Conteúdo

1 Introdução **4**

2 Descrição do Problema **5**

3 Arquitetura de Classes **6**
3.1 Atores de Sistema 6
3.1.1 Client 6
3.1.2 Driver 7
3.2 Interface TemPosição 8
3.3 Viagens 9
3.4 Interface Viagens 10
3.5 Empresa 10
3.6 Interface Fila de Espera 11
3.7 Base de Dados 12
3.8 Diagrama de Classes 13

4 Manual de Utilização **14**
4.1 Página Inicial 14
4.2 Cliente 14
4.2.1 Criar Conta 14
4.2.2 Login 15
4.2.3 Fazer Viagem 15
4.2.4 Area Pessoal 18
4.3 Motorista 18
4.3.1 Criar Conta 18
4.3.2 Login 19
4.3.3 Area Pessoal 19
4.4 Empresa 20
4.4.1 Criar Conta 20
4.4.2 Login 21

5 Conclusão **22**
5.1 Análise crítica 22
5.2 Perspetivas de melhorias e trabalho futuro 22

1 Introdução

O presente documento descreve todo o trabalho realizado para o Projecto da Unidade Curricular de Programação Orientada aos Objetos. O tema proposto foi criar uma aplicação de um serviço de transporte de passageiros. O projecto denomina-se *UMeR* e baseia-se num enunciado proposto pela equipa docente.

Este documento encontra-se estruturado da seguinte forma: primeiro será feita uma breve descrição do problema, de seguida, é descrito ao detalhe a arquitetura de classes; a seguir surgirá um manual de utilização da aplicação e por último a conclusão e melhorias para o futuro da aplicação.

2 Descrição do Problema

Hoje em dia a maior parte das pessoas faz-se acompanhar de um smartphone com acesso a funcionalidades de *Wi-Fi*. Este facto já inspirou a criação de muitas aplicações capazes de melhorar a vida dos seus utilizadores.

O projeto baseia-se na criação de uma aplicação capaz de facilitar o acesso a táxis e às viagens, nos mesmos; com várias funcionalidades. Funcionalidades onde se encontra a criação de utilizador (Cliente ou Motoristas), podendo ainda criar veículos. também com a funcionalidade de registar todas as viagens, dividindo por utilizador.

Para além do registo de viagens um Cliente também deverá conseguir solicitar viagens: ao táxi mais próximo das suas coordenadas ou a um táxi especifico. Ainda podendo fazer uma reserva para um táxi especifico que não esteja disponível, no momento.

Os Motoristas vão conseguir sinalizar que estão disponíveis para serem requisitados, registar uma viagem para um determinado Cliente e ainda registar o preço de determinadas viagem.

3 Arquitetura de Classes

3.1 Atores de Sistema

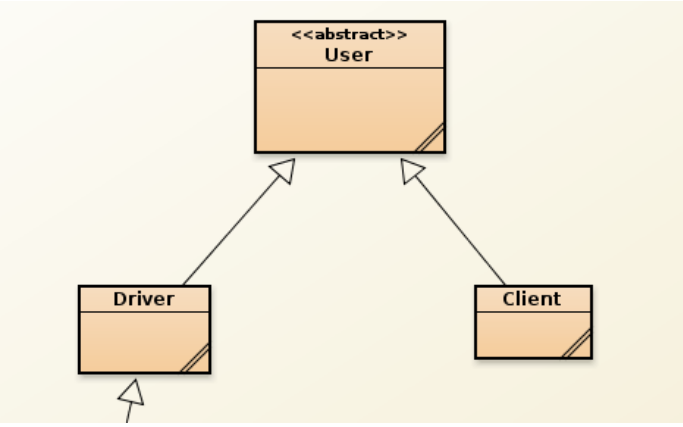

Figura 1: Atores de Sistema

Um objeto do tipo de dados User é um utilizador da aplicação onde vai ser identificado por email, password, nome, morada, data de nascimento e uma lista de viagens. User é uma super classe de Driver e Cliente.

3.1.1 Client

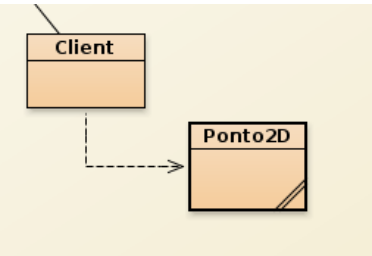

Figura 2: Cliente

Cliente vai ser todo o conteúdo de User mais uma posição (um objeto da classe Ponto2D). A classe Ponto2D contém as coordenadas.

3.1.2 Driver

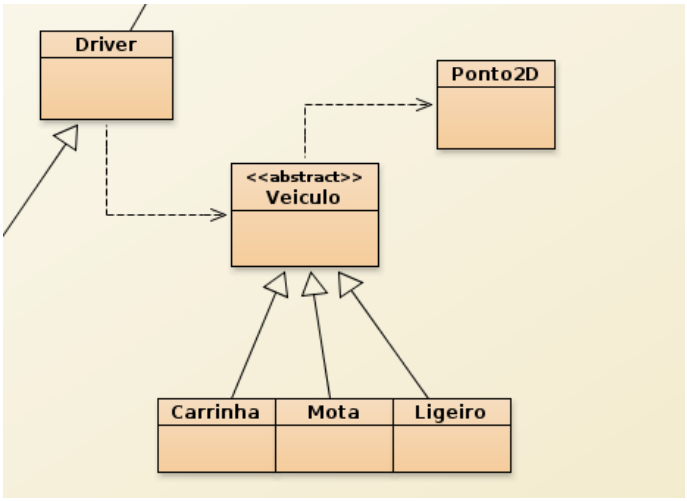

Figura 3: Motorista

A classe Driver para além do que herda de User tem mais características, como grau de cumprimento do serviço, classificação média do motorista, kms totais percorridos e se está disponível para fazer viagens. Para além disso tem um veículo associado.

A classe Veículo é uma classe abstrata onde tem como variáveis de instância a matricula, preço por Km, velocidade média, número de lugares disponíveis, fiabilidade e uma posição de uma objeto do seu tipo de dados. As subclasses de Veiculo (Carrinha, Mota, Ligeiro) tem o número máximo de pessoas que aquele veículo pode levar e a fiabilidade a ele associada.

3.2 Interface TemPosição

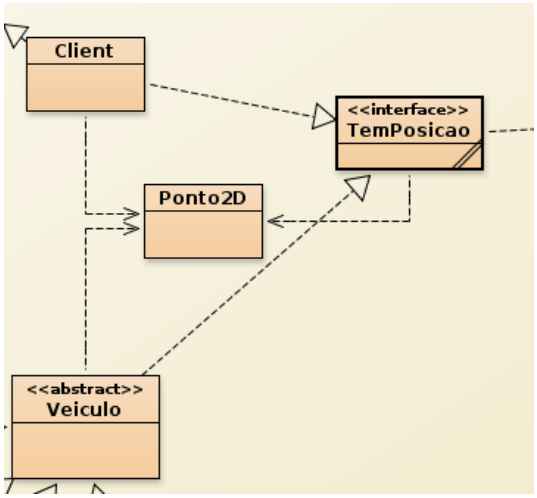

Figura 4: Interface Posicao

A interface TemPosicao tem os métodos que permitem obter e alterar a posição que um objeto se encontra. Esta interface serve para criar alguma uniformidade entre objetos que tem uma posição a eles associada.

3.3 Viagens

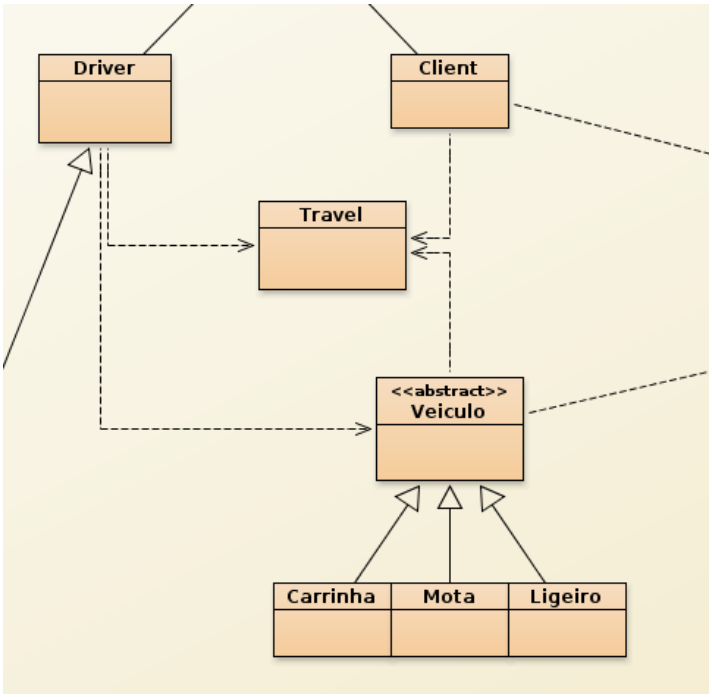

Figura 5: Viagens

Os objetos da classe Travel são históricos de viagens onde cada objeto tem todas as informações sobre uma dada viagem. É através desta classe que se fazem viagens (através dos seus métodos estáticos) e que posteriormente é consultada a informação sobre ela. A classe User e Veiculo tem também implementado nas suas variáveis de instância uma estrutura de dados para guardar essas viagens realizadas.

3.4 Interface Viagens

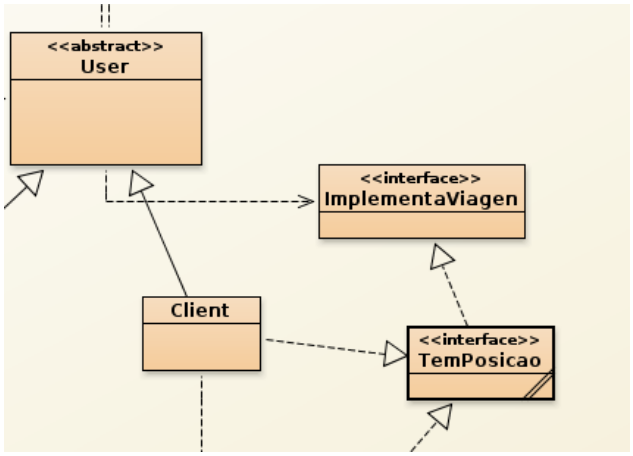

Figura 6: Interface Viagem

Como os objetos das classes Veiculo e User tem uma estrutura de dados que guarda viagens, houve a necessidade de criar uma interface que garantisse um comportamento idêntico entre as duas classes não relacionadas hierarquicamente. Para tal foi criada uma interface ImplementaViagem que implementa métodos de acesso, listagem e procura de viagens realizadas por um objeto.

3.5 Empresa

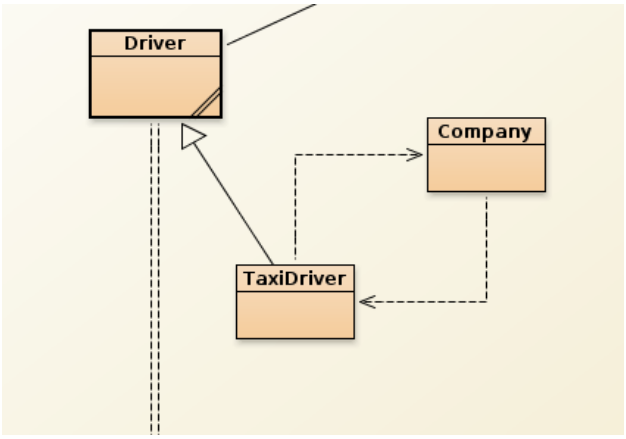

Figura 7: Empresas

A aplicação expandiu o seu volume de negócios e decidiu incorporar empresas. Assim é possível que estas possam criar uma conta e gerir os seus motoristas e veículos através dela. Foi criada a classe Company que cria objetos que representam empresas. Assim,

cada objeto desta classe tem o nome da empresa e a sua palavra passe para o login. Tem também estruturas para guardar os motoristas e veículos que a ela estão associados. Assim passa a ser possível uma empresa poder ser gerida através da aplicação pois são disponibilizadas funcionalidades tais como adicionar motoristas e veículos. A atribuição de veículos a motoristas para poderem trabalhar é feita automaticamente no momento em que ele esteja pronto para trabalhar. As empresas tem também uma área só para estatísticas para serem analisadas pelo gestor da empresa.

Ao criar empresas foi necessário distinguir os motoristas que trabalham por conta própria e os que estão associados a empresas. Para tal foi criada a classe TaxiDriver, subclasse de Driver, que permite que um condutor tenha a si associada uma empresa e um carro da empresa para que ele possa trabalhar. Também é possível assinalar se está disponível para trabalhar para a empresa.

Nota: Os TaxiDrivers podem trabalhar por conta própria, ou seja, com o seu veiculo pessoal, quando não estão a trabalhar para a empresa.

3.6 Interface Fila de Espera

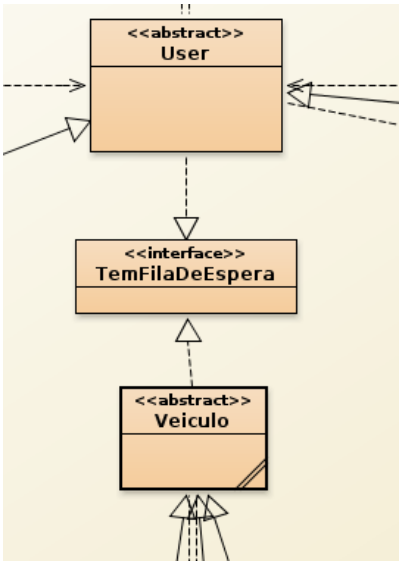

Figura 8: Interface Fila de Espera

A aplicação permite também que quando um cliente solicita um taxista ou uma viatura em especifico e esta não esta disponível, que este fique num fila de espera para que quando for possível efetuar a viagem este seja atendido. Como o cliente pode se inscrever numa fila de espera de um taxista (tipo de dados User) e numa fila de espera de um veiculo (tipo de dados Veiculo) foi criada a interface TemFilaDeEspera de forma a facilitar a atualização destas listas.

3.7 Base de Dados

Para guardar toda a informação relativa ao estado da nossa aplicação foi necessário criar uma classe que tivesse acesso a todos os objetos. Essa classe chama-se DataBase. A DataBase tem estruturas que através de uma chave conseguimos aceder um objeto, por exemplo, a um User, a um Veiculo, a uma Company ou uma Travel. Sendo assim, torna-se dispensável que cada objeto do tipo ImplementaViagens tenha uma estrutura com objetos Travel, bastando então ter uma estrutura com as chaves que dão acesso ao objeto Travel na sua DataBase. Este raciocínio expande-se a praticamente todas as classes que usam composição de outras.

O principal objetivo desta classe é eliminar problemas de colonização de objetos e conseguir manter um maior encapsulamento pois só serão partilhadas chaves e não os próprios objetos.

Visto isto é importante referir que por cada programa só pode existir um objeto do tipo DataBase. É também relevante perceber que todo o sistema perde sentido sem a classe DataBase pois este, durante o tempo de execução, precisa de traduzir constantemente as suas chaves pelos objetos.

3.8 Diagrama de Classes

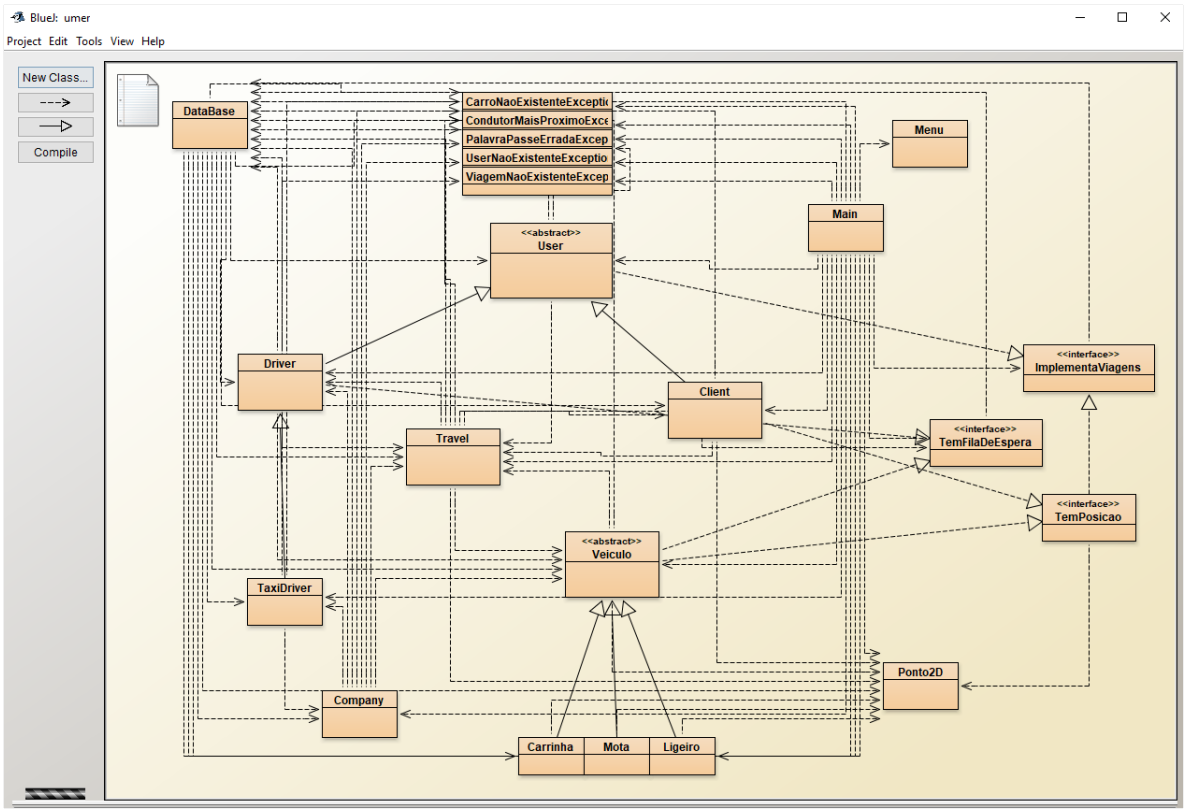

Figura 9: Diagrama de Classes da Umer

4 Manual de Utilização

4.1 Página Inicial

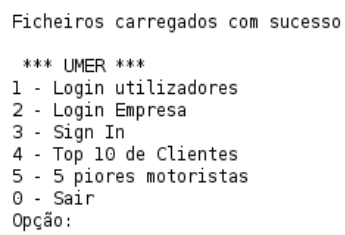

Figura 10: Página Inicial

Nesta página temos 6 opções. Para um utilizador (Cliente e Motorista) fazer login clica-se no número 1. Clica-se no 2 para fazer login com uma Empresa. Se ainda não for um utilizador clica-se o 3 para criar conta. 4 e 5 é para ver os melhores e piores motoristas. Se quiser sair da aplicação clica no 0.

4.2 Cliente

4.2.1 Criar Conta

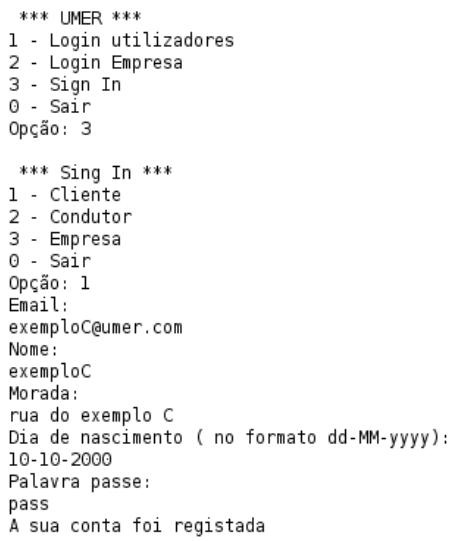

Figura 11: Criar Conta Cliente

Nesta página podemos criar conta na aplicação como Cliente, teremos então de dizer o email, o nome, morada e data de nascimento e password.

4.2.2 Login

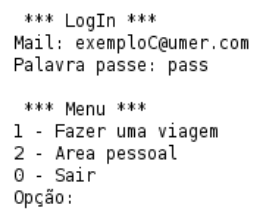

Figura 12: Login como Cliente

Para fazer login como Cliente, depois de criada conta, só tem de colocar o email e password que o levará a um menu apenas de cliente. Nesse menu podemos escolher fazer uma viagem ou ver nossa área pessoal.

4.2.3 Fazer Viagem

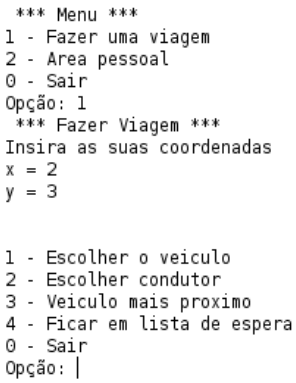

Figura 13: Fazer Viagem

Para fazer uma viagem tem de escolher a opção 1 no menu de cliente. Depois de clicar 1 tem de clocar as coordenadas do local onde se encontra. De seguida em de escolher como quer o seu transporte, por veiculo (1), por condutor (2), pelo veiculo mais próximo (3), ou ficar me lista de espera (4).

```
 *** Menu ***
1 - Fazer uma viagem
2 - Area pessoal
0 - Sair
Opção: 1
 *** Fazer Viagem ***
Insira as suas coordenadas
x = 2
y = 3


1 - Escolher o veiculo
2 - Escolher condutor
3 - Veiculo mais proximo
4 - Ficar em lista de espera
0 - Sair
Opção: 1
Insira a matricula:
11-11-QQ
Para onde deseja ir?
x = 3
y = 4
Viagem efectuada com suceeso
```

Figura 14: Fazer Viagem Escolhendo o Veiculo

Se escolher por veiculo (1), de seguida tem de inserir a Matriculo do veiculo que quer usar e de seguida as coordenadas do local para onde quer ir.

```
 *** Menu ***
1 - Fazer uma viagem
2 - Area pessoal
0 - Sair
Opção: 1
 *** Fazer Viagem ***
Insira as suas coordenadas
x = 2
y = 3


1 - Escolher o veiculo
2 - Escolher condutor
3 - Veiculo mais proximo
4 - Ficar em lista de espera
0 - Sair
Opção: 2
Insira a mail do condutor:
jose@umer.pt
Para onde deseja ir?
x = 3
y = 4
Viagem efectuada com suceeso
```

Figura 15: Fazer Viagem Escolhendo o Condutor

Se escolher por condutor (2), de seguida tem de inserir o email do condutor que quer usar e de seguida as coordenadas do local para onde quer ir.

```
 *** Menu ***
1 - Fazer uma viagem
2 - Area pessoal
0 - Sair
Opção: 1
 *** Fazer Viagem ***
Insira as suas coordenadas
x = 2
y = 3


1 - Escolher o veiculo
2 - Escolher condutor
3 - Veiculo mais proximo
4 - Ficar em lista de espera
0 - Sair
Opção: 3
Para onde deseja ir?
x = 3
y = 4
Viagem efectuada com suceeso
```

Figura 16: Fazer Viagem Escolhendo o Veiculo Mais Proximo

Se escolher por veiculo mais próximo (3), de seguida só tem de colocar as coordenadas do local onde deseja ir e será usado o veiculo mais próximo de si para a viagem.

```
 *** Menu ***
1 - Fazer uma viagem
2 - Area pessoal
0 - Sair
Opção: 1
 *** Fazer Viagem ***
Insira as suas coordenadas
x = 2
y = 3


1 - Escolher o veiculo
2 - Escolher condutor
3 - Veiculo mais proximo
4 - Ficar em lista de espera
0 - Sair
Opção: 4
1 - Escolher o veiculo
2 - Escolher condutor
0 - Sair
```

Figura 17: Fazer Viagem Escolhendo a Lista de Espera

Se escolher pela lista de espera (4), de seguida irá escolher o veiculo ou o condutor q esteja na lista de espera, da forma como anteriormente feita.

4.2.4 Area Pessoal

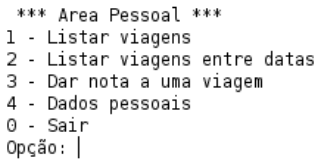

Figura 18: Area Pessoal

Na área pessoal de um cliente pode: (1) ver o histórico de viagens efetuadas, (2) ver o históricos de viagens efetuadas entra duas datas, (3) dar nota a uma viagem e (4) ver os dados pessoais e altera-los.

4.3 Motorista

4.3.1 Criar Conta

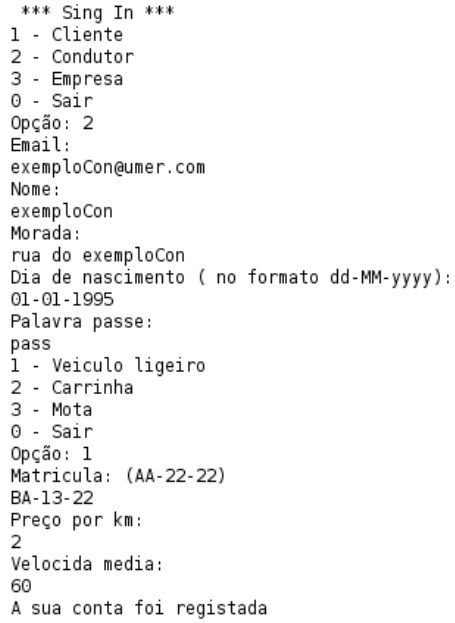

Figura 19: Criar Conta Motorista

Nesta página podemos criar conta na aplicação como Motorista, teremos então de dizer o email, o nome, morada e data de nascimento e password. Depois de criado o Motorista tem de se associar um veiculo, 1 para ligeiro, 2 para mota e 2 para carrinha, coloca-se a matrícula, preço por km e velocidade média.

4.3.2 Login

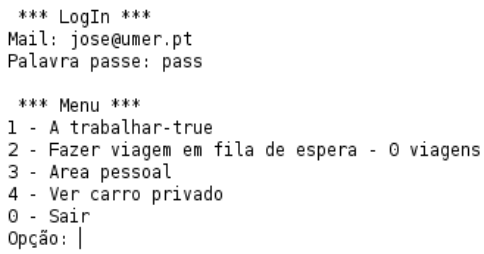

Figura 20: Login como Motorista

Para fazer login como Motorista, depois de criada conta, só tem de colocar o email e password que o levará a um menu apenas de motorista. Nesse menu temos 4 opções: (1) Dizer se está ou não a trabalhar, (2) Fazer as viagens que tem em lista de espera, (3) Informação sobre o motorista e (4) Informação sobre o carro pessoal.

4.3.3 Area Pessoal

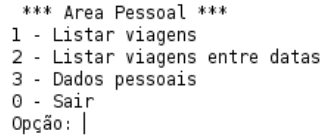

Figura 21: Area Pessoal Motorista

Na área pessoal de um cliente pode: (1) ver o histórico de viagens efetuadas, (2) ver o históricos de viagens efetuadas entra duas datas, (3) ver os dados pessoais e altera-los.

Veiculo de Motorista

```
 *** Menu ***
1 - A trabalhar-true
2 - Fazer viagem em fila de espera - 0 viagens
3 - Area pessoal
4 - Ver carro privado
0 - Sair
Opção: 4
---------------
Matricula: 12-AF-33
Preço por Km: 4.5 euros
Velocidade: 56.0 Km/h
Kilometros: 18.664072038535288 Km
Número de lugares: 4
Fiabilidade: 60
Localização: (3,4)

---------------

1 - Alterar preço por kilometro
2 - Alterar velocidade media
3 - Ver viagens
0 - Sair
Opção:
```

Figura 22: Veiculo de Motorista

Na área do veiculo de um motorista encontra se toda a informação sobre o veiculo e maneira de alterar essa informação.

4.4 Empresa

4.4.1 Criar Conta

```
 *** UMER ***
1 - Login utilizadores
2 - Login Empresa
3 - Sign In
0 - Sair
Opção: 3

 *** Sing In ***
1 - Cliente
2 - Condutor
3 - Empresa
0 - Sair
Opção: 3
Nome da empresa:
exemploEmp
Palavra passe:
pass
A sua conta foi registada
```

Figura 23: Criar Conta Empresa

Nesta página podemos criar conta na aplicação como Empresa, teremos então de dizer o email e password.

4.4.2 Login

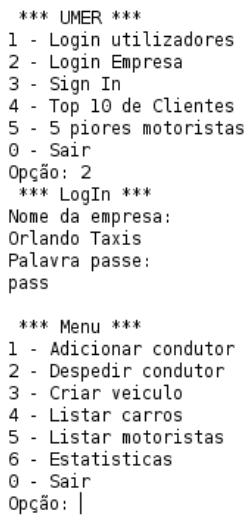

Figura 24: Login como Empresa

Para fazer login como Empresa, depois de criada conta, só tem de colocar o nome da empresa e password que o levará para um menu apenas de empresa. Nesse menu temos 6 opções: (1) Adicionar condutor, (2) Despedir condutor, (3) Criar um veiculo para a empresa (4) Ver os motoristas da empresa, (5) ver os carros da empresa e (6) Ver as estatísticas (lucro total, veiculo com mais viagens, condutor com mais viagens, taxista com maior grau de cumprimento, condutor tem viagens realizadas pela empresa e taxista com melhor classificação).

5 Conclusão

5.1 Análise crítica

Logo no início do semestre ficou estabelecido que o nosso projeto teria de ser construído seguindo um conjunto de regras que estabelecem os pilares de todas as linguagens de programação orientada aos objetos. Para tal, utilizamos Java para o desenvolvimento da nossa aplicação e tivemos de adotar um estilo de programação diferente daquele a que estávamos habituados e que a linguagem Java também aceita, um estilo de código mais imperativo, mais "à la C".

Após nos ser facultado o enunciado do projeto, abrimos o *BlueJ* e começamos logo a tentar perceber o que deveríamos fazer de modo a que pudéssemos avançar com o projeto. Então, começamos a escrever código, mas com o decorrer das aulas e à medida que aprendíamos novos conceitos sobre este novo paradigma dos objetos, apercebemo-nos que todo o código que tínhamos desenvolvido não nos iria ajudar muito, pois provavelmente teríamos de repetir código desnecessariamente e chegaria uma altura em que seria impossível implementar uma funcionalidade requerida sem que fosse preciso repensar na estrutura do código e corríamos o risco de ter de começar do zero novamente.

Finalmente, a implementação de um programa de simulação de viagens como a Umer, ajudou-nos a perceber como funcionam as aplicações semelhantes que estão disponíveis para *download*. Assim, após a realização deste projeto, somos capazes de decifrar um pouco as aplicações que o mercado nos disponibiliza e que faz com que uma simples viagem esteja ao alcance de um simples clique, como é o deste caso. Assim, a nossa avaliação deste trabalho é positiva, uma vez que conseguimos cumprir todos os objetivos e requisitos mínimos propostos e aplicar na prática os conteúdos teóricos aprendidos nas aulas, permitindo a sua consolidação.

5.2 Perspetivas de melhorias e trabalho futuro

Devido ao curto prazo de entrega, muitas das funcionalidades previstas para a Umer ficaram por fazer. A parte relativa a fazer viagens precisava de mais refinamentos ao nível de busca de táxis. Por exemplo, chamar um táxi conforme o número de pessoas que precisamos, chamar um táxi de uma empresa específica ou até pelo tipo de veiculo a que esse pertence. Estes são exemplos de funcionalidades que ficaram por implementar devido à necessidade de cumprir requisitos mínimos.

Fica também por resolver o problema de adicionar veículos de um tipo diferente ao que foi implementado. Para isso teria de haver uma classe *Outro*, subclasse de Veiculo, que tivesse um construtor mais extenso para poder personalizar ao máximo as características do veiculo. Esta classe teria de ter pelo menos uma variável de instância que designaria o tipo do veiculo. Esta foi a maneira mais simples de implementar de modo a resolver o problema. Possivelmente haverá outras maneiras mais complexas e robustas de resolver este problema mas ainda não temos ferramentas para as conseguir implementar.