Memoria práctica ISD 2018/2019

Alejandro Romero Rivera

Laura Iglesias Gondar

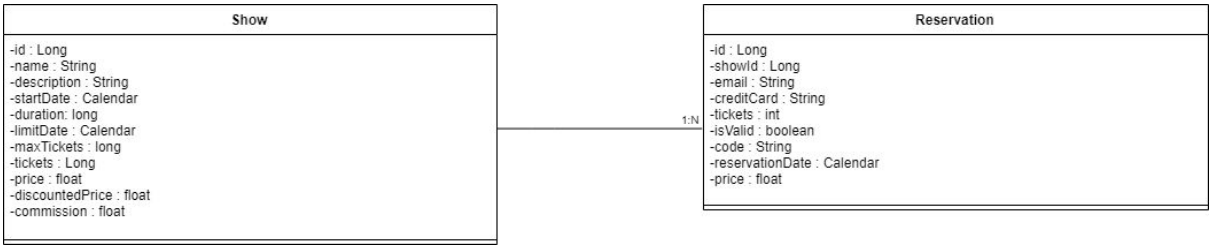
1. Introducción

Desarrollo de un servicio y clientes para la gestión de venta de entradas de espectáculos mediante la tecnología REST y SOAP.

Se ha realizado el trabajo tutelado de implementar la capa de Servicios usando un servicio SOAP.

2. Capa modelo

2.1. Entidades



* El número de entradas restantes es guardado en la entidad Show aunque pueda ser un atributo calculado por motivos de rendimiento y simplicidad del código, debido a que es un atributo que va a ser muy usado por requisitos del cliente.

2.2. DAOs

SqlReservationDao
+create(c : Connection, reservation : Reservation) : Reservation +update(c : Connection, reservation : Reservation) : void +remove(c : Connection, id : Long) : void +findByEmail(c : Connection, email : String) : List<Reservation> +findByCode(c : Connection, code : String) : Reservation

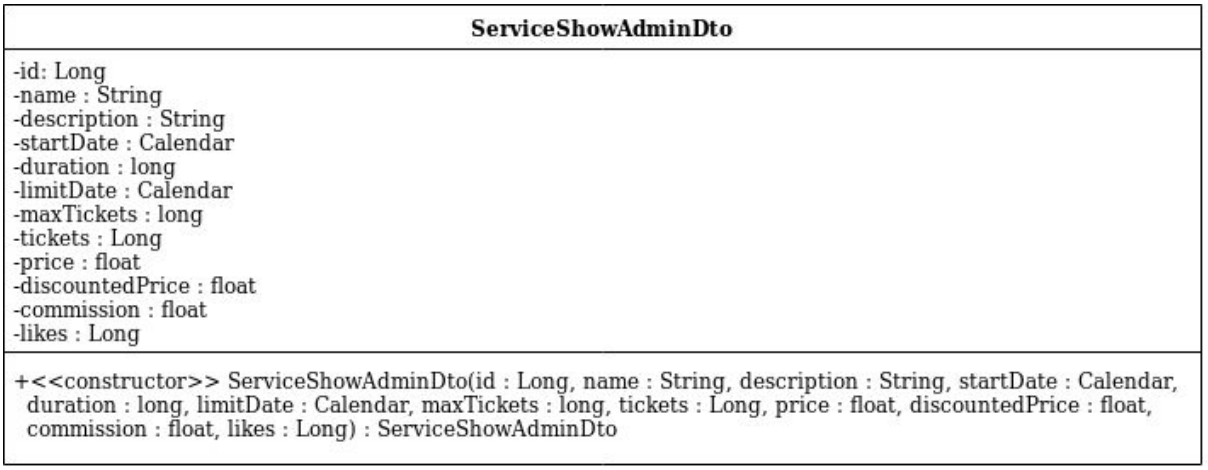
SqlShowDao
+create(c : Connection, show : Show) : Show +update(c : Connection, show : Show) : void +remove(c : Connection, id : Long) : void +find(c : Connection, id : Long) : Show +find(c : Connection, words : String, startDate : Calendar, endDate : Calendar) : List<Show>

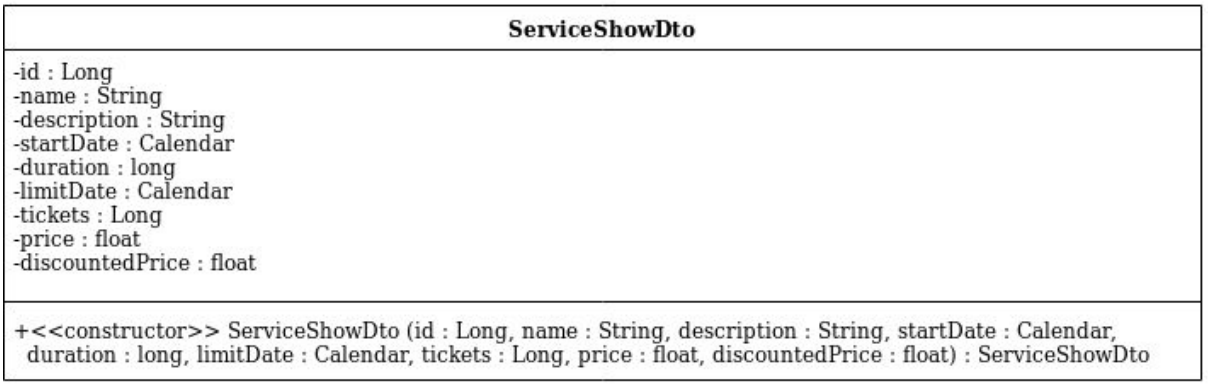
2.3. Fachadas

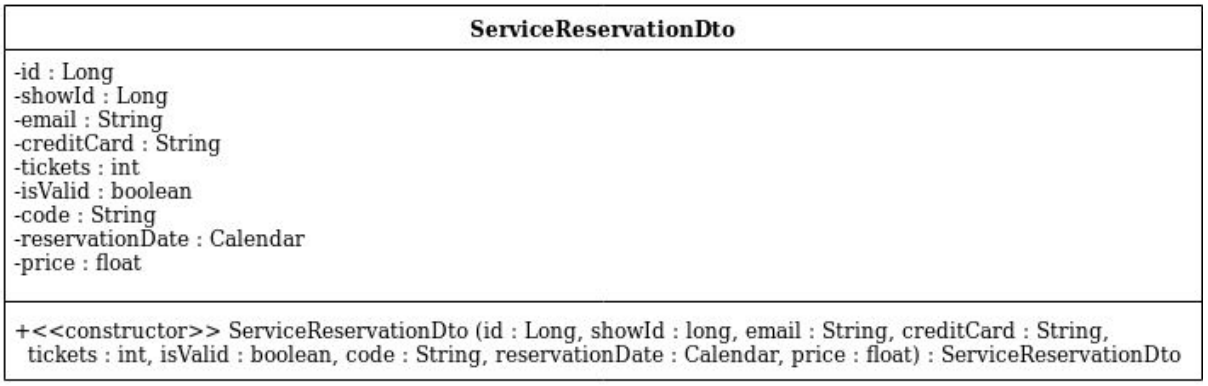
TicketSellerService
+createShow(show : Show) : Show +updateShow(show : Show) : void +findShow(id : Long) : Show +findShows(keywords: String, start : Calendar, end : Calendar) : List<Show> +buyTickets(showId : Long, email : String, cardNumber : String, count : int): Reservation +getUserReservations(email : String) : List<Reservation> +checkReservation(code : String, cardNumber : String) : void

3. Capa de servicios

3.1. DTOs







* ServiceShowAdminDto tiene el campo likes, relacionado con la red social, que puede ser nulo. En caso de ser nulo se considera un valor desconocido, ya sea por que no se ha podido conectar con la red social o no esté funcionando esta.

3.2. REST

Selección de códigos de error:

- Bad Request: Excepciones de peticiones mal formadas, siempre van a fallar no deben repetirse
- Forbidden: Excepciones permanentes de lógica de negocio
- Not Found: Excepciones con un identificador no encontrado o para excepciones temporales que no tengan algún otro código de error

Casos de uso:

Crear evento

Método: POST
URL: remote/ws-app-service/shows
DTO (Entrada): ServiceShowAdminDto
DTO (Salida): ServiceShowAdminDto
Respuestas:
400 Bad Request - Petición mal formada o datos mal formados
201 Created - Evento ha sido creado

Actualizar evento

Método: PUT
URL: remote/ws-app-service/shows/{id}
DTO (Entrada): ServiceShowAdminDto
Respuestas:
400 Bad Request - Petición mal formada o datos mal formados
403 Forbidden - Cuando se intenta reducir el precio cuando ya hay tickets vendidos
404 Not Found - No hay suficientes tickets disponibles o el identificador del evento no es válido
204 No Content - Evento actualizado correctamente

Buscar eventos por palabras clave

Método: GET
URL: remote/ws-app-service/shows
Parámetros:
keywords=<palabras separadas por espacios>
DTO (Salida): ServiceShowDto
Respuestas:
400 Bad Request - Petición mal formada
200 Ok - Petición completada correctamente

Obtener evento por id

Método: GET
URL: remote/ws-app-service/shows/{id}
DTO (Salida): ServiceShowAdminDto
Respuestas:
400 Bad Request - Petición mal formada
404 Not Found - Identificador no encontrado
200 Ok - Petición completada correctamente

Obtener reservas de un usuario

Método: GET
URL: remote/ws-app-service/reservations
Parámetros:
email=<email de usuario>
DTO (Salida): Lista de ServiceReservationDto
Respuestas:
400 Bad Request - Petición mal formada
200 Ok - Petición completada correctamente

Marcar reserva

Método: POST
URL: remote/ws-app-service/reservations/check
Parámetros:
code=<código de reserva>
creditCard=<número de tarjeta de crédito>
Respuestas:
400 Bad Request - Petición mal formada
403 Forbidden - La targeta de crédito no coincide o la reserva ya está marcada
404 Not Found - Codigo de reserva no existe
204 No Content - Petición completada correctamente

Comprar entradas

Método: POST
URL: remote/ws-app-service/reservations
Parámetros:
show=<código del evento>
email=<email del comprador>
creditCard=<número de tarjeta de crédito>
count=<número de entradas a comprar>
DTO (Salida): ServiceReservationDto
Respuestas:
400 Bad Request - Petición mal formada
404 Not Found - No hay suficientes tickets disponibles o fecha límite excedida
201 Created - La reserva ha sido creado

* Un cambio importante respecto a la anterior entrega ha sido la corrección de los DTOs devueltos. Ahora los DTOs de administracion son devueltos al cliente de administración (taquilleros) y los DTOs de clientes al cliente para usuarios normales.

4. Aplicaciones cliente

4.1. DTOs

ClientAdminShowDto
-id : Long -name : String -description : String -startDate : Calendar -duration : long -limitDate : Calendar -maxTickets : long -tickets : long -price : float -discountedPrice : float -commission : float -likes : Long
+<<constructor>> ClientAdminShowDto (id : Long, name : String, description : String, startDate : Calendar, duration : long, limitDate : Calendar, maxTickets : long, tickets : long, price : float, discountedPrice : float, commission : float, likes : Long) : ClientAdminShowDto

ClientReservationDto
-id : Long -showId : Long -email : String -creditCard : String -tickets : int -isValid : boolean -code : String -reservationDate : Calendar -price : float
+<<constructor>> ClientReservationDto (id : Long, showId : long, email : String, creditCard : String, tickets : int, isValid : boolean, code : String, reservationDate : Calendar, price : float) : ClientReservationDto

ClientShowDto
-id : Long -name : String -description : String -startDate : Calendar -endDate : Calendar -limitDate : Calendar -tickets: long -price : float -discountedPrice : float
+<<constructor>> ClientShowDto (id : Long, name : String, description : String, startDate : Calendar, endDate : Calendar, limitDate : Calendar, tickets : long, price : float, discountedPrice : float) : ClientShowDto

4.2. Capa de acceso al servicio

ClientAdminTicketSellerService
+createShow(show : ClientAdminShowDto) : ClientAdminShowDto +updateShow(show : ClientAdminShowDto) : void +findShow(id : Long) : ClientAdminShowDto +checkReservation(code : String, cardNumber : String) : void

ClientTicketSellerService
+findShows(keywords: String) : List<ClientShowDto> +buyTickets(showId : long, email : String, cardNumber : String, count : int): ClientReservationDto +getUserReservations(email : String) : List<ClientReservationDto>

* Ahora el cliente de administración recive DTOs de administración

5. Trabajos tutelados

5.1 SOAP

SoapTicketSellerService
+ createShow(show : ServiceShowAdminDto) : ServiceShowDto
+ updateShow(show : ServiceShowAdminDto)
+ findShow(id : long) : ServiceShowDto
+ findShows(keywords : String) : List<ServiceShowDto>
+ buyTickets(showId : long, email : String, creditCard : String, count : int) : ServiceReservationDto
+ getUserReservations(email : String) :List<ServiceReservationDto>
+ checkReservation(code : String, creditCard : String)

Equivalente al servicio REST pero implementa sus propias excepciones que hay que convertir tanto desde el servicio como desde los clientes.

5.2 Integración con una red social

Nota: Lista de fallos en el servicio de la red social y su documentación encontrados mientra se realizaba la práctica

1. El encabezado de autenticación es: Authorization no Authentication

2. No todas las excepciones poseen el campo “details”, sin embargo en la documentación así aparece

3. Inconsistencias en el campo timestamp entre excepciones y posts, algunos son string otros enteros

4. El borrado de un post genera un error 403 si el id no existe en vez de un 404

5. Límite de 200 caracteres en el que no caben los datos requeridos bien formateados, y no se le puede dar formato a estos

Decisiones tomadas a tener en cuenta:

1. Para evitar duplicar código compartido tanto de lógica como de la red social entre el servicio REST y SOAP, y simplificar la implementación de estos, hemos decidido crear una nueva capa en el servicio que separa el servicio de la capa modelo de los de la capa REST y SOAP

2. La red social se ha implementado como un servicio aparte con su propio paquete con una interfaz genérica de la que se obtienen instancias de la misma forma que la capa modelo, de esta forma es trivial cambiar la implementación de la red social sin afectar a ningún otro componente

3. Para mantener el servicio de la red social stateless sin tener que añadir un nuevo servicio al modelo o cambiarlo, hemos optado por mantener el token en memoria y la búsqueda de eventos es realizada por contenido y no por acceso directo mediante postId

4. Se ha implementado el envío de likes solo para los administradores (los únicos que obtienen espectáculos por id) y no para los usuarios normales

Fachada de la nueva capa que separa REST, SOAP del modelo y la red social:

TicketSellerProxyService
+ createShow(show : ServiceShowAdminDto) : ServiceShowAdminDto
+ updateShow(show : ServiceShowAdminDto)
+ findShow(id : long) : ServiceShowAdminDto
+ checkReservation(code : String, creditCard : String)
+ findShows(keywods : String) : List<ServiceShowDto>
+ buyTickets(showId : long, email : String, creditCard : String, count : int) : ServiceReservationDto
+ getUserReservations(email : String) : List<ServiceReservationDto>

Fachada de la red social:

SocialNetworkService
+ publishShow(show : Show) : SocialNetworkShowPost
+ updateShow(show : Show) : SocialNetworkShowPost
+ getShow(showId : long) : SocialNetworkShowPost
+ commentShow(show : Show, comment : String)

Entidades de la red social:

SocialNetworkShowPost
-postId : Long -likes : long -showId: Long -showName : String -showDescription : String -showStartDate : Calendar -showDuration : long -showLimitDate : Calendar -showMaxTickets : long -showTickets : long -showPrice : float -showDiscountedPrice : float

6. Errores conocidos

En el apartado de la red social, la implementación actual de esta no accede al modelo para guardar su estado, esta decisión ha sido tomada para no complicar más la práctica, de forma que para mantenerla como un servicio stateless se ha sacrificado mucho el rendimiento con respecto al número de conexiones y el tipo de estas respecto a la base de datos.
Un ejemplo de esto se puede ver en el apartado de buscar un espectáculo en la red social, la actual implementación busca por contenido el identificador de un espectáculo aprovechando el hecho de que el texto de un post no puede ser repetido.