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

RESUMEN

En este trabajo fin de grado se ha realizado un estudio sobre las posibilidades existentes en el desarrollo de aplicaciones web. Desde el desarrollo partiendo de cero hasta el uso de software de terceros como son los framework para facilitar el desarrollo y centrarse 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 ademas 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

1. RESUMEN. 1
1.1. Introducción 1
1.1.1. Planteamiento del problema 1
1.1.2. Objetivos 1
1.1.3. Solución propuesta 1
1.2. Estado del arte 2
1.3. Justificación de la solución 2
1.4. Desarrollo de la aplicación 2
1.5. Marco regulador 2
1.6. Entorno socio-económico 3
1.6.1. Presupuesto 3
1.6.2. Impacto socio-económico 3
1.7. Conclusiones 3
1.7.1. Trabajos futuros 3
1.8. Estado del arte 4
1.8.1. Patrones de diseño 4
1.8.2. Patrones de arquitectura 5
1.9. Frameworks 6
1.9.1. Frameworks de java para microservicios 6
1.10. Bases de datos 7
1.10.1. Pros y contras bases de datos relacionales 7
1.10.2. Pros y contras bases de datos no relacionales 7
1.11. Microservicios 8
1.11.1. Pros y contras de los microservicios 8
1.11.2. Ejemplos de empresas que utilizan microservicios 9
1.12. Comparación de SOA con Microservicios 9
1.12.1. Ejemplo de visión monolítica vs visión de microservicio 12
2. JUSTIFICACIÓN DE LA SOLUCIÓN 14

3. DESARROLLO DE LA APLICACIÓN 15
4. MARCO REGULADOR 19
5. ENTORNO SOCIO-ECONÓMICO 20
5.1. Presupuesto 20
5.2. Impacto socio-económico 20
6. CONCLUSIONES 21
6.1. Trabajos futuros 21
BIBLIOGRAFíA 22

ÍNDICE DE FIGURAS

1.1 Arquitectura Monolítica frente Microservicios 10
1.2 Eficiencia microservicios frente SOA 11
1.3 Ejemplo de arquitectura monolitica(SOA) 12
1.4 Ejemplo arquitectura microservicios 13

3.1 Dependencias Swagger necesarias 15
3.2 Clase configuración Swagger 16
3.3 Clase de configuración MongoDB 16
3.4 Estructura microservicio 17

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

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

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 (contenerizar) 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.

1.8. Estado del arte

En la década de los 60 surgió la arquitectura de software, pero hasta 1980 no se integro totalmente en el desarrollo de software.

El Instituto de Ingeniera de Software la define como: "La arquitectura de software es una representación 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 y no se puede lograr sin una visión arquitectónica unificadora.

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:

- Entidades dinámicas en tiempo de ejecución como objetos e hilos.

- Entidades lógicas en tiempo de desarrollo como clases y módulos.

- Entidades físicas como nodos o carpetas.

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.

1.8.1. Patrones de diseño

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

- 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 si. Los más conocidos son Facade, Decorator y Proxy.

- Patrones de comportamiento: gestionan algoritmos, relaciones y responsabilidades entre objetos. El más conocido es Strategy

1.8.2. Patrones de arquitectura

Estos tienen un alto nivel de abstraccion. Estos al igual que los patrones de diseño, solucionan problemas recurrentes de una forma reutilizable.

Los patrones de arquitectura mas usados son:

- Programación por capas: Estructurar 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: Crear aplicaciones usando un conjunto de pequeños servicios que se comunican entre sí pero se ejecutan de forma individual.
- Pipeline: Oorganizar sistemas que procesan una sucesión de datos que pasan a través de tuberías.
- Arquitectura en pizarra: Para la resolución de problemas que se desconoce su estrategia. Está formado por tres componentes:
  - **Pizarra:** memoria que contiene todos los objetos.
  - **Fuente de conocimiento:** módulos especializados.
  - **Componente de control:** selecciona y ejecuta los módulos.
- Arquitectura dirigida por eventos: Maneja principalmente los eventos y está formado por cuatro componentes que son: fuente de evento , escucha de evento , canal y bus de evento.
- Peer-to-peer: Se llama pares a las componentes individuales y estos pueden funcionar como servidor, dando servicio a otros pares, o como cliente, pidiendo servicio a otros pares.
- Modelo Vista Controlador: Divide un programa interactivo en tres partes:
  - **Modelo:** formado por los datos básicos y la funcionalidad del programa.
  - **Vista:** maneja la visualización de la información.
  - **Controlador:** controla la entrada (teclado y ratón) e informa al modelo y la vista de los cambios.

1.9. Frameworks

Estos son estructuras de software ya implementadas en las que un programador puede apoyarse para desarrollar un proyecto propio. Los frameworks están implementados y siguen tanto los patrones de sotfware como los patrones de arquitectura y 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 Angular orientado al desarrollo del front-end o Spring orientado al desarrollo de back-end.

1.9.1. Frameworks de java para microservicios

Encontramos los siguientes orientados a microservicios.

- Spring Boot: ofrece un arquetipo de un microservicio básico, ampliable fácilmente con poco desarrollo.
- Cricket: framework para implementación de arquitectura dirigida por eventos.
- Dropwizard: framework para arquitectura de microservicios por capas.
- Eclipse MicroProfile: arquetipo de la aplicación con las dependencias necesarias.
- Helidon: Oracle lo utilizó internamente bajo el nombre "Java for Cloudz finalmente se ha liberado para uso público.

1.10. Bases de datos

Para el almacenamiento de los datos que vamos a utilizar bases de datos que se pueden definir como un conjunto de información relacionada que se encuentra agrupada ó estructurada. 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.

Actualmente dominan el mercado de las bases de datos IBM, Microsoft y Oracle y en el campo de internet Google.

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. La más exitosa en bases de datos no relacionales es MongoDB seguida por Redis, Elasticsearch y Cassandra.

1.10.1. Pros y contras bases de datos relacionales

Ventajas:

- Gran variedad de información para poder realizar cualquier consulta.
- La información no se completa si a mitad de una operación ocurre un problema.
- Tiene los estándares bien definidos.
- Es sencillo a la hora de programar.

Inconvenientes:

- Crecen demasiado y aumenta el tiempo de respuesta.
- Se tendrá que reestructurar si se realiza cualquier cambio.
- No garantizan el buen funcionamiento si el sistema operativo no cumple los requerimientos mínimos.

1.10.2. Pros y contras bases de datos no relacionales

Ventajas:

- Adaptabilidad para crecimientos o cambios.
- Crecimiento horizontal.
- No es necesario contar con servidores de gran cantidad.
- Cuenta con un algoritmo interno que reformule las consultas para no sobrecargar los servidores.

Inconvenientes:

- No garantiza que si la operación falla se complete la información.
- Se necesitan conocimientos elevados en el uso de esta herramienta pues las operaciones son limitadas.
- No tiene un estándar de lenguaje definido.
- No contiene una interfaz gráfica por lo que es necesario hacer todo mediante consola.

1.11. 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.

Tiene que haber un número mínimo de servicios encargados de gestionar los procesos en común. Cada microservicio es pequeño e independiente y se corresponde con un área de la aplicación.

1.11.1. Pros y contras de los microservicios

Ventajas:

- Libertad de implementar y desplegar los servicios de forma independiente.
- Los microservicios se pueden desarrollar con un equipo de trabajo mínimo.
- Posibilidad de utilizar lenguajes diferentes.
- Con el uso de contenedores el desarrollo y despliegue son más rápidos.
- Fácilmente escalables.

Inconvenientes:

- Pruebas complicados por el despliegue distribuido.
- Puede dar lugar a grandes bloques de información para gestionar.
- Con un gran número de servicios, integrarlos y gestionarlos puede resultar complicado.
- Alto consumo de memoria.
- Fragmentar una aplicación en microservicios puede llevar muchas horas de planificación.

1.11.2. Ejemplos de empresas que utilizan microservicios

Encontramos estas compañías Netflix, Amazon y Ebay.

2. 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.

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.