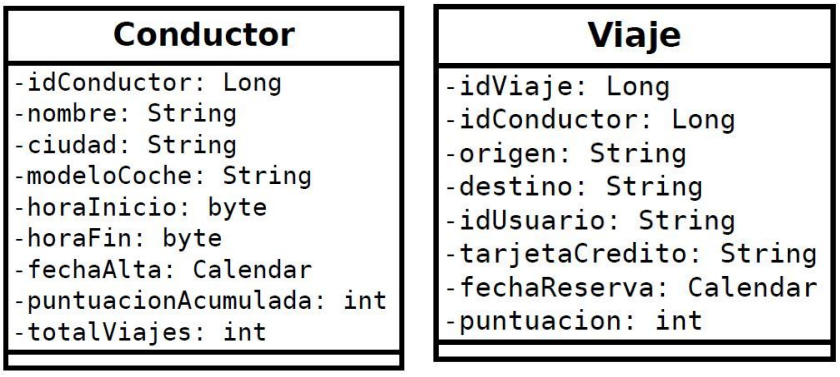
1. Introducción

En esta práctica haremos una aplicación estilo Uber para viajeros y conductores. Para ello haremos un modelo de datos, un cliente y un servicio REST.
Hemos realizado la segunda iteración de la práctica con los resultados aquí mostrados.
No se ha realizado la parte opcional

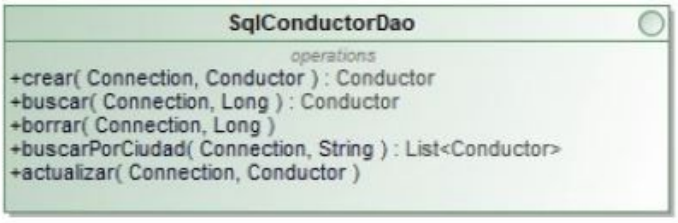
2. Capa modelo

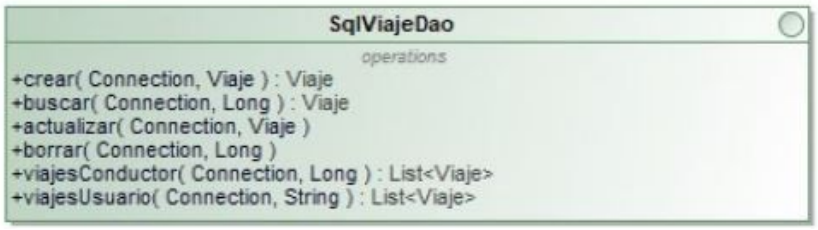
2.1. Entidades



Con el fin de evitar datos redundantes en la BD, decidimos incluir como atributos puntuacionAcumulada y totalViajes. Podríamos haber añadido la media en la BD, pero para evitar redundancia y operaciones en la BD la media vendrá calculada mediante estes dos atributos fuera de ella.
Para simplificar la codificación, hemos modelado la puntuación “nula” con el valor -1.

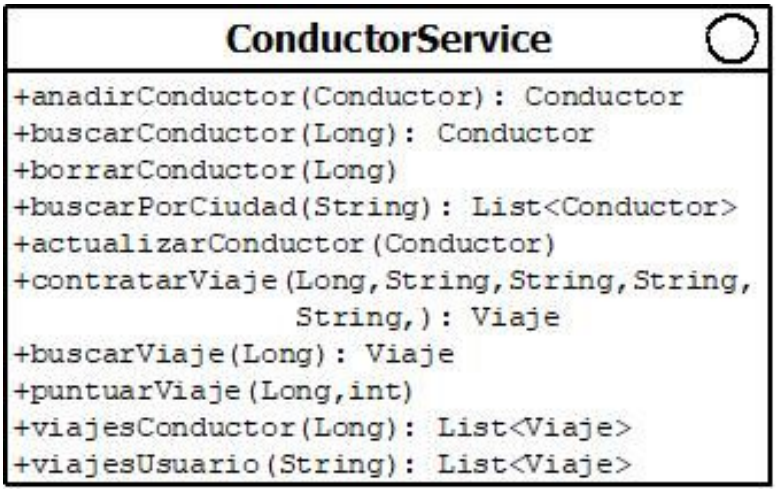
2.2. DAOs





El método buscarPorCiudad permite buscar por ciudad y hora, ya que recibe la hora del sistema y comprueba si la hora actual se encuentra dentro del rango de horas disponibles del conductor. Si no recibe un parámetro nulo en el campo ciudad, este método muestra todos los conductores activos (media > 5 o sin puntuación).

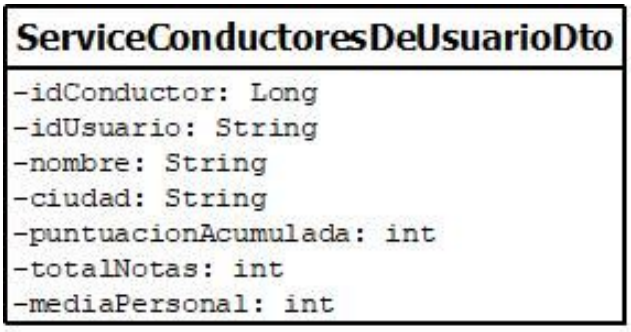
2.3. Fachada

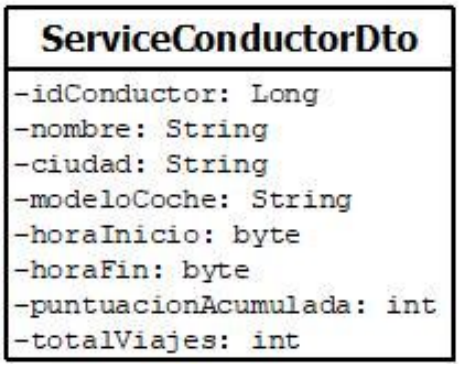


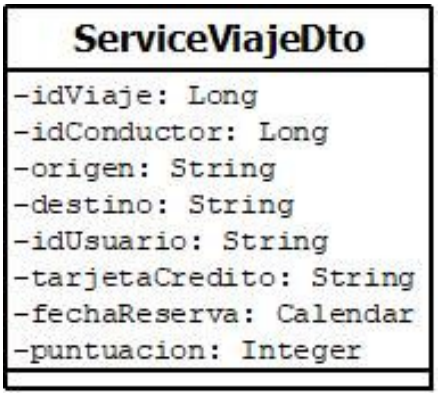
En este punto validamos los datos de entrada.

3. Capa de servicios

3.1. DTOs







Hemos creado un Dto ServiceConductoresDeUsuarioDto debido a que hemos tenido que crear un Servlet específico por claridad por implementar una funcionalidad que no encaja en el Servlet de conductor ni en el de viaje.

Este nuevo DTO muestra la relación que se establece entre un conductor y un usuario.

3.2. REST

Usamos el POST de Viaje tanto para puntuar como para insertar, ya que inicialmente estábamos pasando datos a un para puntuar a un Put mediante un body.form cuando esa función está destinada a métodos Post.

El criterio seguido para escoger los códigos HTTP de respuesta asociados a cada excepción ha sido el siguiente:

SC_NOT_FOUND : Usado para lanzar una excepción de un viaje que no existe(en el servlet de viaje) , o que no se ha encontrado un conductor (en el servlet conductor)

SC_BAD_REQUEST : se lanza como InputValidationException que capturamos en el cliente para lanzar excepción de hora inválida.

SC_CONFLICT : Usado para lanzar una excepción cuando detectamos un conductor no disponible (fuera de horas de trabajo)

SC_FORBIDDEN : No se debe permitir puntuar un viaje más de una vez, por lo tanto detectamos ese caso mediante el siguiente código HTTP de respuesta, asociado a la excepción correspondiente (PuntuacionRepetidaException)

SC_UNAUTHORIZED : Un usuario no está autorizado a puntuar ningún viaje que no sea suyo. En este caso usamos este código para detectar dicha Excepción.

-Añadir conductor
URL: */conductor*
Método de invocación : POST
Parámetros: *nombre(String), ciudad(Str), modeloCoche(Str), horaInicio(byte), horaFin(byte)*
DTO:*Conductor*

-Buscar conductor
URL: */conductor/idConductor*
Método de invocación : *GET*
Parámetros: *idConductor(String)*
DTO:*Conductor*

-Modificar conductor
URL: */conductor/idConductor*
Método de invocación : *PUT*
Parámetros: *idConductor, nombre, ciudad, Coche, horaInicio, horaFin*
DTO: *Conductor*

-Borrar conductor
URL: */conductor/idConductor*
Método de invocación : *DELETE*
Parámetros: *idConductor*
DTO:*Conductor*

-Ver viajes de un conductor
URL: */viaje?idConductor=1*
Método de invocación : *GET*
Parámetros: *idConductor*
DTO:*Viaje*

-Buscar conductores por ciudad
URL: */conductor?ciudad=ciudad*
Método de invocación : *GET*
Parámetros: *Ciudad*
DTO: *Viaje*

-Buscar viajes de un usuario
URL: */viajes?idUsuario=idUsuario*
Método de invocación : *GET*
Parámetros: *idUsuario*
DTO: *Viaje*

-Buscar conductores de un usuario
URL: */conductoresDeUsuario/idUsuario*
Método de invocación : *GET*
Parámetros: *idUsuario*
DTO: *Viaje*

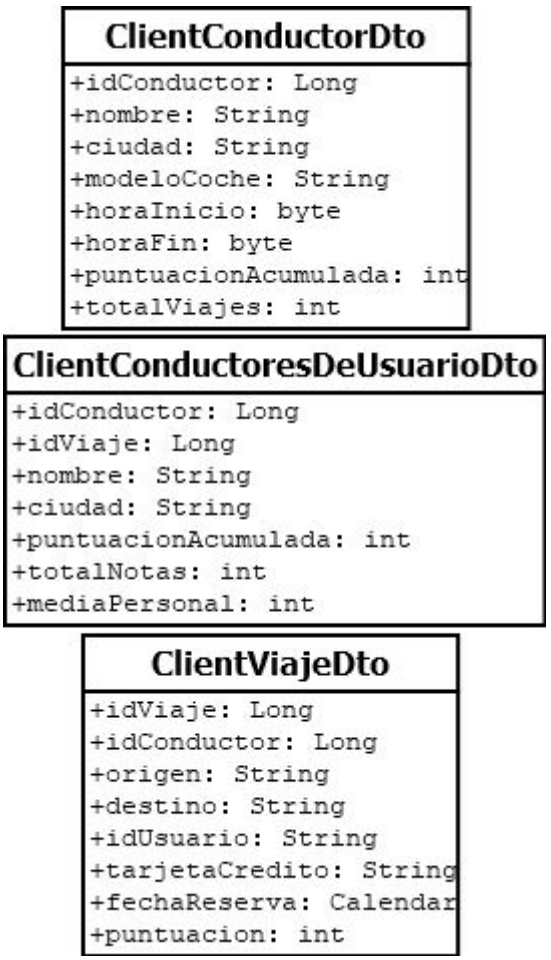
-Contratar Viaje
URL: */viaje*
Método de invocación : *POST*
Parámetros: *idConductor origen destino idUsuario tarjetaCredito(String)*
DTO: *Viaje*

-Puntuar Viaje
URL: */viaje/idUsuario/puntuar*
Método de invocación : *POST*
Parámetros: *idViaje idUsuario puntuacion(int)*
DTO: *Viaje*

-Buscar conductores por usuario
URL: */conductoresDeUsuario/idUsuario*
Método de invocación :*GET*
Parámetros: *idUsuario*
DTO: *conductoresDeUsuario*

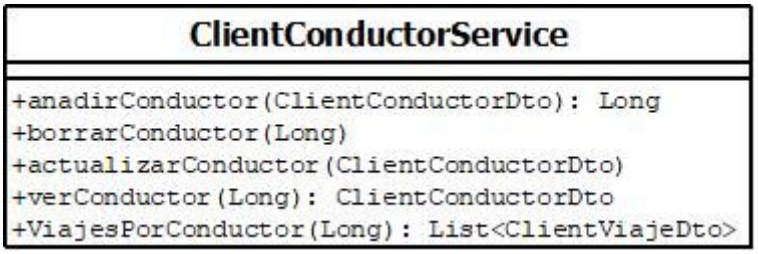
4. Aplicaciones cliente

4.1. DTOs



Las decisiones de diseño en estos DTOs son las mismas que en el servicio, pues estos DTOs son los que recibe el cliente del servicio, así que deben tener los mismos campos.

4.2. Aplicación cliente de conductores: capa de acceso al servicio



4.3. Aplicación cliente de usuarios: capa de acceso al servicio

6. Errores conocidos

-Al pasar la fecha del servicio al cliente, nos dimos cuenta de que la fecha llega como un nulo. Para solventar esto, tendríamos que convertir la fecha en un xml y enviarsela al cliente, haciendo este la operación inversa. Aunque sabemos como arreglarlo, no lo hicimos por cuestión de tiempo, ya que nos dimos cuenta tarde del error.

-Al añadir un viaje con hora inválida, el programa no muestra ningún error por pantalla en el tomcat, pero si en el jetty. No sabemos a qué se debe esto.

-Como nota, en vez de usar el plural en diversas partes del programa (conductores, viajes), comenzamos a hacerlo en singular, no es lo más adecuado.