# ITV Pepilandia

Sergio Pérez Fernández 2º DAM

Logotipo, Icono

Descripción generada automáticamente

# Índice

1. Introducción
2. El problema propuesto
3. Pepilandia SQLite
4. Pepilandia SQL + Spring
5. Pepilandia MongoDB
6. Conclusiones

Introducción

Una serie de proyectos deben ser creados y perfeccionados para las asignaturas de Acceso a Datos y Programación de Servicios y Procesos. Estos se tratan sobre un taller de ITV (Inspección Técnica de Vehículos) de la región de Pepilandia.  
  
Cada proyecto debe cumplir unas condiciones y expectativas mínimas y debe funcionar correctamente, sin errores y en el menor tiempo de ejecución posible utilizando métodos de agilización de procesos y recursos.   
  
Los proyectos deben estar basados en tecnologías diferentes, siendo estas SQL, Spring y MongoDB.   
  
Debe realizarse un diagrama de clases, permitir el cacheado de las entidades relevantes; implementación de medidas de seguridad, como es el cifrado de contraseñas de los trabajadores; emplear un enfoque Railway Oriented Programming y testear todas las clases importantes.   
  
Se otorgarán 3 semanas para realizar este reto.

**El problema propuesto**

El problema se refiere a una central de Inspección Técnica de Vehículos (ITV) en Pepilandia. La central cuenta con un conjunto de trabajadores, de los cuales se necesita registrar la siguiente información: nombre, teléfono, email (único), nombre de usuario (único), contraseña (cifrada) y fecha de contratación. Cada trabajador tiene una especialidad, que puede ser ELECTRICIDAD, MOTOR, MECANICA o INTERIOR. El salario de cada trabajador depende de su especialidad. Para ELECTRICIDAD el salario base es de 1800€, para MOTOR es de 1700€, para MECANICA es de 1600€ y para INTERIOR es de 1750€. Además, se agrega un bono de 100€ por cada tres años de antigüedad en la empresa. Además de los trabajadores regulares, hay un responsable de la ITV que también es un trabajador, pero tiene un plus de dirección de +1000€. El taller de la ITV gestiona las citas en intervalos de 30 minutos. Cada cita será atendida por un trabajador disponible en ese momento. Un trabajador no puede atender más de 4 citas en un intervalo y no puede haber más de 8 citas en el mismo intervalo. Para atender una cita, se requieren los datos del vehículo, como la marca, modelo, matrícula, fecha de matriculación y fecha de última revisión. Además, se necesitan los datos del propietario/a del vehículo, como el DNI, nombre, apellidos y teléfono. La ITV emite un informe favorable o desfavorable después de la inspección. El informe incluye datos sobre el frenado (valor entre 1 y 10 con dos decimales), contaminación (valor entre 20 y 50 con dos decimales), si el frenado es apto o no y si las luces son aptas o no. El informe debe reflejar los datos del trabajador que realizó la inspección, del vehículo y del propietario/a.

Adicionalmente, se deben realizar una serie de consultas:

* Encontrar al trabajador que gana el salario más alto sin ser el responsable.
* Calcular el salario medio de todos los trabajadores que no son responsables.
* Calcular el salario medio de todos los trabajadores agrupados por especialidad.
* Encontrar al trabajador con menos antigüedad.
* Listar los trabajadores ordenados por especialidad y luego por antigüedad.

Diagrama

Descripción generada automáticamente

Este es el diagrama de clases en el que se basa mi interpretación y se traducirá a cómo esté hecho el código. Las relaciones se establecen mediante relaciones y asociaciones, permitiendo el almacenamiento de manera estucturada.

Las entidades **“Trabajador”-“Especialidad”** se tratan con un cero a muchos por parte de especialidad hacia el trabajador ya que puede haber cero o varios trabajadores con la misma especialidad y uno a uno por parte de los trabajadores ya que estos deben tener una única especialidad.

**“Cita”-“Trabajador”** se comporta como un uno a uno desde la cita, ya que esta puede ser realizada por solamente un trabajador y cero a muchos desde el trabajador ya que este puede realizar cero o múltiples citas.

**“Trabajador”-“Informe”** se comporta exactamente igual que cita-trabajador, por lo que la relación es idéntica.

**“Cita”-“Informe”** mantienen una relación de uno a uno ya que una cita aparecerá en un único informe, y un informe solamente tendrá una cita.

**“Cita”-“Vehículo”** mantienen una relación de cero a muchos ya que puede haber realizado una cita con cero o múltiples veces con el mismo vehículo.

**“Vehículo”-“Propietario”** mantiene un uno a uno desde el vehículo ya que este tiene únicamente un dueño verdadero (en función de a nombre de qué DNI esté en los papeles) y uno a muchos desde el propietario ya que puede tener uno o múltiples vehículos iguales pero nunca cero porque sino no sería un propietario.

**“Propietario”-“Informe”** mantienen una relación de cero a muchos desde el propietario ya que puede haber estado involucrado en cero o en múltiples informes, y uno a uno por el opuesto, ya que un informe pertenece a un propietario únicamente.

**Pepilandia SQLite**

Texto

Descripción generada automáticamente

Este proyecto implementará una base de datos en memoria SQLite y los siguientes directorios y elementos:

Una captura de pantalla de un celular con texto e imagen

Descripción generada automáticamenteInterfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamenteInterfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente Texto

Config.properties

Es el proyecto más grande en cuanto a la cantidad de clases y su implementación es la más tediosa, pero más fácil de realizar para un principiante debido a su simpleza. En este proyecto los repositorios implementan un CrudRepository. Veremos ahora cómo se ha implementado la clase Cita.

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente Partimos del CrudRepository

Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamente

Aquí apreciamos su implementación y cómo se “ve” de primera mano el query a la base de datos, obteniendo los datos desde un resultSet directamente y añadiéndolos a la lista.

Texto

Descripción generada automáticamente

Aquí se inicia el controlador para la clase Cita, y se implementa un StateFlow para obtener cambios en tiempo real de compilación.

Texto

Descripción generada automáticamente

Aquí se utiliza el validador para la clase Cita. Luego, toda la logística para que solo se puedan realizar 8 citas por intervalo y 4 por trabajador se implementa en la clase Main.

**Pepilandia SQL + Spring**

Texto

Descripción generada automáticamente

Este proyecto comienza partiendo de Spring initializer, donde de una forma fácil y cómoda se nos será generado un proyecto adaptado para Spring. Su implementación es muy cómoda una vez sabes, pero su inconsistencia a la hora de ser implementada puede generar muchos problemas, quebraderos de cabeza y puede ser algo difícil para un primerizo de Spring. Los datos cargados a Spring se cargan de forma diferente en cuanto al dialecto, a pesar de seguir siendo un script .sql.

Texto

Descripción generada automáticamente

Este proyecto se lanza en localhost en el puerto designado, desde donde podremos realizar consultas mediante programas como Postman. Destaca por implementar anotaciones como @Table, @RestController y los mappings, donde se define qué hay que introducir como petición y de qué modo a la aplicación.

Texto

Descripción generada automáticamente Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente (Vemos que el mapping es de tipo “Get” y pide añadirle un cuerpo adicional a la cadena de conexión, siendo este el ID de la cita que se va a buscar.

Texto

Descripción generada automáticamente

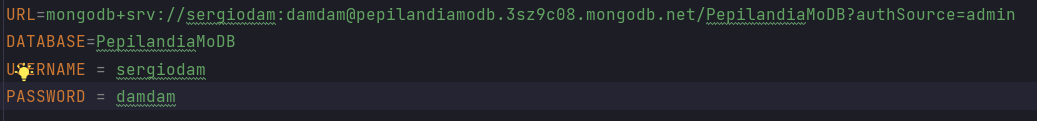
El repositorio es implementado por una clase Servicio, que a su vez es implementada por el Controller de la clase y utiliza una anotación @Service.

Texto

Descripción generada automáticamente

Pepilandia + MongoDB

Este proyecto implementa una base de datos en remoto a la cual nos deberemos conectar mediante una cadena de conexión. La implementación de Mongo es algo más complicada que SQLite pero mucho más fácil que Spring. Además, los datos son visibles en tiempo real desde la página web o aplicación Mongo Compass.



Su DataBaseManager es realmente fácil ya que es simplemente una creación de un cliente Mongo y su configuración a la ruta hasta la cadena de conexión.

Texto

Descripción generada automáticamente

Texto

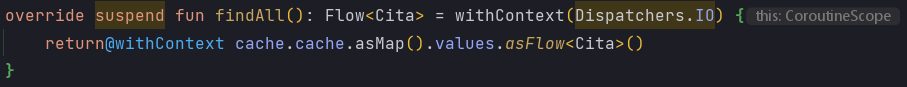
Descripción generada automáticamente

Mongo implementa el uso de las anotaciones @BsonId y @Contextual y newId para la generación automática de id’s para las entidades.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente Texto

Descripción generada automáticamente



La implementación de Caché es muy cómoda e intuitiva, pero requiere de gran tiempo para ser preparada correctamente ya que cada entidad tiene en paralelo otro repositorio sin cachear. Texto

Descripción generada automáticamente

Se implementan todas las funciones con corrutinas para no bloquear el hilo principal.

Texto

Descripción generada automáticamente Todas las contraseñas de los trabajadores están a salvo debido a que estas están cifradas.

Texto

Descripción generada automáticamente

Aquí podemos ver cómo se inician los repositorios y el monitor de Citas.

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media Este es un ejemplo de cómo usar el cliente de MongoDB para crear las colecciones con nombres personalizados.

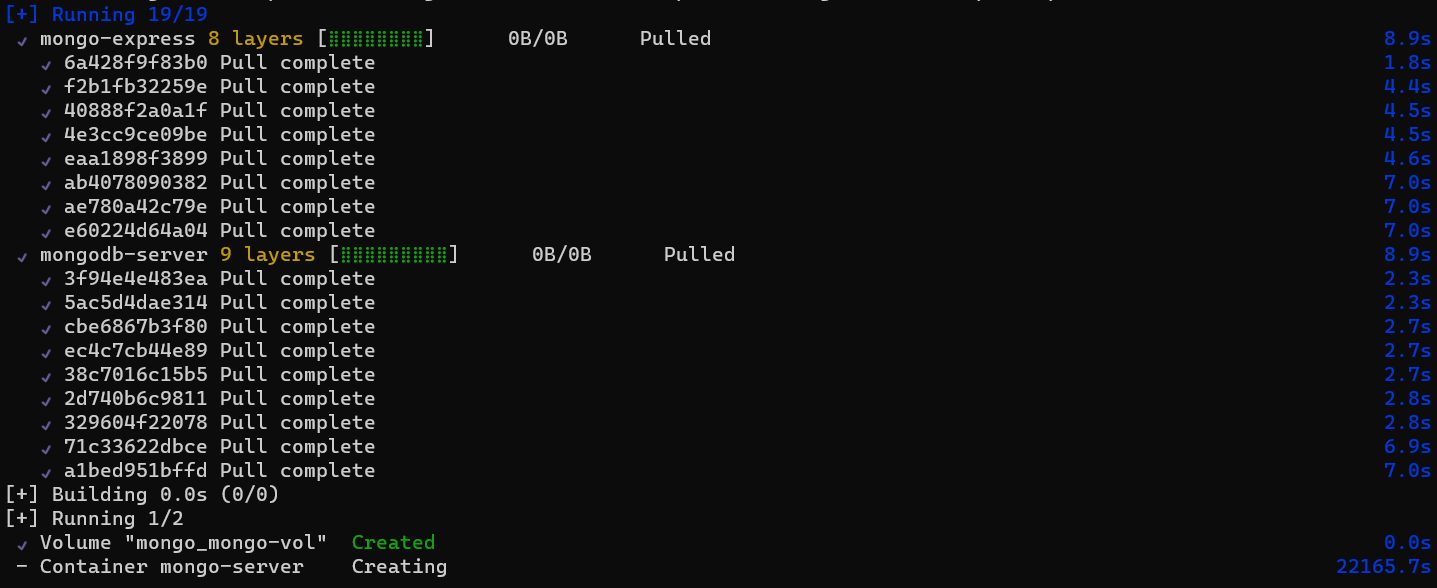
Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Conclusiones

Estas tecnologías son geniales para evitar la repetición constante de código (Mi propio IDE me marcaba líneas de código duplicado entre repositorios en SQL). Las notificaciones en tiempo real fueron geniales para el estudio de Programación de Servicios y Procesos debido al uso de SharedFlow y la implementación de Railway oriented programming ha ayudado a expandir mis horizontes respecto a la corrección de errores, anticipándome a posibles fallos siempre que sean provocados a propósito.

Este proyecto me ha ayudado mucho a comprender en profundidad diferentes tecnologías y a ver lo imposible que es utilizar Docker en mi portátil



Es por eso que todas las cadenas de conexión se levantan en remoto (Salvo las de Spring), porque mi ordenador no es capaz de utilizar Docker Compose.