Programación de Objetos Distribuidos
Trabajo Práctico
gRPC - III

Ejercicio 1

Se desea implementar un **servicio gRPC para una biblioteca virtual** a la cual se le puede pedir prestado y devolver libros mediante la siguiente interfaz:

```
syntax = "proto3";

package library;

import "google/protobuf/wrappers.proto";
import "google/protobuf/empty.proto";

service LibraryService {
 rpc ListBooks(google.protobuf.Empty) returns (ListBooksResponse);
 rpc LendBook(google.protobuf.StringValue) returns (LendBookResponse);
 rpc ReturnBook(LendBookResponse) returns (google.protobuf.Empty);
}

message ListBooksResponse {
 repeated string values = 1;
}

message LendBookResponse {
 string isbn = 1;
 string title = 2;
 string date = 3;
 message AuthorResponse {
   string firstName = 1;
   string lastName = 2;
 }
 AuthorResponse author = 4;
}
```

Requisitos Funcionales:

- La biblioteca tiene una cantidad fija libros disponibles cuyo ese stock no cambia mientras el servicio esté levantando. Ejemplo de la inicialización:

```
Collection<String[]> books = List.of(
        new String[] { "3", "978-0345533487", "A Knight of the Seven
Kingdoms", "2015-10-06", "Martin", "George R. R." },
        new String[] { "4", "978-0441294671", "God Emperor of Dune",
"1987-06-15", "Herbert", "Frank" },
          new String[] { "2", "978-0451210845", "The Gunslinger",
"2003-07-01", "King", "Stephen" },
           new String[] { "5", "978-0307292063", "The Foundation
Trilogy", "2011-11-25", "Asimov", "Isaac" },
         new String[] { "4", "978-0765351494", "Sandworms of Dune",
"2008-07-01", "Herbert", "Brian" },
            new String[] { "1", "978-0307743657", "The Shining",
"2012-06-26", "King", "Stephen" },
        new String[] { "2", "978-0553328257", "The Complete Sherlock
Holmes", "1986-10-01", "Conan Doyle", "Arthur" });
Servant servant = new Servant(books);
```

- Cada libro está identificado por su ISBN (una cadena alfanumérica) debe poseer getters para este valor además del título (String), fecha de publicación (LocalDate) y autor que es un objeto que contiene getters para nombre (String) y apellido (String).
- La biblioteca puede recibir más de un ejemplar del mismo libro para prestar.
- El sistema no tiene usuarios, ni autenticación.
- Los métodos del servicio deben cumplir:
    - ***ListBooks***:
        - Devuelve un listado de los libros que existen en la biblioteca con el formato ISBN-TITULO. Para la respuesta no tiene en cuenta si los libros están prestados o si tienen más de un ejemplar.
        - No tiene condiciones de error.
    - ***LendBook***:
        - Retorna un mensaje con todos los campos relevantes del libro en caso de que el un ejemplar del libro correspondiente al nombre recibido esté disponible para prestar
        - Si el ISBN no es de un libro válido se lo considera un error.
        - Si es un libro válido pero no cuenta con stock para prestarlo se lo considera un error.
    - ***ReturnBook***:
        - Si el libro fue prestado por el servicio queda inmediatamente disponible para prestarlo de nuevo.
        - En caso de que el libro no sea válido o no fue prestado se lo considera un error.

Para ello se pide **implementar un Servant** que responda a los métodos del servicio, un Server que publique el Servant anterior y un Client que consuma el servicio publicado utilizando un stub bloqueante.

Requisitos No Funcionales:

- Debe soportar correr en un entorno thread safe y con usuarios concurrentes. Además de un cliente que pueda ejecutarse en otra computadora física.
- Se debe respetar una separación en tres módulos: api, server y client. Para ello utilizar el arquetipo provisto por la cátedra.
- La interfaz del servicio no puede ser modificada.
- El diseño de estructuras de datos y el manejo de errores y demás casos no especificados quedan a cargo del programador.
- No hace falta persistir el estado de libros y préstamos de una publicación a otra del servicio.

Ejercicio 2

Una empresa que organiza recitales internacionales se quiere asegurar una cierta asistencia antes de confirmarle el recital al artista y a los interesados en asistir.

Para lo cual piensa a los recitales con los siguientes elementos:

- Un nombre para identificarlo
- Número de confirmación: Una vez que la cantidad de entradas reservadas alcanza este número de recital queda confirmado y se confirman las entradas reservadas hasta ese momento y las que se vendan a posteriori.
- Capacidad máxima: El número de entradas que pueden venderse
- Entradas vip: Es una cantidad de entradas preferenciales que se le otorgaran a los primeros que se registraron para el recital (una entrada por cada registro). Una vez que se agotan al resto se le otorgan entradas regulares.

Y los recitales tienen los siguientes estados:

- **A confirmar**: Las entradas vendidas aún no superan el número de confirmación, por lo que las mismas solo fueron reservadas
- **Confirmado**: Cuando se alcanza el número de confirmación, el recital queda confirmado y las entradas reservadas se confirman. Cuando una entrada es confirmada se le envía al cliente un identificador de la misma para que la pueda cambiar. Para beneficiar a los que participan de este sistema existe una cantidad de entradas vip que se otorgan en el orden de reserva/compra hasta agotarse. Una vez confirmado el recital, nuevas compras se confirman directamente.
- **Agotado**: Cuando las ventas alcanzaron la capacidad máxima del recital
- **Cancelado**: El organizador decidió cancelar el recital, por lo cual se notifica a todos los que tengan entradas reservadas/confirmadas de la noticia.

Los **servicios gRPC** que se ofrecen son:

- **RecitalService:** para poder crear y cancelar recitales.

```
syntax = "proto3";

package recitals;

import "google/protobuf/wrappers.proto";
import "google/protobuf/empty.proto";
```

```
service RecitalService {
 rpc createRecital(RecitalCreateRequest) returns (google.protobuf.Empty);
 rpc cancelRecital(google.protobuf.StringValue) returns (google.protobuf.Empty);
}

message RecitalCreateRequest {
 string name = 1;
 uint32 confirmQty = 2;
 uint32 vipQty = 3;
 uint32 maxCapacity = 4;
}
```

- **TicketsService**: Para poder comprar la entrada, utilizando notificaciones asincrónicas:

```
syntax = "proto3";

package recitals;

import "google/protobuf/wrappers.proto";

service TicketsService {
 rpc requestTicket(google.protobuf.StringValue) returns (stream TicketNotification);
}

message TicketNotification {
 NotificationEvent event = 1;
 string content = 2;
}

enum NotificationEvent {
 UNDEFINED_EVENT = 0;
 TICKET_RESERVED = 1;
 VIP_TICKET_CONFIRMED = 2;
 TICKET_CONFIRMED = 3;
 SOLD_OUT_RECITAL = 4;
 CANCELLED_RECITAL = 5;
}
```

Para ello se pide **implementar dos servants (uno para cada servicio gRPC)**, un Server que publique los servants anteriores y un Client que consuma los servicios publicados.

Requisitos No Funcionales:

- Debe soportar correr en un entorno thread safe y con usuarios concurrentes. Además de un cliente que pueda ejecutarse en otra computadora física.

- Se debe respetar una separación en tres módulos: api, server y client. Para ello utilizar el arquetipo provisto por la cátedra.
- La interfaz del servicio no puede ser modificada.
- Debe implementar **un repositorio de recitales**, con al menos los métodos para agregar un recital y para obtenerlo o cancelarlo a partir de su nombre. Ambos servants deberán depender de este repositorio para consultar y persistir (en memoria) la información correspondiente de los recitales.
- Debe implementar **un interceptor** del lado del servidor, que se encargue de capturar los errores esperados lanzados por los servants y/o el repositorio.
- El diseño de estructuras de datos y el manejo de errores y demás casos no especificados queda a cargo del programador.
- No hace falta persistir el estado de los recitales y entradas de una publicación a otra del servicio.