Bases de Datos II

Trabajo Práctico Integrador: Etapa 1

Introducción

Esta primera etapa del Trabajo Práctico Integrador estará basada en la implementación de una aplicación Java orientada al estudio de la persistencia, tanto en términos conceptuales, como en el uso de tecnología en particular. La aplicación consistirá en una arquitectura multicapa clásica, debiéndose implementar una capa de **lógica de aplicación (servicios)** y otra de **acceso a datos (repositorios)** subyacentes.

La cátedra entregará un proyecto modelo implementado con Maven y el framework Spring y una configuración de Hibernate inicial, así como una serie de test cases que se deben satisfacer. Cada grupo deberá implementar todo lo necesario para que estos test cases pasen. En términos concretos, al correr el comando

```
mvn clean install
```

desde la línea de comandos debe obtenerse un build exitoso en donde pasen todos los tests.
Además, se debe entregar un archivo .sql que contenga sólo los comandos MySQL para crear el usuario a utilizar por la aplicación. No se debe utilizar el usuario **root** para acceder a la BBDD.

El proyecto base se encuentra disponible en Github. No realizar un *fork* del proyecto, ya que esto genera un repositorio publico, mientras que la entrega debe realizarse a través de un repositorio privado.

IMPORTANTE: Se debe redefinir el método `getGroupNumber()` de la clase `HibernateConfiguration` para que retorne su número de grupo.

Modelo de dominio de la aplicación a implementar

Se quiere generar una app de viajes turísticos para un emprendimiento local.

Una vez que una persona se ha registrado en el sistema de Turismo, puede elegir alguno de los varios recorridos diarios que la empresa ofrece y comprarlo.
Cada recorrido (*Route*) tiene su nombre, que es representativo del tramo que visita, el valor del mismo, el total en km del recorrido y pueden visualizarse cada una de las paradas (*Stop)* que componen el recorrido. Todos los recorridos se realizan una

vez por día. Además, la aplicación registra comentarios y valoraciones (estrellas) de otros usuarios que han realizado el mismo recorrido.
Dada la cantidad de personas que solicitan los recorridos, la empresa ha tenido que poner un tope máximo de personas por recorrido. Esa cantidad no es general sino que se define para cada uno y han acordado sólo vender los tours mientras el cupo lo permita.
Adicionalmente, se conoce una serie de choferes y guías turísticos que participan del recorrido.
A modo de ejemplo, podemos pensar en el recorrido "*Tour tradicional de Bs. As.*" que cuenta con paradas en distintos lugares emblemáticos de la ciudad de Buenos Aires como Caminito en La Boca, San Telmo, Obelisco, lago de Palermo y Recoleta; o en "*Tour del Delta*" que incluye a San Isidro, el Delta y también al Obelisco.
Una vez que se ha registrado la compra de un usuario para un recorrido determinado (*Purchase*), se le habilita la posibilidad de agregar comentarios y puntaje al recorrido realizado (*Review*) de forma tal que luego puedan usarlo en promociones del emprendimiento y poder captar mayor público interesado en los mismos.
Se ha pensado que los usuarios (*User*) se identifiquen en la aplicación mediante usuario y contraseña, y otros datos que pueden resultar útiles para posteriores ofertas de tours que la empresa realice. Por eso se les solicitan datos personales y de contacto como el nombre, un número de celular, un email, edad y fecha de nacimiento.
Los choferes (*DriverUser*) y guías turísticos (*TourGuideUser*) también son usuarios del sistema, por lo se registran los mismos datos que los usuarios, pero además cuentan con una serie de antecedentes que la empresa quiere mantener. El chofer, de trabajos anteriores, el guía turístico, sobre su formación en turismo. Cabe mencionar que los choferes y los guías no están afectados a un recorrido en particular, sino que pueden participar de varios según la demanda y necesidad de la empresa. En fechas particulares, por ejemplo, hay mayor demanda por los tours de la zona del Delta.
Durante todo el recorrido se ofrecen diversos servicios *(Service*) que un usuario puede contratar. De los servicios se conoce su nombre, su valor, una descripción y el proveedor que lo brinda. Por ejemplo, la confección de algún recuerdo (un mate pintado artesanalmente, una remera o una foto en un lugar determinado) .
De cada proveedor (*Supplier*) se conocen los servicios que ofrece, la razón social y un número de habilitación provincial.
Cada ítem de servicio aparece en la compra junto con el recorrido y la cantidad solicitada del mismo (*ItemService*).
La compra final contendrá el recorrido elegido y todos los servicios solicitados a lo largo del mismo para que el usuario pueda tener control de lo abonado. Además, sobre cada compra se registra un código interno y la fecha.

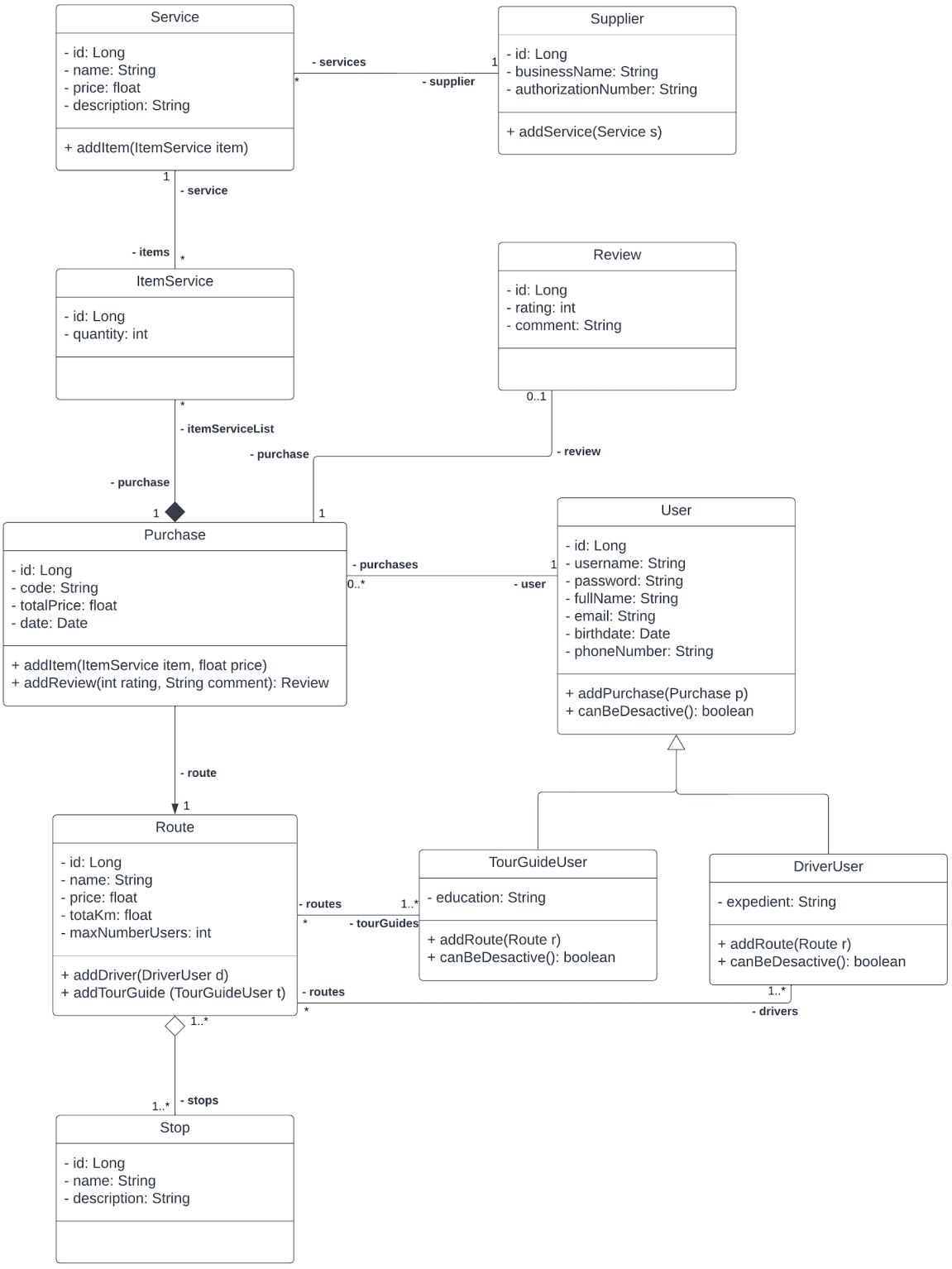

Fig. 1. Diagrama de Clases del Sistema Tours

Solución a implementar

El corriente trabajo consiste en la implementación de un servicio dado por la interfaz `ToursService` proporcionada, con su respectivo repositorio. Dicho servicio debe satisfacer los tests que se proporcionan en el trabajo base. En términos generales, el trabajo posee en dos partes: una primera que consiste en agregar persistencia al modelo descrito, además de CRUDs de las entidades,testeada en `ToursApplicationTest`, y una segunda que se basa en la implementación de una serie de consultas HQL, testada en `ToursQuerysTest`. Estos últimos *tests* inicializará la base de datos mediante la clase utilitaria `DBInitializer` provista. Esta creará un set de datos de prueba que contiene usuarios, recorridos, servicios y compras para que las consultas obtengan resultados. Cabe destacar que, para crear los objetos de prueba, `DBInitializer` utilizará los métodos del servicio que se prueban en la clase `ToursApplicationTest`

Listado de consultas:

- List<Purchase> getAllPurchasesOfUsername(String username)
  Obtiene y retorna el listado de compras realizada por el usuario con el nombre de usuario especificado por parámetro.

- List<User> getUsersWithNumberOfPurchases(int number);
  Obtiene y retorna el listado de usuario que hayan efectuado al menos una cantidad de compras mayor o igual al valor pasado por parámetro.

- List<Supplier> getTopNSuppliersItemsSold(int n);
  Obtiene y retorna los n proveedores (especificado por parámetro) con mayor cantidad de ítems vendidos. Tener en cuenta que la cantidad.

- List<Purchase> getTop10MoreExpensivePurchasesWithServices();
  Obtiene y retorna las 10 compras de mayor precio total y que incluyen servicios.

- List<Route> getTop3RoutesWithMoreStops();
  Obtiene y retorna las 3 rutas con mayor cantidad de paradas.

- Long getCountOfPurchasesBetweenDates (Date start, Date end);
  Obtiene y retorna la cantidad de compras registradas entre dos fechas.

- List<Purchase> getPurchaseWithService(Service service);
  Obtiene y retorna el listado de compras que incluyen el servicio especificado por parámetro.

- Long getMaxServicesOfSupplier();
  Obtiene y retorna la cantidad de servicios que posee el proveedor con mayor cantidad de servicios.

- List<Route> getRoutsNotSell();
  Obtiene y retorna el listado de recorridos que no fueron vendidas.

- List<Route> getTop3RoutesWithMaxAverageRating();
  Obtiene y retorna los tres recorridos con mayor promedio de rating en las reviews de compras asociadas a dicha ruta.

- List<Route> getRoutesWithMinRating();
  Obtiene y retorna los recorridos que tengan al menos una calificación de una estrella (rating)

- Service getMostDemandedService();
  Obtiene y retorna el servicio que más veces fue incluido en compras, teniendo en cuenta la cantidad.

- Route getMostBestSellingRoute();
  Obtiene y retorna la ruta que más veces aparece en compras, es decir, que más tickets vendió.

- List<Service> getServiceNoAddedToPurchases();
  Obtiene y retorna el listado de servicios que no fueron incluidos en ninguna compra.

- List<TourGuideUser> getTourGuidesWithRating1();
  Obtiene y retorna el listado de guías asignados a algún tour con un rating de una estrella.

- DriverUser getDriverUserWithMoreRoutes();
  Obtiene y retorna el usuario de tipo DriverUser que posee más rutas asociadas.

Requisitos detallados que deben satisfacerse en esta etapa

1. Definir el método `HibernateConfiguration.getGroupNumber()` para que devuelva un Integer con el número de grupo.
2. Escribir todo el código necesario para que la aplicación compile y todos los tests de la clase `ToursApplicationTests y ToursQueryTests` pasen. Esto implicará codificar una implementación concreta de la interfaz `ToursService` cuyos métodos están descritos vía Javadoc en la misma y también a través de los tests en `ToursApplicationTests y ToursQueryTests`.
3. Incluir un script SQL para crear el usuario a utilizar por la aplicación en la base de datos.
4. Correr el script anterior y luego `mvn clean install` debe resultar en un *build* exitoso en donde todos los tests pasen
5. El código tiene que estar subido en un repositorio privado de GitHub y la forma de entrega será por este medio. El repositorio privado deberá compartirse con los ayudantes designados para el grupo.

Requisitos técnicos para esta etapa

1. Tener instalado 17 o posterior (`javac -version` desde la línea de comandos debe funcionar)

2. Tener instalado Maven 3.5 (`mvn --version` desde la línea de comando debe funcionar)
3. Tener instalado MySQL 5.7 o posterior (Importante: no utilizar otro DBMS)
4. Es altamente recomendable utilizar un IDE para Java, preferentemente IntelliJ o VSCode - Nota: no se recomienda la utilización de Eclipse ya que estado actual es *deprecated*, teniendo algunos problemas de compatibilidad con JUnit, el cual se utiliza para test cases en el proyecto modelo entregado por la Cátedra.

Fecha de entrega para esta etapa

Viernes 16 de abril de 2025.