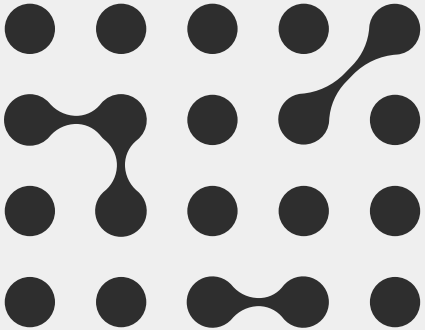
U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Companhia Aérea

O trabalho consiste na criação de um sistema de gestão de informação moderno para uma companhia aérea que satisfaça todas as suas necessidades.



Grupo 84

- Gustavo Costa, turma 9 up202004187@edu.fe.up.pt
- João Matos, turma 9 up202006280@edu.fe.up.pt
- Ricardo Cavalheiro, turma 16 up202005103@edu.fe.up.pt

Faculdade de Engenharia da Universidade do Porto

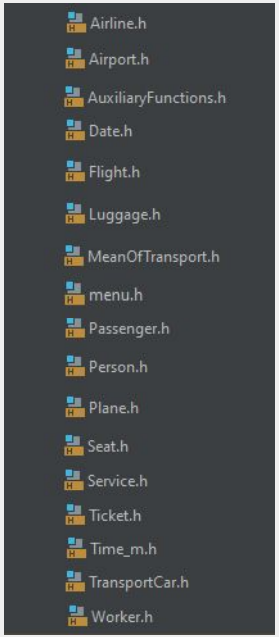
Descrição da Solução

- Criámos classes para as entidades relevantes e tentamos relacioná-las da melhor forma. Temos o exemplo do *'Flight'* que está associado a *'Airport'*, a *'Plane'*, a *'Passenger'*, entre outros que são essenciais à execução de um voo.

- Implementámos as operações *CRUD,* nomeadamente a voos, sendo possível adicioná-los e removê-los ou então alterar atributos como o aeroporto de partida/chegada e ainda a hora.

- Toda esta informação relativamente a aeroportos, aviões, voos, bilhetes, serviços, transportes e trabalhadores está devidamente armazenada em ficheiros (.txt) que são lidos ao inicializar a aplicação podendo ser alterados convenientemente ao longo do decorrer do programa.

- O utilizador, caso seja um cliente, é capaz de visualizar os meios de transporte existentes próximos de cada aeroporto assim como o seu horário. No entanto, caso seja o administrador, tem acesso a diversas funcionalidades, podendo obter informação relativamente a aviões, aeroportos, voos, entre outros, sendo esta listada de inúmeras formas. Neste processo foram utilizados algoritmos de ordenação e seleção de modo a facilitar a sua implementação..



Algoritmos Relevantes

- Foram criados vários métodos para evitar ter código duplicado. Um deles é o "*parseInput*" que verifica se o "*input*" (text) do utilizador é válido e pertence a uma das (n) opções disponíveis, retornando *false* se não pertencer e *true* se pertencer.

```
bool parseInput(int n, const string& text){
    vector<int> values;
    for(int i = 0; i <= n; i++){
        values.push_back(i);
    }
    auto it = values.begin();

    if (isAllDigits(text)) {
        it = std::find(values.begin(), values.end(), stoi(text));
        if (it != values.end()) {
            return true;
        } else {
            std::cerr << std::endl << "Not a valid input!" << std::endl << std::endl;
        }
    } else {
        std::cerr << std::endl << "Not a valid input!" << std::endl << std::endl;
    }
    return false;
}
```

```
void Menu::load() {
    airline.loadPlanes();
    airline.loadAirports();
    airline.loadFlights();
    airline.loadTransports();
    airline.loadWorkers();
    airline.loadServices();
    airline.loadTickets();
}
```

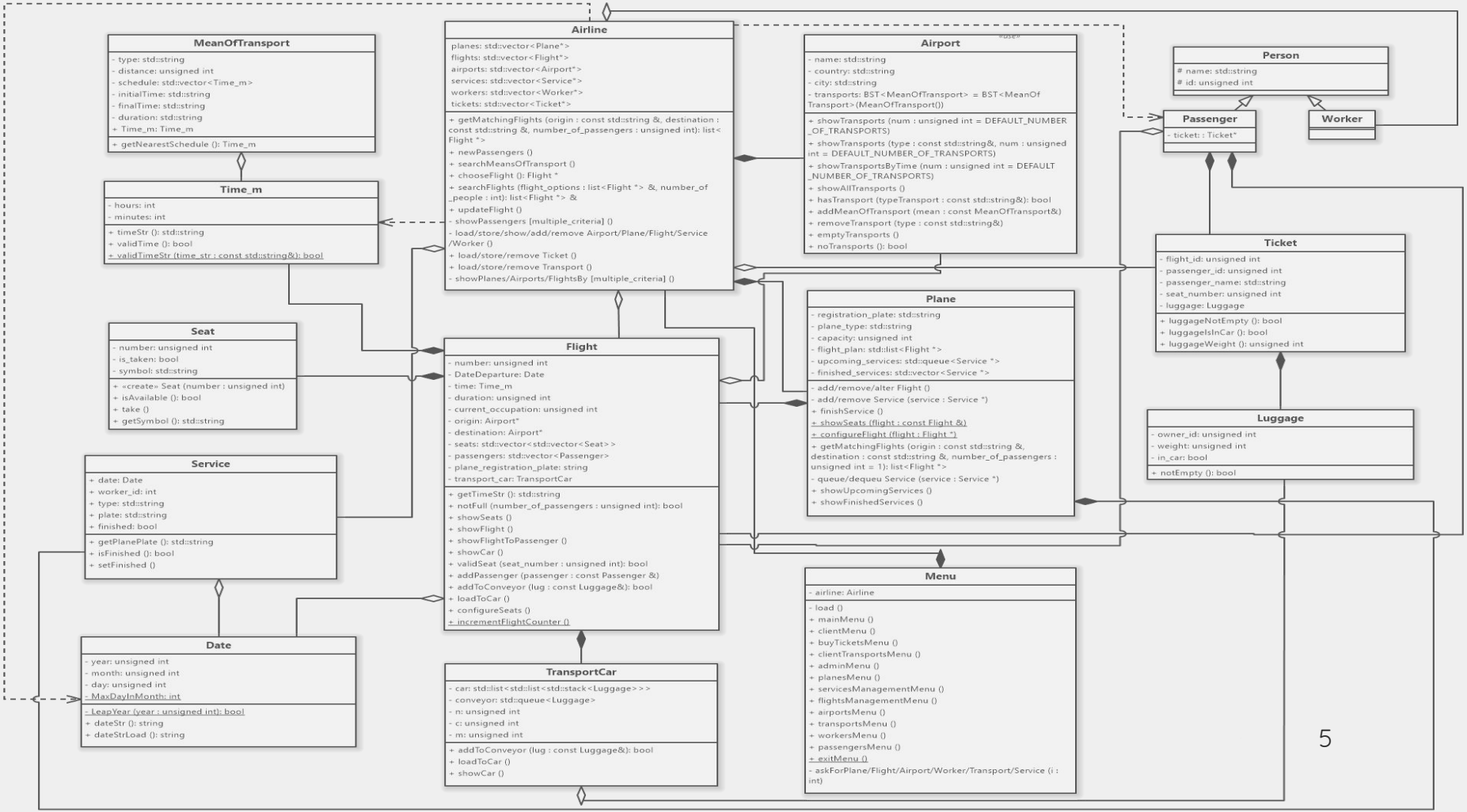
- No início da aplicação temos o método "*load*" que permite ler todos os ficheiros e armazenar toda a informação relevante.

- Foi implementado um método "*checkDate*" que verifica se a data inserida como parâmetro é posterior à data atual.

```
bool checkDate(Date d){
    time_t theTime = time(_Time nullptr);
    struct tm *aTime = localtime(&theTime);

    int day = aTime->tm_mday;
    int month = aTime->tm_mon + 1; // Month is 0 - 11, add 1 to get a jan-dec 1-12 concept
    int year = aTime->tm_year + 1900; // Year is # years since 1900
    Date dNow(year,month,day);

    return  dNow < d;
}
```

Diagrama de Classes

O diagrama de classes encontra-se disponivel no slide seguinte, ou então, para melhor legibilidade considere fazer o *download* da imagem disponível em: https://imgur.com/a/R0Dyfxc


MeanOfTransport
- type: std::string
- distance: unsigned int
- schedule: std::vector<Time_m>
- initialTime: std::string
- finalTime: std::string
- duration: std::string
+ Time_m: Time_m
+ getNearestSchedule (): Time_m
Time_m
- hours: int
- minutes: int
+ timeStr (): std::string
+ validTime (): bool
+ validTimeStr (time_str : const std::string&): bool
Seat
- number: unsigned int
- is_taken: bool
- symbol: std::string
+ «create» Seat (number : unsigned int)
+ isAvailable (): bool
+ take ()
+ getSymbol (): std::string
Service
+ date: Date
+ worker_id: int
+ type: std::string
+ plate: std::string
+ finished: bool
+ getPlanePlate (): std::string
+ isFinished (): bool
+ setFinished ()
Date
- year: unsigned int
- month: unsigned int
- day: unsigned int
- MaxDayInMonth: int
- LeapYear (year : unsigned int): bool
+ dateStr (): string
+ dateStrLoad (): string
Airline
planes: std::vector<Plane*>
flights: std::vector<Flight*>
airports: std::vector<Airport*>
services: std::vector<Service*>
workers: std::vector<Worker*>
tickets: std::vector<Ticket*>
+ getMatchingFlights (origin : const std::string &, destination : const std::string &, number_of_passengers : unsigned int): list<Flight *>
+ newPassengers ()
+ searchMeansOfTransport ()
+ chooseFlight (): Flight *
+ searchFlights (flight_options : list<Flight *> &, number_of_people : int): list<Flight *> &
+ updateFlight ()
- showPassengers [multiple_criteria] ()
- load/store/show/add/remove Airport/Plane/Flight/Service/Worker ()
+ load/store/remove Ticket ()
+ load/store/remove Transport ()
- showPlanes/Airports/FlightsBy [multiple_criteria] ()
Flight
- number: unsigned int
- DateDeparture: Date
- time: Time_m
- duration: unsigned int
- current_occupation: unsigned int
- origin: Airport*
- destination: Airport*
- seats: std::vector<std::vector<Seat>>
- passengers: std::vector<Passenger>
- plane_registration_plate: string
- transport_car: TransportCar
+ getTimeStr (): std::string
+ notFull (number_of_passengers : unsigned int): bool
+ showSeats ()
+ showFlight ()
+ showFlightToPassenger ()
+ showCar ()
+ validSeat (seat_number : unsigned int): bool
+ addPassenger (passenger : const Passenger &)
+ addToConveyor (lug : const Luggage&): bool
+ loadToCar ()
+ configureSeats ()
+ incrementFlightCounter ()
TransportCar
- car: std::list<std::list<std::stack<Luggage>>>
- conveyor: std::queue<Luggage>
- n: unsigned int
- c: unsigned int
- m: unsigned int
+ addToConveyor (lug : const Luggage&): bool
+ loadToCar ()
+ showCar ()
Airport
- name: std::string
- country: std::string
- city: std::string
- transports: BST<MeanOfTransport> = BST<MeanOfTransport>(MeanOfTransport())
+ showTransports (num : unsigned int = DEFAULT_NUMBER_OF_TRANSPORTS)
+ showTransports (type : const std::string&, num : unsigned int = DEFAULT_NUMBER_OF_TRANSPORTS)
+ showTransportsByTime (num : unsigned int = DEFAULT_NUMBER_OF_TRANSPORTS)
+ showAllTransports ()
+ hasTransport (typeTransport : const std::string&): bool
+ addMeanOfTransport (mean : const MeanOfTransport&)
+ removeTransport (type : const std::string&)
+ emptyTransports ()
+ noTransports (): bool
Plane
- registration_plate: std::string
- plane_type: std::string
- capacity: unsigned int
- flight_plan: std::list<Flight *>
- upcoming_services: std::queue<Service *>
- finished_services: std::vector<Service *>
- add/remove/alter Flight ()
- add/remove Service (service : Service *)
+ finishService ()
+ showSeats (flight : const Flight &)
+ configureFlight (flight : Flight *)
+ getMatchingFlights (origin : const std::string &, destination : const std::string &, number_of_passengers : unsigned int = 1): list<Flight *>
- queue/dequeu Service (service : Service *)
+ showUpcomingServices ()
+ showFinishedServices ()
Menu
- airline: Airline
- load ()
+ mainMenu ()
+ clientMenu ()
+ buyTicketsMenu ()
+ clientTransportsMenu ()
+ adminMenu ()
+ planesMenu ()
+ servicesManagementMenu ()
+ flightsManagementMenu ()
+ airportsMenu ()
+ transportsMenu ()
+ workersMenu ()
+ passengersMenu ()
+ exitMenu ()
- askForPlane/Flight/Airport/Worker/Transport/Service (i : int)
Person
name: std::string
id: unsigned int
Passenger
- ticket: : Ticket*
Worker
Ticket
- flight_id: unsigned int
- passenger_id: unsigned int
- passenger_name: std::string
- seat_number: unsigned int
- luggage: Luggage
+ luggageNotEmpty (): bool
+ luggageIsInCar (): bool
+ luggageWeight (): unsigned int
Luggage
- owner_id: unsigned int
- weight: unsigned int
- in_car: bool
+ notEmpty (): bool

Estrutura de Ficheiros

- Os ficheiros que utilizamos foram criados de modo a facilitar a leitura, execução e posterior armazenamento de dados do programa.
  Exemplos: (airports.txt - Nome, Cidade, País)

```
Aeroporto Internacional Humberto Delgado,Lisboa,Portugal
Aeroporto Internacional Francisco Sa Carneiro,Porto,Portugal
Aeroporto Internacional Adolfo Suarez-Barajas,Madrid,Espanha
Aeroporto Internacional de Lyon,Lyon,Franca
Aeroporto Internacional de Amsterdao Schiphol,Amesterdao,Paises Baixos
Aeroporto Marco Polo,Veneza,Italia
```

(flights.txt - Aeroporto de partida, Aeroporto de chegada, Tipo de avião, Data, hora)

```
Aeroporto Internacional Humberto Delgado,Aeroporto Internacional Francisco Sa Carneiro,F-GUGJ,2021,12,22,19:25
Aeroporto Marco Polo,Aeroporto Internacional de Lyon,N102NN,2022,04,02,15:00
Aeroporto Internacional Humberto Delgado,Aeroporto Internacional Francisco Sa Carneiro,N102NN,2022,03,15,17:17
Aeroporto Internacional Humberto Delgado,Aeroporto Internacional Francisco Sa Carneiro,N102NN,2024,02,02,13:13
Aeroporto Internacional O'Hare,Aeroporto Marco Polo,N102NN,2030,02,01,16:16
```

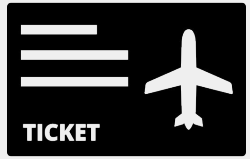
- Estes são lidos no início do programa por métodos de load, que são semelhantes entre si. Temos o caso do "*loadAirports*" que recebe o nome do aeroporto, a sua cidade e o seu país, guardando no sistema para posterior utilização.

```
void Airline::loadAirports() {
    if(!airports.empty()) airports.clear();

    // Create a text std::string, which is used to output the text file
    std::string line, next;

    // Read from the planes.txt text file
    std::ifstream airportsfile( s: "TextFiles/airports.txt");

    // Read the file, line by line
    while(std::getline( &: airportsfile,  &: line)){

        std::stringstream ss(line);
        std::vector<std::string> values;

        // Separate each line by ","
        while (ss.good()) {
            std::string substr;
            std::getline( &: ss,  &: substr,  delim: ',');
            values.push_back(substr);
        }

        std::string name = values.at( n: 0), city = values.at( n: 1), country = values.at( n: 2);

        cities_set.insert(city);
        countries_set.insert(country);

        auto* airport = new Airport( name: name, country: country, city: city);
        this->airports.push_back(airport);
    }
    airportsfile.close();
}
```

Lista de funcionalidades implementadas

- *CRUD* (*Create, Read, Update, Delete*)

Em relação aos voos, é possível adicioná-los através do método *addFlight*, removê-los através de *removeFlight* e é possível também alterar atributos como o aeroporto de chegada, a hora e a data (*updateFlightDestination*; *updateFlightTime*; *updateFlightDate*).
Em relação às restantes classes nomeadamente os aeroportos, aviões, transportes, entre outros, são capazes também de realizar, através dos menus respetivos, operações básicas como ler, adicionar, remover ou alterar.



- Listagens

O utilizador, caso seja um cliente, é capaz de visualizar os meios de transporte existentes próximos de cada aeroporto assim como o seu horário, podendo este ser ordenado de duas formas diferentes: segundo a proximidade do transporte em relação ao aeroporto (listagem total), ou segundo o tipo de transporte (*metro, autocarro ou metro*) desejado pelo utilizador (listagem parcial).

```
void showTransports(const std::string& type, unsigned int num = DEFAULT_NUMBER_OF_TRANSPORTS);
void showTransportsByTime(unsigned int num = DEFAULT_NUMBER_OF_TRANSPORTS);
```

Lista de funcionalidades implementadas

- Listagens filtradas e ordenadas

No entanto, caso seja o administrador, tem acesso a diversas funcionalidades. Este consegue visualizar:
-todos os aeroportos ou apenas os aeroportos de determinadas cidades ou países.
-os aviões ordenados segundo capacidade, tipo ou matrícula.
-o estado de ocupação dos carrinhos de transporte para cada voo.
-os trabalhadores segundo ordem alfabética.
-os meios de transporte da mesma forma que o cliente consegue.
-os serviços realizados ou por realizar de cada avião.
-os passageiros de cada voo segundo ordem alfabética.
-os serviços realizados e por realizar de cada avião.

Estes critérios foram criados para permitir ao administrador aceder a qualquer tipo de informação sem limitações.

Para efetuar estas listagens criámos vários algoritmos de ordenação. Nomeadamente em relação à ordenação dos aviões, foi criado um método *showPlanesByCriteria* que recebe como parâmetro um critério e conforme este ordena os aviões utilizando os métodos *comparePlanesByType*, *comparePlanesByPlate* ou *comparePlanesByCapacity*.

Exemplo de Listagens

- Aeroportos:
  Em relação aos aeroportos, estes podem ser apresentados:
  -todos os existentes no sistema.
  -apenas os específicos a determinados países/cidades.

```
Path: Management\Airports\Show airports
1: Show all Airports from database
2: Show all Airports from one or multiple cities
3: Show all Airports from one or multiple countries
4: Back
0: Exit
```

Exemplo: Cidade - Porto

```
There is 1 Airport.
Airport:
    Name: Aeroporto Internacional Francisco Sa Carneiro
    Country: Portugal
    City: Porto
```

Exemplo: Países - Portugal e Espanha

```
There are 3 Airports.
Airport:
    Name: Aeroporto Internacional Humberto Delgado
    Country: Portugal
    City: Lisboa
Airport:
    Name: Aeroporto Internacional Francisco Sa Carneiro
    Country: Portugal
    City: Porto
Airport:
    Name: Aeroporto Internacional Adolfo Suarez-Barajas
    Country: Espanha
    City: Madrid
```

Destaque de funcionalidades

Admin menu

```
Path: Management
1. Planes
2. Airports
3. Transports
4. Workers
5. Passengers
0. Exit
Choose an option from the above:
```

1: Planes

```
Path: Management\Planes
1. Manage vehicles
2. Manage services
3. Manage flights
4. Back
0. Exit
Choose an option from the above:
```

2: Airports

```
Path: Management\Airports
1. Add an airport
2. Remove an airport
3. Show airports
4. Back
0. Exit
Choose an option from the above:
```

3: Transports

```
Path: Management\Transports
1. Add a transport
2. Remove a transport
3. Show means of transport
4. Back
0. Exit
Choose an option from the above:
```

4: Workers

```
Path: Management\Workers
1. Add a worker
2. Remove a worker
3. Show workers
4. Back
0. Exit
Choose an option from the above:
```

5: Passengers

```
Path: Management\\Passengers
1. Show passengers
2. Back
0. Exit
Choose an option from the above:
```

Tivemos especial preocupação em fazer um menu que permitisse ao utilizador gerir a informação da maneira mais simples e eficiente possível.

Existe também um menu para clientes que permite a compra de bilhetes e consulta de horários de transportes perto de aeroportos. Durante a compra de bilhetes, um diagrama com os lugares disponíveis é mostrado

```
105  106        107  108
109  110        111  112
113  114        115  116
  X  118        119  120
```

Principais dificuldades

- Relacionar as diferentes classes entre si de forma adequada.
- Adicionar, remover e atualizar a informação armazenada em ficheiros.
- Remover os voos, transportes, serviços e bilhetes associados à remoção de um aeroporto.
- Encontrar a causa de *bugs*, devido à dimensão do trabalho.

- O trabalho foi dividido igualmente pelos membros do grupo.

Observações

- Não foi óbvio como implementar e de que modo seria útil gurdar um carrinho de transporte para as bagagens de cada voo, mas consideramos que a opção escolhida é válida e útil.
- Há funcionalidades que não estão atualmente ativas por falta de relevância mas foram pensadas e desenvolvidas (encontrando-se comentadas ou inutilizadas) tal como a visualização dos 3 horários mais próximos da hora atual de um meio de transporte, podendo ordenar meios de transporte com base neste critério.
- Optou-se por não limpar o ecrã após cada operação, de modo a garantir uma melhor compatibilidade entre sistemas operativos e uma vez que a alternativa seria “empurrar” todo o texto prévio para cima.
- O utilizador é impedido de introduzir entradas inválidas, sendo necessário repetir a entrada. (Exemplo: escolha de lugar ocupado, introdução de *string* em vez de inteiro, escolha de aeroporto inexistente, etc.)

Exemplos de execução

Compra de bilhete

```
Path: Client\Buy Tickets
Hello! Please enter the number of people that require a ticket (1 or more).
>1
What is your name?
Joao Matos
Do you want search by airport name or city? (Answer 'city' or 'name')
city
You are searching for the city of the airports

Where will you start the trip?
>Lisboa
And where do you want to go?
>Porto
Please choose one of the following flights:
```

```
Flight no. 3;
Departs on 2024/02/02, 13:13 and has a duration of 5 hours;
Has an occupation of 4/160 seats;
It will depart from Aeroporto Internacional Humberto Delgado in Lisboa a
a Carneiro in Porto.
```

```
 153  154        155    X
   X  158          X  160
Joao Matos, please choose an available seat.
>154
Joao Matos, do you want to include luggage? ('Y' / 'N')
>y
Joao Matos, how much does it weigh?
>2
Joao Matos, do you want your luggage checked-in automatically? ('Y' / 'N')
>y
Have a nice flight!!
```

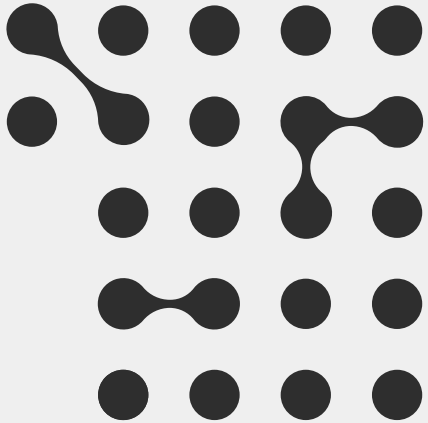
Carrinho de bagagens

```
Path: Management\Planes\Manage flights\Show flights
1. Show all flights from database
2. Show all flights from one or multiple planes
3. Show all flights from/to one or multiple cities
4. Show the occupation of the transport car of a flight
5. Back
0. Exit
Choose an option from the above: 4
```

...

```
Flight no. 9 will take place in the plane with the plate CS-TXH;
Departs on 2022/01/30 at 23:40 and has a duration of 5 hours;
Has an occupation of 2/180 seats;
It will depart from Aeroporto Marco Polo in Veneza and arrive at Aeroporto Interna


Flight no. 10 will take place in the plane with the plate PH-HXG;
Departs on 2022/03/20 at 16:15 and has a duration of 5 hours;
Has an occupation of 0/180 seats;
It will depart from Aeroporto Internacional Humberto Delgado in Lisboa and arrive

Flight no.: 4
Carriages: 2    Stacks per carriage: 3  Maximum items per stack: 2
|0|0|0|  |0|2|2|
```

THANKS



Algoritmos e Estruturas de Dados
2021/22

- CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, and infographics & images by **Freepik**
- Please keep this slide for attribution