UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Análisis comparativo del rendimiento de frameworks para el desarrollo backend con diferentes tecnologías

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Sarabia Chamero, Jaime

Tutor/a: Poza Luján, José Luis

Cotutor/a: Posadas Yagüe, Juan Luís

Cotutor/a externo: RIBES PASTOR, PAU

CURSO ACADÉMICO: 2022/2023

Agradecimientos

A mi familia por darme una educación y ayudarme a llegar hasta aquí.

A mis amigos por acompañarme desde el principio y confiar en mi.

A mi tutor en la empresa, Pau, por enseñarme todo lo aprendido para realizar este proyecto.

Y a ti, Julia, por apoyarme hasta en los momentos más difíciles y ayudarme siempre a seguir a delante.

A todos, gracias por todo.

Resumen

El objetivo de este trabajo de fin de grado es realizar una comparación exhaustiva de cuatro frameworks utilizados en el desarrollo de aplicaciones web y servicios REST, con el fin de identificar cuál de ellos ofrece un mejor rendimiento en términos de velocidad, capacidad de respuesta y escalabilidad. Esta comparación es importante porque la elección del framework adecuado puede tener un gran impacto en el rendimiento y la eficiencia de una aplicación. Además, en un entorno empresarial cada vez más competitivo, es crucial maximizar el rendimiento de las aplicaciones para proporcionar una experiencia de usuario de alta calidad y asegurar la satisfacción del cliente. A través de pruebas de carga y disponibilidad, se podrán comparar los distintos aspectos de rendimiento de cada framework, lo que permitirá a los desarrolladores tomar una decisión informada sobre qué framework utilizar en función de sus necesidades específicas. Para redactar el desarrollo y resultados de este proyecto, se ha utilizado la Norma CCII-N2016-2 para la realización de la Documentación de Proyectos en Ingeniería Informática.

Palabras clave: REST; framework REST; rendimiento; benchmarking

Resum

L'objectiu d'aquest trevall de fi de grau es realitzar una comparació exhaustiva de quatre frameworks utilitzats en el desenvolupament d'aplicacions web i servicis REST, amb la fi d'identificar quin es el que ofereix un millor rendiment en termes de velocitat, capacitat de resposta i escalabilitat. Aquesta comparació es important perque l'elecció del framework adequat pot tindre un gran impacte al rendiment i l'eficència d'una aplicació. A més, en un entorn empresarial cada vegada mes competitiu, es crucial maximitzar el rendiment de les aplicacions per tal de proporcionar una experiència d'usuari d'alta qualitat i assegurar la satisfacció del client. A través de proves de càrrega y disponibilitat, es podran comparar els distints aspectes de rendiment de cada framework, el que permetrà als desenvolupadors prendre una decisió informada sobre qué framework utilitzar en funció de les seues necessitats específiques. Per a redactar el desenvolupament i els resultats d'aquest projecte, s'ha utilizat la "Norma CCII-N2016-2 "per la realització de la Documentació de Projectes en Enginyeria Informàtica.

Palabras clave: REST; framework REST; rendiment; benchmarking

Abstract

This Bachelor's thesis aims to comprehensively compare four frameworks used in the development of web applications and REST services to identify which offers better performance in terms of speed, responsiveness, and scalability. This comparison is critical because choosing the proper framework can significantly impact the performance and efficiency of an application. In an increasingly competitive business environment, it is crucial to maximise application performance to provide a high-quality user experience and ensure customer satisfaction. Through load and availability testing, the different performance aspects of each framework can be compared, allowing developers to make an informed decision about which framework to use based on their specific needs. To write the development and results of this project, the " Norma CCII-N2016-2 "has been used for documentation of computer engineering projects.

Palabras clave: REST; framework REST; performance; benchmarking

Índice general

Índice de figuras

Índice de tablas

Parte I

MEMORIA

Capítulo 1

Introducción

A medida que las nuevas tecnologías avanzan y se popularizan en todos los aspectos de la sociedad, estas tratan de mejorar la eficiencia de los procesos al optimizar las tareas. Para ello, se pueden utilizar herramientas de distintos ámbitos. En el caso de la ingeniería informática, el principal objetivo de los informáticos es optimizar la tarea del desarrollo del código para así poder emplear más tiempo de su trabajo en generar la idea como concepto y, no tanto, en como ejecutar la misma.
Los frameworks son entornos de trabajo que facilitan el desarrollo software al estandarizar ciertos procesos. El uso de los mismos comienza a popularizarse por la necesidad explicada anteriormente y facilitan el trabajo del programador, al evitar escritura repetitiva de código y mantener una consistencia en él.
Existen casi tantos frameworks como aplicaciones web, por tanto, este proyecto viene motivado por la necesidad de optimizar el tiempo que una empresa va a emplear en el desarrollo de un proyecto eligiendo el framework correcto, dependiendo de las necesidades del mismo. Comparar frameworks según su rendimiento, permitirá que la empresa optimice el tiempo empleado en cada proyecto con una visión a futuro. Para realizar un análisis adecuado, se deben determinar unos parámetros que permitan que la comparación entre los frameworks tenga una base en común.
Además de la propia motivación empresarial de este proyecto, la cuál viene dada por la duda de si se estaba utilizando la mejor opción para el desarrollo de los proyectos de esta, se suma una motivación de interés personal con el fin de descubrir más sobre el funcionamiento de estos entornos, que facilitan tanto el trabajo a los desarrolladores de software [10] [9].

Capítulo 2

Objeto del proyecto

Este proyecto busca analizar el rendimiento de cuatro de los frameworks mas utilizados en el mercado, para tratar de obtener así, resultados que determinen como reaccionan estos al ser utilizados, con el fin de comprobar cuál de ellos tiene un rendimiento mas adecuado para el uso empresarial. De esta forma, los proyectos posteriores realizados en la empresa, serán desarrollados teniendo en cuenta los resultados obtenidos en esta comparación, para tratar de utilizar así el entorno con mejor rendimiento, y poder trabajar con las mejores prestaciones posibles.

Para ello, se desarrollarán cuatro aplicaciones del mismo tipo: mismos requisitos y mismas funcionalidades, dos en Java, que serán desarrolladas con los frameworks Spring y Quarkus; y otras dos en Javascript, utilizando Express y Koa.

Los detalles sobre las pruebas de rendimiento y las aplicaciones desarrolladas serán explicadas en el Anexo 16.4.

Capítulo 3

Antecedentes

En los últimos años, el uso de frameworks para agilizar el desarrollo de aplicaciones se ha ido popularizando, y a día de hoy, es raro ver una empresa que no los utilice en sus proyectos. Hasta ahora, la mayoría de empresas había utilizado estos frameworks sin tener en cuenta el rendimiento o las prestaciones que pueden otorgar, y simplemente se guiaban por temas como popularidad, cantidad de librerías propias de la herramienta, facilidad de aprendizaje o usabilidad.

La empresa NTT DATA, más en concreto, el departamento de arquitectura de software, lleva utilizando este tipo de herramientas durante los últimos años para todo tipo de desarrollos en diferentes lenguajes, y la decisión de cual de estos frameworks utilizar la tomaban por criterios meramente subjetivos: de cual se tiene más información, cuál es mas sencillo de aprender para ser utilizado, cuál tiene una mejor interfaz, etc. Sin embargo, no fue hasta mediados de 2022 que comenzaron a plantearse si realmente estaban escogiendo la mejor opción a utilizar desde el punto de vista del rendimiento.

Por ello, el equipo de microservicios comenzó este proyecto de "benchmarking", para tratar de descubrir cuál de todas estas herramientas sería óptimo utilizar en los futuros desarrollos.

En el Capítulo 6 "Definiciones y abreviaturas"se explicará que es un microservicio y su gran importancia en este proyecto.

Capítulo 4

Descripción de la situación actual

4.1. Descripción del entorno actual

Hoy en día, la gran mayoría de empresas utilizan frameworks para desarrollar el software de todos sus proyectos, y entre ellas, el framework por excelencia es Spring, debido a la facilidad de su uso, el lenguaje que soporta (Java), que es uno de los mas utilizados entre los desarrolladores y su gran cantidad de herramientas que lo hacen tan polivalente. Estas características han conseguido que se forme una gran bola de nieve alrededor de esta tecnología, haciendo que se genere una gran cantidad de documentación con respecto a ella y desarrollando gran cantidad de librerías nativas de la aplicación por parte de los propios usuarios.

Sin embargo, si nos enfocamos en el rendimiento, no se ha valorado si es esta realmente la mejor opción a elegir, y si estamos sacrificando otras opciones con mejores prestaciones a cambio de utilizar la herramienta que "tenemos mas a mano". A lo largo de estos últimos años, ya ha habido algunos análisis que dejan ver que Spring no es para nada la mejor opción si a rendimiento nos referimos.

Para entender que se está intentando analizar, primero debemos entender correctamente como funciona un framework:
El trabajo de un framework consiste en facilitar al desarrollador muchas de

las tareas en el desarrollo de una aplicación, así como también dotarle de herramientas "prefabricadas"denominadas librerías, las cuales pueden dividirse en elementos mas pequeños llamados módulos, para ayudar a realizar acciones comunes en las aplicaciones.

Por ejemplo, en el caso de Express, cuenta con una serie de módulos que son utilizados para conectar tu aplicación a una base de datos, siendo solo necesario implantar este módulo a tu programa con una sola línea de código. En el caso de Spring, existen librerías para realizar peticiones a servidores sin tener que elaborar un código al respecto.

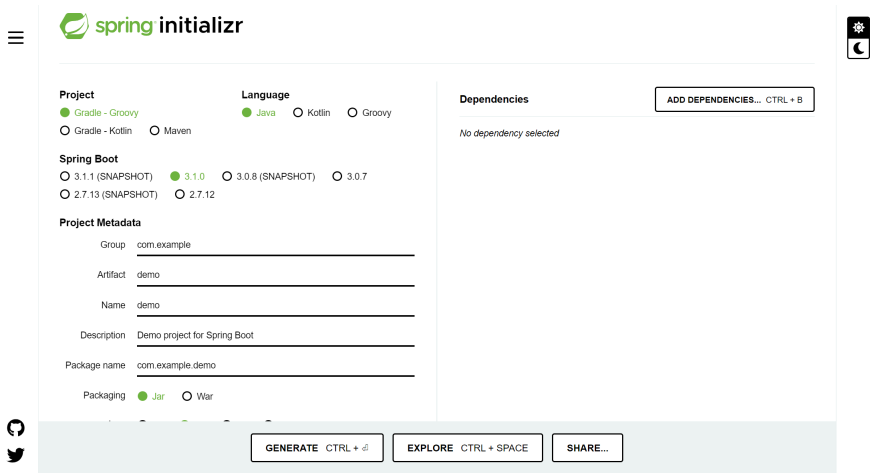

Figura 4.1: Herramienta de Spring Initializr [8]

Además, muchos de estos frameworks tienen herramientas para facilitar la creación de las aplicaciones, con una interfaz muy intuitiva en la que si tienes claro el tipo de aplicación que deseas crear, basta con darle las características que esta tendrá y te desplegará todos los directorios con las librerías necesarias instaladas en ellos.

4.2. Resumen de las características

4.2.1. Análisis cuantitativo

Las características cuantitativas que serán analizadas durante las pruebas de rendimiento serán las siguientes:

- **Latencia media de cada petición**: Se calculará el tiempo medio que tarda el servidor en dar la respuesta a cada una de las peticiones que conforman las aplicaciones de cada framework.
- **Carga máxima**: Se observará la carga máxima de peticiones por segundo que soporta cada aplicación antes de comenzar a presentar fallos.
- **Tiempo máximo y mínimo de respuesta**: Después de realizar el número de peticiones correspondientes en cada aplicación, se determinará cuál de ellas ha tenido el mayor y menor tiempo de respuesta en la petición mas costosa.
- **Consistencia de tiempos de respuesta**: Se buscará que las aplicaciones sean consistentes con los tiempos de respuesta conseguidos, buscando que la variación entre estos sea mínima, para conseguir así un sistema con la mayor fiabilidad posible a lo largo del tiempo.

Capítulo 5

Normas y referencias

Para realizar de forma correcta la documentación de este proyecto, se ha utilizado la norma Norma CCII-N2016-02 Norma Técnica para la realización de la Documentación de Proyectos en Ingeniería Informática [1].

El desarrollo de este proyecto debe cumplir con las normas que dicta el Reglamento de la UE sobre la protección y seguridad de los datos de los usuarios, por ello, se ha desarrollado el proyecto dentro del marco de la LOPD (Ley Orgánica de Protección de Datos de Carácter Personal) (BOE, 2023) [4].

5.1. Métodos, Herramientas, Modelos, Métricas y Prototipos

5.1.1. Métodos y Herramientas

5.1.1.1. Procedimiento

Inicialmente, se decidió el alcance del proyecto teniendo en cuenta la duración en la empresa de los responsables del proyecto.

A continuación, se procedió a realizar una búsqueda de anteriores pruebas de rendimiento similares, para tratar de fijar los parámetros de estas, así como también, una investigación sobre como desarrollar las aplicaciones que se buscaba realizar para posteriormente ser analizadas.

También, y debido a que como se ha mencionado anteriormente, este es un proyecto heredado y previamente empezado, se valorará el estado actual del proyecto observando todos los avances que ya habían sido realizados por otros miembros de la empresa, leyendo conversaciones de grupos de trabajo y analizando los archivos ya subidos al repositorio del proyecto. Además, se utilizó un documento de otro proyecto relacionado con las pruebas de rendimiento, que fue utilizado para sentar algunos de las bases de las pruebas que posteriormente serían realizadas.

5.1.1.2. Recursos

Los recursos empleados para la realización de este proyecto no han sido numerosos. En cuanto a material físico, únicamente ha sido necesario un ordenador, otorgado por parte de la compañía con el que se ha realizado tanto las pruebas de rendimiento y desarrollo de las aplicaciones, como para la búsqueda de la información necesaria con respecto a las necesidades del trabajo.

Por otra parte, en cuanto a los recursos humanos, se ha obtenido ayuda a nivel práctico (errores en las aplicaciones, sugerencias respecto a la elección de herramientas y explicación del uso de estas) de la mano del tutor del proyecto. También, se ha contado con la cooperación de otros trabajadores de la empresa, con los que se tuvieron reuniones periódicas para comprobar el correcto avance del proyecto.

5.1.1.3. Herramientas

Para desarrollar este proyecto, me he ayudado de las siguientes herramientas de Software:

- **Gitlab** :
  Gitlab consiste en una aplicación web dedicada al control de versiones de aplicaciones basada en Git. Se trata de un repositorio remoto en el que cada usuario con acceso puede subir las actualizaciones del proyecto. En este caso, el repositorio de Gitlab en el que estaba ubicado este proyecto, albergaba los primeros pasos por parte de los antiguos dueños del proyecto.

- **Visual Studio Code** :

Visual Studio Code es un editor de código gratuito y altamente utilizado desarrollado por Microsoft. Ofrece una amplia gama de funciones y extensiones para facilitar la escritura y edición de código en diferentes lenguajes de programación. En el caso de este proyecto, VS Code fue utilizado para desarrollar las aplicaciones tanto de Express como de Koa, y se hizo uso de su terminal propia para el arranque de estas.

- **Node.js** :
  Node.js es un entorno de ejecución de código JavaScript de lado del servidor que permite construir aplicaciones escalables y de alto rendimiento. Fue utilizado durante el desarrollo de este proyecto en las aplicaciones de Express y Koa, ya que estos entornos están basados en esta herramienta, así como la mayoría de herramientas que utilizan Javascript en la actualidad.

- **Java** :
  Java es un lenguaje de programación de alto nivel y orientado a objetos, conocido por su portabilidad y amplia adopción en el desarrollo de aplicaciones empresariales. Este lenguaje fue escogido para realizar dos de las aplicaciones (Springboot y Quarkus) al ser el lenguaje más utilizado dentro de la empresa.

- **Javascript** :
  JavaScript es un lenguaje de programación interpretado, de alto nivel y orientado a objetos, ampliamente utilizado para el desarrollo de aplicaciones web interactivas y dinámicas en el lado del cliente. JavaScript se utilizó como posible alternativa a Java para tratar de analizar si su uso mejoraría las aplicaciones de Java.

- **Maven** :
  Apache Maven es una herramienta de gestión y construcción de proyectos de software ampliamente utilizada en el desarrollo de aplicaciones Java. Permite la gestión de dependencias, compilación, prueba y empaquetado de forma eficiente y automatizada, simplificando el proceso de desarrollo y construcción del software. Apache Maven fue utilizada principalmente para la compilación, testeo y empaquetado de las aplicaciones de Spring y Quarkus.

- **JMeter** :
  JMeter es una herramienta de código abierto desarrollada también por

Apache que se utiliza para realizar pruebas de carga y rendimiento en aplicaciones web. Permite simular el comportamiento de múltiples usuarios y medir el rendimiento de una aplicación bajo diferentes condiciones de carga, ayudando a identificar cuellos de botella y problemas de rendimiento. JMeter fue la aplicación utilizada para realizar las pruebas de rendimiento de todas las aplicaciones, después de un estudio del resto de herramientas con la misma utilidad.

- **MongoDB** :
  MongoDB es una base de datos NoSQL de código abierto que almacena datos en documentos JSON, permitiendo una fácil escalabilidad y agilidad en el desarrollo de aplicaciones. Su arquitectura distribuida y capacidades de consultas avanzadas lo convierten en una opción popular para entornos web y móviles. En este caso, se ha utilizado la aplicación "desktop"de este tipo de bases de datos para monitorizar las peticiones de las aplicaciones desarrolladas.

- **Spring Tool Suite 4** :
  Spring Tool Suite 4 (STS) es un entorno de desarrollo integrado basado en Eclipse, diseñado específicamente para el desarrollo de aplicaciones basadas en el framework Spring. Proporciona herramientas y características especializadas para facilitar la creación, pruebas y despliegue de aplicaciones Spring de manera eficiente. STS incluye funcionalidades como autocompletado de código, depuración, gestión de dependencias y soporte para diversas tecnologías asociadas a Spring. Este IDE fue el utilizado para desarrollar y desplegar las aplicaciones basadas en Spring y Quarkus. Cabe destacar que este entorno tiene una funcionalidad que permite desplegar aplicaciones basadas en el framework Spring sin necesidad de utilizar Spring Initalizr.

- **Postman** :
  Postman es una herramienta que permite a los desarrolladores enviar y recibir peticiones HTTP de manera sencilla y eficiente. Con su interfaz intuitiva, los usuarios pueden realizar pruebas y enviar solicitudes a APIs, facilitando el proceso de desarrollo y depuración de aplicaciones basadas en servicios web. Postman fue la aplicación utilizada para realizar las peticiones HTTP correspondientes para comprobar el correcto funcionamiento de las aplicaciones desarrolladas.

- **Typora** :
  Typora es una aplicación de edición de texto minimalista y elegante que proporciona un entorno de escritura centrado en la productividad. Con soporte para formato Markdown en tiempo real, Typora permite a los usuarios crear y editar documentos con facilidad, brindando una experiencia fluida y enfocada en el contenido. Typora fue utilizado para la redacción de la documentación de cada una de las aplicaciones, al ser uno de los únicos editores de texto que soportan la compilación directa del lenguaje Markdown, con el cual se desarrollaron los respectivos "README".

- **Notepad ++** :
  Notepad++ es un editor de texto avanzado y de código abierto para Windows. Ofrece características como resaltado de sintaxis, autocompletado, búsqueda y reemplazo avanzados, así como la capacidad de trabajar con múltiples documentos simultáneamente, convirtiéndolo en una herramienta popular para programadores y usuarios que requieren una herramienta de edición de texto versátil. Notepad fue utilizado para la edición de archivos como los ajustes de STS, para poder agregar las dependencias necesarias al proyecto, como es el caso de Lombok o Maven.

Figura 5.1: Logos de herramientas utilizadas

En cuanto a las herramientas otorgadas por parte de la empresa para la realización del proyecto, se me hizo entrega de un portátil valorado en apro-

ximadamente 700 €.
También, se me dio acceso a diferentes herramientas propias de la empresa para facilitarme el desarrollo del proyecto (Jira, Repositorio Gitlab, Microsoft Teams), así como un seguimiento exhaustivo del proceso del proyecto con reuniones periódicas.

5.1.2. Modelos, Métricas y Prototipos

En cuanto al software, el modelo utilizado en las aplicaciones desarrolladas es el modelo vista-controlador (MVC), debido a los requisitos demandados por el equipo de microservicios del departamento de arquitectura.

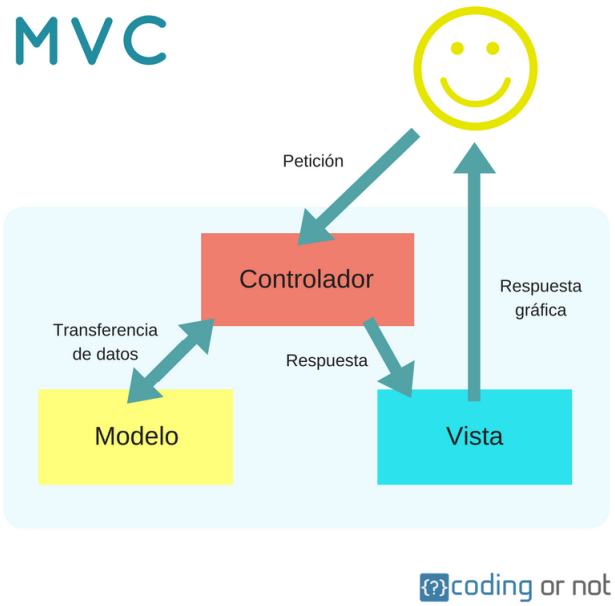

Figura 5.2: Modelo vista-controlador [6]

El modelo utilizado durante la creación del proyecto ha sido el modelo ágil, realizando sprints semanales en los que se iba avanzando en cada tarea y revisando que estas estuvieran desarrolladas correctamente en una reunión semanal.

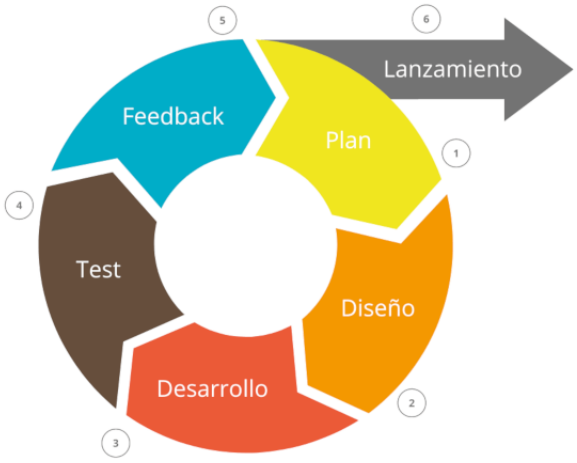

Figura 5.3: Modelo ágil del proyecto

El modelo utilizado en la redacción del proyecto está basado en la norma técnica CCII-N2016-02. Según el CCII (Consejo de Colegios de Ingeniería Informática), "esta norma subsana una importante carencia en el ámbito de la dirección y gestión de proyectos de ingeniería informática, facilitando y promoviendo así la cultura de la normalización en el sector informático español al proporcionar un mecanismo sencillo y eficiente para la especificación documental de los proyectos de ingeniería informática [1]."

En este proyecto, no se ha utilizado ninguna métrica para su realización, y en cuanto a prototipos, previo al desarrollo de las aplicaciones que serían puestas a prueba, se realizó uno para determinar cuales iban a ser sus características y su arquitectura.

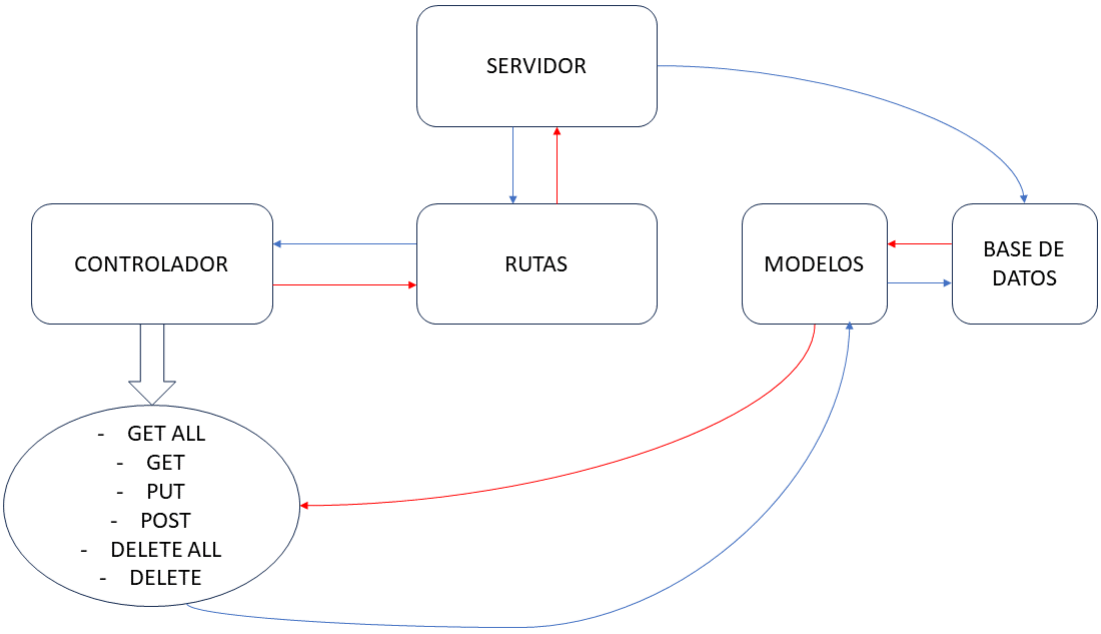

Figura 5.4: Prototipo de las aplicaciones

5.2. Mecanismos de control de calidad aplicados durante la redacción del proyecto

Para poder garantizar y asegurar una calidad correcta de la documentación de este proyecto, se ha decidido aplicar la norma CCII-N2016-02 Norma Técnica para la realización de la Documentación de Proyectos en Ingeniería Informática [1]. A su vez, para asegurar la correcta dirección del proyecto, se realizaron reuniones de seguimiento 2 veces a la semana con el tutor del proyecto, así como también otra reunión semanal más con director del proyecto y el encargado de control de calidad, en las que se informaba del avance del proyecto y se proponían nuevos pasos a seguir o características a añadir. Por otra parte, con el objetivo de depurar la comprensión y legibilidad de esta memoria, se ha hecho uso de la herramienta web Legible.es, que analiza el texto de este para comprobar su legibilidad y recomienda los cambios pertinentes.

Capítulo 6

Definiciones y abreviaturas

A continuación, se definirán algunos de los términos utilizados en este proyecto. En un primer momento, el significado de ciertos términos pueden resultar obvios, pero en este apartado se reflejará el significado específico que conllevan en el ámbito de este proyecto, concretando después de estas definiciones su papel en el trabajo.

- **Base de datos** :
  Una base de datos es una colección estructurada de datos almacenados electrónicamente. Permite almacenar, organizar y recuperar información de manera eficiente. Utiliza tablas para organizar los datos en filas y columnas. Se pueden realizar consultas para buscar y manipular la información. En el contexto de este proyecto, la base de datos será accesible a través de las aplicaciones CRUD desarrolladas, y estará formada por recursos denominados "products", con 3 características: nombre, descripción y código. CREATE (POST), creará un nuevo "product.en esta base de datos, READ (GET), devolverá un "product"dado su identificador propio en esta base de datos (creado por la propia base de datos), o si se quiere, se devolverá una lista con todos los "products.almacenados en esta. UPDATE (PUT), editará un product ya creado y cambiará alguna o todas sus características. DELETE eliminará uno o todos los products de la base de datos.

- **Benchmarking** :
  El benchmarking de una aplicación es un proceso de comparación y evaluación de rendimiento, características o funcionalidades de una aplicación con respecto a estándares o competidores. Se utiliza para

identificar fortalezas y debilidades, establecer metas de mejora y tomar decisiones informadas. Implica la recolección de datos, análisis comparativo y la identificación de áreas para optimización. El benchmarking ayuda a mejorar el rendimiento, la eficiencia y la calidad de una aplicación, permitiendo una mejor toma de decisiones y competitividad en el mercado.

- **CRUD** :
  Una aplicación CRUD es un tipo de aplicación que realiza operaciones básicas en una base de datos: Crear, Leer, Actualizar y Eliminar (Create, Read, Update, Delete). Permite crear nuevos registros, leer información existente, actualizar registros existentes y eliminarlos según sea necesario. Este enfoque se utiliza comúnmente en aplicaciones de gestión de datos donde se requiere interacción con una base de datos para realizar estas operaciones básicas de mantenimiento. Las aplicaciones desarrolladas para este proyecto son todas CRUD, permitiendo así probar

- **Framework** :
  Los frameworks son entornos de trabajo que facilitan el desarrollo software al estandarizar ciertos procesos. Estos ayudan al programador realizando tareas como autocompletar código, aportan librerías y módulos con código útil para ciertas funcionalidades específicas, librandole así de tener que programarlas por el mismo (por ejemplo, conexiones con bases de datos), e incluso ayudan al programador desplegando la estructura de directorios de la aplicación después de determinar ciertos parámetros.

- **HTTP** :
  HTTP (Hypertext Transfer Protocol) es un protocolo utilizado para la transferencia de información en la web. Funciona sobre TCP/IP y sigue un modelo de solicitud-respuesta. Todas las solicitudes (peticiones) están basadas en los encabezados GET, POST, PUT y DELETE, con los que se realizarán las peticiones correspondientes al servidor en cuestión.

- **IDE**:
  Un IDE (Integrated Development Environment) es un software que combina herramientas esenciales para el desarrollo de software, como

un editor de código, un depurador y un compilador, en una sola interfaz. Proporciona un entorno de trabajo completo para programadores, facilitando la escritura, prueba y depuración de código en un solo lugar. En este proyecto, los IDEs utilizados han sido STS y Visual Studio Code, el primero para las aplicaciones Java y el segundo para las Javascript

- **Microservicio** :
  Un microservicio de software es un enfoque arquitectónico en el desarrollo de aplicaciones donde se divide una aplicación en servicios pequeños e independientes, cada uno con su propia funcionalidad y comunicación a través de interfaces bien definidas. Cada microservicio se puede desarrollar, implementar y escalar de manera independiente, lo que permite una mayor modularidad y flexibilidad en el desarrollo de sistemas. Los microservicios suelen trabajar juntos para formar una aplicación completa, y su despliegue y gestión se simplifican al utilizar tecnologías como contenedores y orquestación de servicios. Las aplicaciones desarrolladas en este proyecto están basadas en este tipo de arquitectura. estas peticiones en cada una de ella y comparando el rendimiento de estas.

- **REST** :
  REST (Representational State Transfer) es un estilo arquitectónico utilizado para diseñar servicios web que se basan en los principios fundamentales de la web. Se enfoca en la interoperabilidad, la escalabilidad y la simplicidad al permitir la comunicación entre sistemas distribuidos a través de la manipulación de recursos a través de operaciones estándar basadas en HTTP, como GET, POST, PUT y DELETE. Los servicios REST utilizan URLs para identificar los recursos y los formatos de datos como JSON o XML para representar y transferir la información. Esta arquitectura es ampliamente utilizada en el desarrollo de aplicaciones web y servicios API. Las aplicaciones CRUD desarrolladas en el proyecto están basadas en este estilo arquitectónico para realizar las peticiones.

Capítulo 7

Requisitos iniciales

Este proyecto está conformado por 4 aplicaciones idénticas en cuanto a requisitos, y un proyecto en JMeter, por lo que para analizar los requisitos los englobaremos en 2 productos diferentes. A continuación, se revisarán los requisitos tanto funcionales como no funcionales de ambos productos.

- **Requisitos funcionales:**
  A continuación repasaremos los requisitos funcionales iniciales que se propusieron que deberían tener las aplicaciones:

Producto	Requisitos funcionales
Aplicaciones	• Conexión directa con MongoDB • Microservicios conforman cada app • Utilización de librerías propias • Arquitectura vista-controlador
Proyecto JMeter	• Capturador tiempos de respuesta • Carga idéntica por aplicación • Servidor proxy único por aplicación

Tabla 7.1: Requisitos funcionales

- **Requisitos no funcionales:**
  Los requisitos no funcionales que deberán de ser cumplidos por las aplicaciones son los siguientes:

  - **Rendimiento**: Las aplicaciones desarrolladas deberán buscar tener el máximo rendimiento posible.
  - **Robustez**: Las aplicaciones soportarán fallos en las peticiones sin hacer caer toda la aplicación y presentarán mensajes de error consistentes
  - **Disponibilidad**: Las aplicaciones tendrán que estar disponibles el mayor tiempo posible sin sufrir caídas, sobretodo cuando se estén realizando las pruebas de rendimiento.
  - **Compatibilidad**: Las aplicaciones deben estar preparadas para ser ejecutadas en diferentes entornos (diferentes IDEs, diferentes S.O, diferentes versiones, etc)
  - **Actuación**:Las aplicaciones deben soportar una gran cantidad de peticiones simultáneas sin presentar fallos.
  - **Consistencia**: Los tiempos de respuesta de las aplicaciones deben de ser consistentes, sin grandes variaciones entre dos peticiones idénticas.

Capítulo 8

Alcance

El alcance del proyecto se delimitó durante las fases iniciales de este, teniendo en cuenta la cantidad de responsables del proyecto y sus conocimientos, por lo que se llegó a la conclusión de que el proyecto solo compararía 2 lenguajes de programación diferentes, comparando 2 frameworks en cada uno de ellos. A su vez, y debido al desempeño del departamento responsable en otros proyectos, se consensuó en que las aplicaciones desarrolladas para posteriormente ser analizadas serían CRUD, debido a que son el tipo de aplicación mas común en este sector, por lo que ver como se comportan este tipo de aplicaciones en diferentes entornos sería interesante. A su vez, estas aplicaciones estarían basadas en una arquitectura de microservicios, ya que, de nuevo, el equipo de desarrollo encargado de este proyecto estaba especializado en este tipo de estructura de aplicaciones, por lo que era importante ver como rendían esta clase de aplicaciones en diferentes frameworks y lenguajes. En cuanto al análisis de estas aplicaciones, una vez desarrolladas, se concretaron las características que tendrían las pruebas de rendimiento, y cuales serían los parámetros a analizar, determinando el orden de importancia de estos y las condiciones del entorno en el que se desarrollarían las pruebas.

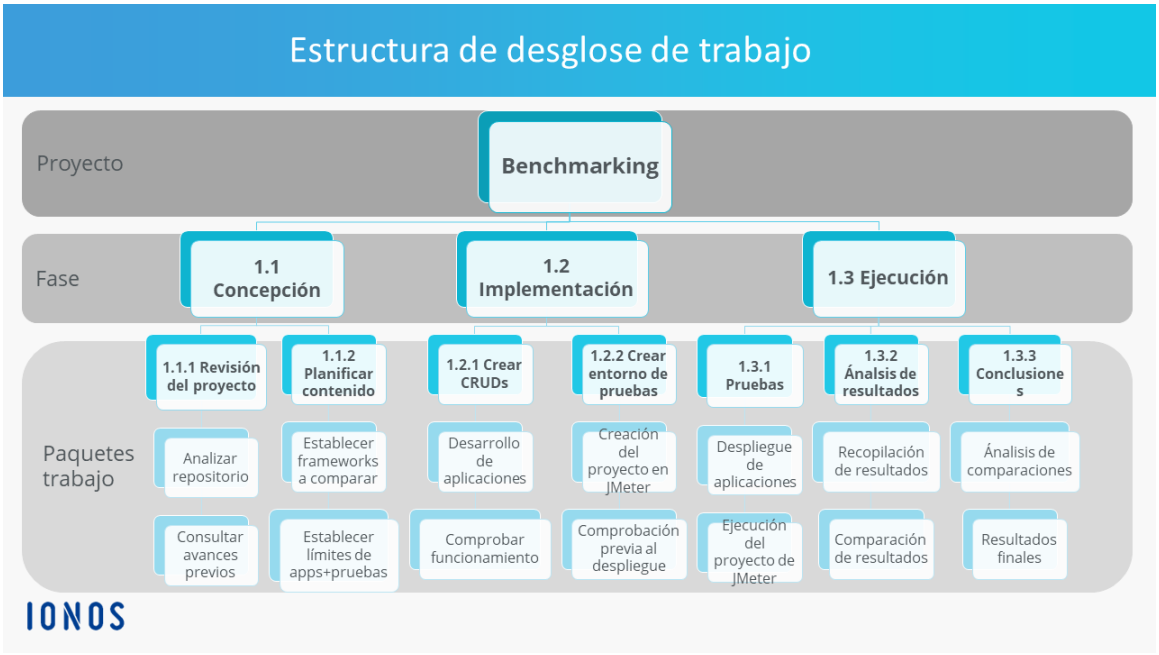

Figura 8.1: EDT del proyecto [2]

El proyecto estuvo separado en 3 fases marcadas:

- **Concepción**: Se establecen las características del proyecto que se va a desarrollar. Se revisan los avances por parte de los antiguos responsables del proyecto, analizando el repositorio y las conversaciones del grupo de trabajo, y se planifica el resto del proyecto a realizar junto con sus características.
- **Implementación**: Se desarrolla el proyecto como tal. Se crean las aplicaciones web y se comprueba su funcionamiento, y se crea el proyecto de pruebas que se va a ejecutar.
- **Ejecución**: Se ejecutan las pruebas de rendimiento en cada una de las aplicaciones, se recompilan y analizan los resultados y por último se obtienen las conclusiones pertinentes en base a estos resultados.

Capítulo 9

Hipótesis y restricciones

Este proyecto surgió de la sospecha de que el framework utilizado por parte de la compañía, Spring, no era el óptimo a utilizar, debido a diversos estudios de terceros que afirmaban que otros entornos como Quarkus ofrecían unas mejores prestaciones y contaban con una mayor potencia. Por ello, la hipótesis inicial fue que las pruebas de carga que se aplicarían a ambos entornos, resultarían en una clara ventaja por parte de Quarkus, demostrándose así que sería aconsejable cambiar de entorno de trabajo en la compañía.

Por otra parte, y debido a la metodología de trabajo del departamento responsable del proyecto, y que este proyecto podría suponer un cambio en las herramientas utilizadas por este departamento, tanto los frameworks como las aplicaciones que, más tarde serían desarrolladas con estos, debían tener una serie de restricciones o características específicas, como son:

- La capacidad de conexión con bases de datos MongoDB. Tanto los frameworks como las aplicaciones deberían de poder conectarse de forma nativa con bases de datos MongoDB, al ser el tipo de bases de datos que se utilizan comúnmente en el resto de proyectos del departamento de arquitectura.
- Tanto los lenguajes de desarrollo de las aplicaciones como los frameworks utilizados deberán de tener la capacidad de desarrollar aplicaciones CRUD con ellos, con todo lo que ello conlleva (poder realizar peticiones HTTP y poder desarrollar aplicaciones web)
- Las aplicaciones desarrolladas deberán de tener una estructura basada

en microservicios, debido a que, aparte de que esto puede influir en el rendimiento final y se buscará comprobar como reaccionan los frameworks al desarrollo de aplicaciones con esta arquitectura, el equipo dentro del departamento responsable, se dedica al desarrollo de aplicaciones basada en esta arquitectura.

Capítulo 10

Estudio de alternativas y viabilidad

Previo al comienzo en el desarrollo del proyecto, se plantearon una serie de alternativas diferentes. A continuación, serán mencionadas junto con la razón por la cuál finalmente no fueron escogidas:

1. La investigación de comparaciones de frameworks ya realizadas, y utilización de los datos recogidos para llevar a cabo las conclusiones. Esta alternativa, después de una larga investigación en diferentes fuentes, fue rechazada debido a que, aparte de que el número de benchmarkings publicados es escaso, estos no son de una calidad suficiente como para considerar el análisis relevante, así como tampoco cumplían muchas de las restricciones que se plantearon desde un principio (Ver el Capítulo 9: Hipótesis y restricciones).

2. Buscar proyectos ya creados en estos entornos y utilizarlos para realizar las pruebas planteadas. Esta opción se descartó rápidamente, debido a que, de nuevo, las aplicaciones desarrolladas debían seguir ciertos estándares que los proyectos encontrados no cumplían, por lo que se optó por la última alternativa.

3. **ALTERNATIVA ESCOGIDA**: Crear aplicaciones idénticas pero utilizando en cada una de ellas un framework diferente, consiguiendo así que el único cambio en el rendimiento de estas fuera determinado por el rendimiento del propio entorno. Posteriormente se realizarían el mismo

tipo de pruebas para cada una de las aplicaciones y se compararían los resultados, obteniendo las conclusiones pertinentes.

En cuanto a la viabilidad del proyecto, esta podrá ser desglosada en 3 apartados diferentes, analizando tanto la viabilidad económica, la técnica y la legal.

10.1. Viabilidad económica

En el apartado 15 " Resumen del presupuesto"se recogerá toda la información con respecto a los costes y márgenes de beneficio del proyecto. Este proyecto ha utilizado todas las herramientas software de código libre, por lo que el coste solo supondría cosas como el hardware utilizado o los recursos humanos empleados.

10.2. Viabilidad técnica

Este proyecto no supone un gran desafío en lo que a viabilidad técnica se refiere, ya que no es necesaria una potencia fuera de lo común, y al ser una comparación, es bastante indiferente la potencia del equipo utilizado, al ser este el mismo para todas las pruebas. Por esto mismo, solo sería necesario un equipo que pudiese afrontar el desarrollo de las aplicaciones y poder aguantar el nivel de unas pruebas de carga bastante costosas, las cuales fueron capaces de ser abordadas con el equipo otorgado por parte de la empresa. En cuanto a lenguajes y entornos elegidos, estos no han supuesto un problema porque son 2 de los lenguajes mas utilizados en el mercado, y todos los frameworks disponen de un gran servicio de mantenimiento que los mantiene actualizados y funcionales sin ningún tipo de fallo.

10.3. Viabilidad legal

Al tratarse de un proyecto realizado bajo el marco de la empresa NTT DATA, toda la recopilación de información y resultados es propiedad de esta, así como también lo son las aplicaciones desarrolladas.

Capítulo 11

Descripción de la solución propuesta

Después de analizar diferentes tipos de benchmarking y de concretar las pruebas de rendimiento que posteriormente se realizarían, se llegó a una conclusión con respecto a la solución final. Primero de todo, se desarrollarían las aplicaciones pertinentes siguiendo las restricciones impuestas por parte del departamento (Ver Capítulo 9). Después de esto, y una vez se ha probado que todas las aplicaciones funcionan correctamente, creando 4 proyectos diferentes en Postman con todas las peticiones posibles para cada uno de los frameworks, se procedió a crear el proyecto de Jmeter, con las funciones necesarias.

Primero de todo, se crearían 4 grupos de hilos diferentes, uno para cada framework, y cada uno de estos estaría conectado con un Servidor Proxy HTTP, el cual a su vez estaría escuchando al mismo puerto que la aplicación de Postman para poder así escuchar las peticiones de la aplicación y almacenarlas en el grupo de hilos correspondiente. Una vez realizada esta tarea, solo quedaría añadir un nuevo elemento, el cual mostraría los tiempos de respuesta, porcentajes de fallo, peticiones por segundo y cantidad de datos enviados y recibidos por segundo una vez se lance cada una de las peticiones.

Por último, se procedería a comenzar con las pruebas de rendimiento en si, concretando una cantidad de peticiones simultaneas, así como el número de veces que se va a lanzar esa cantidad de peticiones. Al terminar, se recopilarán los datos de cada una de las ejecuciones (todas ellas idénticas) y se

procederá a la comparación de los resultados obtenidos.

Los resultados observados serán los mencionados en el análisis cuantitativo (Capítulo 4.2.1):

Capítulo 12

Análisis de Riesgos

A continuación, se han analizado los riesgos a los que está expuesto el proyecto, así como el impacto que estos podrían tener en los resultados del mismo. Se plasmarán estos riesgos valorando su impacto y la probabilidad de que ocurran, y utilizando una matriz de riesgo como la siguiente:

		IMPACTO		
		Bajo	Medio	Alto
PROBABILIDAD	Baja	Muy bajo	Bajo	Medio
	Media	Bajo	Medio	Alto
	Alta	Medio	Alto	Muy alto

Figura 12.1: Matriz de riesgo [5]

Amenaza	**Impacto**	**Probabilidad**	**Riesgo**
Nueva versión de framework con mas rendimiento	Medio	Baja	Bajo
Caída del servidor durante las pruebas	Medio	Media	Medio
Fin de licencia de prueba para documentar (Typora)	Bajo	Media	Bajo
Falta de tiempo para realizar las pruebas en la empresa	Alto	Media	Alto
Dedicación de los expertos	Bajo	Alta	Medio
Compromiso del equipo	Alto	Baja	Medio
Liderazgo del Director del Proyecto	Medio	Baja	Bajo
Cambios significativos inicido del proyecto	Medio	Media	Medio
Sobrecarga en el equipo del proyecto	Alto	Baja	Medio
Desconocimiento de la tecnología	Bajo	Alta	Medio
Pérdida de miembros clave del equipo de proyecto	Medio	Alta	Alto

Tabla 12.1: Análisis de riesgos

A continuación se explicará en profundidad cada uno de los riesgos, junto con sus posibles soluciones:

- **Nueva versión de framework con mas rendimiento**: Existe la posibilidad de que en el futuro surja una nueva versión de alguno de los frameworks, mucho mas optimizada, que haga que los resultados obtenidos en este proyecto queden obsoletos. Este problema es inevitable, pero las conclusiones seguirán sirviendo para los usuarios que utilicen las versiones utilizadas en este benchmarking.

- **Caída del servidor durante las pruebas**: Hay un riesgo de que el servidor caiga mientras se están realizando las pruebas de rendimiento y haya que volver a desplegarlo. Este problema es sencillo de solventar:

simplemente se volverá a desplegar la aplicación y se realizarán las pruebas de nuevo desde el principio.

- **Fin de licencia de prueba para documentar**: La única licencia de pruebas utilizada en el proyecto fue el editor de texto Typora, para realizar la documentación. Tenía una duración de 14 días de prueba, pero si estos días no hubiesen sido suficientes simplemente se habría buscado una aplicación similar para terminar de documentar las aplicaciones.

- **Falta de tiempo para realizar las pruebas en la empresa** : Existe la posibilidad de que los responsables del proyecto no sean capaces de finalizar este antes de su salida de la empresa. Este problema se solucionaría o bien delegando el proyecto a otro responsable de la empresa, o finalizando este por su cuenta fuera de la empresa, con el permiso de esta.

- **Dedicación de los expertos**: Puede ocurrir que los expertos de la materia estén demasiado ocupados con otros proyectos y no puedan ayudar a solucionar este. Este problema no supondría mucho impacto en el proyecto, ya que las herramientas utilizadas no necesitan de mucha experiencia por lo que sería sencillo de solventar sin necesidad de expertos en la materia.

- **Compromiso del equipo** : Es posible que exista cierta falta de compromiso en el equipo de trabajo, ya sea por otras responsabilidades o por falta de profesionalidad. Este problema puede ser solucionado si otros miembros del equipo asumen temporalmente las tareas de los responsables no comprometidos con el proyecto, con el fin de que este siga avanzando

- **Liderazgo del Director del Proyecto**: Cabe la posibilidad de que el director del proyecto no pueda liderar al equipo efectivamente o comunicarse con los responsables del mismo. Este problema tiene como solución una mayor autonomía de los responsables del proyecto, pudiendo avanzar por su cuenta sin necesidad de que nadie les de instrucciones. Esto se consigue con una mayor implicación de los responsables y una clara especificación de objetivos y metas.

- **Cambios significativos iniciado del proyecto** : Hay un riesgo de que los cambios y ampliaciones de requisitos desde el inicio del proyecto

supongan un problema para el mismo. Esto se solucionaría con una buena gestión de requisitos, desechando aquellos que sean insostenibles de abordar.

- **Sobrecarga en el equipo del proyecto**: Existe un riesgo de que el equipo involucrado tenga una saturación de trabajo con otros proyectos. Este riesgo es prácticamente inexistente, debido a que los responsables del proyecto tienen una completa dedicación a este.
- **Desconocimiento de la tecnología**: Es posible que los encargados en el proyecto desconozcan ciertas tecnologías a utilizar en este, y eso suponga un retraso en los tiempos de entrega. Este problema puede ser solventado documentándose correctamente acerca de estas herramientas, y pedir ayuda al tutor del proyecto cuando haya alguna duda con respecto a la utilización de las tecnologías.
- **Pérdida de miembros clave del equipo de proyecto**: Cabe la posibilidad de que el equipo de trabajo pierda miembros clave durante el desarrollo del mismo. Este problema podrá ser solucionado asumiendo las responsabilidades de el miembro perdido por parte de del resto del equipo de trabajo. Esto decelerará el avance del proyecto, pero no supondrá un problema mayor.

Capítulo 13

Organización y gestión del proyecto

13.1. Organización

13.1.1. Actores del proyecto y relaciones entre los mismos

En este proyecto han participado Pau Ribes Pastor, Jaime Sarabia Chamero junto a otros trabajadores de la empresa NTT DATA. A continuación se adjunta una imagen de los actores del proyecto:

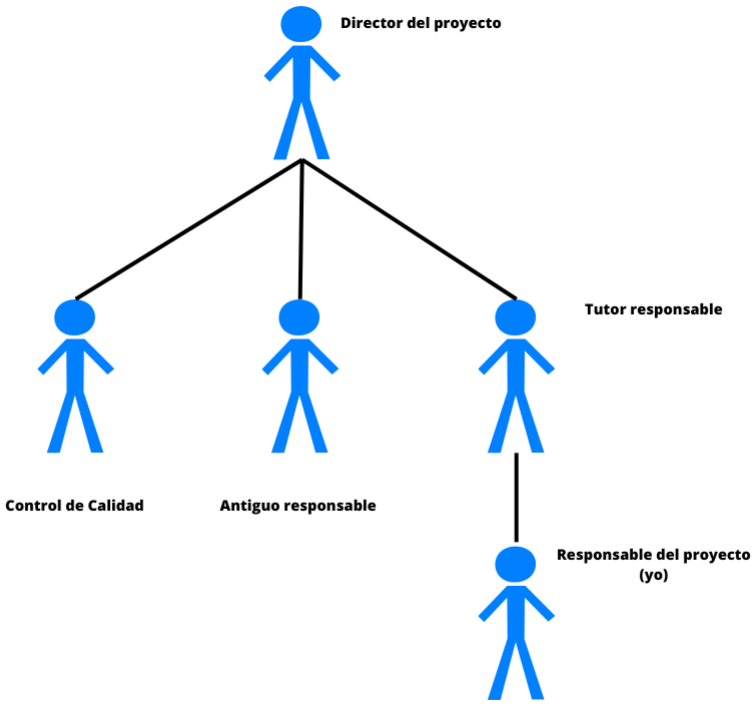

Figura 13.1: Actores del proyecto

13.1.2. Estructura interna

La estructura interna que se ha determinado para el desarrollo del proyecto es la siguiente:

- **Director del proyecto**
- **Tutor responsable**: Pau Ribes Pastor
- **Control de calidad**
- **Antiguo responsable del proyecto**
- **Programador**: Jaime Sarabia Chamero
- **Tester**: Jaime Sarabia Chamero
- **Analista**: Jaime Sarabia Chamero

13.1.3. Roles y responsabilidades

A continuación se explicará el trabajo de cada uno de los integrantes del proyecto:

1. **Director del proyecto**: Es el jefe y máximo responsable del proyecto. Es el encargado de dar instrucciones y quien define los requisitos y características específicas del proyecto.

2. **Tutor responsable**: Es el responsable del desarrollador y tester, guiándole en el proyecto y ayudándole con cualquier problema que pueda surgir. Además, mantiene contacto directo con el director del proyecto para posibles cambios en el alcance o los requisitos.

3. **Control de calidad**: Se encarga de asegurarse de que el proyecto está avanzando correctamente, manteniendo reuniones periódicas para comprobar el estado del proyecto y revisando los objetivos con los responsables de su avance, añadiendo nuevas metas de cara a la próxima reunión.

4. **Antiguo responsable del proyecto**: Su trabajo consiste en explicar a los nuevos encargados del proyecto en que consiste este, comentando cualquier avance realizado previamente y especificando cuales son los objetivos principales del proyecto, para que el nuevo equipo tenga claras las metas que se busca alcanzar.

5. **Ingeniero de Software**: Su responsabilidad es supervisar y coordinar las actividades diarias, así como planificar y controlar las tareas técnicas.

6. **Programador**: Su tarea consistirá en desarrollar las aplicaciones CRUD que serán puestas a prueba, así como de comprobar el correcto funcionamiento de estas realizándoles peticiones de prueba.

7. **Tester**: Será responsable de el uso de Jmeter para realizar las pruebas de rendimiento a cada una de las aplicaciones y posteriormente recabar los resultados de estas.

8. **Analista**: Se encargará de interpretar los resultados obtenidos en las pruebas de rendimiento, comparar cada uno de los frameworks en base a estos resultados, y desarrollar una conclusión teniendo estos en cuenta.

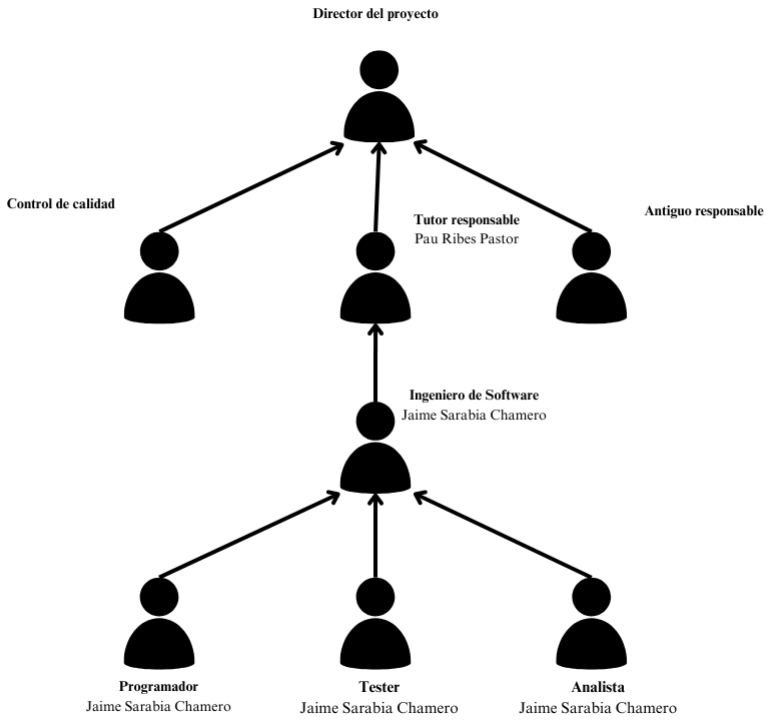

Figura 13.2: Estructura Interna del proyecto

13.2. Gestión del proyecto

13.2.1. Gestión de requisitos

En base a los requisitos iniciales, se planeó un sistema de gestión de los mismos, en el que se establecía un orden de prioridades con respecto a estos. A medida que el proyecto avanzaba, estos requisitos fueron variando debido a contratiempos surgidos o necesidades de la empresa.

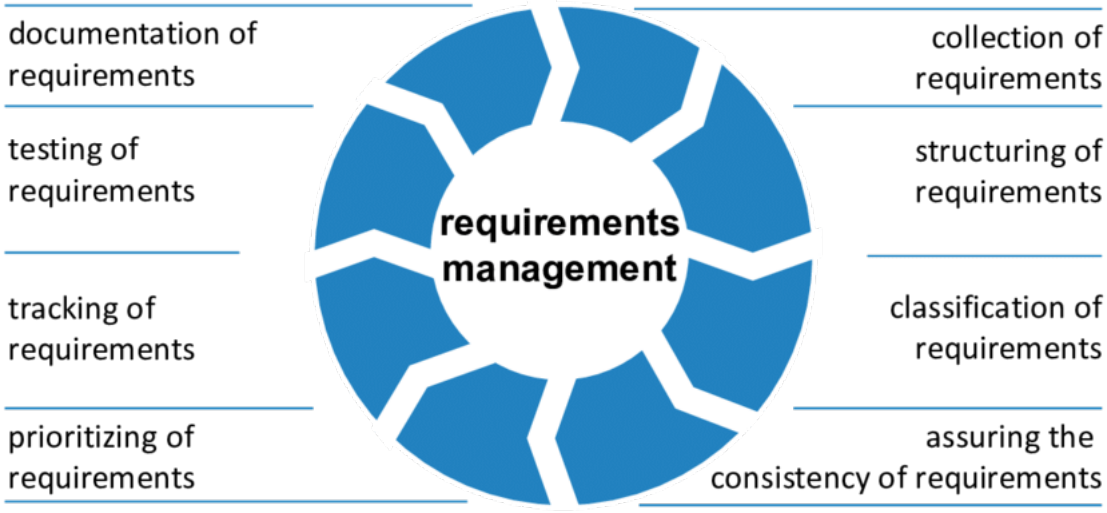

Figura 13.3: Gestión de requisitos[3]

13.2.2. Gestión de incidencias

Durante el proyecto, es muy posible que surjan incidencias que causen imprevistos durante el desarrollo del proyecto, ya sean errores en las aplicaciones desarrolladas, que causen un contratiempo, y haya que volver a fases previas del desarrollo, o problemas técnicos con el equipo utilizado.

Cada una de estas incidencias estaría a su vez clasificada según el impacto que esta generaría, como podemos comprobar en el Capítulo 12 "Análisis de Riesgos".

13.2.3. Gestión de riesgos

Al comienzo del proyecto, se estimaron cuales podrían ser los riesgos de este, y en base a ellos se realizó un sistema de gestión de estos, buscando una posible alternativa en caso de que ocurrieran, evitando así una mayor gravedad y teniendo controlado el futuro del proyecto en todo momento. Por esto mismo, se hizo mucho hincapié en temas como:

- Versiones de las herramientas utilizadas.
- Compatibilidad con diferentes S.O.

- Tiempo necesario para terminar el proyecto.

13.2.4. Gestión y Validación de las pruebas

Para asegurarse una correcta gestión de las pruebas, estas han sido determinantes para el avance del proyecto, así que si la primera tanda de pruebas no era pasada con éxito, se trataría de arreglar los problemas necesarios para que las pruebas fueran exitosas antes de seguir avanzando. Por ello, podemos separar estas pruebas en 3 etapas:

1. **Prueba de funcionalidad de herramientas**:
    - MongoDB
    - JMeter
    - Postman
2. **Prueba de funcionalidad de aplicaciones.**
3. **Prueba de funcionalidad del proyecto de pruebas.**

Capítulo 14

Planificación temporal

En la siguiente figura se muestra un Diagrama de Gantt del proyecto:

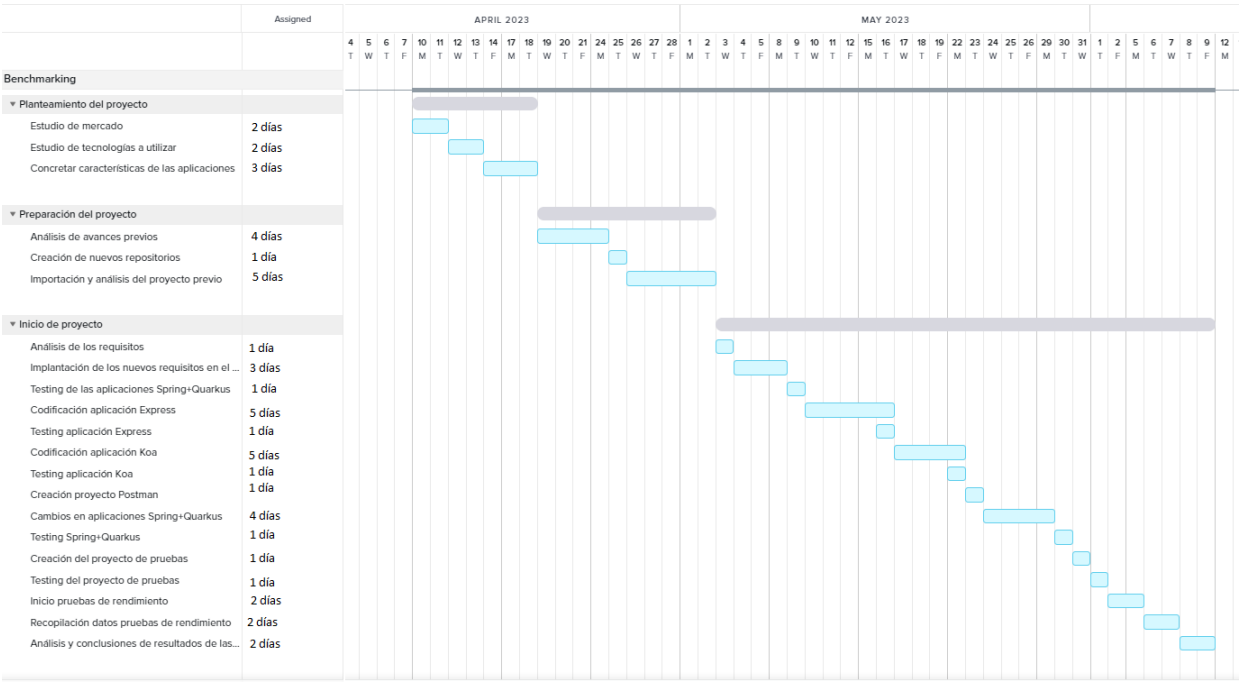

Figura 14.1: Diagrama de Gantt del proyecto

Este diagrama muestra las 3 etapas del desarrollo del proyecto, junto con las tareas de cada una de estas etapas. Junto a cada tarea, se puede observar la duración que supuso.
Las 3 etapas fueron:

- El planteamiento inicial del proyecto, en el que se analizaba el mercado en busca de los frameworks que se probarían, las herramientas que se utilizarían para el desarrollo y se concretaron los requisitos y características de las aplicaciones a desarrollar y de las pruebas de rendimiento.

- La preparación del proyecto, en la que se analizaban los repositorios antiguos del proyecto, se creaban los nuevos directorios necesarios y se preparaba el entorno en el que se desarrollarían las aplicaciones. También se clonó el repositorio remoto en un repositorio local y se comprobó el funcionamiento de las aplicaciones ya desarrolladas.

- El inicio del proyecto, que engloba toda la implantación de aplicaciones, proyecto Postman y proyecto JMeter; despliegue y comprobación del funcionamiento de las aplicaciones, ejecución de las pruebas de rendimiento, recopilación de resultados, análisis de los mismos y elaboración de las conclusiones del proyecto.

Algunas fechas relevantes del proyecto fueron el 19 de abril, en el que se comenzó el desarrollo de las aplicaciones, el 22 de Mayo, en la que se finalizó el desarrollo de estas, el 1 de Junio, cuando se empezó a realizar las pruebas de rendimiento y el 9 de Junio, cuando se obtuvieron los resultados y conclusiones de las mismas y por ende, se finalizó el proyecto.

14.1. Evolución del plan de proyecto

La metodología utilizada para desarrollar este proyecto ha sido la metodología SCRUM.

Con el fin de cumplir los plazos establecidos, cada semana se realizaba una reunión con el director y con el encargado de control de calidad para asegurar el avance de este, así como otras 2 reuniones semanales con el tutor

responsable para evitar tanto errores que entorpeciesen el ritmo del proyecto, como para resolver dudas con respecto al desarrollo del mismo. De esta forma, la planificación del proyecto iba evolucionando a lo largo del tiempo, añadiendo cualquier contratiempo como una nueva tarea a resolver (véase los cambios realizados en las aplicaciones de Spring y Quarkus una vez estas estaban finalizadas después de un cambio en los requisitos del proyecto).

Para comprobar que el ritmo de avance en el proyecto era el adecuado, se hizo uso de una aplicación denominada Jira, la cuál contenía un tablero SCRUM, en el que se recogían todas las tareas del proyecto y se utilizaba para comunicar cualquier tipo de avance en el proyecto, ya sea para pasar una tarea de estado ("To do" a "finished", por ejemplo) o para mediante un comentario, comunicar los avances o problemas en alguna tarea concreta. Con esto se consiguió un control total de todas las tareas y el estado en el que se encontraba cada una.

Capítulo 15

Resumen del Presupuesto

Precios del Software:

Nombre	Descripción	Precio (€)
Visual Studio Code	IDE utilizado para el desarrollo de las aplicaciones de Express y Koa	0
Nodejs	Entorno de ejecución de Javascript	0
JMeter	Herramienta utilizada para realizar las pruebas de rendimiento en las aplicaciones	0
MongoDB	Base de datos utilizada por las aplicaciones	0
Spring Tool Suite 4	IDE utilizado para el desarrollo de las aplicaciones de Spring y Quarkus	0
Postman	Herramienta utilizada para realizar las peticiones HTTP a las aplicaciones desarrolladas	0

Tabla 15.1: Presupuesto Software

Precios del Hardware:
Debido a que el único recurso Hardware utilizado en el proyecto se trata del portátil y este fue otorgado por la empresa y devuelvo posteriormente, se

determinará la amortización del mismo teniendo en cuenta un coste aproximado en base a sus características:

Precio portátil	700€ aproximadamente
Tiempo medio vida útil portátil	48 meses
Tiempo utilización portátil	4 meses
Amortización	700/48*4 = 58,3€

Tabla 15.2: Presupuesto Hardware

Precios de mano de obra:
Como se ha mencionado en el apartado anterior, la duración del proyecto fue de 4 meses. El coste de algunos integrantes del equipo es información privada, así que se tendrá en cuenta solo el coste de aquellos integrantes que no rechacen hacer pública esta información. De esta forma, teniendo en cuenta las 5 horas de trabajo diarias durante aproximadamente 4 meses, el coste de la mano de obra del proyecto será el siguiente:

Cargo	Nombre	Precio/mes (€)	Coste total
Ingeniero de Software	Jaime Sarabia Chamero	540€/mes	2160€

Tabla 15.3: Presupuesto Mano de Obra

Parte II

PROYECTO TÉCNICO

Capítulo 16

ANEXOS

16.1. Anexo - Documentación de entrada

Este proyecto ha sido desarrollado, implementado y analizado de forma telemática con la empresa NTT DATA, por parte de Jaime Sarabia Chamero, alumno de cuarto curso de Ingeniería Informática en la Universidad Politécnica de Valencia. La empresa NTT DATA ha aportado la totalidad de las herramientas utilizadas durante el proyecto, y sus resultados son propiedad intelectual de esta.

16.1.1. NTT DATA

La empresa promotora de este proyecto es NTT DATA, una empresa japonesa de servicios de tecnología de la información. Su trabajo se desarrolla sobre todo en las áreas de TI y BPOs. La oficina encargada del proyecto está compuesta mayoritariamente por empleados de la antigua compañía española Everis que en 2014 fue adquirida por la empresa. NTT DATA cuenta con sedes distribuidas en más de 50 países y con una red de más de 130.000 empleados; concretamente en España, en distintas ciudades, como Madrid, Sevilla o Valencia, siendo esta última el lugar de realización del proyecto [7].

Figura 16.1: Logo NTT DATA

El proyecto se ha desarrollado en el departamento de Arquitectura de Software, concretamente en el equipo de Microservicios, donde hay distintos proyectos encargados de forma transversal entre distintas sedes dentro de la Península, de forma que, el trabajo junto con compañeros de otras sedes es constante y requiere una fluidez en la comunicación para garantizar la viabilidad del proyecto.

16.2. Anexo - Análisis y Diseño del Sistema

Para lograr el objetivo de analizar el rendimiento de ciertas aplicaciones web, se ha tenido que lograr construir un entorno comunicado entre si mediante diversas aplicaciones, que finalmente cumplía la labor de analizar los tiempos de respuesta y otras cualidades de cada una de las aplicaciones

16.2.1. Flujo del sistema

El sistema generado, como se ha mencionado anteriormente, esta conformado por una serie de aplicaciones y herramientas diferentes, que combinadas dan lugar a el proyecto final. A continuación se mostrará un esquema del entorno creado junto con una explicación de cada uno de sus componentes y conexiones, así como el flujo de información que pasa a través de estas:

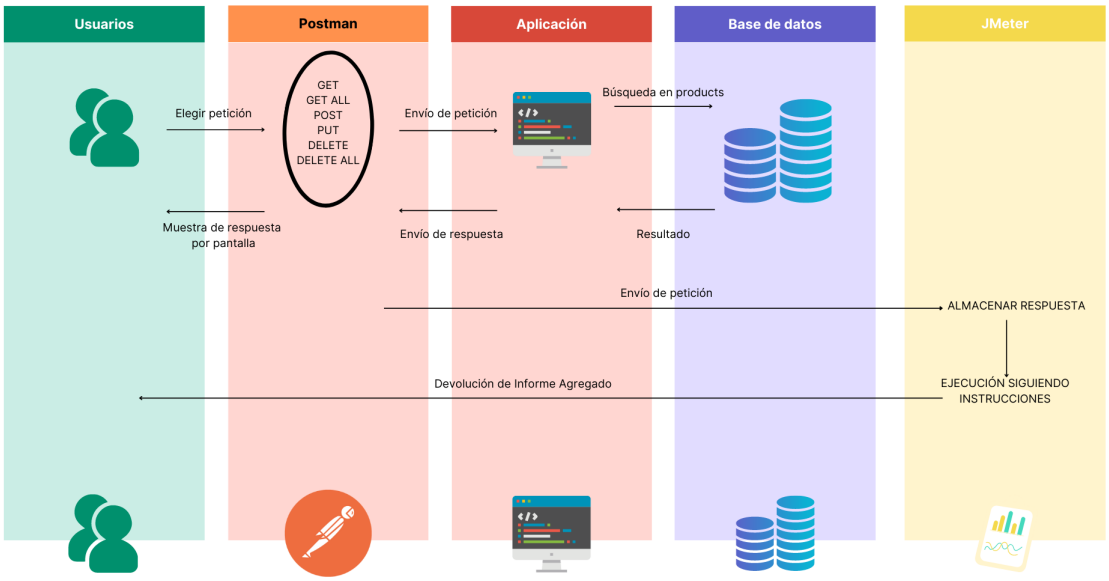

Figura 16.2: Flujo del sistema

En esta figura se puede observar el flujo de trabajo de las aplicaciones y herramientas que componen el proyecto. Si se considera que una aplicación esta inicializada, se puede afirmar que este flujo, se compone de los siguientes

pasos:

1. El usuario realiza una petición HTTP mediante la aplicación Postman a la aplicación que se quiere analizar.
2. Postman, realiza dicha petición al puerto de escucha correspondiente a la aplicación elegida.
3. La aplicación interpreta la petición y, según cual sea, realizará dicha acción en la base de datos, ya sea crear, eliminar, buscar o editar una instancia.
4. La respuesta obtenida en la base de datos, será devuelta a la aplicación Postman con el formato correspondiente.
5. Postman devolverá esta respuesta al usuario, y a la vez, a través del proxy configurado en la aplicación de Postman enviará una copia de la petición realizada a la aplicación JMeter.
6. JMeter almacenará dicha respuesta en el grupo de hilos correspondiente para poder ser utilizada posteriormente.
7. Cuando este grupo de hilos sea ejecutado, todas las peticiones almacenadas en este serán ejecutadas siguiendo las instrucciones descritas en los ajustes del grupo de hilos. Estas opciones van desde cuantas peticiones idénticas se quieren realizar simultáneamente como cuantas veces se desea realizar esta acción.
8. Al terminar todas las peticiones descritas, el Ïnforme agregado"de dicho grupo de hilos, devolverá al usuario los resultados en términos de rendimiento de dicho conjunto de peticiones a la aplicación.

16.2.2. Especificación de conexiones

A continuación, se expondrá como las aplicaciones están interconectadas entre si:

- **POSTMAN - APLICACIÓN**: Postman realizará peticiones HTTP al puerto de escucha de la aplicación elegida, mediante la URL correspondiente a dicha aplicación (En la imagen de ejemplo, esta es "http://localhost:3001/products/64a1b3b73d992df2d96b970f")

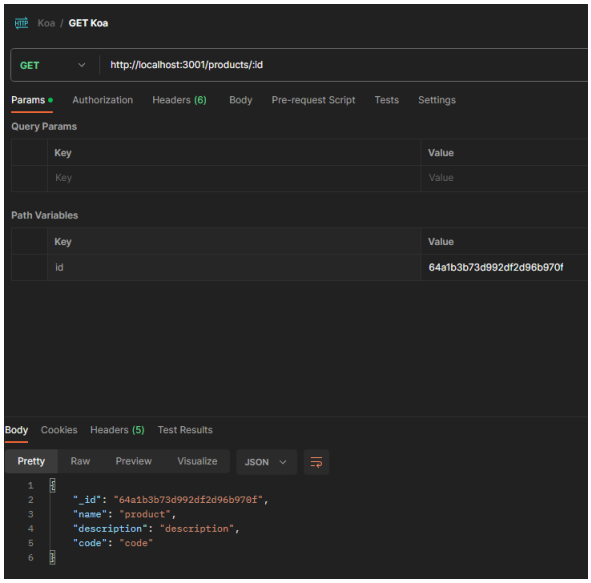

Figura 16.3: Ejemplo Peticion Postman

- **APLICACIÓN - BASE DE DATOS**: La aplicación se conecta con la base de datos para efectuar las instrucciones recibidas mediante un protocolo especificado en los drivers propios de la base de datos, que varían según el lenguaje utilizado. De esta forma, en la aplicación se especifica el denominado Çonnection String"de la base de datos en cuestión, y una vez iniciada la aplicación ambos estarán conectados.
- **POSTMAN - JMETER**: En los ajustes de la aplicación de Postman, se puede especificar un servidor proxy intermediario entre el puerto con el que se esta comunicando y la propia aplicación. Si se especifica el mismo proxy que se crea en la aplicación de JMeter, es posible escuchar desde JMeter las peticiones realizadas al puerto de la aplicación con la que se está tratando de comunicar Postman.

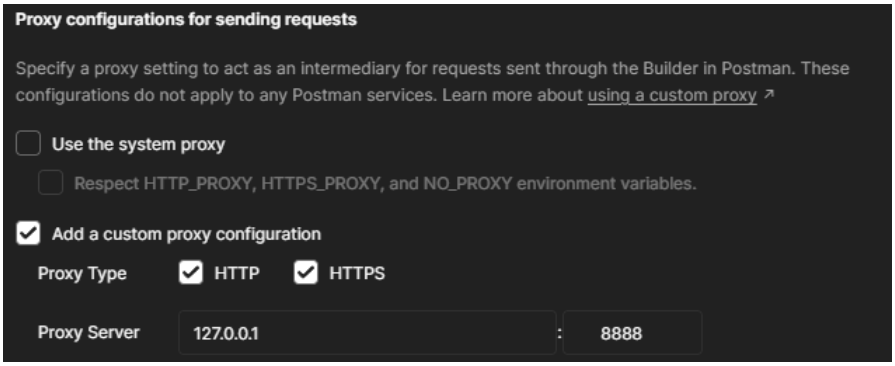

Figura 16.4: Configuración para elegir proxy intermediaro en Postman

16.3. Anexo - Implementación del sistema

16.3.1. Preparación del entorno

Previo al desarrollo de ninguno de los componentes, se clonó el repositorio previamente creado por los antiguos responsables del proyecto y se creó el resto de ramas no creadas aún (Koa y Express).

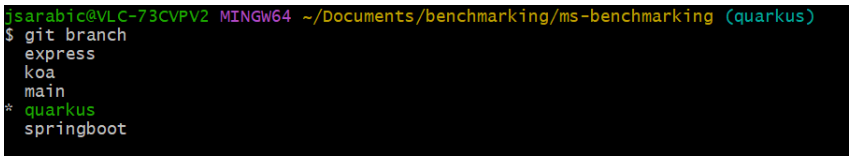

Figura 16.5: Repositorio del proyecto en Git

Posteriormente se abrieron los puertos que más tarde serían los puertos de escucha de cada una de las aplicaciones. Estos serán:

- **localhost:8080**: Este será el puerto de la aplicación creada con Spring
- **localhost:8081**: Este será el puerto de la aplicación creada con Quarkus
- **localhost:3000**: Este será el puerto de la aplicación creada con Express
- **localhost:3001**: Este será el puerto de la aplicación creada con Koa

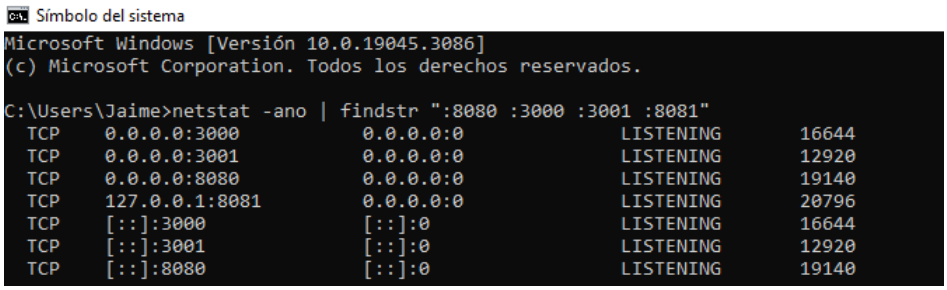

Figura 16.6: Todas las aplicaciones en ejecución en su puerto correspondiente

También se creó la base de datos local MongoDB a la cuál las aplicaciones accederían, junto con su colección denominada "products".

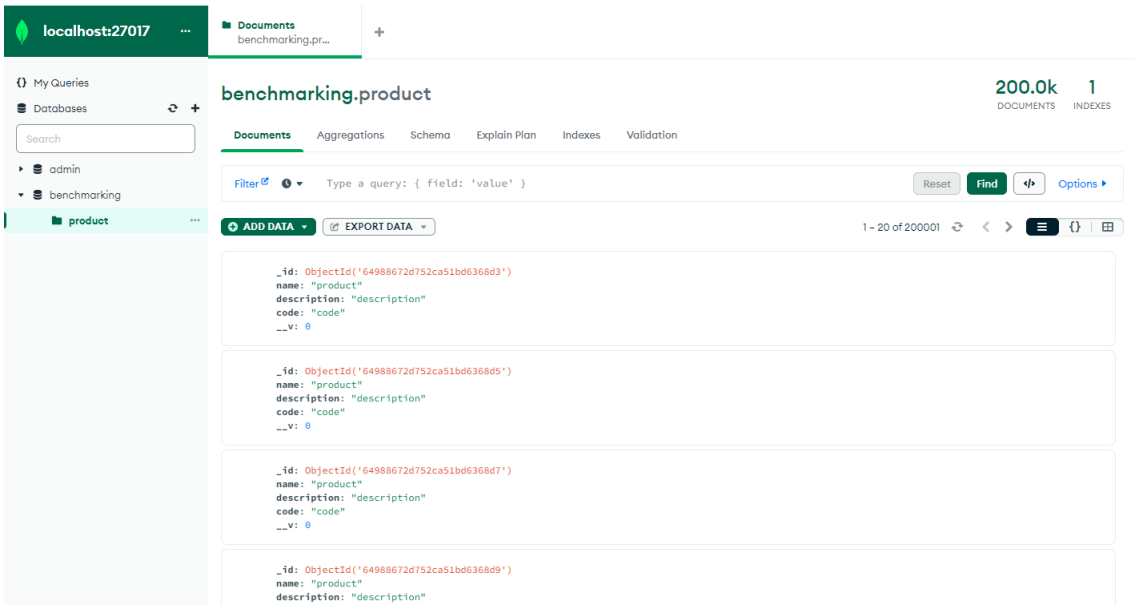

Figura 16.7: Base de datos local MongoDB

16.3.2. Implantación de las aplicaciones

La implantación de cada una de las aplicaciones ha sido muy similar, pero con matices diferenciadores entre ellas. A continuación se desglosará cada una de sus implantaciones

16.3.2.1. Implantación aplicación Spring

Maven, una de las herramientas utilizadas para el desarrollo de las aplicaciones Java, hace uso de un archivo denominado "pom.xml.[en] el que se añaden las librerías, plug-ins y dependencias que usará la aplicación y esta herramienta se encargará de instalarlas sin necesidad de interactuar. Por ello, en ambas aplicaciones Java, este archivo contiene todas las dependencias necesarias para su funcionamiento. Estas dependencias, son nativas de cada uno de los frameworks a analizar, por lo que el rendimiento final de las aplicaciones dependerá de la optimización de estas dependencias creadas por los desarrolladores de cada framework.

La aplicación de Spring y la de Quarkus, fue desarrollada por los antiguos responsables del proyecto en el IDE de STS, como se mencionará mas tarde en el

apartado .[Es]tudios con entidad propia", pero su implantación será explicada en este apartado igualmente. La aplicación de Spring utiliza dependencias propias, y en este caso, fueron las siguientes:

```
https://maven.apache.org/xsd/maven-4.0.0.xsd (xsi:schemaLocation)
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.4</version>
    </parent>
    [redacted]
    <artifactId>products</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>products</name>
    <description>Product API</description>
    <properties>
        <java.version>11</java.version>
        <modelmapper.version>3.1.0</modelmapper.version>
        <openapi-ui.version>1.6.6</openapi-ui.version>
        <jackson-databind.version>0.2.3</jackson-databind.version>
        <embedded-mongo.version>3.4.11</embedded-mongo.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-mongodb</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.modelmapper</groupId>
            <artifactId>modelmapper</artifactId>
            <version>${modelmapper.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springdoc</groupId>
            <artifactId>springdoc-openapi-ui</artifactId>
            <version>${openapi-ui.version}</version>
        </dependency>
        <dependency>
```

Figura 16.8: Archivo pom.xml de Spring

En general se trata de librerías que facilitan las conexiones con bases de datos MongoDB y librerías necesarias para la creación de aplicaciones web.

Por otra parte, existe otro archivo con extensión ".yaml.[en] el que está especificadas ciertas "variables de entorno". Estas variables de entorno consisten en elementos que pueden ser modificados o introducidos de forma externa al código de la aplicación para tratar de ejecutar esta bajo las opciones requeridas en ese momento. Pueden existir diversos archivos de este tipo para una misma aplicación, y es previo a la ejecución de esta cuando se escoge que "profileüsar. El archivo que se observa a continuación se trata del profile "dev", que es el que se utilizó durante el desarrollo de la aplicación:

```yaml
# HTTP Port
server:
  port: ${PORT_DEV_SP}

spring:
  # Spring data mongo starter
  data:
    mongodb:
      uri: ${URI_MONGO_SP}
      database: ${DB_MONGO_DEV_SP}

# Spring Doc
spring-doc:
  api-docs:
    enabled: ${APIDOC_ENAB_DEV}

#Swagger UI
swagger-ui:
  info:
    title: Product API (development)
```

Figura 16.9: Archivo YAML de Spring

Posterior a esto, se desarrolló toda la aplicación con todo el código necesario para el correcto funcionamiento y su capacidad de conexión con la base de datos.

Por último, se creó la llamada run configuration"para la ejecución de la aplicación, que en el IDE de STS, se trata de una forma de ejecutar una aplicación mediante unas opciones determinadas por el usuario (elegir archivo main, que profile de YAML utilizar, y definir las variables de entorno que se vayan a utilizar). Como este IDE está desarrollado para soportar aplicaciones desarrolladas con el framework de Spring, existe una opción denominada "Spring Boot application", que fue la escogida para esta aplicación.

16.3.2.2. Implantación aplicación Quarkus

Tanto Spring como Quarkus utilizan Java por lo que todas las explicaciones se han dado en el apartado anterior. A continuación se adjuntaran los archivos pom.xml y .yaml:

```
https://maven.apache.org/xsd/maven-4.0.0.xsd (xsi:schemaLocation)
<?xml version="1.0"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd" xmlns="http:/
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <modelVersion>4.0.0</modelVersion>
  [redacted]
  <artifactId>products</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <properties>
    <compiler-plugin.version>3.8.1</compiler-plugin.version>
    <embedded-mongo.version>2.13.0.Final</embedded-mongo.version>
    <lombok.version>1.18.22</lombok.version>
    <maven.compiler.release>11</maven.compiler.release>
    <modelmapper.version>2.3.8</modelmapper.version>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    <quarkus.platform.artifact-id>quarkus-bom</quarkus.platform.artifact-id>
    <quarkus.platform.group-id>io.quarkus.platform</quarkus.platform.group-id>
    <quarkus.platform.version>2.12.2.Final</quarkus.platform.version>
    <skipITs>true</skipITs>
    <spring-quarkus.version>5.2.17.RELEASE</spring-quarkus.version>
    <spring-web.version>5.2.SP4</spring-web.version>
    <surefire-plugin.version>3.0.0-M7</surefire-plugin.version>
  </properties>
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>${quarkus.platform.group-id}</groupId>
        <artifactId>${quarkus.platform.artifact-id}</artifactId>
        <version>${quarkus.platform.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>io.quarkus</groupId>
      <artifactId>quarkus-resteasy-jackson</artifactId>
    </dependency>
    <dependency>
      <groupId>io.quarkus</groupId>
      <artifactId>quarkus-mongodb-panache</artifactId>
```

Figura 16.10: Archivo pom.xml de Quarkus

```
# DEV CONFIG

# HTTP Ports
quarkus:
  # Datasource Configuration
  # Local MongoDB config
  mongodb:
    connection-string: ${URI_MONGO_QU}
    database: ${DB_MONGO_DEV_QU}

  swagger-ui:
    always-include: ${SWAGG_INCL_DEV}

  http:
    port: ${PORT_DEV_QU}

# Swagger UI
mp:
  openapi:
    extensions:
      smallrye:
        info:
          title: Product API (development)
```

Figura 16.11: Archivo YAML de Quarkus

Posterior a esto, se desarrolló toda la aplicación con todo el código nece-

sario para el correcto funcionamiento y su capacidad de conexión con la base de datos.

En cuanto a la "run configuration", debido a que no existía la posibilidad de ejecutar una aplicación de Quarkus de forma nativa en el IDE, se tuvo que descargar desde el marketplace de Eclipse un plugin llamado "Quarkus Tools", para poder ejecutar aplicaciones basadas en este framework:

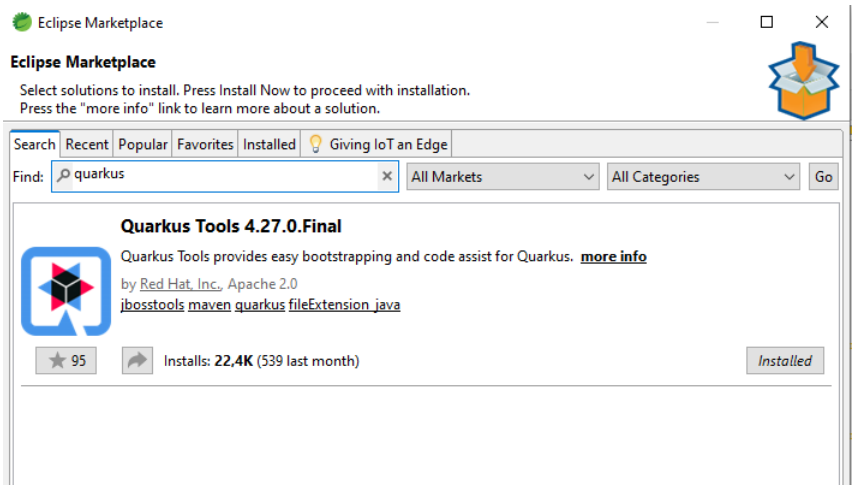

Figura 16.12: Plugin Quarkus Tools

16.3.2.3. Implantación aplicación Express

Antes de comenzar a desarrollar la aplicación, fue necesario crear un directorio en el que estaría hospedado todo el código. Una vez generado, con el fin de crear el entorno node.js necesario para la ejecución de los programas, dentro del repositorio se utilizó el comando "npm init", que inicializó el proyecto node en ese repositorio.

Posteriormente, se descargaron todas las dependencias necesarias para el funcionamiento de la aplicación utilizando los comandos "npm install", seguido de la dependencia que se quería instalar. Estas dependencias puede ser observadas en la Figura 16.12.

De forma similar a lo que se ha mencionado de los archivos pom en las aplicaciones Java, en el entorno de aplicaciones basadas en Node.js, existe

un archivo denominado package.json, en el cual se pueden observar todas las dependencias importadas en el proyecto, junto con la versión que se ha descargado. De hecho, los propios frameworks Express y Koa se reflejan en este archivo, para poder así utilizar sus herramientas durante el desarrollo de las aplicaciones Javascript. En el caso de la aplicación de Express, su archivo package.json tiene el siguiente contenido:

Figura 16.13: Archivo package.json de Express

Las aplicaciones de Express y Koa fueron desarrolladas en el IDE de Visual Studio Code. Como ya se ha mencionado previamente, estas aplicaciones tienen una estructura basada en el modelo vista-controlador. De esta forma, cada elemento de la aplicación esta destinada a una sola tarea. El archivo routesrecibe las peticiones HTTP, y llama a métodos contenidos en el archivo "controller". A continuación se muestra el contenido de ambos archivos:

```
const express = require("express");
const productRouter = express.Router();
const productsController = require('../controllers/productsController');

productRouter.get("/", productsController.getAllProducts)
productRouter.get("/:id", productsController.getProduct);
productRouter.post("/", productsController.createProduct);
productRouter.put("/:id", productsController.updateProduct);
productRouter.delete("/:id", productsController.deleteProduct);
productRouter.delete("/", productsController.deleteAllProducts);

module.exports = productRouter;
```

Figura 16.14: Archivo "productRoutes"de Express

```
const Product = require('../models/product');

const ProductController= {

 getAllProducts : async (req, res) => {

    try {
        let allProducts = await Product.find();
        return res.status(200).json({
            succes: true,
            allProducts,
        });

     } catch (error) {
            return res.status(500).json({
                success: false,
                message: error.message,
            });
        }

 },

 getProduct : async (req, res) => {
   try {
    const {id} = req.params;
    let product = await Product.findById(id);
    if (!product) {
        return res.status(404).send({
            success: false,
            message: "Producto no encontrado",
        });
    }

    return res.status(200).send({
        success: true,
        product
```

Figura 16.15: Archivo "productController"de Express

Para lanzar a ejecución esta aplicación será necesario ejecutar en la terminal de comandos la instrucción "node server.js ", el cual inicializa el contenido del archivo "server.js "que contiene lo siguiente:

```
const express = require("express");
const app = express();
const mongoose = require('mongoose');
const router = require('./routes/products');
require('dotenv').config();
const port = process.env.PORT || 3000;
app.use(express.json({extended: true}));
app.use(express.urlencoded({extended: true}));

app.use("/products", router);

const URL = process.env.MONGODB_URL;
mongoose
        .connect(URL, {})
        .then(()=>{
          console.log("BD is now connected");
        })
        .catch((err)=>{
          console.log(err);
        });

app.listen(port, () =>{
  console.log("Server listening at port 3000");
});
```

Figura 16.16: Archivo "server.js"de Express

16.3.2.4. Implantación aplicación Koa

Todos los pasos son idénticos a los utilizados en la implementación de la aplicación de Express, con la diferencia de las dependencias descargadas, y el desarrollo de los archivos çontroller z routes ". A continuación, se adjuntarán estos archivos junto con el archivo package.json de la aplicación de Koa:

```
{
  "name": "koa_benchmarking",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  ▷ Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "-": "^0.0.1",
    "@koa/router": "^12.0.0",
    "body-parser": "^1.20.2",
    "dotenv": "^16.1.4",
    "koa": "^2.14.2",
    "koa-bodyparser": "^4.4.0",
    "mongodb": "^5.6.0",
    "node": "^20.2.0",
    "save": "^2.9.0"
  }
}
```

Figura 16.17: Archivo package.json de Koa

```
const Router = require("@koa/router");

const {createProduct, getProduct, getProducts, updateProduct, deleteProduct} = require('../API/productsAPI')

const router = new Router({
    prefix:'/products'
})

router.get('/', async ctx =>{
    ctx.body = await getProducts();
})

router.post('/', async ctx=>{
    let product = ctx.request.body;
    product = await createProduct(product);
    ctx.response.status = 200;
    ctx.body = product;
})

router.get('/:id', async ctx=>{
    const id = ctx.params.id;
    ctx.body = await getProduct(id)
})

router.delete('/:id', async ctx=>{
    const id = ctx.params.id;
    await deleteProduct(id);
})

router.put('/:id', async ctx=>{
    const id = ctx.params.id;
    let product = ctx.request.body;
    product = await updateProduct(id,product);
    ctx.response.status = 200;
    ctx.body = product;
})

module.exports = router;
```

Figura 16.18: Archivo "productRoutes"de Koa

```
const products = require('./index').db('benchmarking').collection('product');
const ObjectId = require('mongodb').ObjectId;

const save = async ({name, description, code}) => {
    const result = await products.insertOne({name, description, code});

    return result;
}

const getAll = async () =>{
    const cursor = await products.find();
    return cursor.toArray();
}

const getById = async (id) =>{
    return await products.findOne({_id: new ObjectId(id)});
}

const update = async (id, {name, description, code}) =>{
    const result = await products.replaceOne({_id:new ObjectId(id)}, {name ,description, code});
    return result
}

const removeById = async id =>{
   const result = await products.deleteOne({_id:new ObjectId(id)});
    return result;
}

module.exports = {getAll, getById, removeById, save, update};
```

Figura 16.19: Archivo "products"de Koa

La aplicación de Koa se desplegará también mediante la instrucción "node server.js", el cual inicializa el contenido del archivo "server.js "que contiene lo siguiente:

```
const Koa = require('koa');
const bodyParser = require('koa-bodyparser');
const productRoutes = require('./routes/productRoutes');
require('dotenv').config();
const app = new Koa();
const port = process.env.PORT;
app.use(bodyParser());
app.use(productRoutes.routes()).use(productRoutes.allowedMethods());
app.listen(port);
console.log("App is running in port 3001")
```

Figura 16.20: Archivo "server.js"de Koa

16.3.3. Implantación del proyecto de Postman

Posterior a la implantación de las aplicaciones, se realizó la creación de las peticiones con las cuales se interactuaría con ellas. Estas peticiones consisten en las cabeceras GET, POST, PUT y DELETE, seguidas de la URL correspondiente a la petición deseada. El proyecto implementado en Postman tiene el siguiente aspecto:

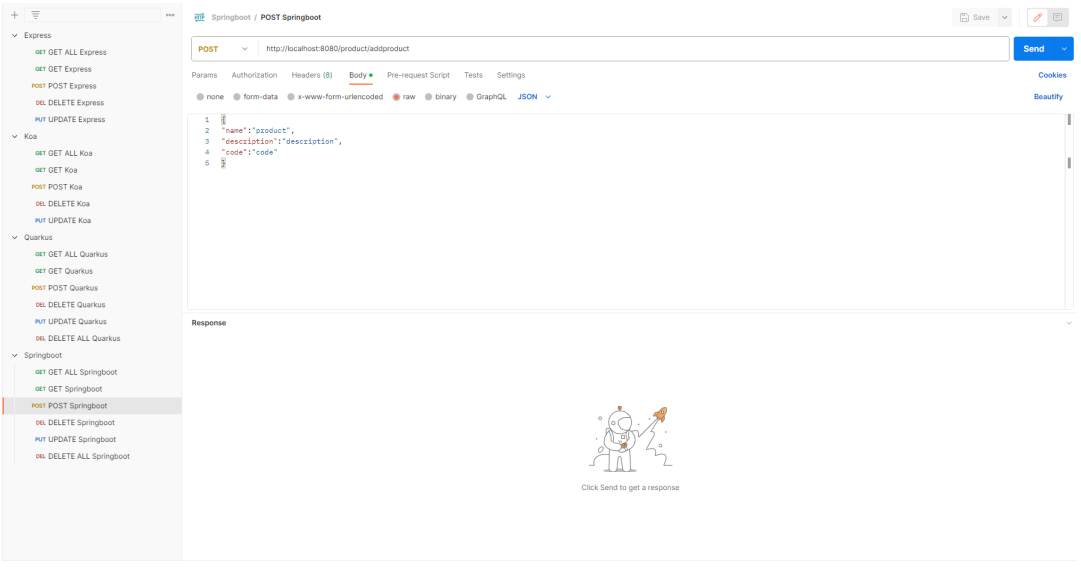

Figura 16.21: Proyecto con las peticiones de Postman

A su vez, se configuró los ajustes de la aplicación para utilizar como servidor intermediario entre Postman y la aplicación web un proxy de JMeter que recoje el esqueleto de las peticiones realizadas (Véase Figura 16.3)

16.3.4. Implantación del proyecto de JMeter

Por último, se creó un proyecto en JMeter que pudiese realizar las pruebas de rendimiento correspondientes a cada aplicación web. Este proyecto constaría de 4 servidores proxy vinculados a 4 grupos de hilos diferentes, que escucharían las peticiones realizadas por la aplicación Postman, para posteriormente repetirlos siguiendo una serie de parámetros establecidos.

Por otra parte, cada uno de estos grupos de hilos estaban conformados por todas las peticioens escuchadas desde Postman y un Receptor"denominado Ïnforme Agregado", que recoge información referente al rendimiento de las aplicaciones correspondientes en base a las respuestas recibidas. Estos grupos de hilos, además, tiene la capacidad de editar la cantidad de peticiones por segundo que se van a realizar una vez sean inicializados, así como también el número de veces que va a repetir esta cantidad. A continuación, se adjunta una imagen con el proyecto de JMeter al completo, junto con un ejemplo de la cantidad peticiones simultáneas que se realizarían y el número de veces que sería repetida:

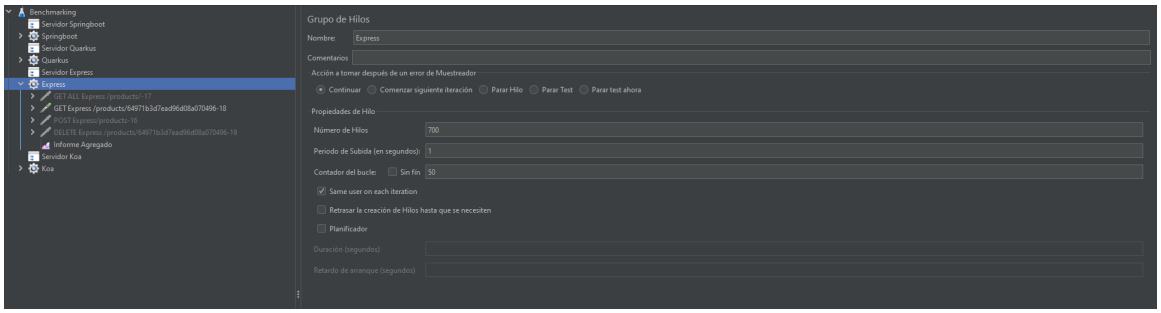

Figura 16.22: Proyecto de pruebas de JMeter

16.4. Anexo - Resultados de las pruebas

Como conclusión a este proyecto, se buscaba obtener una serie de resultados que reflejasen las diferencias en términos de rendimiento de las diversas aplicaciones. Las parámetros que se han analizado para realizar esta comparación han sido explicados en el apartado 4.2.1 ” Análisis cuantitativo”.

Las pruebas se han realizado de la siguiente forma:
A cada una de las aplicaciones se les ha sometido a un flujo de 200 peticiones simultáneas (tanto GET como POST) hasta un total de 200.000, analizando cuantas peticiones por segundo era capaz de realizar esta aplicación. Después de esto, se ha ido aumentando el número de peticiones concurrentes hasta que comenzara a fallar el servidor por culpa de la gran carga. Por último, se han realizado 100 peticiones individuales de gran peso (GET ALL con la base de datos llena de 100.000 products.

Una vez realizadas las pruebas, se han obtenido los siguientes resultados:

- **Quarkus vs Spring**:

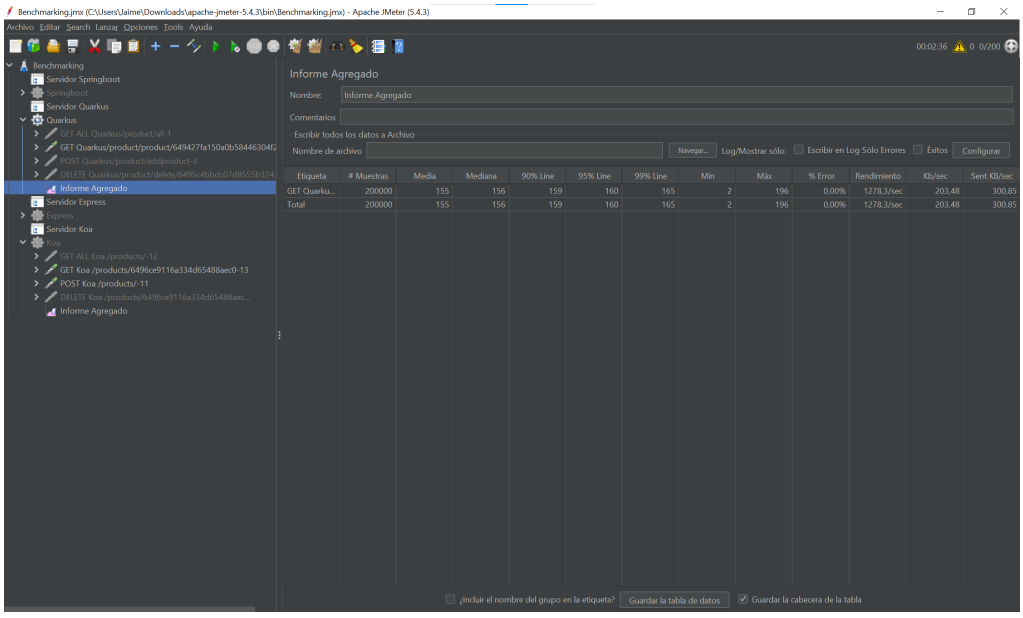

Figura 16.23: 200.000 peticiones GET en Quarkus

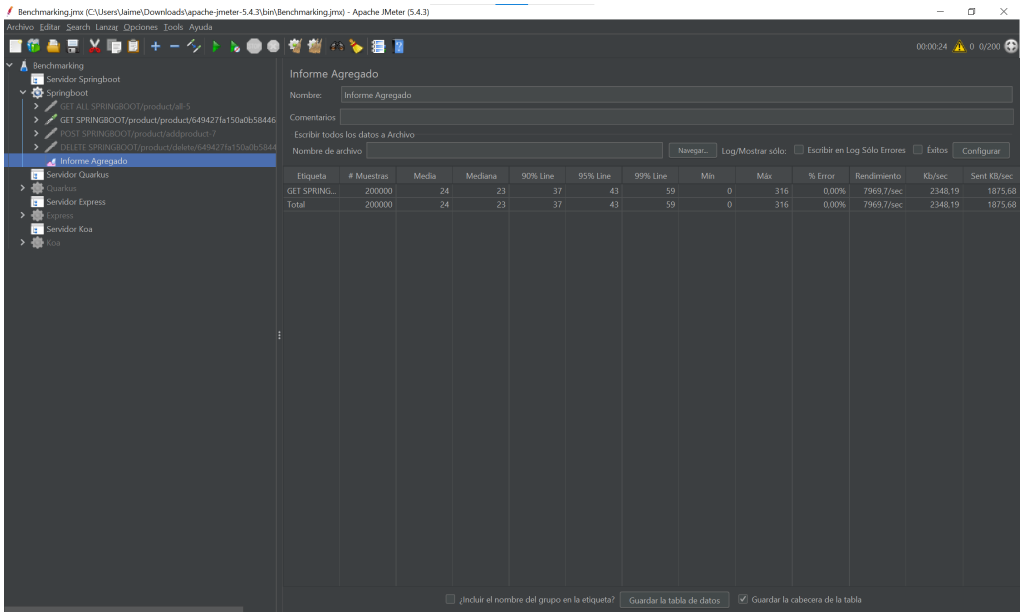

Figura 16.24: 200.000 peticiones GET en Spring

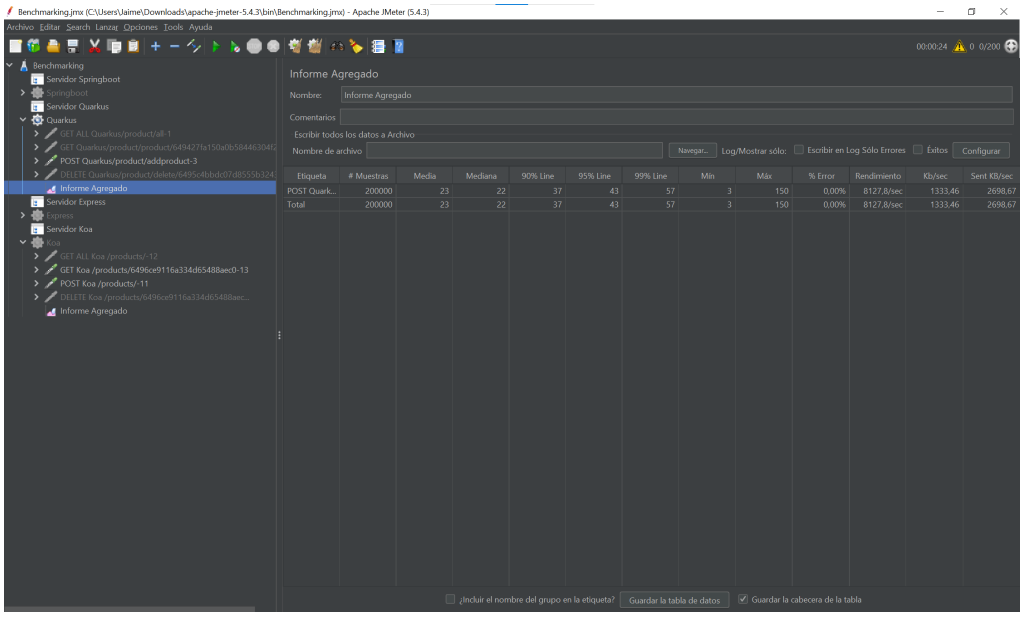

Figura 16.25: 200.000 peticiones POST en Quarkus

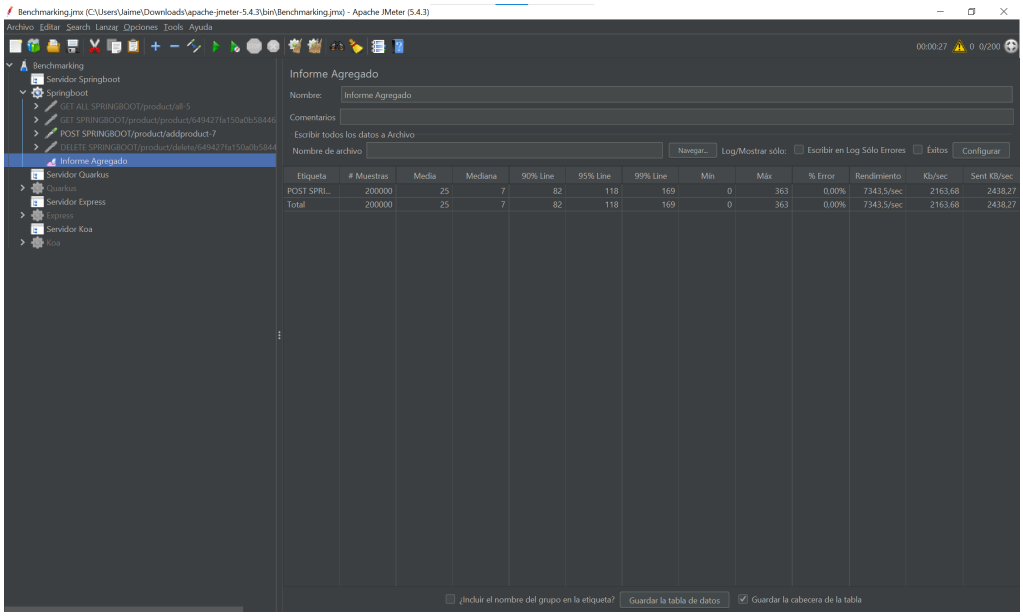

Figura 16.26: 200.000 peticiones POST en Spring

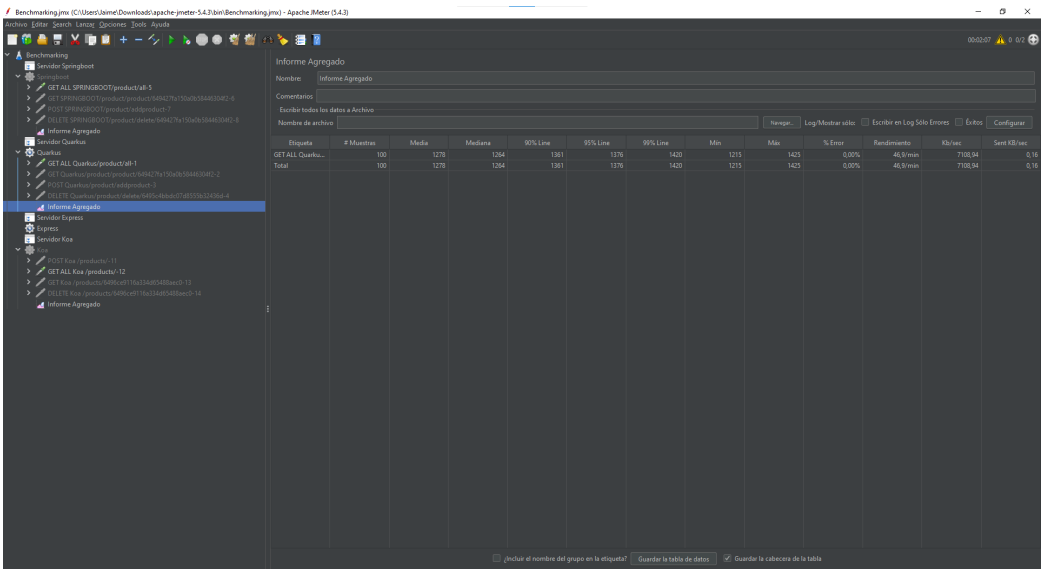

Figura 16.27: 100 peticiones GET ALL en Quarkus

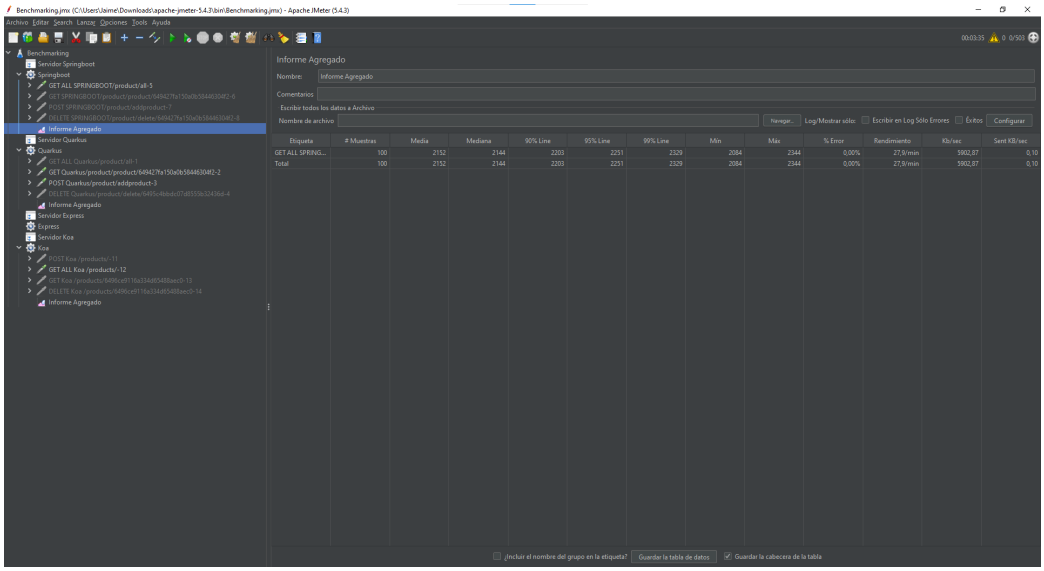

Figura 16.28: 100 peticiones GET ALL en Spring

Analizando los resultados obtenidos, se han obtenido las siguientes conclusiones:

- Quarkus soporta un 60 % mas de peticiones concurrentes que Spring
- Quarkus presenta un grave problema con las peticiones GET, ya que estas son un 550 % más lentas que las realizadas en las aplicaciones de Spring
- Quarkus realiza las peticiones de alto coste casi un 70 % más rápido que Spring

A continuación se adjuntan una serie de gráficas que reflejan estos resultados:

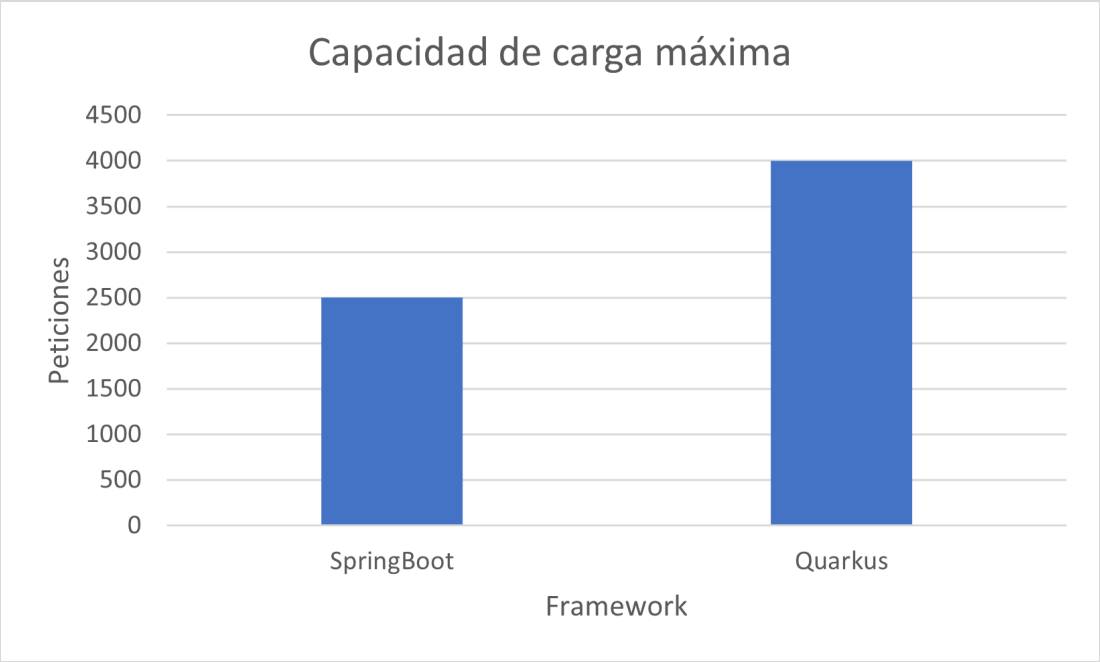

Figura 16.29: Capacidad de carga máxima Spring vs. Quarkus

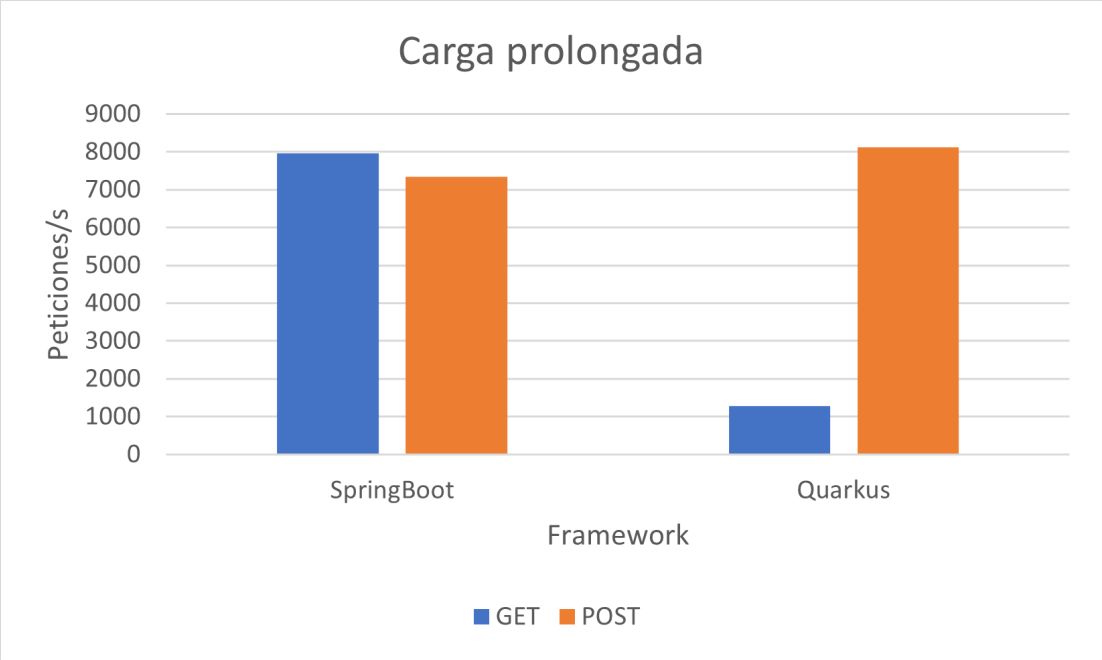

Figura 16.30: Carga prolongada Spring vs. Quarkus

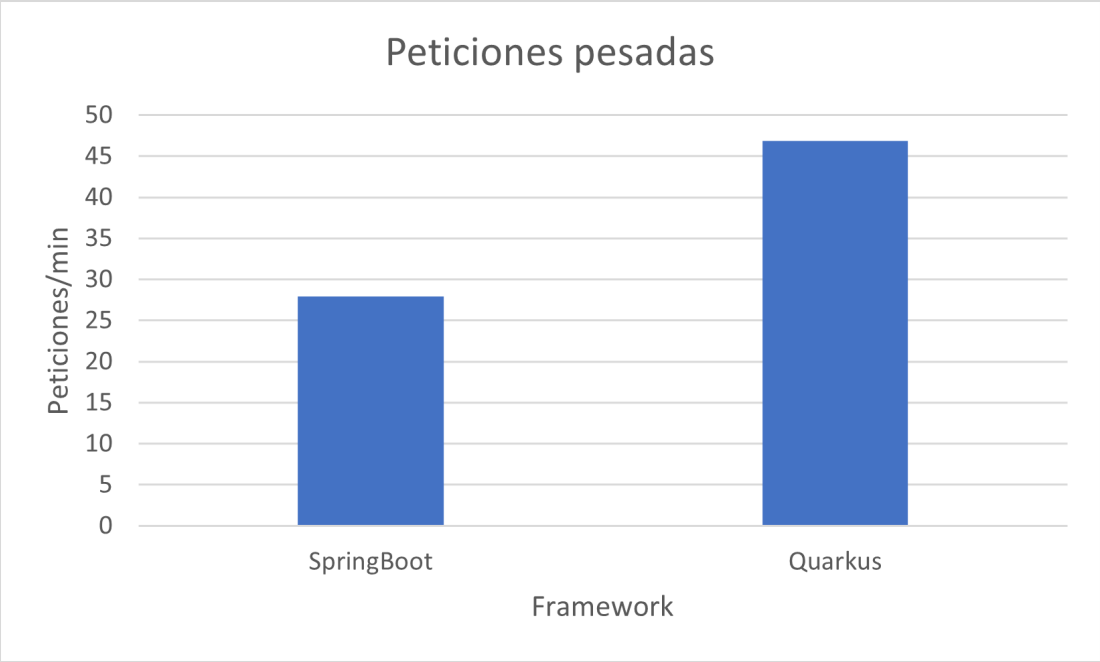

Figura 16.31: Peticiones pesadas Spring vs. Quarkus

- **Express vs Koa**:

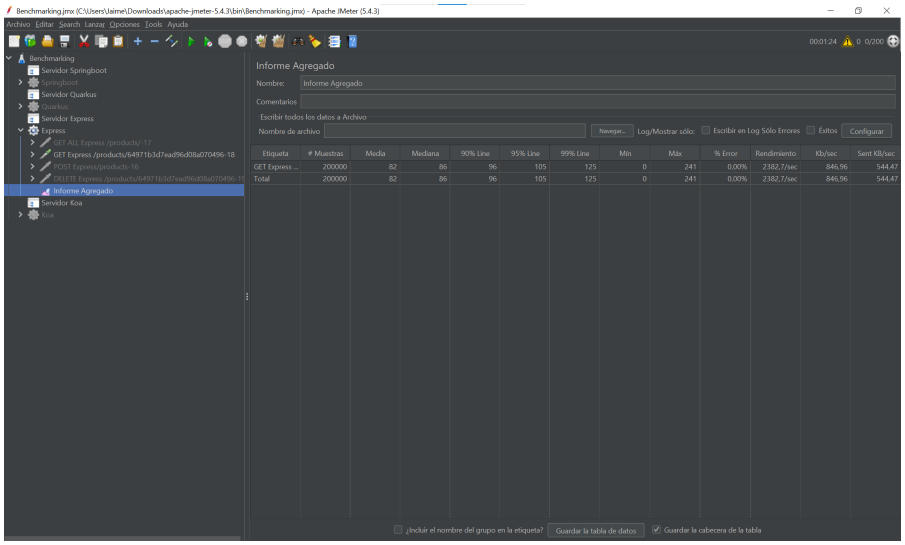

Figura 16.32: 200.000 peticiones GET en Express

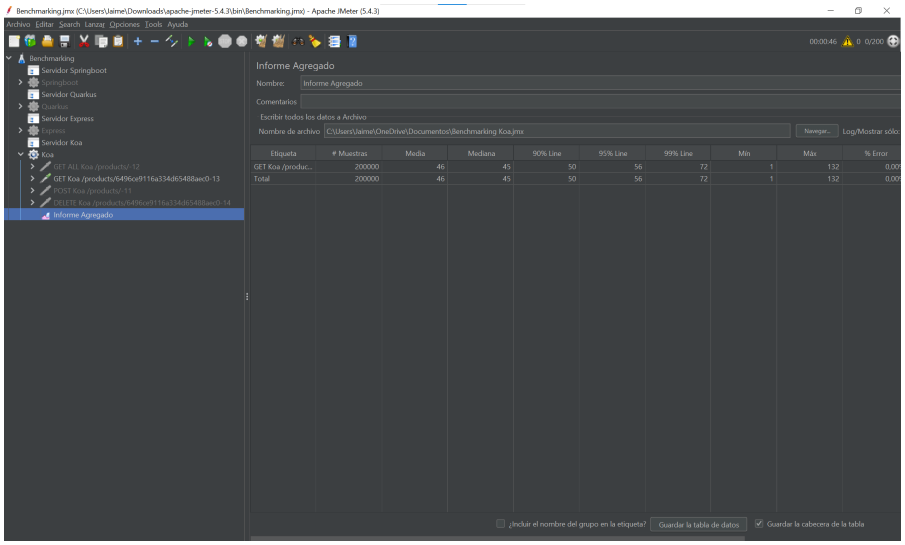

Figura 16.33: 200.000 peticiones GET en Koa

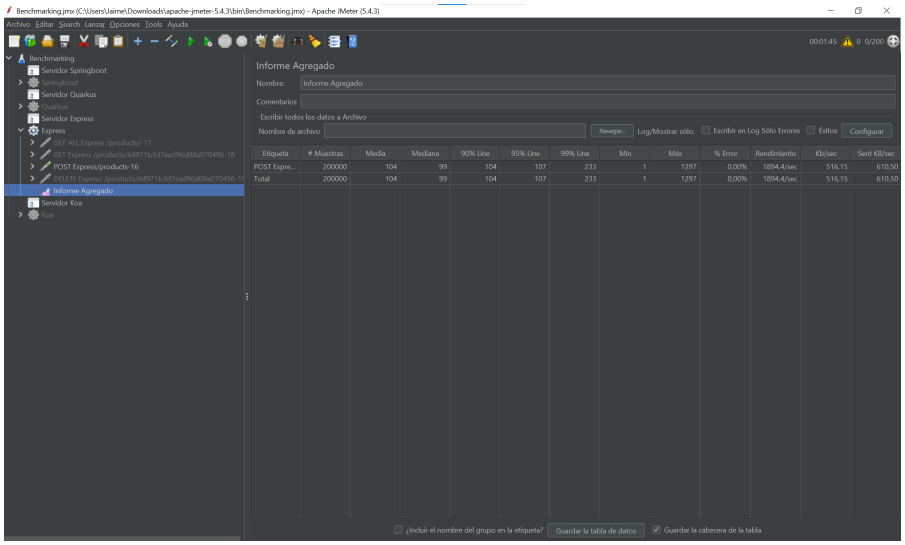

Figura 16.34: 200.000 peticiones POST en Express

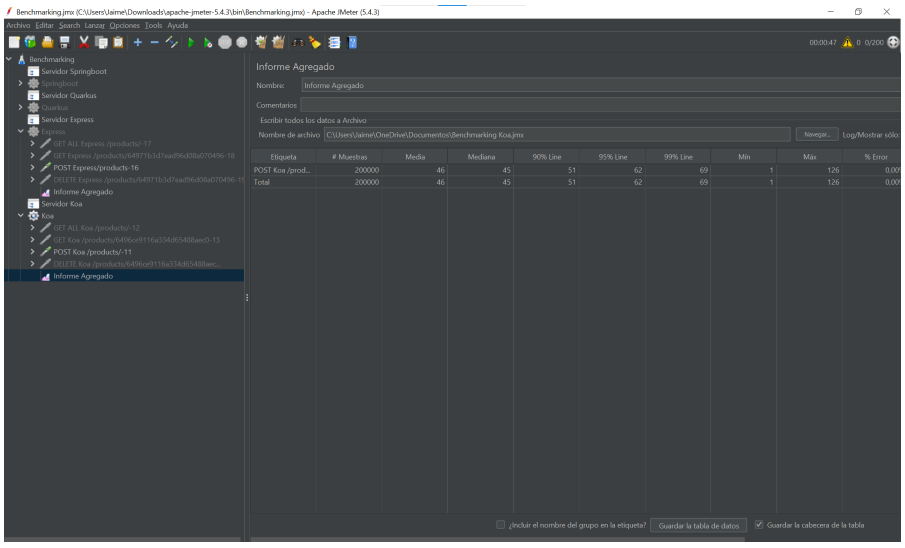

Figura 16.35: 200.000 peticiones POST en Koa

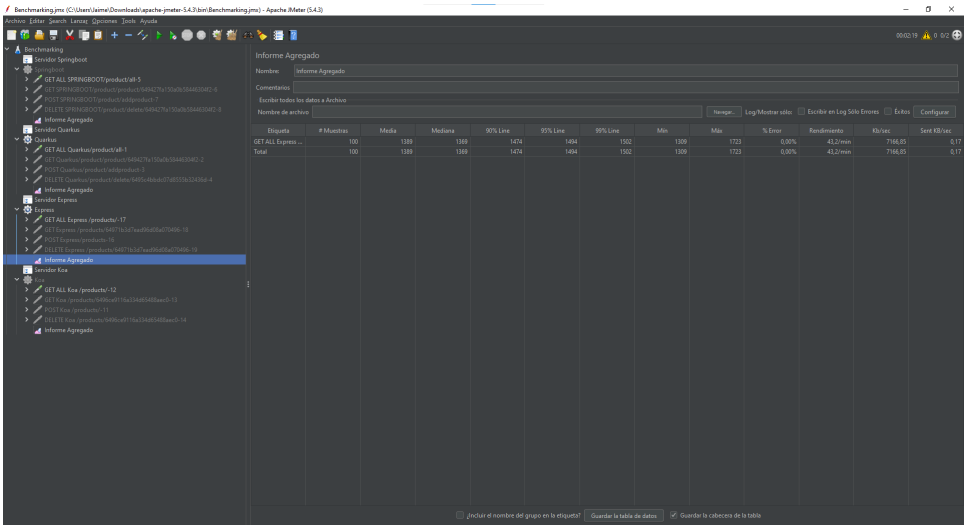

Figura 16.36: 100 peticiones GET ALL en Quarkus

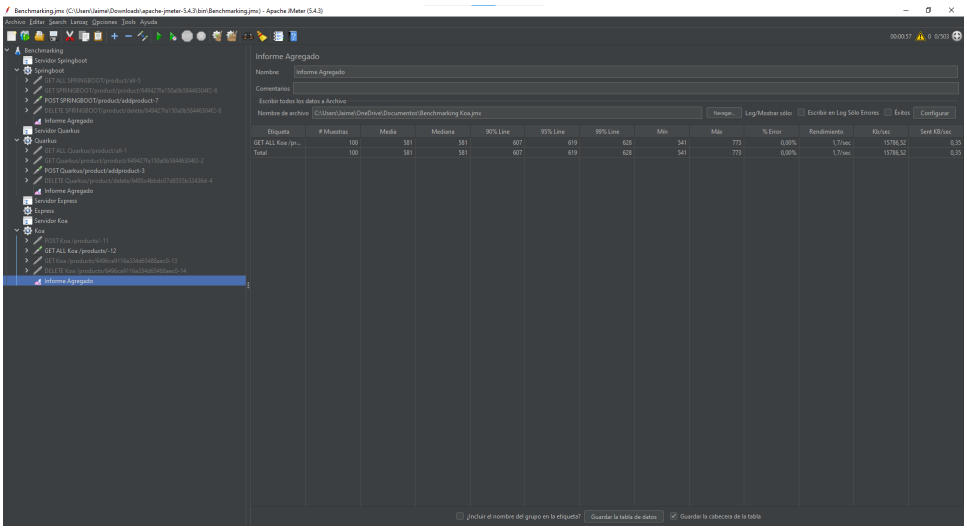

Figura 16.37: 100 peticiones GET ALL en Koa

En base a estos resultados, podemos obtener las siguientes conclusiones:

- Koa soporta un 15 % más de peticiones concurrentes en comparación con Express
- Las peticiones POST de Koa son un 123 % más rápidas que las de Express, y las peticiones GET unn 82 %
- Koa realiza las peticiones de alto coste un 143 % más rápido que Express, siendo el framework que más rápido realiza este tipo de peticiones
- Los tiempos de respuesta a las peticiones de Koa son mucho más consistentes que los de Express.

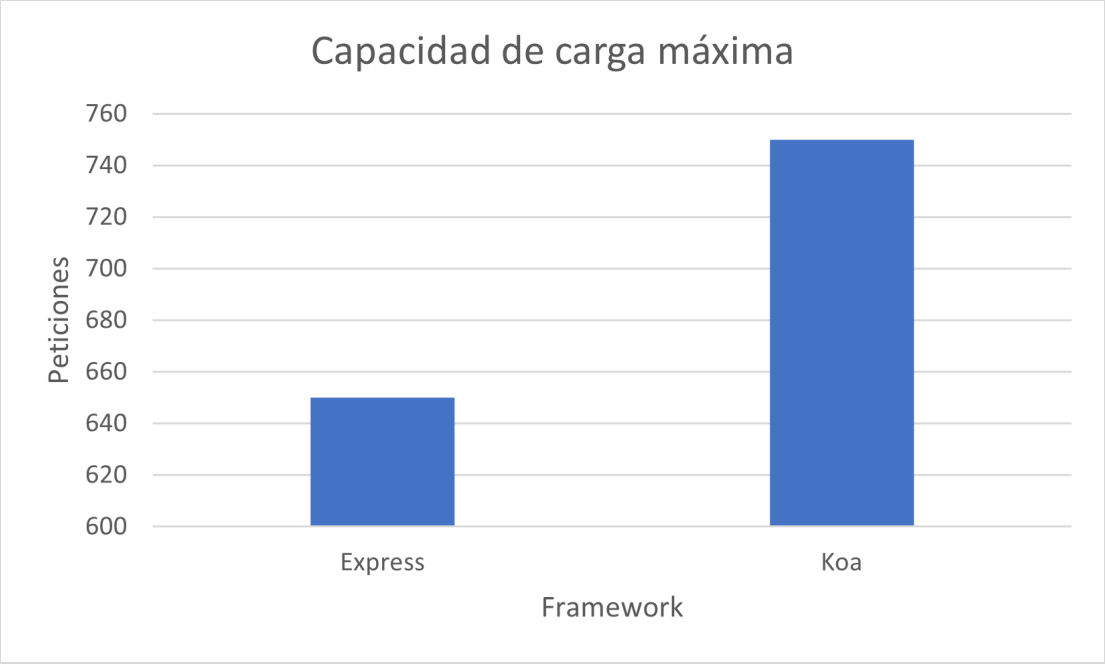

Figura 16.38: Capacidad de carga máxima Express vs. Koa

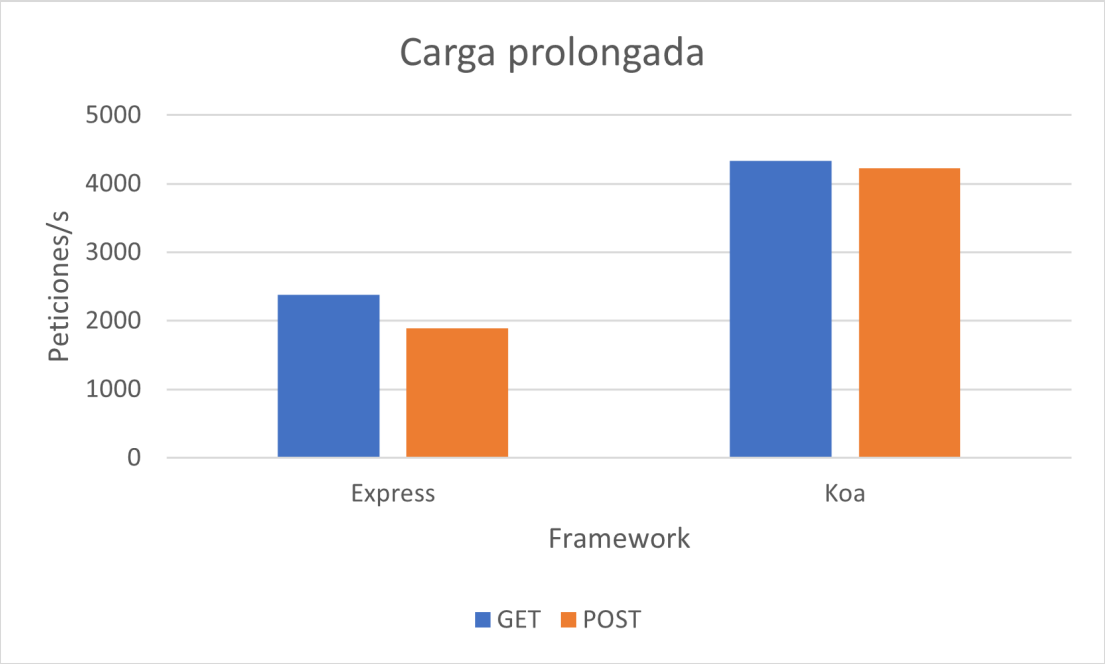

Figura 16.39: Carga prolongada Express vs. Koa

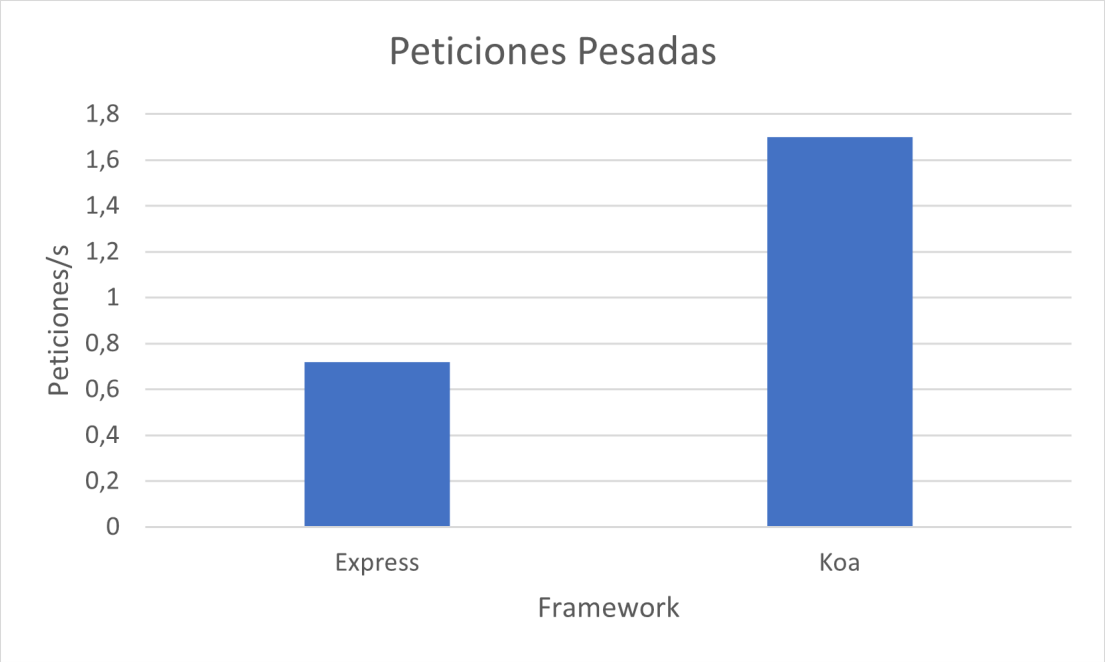

Figura 16.40: Peticiones pesadas Express vs. Koa

16.5. Anexo - Objetivos de Desarrollo Sostenible

El 25 de septiembre de 2015, los líderes de todo el mundo aprobaron un conjunto de metas globales con el fin de eliminar la pobreza, preservar el medio ambiente y garantizar el bienestar para todas las personas, como parte de una nueva estrategia de desarrollo sustentable. Cada meta cuenta con objetivos específicos que deben ser logrados en los próximos 15 años.

Objetivo de Desarrollo Sostenible	Alto	Medio	Bajo	No procede
ODS 1. Fin de la pobreza.				✓
ODS 2. Hambre cero.				✓
ODS 3. Salud y bienestar.				✓
ODS 4. Educación de calidad.		✓		
ODS 5. Igualdad de género.				✓
ODS 6. Agua limpia y saneamiento.				✓
ODS 7. Energía asequible y no contaminante.				✓
ODS 8. Trabajo decente y crecimiento económico.	✓			
ODS 9. Industria, innovación e infraestructuras.	✓			
ODS 10. Reducción de las desigualdades.				✓
ODS 11. Ciudades y comunidades sostenibles.				✓
ODS 12. Producción y consumo responsables.	✓			
ODS 13. Acción por el clima.				✓
ODS 14. Vida submarina.				✓
ODS 15. Vida de ecosistemas terrestres.				✓
ODS 16. Paz, justicia e instituciones sólidas.		✓		
ODS 17. Alianzas para lograr objetivos.				✓

Tabla 16.1: Grado de relación del proyecto con los Objetivos de Desarrollo Sostenible

Mi proyecto tiene gran relación con los siguientes objetivos:

- **Educación de calidad**:
  El proyecto promueve el objetivo de educación de calidad, al promover el aprendizaje y la adquisición de conocimientos tecnológicos, proporcionando información y análisis sobre los frameworks disponibles, lo que contribuye a una educación de calidad en el campo de la tecnología.

- **Trabajo decente y crecimiento económico**:
  Este proyecto promueve el trabajo decente y el crecimiento económico debido a que busca beneficiar el trabajo bien realizado y remunerar así las herramientas con mejor rendimiento en lugar de herramientas con menor calidad. De esta forma si una herramienta es minoritaria pero tiene mayor calidad que la más utilizada, este proyecto ayuda a que la herramienta minoritaria crezca y pueda incluso mejorar mas aún si cabe en el futuro.
  Por otra parte, fomenta el crecimiento económico ya que anima a las empresas a utilizar herramientas que mejoren el rendimiento y su productividad, haciéndolas crecer económicamente.

- **Industria, innovación e infraestructura**:
  También, este proyecto fomenta la industria, innovación e infraestructura, buscando soluciones que mejoren las condiciones actuales, e innovando en el uso de herramientas software para proyectos futuros en todo el mundo.
  Este objetivo, además, promueve el uso eficiente de los recursos, lo que está directamente relacionado con este proyecto en el que se pretende acabar con el uso impuesto de un framework, y sustituirlo por el que realmente permita un mejor aprovechamiento de los recursos disponibles, tanto humanos (el trabajador empleará su tiempo de manera más productiva al tener una herramienta más eficiente), como materiales (los productos obtenidos serán mejores al estar realizados con una herramienta más adecuada para los requisitos del mismo).

- **Producción y consumos responsables**:
  El proyecto se relaciona con el objetivo de producción y consumos responsables aumentando la producción, eficiencia y la optimización de recursos al ayudar a los desarrolladores a seleccionar los frameworks más eficientes en términos de rendimiento, lo que reduce el consumo de recursos computacionales innecesarios. A su vez, contribuye a la reducción del desperdicio tecnológico al permitir la identificación de frameworks obsoletos o poco eficientes, lo que promueve el consumo responsable de herramientas y evita inversiones innecesarias en tecnologías no adecuadas.

- **Paz, justicia e instituciones sólidas**:
  Al utilizar la Norma CCII-N2016-2 para la realización de la Documen-

tación de Proyectos en Ingeniería Informática, se promueve el uso de estándares y buenas prácticas en el campo de la informática. Esto contribuye a fortalecer las instituciones y promover un entorno más justo y equitativo en el ámbito de la ingeniería informática.

Capítulo 17

Especificaciones del sistema

17.1. Lenguajes de programación

Los lenguajes de programación utilizados en este proyecto han sido Java y Javascript. Las aplicaciones web desarrolladas con los frameworks Spring y Quarkus fueron desarrolladas con Java, mientras que las que utilizaron Express y Koa, fueron desarrolladas con Javascript.

Más concretamente, para Java se ha utilizado el paquete de herramientas Java Development Kit (JDK), en su versión 11.0.18. En cuanto a la versión de node.js (y por tanto la versión de Javascript) se está utilizando la 16.13.0. Otros lenguajes menos importantes utilizados durante el desarrollo del proyecto han sido:

- **Markdown**: para la documentación de las aplicaciones (README).
- **XML**: para los archivos pom de Maven (Java).
- **YAML**: para los archivos de variables de entorno en Java.
- **JSON**: para el almacenamiento de los "products.en MongoDB y su estructura en las aplicaciones.

17.2. Librerías y módulos

Como se ha ido mencionando durante toda la memoria, las librerías y módulos utilizados por las aplicaciones son propios de cada uno de los fra-

meworks utilizados. En el apartado 16.3.2, se pueden ver los archivos en los que se especifican todas estas librerías y módulos en profundidad (Figuras referentes a pom.xml y package.json).

17.3. Frameworks

A pesar de que las características de cada framework analizado son muy variadas entre ellos, también es importante tener en cuenta que versión de cada uno de ellos se ha utilizado para realizar esta comparación.

En el caso de Spring, la versión utilizada es la 2.7.4. Esta versión introdujo varias mejoras significativas,la más importante y utilizada en este proyecto, la introducción del módulo Spring MVC para el desarrollo de aplicaciones web.

Por otro lado, la versión de Quarkus es la 2.12.2. Esta versión marcó un hito importante en el proyecto Quarkus al migrar a Jakarta EE 9 (que es una evolución de Java EE) y proporcionar un mejor soporte para Java 11. También presentó mejoras de rendimiento adicionales y actualizaciones en las extensiones existentes.

Asimismo, para Express se ha utilizado la versión 4.18.2. Es la versión más utilizada y actual de Express. Fue lanzada en 2014 y sigue siendo la versión principal en la actualidad. En esta versión, se mejoró la modularidad, la simplicidad y la flexibilidad del framework, además de introducirse un enrrutador independiente y mejorar el manejo de errores. Por otra parte, proporciona una API más intuitiva y fácil de usar.

Por último, la versión 2.14.2 ha sido la elegida para Koa. Esta versión es la más utilizada en la actualidad. Adoptó plenamente las funciones asíncronas (async/await), utilizadas en el proyecto, para manejar el flujo asíncrono. Además, introdujo una sintaxis más clara y simplificada en comparación con las versiones anteriores.

17.4. Sistema operativo

El Sistema Operativo instalada en el equipo es Windows 10 Pro. Este sistema ofrece las siguientes ventajas:

1. Mayor capacidad de hardware: Windows 10 Pro permite aprovechar al máximo la potencia del hardware de tu equipo, lo que es especialmente beneficioso para proyectos que requieren un rendimiento intensivo, como aplicaciones de diseño gráfico o análisis de datos.

2. Gestión avanzada de memoria: Windows 10 Pro está diseñado para gestionar eficientemente grandes cantidades de memoria RAM, lo que resulta útil para proyectos que trabajan con conjuntos de datos complejos o ejecutan múltiples aplicaciones intensivas al mismo tiempo.

3. Herramientas de seguridad mejoradas: Windows 10 Pro incluye características de seguridad avanzadas, como BitLocker, que permite cifrar los discos duros y proteger los datos confidenciales de tu empresa. También ofrece Windows Defender Advanced Threat Protection (ATP) para detectar y responder a amenazas avanzadas de seguridad.

4. Administración de dispositivos y políticas de grupo: Windows 10 Pro ofrece capacidades de administración de dispositivos y políticas de grupo mejoradas. Esto te permite controlar y configurar de forma centralizada los equipos de tu empresa, aplicar políticas de seguridad, instalar software de manera remota y mucho más.

5. Compatibilidad y soporte empresarial: Windows 10 Pro es ampliamente utilizado en entornos empresariales, lo que significa que la mayoría del software y las soluciones de terceros están diseñadas para ser compatibles con este sistema operativo. Además, Microsoft ofrece un soporte extendido y actualizaciones regulares de seguridad para Windows 10 Pro.

6. Integración con servicios y aplicaciones empresariales: Windows 10 Pro se integra con varios servicios y aplicaciones empresariales de Microsoft, como Office 365, Azure, Active Directory y Microsoft Intune. Esto facilita la gestión y la colaboración en proyectos empresariales, así como el acceso a herramientas y recursos adicionales

17.5. Otras aplicaciones

En el desarrollo del proyecto también se han utilizado las siguientes aplicaciones:

- MongoDB: se ha utilizado la versión 6.0.6. Esta versión trajo consigo sobretodo mejoras de rendimiento y arregló una gran cantidad de errores.
- Postman: la versión utilizada es la 7.0.9. Esta versión presentó mejoras adicionales en la experiencia de usuario y la colaboración en equipo. Se introdujeron características como el seguimiento de cambios en las colecciones, la posibilidad de trabajar sin conexión y mejoras en el flujo de trabajo de pruebas y desarrollo.
- JMeter: con la versión 5.6. Esta versión introduce mejoras en el rendimiento y nuevas características, como soporte para la automatización de pruebas de API, mejoras en la visualización de resultados y una mayor estabilidad general.

17.6. Equipo utilizado

El equipo utilizado para realizar las pruebas de rendimiento mediante la aplicación JMeter conectada a su vez a las aplicaciones CRUD creadas previamente tiene las siguientes características:

- Procesador: IntelCore i7-9700F CPU 3 GHz 3 GHZ
- RAM instalada: 16 GB

Capítulo 18

Estudios con entidad propia

Como se ha ido mencionado a lo largo de la memoria, este proyecto fue comenzado por parte de otro grupo dentro de la propia empresa, pero fue traspasado posteriormente. Los avances previos a este traspaso pueden considerarse estudios con entidad propia, porque no han sido desarrollados por parte de el mismo equipo que ha realizado este proyecto. Estos avances consistieron en las aplicaciones, totalmente funcionales, de Spring y Java. Cabe destacar que estas aplicaciones tuvieron que ser modificadas debido a una serie de cambios en los requisitos del proyecto.

Todos los estudios previamente desarrollados por la empresa son de carácter confidencial, por ello no están accesibles al público fuera de la empresa y en esta memoria solo se hace referencia a su existencia pero no a su contenido en sí mismo.

Bibliografía

[1] CCII-N2016-02. https://www.inf.upv.es/www/etsinf/wp-content/uploads/2018/05/norma.pdf.

[2] Estructura de desglose de trabajo. https://www.ionos.es/startupguide/productividad/estructura-de-desglose-de-trabajo/.

[3] Gestión de requisitos. https://www.researchgate.net/figure/Stages-in-the-requirements-management-process-cycle_fig5_320682497.

[4] Ley Orgánica de Protección de Datos. https://www.boe.es/eli/es/lo/2018/12/05/3/con.

[5] Matriz de riesgo. https://www.incibe.es/empresas/blog/analisis-riesgos-pasos-sencillo.

[6] Modelo vista-controlador. https://codingornot.com/mvc-modelo-vista-controlador-que-es-y-para-que-sirve.

[7] NTT DATA website. https://es.nttdata.com/about-us.

[8] Spring Initializr. https://start.spring.io/.

[9] John Carnell and Illary Huaylupo S'anchez. *Spring microservices in action*. Simon and Schuster, 2021.

[10] Badr El Khalyly, Abdessamad Belangour, Mouad Banane, and Allae Erraissi. A comparative study of microservices-based iot platforms. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 11(7):389–398, 2020.