Práctica desarrollo de un servicio de transporte privado

Internet y Sistemas Distribuidos

Araceli Martínez Mateos
Carlos Saavedra Lorente
Felipe
Grupo 3.4
Repositorio isd073
Curso 2017-2018

1. Introducción

Esta práctica consiste en una aplicación capaz de gestionar conductores y viajes usando un diseño por capas. En ella, deberemos de ser capaces de añadir conductores, actualizar sus datos y buscarlos por diferentes criterios (búsqueda por identificador o búsqueda por ciudad en la que opera). Además, la aplicación permitirá a los usuarios contratar viajes con estos conductores, buscar los conductores disponibles en cada momento en su ciudad, puntuar los viajes realizados, consultarlos y ver una lista de los conductores con los que ha viajado.

La gestión se hace a través de una base de datos que manipularemos con una capa modelo. A su vez, en esta capa modelo tenemos dos subcapas principales: una capa de acceso a datos que trabaja directamente con la base de datos ejecutando las consultas SQL para la que utilizaremos MySQL, y una capa de lógica de negocio que implementa los casos de uso de nuestra aplicación.

Permitiremos que nuestra aplicación sea invocada remotamente a través de la red. Para implementamos una capa servicio con una implementación REST.

Además, usaremos:

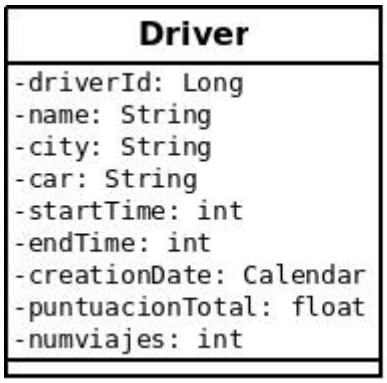
- Maven: para la construcción y ejecución del software.
- Subversion: Para el repositorio de versiones.
- Eclipse: como entorno de trabajo.

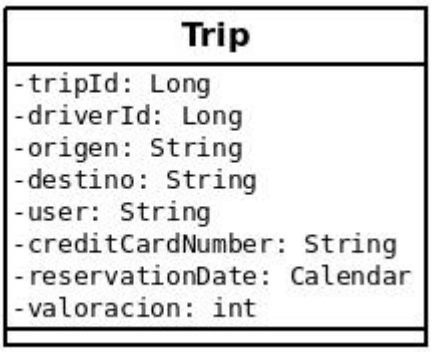
Para esta iteración de la práctica (it2) no se implementa la parte opcional con el servicio web SOAP ni la integración con Facebook.

2. Capa modelo

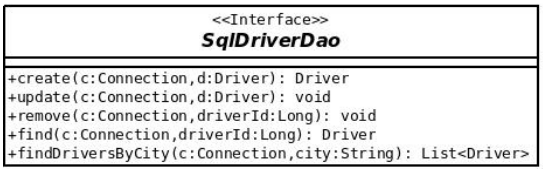
En este módulo se implementan las clases de la lógica de negocio correspondiente a los casos de uso de nuestra aplicación. Se gestiona el acceso a la base de datos además de la lógica de negocio. Los archivos con los que accedemos a la base de datos son los denominados DAO (Data Access Object). También se establecen las entidades que manejamos en la aplicación.

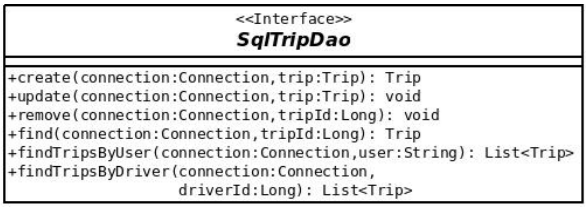
2.1. Entidades



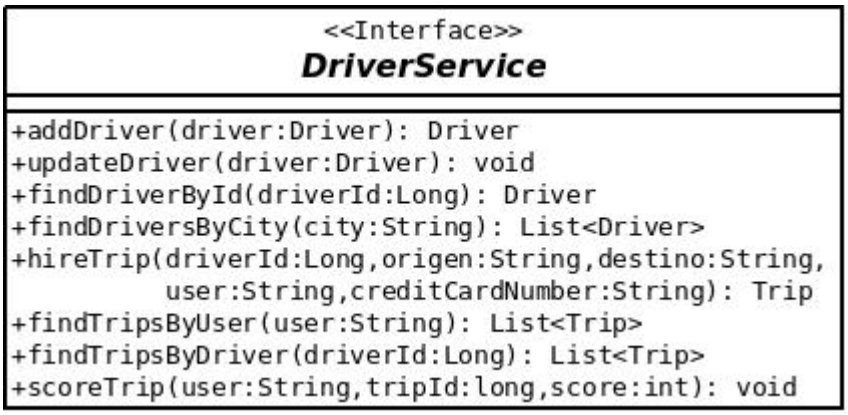


2.2. DAOs





2.3. Fachada



3. Capa Servicios

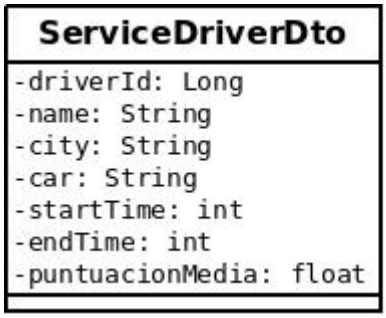
En esta capa se exponen una serie de puntos de entrada para las aplicaciones externas (app Clientes y app Conductores) que deseen invocar los distintos casos de uso de nuestra capa modelo.

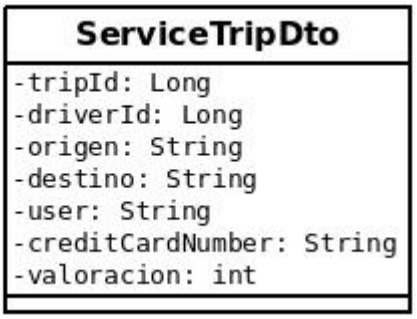
3.1. DTOs

En ServiceDriverDto, exponemos el campo puntuacionMedia, que será el que devolveremos en las respuestas a las peticiones de los clientes, ya que es lo que realmente les interesa a éstos. Se ocultan así los campos puntuacionTotal y numviajes de la entidad Driver.

Así mismo, se oculta también la fecha de alta de los conductores (campo creationDate de la entidad Driver).

Cabe señalar también que en el ServiceTripDto el campo creditCardNumber no será expuesto para los clientes conductores, y que hemos decidido ocultar la fecha de reserva del viaje.





3.2. REST

A continuación, indicamos cómo invocar cada caso de uso que ofrece el servicio

addDriver

La petición a se lanza a /drivers/. Se invoca con POST para crear un nuevo recurso. En el cuerpo en necesario enviar un XML con los tags <name> <city> <car> <startDate> <endDate>. (Dto de entrada: ClientDriverDto).

Devuelve el conductor creado como un ServiceDriverDto (Dto salida) y el código 201 CREATED si el conductor ha sido añadido correctamente, o un código 400 BAD_REQUEST si la petición es errónea (InputValidationException, validación de los datos del conductor).

findDriverById

La petición a se lanza a /drivers/. Se invoca con GET para obtener los datos de un recurso concreto, se indica la id del conductor que se quiere obtener directamente en la url /drivers/<driverId>.

Devuelve el conductor deseado como un ServiceDriverDto (Dto salida) y un código 200 OK si se ha ejecutado correctamente, un código 400 BAD_REQUEST si la petición es errónea (NumberFormatException, invalid driverId), o un 404 NOT_FOUND si el conductor no está en la base de datos (InstanceNotFoundException).

updateDriver

La petición a se lanza a /drivers/. Se invoca con PUT para actualizar el recurso. Se indica la id del conductor a actualizar directamente en la url /drivers/<driverId>. En el cuerpo se pasa un XML con los tags <driverId> <city> <car> <startTime> <endTime>. (Dto de entrada: ClienDriverDto).

En nuestro caso el nombre se envía, aunque no es relevante ya que no está permitido actualizarlo.

Devuelve un código 400 BAD_REQUEST si la petición es errónea (InputValidationException) un 404 NOT_FOUND si el conductor no está en la base de datos (InstanceNotFoundException) o un 204 NO_CONTENT para indicar que se ha ejecutado correctamente pero no devolvemos nada.

findTripsByDriver

La petición a se lanza a /trips/. Se invoca con GET para obtener los viajes de un conductor. Se indica la id del conductor que se quiere obtener directamente en la url / trips /<driverId>.

Devuelve los viajes realizados en una lista de ServiceTripDto (Dto salida) y un código 200 OK si se ha ejecutado correctamente, un código 400 BAD_REQUEST si la petición se ha hecho mal (invalid driverId, NumberFormatException), o un 404 NOT_FOUND si el conductor no está en la base de datos o no ha realizado ningún viaje (InstanceNotFoundException).

hireTrip

La petición a se lanza a /trips/. Se invoca con POST para crear un nuevo recurso. En el cuerpo en necesario enviar un XML con los tags <driverId> <origen> <destino> <usuario> <creditCard>.

Devuelve el viaje contratado como un ServiceTripDto (Dto salida) y código 201 CREATED si se ha ejecutado correctamente, un código 400 BAD_REQUEST si la petición se ha hecho mal (validación de los datos de entrada: comprobación de número de tarjeta valido, i.e. tratamiento de la excepción InputValidationException), un 404 NOT_FOUND si el conductor no está en la base de datos (InstanceNotFoundException) o un 410 GONE si el conductor está inactivo o fuera de horario (InvalidDriverException). Este código no debería utilizarse ya que es permanente, no así la situación que queremos reflejar de que un conductor no esta trabajando en el momento que se quiere contratar, que es algo temporal.

scoreTrip

La petición se lanza a /rating/. Se invoca con POST, con los parámetros <userId> <tripId> <score> en el cuerpo del mensaje.

Devuelve código 204 NO_CONTENT si se ha ejecutado correctamente, Si se ha enviado una valoración del viaje incorrecta devuelve código 400 BAD_REQUEST (InputValidationException), si no se encuentra el viaje que se desea puntuar un 404 NOT_FOUND, o para los casos en que tratemos de puntuar un viaje por segunda vez o no seamos el usuario que ha realizado el viaje 403 FORBIDDEN (InvalidRatingException).

findTripsByUser

La petición a se lanza a /trips/. Se invoca con GET para obtener los viajes de un conductor. Se indica el nombre del usuario que se quiere obtener directamente en la url /trips/?user=<user>.

Devuelve los viajes realizados en una lista de ServiceTripDto (Dto salida) y un código 200 OK si se ha ejecutado correctamente, un código 400 BAD_REQUEST si la petición se ha hecho mal (InputValidationException), o un 404 NOT_FOUND si el usuario no ha realizado ningún viaje (InstanceNotFoundException).

hiredTrips

La petición a se lanza a /drivers/. Se invoca con GET para obtener los conductores que han llevado alguna vez a un usuario concreto. Este usuario se indica directamente en la url /drivers/?user=<user>.

Devuelve los conductores en una lista de ServiceDriverDto (Dto salida) y un código 200 OK si se ha ejecutado correctamente, un código 400 BAD_REQUEST si la petición se ha hecho mal, o un 404 NOT_FOUND si el usuario no ha realizado ningún viaje (InstanceNotFoundException).

findDriversByCity

La petición se lanza a /drivers/. Se invoca con GET para obtener los conductores disponibles en una ciudad determinada. La ciudad de pasa como parámetro en la url /drivers?city=<city>.

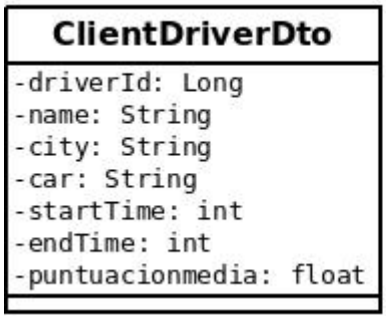
Devuelve los conductores disponibles en una lista de ServiceDriverDto (Dto salida), un código 200 OK si la operación se ha ejecutado correctamente y un código 400 BAD_REQUEST en caso de que la petición sea errónea (InputValidationException).

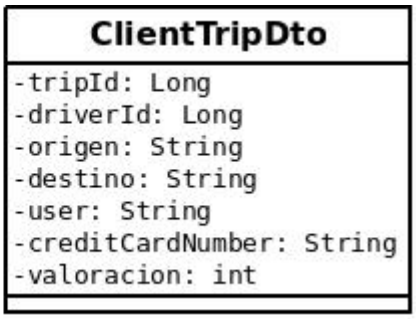
4. Capa Cliente

4.1. DTOs

Como comentamos en el apartado anterior y en concordancia con la visión ofrecida por la capa de servicios, en ClientDriverDto almacenaremos el campo puntuacionmedia para recuperar la puntuación media de un conductor.

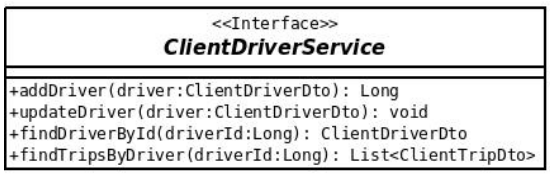
En función del tipo de aplicación cliente (cliente conductor o cliente usuario) que esté solicitando una lista de viajes (operaciones findTripsByDriver en app conductores y findTripsByUser en app usuarios) se podrá visualizar o no el número de tarjeta de crédito (findTripsByUser visualiza el valor de este campo, findTripsByDriver no).

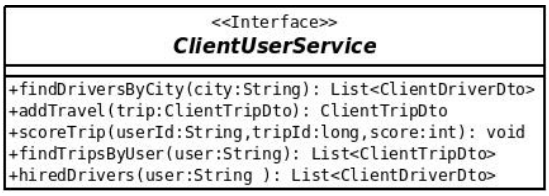




4.2. Capa de acceso al servicio

App conductores (izquierda) y app clientes (derecha):





5. Errores Conocidos

- Caso de uso updateDriver

Al actualizar un Driver, a pesar de que el nombre no se modifica, es necesario introducirlo igual para la petición update desde el cliente.

- Caso de uso findTripsByDriver

Al buscar viajes de un Conductor, si este no tiene viajes, el servicio responde de la misma manera que si el conductor no estuviese dado de alta.

- Caso de uso scoreTrip:

Método scoreTrip perteneciente a la capa de lógica de negocio de la capa modelo (DriverServiceImpl.java), firma del método:

public void scoreTrip (String user, long tripId, int score) throws InstanceNotFoundException, InputValidationException, InvalidRatingException;

El escenario normal de invocación de este método no funciona debido a una comparación de strings mal implementada. Por escenario normal nos referimos a una operación de puntuación de un viaje por parte del mismo usuario que realizó el viaje, la primera vez que trata de puntuar el viaje, y otorgándole éste una puntuación válida (en un rango entre 0 y 10).

En consecuencia, esta funcionalidad no está operativa. No obstante, implementamos igualmente esta operación del lado del cliente y la contemplamos en el servicio, a pesar de que, como se ha comentado, en el momento de invocar la operación bajo el escenario descrito salte una excepción que no debería producirse.

En el resto de los escenarios (casuísticas erróneas), la invocación de esta operación funciona como cabría esperar.

6. Manual de Usuario

Añadir conductores

```
mvn exec:java -Dexec.mainClass="es.udc.ws.app.client.ui.DriverServiceClient" -
Dexec.args="-a '<name>' '<city>' '<car>' <startTime> <endTime>"
```

Actualizar conductor

```
mvn exec:java -Dexec.mainClass="es.udc.ws.app.client.ui.DriverServiceClient" -
Dexec.args="-u <driverId> '<name>' '<city>' '<car>' <startTime> <endTime>"
```

Buscar conductor

```
mvn exec:java -Dexec.mainClass="es.udc.ws.app.client.ui.DriverServiceClient" -
Dexec.args="-f <driverId> "
```

Buscar conductores disponibles

```
mvn exec:java -Dexec.mainClass="es.udc.ws.app.client.ui.UserServiceClient" -
Dexec.args="-f '<city>' "
```

Reservar viajes

```
mvn exec:java -Dexec.mainClass="es.udc.ws.app.client.ui.UserServiceClient" -
Dexec.args="-a <driverId>  '<sourceAddress> ' '<targetAddress> ' '<user>'
'<creditCardNumber>' "
```

Puntuar viaje

```
mvn exec:java -Dexec.mainClass="es.udc.ws.app.client.ui.UserServiceClient" -
Dexec.args="-s  '<user>' <travelId> <points>"
```

Buscar conductores disponibles

```
mvn exec:java -Dexec.mainClass="es.udc.ws.app.client.ui.UserServiceClient" -
Dexec.args="-f '<city>' "
```

Buscar viajes de un conductor

```
mvn exec:java -Dexec.mainClass="es.udc.ws.app.client.ui.DriverServiceClient" -
Dexec.args="-t '<driverId>' "
```

Buscar viajes de un usuario

```
mvn exec:java -Dexec.mainClass="es.udc.ws.app.client.ui.UserServiceClient" -
Dexec.args="-t '<user>' "
```

Buscar conductores de un usuario

```
mvn exec:java -Dexec.mainClass="es.udc.ws.app.client.ui.UserServiceClient" -
Dexec.args="-d '<user>' "
```