tennislab: mongodb reactivo

Mongo Atlas – KtorFit - Caché

IES Luis Vives

AD – Mario Resa y Sebastián Mendoza

Contenido

[Introducción 1](#_Toc126071490)

[Diseño 2](#_Toc126071491)

[Lenguaje y tecnologías 2](#_Toc126071492)

[Diagrama 2](#_Toc126071493)

# 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

Antes de empezar a picar código, una de las partes más importantes del desarrollo de un proyecto es la creación del diagrama de clases. Este diagrama define las relaciones que van a tener las clases entre sí dando un panorama de cómo va a funcionar el proyecto. Hay diversos diagramas que pueden explicar un proyecto, pero en nuestro caso se ha optado por el de clases ya que nos facilita enseñar las propiedades que tienen cada clase, además de poder ver la cardinalidad entre las mismas.

Diagrama, Dibujo de ingeniería

Descripción generada automáticamente

Ilustración Diagrama de clases

### Usuario, pedido, turno y raqueta

Una de las relaciones más importante que tiene este proyecto es la que tiene *Usuario* con *pedido, turno* y *raqueta*. Vamos por partes:

1. Usuario – Pedido.

Diagrama

Descripción generada automáticamente

La relación entre estas dos clases de uno a uno, es decir, un usuario tiene un pedido, y un pedido pertenece a un usuario. La elección de esta relación se ha determinado que sea unidireccional, es decir, una de las clases estará en la otra, pero no viceversa. En este caso, se ha optado a que *Pedido* tenga un *Usuario*, pero que no suceda en el otro sentido. El motivo de esta elección es que nos parecía mas sencillo consultar el pedido de un usuario con la identificación de *Usuario* que tuviera el pedido. Otro motivo ha sido la recursividad, ya que si el usuario tuviera una lista de pedidos, los pedidos también tendrían un usuario y así hasta el infinito. Lo más seguro es que haya una solución más factible en la que funcione la bidireccionalidad, pero nosotros nos decantamos por esta opción por encima de las demás.

1. Usuario – Turno.

Diagrama

Descripción generada automáticamente

La relación entre *Usuario* y *Turno* se da en los casos en la que el usuario es un trabajador. Como se puede ver en la imagen, esta también es una relación unidireccional de uno a muchos: un usuario tiene un turno de trabajo, y un turno es asignado a uno o varios usuarios. Al igual que en el caso anterior, *Turno* tendrá la información de *Usuario* para evitar recursividad, además de que nos parecía más cómodo consultar los turnos de los usuarios desde el propio turno, evitando también que los perfiles de cliente aparecieran un *Turno* en nulo cuando se consultara por la información del *Usuario*.

1. Usuario – Raqueta.

Diagrama

Descripción generada automáticamente

En el caso de la relación entre *Usuario* y *Raqueta* es más sencilla que los anteriores. Aquí hemos optado a embeber *Raqueta* en *Usuario*, ya que no necesariamente debe haber una colección con la información de las raquetas de los clientes. Es por ello que

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)