PRÁCTICA DE INTERNET E SISTEMAS DISTRIBUÍDOS

Sergio García Cascón - sergio.gcascon@udc.es
Pedro Pillado García-Gesto - pedro.pilladog@udc.es
Daniel Quintillán Quintillán - daniel.quintillan@udc.es

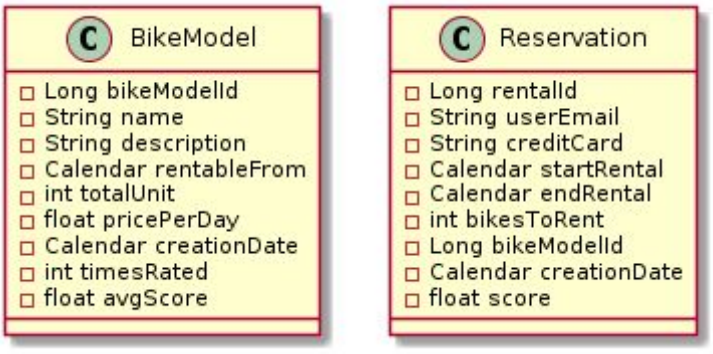
1. Introducción

El objetivo de esta práctica es la elaboración de un servicio de gestión de alquileres de bicicleta mediante la creación de tres capas: capa de acceso a datos, capa de lógica de negocio y capa de servicios.
En el caso particular de esta práctica la aplicación es invocada exclusivamente mediante una API REST ya que la implementación de SOAP corresponde a los trabajos tutelados, que no fueron realizados.

Además de desarrollar las Capas Modelo y Servicios de la aplicación, se requiere la elaboración de un programa cliente que invoque mediante una CLI las operaciones de nuestra Capa Servicios, diferenciando entre los perfiles de administrador del sistema y usuario final.

2. Capa modelo

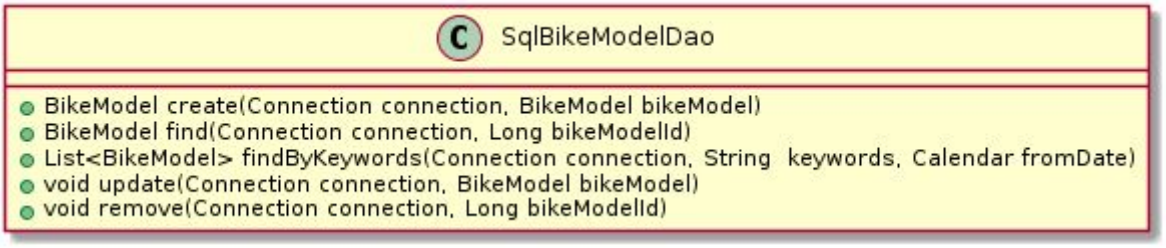
2.1. Entidades

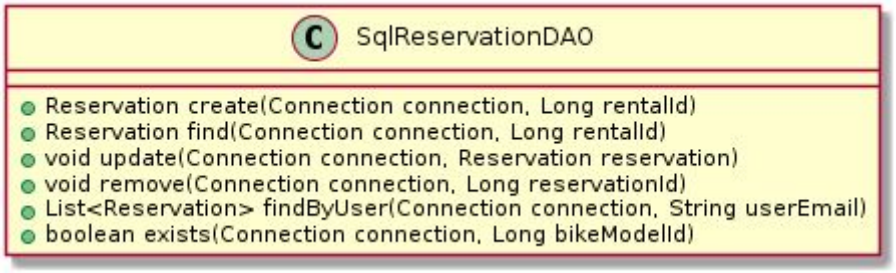


- Aspectos a remarcar:

Los objetos de la clase BikeModel cuentan con los atributos timesRented y avgScore para facilitar el cálculo de la media de las puntuaciones de las reservas cuando el cliente las pida. De no ser así, el cálculo de la puntuación media de las bicicletas implicaría una consulta a la tabla de Reservations para obtener la media de todas las puntuaciones relacionadas con el modelo en cuestión. En esta decisión se busca maximizar la eficiencia de las operaciones findBike y findByKeyWords, ya que se estima que van a realizarse desde el cliente del usuario final con mucha frecuencia.

2.2. DAOs





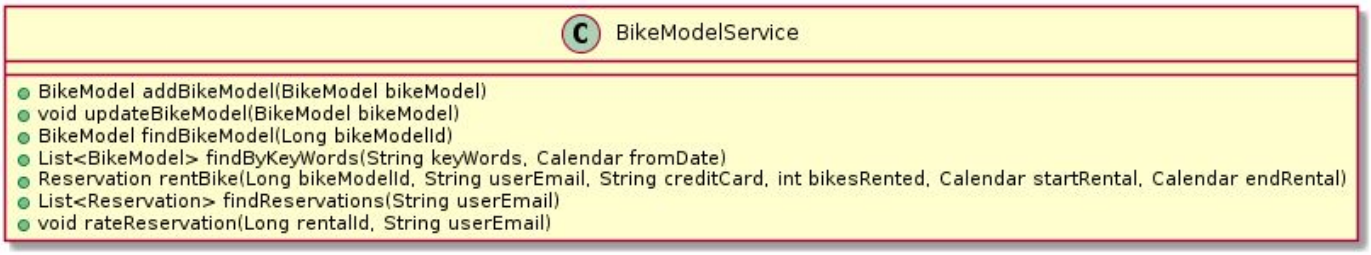
- Aspectos a remarcar:

En la clase SqlReservationDAO se creó el método exists() con el fin de facilitar la comprobación de la existencia de reservas hechas a un determinado modelo de bicicleta. Esta información es necesaria ya que se requiere que el administrador no pueda retrasar la fecha de disponibilidad de una bicicleta si ésta ya ha sido reservada.

Todos los métodos del DAO incluyen como parámetro una instancia de una conexión con la base de datos en la que se realizan las operaciones, además de la información necesaria para realizar cada consulta.

El método remove() aparece exclusivamente en la capa de acceso a datos ya que es necesario de cara a las pruebas. No obstante, ya que no se encuentra entre los requisitos del sistema, se decide no incluirlo en el servicio.

2.3. Fachadas

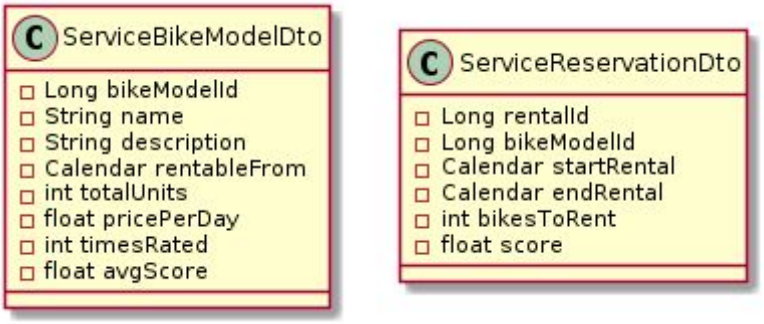


- Aspectos a remarcar:

La fachada del servicio de la capa modelo incluye exclusivamente funciones que representan casos de uso reales que llevará a cabo el usuario o el administrador del sistema.

3. Capa servicios

3.1. DTOs



- Aspectos a remarcar:

Ya que ninguna de las aplicaciones cliente va a necesitar conocer la fecha de creación de un modelo de bicicleta, el ServiceBikeModelDto prescinde de este atributo de BikeModel.
En el caso de ServiceReservationDto la diferencia es más sustancial ya que se prescinde de userEmail, de creditCard y de creationDate.

El usuario necesita userEmail para realizar y buscar reservas por email, pero no está presente en la información devuelta sobre las mismas. Esto se debe a que la búsqueda está basada en el email del usuario que hizo la reserva y no sería útil mostrarlo nuevamente. También es usado para hacer una reserva, pero en este caso, la información no viaja a través de un DTO sino que la creación de la reserva se resuelve por los parámetros de la petición.

Lo mismo pasa con la información de la tarjeta de crédito, ya que solo es usada en la creación de una reserva. Como ya se ha comentado, esta información llega a la capa de lógica de negocio a través de los parámetros de la petición sin estar encapsulada en un DTO. Asimismo, por requisitos del sistema no existe ninguna operación que devuelva información sobre la tarjeta de crédito usada en una reserva.

La ausencia de CreationDate en el DTO de la reserva se debe a los mismos motivos del DTO de bikeModel.

3.2. REST

Funciones del cliente administrador:

addBikeModel:
URL: /bikeModel
Petición: POST
DTO input: BikeModelDto
Códigos de respuesta HTTP: 201 Created, 400 Bad Request

updateBikeModel:
URL: */bikeModel/*{id}
Petición: PUT
DTO input: BikeModelDto
Códigos de respuesta HTTP: 400 Bad Request, 404 Not Found, 204 No Content

findBikeModel:
URL: */bikeModel?bikeModelId={id}*
Petición: GET
DTO output: BikeModelDto
Códigos de respuesta HTTP: 200 Ok, 400 Bad Request, 404 Not Found

Funciones del cliente usuario final:

findByKeyWords:
URL: */bikeModel?keyWords={keywords}&rentableFrom={yyyy-MM-dd}*
Petición: GET
Códigos de respuesta HTTP: 200 Ok, 400 Bad Request
DTO output: Lista de BikeModelDto

rentBike:
URL: /reservation
Petición: POST
Input: Los campos de la operación se incluyen a través de x-www-form-urlencoded
Códigos de respuesta HTTP: 201 Created, 400 Bad Request, 404 Not Found

findReservations:
URL: /reservation?userEmail={email}
Petición: GET
Códigos de respuesta HTTP: 200 Ok, 400 Bad Request
DTO output: Lista de ReservationDto

rateReservation:
URL: /reservation/{id}?userEmail={email}&score={score}
Petición: PUT
Códigos de respuesta HTTP: 204 No Content, 400 Bad Request, 404 Not Found

- Aspectos a remarcar:

En la función findByKeyWords se ha decidido que tanto el parámetro de búsqueda keyWords como rentableFrom sean campos obligatorios para la búsqueda, pudiendo asignar a keyWords una cadena vacía para indicar que no se desean aplicar restricciones en la descripción de las bicicletas.

La función rentBike no tiene DTOs como entrada porque, siendo consecuente con las características de la Capa Modelo, rentBike acepta los parámetros sin encapsular para crear internamente un objeto Reservation del que devuelve el identificador.

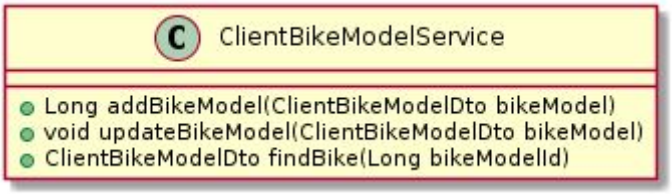
4. Aplicaciones cliente

4.1.DTOs: Administrador

- Aspectos a remarcar:

No es necesario incluir ningún DTO para transferir información sobre reservas ya que las funcionalidades de la aplicación de administrador solo consideran operaciones relativas a modelos de bicicleta (addBikeModel, updateBikeModel y findBike). No se devuelve ni manipula información adicional sobre reservas en estas operaciones.

4.2. Capa acceso al servicio: administrador



- Aspectos a remarcar:

En este caso las funciones representan las interacciones de los usuarios con el sistema de una forma muy similar a las peticiones HTTP que recibirá la capa de servicios.

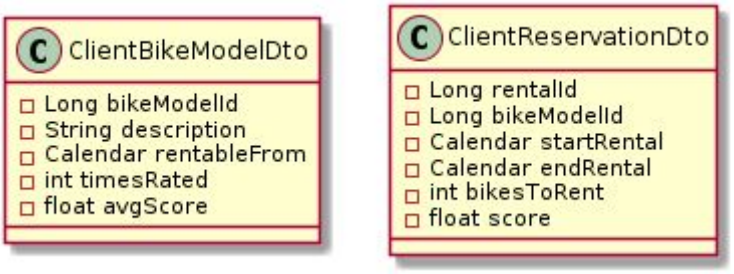
API de la interfaz de cliente: administrador:

-addBike <name> <description> <rentableFrom> <totalUnits> <pricePerDay>

-updateBike <bikeModelId> <name> <description> <rentableFrom> <totalUnits> <pricePerDay>

-findBike <bikeModelId>

4.3. DTOs: usuario

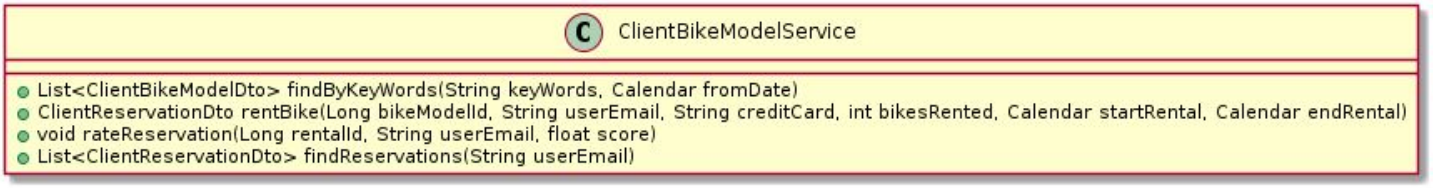


- Aspectos a remarcar:

Los atributos que faltan en ClientBikeModelDto respecto a su DTO homólogo de la capa de servicios son: pricePerDay, totalUnits y name ya que ninguna de las funciones de la aplicación usuario necesitan esa información.

Ya que el cliente del usuario es el que realiza todas las operaciones que tienen que ver con reservas y en el findReservations se necesitan todos los atributos de esta clase, el ReservationDto de la capa de servicios se mantiene íntegro.

4.4. Capa de acceso al servicio: usuario



- Aspectos a remarcar:

En este caso las funciones representan las interacciones de los usuarios con el sistema de una forma muy similar a las peticiones HTTP que recibirá la capa de servicios.

API de la interfaz de cliente: usuario:

-findBikes <keywords> <rentableFrom>

-reserve <bikeModelId> <userEmail> <creditCard> <bikesRented> <startRental> <endRental>

-findReservations <userEmail>

-rateReservation <rentalId> <userEmail> <score>

6. Errores conocidos

- El programa no comprueba si los emails proporcionados por los usuarios siguen el formato “local-part@domain”.
- Si la fecha no cumple el formato yyyy-MM-dd, el programa sigue la pila de errores en vez de imprimir un mensaje de excepción acotado.