Máster en Cloud Apps
Desarrollo y despliegue de aplicaciones en la nube

Curso académico 2019/2020

Trabajo de Fin de Máster

Programación Asíncrona y Reactiva con *Java*

Autor: Marcos de la Calle Samaniego
Tutor: Micael Gallego Carrillo

Universidad
Rey Juan Carlos

Índice de contenidos

Resumen

Vivimos en un mundo asíncrono.

Esta es la premisa sobre la que pivota todo el TFM de **Programación Asíncrona y Reactiva con *Java***. La explosión del *cloud computing* y la creciente importancia del *software* en todos los aspectos de nuestra sociedad complementan dicha afirmación.

Este proyecto pretende ser un punto de partida para que todo aquel que no haya explorado las características del paradigma asíncrono / reactivo tenga la oportunidad de adentrarse en él siguiendo una ruta de aprendizaje clara. El camino nace con las preguntas:

- **¿Por qué es necesario cambiar nuestro modo de diseñar y programar?**
- **¿Qué ventajas aporta la programación reactiva en nuestras aplicaciones?**

Para responderlas, el contenido se sustenta en **3 pilares**:

- **Un repositorio de código** en *GitHub* como lugar central, que contiene enlaces al resto de información y que aloja los ejemplos.
- **Una presentación a modo de curso** que explica paso a paso las definiciones, porqués, historia e implementación de este paradigma.
- **Unos ejemplos de código** para ilustrar las oportunidades, ventajas e inconvenientes de este modo de diseñar y programar y que sirven de apoyo e hilo conductor de la presentación.

El desarrollo del proyecto, a su vez, ha pasado por **diferentes fases**:

- **Fase 1. Búsqueda, análisis y clasificación del contenido** en la que se consultaron centenares de recursos (artículos, libros, vídeos) y se seleccionaron los más relevantes para ser incluidos en el repositorio y que sirviesen de base para la creación de contenido.
- **Fase 2. Creación de contenido** en la que se plantearon y ejecutaron gran parte de los ejemplos y el contenido principal de la presentación.
- **Fase 3. Puesta en marcha y ciclo de *feedback*** que comenzó con el inicio de la impartición del curso a un grupo de personas y la promoción paulatina del contenido por redes sociales.

En los siguientes apartados se explicará con mayor detalle cada uno de estos aspectos.

En definitiva, la programación asíncrona y reactiva supone un reto y el camino para alcanzar un nivel de conocimiento suficiente sigue una curva de aprendizaje bastante abrupta en su comienzo para después suavizarse y destapar su verdadero potencial.

Este TFM pretende ser el punto de partida para que cualquier persona del sector que lo desee pueda comenzar dicho viaje.

Introducción y objetivos

El lenguaje de programación *Java* lleva entre nosotros mucho tiempo. Casualmente, este año 2020 se produce un hito importante en su historia, su 25 aniversario. En este cuarto de siglo *Java* ha evolucionado mucho y se ha convertido en una herramienta de confianza para muchos proyectos y que está presente a lo largo y ancho del mundo.

En 2002, hace ya 18 años, se produjo otro momento importante en la historia de *Java*: el nacimiento del *framework Spring* que ayudó (y lo sigue haciendo) a realizar aplicaciones basadas en esta tecnología de manera más sencilla.

Esta introducción sobre los inicios de *Java* y *Spring* es pertinente porque, al igual que sucede con todas las tecnologías, éstas no nacen conociendo las necesidades futuras de la industria si no que tienen que evolucionar. *Java* y *Spring* se han visto obligados a ello y en estos momentos se encuentran inmersos en su **revolución reactiva**.

La revolución reactiva

La famosa frase *"Software is eating the world"* escrita por Marc Andreessen en 2011 tiene absoluta vigencia hoy en día.

El gran éxito y avance de la industria IT hace que cada vez una mayor parte de nuestra vida esté guiada por algún tipo de *software*. Las redes sociales o la banca y compra en línea son un buen ejemplo de esta afirmación.

Este éxito conlleva también nuevos retos a la hora de diseñar y programar nuestras aplicaciones. En la actualidad nuestras plataformas son cada vez más distribuidas y dependientes entre sí aparte de contar con un trafico manifiestamente superior.

La programación asíncrona y reactiva tiene como objetivo hacerse cargo de la complejidad creciente y aportar soluciones para los problemas actuales. En noviembre de este año, la ***Reactive Foundation*** presentó la evolución de su manifiesto, los **Principios Reactivos**:

1. Responde siempre de manera adecuada.
2. Acepta la incertidumbre y busca la manera de aplicar patrones de confianza.
3. Abraza el fallo y diseña tus sistemas para ser resistentes.
4. Diseña componentes independientes y que colaboren.
5. Asegura la consistencia individual por componente.
6. Procesa de manera asíncrona para eliminar esperas y evitar coordinación excesiva.
7. Desacóplate de manera espacial y apóyate en la red.
8. Adáptate a una demanda variable de recursos.

Estos principios describen perfectamente los problemas a los que nos enfrentamos en nuestro día a día y muestra que quizá la programación asíncrona y reactiva sea el camino que seguir.

Por otro lado, no podemos obviar la tracción que este paradigma ha tenido en los últimos tiempos. La siguiente figura ilustra la evolución en las tendencias de búsqueda sobre programación reactiva en *Java* en los últimos 10 años:

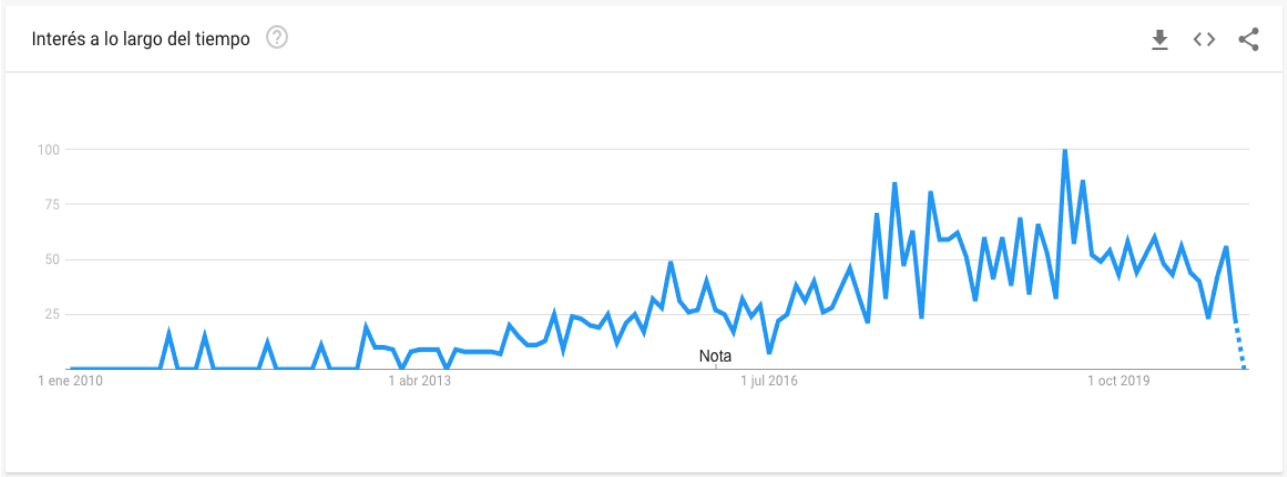

Figura 1 *Tendencias de Google Trends para el término "java reactive programming" desde 2010 hasta la actualidad*

Aligerar la curva de aprendizaje

En puntos anteriores hemos hablado de “revolución” y este hecho implica que el camino para dominar este paradigma es complejo sobre todo viniendo de la programación imperativa.

Por dicho motivo este TFM busca crear una ruta de aprendizaje que facilite la comprensión de todos los conceptos y persigue los siguientes objetivos:

1. Definir de manera sencilla **¿Qué es la programación asíncrona y reactiva?**
2. Mostrar los retos a los que se enfrenta la industria hoy en día y detallar **¿Por qué la programación reactiva puede ser una vía para evolucionar nuestros sistemas?** Además, en este punto se ha incidido en cómo presentar estas bondades desde el punto de vista de negocio ya que una de las barreras iniciales es la justificación de la complejidad que añaden estas soluciones.
3. Conocer la **historia y evolución de los principios reactivos** y más en concretamente en el ecosistema *Java* / *Spring*.
4. **Ilustrar mediante ejemplos** cómo implementar estas soluciones.
5. Ofrecer **consejos sobre los problemas más comunes** y no perder de vista **fases como las de testeo y depuración.**
6. Mostrar cuál puede ser **el futuro de la programación asíncrona y reactiva** en la plataforma *Java* y en el ecosistema *Spring*.

Programación asíncrona y reactiva con Java

El TFM de **Programación Asíncrona y Reactiva con *Java*** tiene como pieza central su repositorio, alojado en *GitHub*:

- **https://github.com/MasterCloudApps-Projects/AsyncReactiveProgramming**

Dentro del mismo aparecen distintos recursos que se describen a continuación.

Enlaces

El repositorio contiene una sección de enlaces seleccionados entre una gran variedad de contenido desde el inicio del proyecto. Están divididos en distintas secciones:

- **Fundamentos**, donde se recoge la información referente a las múltiples especificaciones y manifiestos que conforman el ecosistema asíncrono y reactivo en Java. Entre los más destacados se encuentran El Manifiesto de los Sistemas Reactivos, los Principios Reactivos (de reciente publicación) o la especificación de los *Reactive Streams*.

- **Principales implementaciones de los *Reactive Streams***, que recogen los cuatro actores principales en estos momentos, *Project Reactor*, *RxJava*, *Akka* y *VertX*.

- **Ecosistema *Spring***, contiene enlaces a artículos y vídeos seleccionados que hablan de diferentes aspectos de la evolución reactiva del *framework Spring*. Se puede encontrar contenido sobre fundamentos, buenas prácticas, proyectos recientes como *R2DBC* y *RSocket* o información relacionada con el funcionamiento interno de librerías como *Project Reactor*.

- **Otros ecosistemas**, pretende ser la sección que aglutine las soluciones que aportan otros ecosistemas como *Quarkus* o *Micronaut* al paradigma reactivo.

- ***Project Loom***, aglutina los mejores enlaces sobre el mencionado proyecto de *Java* que pretende cambiar la forma en la que se crean y gestionan los hilos de manera interna. Al ser una tecnología en proceso de creación es fundamental conocer todo aquello que subyace y estar al día de su constante evolución.

- **Librerías y herramientas**, la cual aporta enlaces sobre determinadas utilidades interesantes que pueden ser de gran ayuda en el proceso de migración hacia soluciones asíncronas o reactivas. Aquí encontramos enlaces a visualizadores y herramientas de *profiling* como *VisualVM* o *JMC* o de agentes como *BlockHound*, indispensables en nuestra búsqueda de flujos no bloqueantes.

- **Libros**, que presenta una pequeña lista de lecturas relacionadas con patrones asíncronos y reactivos o programación general reactiva en el ecosistema *Spring*.

- **Cursos**, que contiene enlaces a un puñado de cursos muy interesantes para comprender los modelos de memoria y concurrencia en *Java*, el *framework* de *Executors* o la familia de soluciones reactivas dentro de *Spring*.

- Una sección final que recoge **enlaces a las cuentas en redes sociales** de los autores y autoras más relevantes descubiertos durante el desarrollo del TFM. Asimismo, se aportan **enlaces de cuentas oficiales** de determinados proyectos y un conjunto de **sitios generalistas que aportan información sobre programación.**

Presentación

El TFM se acompaña de una extensa presentación que hace las veces de hilo conductor para ilustrar los diferentes aspectos relacionados con la **Programación Asíncrona y Reactiva con *Java***.

La presentación se puede consultar y descargar en distintos formatos desde la página principal del repositorio y está dividida en diferentes secciones que se detallan a continuación:

- **Secciones 1 y 2, Definición y ¿Por qué?** En estos capítulos se intenta dar una definición formal al concepto de programación reactiva y se listan una serie de ventajas y problemas que intenta resolver este paradigma. Estos apartados están pensados para servir de justificación a la complejidad añadida por este conjunto de prácticas.

- **Sección 3, Historia y evolución.** La cual detalla la evolución de *Java* como lenguaje y describe la irrupción hace aproximadamente una década de las distintas librerías reactivas.

- **Sección 4, Modelo de memoria y concurrencia en *Java*.** Este bloque presenta de manera básica la evolución de los modelos de memoria e hilos y sirve de punto de análisis para descubrir los problemas subyacentes de la plataforma *Java* y como la programación asíncrona y reactiva pretende solventarlos.

- **Sección 5, *Reactive Streams* y *Project Reactor***. Este apartado incide en la especificación de los *Reactive Streams* y en una de sus implementaciones más conocidas. Se presentan conceptos de *publisher*, *subscriber*, *subscription* y *processor* y cómo *Project Reactor* nos brinda los tipos reactivos *Mono* y *Flux*. También se adentra en el concepto de *Scheduler* y su uso.

- **Secciones 6, 7, 8 y 9, *Spring* reactivo**. Estos apartados profundizan en cómo el ecosistema *Spring* ha resuelto los retos que presenta la programación asíncrona y reactiva. Se presenta información sobre *WebClient*, el cliente *HTTP* reactivo que viene sustituir a *RestTemplate*, el acceso reactivo a bases de datos relacionales con *R2DBC* o *RSocket*, un nuevo protocolo de comunicación binario que promete nuevos flujos de información de una manera sencilla.

  Los siguientes bloques tratan sobre las soluciones que este ecosistema ha propuesto para los temas de testeo y depuración, los cuales adquieren un extra de complejidad en contraposición a la programación secuencial e imperativa.

Por último, en el capítulo de trucos y consejos se listan una serie de recomendaciones para comenzar este camino de migración al mundo reactivo evitando caer en errores comunes.

- **Sección 10, Pruebas básicas de rendimiento**. En este punto se recogen publicaciones y comparativas de rendimiento entre el mundo reactivo y el síncrono.

- **Sección 11, *Project Loom***. En la cual se profundiza en mayor medida en la próxima gran revolución de la plataforma *Java* mostrando unos pequeños ejemplos y analizando su comportamiento y las promesas realizadas.

Ejemplos prácticos

El conjunto de recursos aportados se completa con una serie de ejemplos que sirven para guiar la presentación.

Al igual que el resto de información están alojados en el repositorio de GitHub:

- **https://github.com/MasterCloudApps-Projects/AsyncReactiveProgramming/tree/master/examples**

Cabe destacar que cada ejemplo viene acompañado con un fichero de tipo *README* explicando las partes más relevantes y un pequeño listado de guías a seguir para completar el aprendizaje de cada sección.

Excepto los ejemplos más simples, todo el código intenta respetar unas convenciones y buenas practicas y contiene test allá donde sea necesario. El patrón habitual es encontrar el código dividido en paquetes de **dominio, aplicación e infraestructura** que persiguen una separación efectiva de responsabilidades y se inspiran en técnicas comunes de *Domain Driven Design* (DDD) o *Command and Query Responsibility Segregation* (CQRS).

Estos ejemplos no están planteados para que sean excesivamente complejos, pero si para que sean fácilmente actualizables y ampliables en el futuro.

Los ejemplos planteados son los siguientes:

- **Ejemplo 00. *Project Reactor* y operadores.** Contiene una lista de test en los que se ponen a prueba los diferentes operadores que podemos usar en *Project Reactor* junto a sus tipos *Mono* y *Flux*. Además, también contiene ejemplos sobre *Schedulers*, herramientas de *testing* como *StepVerifier*, *TestPublisher* o pequeñas demostraciones de uso de agentes como *ReactorDebugAgent* y *BlockHound*.

- **Ejemplo 01. Comparación básica.** Este ejemplo contiene dos pequeñas aplicaciones que constan de una API simple y persistencia en memoria. Una de ellas está creada utilizando *Spring MVC* y la otra con la alternativa reactiva, *Spring Webflux*. El código también sirve de punto de partida a la explicación de la conveniencia de utilizar aspectos de programación funcional en nuestros proyectos. Por ello se emplean los *Functional Endpoints* en el ejemplo de *Spring Webflux*.

- **Ejemplo 02. *Spring WebClient*.** Este proyecto presenta dos APIs simples, una de animales y otra de personajes de televisión. El ejemplo pretende ilustrar conceptos sobre la manera de realizar una cadena de llamadas / operadores para invocar a un *endpoint HTTP*. Por otra parte, también cuenta con un ejemplo de envío de *Server-Sent Events* o de combinación de múltiples llamadas *HTTP* para ofrecer una única respuesta.

- **Ejemplo 03. *Gateway*.** Este ejemplo difiere algo de los anteriores puesto que no busca la utilización de ninguna herramienta o librería reactiva en concreto. En cambio, su objetivo es mostrar que los patrones reactivos no sólo se circunscriben a una aplicación en particular y que podemos diseñar nuestras plataformas también de manera reactiva. Para ello se presenta una aplicación simple que consta de un *gateway* que recibe peticiones, un servicio de descubrimiento y un numero indefinido de módulos o *workers* que realizan una determinada operación.

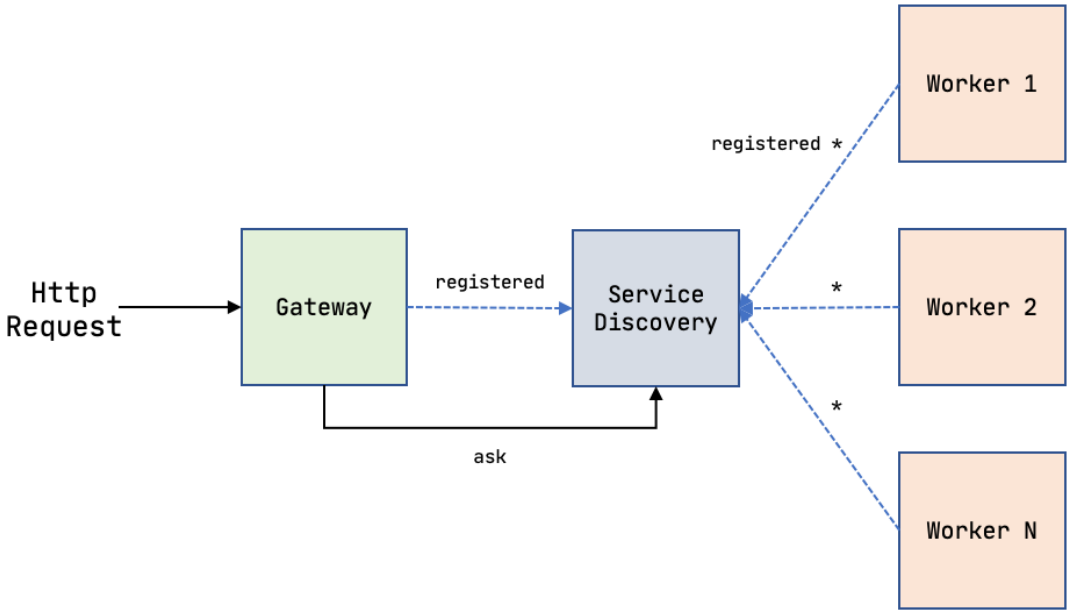

Figura 2 *Diagrama de componentes del ejemplo 03*

- **Ejemplo 04. Acceso a bases de datos relacionales con *R2DBC*.** Contiene una aplicación que pone a disposición del usuario una API de astronautas cuya persistencia reside en una base de datos PostgreSQL. El ejemplo sirve de toma de contacto con la nueva librería de conectividad reactiva con bases de datos relacionales, *R2DBC*. Su objetivo es explorar cuáles son los pasos para su correcta configuración, su utilización básica y las limitaciones que tiene en la actualidad.

- **Ejemplo 05. *RSocket*.** Este ejemplo contiene un pequeño proyecto dividido en dos módulos para probar los diferentes flujos de comunicación que permite el nuevo protocolo binario reactivo *RSocket*. Dichos flujos son, *fire and forget*, donde una determinada petición no espera respuesta, el más común *request-response*, *request-stream*, en el que dada una petición inicial se comunica un flujo de datos o *channel*, que muestra cómo realizar una comunicación completamente bidireccional entre ambos módulos.

- **Ejemplo 06. *Project Loom*.** Busca ser una zona de exploración y trabajo para probar las nuevas APIs que acompañarán a *Java* para la creación de *Virtual Threads*. Este ejemplo es algo particular porque, hoy en día, hay que utilizar versiones de Java 16 de acceso anticipado ya que son características en continuo desarrollo y evolución.

  En este conjunto de pruebas se realiza un simple "Hola Mundo" con *Project Loom*, se hace uso de la nueva API de *Threads* y se exploran los conceptos de concurrencia estructurada y cómo estas nuevas funcionalidades del lenguaje pretenden lidiar con las cancelaciones o *deadlines*.

Conclusiones y próximos pasos

La realización de este proyecto ha sido intensa pero completamente satisfactoria. A continuación, se exponen algunos de los aprendizajes y conclusiones obtenidos durante el transcurso de su preparación.

El coste asociado de la programación reactiva

En todo proceso de adopción de una nueva tecnología uno se ve normalmente deslumbrado por las nuevas características del lenguaje, *framework* o librería que comienza a utilizar.

La **Programación Asíncrona y Reactiva con *Java*** sufre del mismo mal. Hay que tener en consideración que nada de lo que proponen estos paradigmas es estrictamente novedoso, simplemente son reformulaciones de principios que llevan en la comunidad de desarrollo muchos años.

Un punto interesante para considerar es que **debemos ver todos estos recursos como una herramienta y no cómo una razón de ser**. La programación asíncrona viene con unos costes asociados que en determinados casos no nos compensará adquirir. Es indudable que muchos de nuestros casos se uso podrían beneficiarse de estas técnicas, pero no hay que perder de vista que, en la mayoría de las situaciones, **las barreras más pronunciadas residen en la capacitación de los equipos de trabajo** que las implementan y posteriormente las mantienen.

¿Quiere decir esto que la programación reactiva no es útil? En absoluto. Los principios que rigen este paradigma determinan una abstracción mucho más acertada del mundo real. Cuando en la evolución de nuestros sistemas empezamos a requerir características como *timeouts*, reintentos o múltiples integraciones externas debemos empezar a sospechar.

Sin embargo, **la barrera que considero más importante es la de la comunicación**. Es primordial que en una organización se comuniquen correctamente las ventajas y los riesgos que se asumen al cambiar la manera de afrontar los proyectos por una alternativa asíncrona.

Como regla general, si tu aplicación no ha sufrido problemas de escalabilidad el coste de la migración no compensa, pero si, como es cada vez más habitual, tus proyectos comienzan a distribuirse y las dependencias entre sistemas empiezan a ser más acusadas, no hay que dudar en dar una oportunidad a estos planteamientos.

La mejor manera de aprender es enseñar

Durante el proceso de selección de TFM a uno le asaltan muchas dudas sobre la orientación y el desarrollo de su trabajo.

En mi caso particular una de las premisas que manejaba era que **cualquiera que fuese el tema por tratar tenía que concluir con un conocimiento suficiente de las bases** que sustentasen la tecnología utilizada.

La decisión final fue plantear el tema de **Programación Asíncrona y Reactiva con *Java*** como un curso o una formación porque la mejor manera de dominar un tema es afrontarlo con un horizonte de presentación a otras personas.

Una vez completada la tarea no puedo estar más satisfecho del resultado puesto que esa premisa se ha cumplido y estoy impartiendo este material a mis actuales compañeros de trabajo. El *feedback* tras cada sesión y el interés que están mostrando no hacen más que reforzar mis ganas de continuar.

Salir de la zona de confort

Otro de los objetivos personales iniciales antes de comenzar el camino de realización de este proyecto fue el de **utilizar tecnologías y herramientas fuera de mi pila de trabajo habitual**. Estos han sido los resultados:

- Utilización de ***Gradle*** en determinados ejemplos para la gestión de dependencias y construcción de proyectos.
- Emplear versiones de ***Java 14, 15 y 16-ea***.
- Adentrarme en el uso de ***AsciiDoc*** para la documentación de los repositorios como alternativa a *Markdown*. Todo un acierto.
- Utilizar de manera indistinta tanto ***Visual Studio Code*** como ***IntelliJ Idea*** en el transcurso del desarrollo.

Lamentablemente no todas las pretensiones se han cumplido y existen puntos o tareas pendientes que intentaré solventar en un futuro próximo.

Próximos pasos

Pese a que el trabajo de fin de máster tiene unos tiempos marcados, mi intención es que el proyecto continúe y siga siendo actualizado. Estos son los próximos pasos u objetivos que perseguiré:

- **Actualizar paulatinamente los ejemplos** para que dispongan de la última versión de *Spring Boot* y *Java* del momento.
- Incluir ejemplos que cubran otros ecosistemas como ***Micronaut*** o ***Quarkus.***
- Seguir de cerca el desarrollo y evolución de ***Project Loom*** y del resto de proyectos de mejora del lenguaje *Java*.

- Implementar los ejemplos actuales o próximos en **otro lenguaje de la *JVM* como *Kotlin*.**
- Profundizar y **ampliar la sección de pruebas de rendimiento** con ejemplos propios y comparaciones entre ecosistemas y *frameworks*.
- Completar la **versión en inglés de la presentación**.
- Promocionar este paradigma en mi actual empresa y **fomentar el uso de estas tecnologías en futuras ofertas** y proyectos cuando el caso de uso lo aconseje.
- Presentar una versión reducida de alguno de los temas tratados en el TFM como **charla en algún evento de la comunidad** en 2021 o 2022.

Colofón

Este proyecto pone punto final a un viaje iniciado en otoño de 2019. **Mi más sincero agradecimiento** a los alumnos y profesores del Máster en Cloud Apps por todos estos meses de aprendizaje. Sólo puedo decir que **ha sido un completo acierto**.

Referencias

Durante la realización del TFM se han consultado numerosas referencias de todo tipo. La mayoría de ellas se encuentran en el repositorio de código, en muchos casos integradas en cada tipo de recurso: enlaces, ejemplos y presentación.

No obstante, se incluye a continuación la lista de referencias básicas que contiene la presentación que guía este TFM:

Libros

- Reactive Spring, Josh Long
- The Pragmatic Programmer (20th Anniversary Edition), Andrew Hunt & David Thomas
- Effective Java (3rd Edition), Joshua Bloch
- Optimizing Java: Practical Techniques for Improving JVM Application Performance, Benjamin J. Evans & James Gough
- Java Performance (2nd Edition), Scott Oaks

Videos

- How great leaders inspire action, Simon Sinek
- Java Concurrency and MultiThreading (playlist), Jakob Jenkov
- Efficient Java Multithreading and Concurrency with Executors (course), Arun Kumar
- Do's and Don'ts: Avoiding First-Time Reactive Programmer Mines, Sergei Egorov
- Avoiding Reactor Meltdown, Phil Clay
- Benefits of reactive programming with Reactor and Spring Boot 2, Violetta Georgieva

Artículos, noticias y otros enlaces

- Reactive Foundation
- The Reactive Manifesto
- The Reactive Principles
- ¿Qué es el Golden Circle y por qué debes tenerlo en cuenta?, Luis Font
- 5 of the world's biggest network outages, TechRadar
- Snapchat spending $2 billion over 5 years for Google Cloud, ZDNet
- TikTok Agreed to Buy More Than $800 Million in Cloud Services From Google, The Information
- Famous DDoS attacks | The largest DDoS attacks of all time, Cloudflare
- Java 11 Flow
- Project Reactor
- Eclipse VertX
- Akka Streams
- RxJava

- The State of Reactive
- JSR-000133 Java Memory Model and Thread Specification Revision
- JSR 315: Java Servlet 3.0 Specification
- JSR 340: Java Servlet 3.1 Specification
- Life Cycle of a Thread in Java, Baeldung
- Visual VM
- Java Mission Control
- Shakespeare plays Scrabble with some libraries, @akarnokd
- Spring 5 WebClient, Baeldung
- Spring WebClient vs. RestTemplate, Baeldung
- R2DBC
- Spring Data R2DBC – Reference
- R2DBC – Reactive Relational Database Connectivity, Baeldung
- A Quick Look at R2DBC with Spring Data, Baeldung
- RSocket
- Spring Docs – Rsocket
- Introduction to RSocket, Baeldung
- Project Loom and structured concurrency, JVM Advent

Repositorios

- Async and reactive programmig in Java, marcosDLCS
- A List of Post-mortems!, danluu
- Reactive Spring, codeurjc
- Reactive Streams JVM, rective-streams
- Spring R2DBC sample, hantsy
- Rsocket Java, rsocket
- Loom, openjdk