Trabajo Práctico Especial 1

Álvarez Escalante, Tomás (60127)
tomalvarez@itba.edu.ar

Ferreiro, Lucas Agustín (61595)
lferreiro@itba.edu.ar

Gómez Kiss, Román (61003)
romgomez@itba.edu.ar

14/09/2023

Reservas de Atracciones de Parques Temáticos

Grupo 5

72.42 Programación de Objetos Distribuidos

Instituto Tecnológico de Buenos Aires

1. Diseño e implementación de los servicios

La arquitectura del proyecto consta de 3 módulos: cliente, api y server.

1.1. API

En lo que respecta al módulo API, solo se incluyen los archivos con extensión *.proto*, que se emplean para definir los mensajes y servicios utilizados en nuestra aplicación gRPC.

1.2. Cliente

Se utiliza la clase abstracta ***AbstractParams***, que nos ayuda a compartir el comportamiento repetido de los parámetros que recibe cada cliente. También se implementó la interfaz funcional ***Action*** con el método *execute*, para aprovechar el comportamiento común entre las distintas acciones a realizar desde el lado del cliente. Luego, cada uno de los cuatro clientes consta de su respectiva sub carpeta, donde cada uno cuenta con la implementación de su respectivo parser y acciones.

1.3. Server

Finalmente, el módulo del servidor se encarga de alojar la lógica de negocio de nuestros servicios, así como una especie de clase de persistencia llamada ***ParkData***, la cual proporciona una lista de reservas, atracciones y capacidades por sesión, dado que no contamos con una base de datos. Los servicios consumen esta clase para realizar todas las acciones solicitadas para el trabajo, como agregar atracciones, ticket, realizar reservas, entre otras. Posteriormente, la clase ***Server*** se encarga de definir el puerto y manejar todos los aspectos necesarios para poner en funcionamiento nuestro servicio y permitir que el cliente lo consuma.

En lo que respecta a la implementación de los servicios en sí, disponemos de cuatro servicios que corresponden a los tipos de funcionalidades especificados en el enunciado. En cada una de las respectivas implementaciones, se utilizan objetos *StreamObserver* de gRPC para controlar el estado de la transmisión y gestionar las respuestas del servidor al cliente, además de encargarse de la lógica propia de la gestión de la agregación en las respectivas colecciones de nuestra lógica de negocio.

Posteriormente, en el módulo se describen las excepciones que pueden surgir durante las operaciones, siendo cada una específica de un posible problema en dichas operaciones. La gestión de estas excepciones se lleva a cabo mediante la clase ***GlobalExceptionHandlerInterceptor***.

2. Criterios aplicados para el trabajo concurrente

A modo general, se priorizó conseguir una buena sincronización de las operaciones de escritura en las distintas colecciones utilizadas en la clase ***ParkData***.

2.1. ConcurrentHashMap

En primer lugar, se hizo uso de *ConcurrentHashMap* para la gestión de nuestras colecciones en la clase ***ParkData***, el cual nos otorga atomicidad en las operaciones de agregación, edición y eliminación. También nos facilita las operaciones de lectura, ya que la misma es no bloqueante, y a su vez permite escrituras concurrentes. Además, nos permite una segura iteración sobre la colección, ya que esta, a diferencia de un *HashMap*, no lanza excepciones del tipo *ConcurrentModificationException*.

2.2. Synchronized

Se hace uso del operador *synchronized* al momento de agregar nuevas atracciones y tickets, para que dichos agregados en las distintas colecciones se realicen de forma atómica.

2.3. Semáforos

También se optó por agregar una 'capa extra' de sincronización utilizando la siguiente colección:

```
1         private final Map<ServerAttraction, Map<Integer, Semaphore>>
  semaphores = new ConcurrentHashMap<>();
2
```

El uso de esta colección nos permite, por atracción y día, utilizar un semáforo para garantizar una protección eficiente al momento operar sobre la colección de reservas, más específicamente, al momento de reubicar las reservas cuando se asigna la capacidad de una atracción en cierto día y al momento de operar en la colección de reservas.

3. Potenciales puntos de mejora y/o expansión

Como mejoras del proyecto, se considera contar en un futuro con una base de datos para almacenar la información de las atracciones y las reservas, ya que usamos colecciones del tipo:

```
1    private final Map<ServerAttraction, Map<Integer, Map<LocalTime, List
     <ServerBooking>>>> bookings = new ConcurrentHashMap<>();
2
```

El uso de estas colecciones implica un manejo no tan sencillo para gestionar las operaciones necesarias para con el trabajo, por lo que el uso de una base de datos mejoraría la persistencia de la información del parque, puesto que al reiniciar el sistema se comienza de cero la operación del mismo.

Otro punto de mejora podría radicar en que los semáforos sean horario-específicos, a diferencia de nuestra implementación, en la cual solo tenemos en cuenta el día. Además, de esta forma también se podrían realizar algoritmos más específicos y poder atender más operaciones de forma simultánea.