Test de Entrevista Técnica

Desarrollador: Patricio Emiliano Sartore

Fecha de Inicio: 3 de Enero del 2020

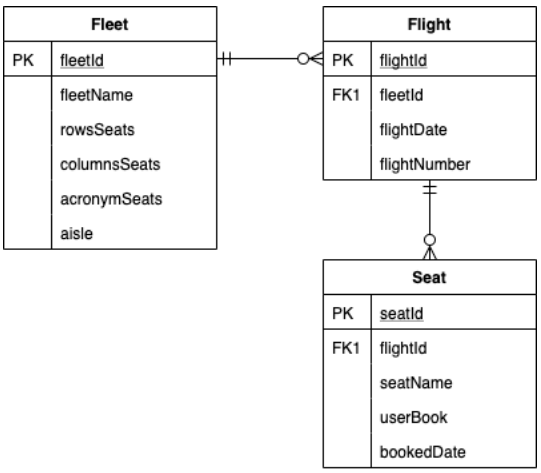
Fecha de Fin: 7 de Enero del 2020

En el siguiente documento se explicará cuales fueron las consideraciones que se tuvieron en cuenta al momento de desarrollar el test, como así las explicaciones de las tecnologías elegidas, las decisiones tomadas y las mejoras que se pueden llevar a cabo.

Tecnologías:

BackEnd y FrontEnd:
Como lo pide en el enunciado del test, se ha utilizado Node.js como motor para el desarrollo del backend, y ReactJS para el desarrollo del frontend en conjunto con el kit de desarrollo CSS llamado Material-UI. La elección de Material-UI se debe a los años que se encuentra en desarrollo y a las mejoras que han presentado con el tiempo, al punto tal, de aportar significativamente legibilidad al código, facilidad del uso de diferentes componentes que vienen incorporados en la librería, y una gran API de documentación y soporte por parte la comunidad de desarrolladores (https://material-ui.com/)

MySql:
La elección de la base de datos ha quedado a mi consideración, es por ello que he decidido utilizar MySQL con Sequelize como ORM para realizar las consultas a la BD. La elección de estas tecnologías radica en el hecho de que se pudieron distinguir 3 grandes entidades necesarias para poder hacer reservas de asientos, y con ellas, relaciones entre sí que se ajustan mejor a un modelo de base de datos relacional. El diagrama de la BD resultante es:



En el diagrama se puede ver como los asientos se encuentran directamente relacionados con un vuelo en especifico y no con un avión en particular. Esto está dado a que se puede tener el mismo número de asiento para vuelos diferentes y en diferentes fechas, y es por ello, que una asociación directa con un avión sería incorrecta. Si bien en el enunciado no se habla de un vuelo, me pareció necesario representar el mismo en el modelo.
A medida que se van realizando las diferentes reservas, los asientos se irán creando en la BD con su vuelo correspondiente.

Como mejora, se puede tener una tabla usuarios y relacionar el usuario que realizó la reserva con los asientos reservados, pero esto ya se escapa de lo requerido por el enunciado.

Docker:
Otra tecnología muy importante para poder hacer uso de los servicios de backend, frontend y base de datos, es la incorporación de Docker. Esta tecnología es necesaria para poder dividir cada uno de los componentes mencionados en diferentes contenedores que convivan independientemente. Las ventajas que brinda Docker es poder poder definir un sistema de buildeo de diferentes imágenes para así poder ejecutarlas sin la necesidad de ningún setup engorroso por parte del desarrollador y sin tener que executar cada servicio de forma individual.
Por medio de docker compose y un único comando, podemos definir como van a ejecutarse los contenedores, cuales van a ser las dependencias de los mismos y abstraernos por completo. Para este test, docker tiene la tarea de levantar tanto el backend y el frontend, como así un servidor de MySQL con dump de una base de datos de prueba que persiste por mas que se borren los contenedores.

Algoritmo:

Se ha pedido que se desarrolle un algoritmo que presenta 4 casos diferentes. Estos casos actúan de una manera u otra dependiendo de la disponibilidad de los asientos en el avión.

La única consideración que se ha tenido en cuenta en dicho algoritmo, es en segundo caso. En dicho caso se habla de que los asientos se deben reservar equilibrados por filas, es decir, que quede la misma cantidad de asientos por fila. Mostrándolo en un ejemplo, en el caso de que se quieran reservar 4 asientos y el número de asientos por fila sin cruzar el pasillo sea de 3, el resultado sería [A1,B1,A2,B2].
Se ha encontrado una problemática cuando el numero de asientos es impar y mayor que el número de asientos por fila sin cruzar el pasillo. Suponiendo el caso de que el avión posea 3 asientos por fila sin cruzar el pasillo y el número de asientos a reservar sea 5 (es el mismo caso con 7), el resultado sería 3 filas de las cuales no existe combinación de forma tal que todas ellas posean el mismo número de asientos reservados. Existe una excepción y es el caso que se reserve un asiento por fila, es decir, que el resultado sea [A1,A2,A3,A4,A5].
Por lo tanto, en caso de que se quiera reservar un número de asientos impar mayor que el número de asientos de una fila sin cruzar el pasillo, se reservarán todos en una única columna.

El algoritmo desarrollado no es del todo óptimo, ya que en el peor de los casos recorre una vez todos los asientos del avión por cada caso que se quiera corroborar. Se puede desarrollar un algoritmo mas eficiente, que a medida que recorra cada asiento del avión, vaya calculando cada uno de los casos. Por cuestiones de tiempo, no he podido desarrollar dicha mejora, pero soy consiente de que puede realizarse un algoritmo mucho mas eficiente. Vale aclarar que, si bien en el peor de los casos se recorre en total 4 veces todos los asientos del avión para buscar asientos libres, el algoritmo tiene

un orden de tiempo de ejecución lineal (4*O(N)) = (O(N)). No obstante, en un uso masivo como lo es producción, estas iteraciones adicionales pueden afectar la performance del server.

Existe la posibilidad de que 2 o más usuarios quieran reservar al mismo tiempo una determinada cantidad de asientos para el mismo vuelo. Actualmente el algoritmo no contempla este caso, pero se debería hacer una mejora para manejar estas peticiones concurrentes.
Una manera de poder hacerlo es incorporando Redis. La idea de Redis, es escribir en información que pueda ser de ayuda en memoria, siendo accesible casi instantáneamente. Por lo tanto, se podría evitar estos finales de carrera poniendo a Redis como un ward al principio de la ejecución de la lógica de reservas de asientos cuando llega la primera petición del primer usuario al servidor y dejando que solo se atienda esta petición. De esta manera, solo un usuario podrá reservar asientos a la vez, protegiendo y evitando de que varios usuarios estén escribiendo en la base de datos al mismo tiempo y se pueda llegarse a posibles inconsistencias.

Consideraciones:

A continuación, se va a detallar una lista de tareas las cuales no se han podido realizar, o bien, pueden llevarse a cabo mejoras sobre las mismas. Ellas son:

- No se han podido realizar test de ningún tipo por falta de tiempo, ya sea de UI como test unitarios.
- Si bien el kit de desarrollo brinda muchas herramientas para poder hacer responsive las diferentes aplicaciones, no se ha podido lograr que la aplicación sea totalmente responsive por falta de tiempo.
- El backend fue desarrollado haciendo uso de javascript. Se podría integrar typescript para poder hacer uno de tipos de datos en los request y response del servidor. Esto trae como ventaja de que tanto el frontend como el backend maneje los mismos tipos de datos y las interfaces entre los mismo sea mas clara, y así lograr un desarrollo individualizado. Los desarrolladores solamente deben saber cuales son los datos de entrada y cuales son los tipos de datos de salida sin conocer la implementación de la lógica.
- Se ha realizado solo el algoritmo de reserva de asientos. La creación de un nuevo avión o el cambio de un asiento de la reserva no se a podido llevar acabo por falta de tiempo.

Ejecución de la aplicación:

Los pasos para correr la aplicación se encuentran en el readme del proyecto. De cualquier manera se deja una breve explicación de como correr la aplicación. Como requisito se debe tener docker instalado.

Para correr la aplicación se debe ejecutar por línea de comandos:

```
docker-compose up app
```

En caso de que se quiera ver los logs del backend y del servidor de mysql y se quiere acceder por phpmyadmin a la base de datos, ejecutar el siguiente comando:

```
docker-compose up app backend mysql phpmyadmin
```

Las URL's de acceso al proyecto son:

- Frontend: http://localhost:3000
- Backend: http://localhost:5000
- Phpmyadmin: http://localhost:8000