U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

# Algoritmo e Estrutura de Dados

# *T2 Boleias Partilhadas*

*2MIEIC1_E*

*(2 de janeiro de 2017)*


Bárbara Silva **up201505628**@fe.up.pt

Igor Silveira **up201505172**@fe.up.pt

Julieta Frade **up201506530**@fe.up.pt

# *Índice*

# *ShareIt*

*ShareIt* é um sistema para a gestão de uma rede social de partilha de boleias baseado no conceito de *carpooling* e *ridesharing*.

A empresa disponibiliza uma vasta rede de carros que os utilizadores que se registem como "driver", isto é, que desejam disponibilizar a sua viatura, devem alugar. O acordo que o utilizador faz ao se registar desta forma é que só pode usar os carros da empresa na partilha de viagens, o que proporciona uma vantagem para o mesmo. Para realizar uma viagem, precisa de obrigatoriamente ter um carro, ou mais, alugado.

Caso o utilizador esteja registado apenas para efeito de partilha de viagem, deve candidatar-se à mesma. De maneira a gerir melhor a procura por determinados destinos de várias pessoas, acima da capacidade das viaturas, a empresa decide priorizar aqueles com relação de amizade mais próxima do motorista, e depois aquelas com distância mais próxima do itinerário original a ser executado pelo motorista.

Relativamente a todos os utilizadores do sistema, aqueles que não o utilizam há mais de 7 dias, passam a ser utilizadores inativos, e neste caso, para efeitos de promoção da aplicação, é pedido ao utilizador, na próxima vez que inicia sessão, que atualize a sua morada. Evidentemente, se um cliente inativo realiza uma nova viagem, passa a ser ativo.

# *Solução Implementada*

1) **Register**
2) **Login**
    a) ADMIN
        i) Users
            (1) Sort by Username
            (2) Sort by Name
        ii) Trip Record
        iii) Transactions
            (1) Collect Payment
        iv) Friendships
        v) Stops
        vi) Search
            (1) Users
                (a) By ID
                (b) By Username
            (2) Trips
                (a) By Driver
                (b) By Month
            (3) Transactions
                (a) By User
                (b) By Month
        vii) Run Trip
        viii) Add Buddy
    b) DRIVER
        i) Account
            (1) Deposit
            (2) Change Username
            (3) Change Address
            (4) Change Password
            (5) Delete Account
        ii) Create Trip
        iii) Add Buddy
        iv) Car Details
            (1) Rent Car
            (2) Discard Car
            (3) Trade Car
            (4) Search Car
        v) Scheduled Trips
            (1) Trip Manager
                (a) Cancel Trip
                (b) Choose Passangers
    c) PASSENGER
        i) Account
            (1) Deposit
            (2) Change Username
            (3) Change Password
            (4) Delete Account
        ii) Join Trip
        iii) Add Buddy
3) **Guest**
    a) Join Trip

A solução implementada foi primeiramente, a criação de uma árvore binária de pesquisa para armazenamento dos carros alugados pelos utilizadores, sendo que a ordenação é efetuada pelo nome da marca e modelo ou ano, caso estes parâmetros sejam todos iguais, pois a empresa possui vários carros iguais, são ordenados pelo ID do utilizador a quem pertencem. Assim, é possível alugar e remover carros, como também alterar o seu dono.

Os candidatos à partilha da viagem são ordenados numa fila de prioridade, a fim de preencher as vagas disponíveis nas viaturas. São usados dois critérios para ordenar esta fila. Primeiro tem-se em conta a proximidade ao driver, ou seja, se é *buddy* ou não, e seguidamente a distância desde a paragem onde o passageiro quer entrar até à primeira paragem da viagem.

Os utilizadores inativos foram guardados numa tabela de dispersão, onde se pode inserir ou remover utilizadores e alterar a sua morada.

# *Casos de Utilização*

## *Classes*

**Agency**

Esta é considerada a classe principal. Foi implementada com o padrão de projetos de software Singleton. Este padrão garante a existência de apenas uma instância de uma classe, mantendo um ponto global de acesso ao seu objeto.

Esta classe contém:

- Vetores: *Users* (vetor de todos os usuários registados), *Trips* (vetor de todas as viagens adicionadas ao sistema), *ActiveTrips* (vetor de viagens registadas mas ainda não realizadas), *Transactions* (vetor de todas as transações dos usuários para a agência), *stopsAvailable* (vetor com as paragens disponíveis na agência), *Cars* (vetor com os carros disponibilizados pela agência), *distancesVec* (vetor com as distâncias entre todos os pontos do mapa).
- Árvore binária de pesquisa: *BST<Vehicle> vehicles***.**
- Tabela de dispersão: *unordered_set<userPtr, inactivePtr, inactivePtr> tabHInactive*

**User**

Esta classe tem como atributos: *ID*, *username, password, balance* (saldo na conta do usuário), *buddies* (vetor de ids dos buddies do usuário), *ntrips* (numero de trips realizadas por o usuário), *Maintenancefee* (manutenção mensal), *lastAccess* (Data do último acesso), *address*.

Esta classe tem como derivadas a classe Driver e a classe Passenger.

**Trip**

Esta classe tem como atributos: *ID*, driv*er* (id do driver da viagem)*, stops* (paragens desta viagem), *date* (dia da viagem), *startTime* (hora de início da viagem), *endTime* (hora de fim da viagem), *candidateQueue* (fila de prioridade com os candidatos à viagem).

**CandidateTrip**

Esta classe tem como atributos: *passenger* (apontador para o passenger que se candidata), *driver* (apontador para o driver da viagem à qual o passenger se candidata)*, distance* (distância da paragem que o passenger entra até à primeira paragem da viagem), *initStop* (primeira paragem do passenger) e *endStop* (código da ultima paragem do passenger).

**Stop**

Esta classe tem como atributos: *code* (o código da paragem), *availableSeats* (nº de lugares disponíveis na viatura, ou seja, passageiros que pode entrar nessa paragem) e *passengers* (id dos passageiros que estão dentro da viatura nessa paragem).

**Transactions**

Esta classe tem como atributos: *id* (id do usuário relativo a esta transação), *date* (dia da transação), *value* (valor transferido).

**Date**

Esta classe tem como atributos: *day, month* e *year*.

**Hour**

Esta classe tem como atributos: *hour* e *minutes*.

**Vehicle**

Esta classe tem como atributos: *brand*, *model*, *year* e *driver* (apontador para o utilizador a quem pertencem o carro de momento).

## *Ficheiros de texto*

**Users.txt**

Lista de todos os usuários no formato:

- *ID ; nome ; 1 se driver, 0 se passenger; balance ; username; password ; numero de viagens; código da morada ; data de último acesso*

**Record.txt**

Lista de todas as viagens adicionadas ao sistema no formato:

- *ID da viagem ; ID do driver ; paragem de inicio ; paragem de fim ; data ; hora de início ; hora de fim*

**ActiveTrips.txt**

Lista de viagens adicionadas ao sistema mas ainda não realizadas no formato:

- *ID da viagem ; ID do driver ; [ código da primeira stop , numero de lugares disponíveis na viatura nessa stop, (ids dos passageiros dentro da viatura nessa stop);* *(repetir para todas as stops da trip)* *] ; data ; hora de inicio ; hora de fim*

**Stops.txt**

Lista de todas as stops disponíveis no Sistema no formato:

- *Código da Paragem ; Nome da Paragem*

**Transactions.txt**

Lista de todas as transações dos usuários para a agência no formato:

- *ID do usuário ; data ; valor da transação*

**Buddies.txt**

Lista das relações entre os usuários no formato:

- *ID do usuário; buddies desse usuário.*

**CandidatesQueues.txt**

Lista de todas as filas de candidatos às viagens ativas no formato:

- *ID da trip; ID do driver; ID do passenger, distância, código da primeira paragem, código da última paragem do passenger;* *(repetir para o resto da fila)*

**Distances.txt**

Lista das distâncias entre todas as paragens no formato:

- *Código paragem 1; Código paragem 2; distância entre elas*

**Vehicles.txt**

Lista de todos os veículos que a empresa tem no formato:

- *Marca ; Modelo ; Nº de lugares ; Ano ; Quantidade*

**VehiclesTree.txt**

Lista de todos os elementos da árvore binária de pesquisa no formato:

- *Marca ; Modelo ; Ano ; ID do utilizador*

## *Programa*

Todo o programa permanece  o mesmo, tendo em conta o projeto anterior, tirando a implementação das novas funcionalidades e algumas melhorias.

Atualmente, após o login de qualquer *user*, no menu ***Account***, este pode também mudar o *username*, a *password*, apagar a conta ou mudar o seu código de morada, algo que não tínhamos na versão anterior.

Caso o utilizador seja um *driver*, agora tem uma nova opção no menu inicial, ***Car Details***, na qual pode: ver a informação dos carros que tem alugados, alugar um novo carro, remover, passar um carro para nome de outro *user*, isto é, podemos ver esta funcionalidade como uma espécie de troca de dono do carro, na qual o *user* diz que carro que dar e a quem, ficando no fim sem o carro respetivo, que passou a ser do *user* escolhido. Pode também pesquisar os carros presentes na árvore binária de pesquisa, podendo ver toda a informação dos mesmos.

Visto agora cada driver ter carros associados, a funcionalidade **Create Trip** teve que ser alterada, de modo a que só é possível criar uma viagem se o utilizador tiver pelo menos um carro associado, e caso tenha mais que um, terá que escolher qual quer utilizar.

A funcionalidade ***Join Trip*** no menu dos *passengers* foi alterada ligeiramente. Agora o utilizador não se junta automaticamente à trip, apenas se candidata. Ao candidatar-se é adicionado à fila de prioridade dessa trip, não sendo possível candidatar-se duas vezes à mesma trip.

Cada *driver* tem agora a opção ***Scheduled Trips*** que mostra todas as suas viagens agendadas. Dentro desta opção o utilizador pode escolher cancelar a trip ou então escolher os passageiros que quer em cada trip sendo o número máximo o número de lugares no seu carro. Se escolher a segunda opção é-lhe apresentada a fila de prioridade dessa trip ordenada e o driver só tem que escolher os passageiros que quer. Não poderá escolher o mesmo passageiro duas vezes, passageiros não presentes na fila, nem fazer esta ação duas vezes.

No início do programa será gerada uma tabela de dispersão com os *User* cuja data do último acesso seja superior à atual em pelo menos 7 dias. Caso um desses utilizadores iniciar sessão no programa e entrar na sua página de conta, será visível um aviso que lhe indica que deverá atualizar a sua morada.  O aviso será sempre mostrado até que o mesmo atualize a sua morada pelo menos uma vez após ser lhe transmitido o aviso, quer seja a mesma que a anterior ou não.

# *Dificuldades*

Ao longo do projeto não houve grandes dificuldades. As maiores dúvidas surgiram na junção da implementação das funcionalidades propostas com o código antigo, assim como interpretar o que nos era pedido no enunciado.

# *Distribuição de Trabalho Pelos Elementos*

O trabalho foi dividido igualmente por todos os membros. A distribuição do trabalho foi feita da seguinte forma:

- *Bárbara Silva* – árvore binária de pesquisa.
- *Igor Silveira* – tabela de dispersão.
- *Julieta Frade* – fila de prioridade.

Visto que tínhamos 3 novas funcionalidades a implementar, achamos mais eficaz dividi-las pelos três. Todas as funções relativas aos mesmos foram implementadas por cada um, tendo em conta que houve algumas feitas em parceria.