BikeRental – Memoria de la Práctica

Santiago Millán y Jorge Paz Ruza – Grupo 3.3

1. Introducción

El objetivo de esta práctica ha sido la aplicación de los conceptos y tecnologías aprendidos en la asignatura para el desarrollo de un servicio, en este caso para una empresa de alquiler de bicicletas. Hemos desarrollado una aplicación que sigue una arquitectura en capas , incluyendo la capa acceso a datos, la capa lógica de negocio y la capa servicios. Esta aplicación puede ser invocada remotamente usando REST . La capa acceso a datos utiliza una base de datos relacional (MySQL) para acceder a la información pertinente.

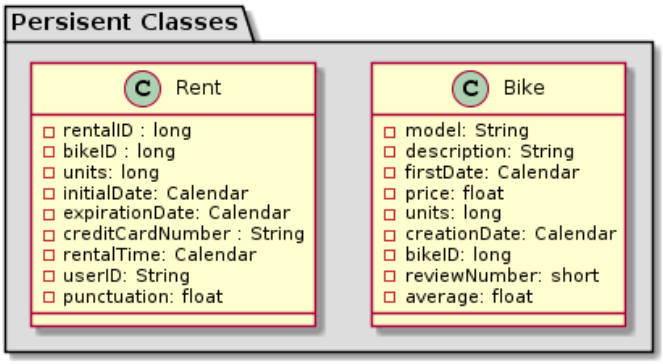
2. Capa Modelo

En la capa modelo de nuestra aplicación podemos encontrar el conjunto de clases que implementan la lógica de negocio de los casos de uso de la aplicación. Además, podemos distinguir dos subcapas dentro de ella:

- La **capa de acceso a datos**, implementada a través de DAOs para acceder a la base de datos de la aplicación
- La **fachada del modelo**, que permite a los servicios ser usada como intermediaria para utilizar la lógica de negocio acceder a la base de datos.

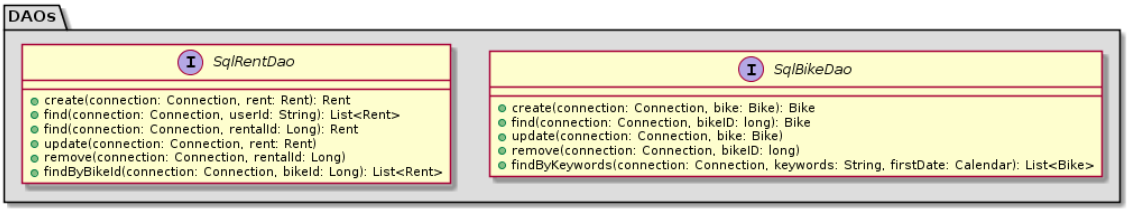
2.1 Entidades

Hemos definido dos entidades persistentes, Bike y Rent, que los DAOs utilizarán como medio para intercambiar información con la BBDD.



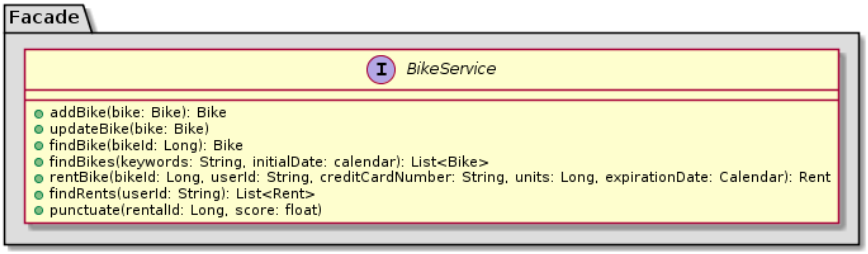
*En el caso del atributo *units*, presente tanto en Bike como en Rent, se nos recomendó utilizar un *longint* en vez de un *shortint* para tener una mayor consistencia entre los distintos tipos de int de las clases, y porque no conocemos el tamaño de BikeRental ni su volumen de negocio.

2.2 DAOs



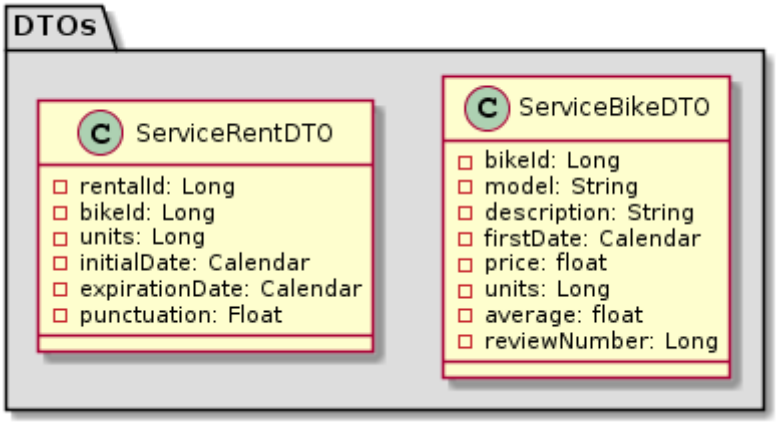
*Pese a que los métodos *update* y *remove* no forman parte de los casos de uso de la aplicación, pero son necesarios para llevar a cabo las pruebas automatizadas

2.3 Fachadas



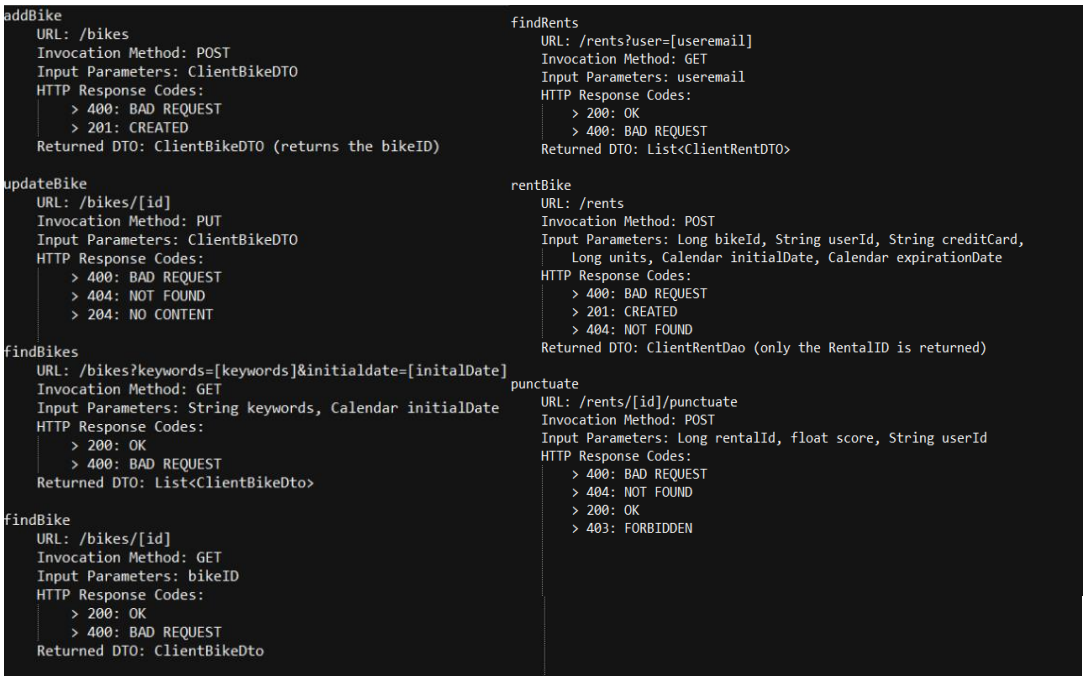
3. Capa Servicios

3.1 DTOs



*En los DTOs del servicio desaparece la información privada sobre el usuario (email, tarjeta de crédito) y la fecha de creación en la BBDD del rent tanto porque no son necesarios para los distintos servicios de la aplicación, como por razones de seguridad.

3.2 REST



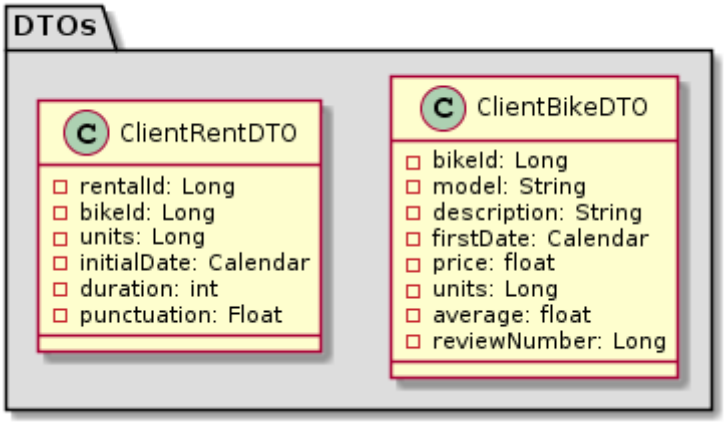
```
addBike
    URL: /bikes
    Invocation Method: POST
    Input Parameters: ClientBikeDTO
    HTTP Response Codes:
        > 400: BAD REQUEST
        > 201: CREATED
    Returned DTO: ClientBikeDTO (returns the bikeID)

updateBike
    URL: /bikes/[id]
    Invocation Method: PUT
    Input Parameters: ClientBikeDTO
    HTTP Response Codes:
        > 400: BAD REQUEST
        > 404: NOT FOUND
        > 204: NO CONTENT

findBikes
    URL: /bikes?keywords=[keywords]&initialdate=[initalDate]
    Invocation Method: GET
    Input Parameters: String keywords, Calendar initialDate
    HTTP Response Codes:
        > 200: OK
        > 400: BAD REQUEST
    Returned DTO: List<ClientBikeDto>

findBike
    URL: /bikes/[id]
    Invocation Method: GET
    Input Parameters: bikeID
    HTTP Response Codes:
        > 200: OK
        > 400: BAD REQUEST
    Returned DTO: ClientBikeDto

findRents
    URL: /rents?user=[useremail]
    Invocation Method: GET
    Input Parameters: useremail
    HTTP Response Codes:
        > 200: OK
        > 400: BAD REQUEST
    Returned DTO: List<ClientRentDTO>

rentBike
    URL: /rents
    Invocation Method: POST
    Input Parameters: Long bikeId, String userId, String creditCard,
        Long units, Calendar initialDate, Calendar expirationDate
    HTTP Response Codes:
        > 400: BAD REQUEST
        > 201: CREATED
        > 404: NOT FOUND
    Returned DTO: ClientRentDao (only the RentalID is returned)

punctuate
    URL: /rents/[id]/punctuate
    Invocation Method: POST
    Input Parameters: Long rentalId, float score, String userId
    HTTP Response Codes:
        > 400: BAD REQUEST
        > 404: NOT FOUND
        > 200: OK
        > 403: FORBIDDEN
```

*Los códigos HTTP devueltos por los servlets han sido decididos con el siguiente criterio:

- 400 (BAD REQUEST): Si la petición incluye parámetros o paths inválidos que el usuario debería corregir
- 201 (CREATED): Si la petición ha creado un recurso individual nuevo
- 404 (NOT FOUND): Corresponde con la InstanceNotFoundException, cuando se intenta actualizar una rent o una bike que no existe, o se pretende alquilar una bike no existente en la BBDD. También se usa con excepciones "temporales" como BikeNotYetAvailable y RentNotYetAvailable, pues queremos que no sean cacheadas por los servidores.
- 200 (OK): Corresponde con la respuesta satisfactoria de una query de búsqueda de Bikes o Rents.
- 204 (NO CONTENT): Indica la puntuación correcta de una rent, o la actualización de una bike (ya que se realiza satisfactoriamente pero no se crean recursos nuevos). Hemos escogido utilizar el 204 en vez de un 200 ya que un código OK incluiría la información actualizada de la correspondiente Bike o Rent en el body de la respuesta, que en este momento no parece algo necesario.
- 403 (FORBIDDEN): Utilizamos el código 403 para el caso particular en el que un usuario intenta puntuar una rent que ya había puntuado anteriormente (operación no permitida según la especificación de los casos de uso).

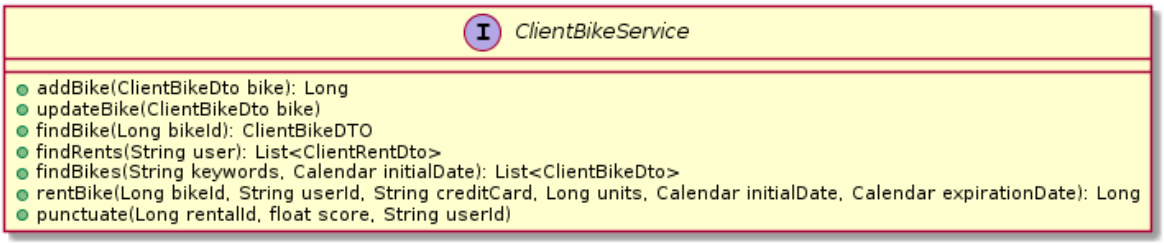
4. Aplicaciones Cliente

4.1 DTOs

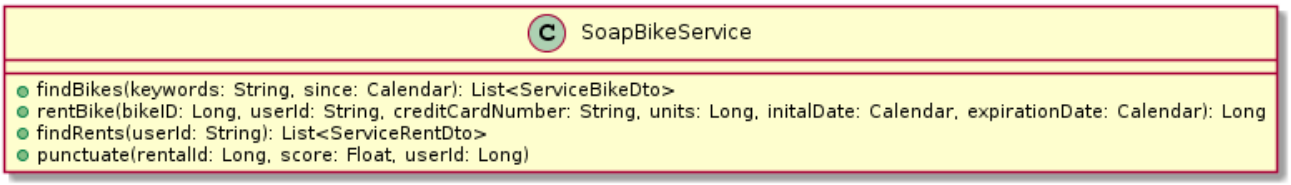


*Conforme a los DTO del Servicio, el DTO de Rent del cliente incluye el campo *duration* en vez de *expirationDate*, ya que hemos considerado que es más digerible para el usuario presentarlo de esta manera

4.2 Capa de Acceso al Servicio



5. SOAP



* En la implementación del servicio SOAP, utilizamos *@webparam(name = "whatever")* al defininir cada parámetro de entrada de los métodos para evitar que en los stubs del cliente generados por WSDL los argumentos aparezcan nombrados como *arg[0]*, *arg[1]*, etc. Los métodos mantienen su nomenclatura original del servicio.