de Redis Client. [En línea] https://quarkus.io/guides/redis.

12. Escribir servicios REST con RESTEasy Reactive. [En línea] https://quarkus.io/guides/resteasy-reactive.

13. Caché Redis. [En línea] https://quarkus.io/guides/cache-redis-reference.

14. Urma, Raoul-Gabriel, Fusco, Mario y Mycroft, Alan. *Modern Java in Action: Lambdas, streams, functional and reactive programming.* s.l. : Manning Publications, 2018.

15. Urma, Raoul-Gabriel, Fusco, Mario y Mycroft, Alan. *Java 8 in Action: Lambdas, Streams, and functional-style programming.* s.l. : Manning Publications, 2014.