Internet y Sistemas Distribuidos
Grupo : isd033

adrian.gvazquez
borja.padin
alberto.gonzalez.guimerans

Diciembre 2019

Contents

1 Introducción **3**

2 Capa modelo **3**
2.1 Entidades 3
2.2 DAOs 5
2.3 Fachadas 5

3 Capa Servicios **5**
3.1 DTOs 5
3.2 REST 6
3.2.1 Bikes 6
3.2.2 Rents 7

4 Aplicaciones cliente **9**
4.1 DTOs 9
4.2 Capa acceso al servicio 9

5 Errores conocidos **10**

1 Introducción

No se ha llevado a cabo el desarrollo de ningún trabajo tutelado para esta entrega.

2 Capa modelo

A continuación detallaremos las entidades persistentes de bicicletas y alquileres, con sus respectivos atributos y acciones para su correcto funcionamiento.

2.1 Entidades

Desde la primera iteración a esta se han llevado a cabo varios cambios en la capa modelo, tales como:

- Cambio del tipo del dato CreditCard de Long a String.
- Eliminación del atributo averageScore por totalScore y numberOfScores para poder realizar una media de forma correcta.
- También se ha añadido una puntuación a los alquileres que a su vez nos permite comprobar si ya ha sido puntuado.

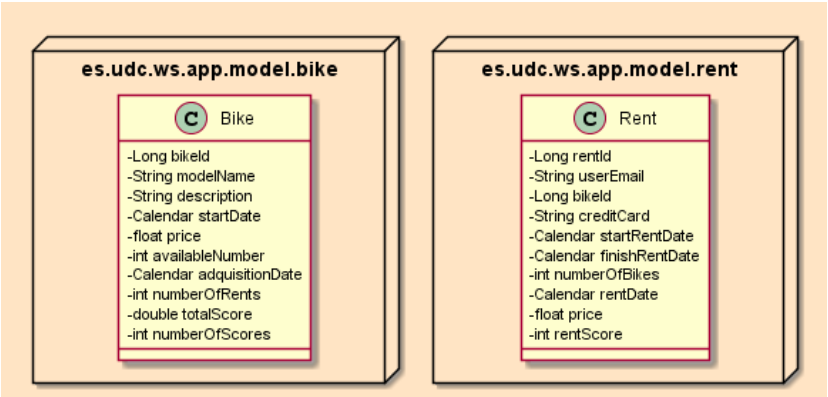

Figure 1: UML entidades modelo.

Bike: Presenta los siguientes atributos:

- **bikeId:** Identificador de la bicicleta.
- **modelName:** Cadena de caracteres que define el nombre de la bicicleta.
- **description:** Cadena que define características básicas que definen a la bicicleta.

- **startDate:** Fecha que indica desde que día se podrá empezar a alquilar.
- **price:** Dato que define el precio de la bicicleta por cada día que se quiera alquilar.
- **availableNumber:** Determinará cuántas bicicletas del modelo hay disponibles para su alquiler.
- **numberOfRents:** Determinará cuántas veces se ha realizado un alquiler de la bicicleta.
- **totalScore:** Valor medio de las puntuaciones que han recibido los alquileres una vez finalizados sobre el modelo de bicicleta.
- **numberOfScores:** Cuántas veces se ha puntuado una bicicleta, no todos los usuarios que han usado la bicicleta tienen porqué querer puntuar, y algunos tendrán un alquiler en curso; por eso no coincidirá con el número de veces alquilada.

Rent: Presenta los siguientes atributos:

- **rentId:** Identificador para los alquileres.
- **userEmail:** Identificación del usuario que a realizado el alquiler
- **bikeId:** Identificador del modelo de bicicleta que se ha alquilado.
- **creditCard:** Número de la tarjeta de crédito con la que se ha realizado el pago del alquiler.
- **startRentDate:** Fecha en la que dará lugar al comienzo del alquiler.
- **finishRentDate:** Fecha en la que finalizará el alquiler.
- **numberOfBikes:** Cantidad de bicicletas del mismo tipo que se han alquilado.
- **rentDate:** Fecha en la cual se realizó el alquiler.
- **price:** Precio por el cual se realizó el alquiler. Se guarda el valor final del mismo, pero se calcula mediante el precio actual del modelo y el número de bicicletas solicitadas.
- **rentScore:** Puntuación que le asigna el usuario al alquiler. Puede hacerlo una vez este ha terminado. Esto también ayudará a comprobar si el usuario ya ha evaluado el alquiler, para no permitir que se puntúe más de una vez el mismo alquiler.

2.2 DAOs

Diagramas UMLs con las distintas interfaces DAO de las entidades, donde se muestran las operaciones que se podrán realizar.

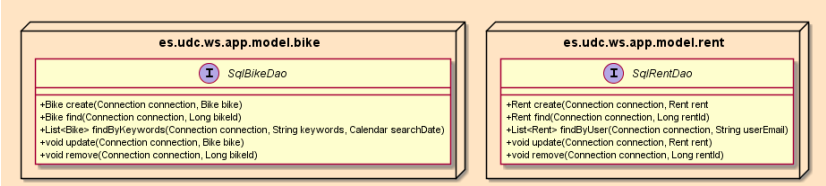

Figure 2: UML DAOs modelo.

Cabe destacar que los métodos remove solo se ven empleados en la capa de modelo para su utilización en los test.

2.3 Fachadas

Por simplicidad se ha llevado a cabo la implementación de una única fachada que contiene un método por cada caso de uso, pero se podría con fachadas separadas (una para los casos de uso bicicletas y otra para los de los alquileres). En esta sección se ha acabado de implementar la funcionalidad de puntuar una bici.

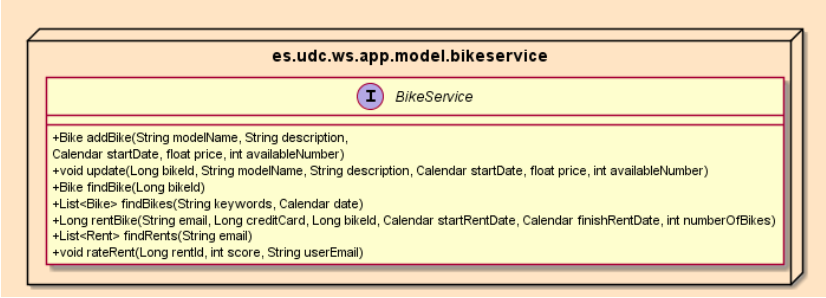

Figure 3: UML fachada modelo.

3 Capa Servicios

3.1 DTOs

Necesitamos dos DTOs para transferencia de datos entre la capa modelo y la capa servicio con ocultacción de datos necesarios en otras capas superiores. Los DTOs utilizados son:

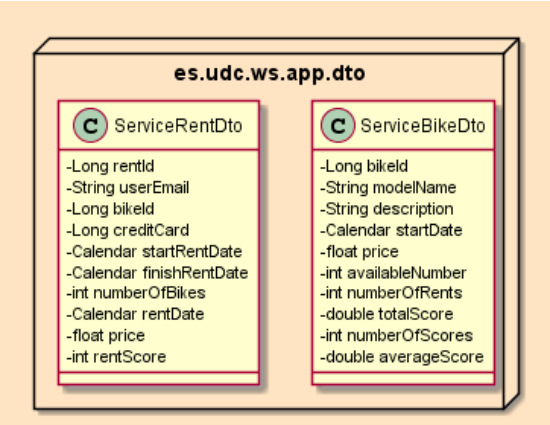

Figure 4: UML DTOs servicio.

3.2 REST

3.2.1 Bikes

addBike:

- URL simplificada del recurso: /bikes
- Método de invocación: POST.
- Parámetros o DTO de entrada: ClientBikeDto (admin).
- Posibles códigos HTTP de respuesta: 201, 400, 403.

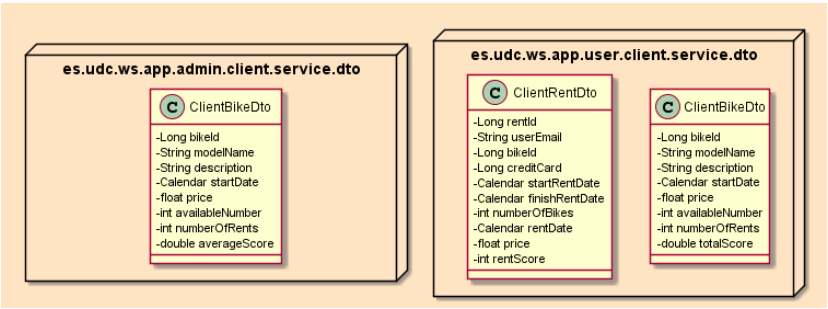

Figure 5: UML DTOs clientes.

update:

- URL simplificada del recurso: /bikes/{id}
- Método de invocación: PUT.

- Parámetros o DTO de entrada: ClientBikeDto (admin : Figura 5).
- Posibles códigos HTTP de respuesta: 204, 400, 403, 404.

findBike:

- URL simplificada del recurso: /bikes/{id}.
- Método de invocación: GET.
- Posibles códigos HTTP de respuesta: 200, 400.
- DTO devuelto: ServiceBikeDto (Figura 4).

findBikes:

- URL simplificada del recurso: /bikes/?startdate=dd/MM/yyy
  ó /bikes/?keywords={String de búsqueda}&startdate=dd/MM/yyy
- Método de invocación: GET.
- Posibles códigos HTTP de respuesta: 200, 400.
- DTO devuelto: ServiceBikeDto (Figura 4).

En esta parte se utiliza un conversor de fechas para saber si la fecha que se nos está pasando es una posible fecha (sólo se busca que se pueda obtener un Calendar de los parámetros de entrada) y en caso de no ser así poder devolver el error con información sobre el fallo.

3.2.2 Rents

rentBike:

- URL simplificada del recurso: /rents
- Método de invocación: POST.
- Parámetros o DTO de entrada: ClientRentDto(User: Figura 5).
- Posibles códigos HTTP de respuesta: 201, 400, 403, 404.

rateRent:

- URL simplificada del recurso:
  /rents/{id}?score={puntuación}&userEmail={userEmail}
- Método de invocación: PUT.
- Posibles códigos HTTP de respuesta: 204, 400.

findRents:

- URL simplificada del recurso: /rents/?userEmail={userEmail}
- Método de invocación: GET.
- Posibles códigos HTTP de respuesta: 200, 400, 404.
- DTO devuelto: ServiceRentDto (Figura 4).

Códigos HTTP y criterio de elección:

- 200 OK: Cuando devolvemos el elemento encontrado correctamente.
- 201 CREATED: Cuando creamos el elemento correctamente y devolvemos información suficiente para identificarlo.
- 204 NO CONTENT: Cuando realizamos la modificación correctamente y no se devuelve ningún dato.
- 400 BAD REQUEST: Cuando algunos de los elementos no contienen información correcta para lanzar las acciones correspondientes. Siempre que se vuelva a intentar realizar la consulta con estos errores la respuesta será la misma. Excepciones que lo devuelven:
  - ParsingBikeException, ParsingRentException ó ParsingException: Devuelto cuando no se ha podido crear el recurso por estar mal formada la estructura del DTO de entrada.
  - InputValidationException: Si no se ha podido crear o buscar el recurso por no corresponder los campos con los que se está creando o buscando con el tipo correcto.
  - InvalidRentPeriodException: Cuando se intenta hacer un alquiler de un período mayor a 15 días.
  - InvalidUserRateException: Si un usuario intenta puntuar un alquiler que no está asignado a su usuario.
- 403 FORBIDDEN: Cuando se intenta hacer una operación sobre un elemento que en este momento no está disponible. En caso de que más adelante vuelva a estar disponible la acción será correcta siempre que se cumplan las condiciones de la lógica de negocio.
  - NumberOfBikesException: Se intenta realizar un alquiler con un número superior de bicicletas de las que están disponibles en ese momento, o intentamos actualizar una bicicleta con un número inferior a 0 unidades.
  - UpdateReservedBikeException: Si se intenta actualizaar una bicicleta que todavía tiene alquileres en curso.

- 404 NOT FOUND: Cuando recibiendo una consulta bien formada y con los parámetros formados correctamente, somos incapaces de encontrar el elemento solicitado. En caso de que más adelante se pueda identificar correctamente el código resultante será distinto.
  - InstanceNotFoundException.
- 405 METHOD NOT ALLOWED: Cuando se intenta acceder a una operación que no está permitida. Código permanente, p.ej: a un usuario nunca le dejaremos borrar una reserva.
  - NotAllowedException.

4 Aplicaciones cliente

Esta parte es la que se encargará de comunicar al cliente/administrador con la capa servicio, de manera que podamos acceder al modelo.

4.1 DTOs

Como en el servicio, necesitamos métodos para encapsular el paso de información con ocultación de datos necesarios en otras capas superiores.

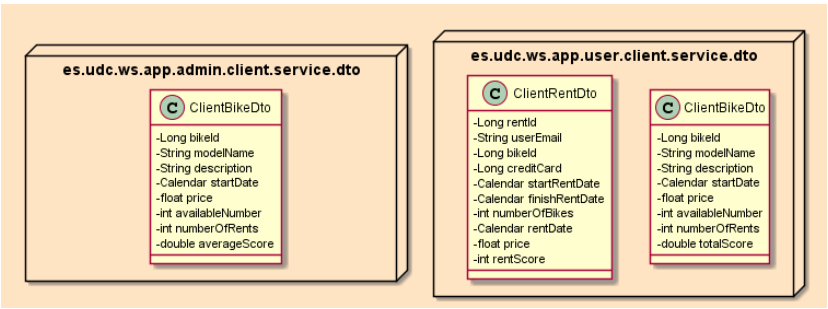

Figure 6: UML DTOs clientes.

4.2 Capa acceso al servicio

A continuación mostramos la interfaz con los casos de uso para las aplicaciones tanto de clientes como de administradores.

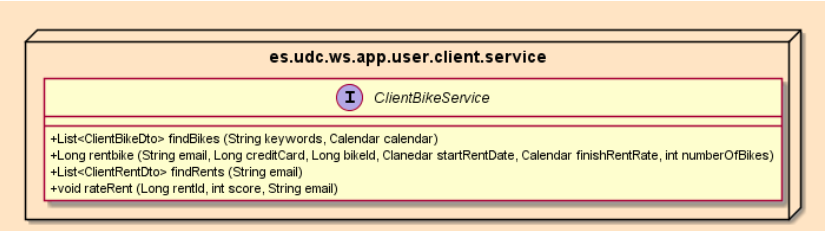

Figure 7: UML Acceso a Servicio Users.

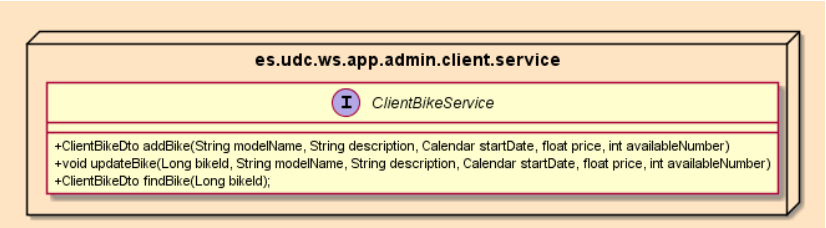

Figure 8: UML Acceso a Servicio Admins.

Usamos el patrón factoría para poder crear una instancia de ClientBikeService, sin que se tenga que conocer la clase que implementa la interfaz. La clase factoría se llama ClientBikeServiceFactory.

5 Errores conocidos