Trabajo Práctico Especial 1:

Tickets de Vuelo

31 de Agosto de 2022

72.42 - Programación de Objetos Distribuidos

Arce, Julián Francisco - 60509

Catalán, Roberto José - 59174

Dell'Isola, Lucas - 58025

Pecile, Gian Luca - 59235

Torrusio, Lucía - 59489

Decisiones de diseño e implementación de los servicios

Para el desarrollo se optó por una dinámica de *Test Driven Design* (TDD) donde para cada *feature* se realizaron sus *tests* correspondientes en el desarrollo y estos debían estar presentes para ser mergeados a la branch principal.

Cada servicio es una clase independiente, instanciada a la hora de iniciar el servidor. Para compartir el estado de la aplicación se implementó inyección de dependencias, pasando por constructor los datos necesarios a cada servicio para poder realizar sus tareas. El uso de inyección de dependencias además facilita el testeo unitario de cada servicio.

El sistema de notificaciones se implementa a través del patrón *observer*, donde se cuenta con una clase `EventsManager`, inyectada a los servicios, que es quien se encarga de realizar las notificaciones correspondientes en los momentos correspondientes. El `EventsManager` es transversal a los servicios e independiente del servicio de notificaciones. Éste último se encarga de suscribir clientes al `EventsManager`.

Para aplicar filtros a la hora de obtener el `SeatMap`, se utiliza una interfaz común `Criteria`, aplicando el patrón *strategy*, permitiendo aplicar distintos filtros de acuerdo a lo pedido por el cliente, y proveyendo código mantenible y extensible.

Criterios aplicados para el trabajo concurrente

Se hace uso de mapas y listas sincrónicas implementadas por java para mantener el estado de la aplicación. Permiten operaciones atómicas, y sólo se debe realizar la sincronización manualmente a la hora de realizar iteraciones sobre la colección.

En el caso de iterar sobre los vuelos o los tickets de un vuelo, se realiza el bloqueado de la colección hasta finalizar su ejecución, liberando el recurso bloqueado lo antes posible, realizando el resto de las operaciones de manera no bloqueante permitiendo así a otras funciones hacer uso de los recursos compartidos.

Potenciales puntos de mejora y/o expansión

En primera instancia una posible mejora para garantizar mejor usabilidad al proyecto es una interfaz gráfica (GUI) para el cliente utilizando javaFX ya que en muchos casos se pueden producir errores de tipeo con la interfaz por línea de comando.

A su vez, se consideró la posibilidad de modificar el diseño del modelo Flight para guardar la información de los asientos en un mapa. De esta manera no se tendría que consultar constantemente la lista de tickets para informarse de qué asientos se encuentran ocupados. Por lo que se centralizaría esta información y habría que administrar menos componentes en la sincronización, reduciendo la probabilidad de error.

Pensando en las típicas funcionalidades que existen en sistemas de vuelos, se podría implementar la posibilidad de administrar múltiples aerolíneas. Si bien se cuenta con la administración de diferentes vuelos, se precisa la información de qué aerolínea controla el mismo. De esta forma cambiaría la lógica de muchas de las funciones, como el “reschedule” a un nuevo vuelo, el mismo deberá ser de la misma empresa.