Tema 03 Bases de Datos Relacionales

Acceso a Datos

Sergio Pérez Sanz y Federico Toledo Baeza
2 DAM 2021-2022

Contenido

Departamento 4
Project 5
Programador 6
Repository 7
Commit 7
BossHistory 8
ProjectAssignment 8
IssueAssignment 8
Clases principales 9
Controladores 10
Servicios 10
  Servicios JDBC 10
  Services JPA 11
Repositorios 12
  Repositorios en JDBC 12
  Respositorios en JPA 12
Mappers 12
Conexión a la base de datos: JDBC 13
Conexión a la base de datos: JPA Hibernate 13
Docker 13
Utils 13

Persistencia en Java mediante JDBC y JPA Hibernate

Introducción

En esta práctica tenemos como objetivo aprender a persistir datos de un programa, en Java, a una base de datos relacional. Para ello implementamos el modelo “MVC” (Model View Controller) de dos formas diferentes:

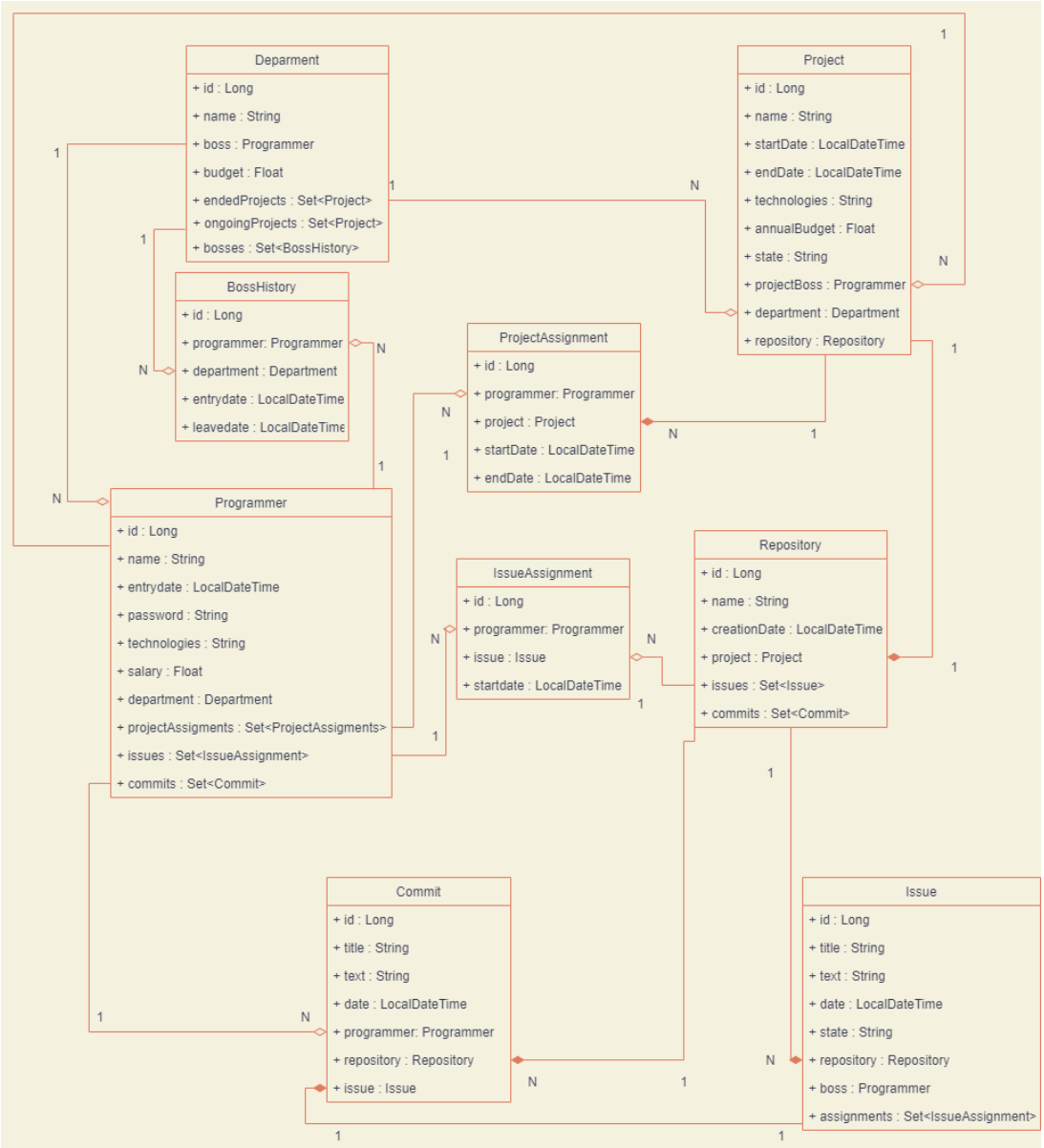
- Realizando consultas SQL nativas por medio de las librerías de JDBC.
- Empleando la implementación de Hibernate de JPA.

JDBC nos permite un control muy profundo de la persistencia, pues en resumen nos obliga a definir en la propia aplicación las diferentes consultas SQL que queremos ejecutar, mientras que Hibernate JPA nos facilita las consultas básicas, y nos permite concentrarnos en las relaciones complejas entre los datos.

En este documento presentamos nuestra implementación de ambas soluciones para una base de datos relacional MariaDB que modela los datos de una empresa: DataAccessSL.

Entidades

La base de datos que empleamos para esta practica se modela con este diagrama de clases:

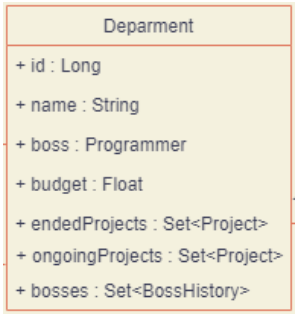


Antes de explicar las relaciones, es necesario algo de contexto sobre los datos a modelar. En DataAccessSL existen departamentos. Dichos departamentos están compuestos por programadores y por proyectos. Los programadores trabajan en proyectos. Los repositorios tienen issues, problemas creados por jefes del proyecto que tienen que resolver los programadores mediante commits. Un programador hace un commit que cierra una issue, la cual puede estar asignada a más de un programador.

Toda la base de datos está resuelta utilizando ***claves autonuméricas*** de tipo long (BIGINT en MariaDB). Las colecciones de datos en Java están implementadas mediante ***la interfaz Set***, pues en ningún momento deseamos tener información duplicada dentro de una lista. Todas las fechas están implementadas en LocalDateTime en java (datetime en MariaDB). En caso de no especificarse lo contrario, todos los campos son por defecto no nullables.

A continuación, se explicará cada una de las entidades del problema desde el punto de vista del DAO que lo representa.

Departamento



Un departamento se modela con un identificador autonumérico, un nombre, un jefe, un presupuesto, una lista de proyectos activos, una lista de proyectos terminados y una lista de jefes.

El jefe debe ser un programador que no participa en proyectos y que pertenezca al departamento.

Jefe es una relación 1 – 1 con programador, pues un Departamento es dirigido por un Programador y un programador solo puede ser jefe de un Departamento. Hemos decidido propagar la clave de programador a departamento, puesto que en una vista de departamento saber los datos del jefe es muy interesante, mientras que un 95% de los programadores no dirigen ningún departamento.

Proyectos es una relación 1 – N. Un departamento tiene varios proyectos (esta relación posteriormente se divide en cuales están terminados y cuales están activos), mientras que un proyecto solo pertenece a un departamento.

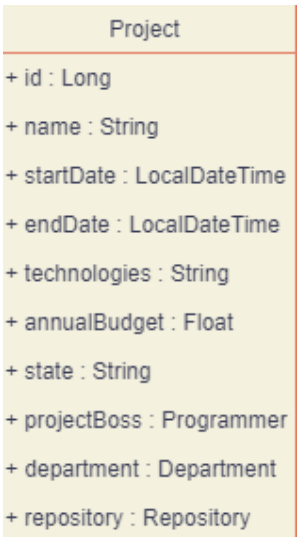
La separación de proyectos por estado se realiza mediante una propiedad de proyecto: estado, de la cual se hablará posteriormente.

La lista histórica de jefes es producto de partir una relación N – N entre programadores y departamento que ha sido dividida a dos relaciones 1 – N. Esta decisión se explica en la entidad BossHistory. Un departamento tiene varios históricos de jefes, uno por cada programador que ha dirigido el departamento, mientras que un histórico de jefe pertenece solo a un departamento.

El DAO ha sido diseñado con los campos projects, bosses y boss transient, véase, que no desean ser serializados. Esta decisión ha sido tomada para evitar recursividad en los datos.

En la base de datos, departamento solo contiene la clave ajena del programador jefe, las demás claves se propagan a la parte de muchos de las relaciones.

Project



Un proyecto consta de un identificador autonumérico, un nombre, una fecha de comienzo y una fecha de fin, una lista de tecnologías, un presupuesto anual, un jefe de proyecto, un departamento al que pertenece y un repositorio.

La fecha de fin del proyecto puede ser nula, indicando así que el proyecto aún no ha acabado.

La cadena de texto estado tiene dos posibles valores: “ended” o “active”.

La cadena de tecnologías se representa en la base de datos como valores concatenados mediante el separador ‘;’, mientras que en Java se representa con un Set<Tecnologías>, siendo Tecnologías un enum con los valores posibles para dichas cadenas. La transformación a colección se realiza mediante la clase TechnologiesParser del paquete utils.

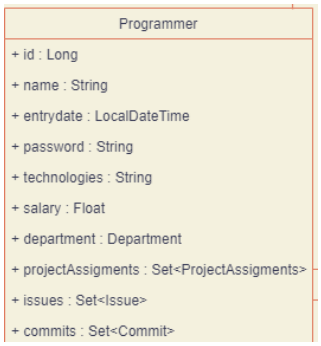
Proyecto presenta una relación N – 1 con departamento, donde un departamento puede tener varios proyectos mientras que un proyecto solo pertenece a un departamento. Es por esto que la clave de departamento se haya en la entidad proyecto.

Proyecto se relaciona con Programador en una relación 1 – 1 donde un programador es jefe de proyecto y un proyecto solo es dirigido por un programador. Al igual que en departamento, la clave se propaga a proyecto.

Proyecto se relaciona con Repositorio en una relación 1 – 1, dado que un repositorio solo pertenece a un proyecto y un proyecto solo tiene un repositorio.

Los campos que representan relaciones han sido declarados con transient en los DAO para evitar su serialización, causante de recursividad en las llamadas SQL.

Programador



Un programador está modelado con una clave autonumérica, un nombre, una fecha de entrada en la empresa, una contraseña, una cadena de tecnologías, un salario, el departamento al que pertenece, los proyectos en los que está asignado, una lista de issues asignadas y una lista de commits realizados.

El campo contraseña esta cifrado con el algoritmo SHA256 empleando la clase Cifrate del paquete utils.

La cadena de tecnologías se comporta igual que en la entidad Proyecto, es una concatenación de valores del enum Technologies por medio del carácter ‘;’.

Programador se relaciona con departamento en una relación 1 – N donde un programador solo pertenece a un departamento, pero un departamento tiene varios programadores.

Programador se relaciona con ProjectAssignment en 1 – N donde un programador tiene varias asignaciones a proyectos, pero una asignación a un proyecto solo pertenece a un programador. Esta relación es producto de partir una relación N – N entre proyecto y programador.

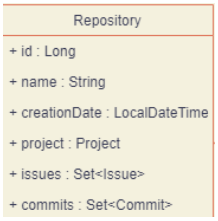
Programador se relaciona con IssueAssignment en 1 – N donde un programador tiene varias asignaciones a issues, pero una asignación a una issue solo pertenece a un programador. Esta relación es producto de partir una relación N – N entre issue y programador.

Un programador se relaciona con commits en 1 – N puesto que un programador tiene varios commits y un commit pertenece a un solo programador.

Los campos que representan relaciones han sido declarados con transient en los DAO para evitar su serialización, causante de recursividad en las llamadas SQL.

La única clave que se mantiene en la entidad programador es la de departamento.

Repository



Un repositorio consta de clave autonumérica, nombre, fecha de creación, proyecto al que está asociado, issues del repositorio y commits del repositorio.

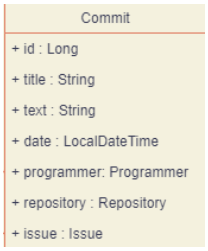
Un repositorio se relaciona con un proyecto en 1 – 1, puesto que cada repositorio tiene un proyecto y viceversa.

Un repositorio se relaciona con varios issues en una relación 1 – N donde los issues solo pertenecen a dicho repositorio.

Un repositorio se relaciona con varios commits en una relación 1 - N donde los commits solo pertenecen a dicho repositorio.

La entidad repositorio de la base de datos contiene la clave del projecto al que pertenece.

Commit



Un commit está formado por una clave autonumérica, un titulo, un texto, una fecha, un programador que lo ha realizado, el repositorio al que pertenece y la issue que resuelve.

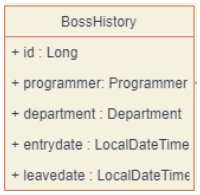
Un commit se relaciona en N – 1 con programador, pues un programador realiza varios commit, pero un commit solo pertenece a un programador.

Un commit se relaciona en N – 1 con repositorio, puesto que un repositorio tiene varios commits, mas un commit solo pertenece a un repositorio.

Un commit se relaciona en 1 - 1 con issue, puesto que una issue se resuelve con un commit y un commit solo resuelve una issue.

Commit contiene todas las claves de sus relaciones, pues es la N en todas sus relaciones excepto en la de issue, en la cual hemos decidido que commit reciba la clave.

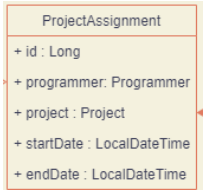
BossHistory



La entidad boss history nace de partir la relación N – N entre programador y departamento. Un boss history contiene los datos de un jefe de departamento, siendo estos el programador, el departamento, la fecha de comienzo de la jefatura y la de fin. La fecha de fin puede ser nula si aun sigue siendo el jefe del departamento.

Contiene las claves ajenas de las dos relaciones, programador y departamento.

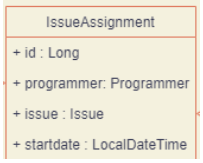
ProjectAssignment



Al igual que boss history, ProjectAssignment es una entidad que parte la relación N – N entre programador y proyecto. ProjectAssignment contiene los datos del programador, del proyecto, la fecha de inicio de la asignación y la de final en caso de haber terminado. En caso contrario el campo es nulo.

Contiene las claves ajenas de programador y departamento.

IssueAssignment



La ultima tabla que parte relaciones N – N es IssueAssignment, que parte la relación entre un programador y los issues. Contiene datos de ambas entidades y una fecha de inicio de la asignación de dicha issue al programador.

Contiene claves ajenas para programador y issue.

Ninguna de las tres tablas pivote tiene sus campos a transient, pues deseamos que la serialización incluya los datos de los dos objetos de la relación.

Implementación de la persistencia en Java

Clases principales

- Controladores: Los controladores son clases con patrón singleton. Dan una fachada de uso para los servicios y nos permiten obtener las entidades de la base de datos en formato JSON. Se le inyecta el servicio del que depende en el constructor. Contienen cinco métodos básicos: obtener todas las entidades, obtener la entidad según el identificador, insertar una entidad en la base de datos, actualizar una entidad de la base de datos y eliminar una entidad de la base de datos.
- Servicios: Los servicios se encargan de obtener todos los datos relacionados con la entidad a la que se les asocia. Por ejemplo, el servicio de BossHistory se encarga de obtener la entidad básica de la base de datos y sustituir las claves ajenas por los datos de las entidades que relaciona: programador y departamento. De esta forma un servicio emplea los repositorios necesarios para poder obtener todos estos datos y generar un DTO con ellos. Los métodos que implementa son findAll(), que nos da todos los DTO de la entidad del servicio; getEntidadById(id: long), el cual genera un DTO de dicha entidad con todos los datos de los objetos con los que se relaciona;insert(entidad: Entidad), que permite persistir una entidad en la base de datos;update(entidad: Entidad), que actualiza una entidad de la base de datos;delete(entidad: Entidad), que elimina una entidad de la base de datos.
- Repositorios: Los repositorios son los encargados de generar DAOs a partir de una tabla de la base de datos. Conocen las sentencias SQL o los métodos de JPA que acceden a las tablas y recuperan los datos de cada entidad modelándolos en un DAO. Tienen los métodos básicos de CRUD al igual que service.
- DTO: Un DTO es un objeto de transferencia de datos. Contiene los campos que representan a un objeto Java del problema Objeto – Relacional. Un DTO se genera con un conjunto de DAO.
- DAO: Un DAO es un objeto de acceso a datos. Contiene los campos que representan a una entidad en una base de datos relacional. Para obtener el DAO se hace una consulta a la base de datos y se contiene en objetos java cada entidad resultante de la consulta.
- Mappers: Son clases que implementan métodos para convertir DAO a DTO y viceversa. Existe un mapper para cada objeto Java.

Controladores

Los controladores son iguales en ambas versiones del proyecto. Acceden a las clases de los servicios, controlan la salida (el controlador no debe permitir que salgan excepciones, debe controlarlas todas) y la formatean en JSON mediante la libreria GSON. Todos ellos implementan el patrón Singleton. Los métodos implementados en los controladores son CRUD + getAllJSON(), método que realiza una select de todos los objetos de ese tipo.

Servicios

La clase abstracta BaseService es buen punto para ver el funcionamiento de los servicios. Esta clase implementa los métodos CRUD básicos del repositorio principal del servicio. Las clases servicio concretas después se encargan de rellenar el DTO con las entidades con las que se relacionan mediante otros repositorios.

Los servicios difieren entre ambas implementaciones debido al cambio de los DAO en JPA. Los DAO en JDBC representaban las relaciones mediante identificadores, mientras que los DAO de JPA ya representan la totalidad de la entidad. Este pequeño cambio implica que en JPA no es necesario rellenar las entidades ajenas, mientras que en JDBC sí.

Servicios JDBC

BossHistoryService: A parte de la implementación CRUD Básica, rellena el DTO mediante los métodos getDepartmentById() y getProgrammerById(). Estos métodos reciben el id guardado en el DAO, acceden al service de cada entidad y realizan una consulta por medio de sus método getById() para obtener el DAO de cada entidad, el cual luego insertan en el DTO final de todos los CRUDS básicos de la clase.

CommitService: A parte de la implementación CRUD Básica, rellena el DTO mediante los métodos getIssueById(), getRepositoryById() y getProgrammerById(). Estos métodos reciben el id guardado en el DAO, acceden al service de cada entidad y realizan una consulta por medio de sus método getById() para obtener el DAO de cada entidad, el cual luego insertan en el DTO final de todos los CRUDS básicos de la clase.

DepartmentService: A parte de la implementación CRUD Básica, rellena el DTO mediante el método fillDepartment(), que llama a los métodos getBossesOfDepartment() y getProjects(). El resultado del método getProjects() es posteriormente separado en proyectos activos y proyectos terminados mediante la API stream. Estos métodos realizan un findAll() en sus respectivos services y después filtran usando API stream solo aquellos registros que pertenecen al departamento en cuestión. En DepartmentService también se implementan dos comprobaciones del update, isBossWorkingAtProyects(), que comprueba si el programador está trabajando en algún proyecto para evitar que haya jefes de departamento que programen; y updateOldBoss, que comprueba si en el update ha cambiado el jefe y genera un nuevo registro en el boss history en caso de que así sea.

IssueAssignmentService: A parte de la implementación CRUD Básica, rellena el DTO mediante los métodos getIssueById() y getProgrammerById(). Estos métodos reciben el id guardado en el DAO, acceden al service de cada entidad y realizan una consulta por medio de sus método getById() para obtener el DAO de cada entidad, el cual luego insertan en el DTO final de todos los CRUDS básicos de la clase.

IssueService: A parte de la implementación CRUD Básica, rellena el DTO mediante los métodos getRepositoryById() y getBossProgrammerById(). Estos métodos reciben el id guardado en el

DAO, acceden al service de cada entidad y realizan una consulta por medio de sus método getById() para obtener el DAO de cada entidad, el cual luego insertan en el DTO final de todos los CRUDS básicos de la clase.

ProgrammerService: A parte de la implementación CRUD Básica, rellena el DTO mediante el método fillProgrammer(), que llama a los métodos getProjectsOfProgrammer(), getCommitsOfProgrammer(), getDepartmentById() y getIssuesOfProgrammer(). Estos métodos realizan un findAll() (salvo getDepartmentById() que utiliza un getById())en sus respectivos services y después filtran usando API stream solo aquellos registros que pertenecen al programador en cuestión.

ProjectAssignmentService: A parte de la implementación CRUD Básica, rellena el DTO mediante los métodos getProjectById() y getProgrammerById(). Estos métodos reciben el id guardado en el DAO, acceden al service de cada entidad y realizan una consulta por medio de sus método getById() para obtener el DAO de cada entidad, el cual luego insertan en el DTO final de todos los CRUDS básicos de la clase.

ProjectService: A parte de la implementación CRUD Básica, rellena el DTO mediante el método fillProject(), que llama a los métodos getDepartmentById() y getBossById(). Estos métodos utilizan un getById() en sus respectivos services para obtener el DAO de cada entidad, el cual luego insertan en el DTO final de todos los CRUDS básicos de la clase. También implementa un método checkRestrictions() que se realiza en el insert y el update, el cual comprueba que el programador no sea jefe de otro proyecto ni de un departamento accediendo a los servicios de Department y de Programmer.

RepositoryService: A parte de la implementación CRUD Básica, rellena el DTO mediante los métodos getProjectById(), getSetIssues() y getSetCommits(). Estos métodos acceden a los servicios de sus entidades para rellenar el DTO de repository con el proyecto al que pertenece el repositorio, sus commits y sus issues. En este Service también se ha implementado el borrado total del repositorio mediante los métodos deleteCommit() y deleteIssue(), métodos llamados durante el delete básico de repository para realizar un borrado en cascada.

Services JPA

Los servicios en JPA funcionan igual que en JDBC a excepción de los métodos getEntityById() que servían para rellenar los DTO. Dado que los DAO de JPA ya incluyen dichos objetos, no hemos necesitado los esos métodos. Sin embargo, en JPA también existen los métodos que rellenan las listas de los DTO y todos los demás métodos que implementan el CRUD básico.

Repositorios

Los repositorios son los encargados de acceder a la base de datos en busca de entidades atómicas, sin dependencias. Obtienen el DAO mediante consultas SQL básicas. Implementan CRUD con la interfaz CRUDRepo. Todos ellos retornan Optionals para lidiar con los nullPointerException. Además, todas las operaciones retornan la entidad con la que se trabaja en caso de acabar correctamente, mientras que lanzan excepciones en caso de no hacerlo.

Repositorios en JDBC

Realizan las consultas usando la clase DatabaseController, que implementa prepared statement por debajo. Manda la query a la base de datos y devuelve el resultado por medio de ResultSet. ResultSet nos permite obtener los resultados mediante clave-valor, resultados que posteriormente encapsulamos en un DAO para retornar.

Respositorios en JPA

Realizan las consultas usando los métodos de JPA. Inicia una transacción al principio de una operación de inserción, actualización o borrado, mientras que para selects solo inicia el controlador de hibernate. Utiliza find() de jpa para realizar la lectura de datos, persist() para insertar registros, merge() para actualizar registros y remove() para eliminarlos. En caso de que las operaciones fallen, se hace rollback a la transacción mediante transaction.rollback(), asegurándonos de que no se corrompen datos en la base de datos.

Mappers

Los mapper son clases que convierten de DAO a DTO (y viceversa) y donde realizamos el corte de la direccionalidad dejando no creados o a nulos aquellos atributos que no nos interesa que generen recursividad

Conexión a la base de datos: JDBC

Para conectar a la base de datos en el proyecto manual utilizamos la clase DatabaseController del paquete database. Esta clase contiene los datos de la conexión, tales como la url, el puerto, el nombre de la base de datos, el usuario y la contraseña para poder realizar la conexión. Estos datos se obtienen de un fichero de preferencias mediante la clase ApplicationProperties, que implementa un archivo para leer datos con el formato clave-valor. El archivo de nuestro proyecto que contiene dichas preferencias es ConectionProperties.properties, disponible en los recursos del proyecto. La clase DatabaseController también implementa métodos para ejecución de CRUD SQL mediante el uso de preparedStatement y result set.

Conexión a la base de datos: JPA Hibernate

Hibernate JPA utiliza el archivo persistence.xml disponible en los recursos en la carpeta META-INF para obtener todos los datos necesarios para la conexión con la base de datos. En este archivo podemos ver que proveedor utiliza Hibernate (en nuestro caso JPA), las clases que se están persistiendo, las propiedades de la conexión y, en caso de tener varias unidades de persistencia, las propiedades de todas ellas. En nuestro caso solo tenemos una unidad de persistencia, la default.

Docker

Para poder usar una base de datos relacional es importante levantar un servicio que contenga dicha base de datos. En esta práctica hemos hecho uso de la herramienta de gestión de contenedores Docker, mediante la cual hemos podido levantar el servicio de MariaDB e inicializar la base de datos con un script SQL.

El uso de esta herramienta se recoge dentro de la carpeta docker de nuestro proyecto.

La carpeta contiene el archivo docker-compose.yml, encargado de ejecutar todo el proceso. Levanta el servicio de MariaDB con la configuración deseada y un servicio adminer para poder visualizar la base de datos desde el navegador web. Dentro de la carpeta mariadb encontramos el Dockerfile que configura el contenedor de mariaDB con las variables de conexión. También copia el archivo de inicialización en la ruta del contenedor de inicialización de base de datos para que se realice dicha generación de datos SQL. Por último, dentro de la carpeta squl tenemos el script de generación de la base de datos del proyecto.

Utils

HibernateProxyTypeAdapter: esta clase la sacamos de internet para solucionar un problema que teniamos con el json que nos daba porque no nos salia registrado en hibernate para serializar la conversion de JSON.

Cifrate: usamos esta clase de Jose Luis para hacer el cifrado SHA256 para la contraseña.

GsonConverter, GsonLocalDateTime y GsonByteArrayToBase: sacamos estas clases de internet para resolver un problema que tiene Java EE 8 con el LocalDateTime a la hora de sacarlo por JSON.

TechnologiesParser: hicimos esta clase para parser del String almacenado en las tablas separadas por puntos y comas en listados para enum de technologies.

HibernateController: usamos esta calse de Jose Luis para usar la gestion del entity manager, su factory y entity transaction para JPA Hibernate con la base de datos para los repositorios.

AplicationProperties: esta clase de Jose Luis para leer el fichero.

ConnectionProperties.properties para los datos de conexion a la base de datos.

Database: hicimos esta clase para probar y comprobar la conexion con la base de datos.