



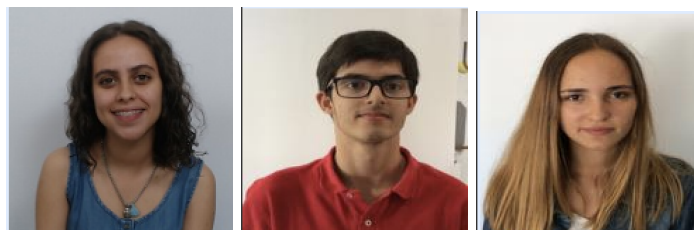
UNIVERSIDADE DO MINHO

DEPARTAMENTO DE INFORMÁTICA

Trabalho Prático
Programação Orientada a Objetos
Grupo 3

Ana Afonso (A85762) Hugo Faria (A81283)
Márcia Teixeira (A80943)

26 de Maio de 2019



(a) Ana.

(b) Hugo.

(c) Márcia.

Conteúdo

1	Introdução	2
1.1	Descrição do Problema	2
1.2	Conceção da Solução	3
2	Classes	3
2.1	User	4
2.1.1	Proprietary	4
2.1.2	Client	5
2.2	Car	5
2.2.1	FuelCar	5
2.2.2	HybridCar	5
2.2.3	EletricCar	6
2.3	Service	6
2.4	Rental	6
2.5	RentalRequest	6
2.6	Weather	7
2.7	WeatherStatus	7
2.8	LoadState	7
3	Classes de Exceções	7
3.1	CarTypeDoesNotExistException	7
3.2	EmailDoesNotExistException	7
3.3	LicensePlateDoesNotExistException	7
3.4	EmailAlreadyExistsException	8
3.5	IncorrectPasswordException	8
4	Implementação e Manual de Utilizador	8
4.1	Registo de Utilizadores	8
4.2	Login	9
4.3	Espaço do Proprietário	9
4.3.1	Registo de um carro	10
4.3.2	Carro registado com sucesso	10
4.3.3	Pedidos de Aluguer	11
4.3.4	Histórico de Alugueres	11
4.4	Espaço do Cliente	12
4.4.1	Aluguer de um carro	12
4.4.2	Tempo de viagem	13
4.4.3	Carro alugado com sucesso	13
4.4.4	Histórico de Alugueres	14
5	Conclusões	15

1 Introdução

O presente relatório descreve o trabalho prático realizado no âmbito da unidade curricular *Programação Orientada a Objetos* (POO).

O objetivo fulcral do projeto assenta em efetuar uma plataforma informática, semelhante à do Airbnb, que consiste na implementação de um sistema de aluguer de carros intitulado *UMCarroJá!*. Esta plataforma permite o aluguer de carros (colocados no sistema por diferentes proprietários) por parte dos clientes, de modo a que cada cliente consiga escolher o carro, mediante as propriedades a este associadas, tendo em conta a sua preferência pessoal.

Neste documento descreve-se sucintamente a aplicação desenvolvida, as classes utilizadas, bem como as estratégias debatidas por todos os elementos do grupo para a elaboração do projeto.

1.1 Descrição do Problema

O projeto proposto integra o desenvolvimento de uma plataforma que permita a criação de um serviço de aluguer de veículos particulares através da internet. Esta aplicação autoriza que o proprietário de um automóvel registre o seu veículo em *UMCarroJá!* e este ser alugado por um cliente registado na mesma aplicação, sendo que o proprietário pode proceder à aceitação ou negação do pedido efetuado pelo cliente interessado.

Encontra-se ainda a necessidade da existência de um histórico de feedback de aluguer de clientes, bem como do automóvel (a fim de informar ao máximo os clientes, de modo a que estes escolham o veículo que desejam).

A localização (em coordenadas GPS) de um veículo é um elemento fundamental a ser considerado não só devido aos veículos particulares se encontrarem estacionados na rua (e não concentrados numa agência de aluguer), mas também por serem de potencial importância para a decisão do cliente (uma vez que este pode optar pelo veículo que se encontra mais perto da sua localização). Um veículo apresenta ainda a indicação do preço cobrado e do consumo médio por quilómetro percorrido, a quantidade de combustível existente (percentagem de gasolina e/ou bateria) e a informação geral do automóvel e do seu proprietário.

Pretende-se ainda que o sistema guarde registo de todas as operações efetuadas e que possua mecanismos para as disponibilizar. O proprietário do veículo é o responsável por abastecer o seu depósito/bateria. Como tal, no ato de aluguer, o cliente deve indicar o destino da viagem de modo a ser possível verificar se o veículo possui a autonomia necessária para o alcançar, uma vez que não é permitido alugar um veículo que não tenha autonomia suficiente para atingir o seu destino. Cada perfil de utilizador deve apenas conseguir aceder às informações e funcionalidades respetivas, assim

Os clientes desta aplicação só podem solicitar o aluguer de um carro:

- mais próximo das suas coordenadas;
- mais barato;
- mais barato dentro de uma distância que estão dispostos a percorrer a pé;
- um carro específico;
- com uma autonomia desejada;

Os proprietários só podem:

- sinalizar o estado de aluguer dos seus automóveis (sendo estes, à espera de aluguer, aceite ou recusado);
- abastecer o veículo;
- alterar o preço por quilómetro;

- aceitar/rejeitar o aluguer de um determinado cliente;
- registar o custo da viagem;

1.2 Conceção da Solução

Para a resolução deste problema foram cruciais quatro momentos.

Numa primeira fase, analisou-se o problema e implementaram-se as classes que seriam necessárias para a realização do trabalho.

Posteriormente, averiguou-se todos os requisitos do trabalho prático e fizeram-se as funções que dão resposta aos objetivos do mesmo.

Em último lugar, criou-se uma interface gráfica de utilizador, com o auxílio da biblioteca *javafx*, que permite o acesso às funcionalidades implementadas.

Por fim, utilizou-se a aplicação JavaFX Scene Builder para a construção do ambiente gráfico. Tomou-se ainda em consideração o nível de programação exigido para a elaboração de uma solução adequada aos requisitos e chegou-se à conclusão que uma abordagem simples e direta seria a resposta.

As classes criadas para o efeito são intuitivas e o projeto é de compreensão acessível a partir do diagrama de classes, contando com a abstração de micro-classes criadas com o objetivo de diminuir a extensão de código existente nas macro-classes.

Desta forma, teve-se como objetivo responder aos requisitos presentes no enunciado e só após o sucesso no supra referenciado, efetuar possíveis melhorias.

O relatório está organizado da seguinte forma: a Secção *Classes* descreve as **classes** adotadas, a Secção *Exceções* refere-se às **classes de exceções** utilizadas. A Secção *Implementação* refere-se à apresentação dos requisitos do trabalho prático e respetiva **implementação e manual de utilizador**.

2 Classes

Com o intuito de detalhar as características das principais entidades do sistema considerou-se necessária a adoção das seguintes classes:

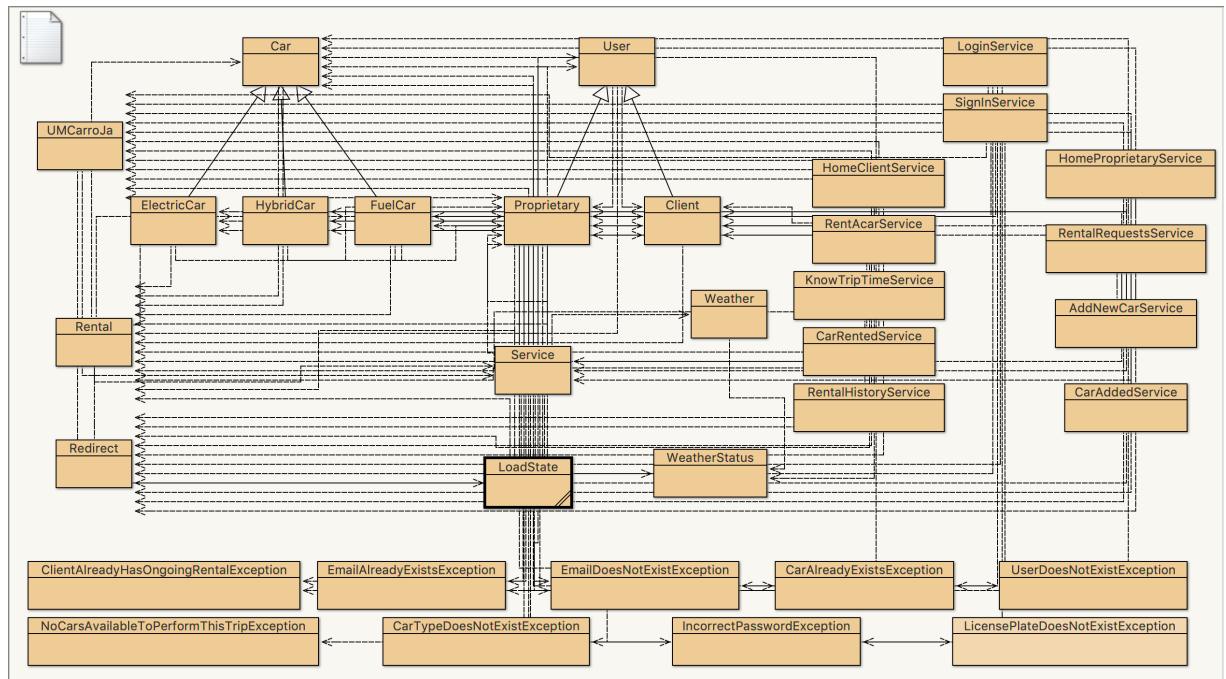


Figura 2: Diagrama de classes BlueJ.

2.1 User

A classe *User* é uma classe abstrata que tem as seguintes variáveis de instância:

```
private String name;
private String email;
private String password;
private String address;
private LocalDate birthDate;
private String nif;
```

O name, email, password, address e nif são atributos essenciais (informação comum) a todos os utilizadores (sejam estes proprietários ou clientes) pelo que se considerou mais útil e intuitivo a elaboração desta classe abstrata.

2.1.1 Proprietary

A classe *Proprietary*, extensão da classe *User*, apresenta outras variáveis de instância que complementam a informação necessária para identificar um proprietário, sendo estas:

```
private List<Car> cars;
private int rating;
private List<Rental> rented;
```

A lista de carros *cars* armazena os carros pertencentes ao proprietário que se encontram no sistema. A lista de alugueres *rented* armazena os alugueres realizados a um proprietário. A variável *rating* contém a avaliação atribuída a um proprietário relativa ao aluguer.

2.1.2 Client

A classe *Client*, extensão da classe *User*, apresenta outras variáveis de instância que complementam a informação necessária para identificar um cliente:

```
private Point2D.Double location;  
private List<Rental> rentals;  
private int rating;  
private int drivingSkill;
```

Para além das variáveis de instância definidas em *User*, o cliente possui ainda uma localização dada pela variável *location* e o histórico de alugueres fornecido pela lista *rentals*. A variável *rating* representa a avaliação atribuída ao cliente pelo proprietário e a variável *drivingSkill* exprime a destreza do cliente a nível de condução.

2.2 Car

A classe *Car* expõe as variáveis de instância necessárias à representação de um veículo:

```
private String brand;  
private int mediumSpeed;  
private double priceKm;  
private double consumeKm;  
private List<Rental> pastRents;  
private int rating;  
private Point2D.Double location;  
private Proprietary proprietary;  
private String licensePlate;  
private int liability;
```

Cada carro é constituído pela sua marca (*brand*), pela sua velocidade média (*mediumSpeed*), pelo seu preço e consumo por quilómetro (*priceKm*) e (*consumeKm*), respetivamente. Do mesmo modo, o veículo também possui uma lista com os alugueres anteriores (*pastRents*), a avaliação atribuída (*rating*), a localização do automóvel (*location*), a matrícula (*licensePlate*) e a fiabilidade associada (*liability*). O veículo apresenta ainda um proprietário (*proprietary*).

2.2.1 FuelCar

```
private double currentFuelAutonomy;  
private double totalFuelAutonomy;
```

Esta subclasse de *Car* tem por objetivo detalhar os componentes extra que especificam os carros movidos a combustível. Assim sendo, definiu-se que a variável *currentFuelAutonomy* representa a autonomia do carro num determinado momento e a variável *totalFuelAutonomy* que representa a autonomia total do carro.

2.2.2 HybridCar

```
private double currentFuelAutonomy;  
private double totalBatteryAutonomy;  
private double currentBatteryAutonomy;  
private double totalFuelAutonomy;
```

Esta subclasse de *Car* tem por objetivo detalhar os componentes extra que especificam os carros híbridos. A autonomia do veículo encontra-se dividida em autonomia do combustível e da bateria, uma vez que se trata de um veículo híbrido. Assim sendo, definiram-se que as variáveis *currentFuelAutonomy* e *currentBatteryAutonomy* representam a autonomia do carro num determinado momento e as variáveis *totalFuelAutonomy* e *totalBatteryAutonomy* correspondem à autonomia total do carro.

2.2.3 EletricCar

```
private double currentBatteryAutonomy;  
private double totalBatteryAutonomy;
```

Esta subclasse de *Car* tem por objetivo detalhar os componentes extra que especificam os carros movidos a bateria. Assim sendo, definiu-se que a variável *currentBatteryAutonomy* representa a autonomia do carro num determinado momento e a variável *totalBatteryAutonomy* que representa a autonomia total do carro.

2.3 Service

```
private Map<String, FuelCar> fuelCars;  
private Map<String, ElectricCar> electricCars;  
private Map<String, HybridCar> hybridCars;  
private Map<String, Client> allClients;  
private Map<String, Proprietary> allProprietaries;
```

Na classe *Service* reúnem-se todas as funções que realizam interações com a implementação da parte gráfica do projeto. Como tal, é nesta classe que se encontram segregados todos os carros existentes bem como todos os clientes, proprietários e pedidos de aluguer.

2.4 Rental

```
private Car rentedCar;  
private Client client;  
private Point2D.Double initialPosCar;  
private Point2D.Double finalPos;  
private String rentalStatus;  
private LocalDateTime rentalDate;  
private LocalDateTime useStartDate;  
private LocalDateTime useFinishDate;  
private int rating;
```

A classe *Rental* contém toda a informação subjacente à efetivação de um aluguer, nomeadamente o automóvel alugado (*rentedCar*), o cliente que procedeu ao aluguer (*client*), o ponto de partida do carro (*initialPosCar*), o ponto de chegada, quer do cliente quer do veículo (*finalPos*), o estado do aluguer (*rentalStatus* - que neste momento só pode ser rejeitado ou aceite), a data de efetuação do aluguer (*rentalDate*), a data de início de uso de uma viatura (*useStartDate*), a data de final de uso de uma viatura (*useFinishDate*) e por fim, possui a avaliação atribuída no final do aluguer (*rating*).

2.5 RentalRequest

```
private Client client;  
private Point2D.Double destiny;
```

```
private String fuelType;  
private String preference;
```

A classe *RentalRequest* contém toda a informação subjacente ao pedido de aluguer. Para tal, alberga o cliente que efetua o pedido (*client*), o destino desejado (*destiny*), o tipo de carro definido pelo seu combustível (*fuelType*) e a preferência em termos de acessibilidade local e/ou monetária (*preference*).

2.6 Weather

```
private List<WeatherStatus> weathers;
```

A classe *Weather* foi criada com o intuito de armazenar todos os estados possíveis de meteorologia.

2.7 WeatherStatus

```
private String weatherATM;  
private int tripTimePenalty;
```

A classe *WeatherStatus* tem como objetivo atribuir a cada estado meteorológico uma penalização no tempo de viagem.

2.8 LoadState

```
private Map<String, Proprietary> allProprietaries;  
private Map<String, Client> allClients;  
private Map<String, FuelCar> fuelCars;  
private Map<String, ElectricCar> electricCars;  
private Map<String, HybridCar> hybridCars;  
private List<RentalRequest> rentalRequests;
```

A classe *loadState* é responsável pelo carregamento inicial de dados para o sistema.

3 Classes de Exceções

A fim de tratar, de forma adequada, eventuais exceções que possam surgir aquando do desenvolvimento do presente projeto, existiu a necessidade de criar classes *Exception*.

3.1 CarTypeDoesNotExistException

A classe *CarTypeDoesNotExistException* refere-se a um tipo de carro ao qual se está a tentar aceder, mas que não existe.

3.2 EmailDoesNotExistException

A classe *EmailDoesNotExistException* refere-se a um email ao qual se está a tentar aceder, mas não existe.

3.3 LicensePlateDoesNotExistException

A classe *LicensePlateDoesNotExistException* refere-se a uma matrícula à qual se está a tentar aceder, mas não existe.

3.4 EmailAlreadyExistsException

A classe *EmailAlreadyExistsException* refere-se a uma tentativa de inserção de um email existente (o que não é possível).

3.5 IncorrectPasswordException

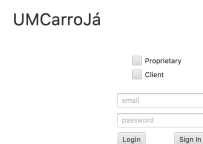
A classe *IncorrectPasswordException* refere-se a uma tentativa de acesso a uma dada conta com uma palavra-passe inválida (o que não é possível).

4 Implementação e Manual de Utilizador

Para a concretização do presente trabalho e através do auxílio das já mencionadas classes, pensa-se que os requisitos propostos se encontram realizados.

Nas seguintes imagens apresenta-se o programa já pré-populado com um conjunto de dados significativos, que permite testar toda a aplicação.

O ponto de partida do programa encontra-se na figura abaixo, onde os utilizadores podem fazer Login ou Sign In.



The screenshot shows the 'UMCarroJá' login interface. At the top, there are two radio buttons: 'Proprietary' and 'Client'. Below these are two input fields labeled 'email' and 'password'. At the bottom, there are two buttons: 'Login' and 'Sign In'.

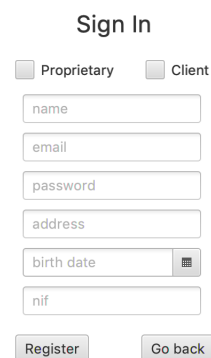
Figura 3: Demonstração da página inicial do programa.

De seguida, apresenta-se uma explicação sucinta acerca da resolução implementada e a demonstração na aplicação.

4.1 Registo de Utilizadores

- **Registar um utilizador, seja este proprietário ou cliente.**

De forma a responder ao ponto acima mencionado, depois do utilizador se deparar com a página inicial do programa, se a intenção do mesmo for **efetuar um registo de um novo utilizador** deverá clicar no botão Sign In que o redirecionará para a seguinte figura.



The screenshot shows the 'Sign In' registration form. At the top, there are two radio buttons: 'Proprietary' and 'Client'. Below these are several input fields: 'name', 'email', 'password', 'address', 'birth date' (with a calendar icon), and 'nif'. At the bottom, there are two buttons: 'Register' and 'Go back'.

Figura 4: Registo de um novo utilizador.

Deste modo, o utilizador terá que preencher obrigatoriamente todos os campos e seleccionar se pretende realizar um registo enquanto proprietário ou cliente.

4.2 Login

- **Validar o acesso à aplicação utilizando as credenciais (email e password), por parte de diferentes atores.**

Como já apresentado na Figura??, se a intenção do utilizador for efetuar login na aplicação deverá preencher os campos apresentados nesse ecrã, bem como seleccionar se pretende aceder à plataforma como proprietário ou cliente.

O nosso sistema possui dois tipos de atores: os proprietários e clientes.

Assim, no nosso sistema de input output é averiguado se foram introduzidas as credenciais corretas de um utilizador e em caso afirmativo, este é enviado para a respetiva dashboard.

No caso de se tratar de um proprietário, este é enviado para a Figura??.

No caso de se tratar de um cliente, este é enviado para a Figura??.

4.3 Espaço do Proprietário

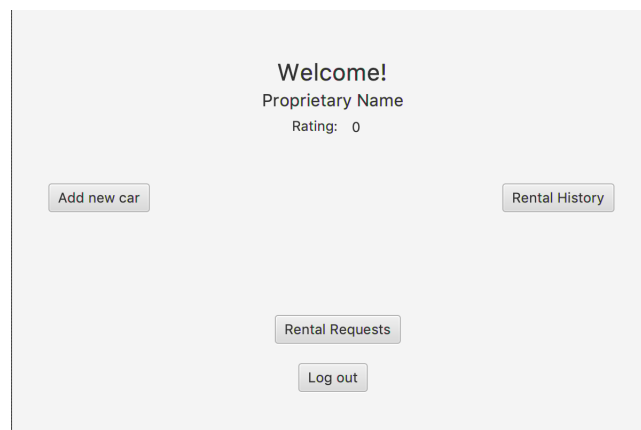
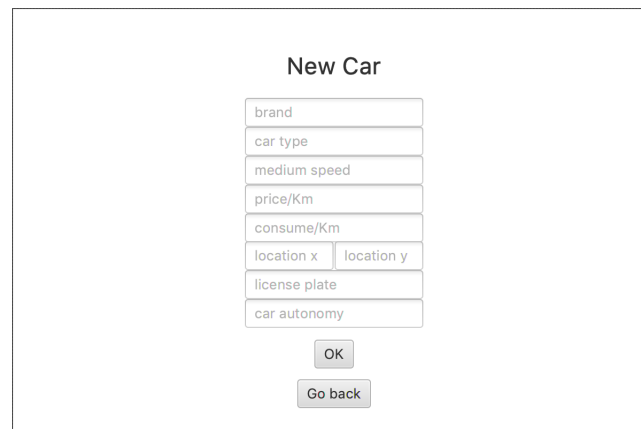


Figura 5: Espaço pessoal do proprietário.

Neste espaço, o proprietário pode registar um carro, aceder aos pedidos de aluguer dos seus veículos e consultar o seu histórico de alugueres anteriores.

4.3.1 Registo de um carro

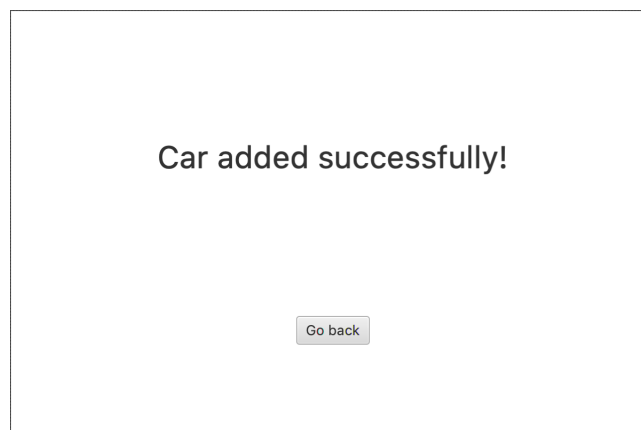


A screenshot of a web form titled "New Car". The form contains several input fields: "brand", "car type", "medium speed", "price/Km", "consume/Km", "location x" and "location y" (side-by-side), "license plate", and "car autonomy". Below the fields are two buttons: "OK" and "Go back".

Figura 6: Registo de um carro.

Neste espaço, o proprietário preenche os dados inerentes ao registo de um carro.

4.3.2 Carro registado com sucesso



A screenshot of a confirmation message box. It has a light gray background and a thin border. The text "Car added successfully!" is centered in a bold, black font. Below the text is a single button labeled "Go back".

Figura 7: Carro registado com sucesso.

Esta mensagem avisa o proprietário que o carro foi registado com sucesso.

4.3.3 Pedidos de Aluguer



Figura 8: Pedidos de Aluguer.

Neste espaço encontram-se os pedidos de aluguer feitos pelo cliente a um proprietário.

4.3.4 Histórico de Alugueres

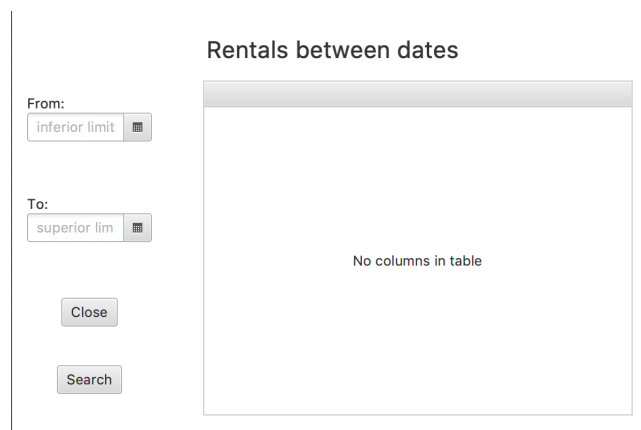
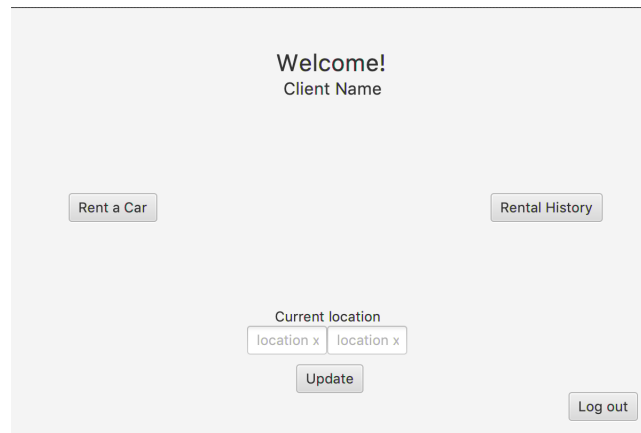


Figura 9: Histórico de Alugueres.

Neste espaço encontra-se guardado o histórico de todos os alugueres feitos a um proprietário.

4.4 Espaço do Cliente

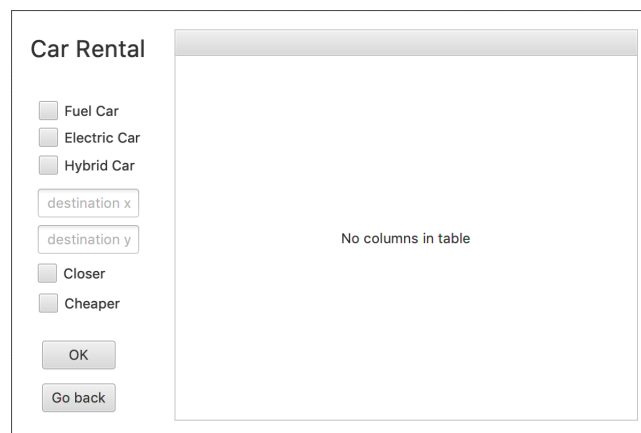


A user dashboard interface with a light gray background. At the top center, it says "Welcome!" followed by "Client Name". Below this, there are two buttons: "Rent a Car" on the left and "Rental History" on the right. In the center, there is a section titled "Current location" containing two input fields labeled "location x" and "location x", and an "Update" button below them. In the bottom right corner, there is a "Log out" button.

Figura 10: Espaço pessoal do cliente.

Neste espaço, o cliente pode alugar um carro e após o aluguer consultar o tempo estimado de viagem e ainda aceder ao seu histórico de alugueres anteriores.

4.4.1 Aluguer de um carro



A car rental form titled "Car Rental". On the left side, there are three checkboxes: "Fuel Car", "Electric Car", and "Hybrid Car". Below these are two input fields labeled "destination x" and "destination y". Further down are two more checkboxes: "Closer" and "Cheaper". At the bottom left are two buttons: "OK" and "Go back". On the right side, there is a large rectangular area that currently displays the text "No columns in table".

Figura 11: Aluguer de um carro.

Neste espaço, o cliente poderá realizar o aluguer de um carro preenchendo os campos fornecidos.

4.4.2 Tempo de viagem

Weather Status:

▼

Estimate trip time: 0

Driving Skill (0-5):

Calculate

Next

Figura 12: Tempo de viagem.

Neste espaço, o cliente pode saber uma estimativa do tempo de viagem, se assim o entender.

4.4.3 Carro alugado com sucesso

Car rented successfully!

Terminate

Figura 13: Carro alugado com sucesso.

Esta mensagem avisa o cliente que o carro foi alugado com sucesso.

4.4.4 Histórico de Alugueres

From:

inferior limit

To:

superior lim

Close

Search

Rentals between dates

No columns in table

Figura 14: Histórico de Alugueres.

Neste espaço encontra-se guardado o histórico de todos os alugueres feitos por um cliente.

5 Conclusões

Face ao problema apresentado, o grupo procedeu a uma análise detalhada acerca da melhor estratégia a adotar e seguidamente operou o discutido.

Para a parte gráfica, o grupo tentou ao máximo criar um ambiente intuitivo e apelativo para um melhor entendimento acerca do funcionamento da plataforma, por parte do utilizador. Apresentando ainda a consciência que esta poderia ter sido melhor implementada, por exemplo através da colocação de animações (tais como, a transição entre páginas e o modo de apresentação de avisos de exceções) não foi realizada, uma vez que a utilização deste novo tipo de estruturas não nos ser exposta como algo empírico e apesar da alargada panóplia que se foi desenvolvendo ao longo do presente projeto, essa implementação ainda surge de forma muito restrita face ao atual conhecimento, que o escasso tempo anula a possibilidade de uma mais vasta aprendizagem.

Salienta-se também, o empenho para que neste trabalho estivesse presente um bom código, com uma performance.

Em suma, não obstante às potenciais melhorias que poderiam ter sido realizadas, nomeadamente fazer com que o sistema aceite mais fatores para a estimativa de tempo de viagem (apresenta-se como condicionantes, a destreza do condutor e a meteorologia) e otimização a nível de código, considera-se que o projeto realizado cumpre todos os requisitos pedidos.