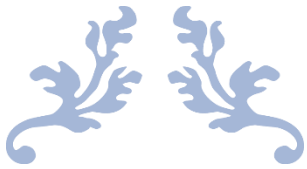


AD-P03-TennislabNoSQL

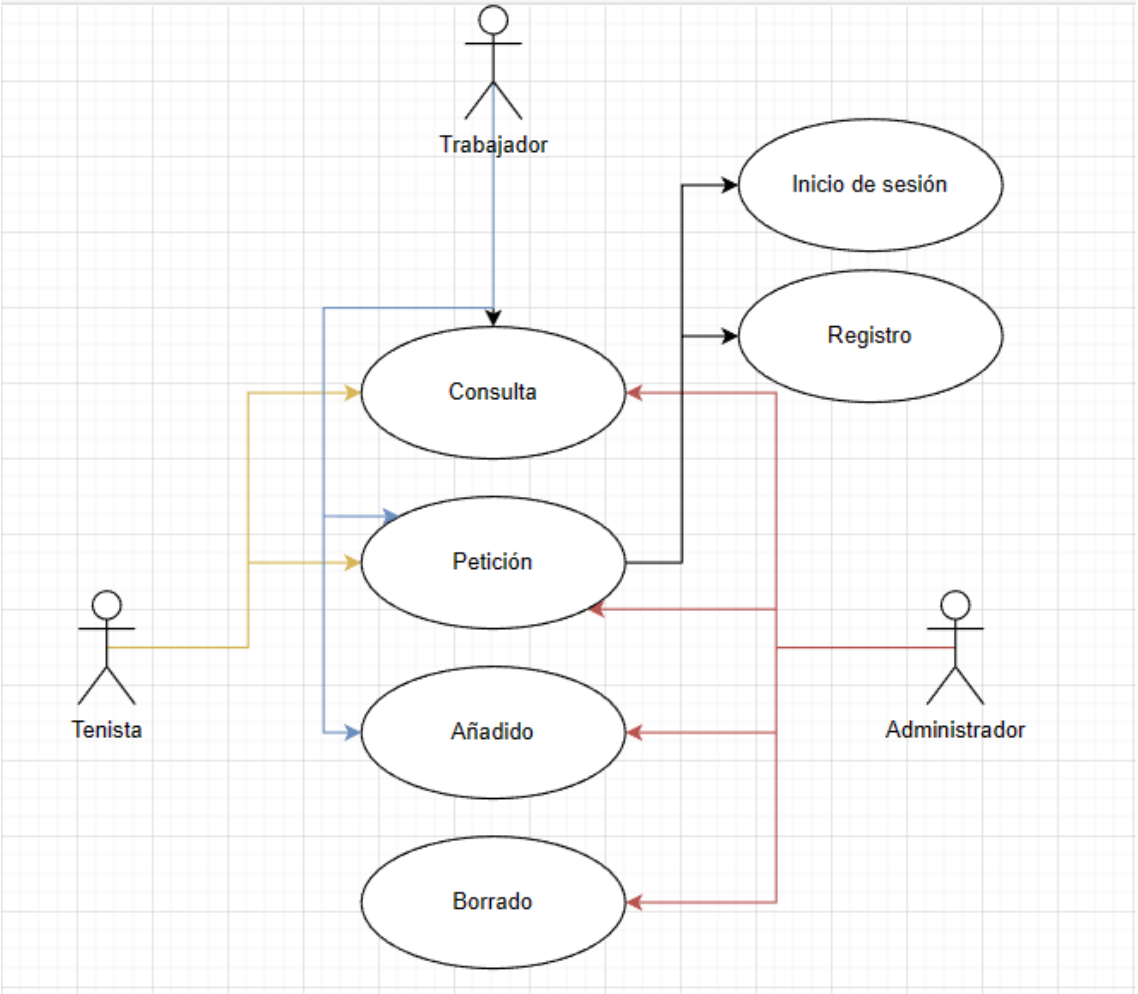
Documentación del proyecto

5 DE FEBRERO DE 2023

IVÁN AZAGRA Y DANIEL RODRÍGUEZ
IES Luis Vives

Contenido

Diagrama de uso:

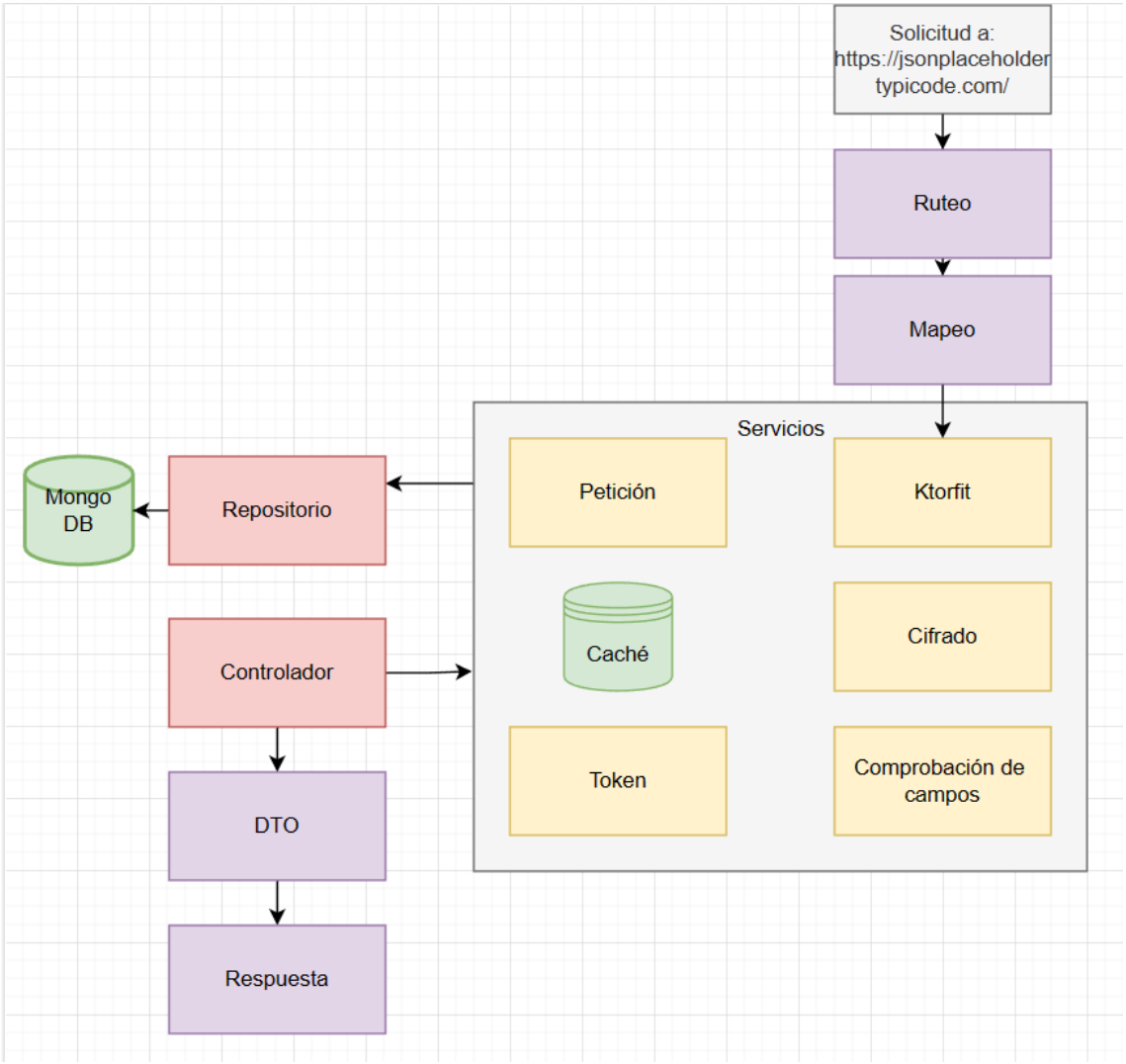


El tenista puede realizar consultas y peticiones para realizar el inicio de sesión o el registro en caso de no tener cuenta en nuestro sistema.

El trabajador puede realizar consultas, peticiones de inicio de sesión o registro y de añadido limitado únicamente a los turnos.

El administrador puede realizar consultas, peticiones de inicio de sesión o registro, añadido de cualquier entidad del sistema y borrado de las mismas, es el único con la capacidad de hacer esta última acción.

Arquitectura:



Comienza con el endpoint de consulta que es ruteado con ktorfit y mapeado a las diferentes entidades que utilizamos en el programa, el repositorio recoge los datos del endpoint consultado y los guarda en la base de datos de mongo y se utiliza para realizar diferentes consultas a esta base de datos, en nuestros servicios tenemos una caché que guardará los datos consultados para agilizar el proceso dentro del lado del cliente, esta misma caché se actualiza cada minuto para tener los datos más recientes.

El controlador utiliza estos servicios para realizar restricciones y comprobaciones, para ello utilizamos los tokens para limitar las funciones dependiendo del tipo de perfil que las quiera realizar, utiliza también un sistema de cifrado para el guardado seguro de las contraseñas, el controlador hace uso de los objetos de transferencia de datos (DTO) para generar las respuestas con los datos.

Requisitos funcionales:

Para los requisitos funcionales empezaré definiendo su significado, un requisito funcional es aquel que define el comportamiento interno del software, ya sean detalles técnicos, cómo manipula la información...

Entre los requisitos funcionales de este programa se encuentran los siguientes:

- El programa posee un sistema de autenticación que funciona con tokens.
- El programa cuenta con un catálogo de productos que puede ser consultado por cualquier tipo de usuario.
- Los empleados pueden consultar el catálogo y solo pueden hacer añadidos en el apartado de turnos.
- Los administradores pueden consultar el catálogo, hacer añadidos en cualquier ámbito (turnos, pedidos, usuarios…) y por último es el único capaz de hacer borrados definitivos en el sistema.
- El sistema cuenta con una función de inactividad o finalización con el que filtrar usuarios y pedidos o tareas para que no aparezcan en el catálogo general y mantener así un registro histórico, aquellas entidades con el atributo de finalización o inactividad en afirmativo no serán visibles más que por el administrador.

Requisitos no funcionales:

La definición de requisitos no funcionales es que son aquellos que no se refieren directamente a las funciones específicas suministradas por el sistema, sino a propiedades del sistema como rendimiento, seguridad o disponibilidad, hablan de cómo hace las cosas el sistema y no de qué hace.

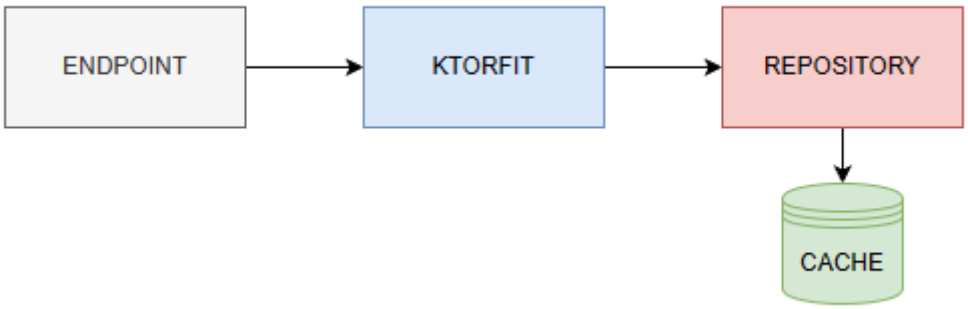
Entre los requisitos no funcionales de este programa se encuentran los siguientes:

- Uso de tokens para identificar usuarios disponibles en nuestra base de datos y cuyo token se utiliza para validar el nivel de acceso a las funcionalidades.
- La aplicación cuenta con una caché para agilizar el proceso de consultas, esta caché se actualiza cada 60 segundos.
- Si se realiza una consulta y este dato no se encuentra en la caché se realizará una petición al repositorio para que lo llame desde la base de datos.
- Hace falta tener conexión a internet para realizar las consultas a la API.
- Este programa está escrito en Kotlin usando Kmongo para trabajar con la base de datos no relacional.
- Caché realizada con cache4k.
- Programa asíncrono con el uso de corrutinas y funciones suspendidas.
- Programa totalmente reactivo.
- Uso de Ktorfit para consultar la API y realizar los routeos.
- Se puede consultar sin conexión siempre que esté la información cacheada.
- La caché no puede actualizarse sin conexión.

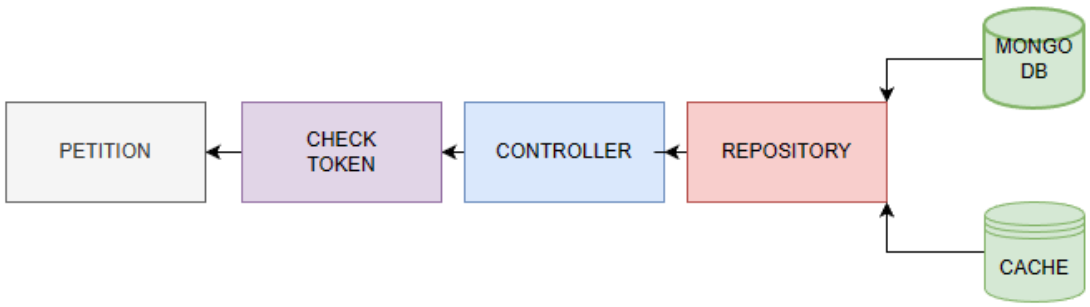
Requisitos de información:

La especificación de estos requisitos es una descripción completa del comportamiento del sistema, este incluye los casos de uso que describen todas las interacciones de los usuarios, estos engloban los requisitos funcionales y no funcionales.

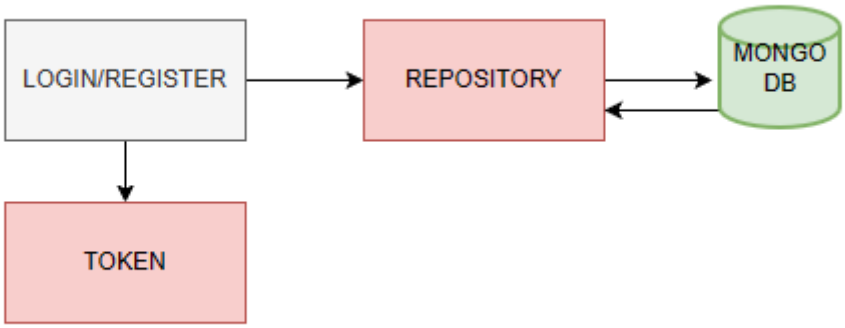
El programa consulta desde el endpoint **https://jsonplaceholder.typicode.com/** y guarda la información obtenida en nuestra base de datos no relacional a través del repositorio, el controlador nos permite comprobar entonces esa información haciendo consultas a nuestra base de datos en vez de a la API, las consultas ya realizadas se guardarán en la caché y esta se actualizará cada 60 segundos para tener la información más actualizada pero sin tener conectado el dispositivo todo el rato directamente a la base de datos realizando consultas de actualización en local.



Tenemos 3 tipos de usuarios, los usuarios normales que solo hacen consultas a el catálogo y no pueden hacer inserciones ni borrados en nuestro sistema pero si pueden realizar peticiones, los usuarios de tipo trabajador los cuales pueden consultar el catálogo y hacer inserciones solo en el apartado de turnos, también pueden realizar peticiones, por último tenemos a los usuarios de tipo administrador, estos pueden consultar el catálogo, hacer inserciones y borrados de todo tipo y realizar peticiones, estos últimos son los únicos con la capacidad de hacer borrados reales, puesto que cada entidad con las que trabajamos tiene un atributo de finalización o actividad que define la disponibilidad en nuestro sistema, si este atributo se encuentra en estado negativo no aparecerá en los registros accesibles para el resto de usuarios y sólo el administrador podrá verlos y realizar borrados reales sobre las entidades que considere.

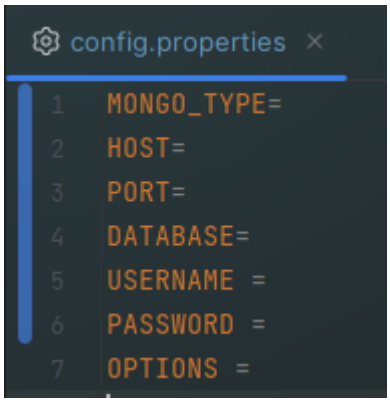


Para el funcionamiento de lo anterior tenemos un sistema de validación basado en tokens con el que dependiendo del nivel de acceso del token podrá realizar las acciones o las tendrá bloqueadas, para esto hemos usado JWT para la generación de esos tokens con los datos de nuestros usuarios, el token se genera cogiendo los datos de todos los parámetros del usuario y después codificándolos, de esta manera dentro del token figura el tipo de usuario del que se trata y podemos distinguir las funcionalidades que tiene disponibles, usamos una clave cifrada con HMAC256 para firmar estos tokens y confirmar su validez a la hora de decodificar los tokens en su posterior uso de confirmación del tipo de usuario.



Tenemos un sistema de cifrado en el que utilizamos Bcrypt, esto se utiliza para cifrar las contraseñas y guardarlas de esta manera para garantizar la seguridad de estas, usamos Bcrypt también para hacer la verificación a la hora de hacer los inicios de sesión para confirmar la autenticidad de las credenciales.

Uso de properties:



Mongo_Type: Sirve para distinguir el tipo de dirección de cadena que se utilizará para conectar con la base de datos de mongo, ya que se puede elegir entre la conexión con DNS SRV record que es lo que indica el “+srv” en la cadena de conexión, también puede ser una conexión con formato estándar de cadena de texto.

En la parte del host irá la dirección a la cuál deberá de conectarse la aplicación y a continuación se especificaría el puerto de funcionamiento y la base de datos a la cual se va a conectar. Por último, se introduce el usuario y la contraseña que utilizará para conectarse, este tiene que ya estar dado de alta en la base de datos para que funcione.

El apartado opciones en nuestro caso está vacío, pero aquí irían las variables que definen el modelo de autenticación, si usa TSL/SSL, si usa encriptado de mongo y si se conecta a un proxy entre otros.

```
MONGO_TYPE=mongodb+srv://
HOST=drmdam2023.7qtbnuz.mongodb.net
PORT=1707
DATABASE=practica03mongoIvanLoli
USERNAME = loli
PASSWORD = 1707
OPTIONS =
```

Explicación de tecnologías utilizadas:

Para la realización de esta aplicación hemos hecho dos versiones, una de ellas se conecta a mongo mediante el uso de ktorfit y trabaja directamente con ello, mientras que la segunda versión funciona a través de spring, utilizándolo para conectarse a el endpoint provisto para realizar la consulta y adquisición de los datos a almacenar.

En ambas versiones hemos usado Kotlin con corrutinas y su librería de serialización para hacer el paso de datos, con las corrutinas conseguimos hacer que la aplicación funcione de manera asíncrona intentando mejorar su rendimiento durante la ejecución.

Como trabajamos devolviendo JSON's con los datos hemos utilizado la librería de Google "Gson" para devolver los datos utilizando pretty printing a su vez para una visualización de datos más legible y limpia. Gson es una librería de código abierto que nos permite la serialización y deserialización de objetos en la JVM aportándonos representación de los objetos con notación JSON.

Para agilizar el tiempo de respuesta hemos implementado un cacheado de la información utilizando para ello la librería cache4k, de esta manera no creamos una caché en una base de datos alternativa en el dispositivo, sino que creamos un guardado en memoria disponible mientras el dispositivo esté encendido, cache4k también soporta funciones como tiempo de expiración y restricciones de tamaño por lo que era una propuesta interesante a implementar, solo soporta el nuevo modelo de memoria nativo de kotlin, pero es multiplataforma.

Necesitábamos hacer uso de tokens para la verificación de usuarios, para ello hemos hecho uso de JWT para que generase los tokens con los datos de nuestros modelos y así tener los tokens de validación que poder consultar y comprobar durante la ejecución del programa. JSON Web Tokens es el estándar para las representaciones de peticiones.

Para el guardado de las contraseñas hemos usado Bcrypt, un cifrador de alta potencia que nos permite implementar una capa de seguridad al no dejar las contraseñas visibles en la base de datos. Se trata de un algoritmo de hash con varias ventajas, entre ellas se encuentra el uso aleatorio de secuencias añadidas a las contraseñas para hacer más difícil la decodificación a la fuerza, de esta manera previene las tablas de hash de palabras claves.