Práctica AD-04

Daniel Castellote y Javier González

2ºDAM

Introducción

Vamos a presentar nuestra práctica de Acceso a Datos donde realizaremos una implantación completa de una BBDD NoSQL en nuestro caso utilizaremos MongoDB como motor de base de datos y mongo-express para representar el resultado.

Además utilizaremos un mapeo a Objetos con Documentos usando Hibernate OGM y JPA para controlar todas las relaciones y restricciones entre nuestras entidades.

El enunciado de la práctica nos pide una simulación de una empresa informática, para dicha empresa nos encontramos entidades como departamento, programador…

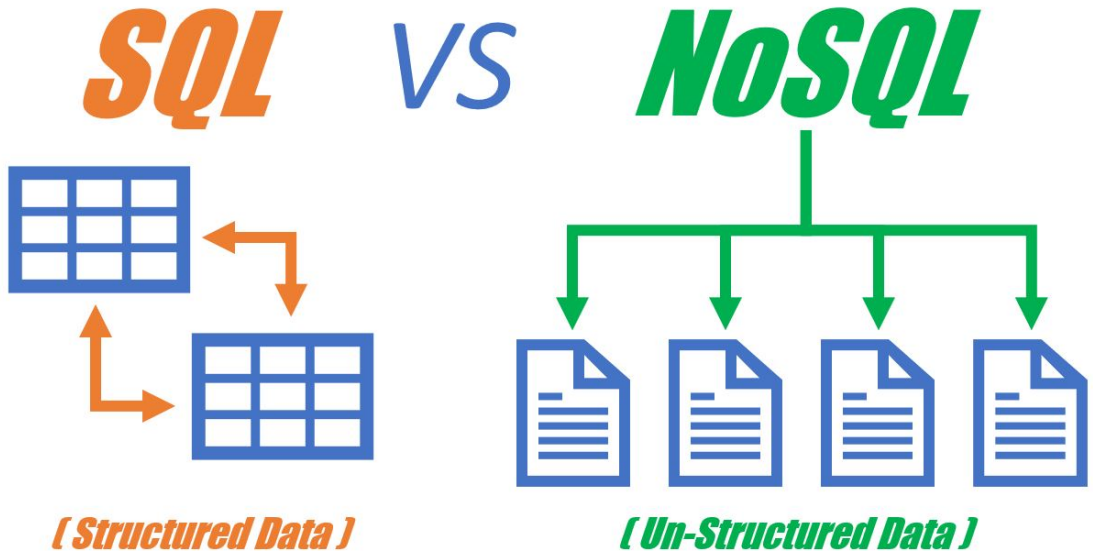
Para todas aquellas entidades plantearemos relaciones entre ellas, lo más importante para ello y el objetivo de la práctica es controlar la navegabilidad y cardinalidad, donde tendremos que romper con la bidireccionalidad con las relaciones muchos a muchos y que nuestra base de datos sea capaz de realizar las consultas básicas CRUD sin fallos ni errores a la hora de cargar y consultar nuestros datos.

Primeramente crearemos un esquema de las entidades y las relaciones mostradas en un diagrama de clases estructurado. Después pasaremos a programar paso a paso nuestra BBDD con una arquitectura de 3 niveles y por último realizaremos test unitarios probando y comprobando nuestro código y consultas y programaremos la salida de nuestras consultas por archivos Json.

Diferencias y ventajas de NoSQL y SQL

En función del propósito de aplicación, NoSQL puede ofrecer ciertas ventajas frente a las clásicas bases de datos relacionales. En las situaciones en las que los sistemas SQL se sobrecargan, por ejemplo, al manejar Big Data, las bases de datos NoSQL permiten leer y procesar grandes volúmenes de datos en tiempos récord.

Las bases de datos NoSQL se alejan de los esquemas rígidos de los sistemas SQL y tienden hacia modelos más flexibles que son ideales para el procesamiento de grandes volúmenes de datos. Gracias al almacenamiento en clústeres de hardware distribuidos están menos expuestas a las interferencias y errores.



Bases de datos orientadas a documentos

En las bases de datos NoSQL orientadas a documentos, los datos se almacenan directamente en documentos de diferentes longitudes. No hace falta que los datos estén estructurados. Las bases de datos NoSQL orientadas a documentos son especialmente adecuadas para sistemas de

gestión de contenidos y blogs. Usamos JSON como formato de archivo, ya que permite el intercambio de datos más rápido.

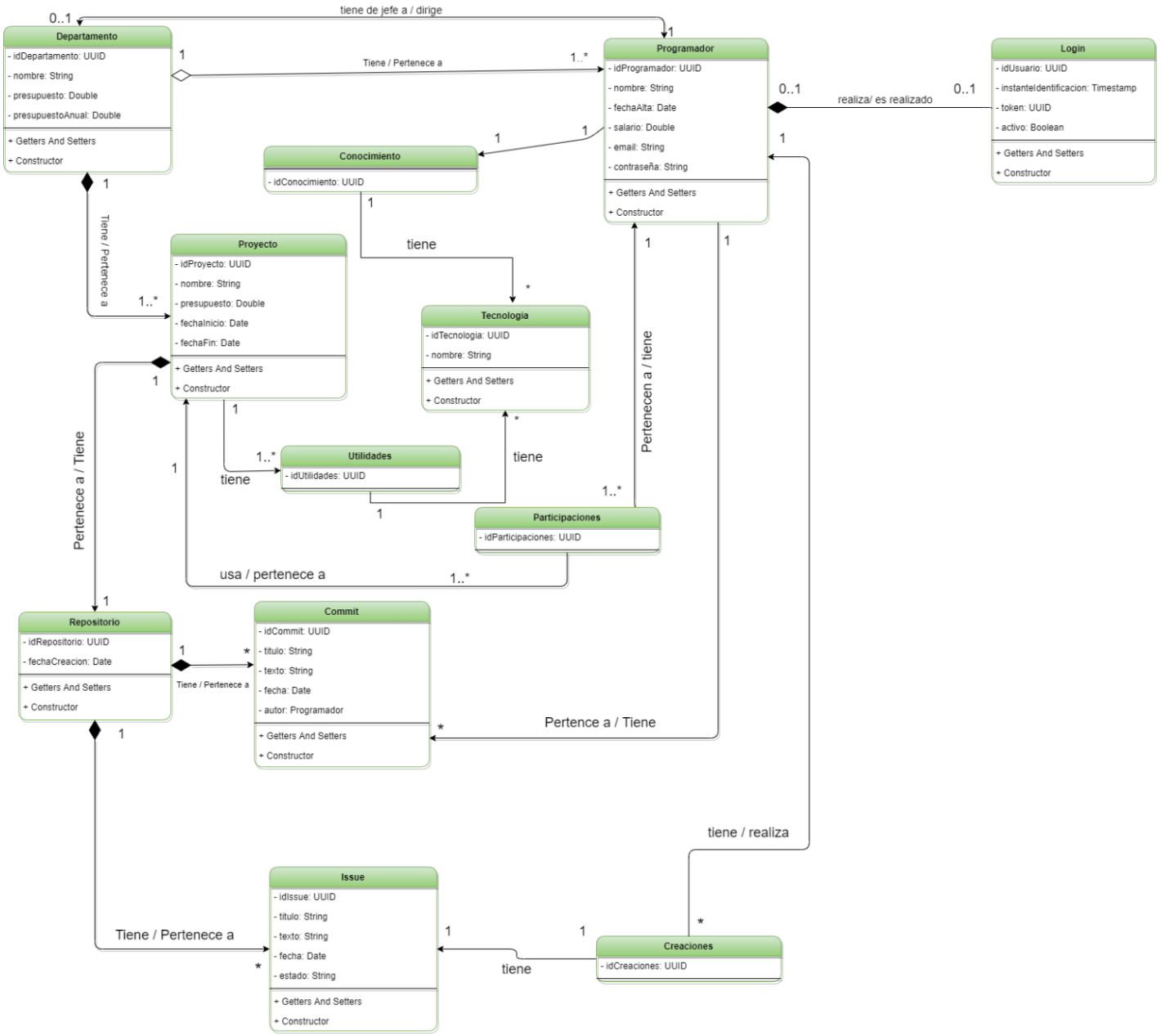
Clases con @Embeddable y @Embedded

Las bases de datos orientadas a documentos facilitan y dan la capacidad de usar documentos embebidos. La ventaja de usar objetos embebidos es que son reusables y se pueden mapear dos entidades a la misma tabla de nuestra base de datos.

En nuestra práctica las clases y entidades se crean según se indica en la clase “Data”. En caso de querer embeber debemos marcar la clase con que embebe con @Embeddable para embeber otra que le asignaremos @Embedded. Se mostrarán todas las entidades en la misma tabla como hemos dicho.

Los casos de uso de esta anotación en nuestra práctica se darían en Tecnologias->Programador, Issue->Repositorio y para un documento que es embebido y embebe sería un ejemplo el Programador->Departamento.

Diagrama



Para estudiar nuestro diagrama nos fijaremos sobre todo en la cardinalidad , hemos controlado nuestras entidades para que las cardinalidades fueran sencillas y fácilmente controladas.

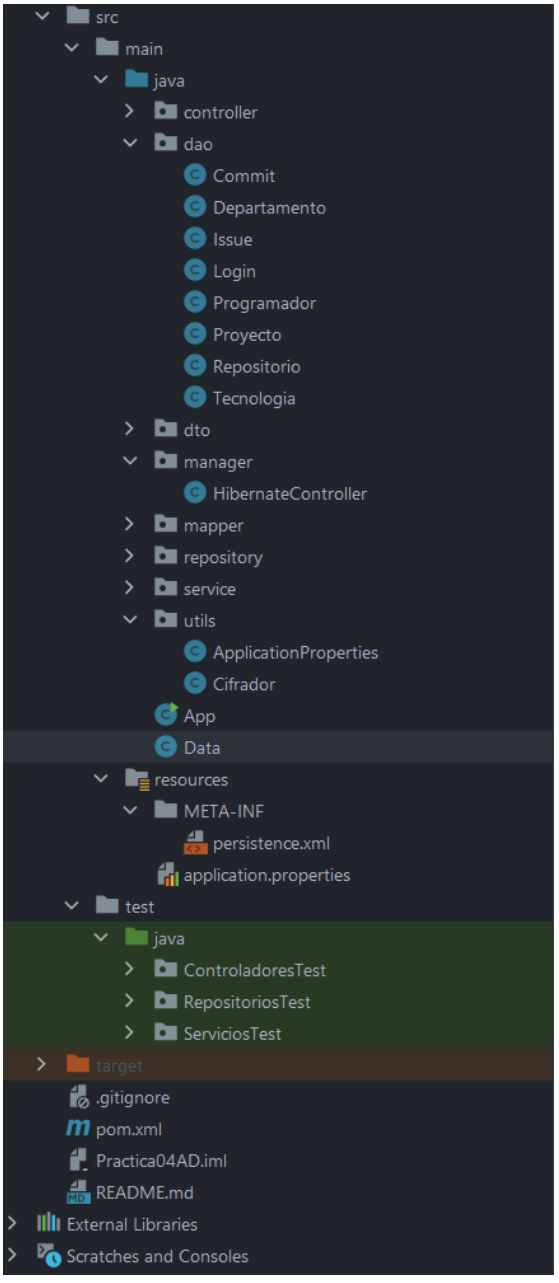
En algunos casos nos hemos encontrado que entre dos entidades había cardinalidad muchos a muchos, para ello hemos creado clases intermedias

por ejemplo entre Programadores y Repositorios donde hay muchos programadores asignados a un repositorio y un programador puede tener asignados varios repositorios, ante esta situación creamos la entidad Tecnología que será la que romperá este muchos a muchos.

Por último hemos indicado varias agrupaciones entre entidades como la composición en algunos casos ya que existen entidades que dependen de la otra y en caso de ser eliminadas no podrían existir sin la otra.

Ejemplo: Un repositorio no puede existir si no está asignado a un proyecto.

Después en el código con estas composiciones, tendremos la opción de realizar entidades embebidas dentro de otras que significaran que introduciremos la entidad dependiente en la otra pasando a formar parte de la clase padre.



Estructura de nuestro proyecto a tres niveles con utilización de JPA e Hibernate

Relaciones entre Entidades:

Explicaremos cada una de la relación que hemos decidido hacer entre las entidades dadas y como han sido implementadas en el proyecto.

Programador:

La entidad programador para nosotros ha sido la entidad más central a la hora de tener relación con otras entidades. Por ello es la que más dificultad tiene de manejar navegabilidad.

Programador-Departamento:

Los departamentos están considerados por tener un conjunto o lista de programadores en ella, es por eso que a partir de ahí la relación es claramente uno a muchos ya que para un programador entendemos que solo podrá estar en único departamento trabajando. Además hemos considerado añadir una asociación ya que las dos entidades dependen del otro es decir un Programador no podría tener acceso a los Proyectos si no está asociado a un Departamento que lo dirija y en cuanto al Departamento no existiría sin Programadores en él y es necesario para la elección de un jefe. A la hora de la elección de un jefe de departamento, el programador será o no jefe y solo existirá uno propio por departamento.

Programador-Login:

Los programadores a la hora de trabajar necesitaran un Login que registre las entradas de estos y si están activos. Es por eso que esta relación será un OneToOne para un solo programador existirá un login único para el

mismo programador, lo mismo para los login solo tendrán información de un programador. Además hemos considerado añadir una composición clara para el Login ya que esta entidad depende de la existencia de un programador para llevarse a cabo, a la hora de la implementación es un claro ejemplo de posibilidad de embeber una entidad hija en una padre en este caso en la de Programadores. Por último gracias al Login el programador deberá proporcionar un email y una contraseña que será Cifrada en nuestro caso por SHA256, estos campos mandados al login devolverán al programador un token de conexión.

Programador-Proyecto:

Los programadores tendrán asociados proyectos, estos proyectos entendemos que pertenecen a un departamento pero podremos indicar qué programador está asociado y cuantos programadores y cuales están formando parte de este. Lo primero de todo y la mayor dificultad encontrada es que estamos ante un caso de relación ManyToMany es decir un programador puede pertenecer a varios proyectos y en los proyectos existen varios programadores al cargo, entonces para afrontar esta relación crearemos una entidad o campo intermedio en la base de datos, en nuestro caso hemos creado un campo tecnologías que partirá esta relación. Gracias a este campo nuevo podremos almacenar los programadores con una tecnología asociada y a su vez una tecnología asociada al proyecto así podremos navegar entre la entidad Programador y Proyecto. Por último se asignará a un programador que sea jefe de proyecto, este no trabajara en el proyecto pero sí podrá dirigirlo.

Programador-Commit:

Los programadores durante el proyecto para trabajar crearán commits sobre los repositorios, entonces para controlar navegabilidad podremos acceder a través de los commits para saber su autor o programador asociado y lo mismo para los programadores donde podremos obtener la lista de commits que han realizado. Esta relación consta de un OneToMany,

donde nos encontraremos que para un programador existen varios commits y que un commit solo puede ser creado por un programador.

Programador-Issues:

En cuanto a las Issues están y son creadas en un repositorio, pero existe dificultad para crear la relación con los programadores. Está relacion se trata de un ManyToMany ya que para un programador existen varias issues en el repositorio pero a su vez las issues estan asociadas y estan creadas para varios programadores, para cortar este ManyToMany crearemos un campo intermedio este campo lo llamaremos Creaciones por lo cual podremos acceder desde un programador que realizará una creación y la propia issue tendrá un identificador para sacar los programadores asociados.

Proyecto:

Proyecto-Departamento:

Para los proyectos podremos acceder a ellos a través del departamento al que está destinado. La relación es un OneToMany varios proyectos para un departamento y un proyecto solo está destinado para un departamento. Además hemos implementado una composición para los proyectos donde estos no pueden ser creados en la bbdd si no existe un departamento al que asociarlo.

Proyecto-Repositorio:

Existen repositorios donde destinar los proyectos , para un proyecto existen varios repositorios donde repartir el proyecto y a la vez un repositorio solo está asociado a un proyecto, por último indicamos una composición entre las dos entidades ya que un repositorio no será insertado en la bbdd si no existe un proyecto en activo.

Repositorio:

Repositorio-Commit y Repositorio-Issue:

Para las commits y las issues creadas en los repositorios, habrá relación de composición OneToMany donde en un repositorio habrán varios commits y varias issues, podremos consultar la lista de commits e issues y toda su información, por último los commits y las issues no se podrán consultar si no están asociados a un repositorio.

Ejecución del proyecto

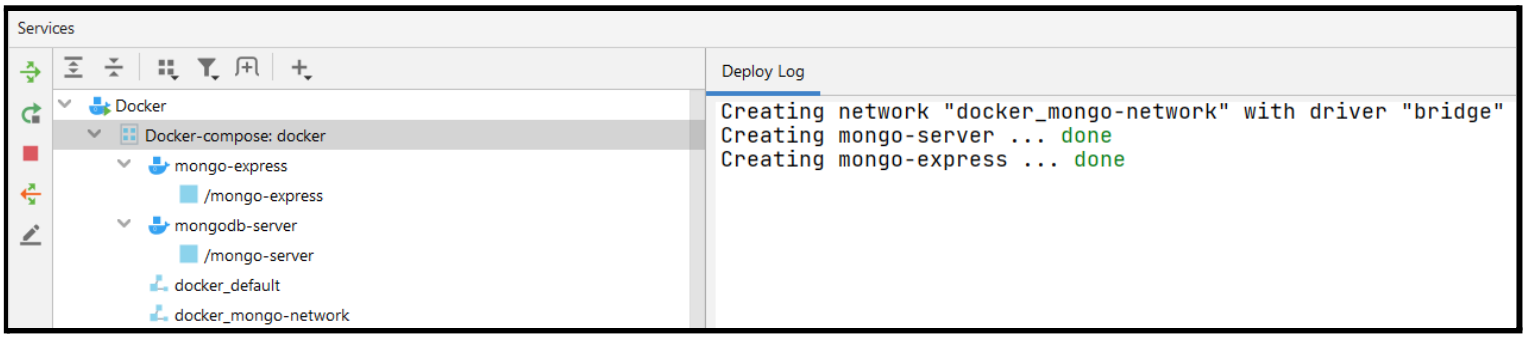
Docker

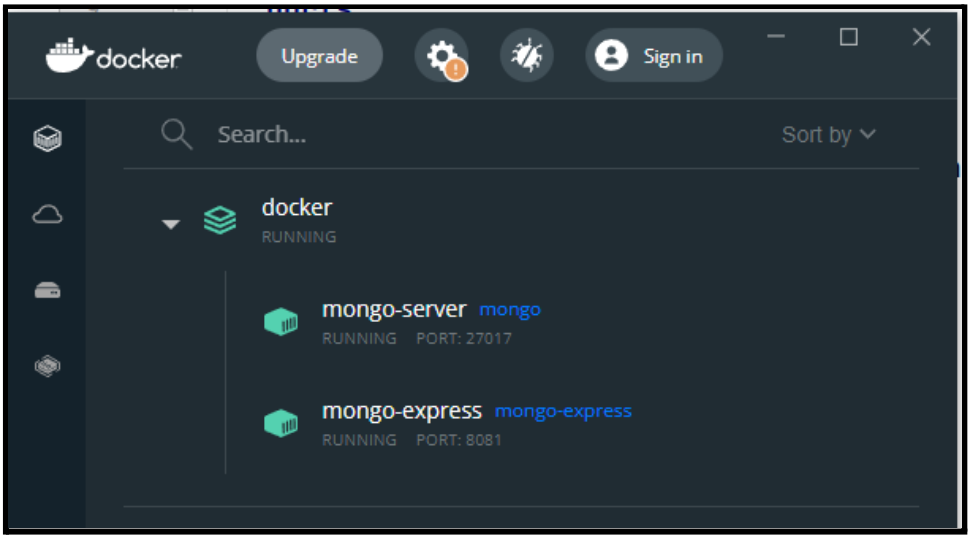
En primer lugar debemos tener en cuenta que necesitamos disponer de un espacio de almacenamiento para alojar nuestro proyecto. Para ello utilizaremos mongodb como motor y mongo express para la visualización de datos. Todo esto lo conseguiremos fácilmente lanzando nuestro docker-compose.yml que se encuentra en la carpeta "docker" del proyecto.

Si nos paramos a analizarlo vemos que se lanzan dos imágenes. Una de mongo y otra de mongo-express en dos contenedores. Ambos contienen unos puertos asignados y uno de ellos expuesto para poder conectarnos. Disponen de un usuario y contraseña de administrador para poder tener todos los permisos en la base. Además de ello le asignamos un volumen y a mongo express le pedimos que se reinicie cuando se pare.

El tipo de conexión en la network será de tipo bridge.

Nuestro docker estará lanzado sin problema y esperando a que se inserten nuevas bases de datos con colecciones en ella.

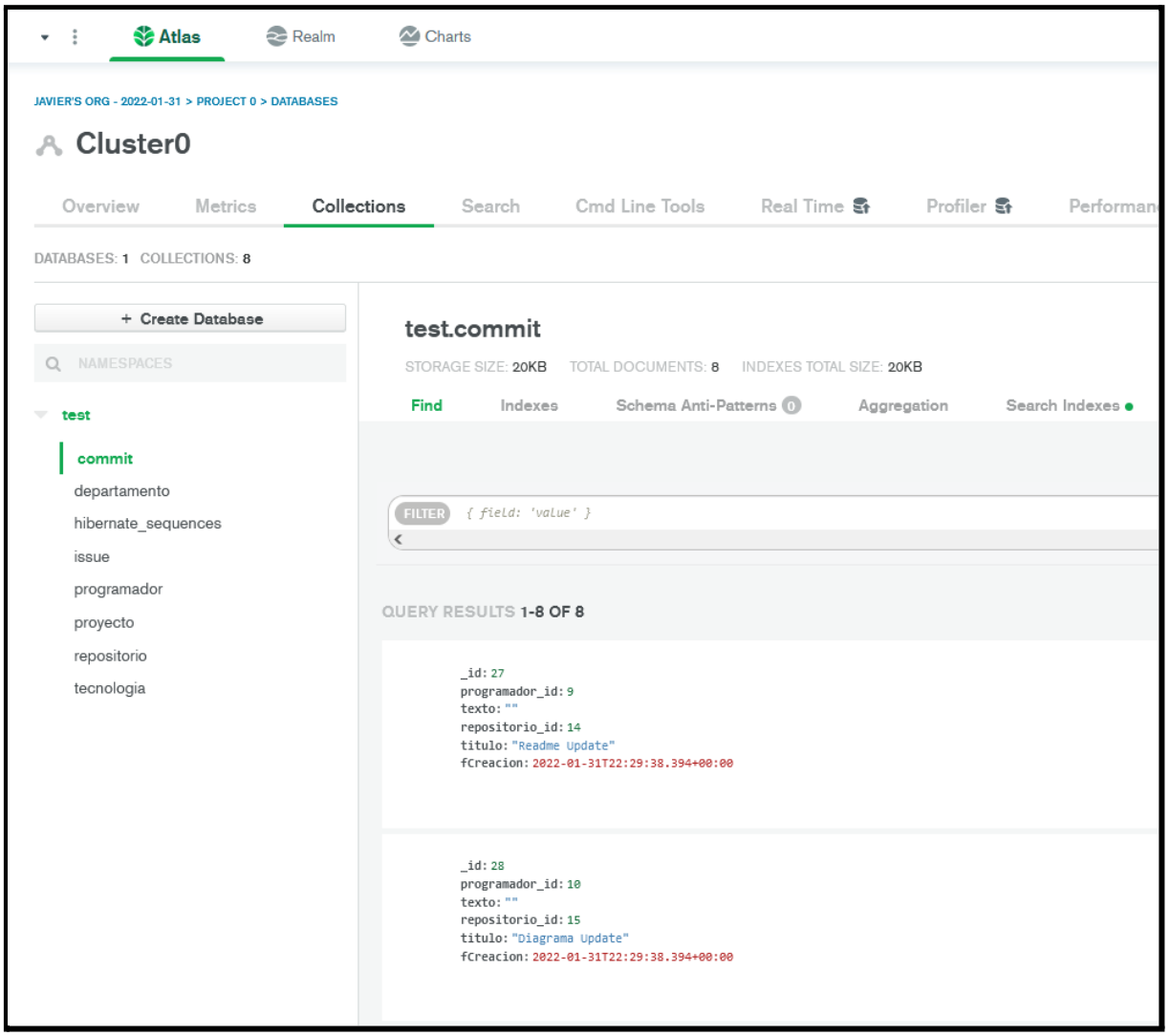




Mongo Atlass

Otra opción que hemos aprendido en estas últimas semanas es el uso de mongo atlass. El funcionamiento parte de la creación de un cluster personal hosteado por AWS, Azure o Google Cloud. Una vez creado podemos crear bases de datos en este espacio. Nosotros hemos insertado nuestros datos de la misma manera que hacíamos en docker pero insertando el controlador que nos proporciona el propio Mongo Atlass. En él cambia el link de conexión y tenemos que asignarle nuestro cluster en el host del fichero de persistencia para hacerlo del mismo modo.

La principal diferencia que tenemos es que de esta manera los datos se quedan subidos y cualquiera que disponga de un usuario de acceso puede consultarlo además de estar su IP personal en la lista de IP permitidas.



Dependencias y Persistence

Hemos implementado varias dependencias al proyecto las más básicas y necesarias serán:

Hibernate-ogm para mongo, librerías de JPA y el driver de Mongodb.

Dependencias extras:

Mockito para los test, lombok para agilizar el proyecto y Gson para pasar el resultado en formato Json.

Para el persistence esta ha sido nuestra configuración:

```
<persistence-unit name="default" transaction-type="JTA">
    <description>implantacion de una BBDD NOSQL con Hibernate JPA</description>

    <provider>org.hibernate.ogm.jpa.HibernateOgmPersistence</provider>

    <class>dao.Departamento</class>
    <class>dao.Programador</class>
    <class>dao.Proyecto</class>
    <class>dao.Repositorio</class>
    <class>dao.Commit</class>
    <class>dao.Issue</class>
    <class>dao.Login</class>
    <class>dao.Tecnologia</class>

    <properties>
        <property name="hibernate.ogm.datastore.provider" value="mongodb" />
        <property name="hibernate.ogm.datastore.database" value="data" />
        <property name="hibernate.ogm.datastore.host" value="127.0.0.1" />
        <property name="hibernate.ogm.datastore.port" value="27017" />
        <property name="hibernate.ogm.datastore.username" value="mongoadmin" />
        <property name="hibernate.ogm.datastore.password" value="mongopass" />
        <property name="hibernate.ogm.datastore.create_database" value="true" />

    </properties>
```

Data

Sigue una estructura de patrón “Fachada” en la que insertamos datos utilizando el controlador de hibernate controlando las transacciones de datos en la base de mongo. Además reseteamos la misma mediante el método removeData() cada vez que intentemos conectarnos a ella. En este mismo método insertamos el string de conexión a la Base mongo.

Si preferimos acceder a mongo atlass tenemos comentado el driver para ver como lo hicimos. Importante cambiar el campo “host” en persistance.

Dao

Clases POJO de las entidades y tablas principales como intermedias.Commit, Conocimiento, Creaciones, Departamento, Issue, Participaciones, Programador, Proyecto, Repositorio, Tecnología, Utilidades. En este modelo encontraremos anotaciones de JPA las principales y más utilizadas en la práctica serán @Entity para indicar la entidad de nuestra BBDD y cómo será representada, con @Table podremos poner nombre a la tabla entidad, por último podremos indicar consultas por defectos con la anotación @NameQuery estas consultas serán desarrolladas en los repositorios. Para los atributos, la entidad nos pedirá indicar un ID con la anotación @ID además indicamos como se generara esta id con @GenerateValue en nuestro caso secuencialmente para cada entidad. Por último, las relaciones según hayamos indicado en el diagrama pondremos en caso de la cardinalidad uno a muchos pondremos @OneToMany aparte añadiremos restricciones importantes como el tipo de Cascada y que atributo será mapeado. Por otro lado al otro lado de la relación pondremos la anotación @ManyToOne aquí indicaremos el atributo mapeado anteriormente y la columna o campo que se unirá con la otra entidad relacionada con la anotación @JoinColumn. Por último en el método ToString tendremos cuidado al imprimir los datos por pantalla porque podría darnos error de StackOverflow, por eso indicaremos para imprimir solo los atributos no relacionados con otras entidades.

Service

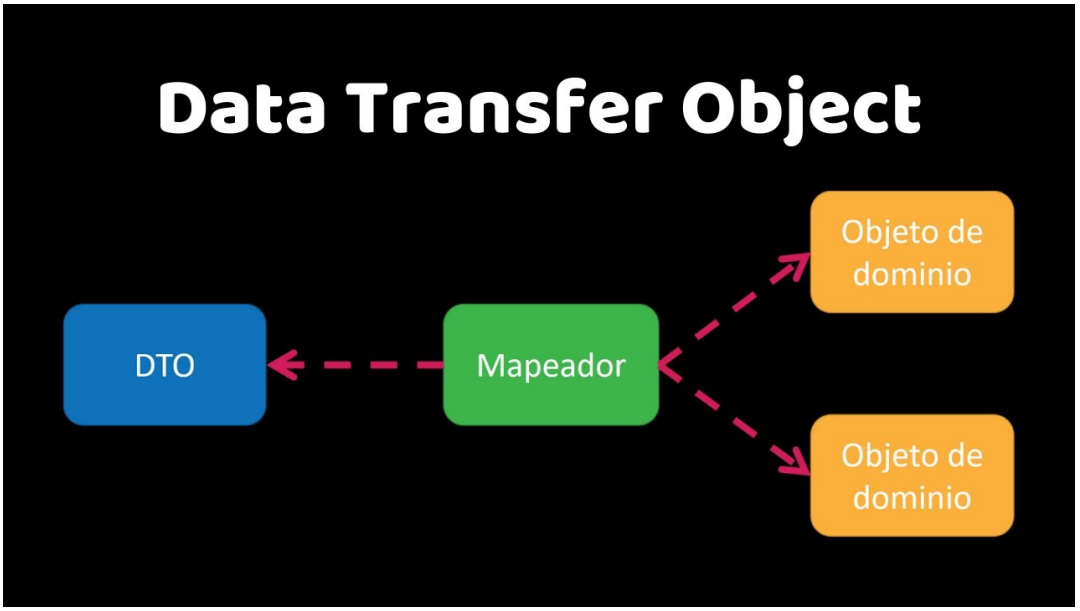
Cuenta con una clase Base Service de la que heredan todas las demás clases (clase de cada entidad) como commits, programadores, departamentos… Utiliza los métodos mapper para los dto que le pasamos por parámetros.

Repository

El repositorio se comunica con la base de datos. Partiendo de CrudRepository como interfaz, el resto de clases creadas para cada entidad y tablas intermedias debe cumplir con los métodos implementados. Nuevamente se crea una clase para cada una llamada “Repo _____” (commit, departamento, programador…). Nuestro repositorio almacena las consultas básicas CRUD y podremos añadir más consultas como las del enunciado según la clase relacionada.

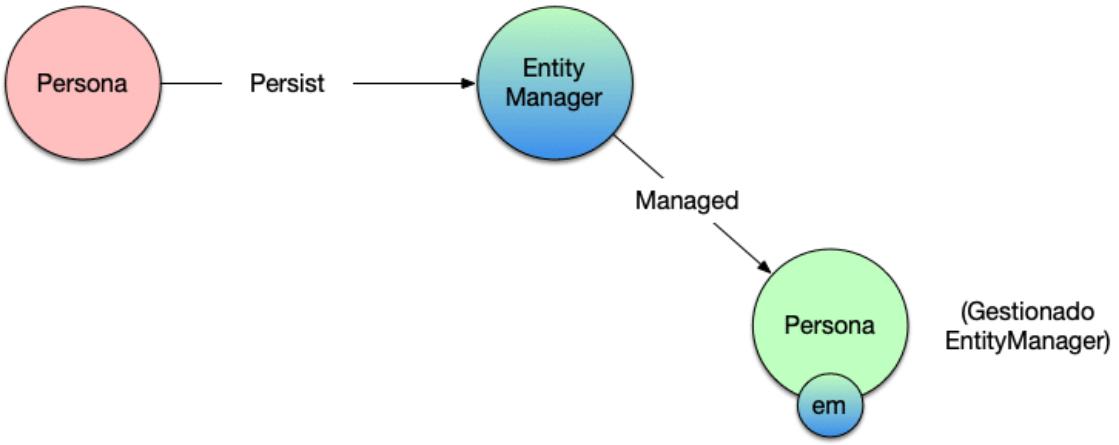
DTO

Cuenta con una clase para cada documento y trata de transmitir la información recibida por una entrada dada a una salida y así crear un formato organizado de datos gracias a su patrón. Cada clase implementada “_____ DTO” se encarga de la información del tipo que indica su nombre.



Hibernate Controller

En la clase Hibernate Controller manejamos las entidades y transacciones.



Hacemos uso de los métodos implementados en “Data” a la hora de crear los documentos con los que trabajamos

Controller

Clase encargada de controlar el servicio y sus métodos. Introducimos los datos obtenidos en listas en caso de que existan y en los métodos indicados transformamos los objetos en JSON utilizando Gson de apoyo.

Útiles

Tendremos un paquete útiles con las clases para las application properties, esta clase leerá toda la información creada en el fichero .properties, al iniciar la app instanciamos la clase para presentar los nombres de los creadores y el título. Además tendremos la clase Cifrador, esta clase cifrará las contraseñas de los programadores al hacer login.

Test

La parte de los test lo hemos repartido entre:

Test a los repositorios que lo haremos con la librería de test de java Junit 5 estos tests se comprobará fácilmente con la salida esperada de cada una de las consultas.

Test a los controladores y servicios con Mockito, creará un mock del resultado esperado y así podremos comprobar el código

App - Data

Una vez lanzado docker o bien asignado el host además del controlador de mongo atlass podemos lanzar el proyecto.

Primero debemos revisar el fichero de persistencia para ver las propiedades de hibernate en la base de datos.

Ejecuta el inicio de la base de datos y los métodos encargados de las consultas y operaciones CRUD de cada clase.