ACCESO A DATOS

TENIS LAB BARCELONA

Realizado por:

Jorge Sánchez Berrocoso
Daniel Rodríguez Fernández
Alfredo Rafael Maldonado Pertuz

03/2/2023

Proyecto	Tennis Lab Barcelona		
Entregable	Especificación de Requisitos		
Autor	Daniel Rodríguez Fernández – Jorge Sánchez Berrocoso – Alfredo Maldonado		
Versión/Edición	V1.1	**Fecha Versión**	03/2/2023
Aprobado por		**Fecha Aprobación**	
		Nº Total de Páginas	52

PROJECT MANAGER
Daniel Rodríguez Fernández Alfredo Rafael Maldonado Pertuz Jorge Sánchez Berrocoso

Tabla de contenido

1. Introducción

Se ha planteado un nuevo problema en el módulo de acceso a datos de segundo de desarrollo de aplicaciones multiplataforma. En general hay que crear una solución para una tienda de material de tenis, esta solución nos debe permitir gestionar la personalización de material de los clientes, adquisición de nuevo material, gestionar los usuarios, gestionar los turnos de nuestros empleados así como las tareas que tienen asignadas.

Para resolver este problema vamos a implementar nuevas tecnologías que no habíamos implementado en la anterior práctica, como por ejemplo el uso de base de datos no relacionales, implementación de API's, cachear los datos almacenados.

2. Descripción del problema propuesto

Nos hemos reunido con el cliente donde nos ha entregado un documento con todas las especificaciones que debe cumplir este programa, así como con el jefe de la empresa quien nos ha indicado el uso de las nuevas tecnologías que debemos implementar para resolver dicho proyecto, a continuación dejamos el documento de partida para el desarrollo de este software:

En nuestra aplicación se conectan distintos usuarios. Pero debemos tener en cuenta que este servicio lo tenemos externalizado, por lo que debemos hacer uso de una API REST para operar con todo lo relacionado con él desde el endpoint: https://jsonplaceholder.typicode.com/users, estando este servicio cacheado y sincronizado en nuestro sistema para las operaciones CRUD y que la cache se refresca cada 60 segundos de manera automática.

Trabajamos con varias máquinas, que son de encordar o de personalización, como ya sabemos.

Los pedidos pueden estar formados por varias tareas o partes de trabajo a realizar que recibimos de un tenista y son asignados a un encordador y tienen un estado. Tenemos un tope de entrega marcado por una fecha. Los pedidos tienen una fecha de entrada, una fecha de salida programada y de salida final y un precio asociado que es la suma de todas las acciones. La fecha de salida final será inicialmente la programada, luego se actualizará a la real tal y como vimos en la práctica anterior. Debemos tener en cuenta restricciones:

Para cada tarea del pedido o acción a realizar a parte de almacenarla en nuestra base de datos la vamos a almacenar remotamente en un histórico en la nube al que podremos acceder mediante el endpoint: https://jsonplaceholder.typicode.com/todos. Necesitamos saber la raqueta o raquetas (una acción por raqueta) con la que trabajar si se necesita. Teniendo en cuenta todas las tareas posibles, precios y restricciones de la práctica anterior.

Debemos tener en cuenta que un encordador no puede tener más de dos pedidos activos por turno. Del turno nos interesa saber comienzo y fin del mismo. Un encordador no puede usar otra máquina si ya tiene asignada una en un turno determinado.

Además, como vendemos distintos productos del tipo: raquetas, cordajes, y complementos como overgrips, grips, antivibradores, fundas, etc. Necesitamos saber el tipo, marca, modelo, precio y stock del mismo. Ten en cuenta que solo podrá realizar operaciones CRUD el encargado del sistema, pero la asignación de pedidos la pueden hacer también los encordadores.

Por otro lado, nos interesa mantener el histórico de los elementos del sistema y:

- *Sistema de gestión de usuarios local y remoto sincronizados.*
- *CRUD completo de los elementos que consideres necesarios.*
- *Sistema de errores y excepciones personalizados.*
- *Información completa en JSON de un pedido.*
- *Listado de pedidos pendientes en JSON.*
- *Listado de pedidos completados en JSON.*
- *Listado de productos y servicios que ofrecemos en JSON.*
- *Listado de asignaciones para los encordadores por fecha en JSON.*
- *Gestión de tareas locales y remoto, histórico remoto.*

1. *Completar la información que te falta hasta tener los requisitos de información completos.*
2. *Realizar el Diagrama de Clases asociado, mostrando la semántica, navegabilidad y cardinalidad de las relaciones, justificando la respuesta con el máximo detalle. Obtener las colecciones existentes y aplicar patrones de diseño NoSQL razonando en cada punto la decisión adoptada y posibles mejoras en la resolución.*
3. *Implementación y test de repositorios, servicios y controladores reactivos de las operaciones CRUD y otras operaciones relevantes aplicando las restricciones indicadas usando solo MongoDB y un framework de inyección de dependencias. Elegir una colección en cada implementación y escuchar sus cambios en tiempo real.*
4. *Implementación y test de repositorios, servicios y controladores reactivos de las operaciones CRUD y otras operaciones relevantes aplicando las restricciones indicadas usando Spring Data MongoDB usando el sistema propio de inyección de dependencias que tiene Spring Data. Elegir una colección en cada implementación y escuchar sus cambios en tiempo real.*
5. *En las dos posibles implementaciones Se deben tipificar los resultados tanto correctos como incorrectos, como operaciones válidas o inválidas con el sistema que creas más acertado y mostrar as salidas en JSON de manera correcta con los DTOs adecuados.*
6. *Nuestro programa debe llamarse con un JAR de la siguiente manera: java -jar tennislab.jar.*

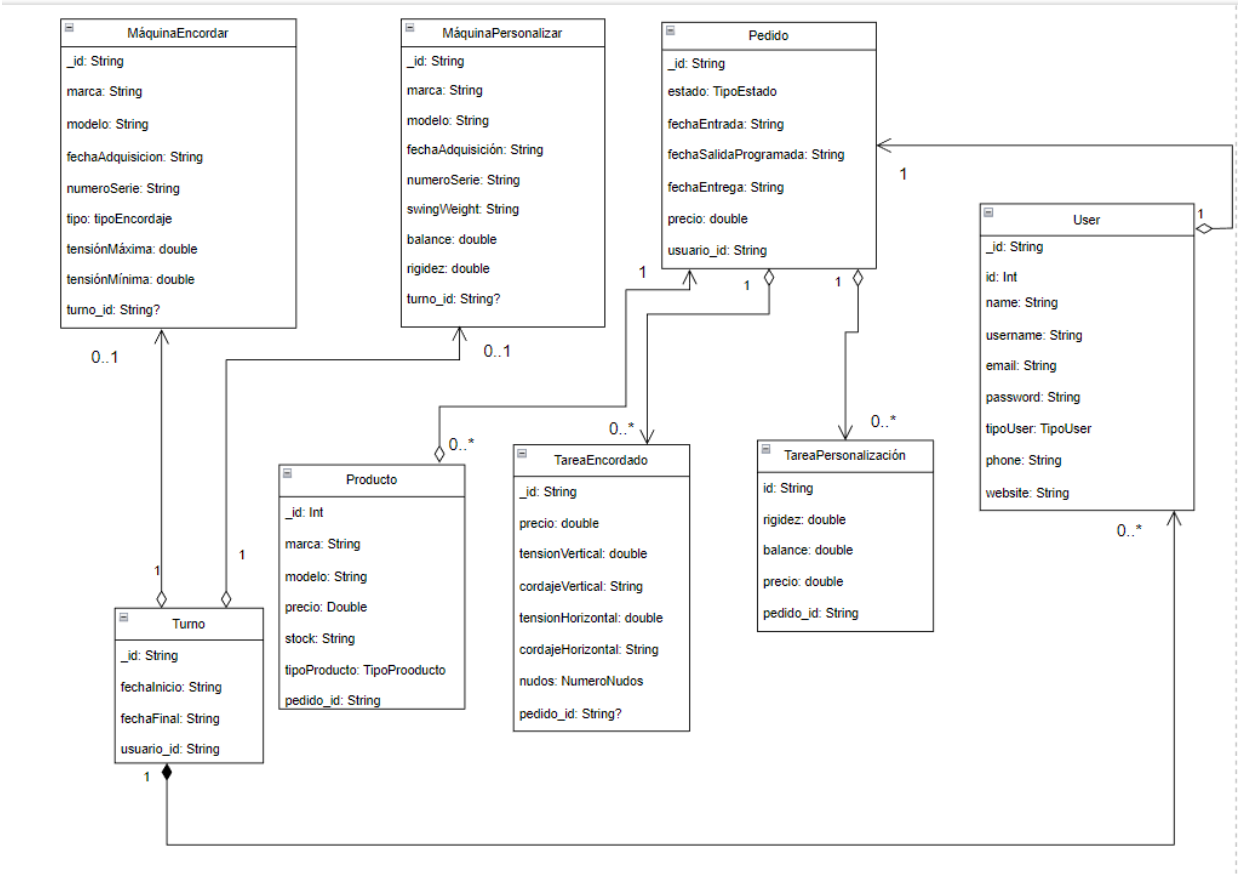
Se debe entregar:

- *Repositorio GitHub Personal y el de entrega con la solución en el que incluyas:*
- *Readme explicando el proyecto y el nombre de los integrantes. Usa Markdown y mejora su estilo. Si no perderás puntos por la presentación.*
- *Código fuente comentado y perfectamente estructurado con JDoc/KDoc. Además de los gitignore adecuados y que siga el flujo de trabajo GitFlow.*
- *No se deben incluir los ejecutables si no se deben poder crear los jar desde el propio proyecto. Asegúrate que se puede crear y que los ficheros de configuración de la base de datos, así como datos de ejemplo están en directorios que se pueden ejecutar o se pueden leer desde resources.*
- *Documentación en PDF donde expliques el diseño y propuesta de solución, así como clases y elementos usados haciendo especial énfasis en:*
- *Requisitos de Información.*
- *Diagrama de clases y justificación de este.*

- *Alternativas y justificación detallada del modelo NoSQL. Arquitectura del sistema y patrones usados.*
- *Explicación de forma de acceso a los datos.*
- *La no entrega de este fichero invalidará la práctica.*
- *La aplicación no debe fallar y debe reaccionar antes posibles fallos asegurando la consistencia y calidad de esta.*
- *Enlace en el readme al vídeo en YouTube donde se explique las partes más relevantes de la practica y se muestre su ejecución. La duración del vídeo debe ser unos 30 minutos. La no entrega de este vídeo y donde se vea su ejecución anulará el resultado de la práctica.*
- *Repositorio oficial de la entrega Enlace de entrega: https://classroom.github.com/a/GSctLb_o. La subirán los dos miembros del equipo, si no está en este repositorio se invalidará la práctica no pudiéndose entregar por otros medios.*

3. Diagramas

3.1. Diagrama de Clase



3.2. Justificación del diagrama de clase

A continuación explicamos la implementación del diagrama de clase que hemos desarrollado a partir de las indicaciones y requisitos del cliente.

Primero de todo vamos a identificar todas las clases que hemos implementado:

- User
- Turno
- TareaEncordado
- TareaPersonalizacion
- MaquinaEncordar
- MaquinaPersonalizar
- Pedido
- Producto

Entre estas clases se han creado relaciones para formar la arquitectura del software, más adelante explicaremos cada relación que se produce en el diagrama de clase.

Una de las primeras justificaciones que queremos dar es que hemos roto todas las herencias que se podían producir en el modelo de esta forma nos ha parecido más rápido a la hora de gestionar las relaciones entre las clases. Tecnológicamente también aportamos una explicación de esta forma a la hora por ejemplo de realizar una consulta la hacemos directamente sobre la clase a la hora de buscar será más rápido de la otra forma tendríamos que buscar la clase no heredada y de ahí saber qué tipo es en que hereda y ejecutar la consulta por lo tanto el tiempo sería mayor. Es verdad que utilizamos una NoSQL y por lo tanto podríamos embeber los datos o referenciar para hacer la herencia pero sí lo cambiamos a una SQL habría que hacer un mayor número de cambios en el modelo y en la implementación del software.

Vamos a comentar cada una de las clases y qué relación tiene con cada una de las clases del modelo.

- Users: esta clase hace referencia a los usuarios registrados de nuestro sistema, de cada usuario recogeremos algunos datos como puede ser su contraseña que estará encriptada con bcrypt o el tipo de usuario que es que le dará más o menos privilegios de realizar operaciones en el sistema.
  Para obtener los usuarios lo hemos hecho de forma diferente para obtener y operar con nuestros usuarios hemos tenido que implementar una API Rest en concreto la del siguiente endpoint "https://jsonplaceholder.typicode.com/todos", a través de este endpoint haremos las operaciones GET, POST, DELETE… sobre los usuarios.
  Tenemos que tener en cuenta que la clase usuarios es prescindible para poder operar con el resto de clases.
- Turno: con la clase turno registramos cada vez que un usuario que es de tipo empleado ha comenzado una nuevo turno, registrado su hora de inicio y su hora de fin hay que tener en cuenta que esta clase tiene una relación con Users de forma referenciada, ya que cada vez que registramos un turno a ese turno le asignamos el id del usuario que registra ese turno.
- Pedido: los pedido registran las solicitudes que realizan los usuarios, hay que destacar que los pedidos están relacionados con las tareas y los usuarios, primero explicamos la relación que tiene con los usuarios.

Un pedido esta realizado por un usuario por lo tanto a ese pedido se le asigna el usuario que lo realiza, para hacer esta asignación lo hacemos por referencia al pedido le asignamos el id del usuario que realiza el pedido.
Ahora explicamos la relación que tiene los pedidos con las tareas de personalización y de encordar. Un pedido esta formado por tareas, puede tener una o varias tareas pero nunca podrá ser null.

- TareaEncordado: es una clase que hace la composición al pedido, contiene su id, y el resto de los campos propios de la clase, además tiene un campo que es el número de pedido nunca puede ser null, una tarea siempre tiene que estar asociada a un pedido, lo hacemos de forma referenciada.
- TareaPersonalizacion: es otra clase que hace la composición del pedido, contiene su id, y el resto de campos que son propios de la clase, además tiene un campo que hace referencia el número de pedido que nunca podrá ser null, es decir que una tarea siempre tiene que estar asociada a un pedido, lo hacemos de forma referenciada a la clase pedido, de esta forma es mucho más rápido ya que no es necesario embeber todo solo aquello que es necesario.
- MaquinaEncordado: es una clase que utilizamos para registrar todas las maquinas de encordado existentes, hay que decir que comparte campos con las máquinas de personalización pero ya que hemos decidido que no exista herencia entre clases, cada clase de maquina tendrá sus propios campos.
  Además cada máquina de encordado tendrá asociado un turno de esta forma podemos filtrar y limitar el numero de maquinas que puede utilizar un usuario en cada turno.
- MaquinaPersonalizacion: es una clase que utilizamos para registrar todas las máquinas de personalización existentes, hay que decir que comparte campos con las máquinas de encordado pero ya que hemos decidido que no exista herencia entre clases, cada clase de maquina tendrá sus propios campos.
  Además cada máquina de encordado tendrá asociado un turno de esta forma podemos filtrar y limitar el número de máquinas que puede utilizar un usuario en cada turno.
- Producto: son productos o servicios que podemos ofrecer y pueden obtener los clientes, tiene atributos uno de ellos es el stock que es la cantidad de elementos existentes de ese producto.
  Tiene una relación con los pedido, la relación se hace de forma referenciada, para asignar un producto a un pedido hay que introducir el número de pedido que tenía asignado.

Relaciones:

Pedido-User: Es una relación 1-1 porque se ha decido para que el usuario esté dado de alta en la plataforma tiene que ser con un pedido, y por otro lado solo puede hacer un pedido en el momento.
Pedido-Producto: Es una relación 1-1* porque al hacer un pedido obligatoriamente tiene que haber un producto en el pedido y un máximo indefinido.
User-Turno: Es una relación 1-0..N porque tiene que haber un usuario para que pueda realizar el turno de trabajo, además sin no hay algún trabajador disponible no puede haber un turno y por ello no hay una máquina en uso ni una o varias tareas que se vayan a cumplir durante ese turno.

Turno-MaquinaEncordar: Se trata de una relación 0..N-1 porque la máquinaEncordar siempre va a existir aunque no se esté realizando ningún turno en ese momento, mientras que puede haber varios turnos que hagan uso de máquinas.

Turno-MaquinaPersonalizar: Al igual que el anterior, se trata de una relación 0..N-1 porque que la máquinaPersonalizar siempre va a existir aunque no se esté realizando ningún turno en ese momento, mientras que puede haber varios turnos que hagan uso de máquinas personalizadoras.

4. Requisitos

4.1. Requisitos Funcionales

Un requisito funcional define una función del sistema de software o sus componentes.

Los requisitos funcionales son: cálculos, detalles técnicos, manipulación de datos y otras funcionalidades específicas que nuestro sistema debe cumplir.

Cod	Requisito Funcional	Check
001	Al iniciar el programa pedirá los datos de inicio de sesión	✓
002	Al meter los datos de inicio de sesión el programa comprobará que está sesión es correcta.	✓
003	Al iniciar una sesión de Gerente correcta nos mostrará el menú de inicio de gerente.	✓
004	Al iniciar una sesión de Gerente correcta nos mostrará el menú de inicio de gerente.	✓
005	Al meter los datos de inicio de sesión erróneos el programa informará de ello y volverá al menú de inicio de sesión	✓
006	Al seleccionar el gerente cambio de contraseña en el menú el programa le mostrará las opciones del menú de cambio de contraseña	✓
007	Al seleccionar el empleado cambio de contraseña en el menú el programa pedirá su antigua contraseña y la nueva	✓
008	El sistema cuando el usuario inserte los datos correspondientes correctos en la opción de “cambiar contraseña” este comprobará los datos y cambiará está confirmará la acción.	✓
009	Al seleccionar el Gerente cambio de contraseña en el menú el programa le mostrará las opciones de cambio de contraseña.	✓
010	Al seleccionar el Gerente cambio de contraseña propia en el menú de cambio de contraseña le mostrará los empleados disponibles para el cambio.	✓
011	Al seleccionar el Gerente “Modificar Usuario”, el sistema le mostrará todos los usuarios del sistema; y pedirá que seleccione uno.	✓
012	Al seleccionar el Gerente “Modificar Usuario” y elegir un usuario, el sistema le mostrará todos datos de este y pedirá los datos nuevos.	✓
013	Al seleccionar el Gerente “Modificar Usuario”, elegir un usuario e introducir los datos, comprobará estos y mostrará un mensaje de confirmación u error dependiendo de si los datos son correctos y se han guardado correctamente.	✓
014	Al seleccionar el gerente “crear usuario” en el menú el programa le pedirá los datos para crear un usuario nuevo.	✓
015	Al seleccionar el Gerente “crear usuario” e introducir los datos, comprobará estos y mostrará un mensaje de	✓

	confirmación u error dependiendo de si los datos son correctos y se han guardado correctamente.	
016	Al seleccionar el gerente “Crear Pedido” en el menú el programa le mostrará un mensaje con el pedido a crear y opciones para añadir tarea, u cambiar datos de pedido.	✓
017	Al seleccionar el gerente “Crear Pedido”, y seleccionar la opción “modificar Datos” el programa pedirá los nuevos datos el pedido.	✓
018	Al seleccionar el gerente “Crear Pedido” seleccionar la opción “modificar Datos” e introducir los datos el programa comprobará los datos, mostrará la confirmación u error de los campos y volverá el menú de pedido.	✓
019	Al seleccionar el gerente “Crear Pedido”, y seleccionar la opción “añadir tarea” el programa pedirá los datos de la tarea, comprobará los datos, mostrará un mensaje de error o confirmación de estos y volverá al menú de pedido.	✓
020	Al seleccionar el gerente “Crear Pedido”, y seleccionar la opción “modificar tarea” el programa pedirá seleccionar la tarea a modificar, comprobará los datos, mostrará un mensaje de error o confirmación de estos e irá al menú correspondiente.	✓
021	Tras seleccionar la opción de crear pedido, si el cliente confirma la creación el programa guardará los datos correspondientes y mostrará un message de confirmación.	✓
022	Al seleccionar el gerente “Crear turno” el programa mostrará el menú correspondiente y pedirá los datos para la creación.	✓
023	Al seleccionar el gerente “Crear turno” e introducir los datos correspondientes, el programa comprobará los datos, mostrará un mensaje de confirmación u error y devolverá al menú anterior.	✓
024	Al seleccionar el gerente “modificar turno” e introducir los datos correspondientes, el programa comprobará los datos, mostrará un mensaje de confirmación u error y devolverá al menú anterior.	✓
025	Al seleccionar el gerente “modificar turno” e introducir los datos correspondientes, el programa comprobará los datos, mostrará un mensaje de confirmación u error y devolverá al menú anterior.	✓
026	Al seleccionar el gerente “asignar tareas” en el menú el programa le mostrará las tareas y empleados del sistema, y pedirá los datos para la asignación.	✓
027	Al seleccionar el gerente “asignar tareas” e introducir los datos para la asignación, el programa comprobará los datos y mandará un mensaje de confirmación u error la introducción de datos en el sistema.	✓

028	Al seleccionar el gerente “Modificar tarea” en el menú el programa le mostrará las tareas del sistema, y pedirá los datos para la asignación.	✓
029	Al seleccionar el gerente “Modificar tareas” e introducir los datos para la tarea, el programa comprobará los datos y mandará un mensaje de confirmación u error la introducción de datos en el sistema.	✓
030	Al seleccionar el gerente “crear Maquina” en el menú el programa le pedirá los datos de la maquina nueva.	✓
031	Al seleccionar el gerente “crear Maquina” e introducir los datos de la maquina nueva, el sistema comprobará que los datos son correctos y mostrará un mensaje de error o confirmación de la introducción e los datos en el sistema.	✓
032	Al seleccionar el gerente “Modificar Maquina” en el menú el programa le pedirá los datos de la maquina nueva,	✓
033	Al seleccionar el gerente “Modificar Maquina” e introducir los datos de la máquina, el sistema comprobará que los datos son correctos y mostrará un mensaje de error o confirmación de la introducción e los datos en el sistema.	✓
034	Al seleccionar el gerente “Crear producto” en el menú el programa le pedirá los datos correspondientes a la creación.	✓
035	Al seleccionar el gerente “Crear Raqueta” en el menú el programa le pedirá los datos correspondientes a la creación.	✓
036	Al seleccionar el gerente “Crear Raqueta” e introducir los datos el sistema comprobará que los datos son correctos y mostrará un mensaje de error o confirmación de la introducción de los datos en el sistema.	✓
037	Al seleccionar el gerente “Crear Producto” e introducir los datos el sistema comprobará que los datos son correctos y mostrará un mensaje de error o confirmación de la introducción de los datos en el sistema.	✓
038	Al seleccionar el Empleado “Modificar tarea” en el menú el programa le mostrará las tareas del sistema, y pedirá los datos para la asignación.	✓
039	Al seleccionar el Empleado “Modificar tareas” e introducir los datos para la tarea, el programa comprobará los datos y mandará un mensaje de confirmación u error la introducción de datos en el sistema.	✓
040	Al seleccionar el Empleado “modificar turno” e introducir los datos correspondientes, el programa comprobará los	✓

	datos, mostrará un mensaje de confirmación u error y devolverá al menú anterior.	
041	Al seleccionar el Empleado “modificar turno” e introducir los datos correspondientes, el programa comprobará los datos, mostrará un mensaje de confirmación u error y devolverá al menú anterior.	✓
042	Al seleccionar el Empleado “modificar turno” e introducir los datos correspondientes, el programa comprobará los datos, mostrará un mensaje de confirmación u error y devolverá al menú anterior.	✓
043	Al seleccionar el Empleado “asignar tareas” en el menú el programa le mostrará las tareas y empleados del sistema, y pedirá los datos para la asignación.	✓
044	Al seleccionar el Empleado “asignar tareas” e introducir los datos para la asignación, el programa comprobará los datos y mandará un mensaje de confirmación u error la introducción de datos en el sistema.	✓
045	Al seleccionar el Empleado “Modificar tarea” en el menú el programa le mostrará las tareas del sistema, y pedirá los datos para la asignación.	✓
046	Al seleccionar el Empleado “Modificar tareas” e introducir los datos para la tarea, el programa comprobará los datos y mandará un mensaje de confirmación u error la introducción de datos en el sistema.	✓
047	Al seleccionar el Empleado “Modificar tarea” en el menú el programa le mostrará las tareas del sistema, y pedirá los datos para la asignación.	✓
048	Al seleccionar el Empleado “Modificar tareas” e introducir los datos para la tarea, el programa comprobará los datos y mandará un mensaje de confirmación u error la introducción de datos en el sistema.	✓
049	Al seleccionar el Usuario “Volver” el programa iniciará el menú de sesión.	✓
050	Al seleccionar el Usuario “Salir” el programa se realizará una copia de seguridad y se cerrará.	✓
051	Al seleccionar el gerente “Imprimir Pedido” el sistema mostrará un listado de los pedidos en el sistema.	✓
052	Al seleccionar el gerente “Imprimir Pedido” he introducir pedido que se quiera, el sistema comprobará los datos, mostrará un mensaje de confirmación u error y en caso de confirmación creará fichero json en una carpeta del sistema.	✓
053	Al seleccionar el gerente “Imprimir Pedidos Pendientes” el sistema creará un listado de los pedidos pendientes en el sistema en Json.	✓
054	Al seleccionar el gerente “Imprimir Pedidos Completos” el sistema creará un listado de los pedidos completos en el sistema en Json.	✓

055	Al seleccionar el gerente “Imprimir Listado productos y servicios” el sistema creará un listado de los productos y servicios en el sistema en Json.	✓
056	Al seleccionar el gerente “Imprimir Listado asignaciones” el sistema creará un listado desde las asignaciones ordenado por fecha en el sistema en json.	✓

4.2. Requisitos No Funcionales

Los Requisitos No Funcionales se refieren a todos los requisitos que describen características de funcionamiento.

Se suelen clasificar en:

Requisitos de calidad de ejecución, que incluyen seguridad, usabilidad y otros medibles en tiempo de ejecución.

Requisitos de calidad de evolución, como testeabilidad, extensibilidad o escalabilidad, que se evalúan en los elementos estáticos del sistema software.

Cod	Requisito No Funcional	Check
RNF1	Ktorfit para realizar las operaciones de la API.	✓
RNF2	Mockito: para hacer test de los controladores.	✓
RNF3	Kmongo para realizar la conexión con la base de datos NoSQL con MongoDB.	✓
RNF4	Koin: realizar inyección de dependencias	✓
RNF5	Implementar SQLite para realizar la memoria caché	✓
RNF6	Serialization: kotlin	✓
RNF7	Fechas serializadas en todos los atributos con LocalDate implementando serialization.	✓
RNF8	Actualización de datos en tiempo real.	✓
RNF9	Ejecución del programa a través de un JAR.	✓
RNF10	SpringBoot como alternativa para montar el proyecto	✓
RNF11	Despliegue a través de docker.	✓

4.3. Requisitos de Información

Cod	Requisito de Información	Check
RI1	Users: • _id: String • name: String • username: String • email: String • password: String • tipoUser: String • phone: String • website: String	✓
RI2	Turno: • _id: String • fechaInicio: LocalDate fechaFinal: LocalDate • usuario_id: String	✓
RI3	MaquinaEncordar: • _id: String • marcaModelo: String modelo: String • fechaAdquisición: LocalDate	✓

	• numeroSerie: String • tipo: TipoEncordaje • tensiónMáxima: double • tensiónMínima: double • turno_id: String?	
RI4	MaquinaPersonalizar: • _id: String • marca: String modelo: String • fechaAdquisición: LocalDate • numeroSerie: String • swingweight: String • balance: double • rigidez: double • turno_id: string?	✓
RI5	Pedido: • _id: String • estado: String • fechaEntrada: LocalDate • fechaSalidaProgramada: LocalDate • fechaEntrega: LocalDate • precio: double • usuario_id: String	✓
RI6	Producto: • _id: String • marca: String • modelo: String • precio: double • stock: string • tipoProducto: TipoProducto • pedido_id: String	✓
RI7	TareaEncordado: • _id: String • precio: String • tensionVertical: String • cordajeVertical: double • tensionHorizontal: string • cordajeHorizontal: TipoProducto • nudos: NumeroNudos • pedido_id: String?	✓
RI8	TareaPersonalización: • _id: String • rigidez: String • peso: String • balance: double • precio: string	✓

	• pedido_id: String	

5. Alternativas y justificación detallada del modelo NoSQL.

Las base de datos relaciones (SQL) almacenan datos de manera estructurada y las NoSQL lo hacen en su formato original.

Las bases de datos SQL proporcionan una capacidad de escalar baja, en comparación con las NoSQL. Esta es una de las principales ventajas de las NoSQL, ya que están pensadas para grandes volúmenes de información como el Big Data. Lo anterior es debido a que las SQL están centralizadas y las NoSQL distribuidas, posibilitando que se ejecuten en múltiples máquinas pero con muy pocos recursos (RAM, CPU, disco…).

La adaptación a los cambios de las bases de datos SQL es poca y puede ser compleja. Sin embargo, las bases de datos NoSQL son totalmente flexibles.

Las bases de datos SQL están totalmente estandarizadas y las bases de datos NoSQL carecen de homogeneización.

Las bases de datos SQL se utilizan en múltiples aplicaciones de todo tipo, las bases de datos NoSQL se emplean principalmente para el Big Data (por ejemplo en redes sociales).

Las bases de datos SQL proporcionan consistencia en los datos (integridad). Sin embargo, las NoSQL, al buscar rapidez, no ponen el foco en esta característica.

La rapidez de ambas bases de datos va a depender del contexto o de su uso: en datos estructurados las bases de datos SQL son más rápidas, pero como vimos anteriormente, el Big Data no es estructurado y es ahí donde consiguen mucha mayor rapidez las NoSQL.

6. Evaluación y Análisis de Código

Paquetes en común que comparten ambos proyectos y no se diferencian entre los proyectos de Ktor y SpringBoot.

Docker: hemos creado un paquete docker donde lanzamos contenedores que contiene la base de datos de mongo.

```
services:
  # MONGO DB
  mongodb-server:
    image: mongo
    container_name: mongo-server
    ports:
      - 27017:27017
    expose:
      - 27017
    environment:
      MONGO_INITDB_ROOT_USERNAME: mongoadmin
      MONGO_INITDB_ROOT_PASSWORD: mongopass
      MONGO_INITDB_DATABASE: test
      # ME_CONFIG_MONGODB_URL: mongodb://root:example@mongo:27017/
    command: --auth
    # Monto la unidad y le cargo unos datos
    volumes:
      - ./init:/docker-entrypoint-initdb.d
      - mongo-vol:/data/db
    networks:
      - mongo-network
    # restart: always

  # MONGO EXPRESS
  mongo-express:
    image: mongo-express
```

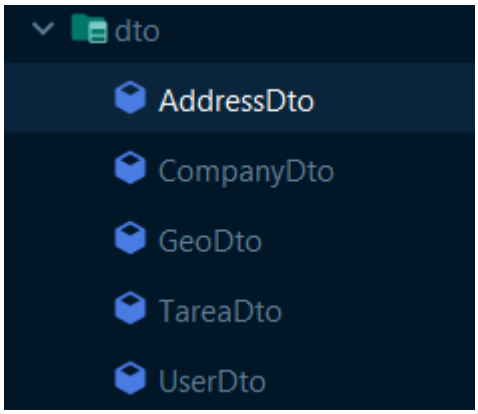
```
    container_name: mongo-empress
    ports:
      - 8081:8081
    networks:
      - mongo-network
    depends_on:
      - mongodb-server
    environment:
      ME_CONFIG_MONGODB_ADMINUSERNAME: mongoadmin
      ME_CONFIG_MONGODB_ADMINPASSWORD: mongopass
      ME_CONFIG_MONGODB_SERVER: mongodb-server
    restart: unless-stopped

# Mi volumenes de datos compartidos
volumes:
  mongo-vol:

# Si queremos que tengan una red propia a otros contenedores
networks:
  mongo-network:
    driver: bridge
```

Data: datos manuales que vamos a introducir en la base de datos de MongoDB.

```
object Data {

    val formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd'T'HH:mm:ss.SSSXXX")


    fun getUsers(): List<User> {
        return listOf(
            User(
                id = "55",
                name = "Jorge",
                username = "Juan",
                email = "juan@gmail.com",
                password = "hola",
                tipoUser = User.TipoUsuario.ADMIN,
                phone = "736363736",
                website = "http://www.juan",

            ))
    }
    fun getTurnos(): List<Turno> {
        return listOf(
            Turno(
                fechaInicio =  LocalDateTime.parse("2022-02-
15T00:00:00.000+05:30", formatter),
                fechaFinal =  LocalDateTime.parse("2024-02-
15T00:00:00.000+05:30", formatter),
            ))
    }

    fun getPedidos(): List<Pedido> {
        return listOf(
            Pedido(
                estado = Pedido.TipoEstado.EN_PROCESO,
                fechaEntrada =  LocalDateTime.parse("2021-02-
15T00:00:00.000+05:30", formatter),
                fechaSalidaProgramada =  LocalDateTime.parse("2022-02-
15T00:00:00.000+05:30", formatter),
                fechaEntrega =  LocalDateTime.parse("2022-02-
15T00:00:00.000+05:30", formatter),
                precio = 222.2
```

```
            ))
    }

    fun getTareasPersonalizacion(): List<TareaPersonalizacion> {
        return listOf(
            TareaPersonalizacion(
                rigidez = 8.00,
                peso = 1.20,
                balance = 1.10,
                precio = 222.2
            )
        )
    }

    fun getTareaEncordado(): List<TareaEncordado> {
        return listOf(
            TareaEncordado(
                precio = 8.00,
                tensionVertical = 1.20,
                cordajeVertical = "Cordaje Vertical",
                tensionHorizontal = 1.10,
                cordajeHorizontal = "Cordaje Horizontal",
                nudos = TareaEncordado.NumeroNudos.DOS,
            )
        )
    }

    fun getMaquinaEncordado(): List<MaquinaEncordar> {
        return listOf(
            MaquinaEncordar(
                marca = "Maquina marca Encordado 1",
                modelo = "Maquina modelo Encordado 1",
                fechaAdquisicion = LocalDateTime.parse("2022-02-
15T00:00:00.000+05:30", formatter),
                numeroSerie = "3345",
                tipo = MaquinaEncordar.TipoEncordaje.AUTOMATICA,
                tensionMaxima = 12.32,
                tensionMinima = 11.00,
            )
        )
    }

    fun getMaquinaPersonalizacion(): List<MaquinaPersonalizar> {
        return listOf(
            MaquinaPersonalizar(
                marca = "Maquina marca Personalizacion 1",
                modelo = "Maquina modelo Personalizacion 1",
                fechaAdquisicion =  LocalDateTime.parse("2022-02-
15T00:00:00.000+05:30", formatter),
                numeroSerie = "3345",
                swingweight = "true",
                balance = 12.32,
                rigidez = 11.00,
            )
        )
    }
}
```

Dto: el paquete dto es el mismo en ambos proyectos ya que hay datos de la clase que están embebidos. Podemos ver los paquetes que hemos tenido que crear y el contenido de uno de los paquetes que embebemos, le implementamos la anotación Serializable.




```
@Serializable
data class AddressDto(
    val street: String,
    val suite: String,
    val city: String,
    val zipcode: String,
    val geo: GeoDto
)
```

Excepciones: hemos creado excepciones personalizadas para cada clase y de esta forma tener una excepción por cada elemento. En la siguiente imagen podemos ver por ejemplo las excepciones que tenemos creadas para la clase MaquinaEncordar.

```
class MaquinaEncordarException(message: String) : RuntimeException(message)
class MaquinaEncordarControllerException(message: String) :
RuntimeException(message)
class MaquinaEncordarCachedException(message: String) : RuntimeException(message)
```

Mapper: utilizamos esta clase para convertir los datos del usuario de la clase a la base de datos y de la base de datos al modelo basado en la API.

```
fun toModel(user: UserDto): User {
        return User(
            id = user.id,
            name = user.name,
            username = user.username,
            email = user.email,
            password = Contraseñas.encriptar("secreto"),
            tipoUser = User.TipoUsuario.CLIENTE,
            phone = user.phone,
            website = user.website
        )
}

fun databaseToModel(entity: database.User) : User {
    if (entity.tipoUser == "ADMIN") {
        return User(
            _id = ObjectId(entity.id),
            id = entity.id,
            name = entity.name,
            username = entity.username,
            email = entity.email,
            password = entity.password,
```

```
            tipoUser = User.TipoUsuario.ADMIN,
            phone = entity.phone,
            website = entity.website
        )
    } else if (entity.tipoUser == "CLIENTE") {
        return User(
            _id = ObjectId(entity.id),
            id = entity.id,
            name = entity.name,
            username = entity.username,
            email = entity.email,
            password = entity.password,
            tipoUser = User.TipoUsuario.CLIENTE,
            phone = entity.phone,
            website = entity.website
        )
    } else {
        return User(
            _id = ObjectId(entity.id) ,
            id = entity.id,
            name = entity.name,
            username = entity.username,
            email = entity.email,
            password = entity.password,
            tipoUser = User.TipoUsuario.TRABAJADOR,
            phone = entity.phone,
            website = entity.website
        )
    }
}
```

Serializers: utilizamos esta clase para serializar las fechas con Kotlin.

```
object LocalDateTimeSerializer : KSerializer<LocalDateTime> {
    override val descriptor = PrimitiveSerialDescriptor("LocalDateTime",
PrimitiveKind.STRING)

    override fun deserialize(decoder: Decoder): LocalDateTime {
        return LocalDateTime.parse(decoder.decodeString())
    }

    override fun serialize(encoder: Encoder, value: LocalDateTime) {
        encoder.encodeString(value.toString())
    }
}
```

KtorFitRest: creamos los metodos de los endpoint con Ktorfit usando los @GET, @POST, @PATCH, @PUT, @DELETE.

```
interface KtorFitRest {
 @GET("users")
 suspend fun getAll(@Query("page") page: Int = 0, @Query("per_page") perPage: Int
= 0): List<UserDto>
 @GET("todos")
 suspend fun getAllTareas(@Query("page") page: Int = 0, @Query("per_page")
perPage: Int = 0): List<TareaDto>
 @POST("todos")
 suspend fun create(@Body usuario: TareaDto): TareaDto
 @PUT("todos/{id}")
 suspend fun update(@Path("id") id: String, @Body usuario: TareaDto): TareaDto
 @PATCH("todos/{id}")
 suspend fun upgrade(@Path("id") id: Long, @Body usuario: TareaDto): TareaDto
 @DELETE("todos/{id}")
```

```
    suspend fun delete(@Path("id") id: String): TareaDto
}
```

Ktorfitclient: creamos el cliente Ktorfit que se encarga de consumir la API y el endpoint correspondiente.

```
@Service
object KtorFitClient {
    private const val API_URL = "https://jsonplaceholder.typicode.com/"
    private val ktorfit by lazy {
        Ktorfit.Builder()
            .httpClient {
                install(ContentNegotiation) {
                    json(Json { isLenient = true; ignoreUnknownKeys = true })
                }
                install(DefaultRequest) {
                    header(HttpHeaders.ContentType, ContentType.Application.Json)
                }
            }
            .baseUrl(API_URL)
            .build()
    }
    val instance by lazy {
        ktorfit.create<KtorFitRest>()
    }
}
```

Utils/Contraseña: creamos esta clase para cifrar las contraseñas del usuario y descifrarlas con implementando Bcrypt para ello.

```
object Contraseñas {
    fun encriptar(password:String):String{
        return Hashing.sha512().hashString(password,
StandardCharsets.UTF_8).toString()
    }
}
```

CacheClient: servicio para cachear los datos de los usuarios, implementamos el Driver de SQLite y que lo haga en memoria.

```
@Service
object CacheClient {
    private val driver: SqlDriver = JdbcSqliteDriver(JdbcSqliteDriver.IN_MEMORY)

    init {
        AppDatabase.Schema.create(driver)
    }

    val queries = AppDatabase(driver).appDatabaseQueries

    fun removeAllData() {
        queries.transaction {
            queries.removeAllUsers()
        }
    }
}
```

Listados: los utilizamos para crear los servicios de imprimir datos de listados en formatos como JSON por ejemplo.

```
object Listados {
    var productos= mutableListOf<Producto>()
    var servicios = mutableListOf<TareaDto>()
    var pedidosCompletados = mutableListOf<Pedido>()
    var pedidosPendientes = mutableListOf<Pedido>()
    var tareasCreadasEncordar = mutableListOf<TareaEncordado>()
    var tareasCreadasPersonalizar = mutableListOf<TareaPersonalizacion>()
}
```

6.1. Ktor

Aquí explicamos cada elemento del programa en Ktor que se diferencia del proyecto realizado con SpringBoot.

Implementación de Koin: hemos utilizado Koin como inyector de dependencias, debido a que el uso de anotaciones tal y como implementa koin nos parece mucho más sencillo y tecnológicamente nos ofrece un mayor número de ventajas sobre otros inyectores de dependencias como puede ser Dagger2. Con este inyector de dependencias hemos conseguido evitar crear grandes constructores y su mantenimiento de esta forma hemos ahorrado tiempo programando ciertas parte del software.

Repositorios: a diferencia con spring no se crean los métodos de forma automática sino que los tenemos que implementar de forma manual por nuestra parte. Para ello utilizando principios SOLID, vamos a implementar el principio de responsabilidad única de forma que el repositorio solo se encargue de almacenar también implementamos interfaces para evitar acoplar código.

A continuación podemos ver el ejemplo de la implementación de uno de los repositorios. Primero vemos la interfaz y a continuación la implementación.

```
interface IUsersRepository : CrudRepository<User, String> {}
```

```
//Hay que hacerlo con flujos para que sea reactivo
@Single
@Named("UserRepository")
class UserRepository : IUsersRepository {

    /**
     * Find all: Método para obtener todos los usuarios.
     *
     * @return Devuelve todos los usuarios.
     */
    override fun findAll(): Flow<User>{
        val list =
MongoDbManager.database.getCollection<User>().find().publisher.asFlow()
        try {
            return list
        }catch (e: Exception){
            throw UserException("Ha ocurrido un error al obetener los
usuarios")
        }

    }

    /**
     * Find all users: Metodo para obtener todos los usuarios.
     *
     * @return Devuelve
     */
    fun findAllUsers(): Flow<User>{
        val list =
MongoDbManager.database.getCollection<User>().find().publisher
```

```
        try {
            return list.asFlow()
        }catch (e: Exception){
            throw UserException("Ha ocurrido un error al obetener los
usuarios")
        }

    }

    /**
     * Find by id: Método para obtener un usuario por su id.
     *
     * @param id
     * @return Devuelve un usuario.
     */
    override suspend fun findById(id: String): User? {
        val enti
=MongoDbManager.database.getCollection<User>().findOneById(id)
        try {
            return enti

        }catch (e: Exception){
            throw UserException("Ha ocurrido un error al obtener el usuario
con id: $id")
        }
    }

    /**
     * Find by email: Método para obtener un usuario por su email.
     *
     * @param email
     * @return retorna el usuario con su email.
     */
    suspend fun findByEmail(email: String): User? {
        val list = findAll().toList()
        val enti = list.filter{ it.email == email }.firstOrNull()
        try {
            return enti

        }catch (e: Exception){
            throw UserException("Ha ocurrido un error al obtener el usuario
con email: $email")
        }
    }


    /**
     * Save: Método para insertar un usuario.
     *
     * @param entity
     */
    override suspend fun save(entity: User) {
        try {
            val enti =
MongoDbManager.database.getCollection<User>().save(entity)
        }catch (e: Exception){
            throw UserException("Ha ocurrido un error al insertar el usuario
$entity")
        }
    }

    /**
     * Delete: Método para borrar un usuario.
     *
     * @param _id
```

```
     */
    override suspend fun delete(_id: String ) {
        try {
            val enti =
MongoDbManager.database.getCollection<User>().deleteOneById(_id)
        }catch (e: Exception){
            throw UserException("Ha ocurrido un error al borrar el usuario ")
        }
    }

    /**
     * Update: Método para actualizar un usuario.
     *
     * @param entity
     */
    override suspend fun update(entity: User) {
        try {
            val enti =
MongoDbManager.database.getCollection<User>().updateOneById(entity._id,
entity)
        } catch (e: Exception) {
            throw UserException("Ha ocurrido un error al actualizar el usuario
$entity")
        }
    }
}
```

Controllers: se encargan de gestionar la lógica del programa según sea necesario almacenar en la memoria caché, en la API de forma remota o en MongoDB.

```
@Single
@Named("TennisLabController")
class TennisLabController(
    @Named("UserRepository") private var userRepository: UserRepository,
    @Named("UserRepositoryCache") private var userRepositoryCache:
UserRepositoryCache,
    @Named("TurnoRepository") private var turnoRepository: TurnoRepository,
    @Named("TareaPersonalizacionRepository") private var
tareaPersonalizarRepository : TareaPersonalizacionRepository,
    @Named("TareaEncordadoRepository") private var tareaEncordarRepository :
TareaEncordadoRepository,
    @Named("PedidoRepository") private val pedidoRepository: PedidoRepository,
    @Named("KtorfitRepository") private val ktorfitRepository :
KtorfitRepository,
    @Named("ProductoRepository") private val productoRepository:
ProductoRepository,
    @Named("MaquinaPersonalizarRepository") private var
maquinaPersonalizarRepository : MaquinaPersonalizarRepository,
    @Named("MaquinaEncordarRepository") private var maquinaEncordarRepository
: MaquinaEncordarRepository,
) {

    //-------------------------------------USERS-------------------------------
------------------------

    suspend fun getUsersByEmail(email: String, password: String) : User?{
        var user = userRepository.findByEmail(email)
        if(user == null){
            throw UserControllerException("El usuario no exite")
        }else{
            if (user.password != Contraseñas.encriptar(password)){
                throw UserControllerException("Contraseña incorrecta")
            }
```

```
        }
        return user
    }

    suspend fun getUsersById(id: String) : User?{
        val userCache = userRepositoryCache.findById(id)
        if(userCache == null) {
            var user = userRepository.findById(id)
            if (user == null) {
                throw UserControllerException("El usuario no existe")
            }
            return user
        }else{
            return userCache
        }
    }

    suspend fun añadirUser(entity: User){
        var usuario = userRepository.findByEmail(entity.email)
        if(usuario != null){
            throw UserControllerException("Ya hay un usuario con ese email")
        }else{
            userRepository.save(entity)
            userRepositoryCache.save(entity)
        }
    }

    suspend fun findAllUsers() : Flow<User> {
        val listUsuarios = userRepositoryCache.findAll()
        if (listUsuarios == null){
            return userRepository.findAll().flowOn(Dispatchers.IO)
        }else{
            return listUsuarios
        }

    }

    suspend fun updateUser(entity: User){
        userRepositoryCache.update(entity)
        userRepository.update(entity)
    }


    suspend fun deleteUser(_id: String){
        userRepository.delete(_id)
        userRepositoryCache.delete(_id)
    }

    //-----------------------------------------------------TURNOS------------
    suspend fun findAllTurnos() : Flow<Turno> {
        return turnoRepository.findAll().flowOn(Dispatchers.IO)
    }


    suspend fun getTurnosById(id: String) : Turno?{
        var user = turnoRepository.findById(id)
        if (user == null) {
            throw TurnoControllersException("El turno no existe")
        }
        return user
    }

    suspend fun añadirTurnos(entity: Turno){
        turnoRepository.save(entity)
    }
```

```
    suspend fun actualizarTurnos(entity: Turno){
        turnoRepository.update(entity)
    }


    suspend fun deleteTurnos(_id: String){
        turnoRepository.delete(_id)
    }


    //-----------------------------------------------------TAREAS---------------

    suspend fun findAllTareaPersonalizar() : Flow<TareaPersonalizacion>{

        return tareaPersonalizarRepository.findAll().flowOn(Dispatchers.IO)
    }


    suspend fun getTareasPersonalizacionById(id: String) :
TareaPersonalizacion?{

        var tarea = tareaPersonalizarRepository.findById(id)
        if (tarea == null) {
            throw TareaPersonalizacionControllerException("La tarea de
personalizacion no existe")
        }else{
            return tarea
        }
    }

    suspend fun añadirTareasPersonalizacion(entity: TareaPersonalizacion){
        val pedido = pedidoRepository.findById(entity.pedido_id!!)
        if (pedido != null) {
            val tarea = TareaDto(
                userId = pedido.usuario_id,
                id = entity._id,
                title = "Tarea Personalizacio",
                completed = pedido.estado.toString()
            )
            ktorfitRepository.create(tarea)
        }
        tareaPersonalizarRepository.save(entity)
    }

    suspend fun actualizarTareasPersonalizacion(entity: TareaPersonalizacion){
        tareaPersonalizarRepository.update(entity)
    }

    suspend fun deleteTareasPersonalizacion(_id: String){
        tareaPersonalizarRepository.delete(_id)
    }

    suspend fun findAllTareaEncordar() : Flow<TareaEncordado>{

        return tareaEncordarRepository.findAll().flowOn(Dispatchers.IO)
    }

    suspend fun getTareasEncordarById(id: String) : TareaEncordado?{

        var tarea = tareaEncordarRepository.findById(id)
        if (tarea == null) {
            throw TareaEncordadoControllerException("La tarea de encordado no
existe")
        }
```

```
        return tarea
    }


    suspend fun añadirTareasEncordado(entity: TareaEncordado){
        val pedido = pedidoRepository.findById(entity.pedido_id!!)
        if (pedido != null) {
            val tarea = TareaDto(
                userId = pedido.usuario_id,
                id = entity._id,
                title = "Tarea de Encordado",
                completed = pedido.estado.toString()
            )
            ktorfitRepository.create(tarea)
            println("POST -> TAREA SUBIDA")
        }
        tareaEncordarRepository.save(entity)
    }

    suspend fun actualizarTareasEncordado(entity: TareaEncordado){
        tareaEncordarRepository.update(entity)
    }


    suspend fun deleteTareasEncordadp(_id: String){
        tareaEncordarRepository.delete(_id)
    }

//-----------------------------------------------------PRODUCTOS------------


    suspend fun findAllProducto() : Flow<Producto>{
        return productoRepository.findAll().flowOn(Dispatchers.IO)
    }


    suspend fun getProductoById(id: String) : Producto?{
        var producto = productoRepository.findById(id)
        if (producto == null) {
            throw ProductoControllerException("El producto no existe")
        }
        return producto
    }

    suspend fun añadirProducto(entity: Producto){
        productoRepository.save(entity)
    }

    suspend fun actualizarProducto(entity: Producto){
        productoRepository.update(entity)
    }


    suspend fun deleteProducto(id: String){
        productoRepository.delete(id)
    }

    //-----------------------------------------------PEDIDOS-----------------


    suspend fun findAllPedidos() : Flow<Pedido>{
        return pedidoRepository.findAll().flowOn(Dispatchers.IO)
    }
```

```
    suspend fun getPedidoById(id: String) : Pedido?{
        var pedido = pedidoRepository.findById(id)
        if (pedido == null) {
            throw PedidoControllerException("El pedido no existe")
        }
        return pedido

    }

    suspend fun añadirPedidos(entity: Pedido){
        pedidoRepository.save(entity)
    }

    suspend fun actualizarPedido(entity: Pedido){
        pedidoRepository.update(entity)
    }


    suspend fun deletePedido(id: String){
        pedidoRepository.delete(id)
    }

    //-------------------------------------------------MAQUINAS----------------


    suspend fun findAllMaquinaPersonalizar() : Flow<MaquinaPersonalizar>{
        return maquinaPersonalizarRepository.findAll().flowOn(Dispatchers.IO)
    }


    suspend fun getMaquinaPersonalizacionById(id: String) :
MaquinaPersonalizar?{
        var maquina = maquinaPersonalizarRepository.findById(id)
        if (maquina == null) {
            throw MaquinaPersonalizarControllerException("La maquina de
personalizacion no existe con ese id")
        }
        return maquina

    }

    suspend fun añadirMaquinaPersonalizacion(entity: MaquinaPersonalizar){
        maquinaPersonalizarRepository.save(entity)
    }

    suspend fun actualizarMaquinaPersonalizacion(entity: MaquinaPersonalizar){
        maquinaPersonalizarRepository.update(entity)
    }


    suspend fun deleteMaquinaPersonalizacion(_id: String){
        maquinaPersonalizarRepository.delete(_id)
    }



    suspend fun findAllMaquinaEncordar() : Flow<MaquinaEncordar>{
        return maquinaEncordarRepository.findAll().flowOn(Dispatchers.IO)
    }


    suspend fun getMaquinaEncordarById(id: String) : MaquinaEncordar?{

        var maquina = maquinaEncordarRepository.findById(id)
        if (maquina == null) {
```

```
            throw MaquinaEncordarControllerException("La maquina de encordar
no existe con ese id")
        }
        return maquina
    }

    suspend fun añadirMaquinaEncordar(entity: MaquinaEncordar){
        maquinaEncordarRepository.save(entity)
    }

    suspend fun actualizarMaquinaEncordar(entity: MaquinaEncordar){
        maquinaEncordarRepository.update(entity)
    }


    suspend fun deleteMaquinaEncordar(_id: String){
        maquinaEncordarRepository.delete(_id)
    }


    //-----------------------------------------------API REST ---------------

    suspend fun findAllUsersApi() : Flow<UserDto>{
        return ktorfitRepository.findAll().flowOn(Dispatchers.IO)
    }

    suspend fun findAllTareasApi() : Flow<TareaDto>{
        return ktorfitRepository.findAllTareas().flowOn(Dispatchers.IO)
    }


    suspend fun añadirTareaApi(entity: TareaDto){
        ktorfitRepository.create(entity)
    }


    suspend fun actualizarTareaApi(tarea: TareaDto) {
        ktorfitRepository.update(tarea)
    }

    suspend fun deleteTareaApi(id: String) {
        ktorfitRepository.delete(id)
    }

}
```

Cache: SQLDelight es una biblioteca para el manejo de bases de datos en Android que permite crear una cache de la base de datos en el dispositivo para un acceso más rápido. Para actualizar la cache cada 60 segundos desde el repositorio, se puede utilizar una tarea programada para ejecutar la actualización periódicamente. Haciendo un findAll de la base de datos de mongo insertándolo en la caché.

```
private val logger = KotlinLogging.logger {}
private const val REFRESH_TIME = 6 * 10000L
@Single
@Named("UserRepositoryCache")
class UserRepositoryCache(@Named("CacheClient") client: CacheClient ) :
IUserRepositoryCached {
    private val remote = UserRepository()
    private val cached = client.queries
```

```
    suspend fun refresh() = withContext(Dispatchers.IO) {
        launch {
            do {
                cached.removeAllUsers()
                val listUsuarios = remote.findAll()
                listUsuarios.collect {

cached.insertUser(it._id,it.id,it.name,it.username,it.email,it.password,it.tip
oUser.toString(),it.phone,it.website)
                }
                delay(REFRESH_TIME)
            } while (true)
        }
    }

    override fun findAll(): Flow<User> {
        val listaUsuarios = mutableListOf<User>()
        try {
            val list =  cached.selectUsers().executeAsList()
            list.forEach {
                val a = databaseToModel(it)
                listaUsuarios.add(a)
            }
            return listaUsuarios.asFlow()
        }catch (e: UserCachedException){
            throw UserCachedException("Error al obtener todos los usuarios de
la cache")
        }
    }

    override suspend fun findById(id: String): User? {
        try {
            val user = cached.selectByIdUser(id).executeAsOneOrNull()
            if (user != null){
                val entity = databaseToModel(user)
                return entity
            }else{
                throw UserCachedException("No se encuentra el usuario con el
id $id")
            }
        }catch (e: UserCachedException){
            throw UserCachedException("Error al obtener el usuario con id:
$id")
        }
    }

    suspend fun save(entity: User) {
        try {

cached.insertUser(entity._id,entity.id,entity.name,entity.username,entity.emai
l,entity.password,entity.tipoUser.toString(),entity.phone,entity.website)
        }catch (e: UserCachedException){
            throw UserCachedException("Error al insertar usuario en la cache")
        }

    }

    override suspend fun update(entity: User) {
        try {

cached.updateUser(entity.id,entity.name,entity.username,entity.email,entity.pa
ssword,entity.tipoUser.toString(),entity.phone,entity.website,entity._id)
        }catch (e: UserCachedException){
            throw UserCachedException("Error al actualizar usuario en la
cache")
```

```
        }
    }


    override suspend fun delete(id:String) {
        try {
            cached.deleteUser(id)
        }catch (e: UserCachedException){
            throw UserCachedException("Error al borrar usuario de la cache")
        }
    }
}
```

View: se encarga de toda la lógica que gestiona el usuario así como el Login del usuario, calculo de pedidos, productos y tareas.

```
@Single
class TennisLabView(
    @Named("TennisLabController") private var tennisLabController:
TennisLabController
) {
   private val formatter = DateTimeFormatter.ofPattern("yyyy-MM-
dd'T'HH:mm:ss.SSSXXX")
    private lateinit var usuarioo : User
    private var pedido :Pedido? = null
    private lateinit var pedidoCliente : Pedido

    suspend fun login() {
        var email: String
        var password: String

        println("Iniciar Sesión")

        do {
            println("Correo: ")
            email = readln()
        } while (!email.matches(Regex("[a-zA-Z0-9_]+([.][a-zA-Z0-9_]+)*@[a-zA-
Z0-9_]+([.][a-zA-Z0-9_]+)*[.][a-zA-Z]{2,5}")))

        println("Contraseña: ")
        password = readln()

        var user = tennisLabController.getUsersByEmail(email,password)
        if (user != null){
            if(user.tipoUser == User.TipoUsuario.CLIENTE){
                usuarioo = user
                pedidoCliente = Pedido(
                    _id = "12345678914",
                    estado = Pedido.TipoEstado.RECIBIDO,
                    fechaEntrada = LocalDateTime.parse("2022-12-
27T00:00:00.000+05:30", Data.formatter),
                    fechaSalidaProgramada = LocalDateTime.parse("2023-08-
27T00:00:00.000+05:30", Data.formatter),
                    fechaEntrega = LocalDateTime.parse("2023-12-
27T00:00:00.000+05:30", Data.formatter),
                    precio = 34.3,
                    usuario_id = usuarioo._id
                )
                tennisLabController.añadirPedidos(pedidoCliente)
                bajarApi()
                accionesCliente()
                listasJson()
                println("Inicio de sesión terminado")
                exitProcess(0)
```

```
            }else if (user.tipoUser == User.TipoUsuario.TRABAJADOR){
                usuarioo = user
                pedidoCliente = Pedido(
                    _id = "12345678914",
                    estado = Pedido.TipoEstado.RECIBIDO,
                    fechaEntrada = LocalDateTime.parse("2022-12-
27T00:00:00.000+05:30", Data.formatter),
                    fechaSalidaProgramada = LocalDateTime.parse("2023-08-
27T00:00:00.000+05:30", Data.formatter),
                    fechaEntrega = LocalDateTime.parse("2023-12-
27T00:00:00.000+05:30", Data.formatter),
                    precio = 34.3,
                    usuario_id = usuarioo._id
                )
                tennisLabController.añadirPedidos(pedidoCliente)

                bajarApi()
                accionesEncordador()
                listasJson()
                println("Inicio de sesión terminado")
                exitProcess(0)

            }else if (user.tipoUser == User.TipoUsuario.ADMIN){
                usuarioo = user
                pedidoCliente = Pedido(
                    _id = "12345678914",
                    estado = Pedido.TipoEstado.RECIBIDO,
                    fechaEntrada = LocalDateTime.parse("2022-12-
27T00:00:00.000+05:30", Data.formatter),
                    fechaSalidaProgramada = LocalDateTime.parse("2023-08-
27T00:00:00.000+05:30", Data.formatter),
                    fechaEntrega = LocalDateTime.parse("2023-12-
27T00:00:00.000+05:30", Data.formatter),
                    precio = 34.3,
                    usuario_id = usuarioo._id
                )
                tennisLabController.añadirPedidos(pedidoCliente)

                bajarApi()
                accionesAdministrador()
                listasJson()
                println("Inicio de sesión terminado")
                exitProcess(0)
            }

        }else{
            println("Email o contraseña incorrectos")
            throw TennisLabControllerException("\"Email o contraseña
incorrectos\"")
        }


    }


    suspend fun bajarApi(){
        val list = tennisLabController.findAllUsersApi().toList()
        list.forEach {
            val u = toModel(it)
            println("Insetando usuarios")
            tennisLabController.añadirUser(u)
            println(u)
        }
```

```
        val listaTareas = tennisLabController.findAllTareasApi().toList()
        listaTareas.forEach {
            Listados.servicios.add(it)
        }
    }


    suspend fun accionesCliente(){
        var pedidoCliente = Pedido(
            _id = "123456789",
            estado = Pedido.TipoEstado.RECIBIDO,
            fechaEntrada = LocalDateTime.parse("2022-12-
27T00:00:00.000+05:30", Data.formatter),
            fechaSalidaProgramada = LocalDateTime.parse("2023-08-
27T00:00:00.000+05:30", Data.formatter),
            fechaEntrega = LocalDateTime.parse("2023-12-
27T00:00:00.000+05:30", Data.formatter),
            precio = 34.3,
            usuario_id = usuarioo!!._id
        )
        tennisLabController.añadirPedidos(pedidoCliente)
        realizarPedidoCliente()

    }

    private suspend fun accionesEncordador(){
        realizarPedidoCliente()
        comprobarPedidoCliente()
        cancelarPedidoCliente()
        consultarTodosPedidos()
        consultarTodosTurnos()
        consultarTodasMaquinas()
    }

    private suspend fun accionesAdministrador(){
        realizarPedidoCliente()
        comprobarPedidoCliente()
        cancelarPedidoCliente()
        consultarTodosPedidos()
        consultarTodosTurnos()
        consultarTodasMaquinas()
        crearUsuario()
        crearTurnos()
        crearMaquinas()
        borrarUsuario()
        borrarTurnos()
        borrarMaquinas()
    }


    private suspend fun crearUsuario() {
        println("CREAR USUARIO ADMIN")
        val user = User(
            _id = "555",
            id = "557",
            name = "pepe",
            username = "pepe345",
            email = "pepe@gmail.com",
            password = Contraseñas.encriptar("pepereina"),
            tipoUser = User.TipoUsuario.CLIENTE,
            phone = "665142892",
            website = "http://www.pepe.com",

            )
```

```
        tennisLabController.añadirUser(user)
        val userr = tennisLabController.getUsersById(user._id)
        if (userr != null) {
            println("User creado correctamento $userr")
        }else {
            println("Error al crear user: $userr")
        }
    }


    private suspend fun borrarUsuario() {
        println("BORRAR USUARIO")
        val user = tennisLabController.getUsersById("555")
        if (user != null) {
            tennisLabController.deleteUser(user._id)
            println("Usuario borrado correctamento $user")
        }else {
            println("No se encuentra el user con id 555 ")
        }
    }
    private suspend fun crearTurnos() {
        println("CREAR TURNO ADMIN")
        val turno = Turno(
            _id = "999",
            fechaInicio = LocalDateTime.parse("2022-12-27T00:00:00.000+05:30",
Data.formatter),
            fechaFinal = LocalDateTime.parse("2022-12-27T00:00:00.000+05:30",
Data.formatter),
            usuario_id = usuarioo!!._id
        )
        tennisLabController.añadirTurnos(turno)
        val turnoo = tennisLabController.getTurnosById(turno._id)
        if (turnoo != null) {
            println("Turno creado correctamento $turno")
        }else {
            println("Error al crear turno: $turno")
        }
    }

    private suspend fun borrarTurnos() {
        println("BORRAR TURNO ADMIN")
        val turnoo = tennisLabController.getTurnosById("999")
        if (turnoo != null) {
            tennisLabController.deleteTurnos(turnoo._id)
            println("Turno borrado correctamento $turnoo")
        }else {
            println("No se encuentra el turno con id 999 ")
        }
    }

    private suspend fun crearMaquinas() {
        println("CREAR MAQUINA ENCORDAR")
        val turno = tennisLabController.getTurnosById("999")
        val maquinaEncordar = MaquinaEncordar(
            _id = "888",
            marca = "Siemens",
            modelo = "Sparta",
            fechaAdquisicion = LocalDateTime.parse("2022-12-
27T00:00:00.000+05:30", Data.formatter),
            numeroSerie = "20234",
            tipo = MaquinaEncordar.TipoEncordaje.MANUAL,
            tensionMaxima = 56.5,
            tensionMinima = 23.9,
            turno_id = turno?._id
        )
```

```
        tennisLabController.añadirMaquinaEncordar(maquinaEncordar)
        val maquinaaa =
tennisLabController.getMaquinaEncordarById(maquinaEncordar._id)
        if (maquinaaa != null) {
            println("Maquina de Encordar creada correctamento $maquinaaa")
        }else {
            println("Error al crear maquina de encordar: $maquinaEncordar")
        }


        println("CREAR MAQUINA PERSONALIZAR")
        val turnoo = tennisLabController.getTurnosById("999")
        val maquinaPersonalizar = MaquinaPersonalizar(
            _id = "111",
            marca = "Siemens",
            modelo = "Sparta",
            fechaAdquisicion = LocalDateTime.parse("2022-12-
27T00:00:00.000+05:30", Data.formatter),
            numeroSerie = "20234",
            swingweight = "453",
            balance = 56.5,
            rigidez = 23.9,
            turno_id = turno?._id
        )
        tennisLabController.añadirMaquinaPersonalizacion(maquinaPersonalizar)
        val maquinaa =
tennisLabController.getMaquinaPersonalizacionById(maquinaPersonalizar._id)
        if (maquinaa != null) {
            println("Maquina de Personalizar creada correctamento $maquinaa")
        }else {
            println("Error al crear maquina de personalizar:
$maquinaEncordar")
        }
    }


    private suspend fun borrarMaquinas() {
        println("BORRAR MAQUINA ENCORDADORA")
        val maquinaa = tennisLabController.getMaquinaEncordarById("888")
        if (maquinaa != null) {
            tennisLabController.deleteMaquinaEncordar(maquinaa._id)
            println("Maquina encordadora borrada correctamento $maquinaa")
        }else {
            println("No se encuentra la maquina encordadora con id 888 ")
        }


        println("BORRAR MAQUINA PERSONALIZADORA")
        val maquinaPersonalizadora =
tennisLabController.getMaquinaPersonalizacionById("111")
        if (maquinaPersonalizadora != null) {

tennisLabController.deleteMaquinaPersonalizacion(maquinaPersonalizadora._id)
            println("Maquina encordadora borrada correctamento
$maquinaPersonalizadora")
        }else {
            println("No se encuentra la maquina encordadora con id 111 ")
        }
    }




    private suspend fun comprobarPedidoCliente() {
        println("COMPROBAR PEDIDOS CLIENTE")
```

```
        val lista = tennisLabController.findAllPedidos().toList()
        val list = lista.filter { it.usuario_id == usuarioo!!._id }
        obtenerPedidos(list)
    }

    private suspend fun cancelarPedidoCliente() {
        val lista = tennisLabController.findAllPedidos().toList()
        val list = lista.filter { it.usuario_id == usuarioo!!._id }
        val pedido = tennisLabController.getPedidoById(list[1]._id)
        if (pedido == null) {
            println("PEDIDO INCORRECTO")
        }else{
            tennisLabController.deletePedido(pedidoCliente._id)
            println("PEDIDO BORRADO")
        }
    }

    private fun obtenerPedidos(list: List<Pedido>){

        list.forEach {
            println("""
                    ${it.toString()}
                    """)
        }
    }

    private suspend fun realizarPedidoCliente(){

        val tarea1 = crearTareaEncordado()
        tennisLabController.añadirTareasEncordado(tarea1)
        Listados.tareasCreadasEncordar.add(tarea1)


        val tarea2 = crearTareaPersonalizar()
        println("Añadiendo TAREA DE PERSONALIZACIÓN $tarea2")
        tennisLabController.añadirTareasPersonalizacion(tarea2)
        Listados.tareasCreadasPersonalizar.add(tarea2)



        val tarea3 = crearTareaProducto()
        println("Añadiendo PRODUCTO $tarea3")
        tennisLabController.añadirProducto(tarea3)
        Listados.productos.add(tarea3)



        val pedido =
crearPedido(Listados.tareasCreadasEncordar,Listados.tareasCreadasPersonalizar,
Listados.productos)
        tennisLabController.añadirPedidos(pedido)
        println("Pedido creado: $pedido")

        obtenerPedidos(tennisLabController.findAllPedidos().toList())
        println("Saliendo de la creacion de tareas")

        Listados.tareasCreadasPersonalizar.clear()
        Listados.tareasCreadasEncordar.clear()
        Listados.productos.clear()
    }

    private fun crearPedido(tareasEncordar: MutableList<TareaEncordado>,
tareasPersonalizar: MutableList<TareaPersonalizacion>, producto:
MutableList<Producto>) : Pedido{
        var precioo : Double = 0.0
```

```
        tareasEncordar.forEach {
            precioo = precioo + it.precio
        }

        tareasPersonalizar.forEach {
            precioo = precioo + it.precio
        }

        producto.forEach {
            precioo = precioo + it.precio
        }

        val pedido = Pedido(
                    _id = newId<Pedido>().toString(),
                    estado = Pedido.TipoEstado.RECIBIDO,
                    fechaEntrada = LocalDateTime.parse("2022-12-
27T00:00:00.000+05:30", Data.formatter),
                    fechaSalidaProgramada = LocalDateTime.parse("2022-12-
27T00:00:00.000+05:30", Data.formatter),
                    fechaEntrega =LocalDateTime.parse("2022-12-
27T00:00:00.000+05:30", Data.formatter),
                    precio = precioo,
                    usuario_id = usuarioo!!._id
        )
        return pedido
    }

   private fun crearTareaPersonalizar(): TareaPersonalizacion{
        val precio = 20.0
        val peso = 28.0
        val balance = 56.6
        val rigidez = 98.9


        val tareaPersonalizar = TareaPersonalizacion(
            _id = newId<TareaPersonalizacion>().toString(),
            rigidez = rigidez,
            peso = peso,
            balance = balance,
            precio = precio,
            pedido_id = "12345678910")

       return tareaPersonalizar
    }

    private fun crearTareaEncordado() : TareaEncordado {
        val precio : Double = 20.0
        val tensionVertical = 10.3
        val tensionHorizontal = 2.2
        val cordajeVertical = "Si"
        val cordajeHorizontal = "No"

        val tarea = TareaEncordado(
                    _id = newId<TareaEncordado>().toString(),
                    precio = precio,
                    tensionVertical = tensionVertical,
                    cordajeVertical = cordajeVertical,
                    tensionHorizontal = tensionHorizontal,
                    cordajeHorizontal = cordajeHorizontal,
                    nudos = TareaEncordado.NumeroNudos.CUATRO,
                    pedido_id = "12345678910"
            )
        return tarea
    }
```

```
    private fun crearTareaProducto() : Producto {
        val producto =
            Producto(
                marca = "Babolat",
                modelo = "DAM",
                precio = 22.0,
                stock = "12",
                tipoProducto = Producto.TipoProducto.RAQUETA,
                pedido_id = pedidoCliente._id
            )
        return producto
    }


    private suspend fun consultarTodosPedidos(){
        println("TODOS LOS PEDIDOS")
        val list = tennisLabController.findAllPedidos().toList()
        println(list)
        println("PEDIDOS COMPLETADOS")
        val listaCompletados = list.filter { it.estado ==
Pedido.TipoEstado.TERMINADO }.toList()
        listaCompletados.forEach {
            println(it)
            Listados.pedidosCompletados.add(it)
        }
        println(listaCompletados)
        println("PEDIDOS PENDIENTES")
        val listadoPendientes = list.filter { it.estado ==
Pedido.TipoEstado.EN_PROCESO }.toList()
        listadoPendientes.forEach {
            Listados.pedidosPendientes.add(it)
            println(it)
        }
        println(listadoPendientes)

        println("PRODUCTOS")
        val productos = tennisLabController.findAllProducto().toList()
        productos.forEach {
            println(it)
            Listados.productos.add(it)
        }
        println(listadoPendientes)

        println("LISTADOS SERVICIOS")
        val listaServicios =
tennisLabController.findAllTareaEncordar().toList()
        println(listaServicios)
        listaServicios.forEach {
            val pedido = it.pedido_id?.let { it1 ->
tennisLabController.getPedidoById(it1) }
            if (pedido != null){
                val tarea = TareaDto(
                    pedido.usuario_id,
                    it._id,
                    "Tarea Encordar",
                    pedido.estado.toString()
                )
                println(tarea.toString())
                Listados.servicios.add(tarea)
            }
        }

        val listaaServicios =
```

```
tennisLabController.findAllTareaPersonalizar().toList()
        listaaServicios.forEach {
            val pedido = tennisLabController.getPedidoById(it.pedido_id!!)
            if (pedido != null){
                val tarea = TareaDto(
                    pedido.usuario_id,
                    it._id,
                    "Tarea Personalizar",
                    pedido.estado.toString()
                )
                println(tarea.toString())
                Listados.servicios.add(tarea)
            }
        }

        println(("LISTADO DE ASIGNACIONES ENCORDADORES JSON"))

        val pedidos = tennisLabController.findAllPedidos().toList()
        pedidos.forEach {
            var asignaciones = AsignacionesEncordadores(
                idPedido = it._id,
                encordador_id = it.usuario_id,
                fecha = LocalDate.now().toString(),
            )
            println(asignaciones.toString())
            Listados.asignaciones.add(asignaciones)
        }
    }

    private suspend fun consultarTodosTurnos(){
        println("TODOS LOS TURNOS")
        val list =  tennisLabController.findAllTurnos().toList()
        println(list)

    }

    private suspend fun consultarTodasMaquinas(){
        println("TODOS LAS MAQUINAS DE PERSONALIZAR")
        val list =  tennisLabController.findAllMaquinaPersonalizar().toList()
        println(list)

        println("TODOS LAS MAQUINAS DE ENCORDAR")
        val list1 =  tennisLabController.findAllMaquinaEncordar().toList()
        println(list1)
    }


    suspend fun listadoPedidosCompletados() = withContext(Dispatchers.IO) {
        println("LISTADOS PEDIDOS COMPLETADOS JSON")
        var listado = async {
            FicheroJsonPedidosCompletados().writeFichero(
                System.getProperty("user.dir") + File.separator + "Listas" +
File.separator + "listadosPedidosCompletados.json",
                Listados.pedidosCompletados.toList()
            )
        }
        listado.await()
    }

    suspend fun listadoPedidosPendientes() = withContext(Dispatchers.IO) {
        println("LISTADOS PEDIDOS PENDIENTES JSON")
        var listado = async {
            FicheroJsonPedidosCompletados().writeFichero(
                System.getProperty("user.dir") + File.separator + "Listas" +
File.separator + "listadosPedidosPendientes.json",
```

```
                Listados.pedidosPendientes.toList()
            )
        }
        listado.await()
    }

    suspend fun listadoAsignacionesEncordadores() =
withContext(Dispatchers.IO) {

        println("LISTADOS ASGINACIONES ECORDADORES")
        var listado = async {
            FicheroJsonAsignaciones().writeFichero(
                System.getProperty("user.dir") + File.separator + "Listas" +
File.separator + "listadosAsignacionesEncordadores.json",
                Listados.asignaciones.toList()
            )
        }
        listado.await()
    }


    suspend fun listadoProductos() = withContext(Dispatchers.IO) {
        println("LISTADOS PRODUCTOS")

        var listado = async {
            FicheroJsonProductos().writeFichero(
                System.getProperty("user.dir") + File.separator + "Listas" +
File.separator + "listadosProductos.json",
                Listados.productos.toList()
            )
        }
        listado.await()
    }

    suspend fun listadoServicios() = withContext(Dispatchers.IO) {
        println("LISTADOS SERVICIOS")

        var listado = async {
            FicheroJsonServicios().writeFichero(
                System.getProperty("user.dir") + File.separator + "Listas" +
File.separator + "listadosServicios.json",
                Listados.servicios.toList()
            )
        }
        listado.await()
    }

    suspend fun listasJson() = withContext(Dispatchers.IO) {
        if (!Files.isDirectory(Path.of(System.getProperty("user.dir") +
File.separator + "Listas"))) {
            Files.createDirectories(Path.of(System.getProperty("user.dir") +
File.separator + "Listas"))
        }

        println("")
        var productos = async {
            listadoProductos()
        }

        val servicios = async {
            listadoServicios()
        }
        var asignacionesEncordadores = async {
            listadoAsignacionesEncordadores()
        }
```

```
        var completados = async {
            listadoPedidosCompletados()
        }

        var pendientes = async {
            listadoPedidosPendientes()
        }
        productos.await()
        servicios.await()
        asignacionesEncordadores.await()
        completados.await()
        pendientes.await()
    }
}
```

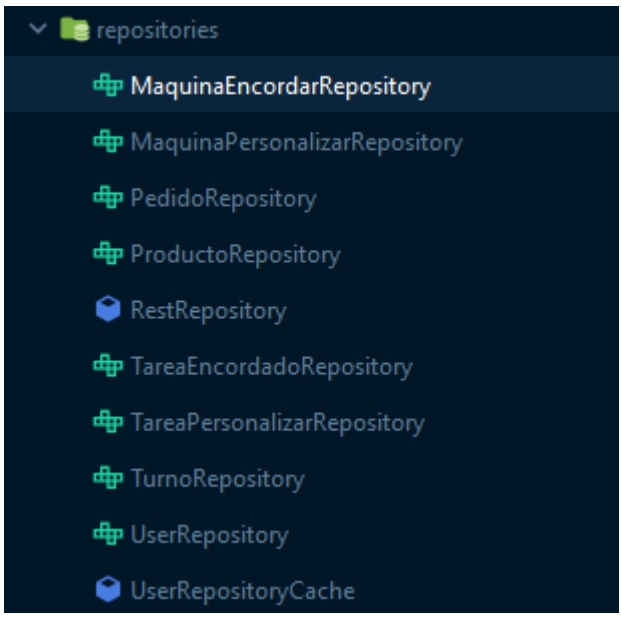
6.2. SpringBoot

Aquí explicamos cada elemento del programa en SpringBoot que se diferencia del proyecto realizado con Ktor.

Models: son cada una de las clases que tenemos creadas introducimos el @Document para identificar el nombre de la clase.

```
@Document("users")
data class User(
    @Id
    val _id: ObjectId = ObjectId.get(),
    val id: String,
    val name: String,
    val username: String,
    val email: String,
    val password: String,
    val tipoUser: TipoUsuario,
    val phone: String,
    val website: String,
) {

    enum class TipoUsuario(tipoUser: String) {
        CLIENTE("CLIENTE"),
        TRABAJADOR("TRABAJADOR"),
        ADMIN("ADMIN");

        companion object {
            fun from(tipoUser: String): TipoUsuario {
                return when (tipoUser.uppercase()) {
                    "CLIENTE" -> CLIENTE
                    "TRABAJADOR" -> TRABAJADOR
                    "ADMIN" -> ADMIN
                    else -> throw IllegalArgumentException("Tipo de usuario
no válido")
                }
            }
        }
    }

    override fun toString(): String {
        return "User(_id='$_id', id='$id', name='$name',
username='$username', email='$email', password='$password',
tipoUser=$tipoUser, phone='$phone', website='$website')"
    }
}
```

Repositories: implementamos los repositories a través de una interfaz, como spring se encarga de implementar los métodos del repositorio.




```
interface MaquinaEncordarRepository: CoroutineCrudRepository<MaquinaEncordar,
ObjectId> {
}
```

Controllers: se encarga de gestionar los repositorios y hacer las operaciones CRUD básicas así como la gestión de memoria caché.

```
@Controller
class TennisLabController
@Autowired constructor(
    private val userRepository: UserRepository,
    private val turnoRepository: TurnoRepository,
    private val tareaEncordadoRepository: TareaEncordadoRepository,
    private val tareaPersonalizarRepository: TareaPersonalizarRepository,
    private val productoRepository: ProductoRepository,
    private val pedidoRepository: PedidoRepository,
    private val maquinaEncordadoRepository: MaquinaEncordarRepository,
    private val maquinaPersonalizarRepository: MaquinaPersonalizarRepository,
    private val userRepositoryCache: UserRepositoryCache,
    private val restRepository: RestRepository,

    ){


    suspend fun getUsersByEmail(email: String, password: String) : User?{
        var user = findByEmail(email)
        if(user == null){
            throw UserControllerException("El usuario no exite")
        }else{
            if (user.password != Contraseñas.encriptar(password)){
                throw UserControllerException("Contraseña incorrecta")
```

```
            }
        }
        return user
    }


    suspend fun findByEmail(email: String): User? {
        val list = userRepository.findAll().toList()
        val enti = list.filter{ it.email == email }.firstOrNull()
        try {
            return enti
        }catch (e: Exception){
            throw UserException("Ha ocurrido un error al obtener el usuario
con email: $email")
        }
    }

    suspend fun insertarUsuario(user: User){
        userRepository.save(user)
    }


    suspend fun actualizarUsuario(user: User) {
        userRepository.save(user)
    }

    suspend fun borrarUsuario(id: ObjectId) {
        userRepository.deleteById(id)
    }

    suspend fun findByUserId(_id: ObjectId) : User? {
        return userRepository.findById(_id)
    }

    suspend fun obtenerTodosLosUsuarios() : Flow<User> {
        return userRepository.findAll()
    }

    suspend fun insertarTurno(turno: Turno){
        turnoRepository.save(turno)
    }

    suspend fun actualizarTurno(turno: Turno) {
        turnoRepository.save(turno)
    }

    suspend fun borrarTurno(id: ObjectId) {
        turnoRepository.deleteById(id)
    }

    suspend fun findByTurnoId(_id: ObjectId) : Turno? {
        return turnoRepository.findById(_id)
    }

    suspend fun obtenerTodosLosTurnos() : Flow<Turno> {
        return turnoRepository.findAll()
    }

    suspend fun insertarTareaEncordado(tareaEncordado: TareaEncordado){
        tareaEncordadoRepository.save(tareaEncordado)
    }

    suspend fun actualizarTareaEncordado(tareaEncordado: TareaEncordado) {
        tareaEncordadoRepository.save(tareaEncordado)
```

```
    }

    suspend fun borrarTareaEncordado(id: ObjectId) {
        tareaEncordadoRepository.deleteById(id)
    }

    suspend fun findByTareaEncordadoId(tareaEncordado: TareaEncordado){
        tareaEncordadoRepository.findById(tareaEncordado._id)
    }

    suspend fun obtenerTodosLosTareaEncordado() : Flow<TareaEncordado> {
        return tareaEncordadoRepository.findAll()
    }

    suspend fun insertarTareaPersonalizar(tareaPersonalizar:
TareaPersonalizacion){
        tareaPersonalizarRepository.save(tareaPersonalizar)
    }

    suspend fun actualizarTareaPersonalizar(tareaPersonalizar:
TareaPersonalizacion) {
        tareaPersonalizarRepository.save(tareaPersonalizar)
    }

    suspend fun borrarTareaPersonalizar(id: ObjectId) {
        tareaPersonalizarRepository.deleteById(id)
    }

    suspend fun findByTareaPersonalizarId(tareaPersonalizar:
TareaPersonalizacion){
        tareaPersonalizarRepository.findById(tareaPersonalizar._id)
    }

    suspend fun obtenerTodosLosTareaPersonalizar() :
Flow<TareaPersonalizacion> {
       return tareaPersonalizarRepository.findAll()
    }

    suspend fun insertarProducto(producto: Producto){
        productoRepository.save(producto)
    }

    suspend fun actualizarProducto(producto: Producto) {
        productoRepository.save(producto)
    }

    suspend fun borrarProducto(id: ObjectId) {
        productoRepository.deleteById(id)
    }

    suspend fun findByProductoId(producto: Producto){
        productoRepository.findById(producto._id)
    }

    suspend fun obtenerTodosLosProducto() : Flow<Producto> {
        return productoRepository.findAll()
    }

    suspend fun insertarPedido(pedido: Pedido){
        pedidoRepository.save(pedido)
    }

    suspend fun actualizarPedido(pedido: Pedido) {
        pedidoRepository.save(pedido)
    }
```

```
    suspend fun borrarPedido(id: ObjectId) {
        pedidoRepository.deleteById(id)
    }

    suspend fun findByPedidoId(_id: ObjectId) : Pedido?{
        return pedidoRepository.findById(_id)
    }

    suspend fun obtenerTodosLosPedido(): Flow<Pedido> {
       return pedidoRepository.findAll()
    }

    suspend fun insertarMaquinaEncordado(maquinaEncordado: MaquinaEncordar){
        maquinaEncordadoRepository.save(maquinaEncordado)
    }

    suspend fun actualizarMaquinaEncordado(maquinaEncordado: MaquinaEncordar)
{
        maquinaEncordadoRepository.save(maquinaEncordado)
    }

    suspend fun borrarMaquinaEncordado(id: ObjectId) {
        maquinaEncordadoRepository.deleteById(id)
    }

    suspend fun findByMaquinaEncordadoId(_id: ObjectId) : MaquinaEncordar?{
        return maquinaEncordadoRepository.findById(_id)
    }

    suspend fun obtenerTodosLosMaquinaEncordado() : Flow<MaquinaEncordar> {
        return maquinaEncordadoRepository.findAll()
    }

    suspend fun insertarMaquinaPersonalizar(maquinaPersonalizar:
MaquinaPersonalizar){
        maquinaPersonalizarRepository.save(maquinaPersonalizar)
    }

    suspend fun actualizarMaquinaPersonalizar(maquinaPersonalizar:
MaquinaPersonalizar) {
        maquinaPersonalizarRepository.save(maquinaPersonalizar)
    }

    suspend fun borrarMaquinaPersonalizar(id: ObjectId) {
        maquinaPersonalizarRepository.deleteById(id)
    }

    suspend fun findByMaquinaPersonalizarId(_id: ObjectId) :
MaquinaPersonalizar?{
       return maquinaPersonalizarRepository.findById(_id)
    }

    suspend fun obtenerTodosLosMaquinaPersonalizar() :
Flow<MaquinaPersonalizar> {
       return maquinaPersonalizarRepository.findAll()
    }



    //-----------------------------------------------CACHE-------------------
----------------------
    suspend fun insertarUsuarioCache(user: User){
        userRepositoryCache.save(user)
    }
```

```
    suspend fun actualizarUsuarioCache(user: User) {
        userRepositoryCache.update(user)
    }

    suspend fun borrarUsuarioCache(id: ObjectId) {
        userRepositoryCache.delete(id)
    }

    suspend fun findByUserIdCache(id: String) {
        userRepositoryCache.findById(id)
    }

    suspend fun obtenerTodosLosUsuariosCache() : Flow<User> {
        return userRepositoryCache.findAll()
    }

    //-----------------------------------------------------REST--------------------
--------------------------------

    suspend fun getAllApiUsers() : Flow<User> {
        val flujo = mutableListOf<User>()
        val list =  restRepository.findAll().toList()
        list.forEach {
            val user = toModel(it)
            flujo.add(user)

        }
        return flujo.asFlow()
    }

    suspend fun getAllApiTareas() : Flow<TareaDto> {
        return restRepository.findAllTareas()

    }


    suspend fun insertarTareaRest(tarea: TareaDto){
        restRepository.create(tarea)
    }


    suspend fun actualizarTareaRest(tarea: TareaDto) {
        restRepository.update(tarea)
    }

    suspend fun deleteTareaApi(id: String) {
        restRepository.delete(id)
    }

}
```

7. Referencias

- https://ktor.io/
- https://spring.io/
- https://insert-koin.io/
- https://litote.org/kmongo/
- https://www.sqlite.org/index.html
- https://mockk.io/
- https://www.baeldung.com/mockito-series
- https://www.baeldung.com/kotlin/mockito
- https://www.baeldung.com/kotlin/spring-boot-kotlin
- https://docs.docker.com/compose/
- https://kotlinlang.org/docs/serialization.html

8. Librerías utilizadas

- Ktor
  https://ktor.io/
- Spring
  https://spring.io/
- Koin
  https://insert-koin.io/
- KMongo
  https://litote.org/kmongo/
- SQLite
  https://www.sqlite.org/index.html
- MockK
  https://mockk.io/

9. Enlace al proyecto

Enlace → https://github.com/jorgesanchez3212/AD-P03-TennisLab

10. Enlace al video

Enlace →

Don Jorge Sánchez Berrocoso

Don Daniel Rodriguez Fernandez

Don Alfredo Maldonado Pertuz

Damos por finalizada y entregada la práctica 03 de Acceso a datos en,

Madrid a 05 de Febrero de 2023.