uc3m | Universidad Carlos III de Madrid

Grado Ingeniería de Sistemas Audiovisuales

2018-2019

*Trabajo Fin de Grado*

# “Diseño e implementación de un microservicio con Spring”

---

Jesús Rienda Iáñez

Tutor/es

Carmen Pelaez Moreno

Leganés, 2019


*[Incluir en el caso del interés en su publicación en el archivo abierto]*

Esta obra se encuentra sujeta a la licencia Creative Commons **Reconocimiento - No Comercial - Sin Obra Derivada**

# RESUMEN

En este trabajo fin de grado se ha realizado un estudio sobre las posibilidades existentes en el desarrollo de aplicaciones web, desde el diseño partiendo de cero hasta el uso de software de terceros como son los frameworks que facilitan el desarrollo y se centran en la funcionalidad.

Para el correcto diseño de nuestra arquitectura ha sido necesario descomponer e investigar cada integrante de esta. Por una parte se ha investigado sobre los patrones de diseño existentes y sus funcionalidades, también sobre tipos de arquitecturas, framework y además sobre las bases de datos.

Para afianzar todo lo desarrollado en este trabajo se ha implementado un microservicio basado en listas de reproducción de Spotify, utilizando para ello las ultimas tecnologías de desarrollo como Spring Boot, Spring Data y una base de datos muy potente como MongoDB. Este microservicio hace de intermediario con la base de datos filtrando, por ejemplo, las listas de reproducción que contienen canciones que ha escuchado un usuario.

**Palabras clave:** Microservicio, aplicaciones web, SOA, API, Swagger, MongoDb, Lombok, Spring Boot

# DEDICATORIA

# ÍNDICE GENERAL

# ÍNDICE DE FIGURAS

# ÍNDICE DE TABLAS

# 1. RESUMEN

## 1.1. Introducción

El mundo de la tecnología se basa en tendencias, hoy en día los microservicios están de moda, gracias a grandes compañías como Amazon, Ebay, Twitter, Paypal, Netflix, etc. Pero realmente es una tecnología que ha surgido gracias a la aparición de la nube(cloud) donde todas estas empresas se están llevando sus software

En la nube no necesitas servidores físicos, con el mantenimiento y la inversión que esto supone, sino que solo pagas por los recursos que necesitas en cada momento. Esto es una gran revolución para la informática, te permite gestionar los picos de actividad sin perder servicio.

### 1.1.1. Planteamiento del problema

Contamos con datos reales de Spotify que contienen información de las listas de reproducción. Tenemos también datos de usuarios con sus canciones.

Necesitamos una base de datos donde almacenar todos los parámetros y vamos a crear un programa para realizar las consultas.

Este programa contará con tiempos de respuesta bajos ya que podrán consultarlo muchas personas a la vez.

### 1.1.2. Objetivos

Diseñar una arquitectura de software adecuada para el problema.

Crear una base datos con los ficheros de listas.

Implementar un microservicio.

### 1.1.3. Solución propuesta

Como solución a nuestro problema vamos a diseñar una arquitectura de microservicios. Implementamos un microservicio con java mediante el framework Spring Boot y para llamar a este utilizaremos Swagger. La BBDD estará almacenada en Azure y sera CosmosDB

## 1.2. Estado del arte

El Instituto de Ingeniera de Software la define la arquitectura de software como: -epresentación del sistema que ayuda a comprender cómo se comportará un programa."

La arquitectura es la principal portadora de cualidades del sistema, como el rendimiento, la modificabilidad y la seguridad. El sistema se divide en elementos de software llamados módulos. Las propiedades de estos elementos pueden ser de dos tipos, internas y externas.

Las **propiedades internas** son las que definen el lenguaje en el que está desarrollado y todos los detalles de la implementación, como pueden ser objetos, clases, modulos, hilos.

Las **propiedades externas** son los contratos que existen entre módulos y que permiten a otros módulos establecer dependencias/conexiones entre ellos.

Existen patrones de diseño, arquitectónicos y frameworks que nos proponen soluciones a nuestro problema.

### 1.2.1. Patrones de diseño

Los patrones de diseño no aparecieron hasta 1994 en un libro llamado "Patrones de diseño"que a día de hoy sigue siendo un modelo de referencia. Los definidos en el libro se dividen en función del problema a resolver, tenemos:

- Patrones creacionales: utilizados para facilitar la creación de nuevos objetos. Los más usados son Singleton y Builder
- Patrones estructurales: facilitan la modelización de nuestro software, definiendo la forma en que las clases se relacionan entre sí. Los más conocidos son Facade, Decorator y Proxy.
- Patrones de comportamiento: gestionan algoritmos, relaciones y responsabilidades entre objetos. El más conocido es Strategy

### 1.2.2. Patrones de arquitectura

Los patrones de arquitectura definen las cualidades del sistema, éstos al igual que los patrones de diseño, solucionan problemas recurrentes de una forma reutilizable.

Los patrones de arquitectura más usados son:

- Programación por capas: Estructura programas que pueden descomponerse en subtareas.

- Cliente-servidor: Cuenta con un servidor y múltiples clientes, el servidor será el que de servicio a diversos componentes del cliente y los clientes solicitarán servicio al servidor.
- Arquitectura orientada a servicios: Da soporte mediante la creación de servicios.
- Arquitectura de microservicios: Crea aplicaciones usando un conjunto de pequeños servicios que se comunican entre sí pero se ejecutan de forma individual.
- Modelo Vista Controlador: Divide un programa interactivo en tres partes: El modelo formado por los datos básicos y la funcionalidad del programa, la vista que maneja la visualización de la información y el controlador que gestiona la entrada e informa al modelo y la vista de los cambios.

### 1.2.3. Frameworks

Los framework son estructuras de software ya implementadas en las que un programador puede apoyarse para desarrollar un proyecto propio, están implementados y siguen tanto los patrones de sotfware como los patrones de arquitectura; suelen incluir ficheros de configuración, librerías, etc.

Existen frameworks para prácticamente todos los lenguajes de programación pero es importante valorar y elegir el framework que mejor se adapta al problema que vas a resolver, por ejemplo serían Angular orientado al desarrollo del front-end o Spring orientado al desarrollo de back-end.

Para el desarrollo de aplicaciones en java con microservicios tenemos estos frameworks: Spring Boot que ofrece un arquetipo de un microservicio básico y es ampliable fácilmente con poco desarrollo, Dropwizard que es un framework para arquitectura de microservicios por capas, Eclipse MicroProfile ofrece un arquetipo de la aplicación con las dependencias necesarias y Helidon que Oracle utilizó internamente bajo el nombre "Java for Cloudz finalmente se ha liberado para uso público.

### 1.2.4. Bases de datos

Las bases de datos son sistemas formados por grupos de datos que permiten la consulta, modificación y borrado de los mismos. Se crearon para poder almacenar grandes cantidades de información.

Las primeras en surgir fueron las relacionales, en las que los datos están relacionados entre sí. Más tarde surgieron las bases de datos no relacionales(NoSQL) que son las que no contienen un identificador para relacionar un conjunto de datos con otro.

Las ventajas de las bases de datos relacionales son: Tienen los estándares bien definidos, son sencillas a la hora de programar y existe mucha información sobre ellas. Por otra

parte los inconvenientes de éstas son: Crecen demasiado por lo que aumenta el tiempo de respuesta y para realizar un cambio normalmente habrá que reestructurarlas.

En cuanto a las ventajas de las bases de datos no relacionales podemos encontrar: Se adaptan en crecimientos o cambios sin ningún problema, no es necesario contar con servidores grandes. Los inconvenientes en este caso son: No esta garantizado que si la operación falla se complete la información, se necesitan conocimientos elevados en el uso de la herramienta, no tienen un lenguaje estándar definido y todo has de hacerlo por consola ya que no disponen de interfaz gráfica.

### 1.2.5. Microservicios

Son útiles para desarrollar una aplicación, dividiéndola en una serie de pequeños servicios, que se ejecutan de forma independiente y se comunican entre sí. Cada microservicio es pequeño y pertenece a un área de la aplicación.

Las ventajas de éstos son: Ofrecen la libertad de implementar y desplegar los servicios de forma independiente, se pueden desarrollar con un equipo de trabajo mínimo, ofrecen la posibilidad de utilizar lenguajes de programación diferentes y son fácilmente escalables. Éstos también tienen inconvenientes: Las pruebas son complicadas por el despliegue distribuido, tienen un alto consumo de memoria, cuando tienes un gran número de servicios es complicado integrarlos y gestionarlos.

## 1.3. Justificación de la solución

Elegimos arquitectura de microservicios ya que ofrecen escalabilidad y alta disponibilidad.

Para el desarrollo elegimos Spring Boot que cuenta con las ventajas de Sprint y es el framework que mejor se adapta a nuestro proyecto ya que es el más antiguo y de mayor uso.

Para el desarrollo elegimos Spring Boot, no solo nos aporta ventajas de cualquier framwork de microservicios sino que también tenemos todas la funcionalidades de Spring como la conexión con base de datos y la arquitectura en capas.

Para la gestión de dependencias elegimos Maven ya que esta dirigido para java y nos va a empaquetar la aplicación.

Utilizamos Lombok para evitar código repetitivo. Ademas ofrece una implementación del patrón de diseño builder.

La comunicación con el microservicio va a ser mediante REST, protocolo simple y eficaz para realizar las distintas operaciones.

Todo servicio REST necesita un documento(API) donde se describa la entrada salida

de él, en este caso vamos a utilizar anotaciones de

Utilizamos Swagger para autogenerar el documento API en el que se definen las entradas y salidas de todos los servicios desarrollados.

Para almacenar los datos con los que contamos la mejor opción es MongoDB, con esta no es necesario adaptar los datos ya que se guardan con el formato actual. Para que sea más eficiente pondremos nuestra BBDD en la nube y utilizaremos CosmosDB.

### 1.4. Desarrollo de la aplicación

El primer es tener instalado Eclipse, de aquí vamos a la web https://start.spring.io/ usando el framework Spring Boot. Elegimos Maven para el control de dependencias, escogemos java como lenguaje y por ultimo las dependencias que queremos incluir, MongoDB, Spring Web, Lombok.

Swagger es un software para diseñar, documentar y consumir servicios Rest.Para configurar Swagger es necesario añadir una clase de configuración con la anotacción @EnableSwagger2 para habilitar la generación del dominio.

Creamos un repositorio en github (https://github.com/jesusRienda/microservicio) para poder guardar lo que tenemos hasta ahora.

El arquetipo contiene una clase App que hay que ejecutar para levantar el microservicio en local, para hacerlo al tener una dependencia de MongoDb tenemos que establecer la conexión con la BBDD.

Crear en azure una instancia de CosmosDb y configurar el microservicio para que apunte a esa base de datos. Creamos una clase java llamada MongoDbConfig y mediante la anotación @Value obtenemos de application.properties la conexión a cosmos y el nombre de la BBDD. Una vez hecho esto ya podemos levantar el microservicio.

Lo siguiente es crear las tres capas que vamos a utilizar, la capa de persistence, donde se definen las querys, la capa de service donde se implementa la lógica de negocio y la capa controller donde se implementan los verbos http mediante los que se va a llamar al servicio.

Una vez tenemos los métodos necesarios en la capa de persistencia pasamos a la capa de servicio en la que vamos a crear una interfaz con los métodos que vamos a exponer a la capa de presentación y una implementación de esa interfaz.

En esta capa se llaman a los métodos necesarios de la capa anterior y además puedes llamar a métodos de un servicio diferente.

La capa de presentación es la encargada de serializar un json a un objeto y en función del recurso, llamar a un service u otro. En ella se definen los recursos y los verbos de la petición.

## 1.5. Marco regulador

No existen leyes que se apliquen en el desarrollo de un microservicio. Sin embargo si se trata con datos reales hay que tener en cuenta dos leyes:

- Ley Orgánica de Protección de Datos Personales y garantía de los derechos digitales.
- Ley General de Telecomunicaciones.

## 1.6. Entorno socio-económico

### 1.6.1. Presupuesto

El proyecto se ha desarrollado durante 8 meses con un total de 300 horas empleadas lo que nos supone un total de 4500 euros. Los servicios contratados son Azure CosmosDB que nos supone un total de 1069,34 euros. En los costes totales del proyecto sumamos la mano de obra y los servicios contratados y nos saldría un total de 5569,34 euros.

### 1.6.2. Impacto socio-económico

Uno de los objetivos de desarrollar aplicaciones con microservicios es que estas sean fácilmente mantenibles y que puedan evolucionar según las necesidades de la sociedad.

Una de las ventajas de implantar los microservicios en la nube es la mejor utilización de los recursos disponibles para aprovecharlos de manera eficiente abasteciéndose con tecnologías renovables.

Cuando se esperaba un numero grande de peticiones era necesario un gran número de servidores que suponía recursos sin utilizar una vez dado el servicio, esto lo solucionan los microservicios en la nube.

Un inconveniente de esto es para una aplicación pequeña el coste es más elevado.

## 1.7. Conclusiones

Hemos cumplido el objetivo de diseñar una arquitectura capaz de recibir multitud de peticiones.

Para la base de datos elegimos una en la nube, aunque esta es la mejor opción, nos hemos dado cuenta que es muy caro para una pequeña aplicación ya que cobran por uso y almacenamiento. Sin embargo si que hemos podido ver que para el problema que teníamos, la base de datos elegida es ideal.

Implementar el microservicio ha sido una tarea de investigación ya que todas las tecnologías utilizadas son bastante novedosas y es complicado encontrar información al respecto. Pero gracias a ello hemos conseguido una aplicación limpia y fácilmente mantenible.

### 1.7.1. Trabajos futuros

Posibles trabajos futuros que podrían mejorar o complementar nuestra aplicación.

-Desarrollar una aplicación web para visualizar los datos.

-Añadir nuevas funcionalidades sobre los datos existentes.

-Dockerizar la aplicación para integrarla en un contenedor y desplegarlo en la nube.

-Incluir un gestor de contenedores en Azure para gestionar el escalado y otras configuraciones de las aplicaciones que tengamos dockerizadas.

# BIBLIOGRAFÍA


[1] S. E. Institute, "Software Architecture", 2017. [En línea]. Disponible en: https://www.sei.cmu.edu/research-capabilities/all-work/display.cfm?customel_datapageid_4050=21328.

[2] C. Alexander, S. Ishikawa y M. Silverstein, *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, 1977.

[3] E. Gamma, *Patrones de diseño: elementos de software orientado a objetos reutilizable*, ép. Addison-Wesley professional computing series. Pearson Educación, 2002. [En línea]. Disponible en: https://books.google.es/books?id=gap_AAAACAAJ.

[4] "APLICACIONES HÍBRIDAS: ¿QUÉ SON Y CÓMO USARLAS?", [En línea]. Disponible en: https://www.nextu.com/blog/aplicaciones-hibridas-que-son-y-como-usarlas/.

[5] G. Peiretti, "Strategy Pattern con Spring Boot", 2017. [En línea]. Disponible en: https://experto.dev/strategy-pattern-spring-boot/.

[6] E. F. Codd, "A Relational Model of Data for Large Shared Data Banks", *Commun. ACM*, vol. 26, n.º 1, pp. 64-69, ene. de 1983. DOI: 10.1145/357980.358007. [En línea]. Disponible en: http://doi.acm.org/10.1145/357980.358007.

[7] S. Newman, *Building Microservices: Designing Fine-Grained Systems*, 1st. O'Reilly Media, 2015, p. 280.

[8] L. Mauersberger, "A brief history of microservices", *blog.leanix.net*, 2017. [En línea]. Disponible en: https://blog.leanix.net/en/a-brief-history-of-microservices.

[9] N. Dragoni et al., "Microservices: Yesterday, Today, and Tomorrow", en *Present and Ulterior Software Engineering*, M. Mazzara y B. Meyer, eds. Cham: Springer International Publishing, 2017, pp. 195-216. DOI: 10.1007/978-3-319-67425-4_12. [En línea]. Disponible en: https://doi.org/10.1007/978-3-319-67425-4_12.

[10] R. RV, *Spring Microservices*. Packt Publishing, 2016. [En línea]. Disponible en: https://books.google.es/books?id=pwNwDQAAQBAJ.

[11] J. del Estado, "Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales.", 2018. [En línea]. Disponible en: https://www.boe.es/eli/es/lo/2018/12/05/3.

[12] ——, "Ley 9/2014, de 9 de mayo, General de Telecomunicaciones.", 2014. [En línea]. Disponible en: https://www.boe.es/eli/es/l/2014/05/09/9.