Airline System

Eduardo Duarte – up202004999

Ian Gomes – up202000707

Igor Diniz – up202000162

O problema

- A recém nascida IEI Airlines está a entrar no mercado e precisa de um sistema com tecnologia topo de gama para gerir os seus recursos de diversos tipos
- Pretende-se que esse sistema faça gestão de voos, aviões, passageiros, passagens e bagagens, para além de apresentar informações sobre transportes de todos os aeroportos para os clientes da companhia aérea

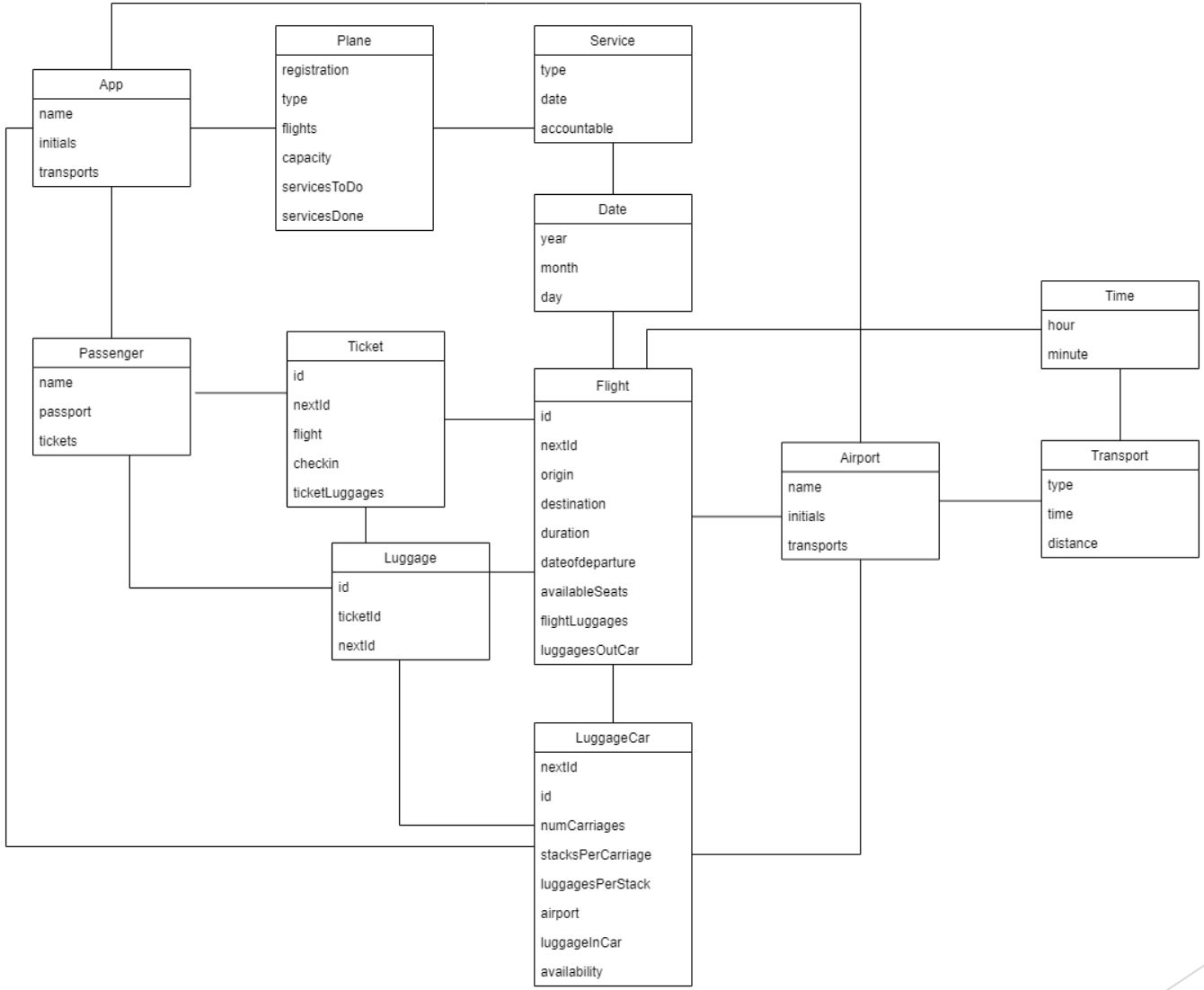
A solução

- Para suprir as necessidades da nova empresa, o sistema desenvolvido contém as classes Airport, Flight, Luggage, LuggageCar, Plane, Passenger, Service, Ticket e Transport, além de classes auxiliares como Date, Time e SortForms

- Todas as classes possuem funções *getters* e *setters*, além de permitirem, através dos menus do programa, adicionar, remover, editar e visualizar elementos, sendo todas essas mudanças salvas em ficheiros para uso futuro

- O sistema também permite que o utilizador filtre e ordene por os elementos de diversas maneiras, gerando assim uma facilidade em encontrar os itens desejados

Diagrama de classes



Estrutura de ficheiros

- Todos os ficheiros que guardam informações sobre os itens estão disponíveis na pasta cmake-build-degub. Estes são lidos na inicialização do programa e são sobrescritos com a operação de *save*

- Como todos os ficheiros possuem estruturas de armazenamento com outras classes dentro, foi necessário criar uma maneira de guardar essas informações também. Todas os modelos de leitura estão disponíveis no doxygen (app.h)

- Como exemplo, temos os ficheiros de Airport que são lidos desta maneira:
- airport name, airport initials
- transport type, transport distance, transport time (n vezes)

Funcionalidades Implementadas

- CRUD – Completa: (Create = *Create*; Read = *Find* e *Show*; Update = *Find* (Opção após encontrar o objeto; Delete = *Remove*)

- Listagem – Completa: Os elementos são guardados por classes utilizando a estrutura de dados apropriada e podem ser ordenados e visualizados pelo utilizador

- Pesquisa – Completa: O utilizador utiliza a opção *Find* numa classe, que faz pesquisa com base em atributos únicos (chave primária)

Funcionalidades Implementadas

- O sistema também conta com uma série de restrições para evitar erros dos utilizadores, como por exemplo não permitir que um objeto de uma classe seja criado se já existe outro com as mesmas características associado à lista.

- Outra restrição implementada é a de só se poder criar itens associados a outros itens se estes já existem no sistema, portanto não é possível criar um voo para um aeroporto não criado ou adicionar uma passagem à um cliente removido.

Destaque de funcionalidade

- A opção *Show*, encontrada em todas as classes, dá ao utilizador diversas opções de ordenação em relação aos atributos da classe, além de oferecer opções inversas, como por exemplo: ordem alfabética ascendente e descendente.

- Juntamente com as opções de ordenação, também há filtros, portanto o utilizador pode dar ranges para algumas variáveis, deixando muito mais simples encontrar um ou mais objetos desejados.

Dificuldades encontradas

- Uma das principais dificuldades foi a estrutura do menu principal, que por mais que apresente quase as mesmas opções à todas as classes, tem uma estrutura mais complexa e extensa, uma vez que lida com *inputs* e diferentes resultados para uma mesma seleção

- Outra dificuldade foram a criação e leitura de ficheiros, pois alguns deles tem de guardar listas e/ou pilhas, além de diversos atributos de cada classe, portanto foi necessário criar o modelo de guardamento previamente explicado, o que também trouxe a necessidade de remodelação de diversas áreas do código, que o tornou mais complexo.

- Consideramos a contribuição dos três membros do grupo de igual importância e equivalência para a realização do projeto