



Instituto Superior Politécnico de Tecnologias e Ciências
(ISPTEC)
Departamento de Engenharia e Tecnologias

Engenharia de Software II-Tópicos Especiais III

Relatório

Desenvolvimento de um Aplicativo de Táxi - uTaxi

Elaborado por:	Bissala Pedro - 20130850
Curso	Engenharia Informática
Turma	Especial
Ano académico	2022/2023
Docente	Judson Quissanga Coge Paiva

Luanda, 03 de Julho de 2023

Índice

Introdução	1
Metodologia	1
Diagrama de classe	2
Justificação das classes	2
Organização do projecto	3
Resultados	7
Conclusão	10
Referências Bibliográficas	11

Introdução

Os serviços de táxi desempenham um papel fundamental na mobilidade e no transporte público no nosso país. Oferecem uma forma conveniente e acessível de deslocamento para os cidadãos, especialmente nas áreas urbanas, onde o transporte público pode ser limitado ou inadequado.

Os táxis proporcionam uma opção de transporte acessível para pessoas que não possuem veículos particulares ou que desejam evitar os custos de possuir um carro, como manutenção, seguro e combustível. Isso permite que os passageiros se desloquem de forma rápida e conveniente pela cidade, sem as restrições associadas ao uso do transporte público ou a necessidade de caminhar longas distâncias.

O setor de táxis em Angola é uma importante fonte de emprego para muitos motoristas. Ao trabalhar como taxista, os indivíduos têm a oportunidade de ganhar uma renda e sustentar suas famílias. Além disso, o setor de táxis também contribui indiretamente para a economia local por meio do pagamento de impostos e taxas.

As Tecnologias de Informação e Comunicação auxiliam na dinamização em diversas formas, o desenvolvimento de um aplicativo de Táxi não se ausenta disso. Foi-me proposto um problema relacionado com serviços de transporte de passageiros que faça a concorrência a um serviço muito conhecido como a UTEC, pois pretende-se que a aplicação a ser desenvolvida dê suporte a toda a funcionalidade que permita que um utilizador realize uma viagem num dos táxis da UTEC. O processo deve abranger todos os mecanismos de criação de utilizadores, motoristas, automóveis e posteriormente a marcação das viagens, a realização das mesmas e respectiva imputação do preço. Pretende-se também que o sistema guarde registo de todas as operações efectuadas e que depois tenha mecanismos para as disponibilizar (exemplo: viagens de um utilizador, extracto de viagens de um taxi num determinado período, valor facturado por um taxi num determinado período, etc.).

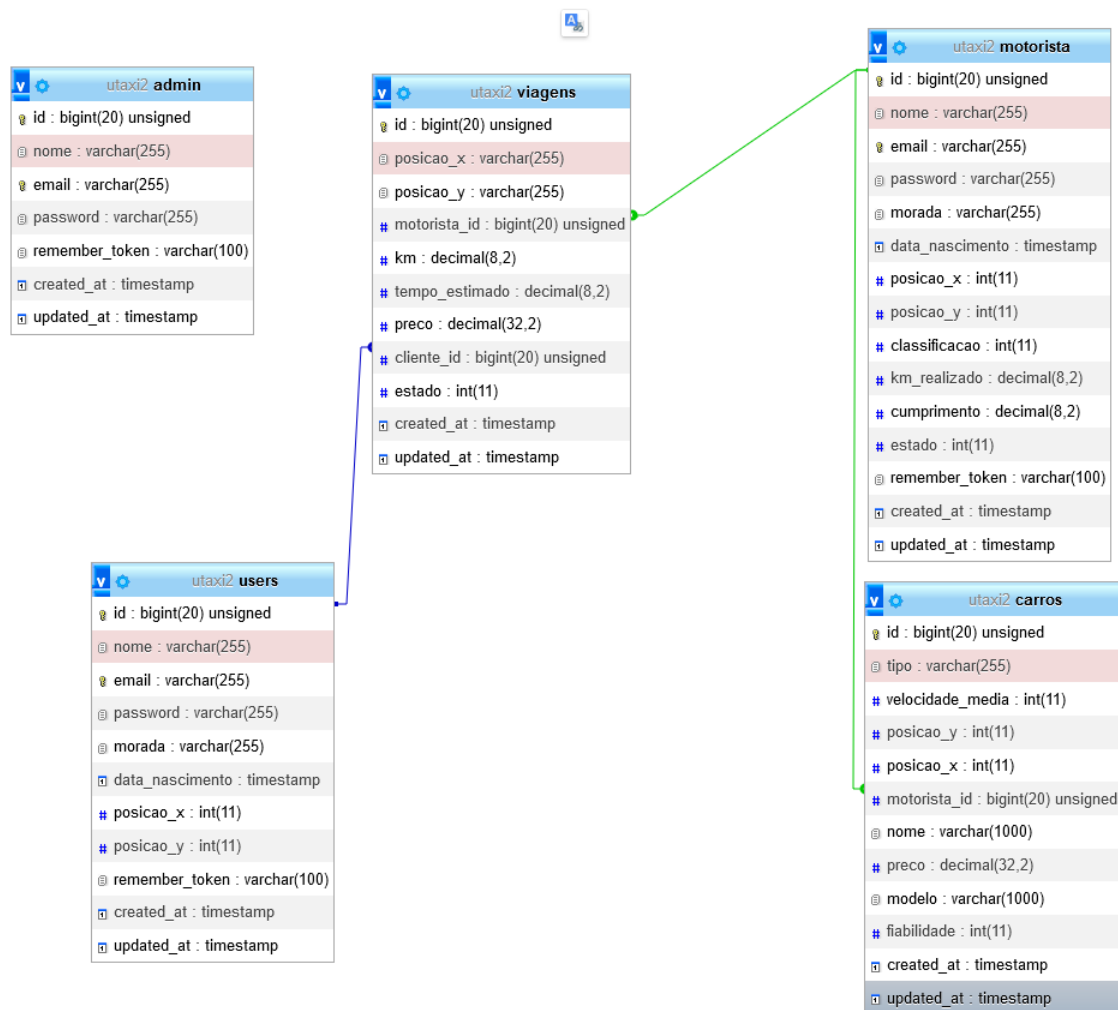
Cada perfil de utilizador deve apenas conseguir aceder às informações e funcionalidades respectivas.

Com base as informações supracitadas, criou-se um aplicativo cujo nome é **uTaxi**.

Metodologia

Para o projecto usou-se PHP, HTML5 e CSS3 de acordo com modelo de **MVC** (Model View Controller) através da Framework Laravel devido as suas características de fácil uso e que oferece uma estrutura robusta e elegante para desenvolver aplicativos de alta qualidade. Sobre a base de dados achou-se conveniente usar o MySQL através de PHPMyAdmin do Laragon.

Diagrama de classe



Justificação das classes

Classe Usuário: Representa um usuário do aplicativo de táxi. Armazena informações como ID, nome, sobrenome, email, senha, número de telefone e endereço do usuário.

Classe Motorista: Representa um motorista registrado no aplicativo de táxi. Possui atributos como ID, nome, sobrenome, email, senha, número de telefone, matrícula, data de nascimento e classificação média dada pelos passageiros.

Classe Carros: Representa um veículo utilizado pelos motoristas do aplicativo de táxi. Armazena informações sobre o veículo, como ID, marca, modelo, ano, cor, número de assentos, placa e status (disponível, em serviço, fora de serviço).

Classe Viagem do usuário: Representa uma viagem solicitada por um usuário. Contém informações sobre a viagem, como ID, ID do usuário que solicitou a viagem, ID do motorista atribuído à viagem, ID do veículo, local de origem, destino, data e hora de partida, data e hora de chegada, valor da viagem e status da viagem.

Classe Pagamento: Representa um pagamento realizado para uma viagem. Armazena informações como ID do pagamento, ID da viagem associada, método de pagamento utilizado, valor do pagamento e data e hora em que o pagamento foi realizado.

Classe Avaliação: Representa uma avaliação dada por um usuário a um motorista após uma viagem. Contém informações como ID da avaliação, ID da viagem associada, ID do usuário que fez a avaliação, ID do motorista avaliado, classificação dada, comentário e data e hora da avaliação.

Organização do projecto

O projeto está organizado de acordo com o padrão de arquitetura MVC (Model-View-Controller) que adaptou-se com o framework Laravel e linguagem de programação que foi utilizado.

Pasta "Models" (Modelos): Nesta pasta, foram armazenadas as classes que representam os modelos de dados do aplicativo, responsáveis por interagir com o banco de dados ou fornecer lógica de negócios. Cada modelo é representado por um arquivo separado.

Pasta "Views" (Visualizações): Armazenou-se os arquivos relacionados à interface do usuário do seu aplicativo, como templates HTML, arquivos de exibição de dados, arquivos CSS e imagens. A estrutura de pastas dentro de "Views" é organizada com base nos diferentes recursos ou módulos do aplicativo.

Pasta "Controllers" (Controladores): Nesta pasta, foram colocados os controladores do aplicativo. Cada controlador é responsável por receber as solicitações dos usuários, processar os dados e interagir com os modelos e as visualizações apropriados. Cada controlador é representado por um arquivo separado.

Pasta "Routes" (Rotas): Foram definidas as rotas do aplicativo que mapeiam as URLs recebidas para os controladores e ações correspondentes. Essas rotas são responsáveis por direcionar as solicitações HTTP para os controladores corretos.

Pasta "Libraries" (Bibliotecas): O aplicativo tem bibliotecas ou classes auxiliares que fornecem funcionalidades específicas compartilhadas entre os

modelos, controladores ou visualizações. Essas classes são armazenadas em uma pasta "Libraries" separada.

Pasta "Config" (Configurações): É nesta pasta por onde são armazenados arquivos de configuração do aplicativo, como configurações de banco de dados, configurações de ambiente, configurações de roteamento, etc.

Pasta "Public" (Público): Esta pasta contém os recursos públicos do aplicativo, como arquivos CSS, imagens e scripts JavaScript. Esses arquivos são acessíveis diretamente pelos navegadores dos usuários.

```

1 <?php
2
3 namespace App\Http\Controllers\Admin;
4
5 use App\Http\Controllers\Controller;
6 use App\Models\Carros;
7 use App\Models\Motoristas;
8 use App\Models\User;
9 use App\Rules\Dinheiro;
10 use Illuminate\Http\Request;
11
12 class CarrosController extends Controller
13 {
14     function index(){
15         return view("admin.carros.index");
16     }
17
18     function store(Request $request){
19
20         $request->validate([
21             "nome"=>["required"],
22             "modelo"=>["required"],
23             "preco"=>["required", new Dinheiro("O preço esta mal formatado")],
24             "tipo"=>["required"],
25             "velocidade_media"=>["required", "integer"],
26             "fiabilidade"=>["required", "integer"],
27             "p_y"=>["required", "integer"],
28             "p_x"=>["required", "integer"],

```

Fig1: Controllers colocado no diretório HTTP, com a descrição supracitada.

```

1 <?php
2
3 namespace App\Models;
4 use Illuminate\Foundation\Auth\User as Authenticatable;
5 use Illuminate\Database\Eloquent\Model;
6
7 /**
8  * App\Models\Admin
9  *
10  * @method static \Illuminate\Database\Eloquent\Builder|Admin newModelQuery()
11  * @method static \Illuminate\Database\Eloquent\Builder|Admin newQuery()
12  * @method static \Illuminate\Database\Eloquent\Builder|Admin query()
13  * @property int $id
14  * @property string $nome
15  * @property string $email
16  * @property string $password
17  * @property string|null $remember_token
18  * @property \Illuminate\Support\Carbon|null $created_at
19  * @property \Illuminate\Support\Carbon|null $updated_at
20  * @method static \Illuminate\Database\Eloquent\Builder|Admin whereCreatedat($value)
21  * @method static \Illuminate\Database\Eloquent\Builder|Admin whereEmail($value)
22  * @method static \Illuminate\Database\Eloquent\Builder|Admin whereId($value)
23  * @method static \Illuminate\Database\Eloquent\Builder|Admin whereNome($value)
24  * @method static \Illuminate\Database\Eloquent\Builder|Admin wherePassword($value)
25  * @method static \Illuminate\Database\Eloquent\Builder|Admin whereRememberToken($value)
26  * @method static \Illuminate\Database\Eloquent\Builder|Admin whereUpdatedAt($value)
27  * @mixin \Eloquent
28  */

```

Fig2: Models conforme a descrição supracitada.

```

1 @extends("admin.app", ["titulo"=>"Adicionar cliente"])
2
3
4 @section("content")
5     <div class="container mt-5">
6
7         <div class="row justify-content-center">
8             <div class="col-md-6">
9
10                 <div class="card">
11                     <div class="card-header p-3"><h2 class="m-0">Adicionar cliente</h2></div>
12                     @if ($errors->count() != 0)
13                         <div class="p-3">
14                             <div class="alert alert-danger">{{ $errors->first() }}</div>
15                         </div>
16                     @endif
17                     <div class="card-body">
18                         <form method="post" action="/admin/clients/add">
19                             @csrf
20
21                             <div class="form-group mb-3">
22                                 <label>Nome</label>
23                                 <input value="{{old('nome')}}" type="text" placeholder="Nome" class="form-control" name="nome">
24                             </div>
25
26                             <div class="form-group mb-3">
27                                 <label>Email</label>
28                                 <input value="{{old('email')}}" placeholder="Email" type="email" class="form-control" name="email">

```

Fig3: Uma das views conforme a descrição supracitada.

```

70 Route::post('/mudar_estado', [\App\Http\Controllers\Motorista\HomeMotorista::class, "mudar_estado"]);
71
72
73 Route::prefix("viagens")->group(function(){
74     Route::post('/processar', [\App\Http\Controllers\Motorista\ViagensController::class, "processViagem"]);
75     Route::post('/confirmar', [\App\Http\Controllers\Motorista\ViagensController::class, "viagemConfirmar"]);
76     Route::post('/publicar', [\App\Http\Controllers\Motorista\ViagensController::class, "publicarViagem"]);
77
78     Route::get('/fila', [\App\Http\Controllers\Motorista\ViagensController::class, "fila"]);
79     Route::get('/realizadas', [\App\Http\Controllers\Motorista\ViagensController::class, "realizadas"]);
80
81     Route::get('/{id}', [\App\Http\Controllers\Motorista\ViagensController::class, "show"]->withoutMiddleware([CheckViagemM]);
82     Route::post('/mudar_preco/{id}', [\App\Http\Controllers\Motorista\ViagensController::class, "alterarPreco"]->withoutMid;
83     Route::post('/concluido/{id}', [\App\Http\Controllers\Motorista\ViagensController::class, "marcarConcluido"]->withoutMi;
84     Route::get('/atender/{id}', [\App\Http\Controllers\Motorista\ViagensController::class, "atender"]);
85     })->middleware(["auth:motorista"]);
86
87     Route::post("/login", [LoginController::class, "loginMotorista"]->withoutMiddleware([CheckViagemMotoristaMiddleware::class];
88     Route::get("/login", [LoginController::class, "loginMotoristaView"]->withoutMiddleware([CheckViagemMotoristaMiddleware::cla;
89
90     Route::get("/register", [LoginController::class, "registerMotoristaView"]->withoutMiddleware([CheckViagemMotoristaMiddleware;
91     Route::post("/register", [LoginController::class, "registerMotorista"]->withoutMiddleware([CheckViagemMotoristaMiddleware::c;
92
93     Route::get("/", [\App\Http\Controllers\Motorista\HomeMotorista::class, "index"]->middleware(["auth:motorista"]);
94 });
95
96 Route::get("/sair", [IndexController::class, "sair"]);
97 Route::post("/login", [LoginController::class, "loginCliente"]);

```

Fig4: As rotas conforme a descrição supracitada.

```

1 APP_NAME=Laravel
2 APP_ENV=local
3 APP_KEY=base64:Cbn2YCz9/yGlp025Jo+jk5dKWOV1rVfIoIdfLEDQW=
4 APP_DEBUG=true
5 APP_URL=http://localhost
6
7 LOG_CHANNEL=stack
8 LOG_DEPRECATIONS_CHANNEL=null
9 LOG_LEVEL=debug
10
11 DB_CONNECTION=mysql
12 DB_HOST=127.0.0.1
13 DB_PORT=3306
14 DB_DATABASE=utaxi
15 DB_USERNAME=root
16 DB_PASSWORD=
17
18 BROADCAST_DRIVER=log
19 CACHE_DRIVER=file
20 FILESYSTEM_DISK=local
21 QUEUE_CONNECTION=sync
22 SESSION_DRIVER=file
23 SESSION_LIFETIME=120
24
25 MEMCACHED_HOST=127.0.0.1
26
27 REDIS_HOST=127.0.0.1
28 REDIS_PASSWORD=null

```

Fig5: Requisições de envio e recebimento de respostas entre php e base de dados.

```

35
36 'connections' => [
37
38     'sqlite' => [
39         'driver' => 'sqlite',
40         'url' => env('DATABASE_URL'),
41         'database' => env('DB_DATABASE', database_path('database.sqlite')),
42         'prefix' => '',
43         'foreign_key_constraints' => env('DB_FOREIGN_KEYS', true),
44     ],
45
46     'mysql' => [
47         'driver' => 'mysql',
48         'url' => env('DATABASE_URL'),
49         'host' => env('DB_HOST', '127.0.0.1'),
50         'port' => env('DB_PORT', '3306'),
51         'database' => env('DB_DATABASE', 'forge'),
52         'username' => env('DB_USERNAME', 'forge'),
53         'password' => env('DB_PASSWORD', ''),
54         'unix_socket' => env('DB_SOCKET', ''),
55         'charset' => 'utf8mb4',
56         'collation' => 'utf8mb4_unicode_ci',
57         'prefix' => '',
58         'prefix_indexes' => true,
59         'strict' => true,
60         'engine' => null,
61         'options' => extension_loaded('pdo_mysql') ? array_filter([
62             PDO::MYSQL_ATTR_SSL_CA => env('MYSQL_ATTR_SSL_CA'),

```

Fig6: Estrutura completa de todas as requisições das conexões da bases de dados.

Resultados

Em função do pedido e como resposta desenvolveu-se um aplicativo uTaxi para atender a demanda.

Em anexo os seguintes resultados:

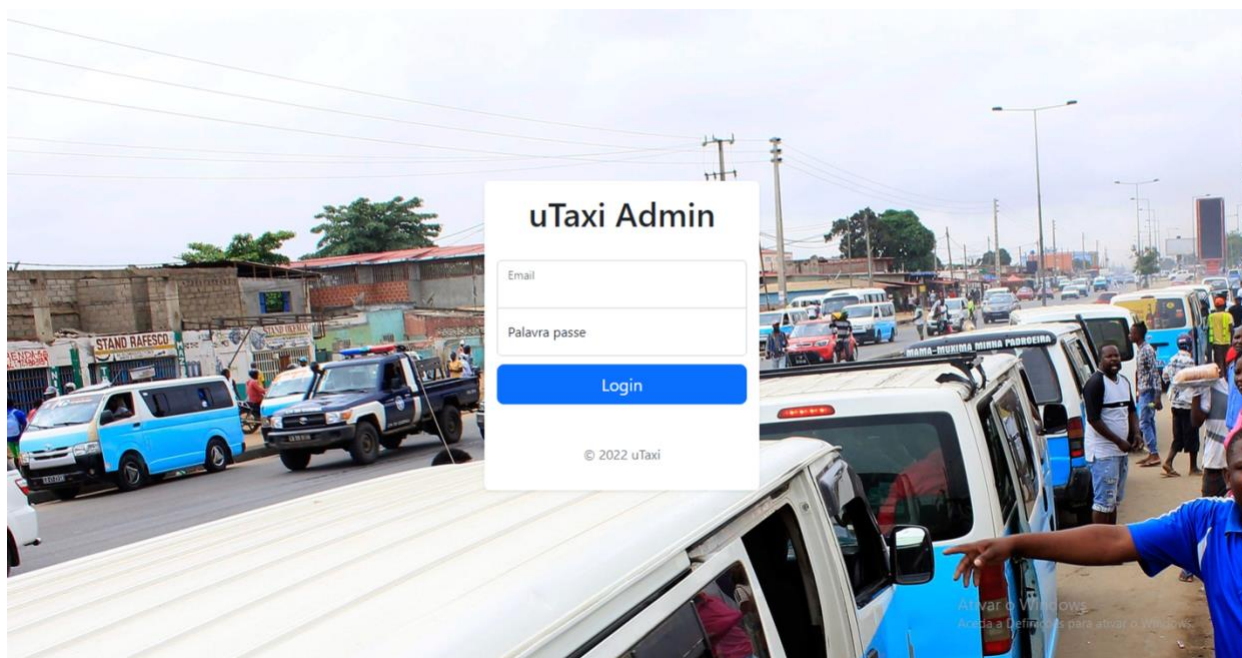


Fig7: Login para os administradores do sistema.

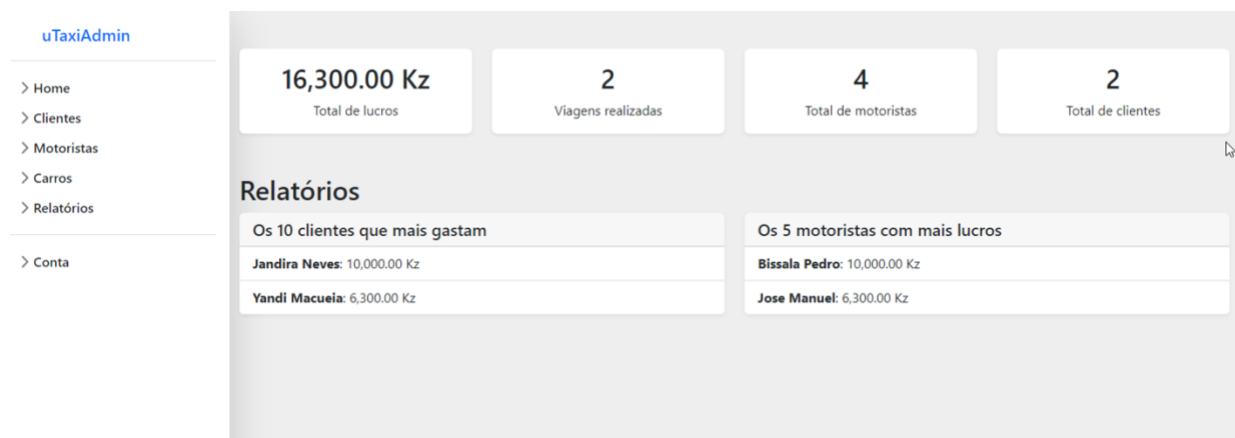


Fig8: Estatísticas da gerência total do sistema disponível a penas para os administradores.

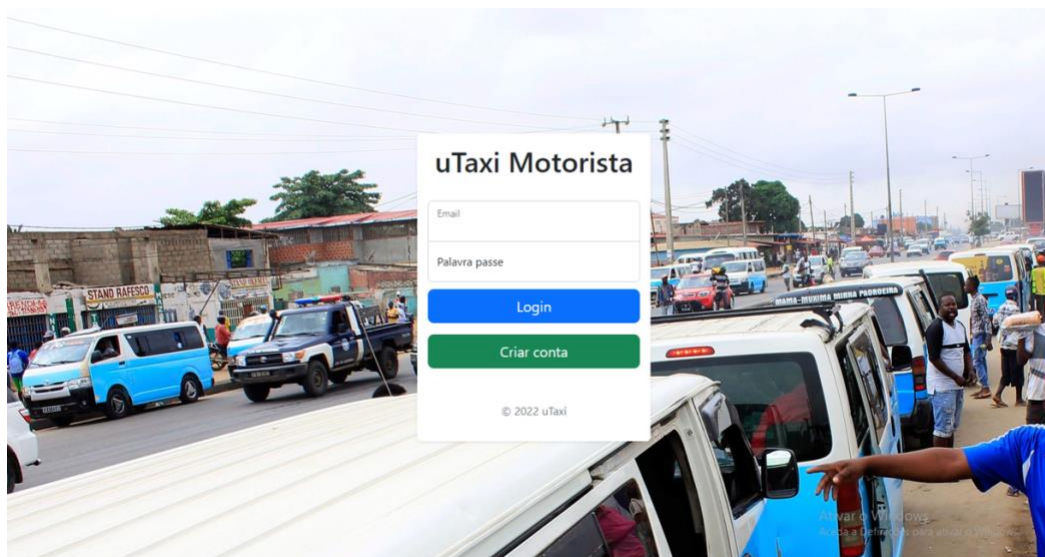


Fig9: Login para os motoristas.

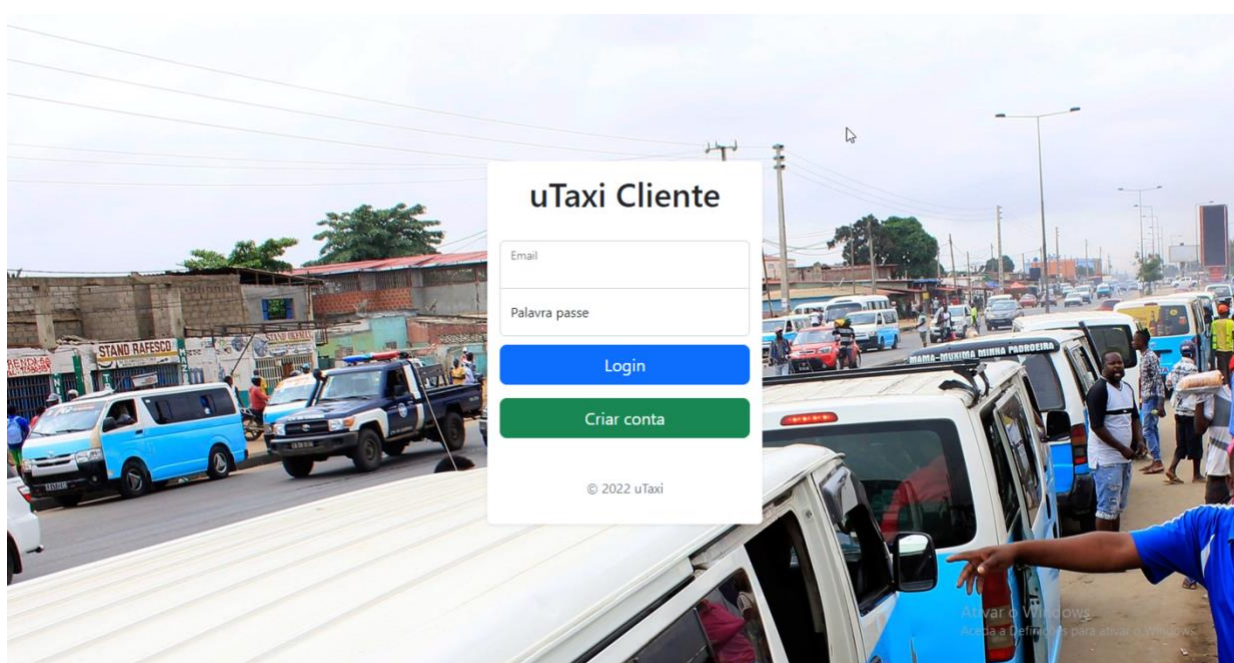


Fig10: Tela de login para clientes.

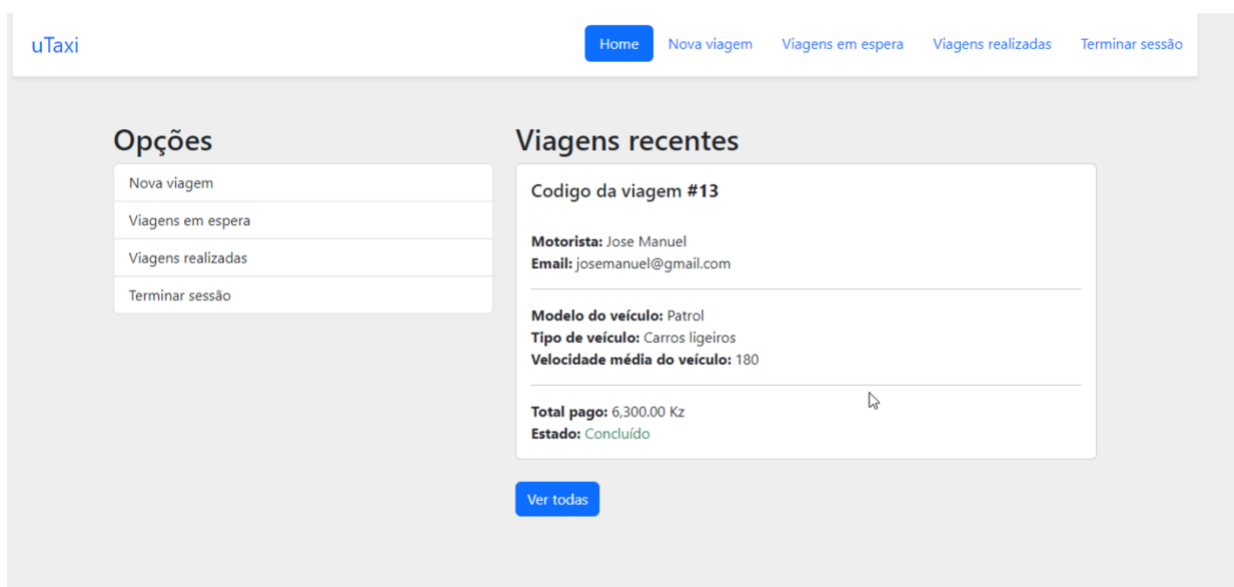
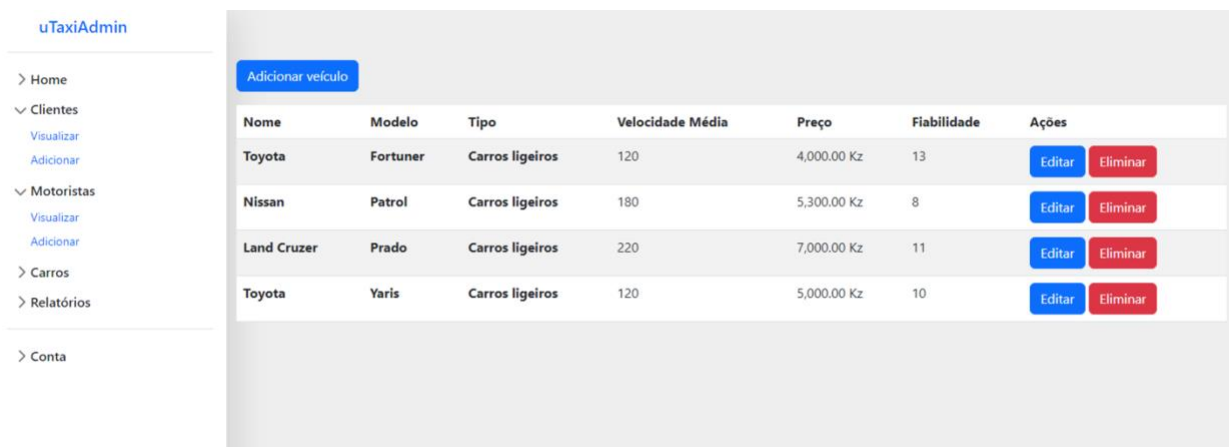


Fig11: Ambiente geral do cliente.



The screenshot displays the uTaxiAdmin web application. On the left is a sidebar menu with the following items: Home, Clientes (with sub-items Visualizar and Adicionar), Motoristas (with sub-items Visualizar and Adicionar), Carros, Relatórios, and Conta. The main content area features a blue button labeled 'Adicionar veículo' at the top left. Below it is a table with the following columns: Nome, Modelo, Tipo, Velocidade Média, Preço, Fiabilidade, and Ações. The table contains four rows of vehicle data, each with 'Editar' and 'Eliminar' buttons in the 'Ações' column.

Nome	Modelo	Tipo	Velocidade Média	Preço	Fiabilidade	Ações
Toyota	Fortuner	Carros ligeiros	120	4,000.00 Kz	13	<button>Editar</button> <button>Eliminar</button>
Nissan	Patrol	Carros ligeiros	180	5,300.00 Kz	8	<button>Editar</button> <button>Eliminar</button>
Land Cruiser	Prado	Carros ligeiros	220	7,000.00 Kz	11	<button>Editar</button> <button>Eliminar</button>
Toyota	Yaris	Carros ligeiros	120	5,000.00 Kz	10	<button>Editar</button> <button>Eliminar</button>

Fig12: Gerenciamento dos veículos por parte dos administradores.

Conclusão

Em suma, o aplicativo funciona na versão web conforme a orientação do Sr Professor de poder escolher quer web ou Mobile conforme o tempo.

A maior dificuldade consistiu no que diz respeito a implementação das coordenadas euclidianas para os pontos(x,y) da parte do cliente para a origem e o destino.

Por outro lado, a separação de responsabilidades, pois o MVC exige uma clara separação de responsabilidades entre os modelos, controladores e visualizações. Isso significa que se precisa entender como dividir e organizar a lógica de negócios, a interação com o banco de dados e a apresentação de dados de maneira adequada. Identificar corretamente qual código pertence a cada componente foi um desafio.

Referências Bibliográficas

MVC:

"Pro ASP.NET MVC 5" por Adam Freeman (2013)

"Clean Architecture: A Craftsman's Guide to Software Structure and Design" por Robert C. Martin (2017)

"Domain-Driven Design: Tackling Complexity in the Heart of Software" por Eric Evans (2003)

Laravel:

"Laravel: Up and Running" por Matt Stauffer (2016)

"Laravel: From Apprentice To Artisan" por Taylor Otwell (2019)

"Learning Laravel 8: Building Practical Applications" por Hardik Dayani (2021)

PHP:

"PHP: The Good Parts" por Peter MacIntyre (2010)

"Modern PHP: New Features and Good Practices" por Josh Lockhart (2015)

"PHP Objects, Patterns, and Practice" por Matt Zandstra (2017)

HTML5:

"HTML5: Up and Running" por Mark Pilgrim (2010)

"HTML5 and CSS3: Building Responsive Websites" por Alok Prasad (2011)

"HTML5: The Missing Manual" por Matthew MacDonald (2014)

CSS3:

"CSS3: The Missing Manual" por David Sawyer McFarland (2012)

"CSS: The Definitive Guide" por Eric A. Meyer (2017)

"CSS3 in Easy Steps" por Mike McGrath (2013)