tennislab: mongodb reactivo

Mongo Atlas – KtorFit - Caché

IES Luis Vives

AD – Mario Resa y Sebastián Mendoza

Contenido

[Introducción 1](#_Toc125996934)

[Diseño 2](#_Toc125996935)

[Lenguaje y tecnologías 2](#_Toc125996936)

# Introducción

La forma en la que accedemos a la información ha ido cambiando a lo largo de los años, tanto que es posible ver que, de un año para otro, la forma en la que lo hacemos va difiriendo según las necesidades volátiles que pide el mundo. Es por ello que el ideal de un buen programador es estar en un continuo aprendizaje, una constante actualización, para estar siempre a la vanguardia de la tecnología y la información. Este hecho nos trae precisamente aquí, a este proyecto. Vistas las tecnologías y el diseño de BBDD (Bases de Datos) enfocadas a la estructura relacional, este proyecto trae a primera fila las BBDD NoSQL, siglas de ***Not Only SQL***, es decir no solo SQL (no relacionales).

Las BBDD no relacionales se caracterizan en que no se utilizan estructuras tan sencillas como las tablas, ni se almacena sus datos en forma de registros o campos. Esto nos indica que hay cierta flexibilidad a la hora de almacenar información y que es muy fácil adaptarse a las necesidades de cualquier proyecto a desarrollar.

A parte de acceder a la información en una BBDD de forma local, tenemos la opción de hacerlo de forma remota, las API. Una API, o *interfaz de programación de aplicaciones*, es un conjunto de reglas que definen cómo pueden las aplicaciones o los dispositivos conectarse y comunicarse entre sí. Una API REST es una API que cumple los principios de diseño del estilo de arquitectura REST o *transferencia de estado representacional*. Por este motivo, las API REST a veces se conocen como API RESTful*.*[[1]](#footnote-1)

Y siguiendo en la línea del tipo de acceso a la información, también nos encontramos con las cachés, información guardada de forma local y temporal de cierta cantidad de información.

La unión de estas tres tecnologías da origen a este proyecto, una pequeña ampliación al anterior y que nos enseña que siempre hay diferentes formas de resolver un problema.

# Diseño

## Lenguaje y tecnologías

Siguiendo la línea de los anteriores proyectos, en este también se ha optado por **Kotlin** como lenguaje de programación. A la hora de usar una BBDD se ha utilizado MongoDB reactivo, más específicamente **Mongo Atlas**. La decisión de esto es para evitar de primeras los contenedores en Docker, además de ser una alternativa bastante factible y diferente a la que no estamos acostumbrados de ver.

Como el proyecto necesita de acceso a una API Rest, se ha optado por utilizar **KtorFit**: es un cliente HTTP / procesador de símbolos Kotlin para Kotlin multiplataforma (Js, Jvm, Android, iOS, Linux) que utiliza clientes KSP y Ktor inspirados en Retrofit[[2]](#footnote-2).

Para terminar, y no menos importante, la caché. En esto caso se ha utilizado **Cache4K**: cache4k proporciona una caché de clave-valor en memoria simple para Kotlin Multiplatform, con soporte para desalojos basados en el tiempo (expiración) y basados en el tamaño.[[3]](#footnote-3)

Mongo Atlas se utilizará para guardar todo el contenido de la aplicación, KtorFit se usará para acceder a la API Rest dada y Cache4K para tener los datos especificados guardados en memoria de forma temporal.

## Diagrama

1. API REST - https://www.ibm.com/es-es/cloud/learn/rest-apis [↑](#footnote-ref-1)
2. KtorFit - https://foso.github.io/Ktorfit/ [↑](#footnote-ref-2)
3. Cache4K - https://reactivecircus.github.io/cache4k/ [↑](#footnote-ref-3)