



**Escola Superior  
de Tecnologia  
e Gestão**

Politécnico de Coimbra

**Licenciatura em Engenharia Informática**

**Manual Programador**

**Gestor de Alojamentos**

**Muaiad Mhd Fahd Al Hadad**

Oliveira do Hospital, maio de 2024

# CONTEÚDO

1. Introdução .....	5
1.2 Objetivo do Projeto.....	5
1.3 Tecnologias Utilizadas.....	5
1.4 Funcionalidades Principais.....	5
1.5 Segurança e Privacidade .....	5
2. install project .....	6
2.1 Fazer clone ao projeto de GitHub .....	6
2.2 Instar Laravel .....	6
2.3 Configurar o Banco de Dados .....	6
2.4 Executar o Servidor de Desenvolvimento.....	6
3. desenvolvimento .....	6
3.1 Página Inicial.....	6
3.1.1 Front end.....	6
.....	7
3.1.2 Back end .....	14
3.2 Detalhe.....	19
3.2.1 Front end.....	19
3.2.2 Back-end.....	23
3.3 Login .....	24
3.3.1 Front end.....	24
3.3.2 Back-end .....	25
3.4 Registar .....	28
3.4.1 front end.....	28
3.4.2 Back-end.....	29
2.5 Email enviados.....	31
2.5.1 Back-end.....	31
3.6 Validação código.....	32
3.6.1 Back-end.....	32
3.7 Recuperar a Password.....	35
3.7.1 Back-end.....	35
3.8 Gestor área pessoal .....	36

3.8.1 Front-end .....	36
3.8.2 Back-end.....	37
3.9 Adicionar .....	39
3.9.1 Back-end.....	39
3.10 Profile .....	40
3.10.1 Back-end.....	40
3.11 Back-end remover utilizador .....	42
3.12 Back-end consultar anúncio .....	44
3.13 Back-end aprovar e reprovar anúncio .....	46
3.14 Back-end mudar estado de um anúncio .....	47
3.15 Senhorio área pessoal .....	48
3.15.1 Back-end.....	48
3.16 Editar anúncio.....	49
3.16.1 Back-end.....	49
3.17 Adicionar anúncio .....	53
3.17.1 Front-end .....	53
3.17.2 Back-end.....	57
3.18 ControllerChat.....	60
3.18.1 Métodos .....	60
2.18.2 Back end .....	61
3.19 Adicionar para os favoritos .....	62
3.19.1 back end.....	62

## Conteúdo das figuras

Figure 1-Base Dados .....	6
Figure 2- cabeçalho da página com login e sem .....	7
Figure 3- Java Script de menu de utilizador .....	8
Figure 4- Lógica de mostra os utilizadores .....	9
Figure 5- filtragem .....	11
Figure 6- As ofertas.....	13
Figure 7- recuperar os dados das propriedades imobiliárias para exibição na página inicial.....	14
Figure 8- exibir a página inicial do site .....	16
Figure 9- Filtragem .....	18
Figure 10- fotos de propriedade .....	20
Figure 11-específicos da propriedade .....	21
Figure 12-Map .....	22
Figure 13-recuperar os detalhes de uma casa específica .....	23
Figure 14-login form .....	24
Figure 15-toast.....	25
Figure 16-controller de login .....	26
Figure 17-form registar .....	28
Figure 18-controler registar.....	29
Figure 19-controlar email.....	32
Figure 20-controlar ativação de utilizador.....	33
Figure 21-controlar reenviar código de ativação.....	34
Figure 22-controlar Forgot Password.....	35
Figure 23-controlar de página principal de gestor.....	38
Figure 24-controlar add gestor .....	39
Figure 25-dados do utilizador.....	40
Figure 26-update dados de utilizador .....	41
Figure 27-remover utilizador .....	42
Figure 28-consultar anúncio .....	44
Figure 29-Aprovar e reprovar anúncio .....	46
Figure 30-mudar estado de um anúncio .....	47
Figure 31-controlar área pessoal senhorio.....	48
Figure 32-enviar dados de anúncio para página de edit .....	50
Figure 33-submeter editar .....	52
Figure 34-Html adicionar anúncio .....	55
Figure 35-controlar adicionar anúncio.....	59
Figure 36-Chat .....	61
Figure 37-enviar chat.....	62
Figure 38- favoritos.....	62

## 1. INTRODUÇÃO

Bem-vindo ao manual de desenvolvimento para o projeto de gestão de alojamentos da Escola Superior de Tecnologia e Gestão de Oliveira do Hospital (ESTGOH). Este manual destina-se a fornecer orientações detalhadas sobre a implementação e manutenção do sistema, desenvolvido utilizando PHP Laravel, HTML, JavaScript, MySQL e CSS.

### 1.2 Objetivo do Projeto

O projeto visa criar uma solução abrangente e intuitiva para a gestão de alojamentos, direcionada especificamente aos alunos da ESTGOH. O sistema facilitará a busca, reserva e comunicação entre estudantes e senhorios, proporcionando uma plataforma centralizada e eficaz.

### 1.3 Tecnologias Utilizadas

O projeto utiliza uma combinação de linguagens e tecnologias web, incluindo:

- PHP Laravel: “Framework” PHP moderno e poderoso para o desenvolvimento de aplicativa web.
- HTML: Linguagem de marcação para a criação da estrutura da “interface” do utilizador.
- JavaScript: Linguagem de programação para tornar a “interface” do utilizador interativa e dinâmica.
- MySQL: sistema de gestão de banco de dados relacional para armazenamento e recuperação de informações críticas.
- CSS: Linguagem de estilo para definir a apresentação e o layout da “interface” do utilizador.

### 1.4 Funcionalidades Principais

Entre as funcionalidades principais do sistema estão:

- Busca de quartos disponíveis com critérios específicos (localização, preço, comodidades).
- Comunicação direta entre alunos e senhorios para negociações e esclarecimento de dúvidas.
- Painel de controle para senhorios gerirem as suas propriedades e responderem às consultas dos alunos.

### 1.5 Segurança e Privacidade

A segurança e a privacidade dos dados dos utilizadores são prioridades essenciais. O sistema foi desenvolvido conforme as regulamentações de proteção de dados para garantir a segurança das informações dos utilizadores.

## 2. INSTALL PROJECT

### 2.1 Fazer clone ao projeto de GitHub

Clonar o Projeto do GitHub: abra o terminal ou prompt de comando e navegue até o diretório onde deseja armazenar o projeto. Em seguida, execute o seguinte comando para clonar o repositório do GitHub:

→ **git clone [https://github.com/MuaiadHadad/Project\\_Gestor\\_Alojamento](https://github.com/MuaiadHadad/Project_Gestor_Alojamento)**

### 2.2 Instar Laravel

Instalar o Laravel: Antes de instalar o Laravel, certifique-se de que possui o Composer instalado. Se ainda não o tiver, pode baixá-lo e instalá-lo em <https://getcomposer.org/>. Após instalar o Composer, navegue até o diretório do projeto clonado e execute o seguinte comando para instalar as dependências do Laravel:

→ **composer install**

### 2.3 Configurar o Banco de Dados

Renomeie o arquivo .env.example para .env no diretório raiz do projeto.

Abra o arquivo .env e configure as informações do banco de dados, incluindo DB\_CONNECTION, DB\_HOST, DB\_PORT, DB\_DATABASE, DB\_USERNAME e DB\_PASSWORD.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=gestor_alojamento
DB_USERNAME=root
DB_PASSWORD=rootroot
```

Figure 1-Base Dados

Não esquece de instalar as tabelas que estão em anexo.

### 2.4 Executar o Servidor de Desenvolvimento

Após instalar as dependências do Laravel e configurar o banco de dados, pode iniciar o servidor de desenvolvimento executando o seguinte comando:

**php artisan serve**

## 3. DESENVOLVIMENTO

### 3.1 Página Inicial

#### 3.1.1 FRONT END

Aqui nesta parte vamos explicar como foi implementado o front-end de página inicial:

```

<div class="top-row">
    <div class="inner">
        <a class="main-logo small" href="https://www.estgoh.ipc.pt">
            
            <div style="..." class="title-sm-main">Escola Superior de <br>Tecnologia e Gestão<br>Politécnico de Coimbra</div>
        </a>
        @if($DadosUser!=null)
        <ul class="site-options">
            <li class="dropdown-menu-user-div-container">
                @foreach($DadosUser as $user)
                    
                    <i class="fa fa-inverse">{{$user->Email}}</i>
                @endforeach
                <a class="dropdown-menu-user"><i class="fa fa-angle-down fa-inverse"></i></a>
            </li>
        </ul>
        @else
        <ul class="site-options">
            <li><a class="js-layerSearchToggle"><i class="fa fa-search fa-2x fa-inverse"></i></a></li>
            <li class="dropdown-menu-user-div-container">
                <a class="dropdown-menu-user"><i class="fa fa-user fa-2x fa-inverse"></i></a>
                <a class="dropdown-menu-user"><i class="fa fa-angle-down fa-inverse"></i></a>
            </li>
        </ul>
        @endif
    </div>

```

Figure 2- cabeçalho da página com login e sem

Divisões e classes CSS: A primeira div (<div class="top-row">) serve como contêiner principal para a linha superior da barra de navegação. Dentro dela, há outra div (<div class="inner">) que pode ser usada para centralizar ou agrupar elementos.

Logo e Título: Dentro da <div class="inner">, há um link (<a>) que provavelmente representa o logotipo da página ou do site. Ele contém uma imagem (<img>) e um título (<div class="title-sm-main">) que descreve a instituição ou o propósito da página.

Condicional PHP (@if): O código PHP entre @if e @else verifica se há dados de utilizador disponíveis (\$DadosUser). Dependendo disso, diferentes elementos HTML são renderizados. Se houver dados de utilizador, será provavelmente exibido um menu suspenso com informações do utilizador e um ícone de seta para baixo (<i class="fa fa-angle-down fa-inverse"></i>). Caso contrário, serão exibidos outros elementos, como um ícone de busca (<i class="fa fa-search fa-2x fa-inverse"></i>) e um ícone de utilizador (<i class="fa fa-user fa-2x fa-inverse"></i>).

Iteração sobre os dados do utilizador: se houver dados de utilizador disponíveis, o código PHP utiliza um loop foreach para iterar sobre esses dados. que ele exibe o avatar do utilizador e o endereço de e-mail (\$user->Email).

Ícones FontAwesome: Os ícones utilizados (<i class="fa fa-..."></i>) são da biblioteca FontAwesome, que fornece uma ampla variedade de ícones vetoriais.

Geralmente, este trecho de código é responsável por exibir o cabeçalho da página, incluindo o logotipo, título, elementos de navegação e informações do utilizador, dependendo do contexto da autenticação do utilizador.

```
<script>
    document.addEventListener("DOMContentLoaded", function() {
        var userDropdown = document.querySelector('.dropdown-menu-user-div-container');
        var dropdownMenu = document.querySelector('.dropdown-menu-user-div');

        userDropdown.addEventListener('click', function(e) {
            e.stopPropagation();
            dropdownMenu.classList.toggle('show');
        });

        // Fechar o menu quando clicar fora dele
        document.addEventListener('click', function() {
            dropdownMenu.classList.remove('show');
        });

        // Evitar que o menu feche quando se clica dentro dele
        dropdownMenu.addEventListener('click', function(e) {
            e.stopPropagation();
        });
    });

</script>
```

Figure 3- Java Script de menu de utilizador

Evento 'DOMContentLoaded': esse evento é acionado quando todo o conteúdo do DOM foi carregado. Isso significa que o “script” será executado assim que a estrutura HTML estiver pronta para ser manipulada.

Seleção de Elementos: O código seleciona dois elementos do DOM:

userDropdown: Representa o contêiner do menu suspenso.

dropdownMenu: Representa o próprio menu suspenso.

Evento de Clique no contêiner do menu suspenso: quando o contêiner do menu suspenso é clicado, o evento de clique é acionado. Dentro desse evento, é chamado `e.stopPropagation()`, o que impede que o evento se propague para o elemento pai, e então é alternada a classe “show” no menu suspenso. Isso provavelmente mostra ou esconde o menu, dependendo do seu estado atual.

Evento de Clique fora do menu suspenso: esse evento é adicionado ao documento inteiro. Quando qualquer parte do documento é clicada, o menu suspenso é fechado. Isso é feito removendo a classe “show” do menu suspenso.

Evento de Clique no menu suspenso: esse evento é adicionado ao próprio menu suspenso. Quando o menu suspenso é clicado, o evento é acionado. Dentro dele, `e.stopPropagation()` é chamado para evitar que o evento se propague para o elemento pai. Isso garante que o menu não seja fechado quando se clica dentro dele.

Em resumo, esse “script” JavaScript adiciona funcionalidades de exibição e ocultação ao menu suspenso, permitindo que ele seja aberto ao clicar num determinado elemento e fechado ao clicar

fora dele, mas não fechado ao clicar no próprio menu suspenso. Isso proporciona uma melhor experiência de utilizador ao interagir com o menu.

```

<div class="dropdown-menu-user-div">
    @if( isset($DadosUser) && $DadosUser!=null)
        @foreach($DadosUser as $user)
            @if($user->Tipo=='senhorio')
                <ul>
                    <li><a href="/Senhorio/Profile"><i class="fa fa-indent"></i> Profile</a></li>
                    <li><a href="/Senhorio"><i class="fa fa-sign-out"></i>Área pessoal</a></li>
                    <li><a href="/logout"><i class="fa fa-sign-out"></i> Logout</a></li>
                </ul>
            @endif
            @if($user->Tipo=='gestor')
                <ul>
                    <li><a href="gestor/profile"><i class="fa fa-indent"></i> Profile</a></li>
                    <li><a href="/gestor"><i class="fa fa-sign-out"></i>Área pessoal</a></li>
                    <li><a href="/logout"><i class="fa fa-sign-out"></i> Logout</a></li>
                </ul>
            @endif
            @if($user->Tipo=='Aluno')
                <ul>
                    <li><a href="gestor/profile"><i class="fa fa-indent"></i> Profile</a></li>
                    <li><a href="/gestor"><i class="fa fa-sign-out"></i>Área pessoal</a></li>
                    <li><a href="/logout"><i class="fa fa-sign-out"></i> Logout</a></li>
                </ul>
            @endif
        @endforeach
    @else
        <ul>
            <li><a href="/login"><i class="fa fa-sign-in"></i> Login</a></li>
            <li><a href="/register"><i class="fa fa-user-plus"></i> Registar</a></li>
        </ul>
    @endif
</div>

```

**Figure 4- Lógica de mostra os utilizadores**

Divisão e classe CSS: A div com a classe dropdown-menu-user-div representa provavelmente o contiver do menu suspenso.

Condicional PHP (@if): Assim como no trecho anterior, este código PHP verifica se existem dados de utilizador disponíveis (\$DadosUser) e se não estão vazios. Se os dados estiverem presentes, o código itera sobre esses dados usando um “loop” foreach.

Condições IF no “loop”: no “loop” foreach, o código verifica o tipo de utilizador (\$user->Tipo) e exibe opções de menu específicas com base nesse tipo. Se o utilizador for do tipo "senhorio", por exemplo, serão exibidas opções relacionadas a senhorios, como perfil, área pessoal e logout. O mesmo ocorre para os tipos de utilizador "gestor" e "Aluno".

Ligações de navegação: Para cada tipo de utilizador, são exibidas ligações de navegação (<a href="...">) para diferentes partes do sítio “web”, como perfil, área pessoal e logout. Cada ligação é acompanhada por um ícone da biblioteca FontAwesome.

Condição ELSE: Se não houver dados de utilizador disponíveis, o código exibe opções de menu padrão para utilizadores não autenticados, como “login” e registo.

Geralmente, este trecho de código é responsável por exibir um menu suspenso dinâmico que se adapta ao tipo de utilizador autenticado, exibindo opções específicas para cada tipo de utilizador, ou opções padrões para utilizadores não autenticados.

Condição ELSE: Se não houver dados de utilizador disponíveis, o código exibe opções de menu padrão para utilizadores não autenticados, como “login” e registo.

Geralmente, este trecho de código é responsável por exibir um menu suspenso dinâmico que se adapta ao tipo de utilizador autenticado, exibindo opções específicas para cada tipo de utilizador, ou opções padrões para utilizadores não autenticados.

```

<form action="/inicio" method="POST">
    @csrf
    <div class="center_home_1 clearfix">
        <div class="center_home_1i1 clearfix">
            <div class="col-sm-4">
                <div class="center_home_1i11 clearfix">
                    <input name="preco" placeholder="Preço" class="form-control" type="text">
                </div>
            </div>
            <div class="col-sm-4">
                <div class="center_home_1i11 clearfix">
                    <input name="destance" placeholder="Distância por Metros" class="form-control" type="text">
                </div>
            </div>
            <div class="col-sm-4">
                <div class="center_home_1i11 clearfix">
                    <select name="Tipo" class="form-control" name="property">
                        <option value="Tipo">Tipo de Propriedade</option>
                        <option value="Casa">Casa</option>
                        <option value="Apartamento">Apartamento</option>
                        <option value="Estúdio">Estúdio</option>
                    </select>
                </div>
            </div>
        </div>
    </div>
<form action="/inicio" method="POST">
    @csrf
    <div class="center_home_1 clearfix">
        <div class="center_home_1i1 clearfix">
            <div class="col-sm-4">
                <div class="center_home_1i11 clearfix">
                    <input name="preco" placeholder="Preço" class="form-control" type="text">
                </div>
            </div>
            <div class="col-sm-4">
                <div class="center_home_1i11 clearfix">
                    <input name="destance" placeholder="Distância por Metros" class="form-control" type="text">
                </div>
            </div>
            <div class="col-sm-4">
                <div class="center_home_1i11 clearfix">
                    <select name="Tipo" class="form-control" name="property">
                        <option value="Tipo">Tipo de Propriedade</option>
                        <option value="Casa">Casa</option>
                        <option value="Apartamento">Apartamento</option>
                        <option value="Estúdio">Estúdio</option>
                    </select>
                </div>
            </div>
        </div>
    </div>
<div class="col-sm-4">
    <div class="center_home_1i11 clearfix">
        <select name="Sexo" class="form-control" name="beds" id="beds">
            <option value="Sexo">Gênero</option>
            <option value="Masculino">Masculino</option>
            <option value="Feminino">Feminino</option>
            <option value="M/F">feminino e masculino</option>
        </select>
    </div>
</div>
<div class="col-sm-4">
    <div class="center_home_1i11 clearfix">
        <button style="..." class="mgt text-center"><a class="button_1 block mgt"><i class="fa fa-filter"></i> Filtrar</a></button>
    </div>
</div>

```

Figure 5- filtragem

Seção e classes CSS: O formulário está envolto numa seção com o ID "center" e as classes "center\_list" e "clearfix". Essas classes são provavelmente usadas para estilização CSS e layout.

Container e linhas: Na seção, há um contêiner (`<div class="container">`) e uma linha (`<div class="row">`). Esses elementos podem ser usados para organizar o conteúdo numa grade, provavelmente usando um ‘framework’ CSS como Bootstrap.

Formulário: Na linha, há um formulário (`<form>`) com um atributo `action` definido para "/inicio" e um método `method` definido como "POST". Isso indica que os dados do formulário serão enviados para o URL "/inicio" usando o método POST.

Campos de entrada e seleção: O formulário contém vários campos de entrada e seleção:

Um campo de entrada para o preço (`<input name="preco" ...>`).

Um campo de entrada para a distância em metros (`<input name="destance" ...>`).

Um menu suspenso para selecionar o tipo de propriedade (`<select name="Tipo" ...>`).

Um menu suspenso para selecionar o número de quartos (`<select name="n_quaros" ...>`).

Um menu suspenso para selecionar o gênero (`<select name="Sexo" ...>`).

Botão de envio: Há um botão de envio no final do formulário, estilizado para ocupar 100% da largura do seu contêiner. Este botão será provavelmente usado para enviar o formulário após o utilizador ter selecionado os critérios de filtro.

Ícone do filtro: O botão de envio contém um ícone de filtro (um ícone FontAwesome representado pela classe "fa fa-filter").

Geralmente, este formulário permite que os utilizadores filtrem propriedades imobiliárias com base em critérios como preço, distância, tipo de propriedade, número de quartos e gênero. Ao submeter o formulário, os dados serão enviados para "/inicio" para processamento adicional.

```
<div class="popular_1 text-center clearfix">
    <div class="col-sm-12">
        <h1 class="mgt"> Lista de Propriedades</h1>
        <p>Encontre quarto com melhor classificação para você.</p>
    </div>
</div>

@if(isset($DataQuartoAtive) && $DataQuartoAtive !=null)
<div class="popular_2 clearfix">
    <h2 class="mgt">Renda por Quartos</h2>
    <hr>
    @foreach($DataQuartoAtive as $QuartoAtive)
        <div class="col-sm-3">
            <div class="popular_21 clearfix">
                <div class="popular_211 clearfix">
                    <a href="/inicio/{{$QuartoAtive->idnow}}/quarto">
                        <h5 class="mgt"><a href="/inicio/{{$QuartoAtive->idnow}}/quarto">{{ $QuartoAtive->Tipo }}</a></h5>
                    </div>
                </div>
                <div class="popular_213 clearfix">
                    <h5 class="mgt"><a href="/inicio/{{$QuartoAtive->idnow}}/quarto">{{ $QuartoAtive->Titulo }}</a></h5>
                    <h4 class="col_1">{{ $QuartoAtive->Preço }}€ <span class="col_2">Month</span></h4>
                    <h6>@if($QuartoAtive->Genero =='Masculino')<i class="fa fa-male col_2"></i>@elseif($QuartoAtive->Genero =='Feminino')<i class="fa fa-female col_2"></i>
                    @endif {{ $QuartoAtive->Genero }} <span><i class="fa fa-university col_2"></i>{{ $QuartoAtive->area }} m</span></h6>
                    <h5 class="inline"><a class="button_1" href="/inicio/{{$QuartoAtive->idnow}}/quarto"><i class="fa fa-info-circle"></i> Detalhes</a></h5>
                    <h5 class="inline"><a class="button_1" href="detail.html"><i class="fa fa-heart" style="color: #ccc;"></i></a></h5>
                </div>
            </div>
        @endforeach
    </div>
</div>
```

Figure 6- As ofertas

Seção e classes CSS: A seção tem o “ID” “popular” e as classes “container” e “row”, que provavelmente são usadas para estilização CSS e layout.

Título e descrição: Na seção, há um elemento h1 com o título “Lista de Propriedades” e um parágrafo (

) com uma descrição “Encontre quarto com melhor classificação para você.”

Condisional PHP (@if): Há duas estruturas condicionais @if que verificam se há dados de propriedades ativas disponíveis. Se houver, esses dados são iterados usando um loop foreach. Para cada propriedade ativa, são exibidas informações como tipo, título, preço, gênero, área e botões para detalhes e favoritos.

Botões de paginação: No final da seção, há um bloco de código PHP que gera uma lista de botões de paginação usando um loop for. Cada botão de página é um link que aponta para uma página específica.

Geralmente, esta seção exibe uma lista de propriedades imobiliárias, incluindo quartos e casas, com informações como tipo, título, preço, gênero, área e botões para detalhes e favoritos. Também incluem botões de paginação para navegar entre as diferentes páginas de resultados.

### 3.1.2 BACK END

```

public function GetPageInicial($pagina = 1){
    $resultadosPorPagina = 8;
    $quantoporQuarto=4;
    $quantoporCasa=4;
    $indiceInicial = ($pagina - 1) * $resultadosPorPagina;
    if(session( key: 'ActivationToken')!=null){
        $token=session( key: 'ActivationToken');
        $user = DB::table( table: 'user')->where( column: 'ActivationToken', $token)->get();
        $totalQuartos = DB::table( table: 'quarto')->where( column: 'estado', operator: 'Ativo')->count();
        $totalCasas = DB::table( table: 'casa_completa')->where( column: 'estado', operators: 'Ativo')->count();
        if($totalCasas<4){
            $quantoporQuarto=8-$totalCasas;
        }
        if($totalQuartos<4){
            $quantoporCasa=8-$totalQuartos;
        }
        $QuartosAtive =DB::table( table: 'quarto')
            ->join( table: 'banho', first: 'banho.id_quarto', operator: '=', second: 'quarto.id')
            ->join( table: 'contato', first: 'contato.id_quarto', operator: '=', second: 'quarto.id')
            ->join( table: 'cozinha', first: 'cozinha.id_quarto', operator: '=', second: 'quarto.id')
            ->join( table: 'endereco', first: 'endereco.id_quarto', operator: '=', second: 'quarto.id')
            ->join( table: 'quartos_de_casa', first: 'quartos_de_casa.id_quarto', operator: '=', second: 'quarto.id')
            ->join( table: 'sala', first: 'sala.id_quarto', operator: '=', second: 'quarto.id')
            ->join( table: 'servicos', first: 'servicos.id_quarto', operator: '=', second: 'quarto.id')
            ->join( table: 'outros', first: 'outros.id_quarto', operator: '=', second: 'quarto.id')
            ->leftJoin( table: 'midia_de_casa', function ($join) {
                $join->on('midia_de_casa.id_quarto', '=', 'quarto.id')
                ->whereRaw('midia_de_casa.id = (SELECT id FROM midia_de_casa WHERE id_quarto = quarto.id ORDER BY id ASC LIMIT 1)');
            })
            ->where( column: 'quarto.estado', operator: 'Ativo')
            ->select( columns: 'midia_de_casa.*','quarto.id as idnow','quarto.*', 'banho.*', 'contato.*', 'cozinha.*', 'endereco.*'
            , 'quartos_de_casa.*'
            , 'sala.*', 'servicos.*', 'outros.*')
            ->skip($indiceInicial)
    }
    ->take($quantoporQuarto)
    ->get();
}

$CasaAtive =DB::table( table: 'casa_completa')
    ->join( table: 'banho', first: 'banho.id_casa', operator: '=', second: 'casa_completa.id')
    ->join( table: 'contato', first: 'contato.id_casa', operator: '=', second: 'casa_completa.id')
    ->join( table: 'cozinha', first: 'cozinha.id_casa', operator: '=', second: 'casa_completa.id')
    ->join( table: 'endereco', first: 'endereco.id_casa', operator: '=', second: 'casa_completa.id')
    ->join( table: 'sala', first: 'sala.id_casa', operator: '=', second: 'casa_completa.id')
    ->join( table: 'servicos', first: 'servicos.id_casa', operator: '=', second: 'casa_completa.id')
    ->join( table: 'outros', first: 'outros.id_casa', operator: '=', second: 'casa_completa.id')
    ->leftJoin( table: 'midia_de_casa', function ($join) {
        $join->on('midia_de_casa.id_casa', '=', 'casa_completa.id')
        ->whereRaw('midia_de_casa.id = (SELECT id FROM midia_de_casa WHERE id_casa = casa_completa.id ORDER BY id ASC LIMIT 1)');
    })
    ->where( column: 'casa_completa.estado', operator: 'Ativo')
    ->select( columns: 'midia_de_casa.*','casa_completa.id as idnow','casa_completa.*', 'banho.*', 'contato.*', 'cozinha.*', 'endereco.*'
    , 'sala.*', 'servicos.*', 'outros.*')
    ->skip($indiceInicial)
    ->take($quantoporCasa)
    ->get();

$totalResultados = $totalQuartos + $totalCasas;
$totalPginas = ceil( num: $totalResultados / $resultadosPorPagina);
return view( view: 'inicio_index', ['DataCasaAtive'=>$CasaAtive, 'DataQuartoAtive'=>$QuartosAtive, 'DadosUser'=>$user
, 'totalPginas'=>$totalPginas,
'pagina' => $pagina]);

```

Figure 7- recuperar os dados das propriedades imobiliárias para exibição na página inicial

Este código PHP é responsável por recuperar os dados das propriedades imobiliárias para exibição na página inicial do site:

Métodos Utilizados:

**GetPageInicial(\$pagina):** Este método é responsável por recuperar os dados das propriedades imobiliárias para exibição na página inicial do site web. Ele recebe o número da página como parâmetro e executa consultas ao banco de dados para obter os dados dos quartos ativos e das

casas completas. Além disso, ele implementa a lógica de paginação para exibir um número específico de resultados por página.

#### Técnicas Utilizadas:

**Consulta SQL Complexa:** O código utiliza consultas SQL complexas que envolvem várias junções e subconsultas para recuperar dados das tabelas relacionadas. Isso permite recuperar informações detalhadas sobre os quartos e casas, incluindo características como banheiros, cozinhas, serviços, etc.

**Paginação:** A técnica de paginação é implementada para dividir os resultados em várias páginas, facilitando a navegação do utilizador. A lógica de paginação calcula o número total de páginas com base no número de resultados e no número desejado de resultados por página.

#### Lógicas Utilizadas:

**Lógica de Recuperação de Dados:** O método GetPageInicial utiliza uma lógica para recuperar apenas as propriedades imobiliárias ativas do banco de dados. Isso é feito por condições de filtragem nas consultas SQL, garantindo que apenas os quartos e casas ativos sejam exibidos na página inicial.

**Lógica de Paginação:** A lógica de paginação calcula o índice inicial para a consulta com base no número da página e no número de resultados por página. Além disso, ela determina o número total de páginas necessárias para exibir todos os resultados disponíveis.

**Verificação de Sessão e Controle de Acesso:** O código inclui uma lógica para verificar se há uma sessão ativa do utilizador. Se houver uma sessão ativa, determinados parâmetros de exibição podem ser ajustados com base nas informações do utilizador logado.

#### Considerações Finais:

Esses métodos, técnicas e lógicas são fundamentais para garantir que a página inicial do sítio web apresente de forma eficiente e organizada as propriedades imobiliárias disponíveis. A utilização de consultas SQL complexas, lógica de paginação e verificação de sessão contribuem para uma experiência do utilizador mais fluida e personalizada.

```

public function index($pagina = 1){
    $resultadosPorPagina = 8;
    $indiceInicial = ($pagina - 1) * 4;
    if(session('key') == 'ActivationToken') {
        $token = session('key');
        $user = DB::table('user')->where('ActivationToken', $token)->get();
        $QuartosAtive = DB::table('quarto')
            ->join('banho', 'banho.id_quarto', '=', 'quarto.id')
            ->join('contato', 'contato.id_quarto', '=', 'quarto.id')
            ->join('cozinha', 'cozinha.id_quarto', '=', 'quarto.id')
            ->join('endereco', 'endereco.id_quarto', '=', 'quarto.id')
            ->join('quartos_de_casa', 'quartos_de_casa.id_quarto', '=', 'quarto.id')
            ->join('sala', 'sala.id_quarto', '=', 'quarto.id')
            ->join('servicos', 'servicos.id_quarto', '=', 'quarto.id')
            ->join('outros', 'outros.id_quarto', '=', 'quarto.id')
            ->leftJoin('midia_de_casa', function ($join) {
                $join->on('midia_de_casa.id_quarto', '=', 'quarto.id')
                    ->whereRaw('midia_de_casa.id = (SELECT id FROM midia_de_casa WHERE id_quarto = quarto.id ORDER BY id ASC LIMIT 1)');
            })
            ->where('quarto.estado', 'Ativo')
            ->select(['midia_de_casa.*', 'quarto.id as idnow', 'quarto.*', 'banho.*', 'contato.*', 'cozinha.*', 'endereco.*',
                'quartos_de_casa.*',
                'sala.*', 'servicos.*', 'outros.*'])->get();
        $CasaAtive = DB::table('casa_completa')
            ->join('banho', 'banho.id_casa', '=', 'casa_completa.id')
            ->join('contato', 'contato.id_casa', '=', 'casa_completa.id')
            ->join('cozinha', 'cozinha.id_casa', '=', 'casa_completa.id')
            ->join('endereco', 'endereco.id_casa', '=', 'casa_completa.id')
            ->join('sala', 'sala.id_casa', '=', 'casa_completa.id')
            ->join('servicos', 'servicos.id_casa', '=', 'casa_completa.id')
            ->join('outros', 'outros.id_casa', '=', 'casa_completa.id')
            ->leftJoin('midia_de_casa', function ($join) {
                $join->on('midia_de_casa.id_casa', '=', 'casa_completa.id')
                    ->whereRaw('midia_de_casa.id = (SELECT id FROM midia_de_casa WHERE id_casa = casa_completa.id ORDER BY id ASC LIMIT 1)');
            })
            ->where('casa_completa.estado', 'Ativo')
            ->select(['midia_de_casa.*', 'casa_completa.id as idnow', 'casa_completa.*', 'banho.*', 'contato.*', 'cozinha.*', 'endereco.*',
                'sala.*', 'servicos.*', 'outros.*'])->get();
        $totalQuartos = $QuartosAtive->count();
        $totalCasas = $CasaAtive->count();
        $totalResultados = $totalQuartos + $totalCasas;
        $totalPaginas = ceil($totalResultados / $resultadosPorPagina);
        $quantoporQuarto = 4;
        $quantoporCasa = 4;
        if ($totalCasas < 4) {
            $quantoporQuarto = 8 - $totalCasas;
        }
        if ($totalQuartos < 4) {
            $quantoporCasa = 8 - $totalQuartos;
        }
        $QuartosPaginados = $QuartosAtive->skip(max(0, $indiceInicial))->take($quantoporQuarto);
        $CasasPaginadas = $CasaAtive->skip(max(0, $indiceInicial))->take($quantoporCasa);

        return view('inicioIndex', [
            'DataCasaAtive' => $CasasPaginadas,
            'DataQuartoAtive' => $QuartosPaginados,
            'totalPaginas' => $totalPaginas,
            'DadosUser' => $user,
            'pagina' => $pagina
        ]);
    }
}

```

Figure 8- exibir a página inicial do site

#### Métodos Utilizados:

`index($pagina)`: Este método é responsável por exibir a página inicial do site, que inclui uma lista de propriedades imobiliárias disponíveis para locação. Ele recebe o número da página como parâmetro para implementar a paginação dos resultados.

#### Técnicas Utilizadas:

**Consulta SQL Complexa:** O código utiliza consultas SQL complexas que envolvem várias junções e subconsultas para recuperar dados detalhados sobre as propriedades imobiliárias. Isso inclui características como banheiros, cozinhas, salas, serviços, etc.

Paginação: A técnica de paginação é implementada para dividir os resultados em várias páginas, facilitando a navegação do utilizador. A lógica de paginação calcula o número total de páginas com base no número de resultados e no número desejado de resultados por página.

Lógicas Utilizadas:

Lógica de Recuperação de Dados: O método index utiliza consultas ao banco de dados para recuperar dados sobre os quartos e casas disponíveis para locação. Ele verifica se há uma sessão ativa do utilizador para personalizar a exibição com base nas informações do utilizador logado.

Lógica de Paginação: A lógica de paginação calcula o índice inicial para a consulta com base no número da página e no número de resultados por página. Além disso, ela determina o número total de páginas necessárias para exibir todos os resultados disponíveis.

Ajuste de Quantidade de Resultados por Página: O código inclui uma lógica para ajustar a quantidade de resultados por página com base no número total de quartos e casas disponíveis. Isso garante que o número ideal de resultados seja exibido em cada página, considerando o número total de resultados disponíveis.

Considerações Finais:

Esses métodos, técnicas e lógicas são cruciais para garantir que a página inicial do sítio web apresente de forma eficiente e organizada as propriedades imobiliárias disponíveis para locação. A utilização de consultas SQL complexas, lógica de paginação e verificação de sessão contribuem para uma experiência do utilizador mais fluida e personalizada.

```

public function Filtrar(Request $request){
    if ($request->has('ActivationToken') != null) {
        $token = session('ActivationToken');
        $user = DB::table('user')->where('column', 'ActivationToken', $token)->get();
        $query = DB::table('casa_completa')
            ->join('table: endreco', 'first: endreco.id_casa', 'operator: '=', 'second: casa_completa.id')
            ->leftJoin('table: midia_de_casa', 'function ($join) {
                $join->on("midia_de_casa.id_casa", '=', 'casa_completa.id')
                | ->whereRaw("midia_de_casa.id = (SELECT id FROM midia_de_casa WHERE id_casa = casa_completa.id ORDER BY id ASC LIMIT 1)");
            });
        $queryQuarto = DB::table('quarto')
            ->join('table: endreco', 'first: endreco.id_quarto', 'operator: '=', 'second: quarto.id')
            ->leftJoin('table: midia_de_casa', 'function ($join) {
                $join->on("midia_de_casa.id_quarto", '=', 'quarto.id')
                | ->whereRaw("midia_de_casa.id = (SELECT id FROM midia_de_casa WHERE id_quarto = quarto.id ORDER BY id ASC LIMIT 1)");
            });
        if ($request->has('preco') && $request->input('preco')!=null) {
            $preco = $request->input('preco');
            $query->where('column: casa_completa.Preco', 'operator: '<=', $preco);
            $queryQuarto->where('column: quarto.Preco', 'operator: '<=', $preco);
        }

        if ($request->has('destance') && $request->input('destance')!=null) {
            $distancia = $request->input('destance');
            $distanciaKm = $distancia / 1000;
            $query->where(DB::raw('value: "REPLACE(SUBSTRING_INDEX(endreco.Distancia, " ", 1), ",", ".")'), 'operator: '<=', $distanciaKm);
            $queryQuarto->where(DB::raw('value: "REPLACE(SUBSTRING_INDEX(endreco.Distancia, " ", 1), ",", ".")'), 'operator: '<=', $distanciaKm);
        }
    }

    if ($request->has('Tipo') && $request->input('Tipo')!="Tipo") {
        $tipo = $request->input('Tipo');
        $query->where('column: casa_completa.Tipo', 'operator: '=', $tipo);
        $queryQuarto->where('column: quarto.Tipo', 'operator: '=', $tipo);
    }

    if ($request->has('n_quartos') && $request->input('n_quartos')!="n_quartos") {
        $n_quartos = $request->input('n_quartos');
        $query->where('column: casa_completa.N_quartos', 'operator: '<=', $n_quartos);
        $queryQuarto->where('column: quarto.N_quartos', 'operator: '<=', $n_quartos);
    }

    if ($request->has('Sexo') && $request->input('Sexo')!="Sexo") {
        $sexo = $request->input('Sexo');
        $query->where('column: casa_completa.Genero', 'operator: '=', $sexo);
        $queryQuarto->where('column: quarto.Genero', 'operator: '=', $sexo);
    }

    $query->where('column: casa_completa.estado', 'operator: '=', 'value: Ativo');
    $queryQuarto->where('column: quarto.estado', 'operator: '=', 'value: Ativo');

    $resultadosQuarto = $queryQuarto->select('columns: midia_de_casa.*','quarto.*','quarto.id as idnow')->get();
    $resultadosCasa = $query->select('columns: midia_de_casa.*','casa_completa.*','casa_completa.id as idnow')->get();

    return view('view: inicio\index',[ 'DadosUser' => $user,'resultadosCasa' => $resultadosCasa,'resultadosQuarto'=>$resultadosQuarto]);
}

```

**Figure 9- Filtragem**

Esta função filtrar é responsável por filtrar propriedades imobiliárias e quartos disponíveis com base nos critérios especificados pelo utilizador e exibir os resultados na página inicial do sitio web. Aqui está uma explicação passo a passo do que o código faz:

Verificação do “Token” de Ativação de Sessão:

Primeiro, verifica se há um token de ativação de sessão armazenado. Isso provavelmente é usado para identificar o utilizador logado.

Consulta ao Banco de Dados:

A função realiza consultas ao banco de dados para recuperar os detalhes das propriedades e quartos disponíveis.

As consultas são elaboradas para selecionar informações relevantes das tabelas casa\_completa (para propriedades) e quarto (para quartos).

As consultas utilizam junções internas (join) e externas (leftJoin) para garantir que todas as informações necessárias sejam obtidas.

Filtragem com base nos Parâmetros da Solicitação:

Os filtros são aplicados às consultas com base nos parâmetros fornecidos na solicitação (Request).

Os parâmetros considerados incluem preço máximo (preco), distância máxima (destance), tipo de propriedade (Tipo), número de quartos (n\_quartos) e sexo (Sexo).

Se os parâmetros estiverem presentes na solicitação e não forem nulos, eles são aplicados às consultas para filtrar os resultados.

Condições de Estado:

As consultas também incluem condições para garantir que apenas propriedades e quartos com estado "Ativo" sejam retornados.

Recuperação de Resultados:

Os resultados das consultas são armazenados em variáveis, como \$resultadosCasa para propriedades e \$resultadosQuarto para quartos.

Retorno da Visualização:

Por fim, os resultados são passados para uma visualização chamada 'inicio\index' para serem exibidos na página inicial do sítio web.

Além dos resultados, também são passadas informações sobre o utilizador logado, obtidas anteriormente com base no “token” de ativação da sessão.

Essa função desempenha um papel fundamental na exibição dinâmica de propriedades e quartos com base nos critérios de pesquisa do utilizador, melhorando a experiência do utilizador ao procurar por acomodações.

## 3.2 Detalhe

### 3.2.1 FRONT END

Este trecho de código HTML é responsável por exibir detalhes de uma propriedade imobiliária na página de detalhes do imóvel.

Em resumo, esse trecho de código cria uma página detalhada para cada propriedade imobiliária, exibindo todas as informações relevantes e facilitando a visualização e o contato para possíveis interessados.

```

<section id="center" class="clearfix center_detail" style="...>
    <div class="center clearfix">
        <div id="carousel" class="carousel slide carousel-fade" style="...>
            <div class="carousel-inner" style="...>
                @if($PhotoCasa !=null)
                    @php $slideNo = 0 @endphp
                    @foreach($PhotoCasa as $photoquarto)
                        @if($slideNo == 0)
                            <div data-slide-no="{{ $slideNo }}" class="item carousel-item active">
                                
                            </div>
                        @else
                            <div data-slide-no="{{ $slideNo }}" class="item carousel-item">
                                
                            </div>
                        @endif
                        @php $slideNo++ @endphp
                    @endforeach
                @endif
            </div>
            
            <a class="left carousel-control kb_control_left" href="#carousel" role="button" data-slide="prev">
                <span class="fa fa-angle-left kb_icons" aria-hidden="true"></span>
                <span class="sr-only">Previous</span>
            </a>
            
            <a class="right carousel-control kb_control_right" href="#carousel" role="button" data-slide="next">
                <span class="fa fa-angle-right kb_icons" aria-hidden="true"></span>
                <span class="sr-only">Next</span>
            </a>
        </div>
    </div>
</section>

```

**Figure 10- fotos de propriedade**

O código começa com um elemento `<div>` que contém um carousel de imagens, permitindo que os utilizadores vejam várias fotos da propriedade. Isso é implementado usando a biblioteca Bootstrap Carousel.

As imagens são obtidas dinamicamente a partir do objeto `$PhotoCasa`, e para cada imagem, é criado um elemento `<div>` com a classe `item carousel-item`. A primeira imagem recebe a classe adicional `active` para ser exibida inicialmente.

```

<div class="row">
    @if($dadosCasaAtive !=null)
        @foreach($DadosCasaAtive as $dadoscasa)
            <div class="list_detail_1 clearfix" >
                <div class="col-sm-16">
                    <div class="list_detail_11 clearfix" >
                        <h3 class="mgt">{{ $dadoscasa->Preco}}€ / <span class="span_1">Por mês</span> <span class="span_2">Aluguel</span></h3>
                        <p><i class="fa fa-map-marker"></i> {{ $dadoscasa->Endereco}}, {{ $dadoscasa->Codigo_postal}}</p>
                    </div>
                    <div class="list_detail_112 clearfix" >
                        <h4 class="mgt">Informações da propriedade</h4>
                        <hr>
                        <div class="list_detail_112i clearfix" >
                            <div class="col-sm-3 space_left" >
                                <h5 class="mgt">Distância</h5>
                                <h6 class="col_1">{{ $dadoscasa->Distancia}}</h6>
                            </div>
                            <div class="col-sm-3 space_left" >
                                <h5 class="mgt">Sexo</h5>
                                <h6 class="col_1">{{ $dadoscasa->Genero}}</h6>
                            </div>
                            <div class="col-sm-3 space_left" >
                                <h5 class="mgt">Área total do quarto</h5>
                                <h6 class="col_1">{{ $dadoscasa->area}}</h6>
                            </div>
                            <div class="col-sm-3 space_left" >
                                <h5 class="mgt">Número de Quartos</h5>
                                <h6 class="col_1">{{ $dadoscasa->N_quartos}}</h6>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
<div class="home_inner_i clearfix" >
    <div class="col-sm-3 space_left" >
        <h5 class="mgt"><i class="icon roupa-cama col_1">
            @if($dadosquartocasa->roupa_de_cama==0)
                <i class="icon no"></i>
            @endif
            </i> roupa de cama </h5>
    </div>
    <div class="col-sm-3 space_left" >
        <h5 class="mgt"><i class="icon cama col_1">
            @if($dadosquartocasa->cama==0)
                <i class="icon no"></i>
            @endif
            </i> cama</h5>
    </div>
    <div class="col-sm-3 space_left" >
        <h5 class="mgt"><i class="icon mesa-cama col_1">
            @if($dadosquartocasa->mesa_cabeceira==0)
                <i class="icon no"></i>
            @endif
            </i> mesa cabeceira </h5>
    </div>
    <div class="col-sm-3 space_left" >
        <h5 class="mgt"><i class="icon lambada col_1">
            @if($dadosquartocasa->Candeeiro_de_mesa_do_estudo==0)
                <i class="icon no"></i>
            @endif
            </i> Candeeiro</h5>
    </div>
</div>

```

Figure 11-específicos da propriedade

São exibidos os detalhes específicos da propriedade.

O loop @foreach percorre cada propriedade em \$DadosCasaAtive e exibe informações como preço, endereço e outras características, como distância, sexo, área total, número de quartos, etc.

Se houver informações de quartos específicos (\$DadosQuartoCasa), elas são exibidas em um loop adicional.

Descrição, Cozinha, Sala e Banheiro:

A descrição da propriedade é exibida num parágrafo.

As características da cozinha, sala e casa de banho são exibidas em listas, mostrando quais recursos estão disponíveis na propriedade.

```
<div class="list_detail_1l2 clearfix">
    <h4 class="mgt">Local</h4>
    <hr>
    <script>
        var letValue = "{{ $dadoscasa->let }}";
        letValue = letValue.replace(/\([()]/g, '');
        var coordinates = letValue.split(',');
        var lat = parseFloat(coordinates[0]);
        var lng = parseFloat(coordinates[1]);
        function initMap() {
            const myLatLng = { lat: 40.35827117617252, lng: -7.8569972177657075 };
            const map = new google.maps.Map(document.getElementById("map"), {
                zoom: 15,
                center: myLatLng,
            });
            const icon_escola = "https://img.icons8.com/emoji/55/school-emoji.png";
            var estgoh = { lat: 40.361008858094266, lng: -7.861192556524104 };
            new google.maps.Marker({
                position: estgoh,
                icon:icon_escola,
                map: map,
                title: "Hello World!",
            });
            const icon_casa = "https://img.icons8.com/plasticine/75/order-delivered.png";
            var casa = { lat:lat,lng:lng };
            new google.maps.Marker({
                position: casa,
                icon:icon_casa,
                map: map,
                title: "Hello World!",
            });
        }
    </script>
    <div id="map" style="..."></div>
```

Figure 12-Map

Um mapa do Google é exibido, mostrando a localização da propriedade e de pontos de interesse próximos, como escolas.

As coordenadas da propriedade são obtidas do objeto \$dadoscasa e usadas para marcar a sua localização no mapa.

### 3.2.2 BACK-END

```

if ($session['key']['ActivationToken'] != null) {
    $token = session('key['ActivationToken'];
    $user = DB::table('table: user')->where(column: 'ActivationToken', $token)->get();
    $Casativate = DB::table('table: casa_completa')
        ->join(table: 'banho', first: 'banho.id_casa', operator: '=', second: 'casa_completa.id')
        ->join(table: 'contato', first: 'contato.id_casa', operator: '=', second: 'casa_completa.id')
        ->join(table: 'cozinha', first: 'cozinha.id_casa', operator: '=', second: 'casa_completa.id')
        ->join(table: 'endereco', first: 'endereco.id_casa', operator: '=', second: 'casa_completa.id')
        ->join(table: 'sala', first: 'sala.id_casa', operator: '=', second: 'casa_completa.id')
        ->join(table: 'servicos', first: 'servicos.id_casa', operator: '=', second: 'casa_completa.id')
        ->join(table: 'outros', first: 'outros.id_casa', operator: '=', second: 'casa_completa.id')
        ->join(table: 'user', first: 'user.id', operator: '=', second: 'casa_completa.id_user')
        ->where(column: 'casa_completa.id', $id)
        ->select(columns: 'contato.Email as EmailQuarto','user.*','cozinha.Micro-ondas as Micro','casa_completa.id as idnow','casa_completa.*', 'banho.*', 'contato.*',
            , 'sala.*', 'servicos.*', 'outros.*','servicos.Wi-Fi as wifi')->get();
    if(!$Casativate){
        abort(code: 404, message: 'Bad Request.');
    }
    $PhotoCasa =DB::table('table: casa_completa')
        ->join(table: 'midia_de_casa', first: 'midia_de_casa.id_casa', operator: '=', second: 'casa_completa.id')
        ->where(column: 'casa_completa.id', $id)
        ->select(columns: 'midia_de_casa.*')->get();
    $QuartosCasa =DB::table('table: casa_completa')
        ->join(table: 'quartos_de_casa', first: 'quartos_de_casa.id_casa', operator: '=', second: 'casa_completa.id')
        ->where(column: 'casa_completa.id', $id)
        ->select(columns: 'quartos_de_casa.*')->get();
    return view('view: inicio\detalhe_casa', ['DadosUser' => $user,'DadosCasaAtiva'=>$Casativate,'PhotoCasa'=>$PhotoCasa,'DadosQuartoCasa'=>$QuartosCasa]);
}

```

Figure 13-recuperar os detalhes de uma casa específica

Essa função GetPageDetalheCasa é responsável por recuperar os detalhes de uma casa específica com base no seu ID e exibir esses detalhes na página de detalhes da casa. Aqui está uma explicação do que o código faz:

Verificação do Token de Ativação de Sessão:

Verifica se há um token de ativação de sessão armazenado para identificar o utilizador logado.

Consulta ao Banco de Dados:

Realiza várias junções (join) entre diferentes tabelas para recuperar todos os detalhes relevantes da casa, como informações de contato, detalhes da cozinha, da casa de banho, da sala, serviços oferecidos, etc.

Esses detalhes são recuperados com base no ID da casa fornecido como parâmetro para a função.

Verificação da Existência da Casa:

Verifica se a casa com o ID fornecido existe no banco de dados. Se não existir, retorna um erro 404.

Recuperação de Fotos da Casa:

Recupera as fotos da casa com base no seu ID.

Recuperação dos Quartos da Casa:

Recupera os quartos disponíveis na casa com base no seu ID.

Retorno da Visualização:

Retorna uma visualização que exibe os detalhes da casa, as fotos e os quartos disponíveis.

Se um utilizador estiver logado, os detalhes do utilizador são passados para a visualização. Caso contrário, é passado null para indicar que nenhum utilizador está logado.

Essa função é crucial para exibir informações detalhadas sobre uma casa específica na página do sítio web, fornecendo aos utilizadores uma visão abrangente dos serviços e comodidades oferecidos pela propriedade.

## 3.3 Login

### 3.3.1 FRONT END

```
<div class="container-main">
  <div class="container-login">
    <div class="form-box-login ">
      
      <form action="/login" method="POST">
        @csrf
        <div class="input-group-login ">
          <label for="Email">Email</label>
          <input type="text" id="Email" name="Email" required>
        </div>
        <div class="input-group-login ">
          <label for="password">Password</label>
          <input type="password" id="password" name="password" required>
          <a href="/Forgot"><i class="fa fa-send"></i> Forget Password</a>
        </div>
        <button type="submit">Login</button>
      </form>
    </div>
  </div>
</div>
```

Figure 14-login form

Esse trecho de código HTML representa um formulário de “login”. Ele permite que os utilizadores insiram o seu endereço eletrónico e senha para fazer “login”. Após preencher os campos, os utilizadores podem clicar no botão “Login” para enviar os dados do formulário para uma rota especificada, geralmente no backend do aplicativo “web”, onde o servidor processará as informações e autenticará o utilizador. A ligação “Forgot Password” permite que os utilizadores solicitem uma redefinição de senha caso tenham esquecido a senha.

```
<div style="...>
  @if(session('Error')!=null)
    <div class="toast">
      <div class="toast-content" style="...>
        <i class="fa fa-solid fa-remove error"></i>
        <div class="message">
          <span class="text text-1">Error</span>
          <span class="text text-2">{{session('Error')}}</span>
        </div>
      </div>
      <i class="fa fa-solid fa-close close"></i>
      <div class="progress error active"></div>
    </div>
  @endif
</div>
```

Figure 15-toast

Este código HTML exibe uma caixa de mensagem de erro de forma dinâmica na interface do utilizador. Ele verifica se há uma variável de sessão chamada 'Error' definida. Se essa variável estiver definida e não for nula, uma caixa de mensagem de erro será exibida, contendo um ícone de erro, uma mensagem de erro e uma barra de progresso para indicar que a mensagem está a ser exibida. O utilizador pode fechar a caixa de mensagem de erro clicando num ícone de error.

### 3.3.2 BACK-END

```
└─ MuaiadHadad
    public function login(Request $request)
    {
        $Email = $request->input('key: Email');
        $password = $request->input('key: password');

        $user = DB::table('user')->where('column: Email', '$Email')->first();

        if (!$user || !password_verify($password, $user->Password)) {
            return back()->with(['Error' => 'O e-mail ou palavra passa está incorreto!']);
        }

        if ($user->Estado == "Ativo") {
            $activationToken = Str::random('length: 60');
            DB::table('user')->where('column: Email', '$user->Email')->update([
                'ActivationToken' => $activationToken
            ]);

            session(['ActivationToken' => $activationToken]);
            session(['tipo_usuario' => $user->Tipo]);
            session(['Email' => $user->Email]);

            if ($user->Tipo == 'aluno') {
                return redirect('to: /aluno');
            }

            if ($user->Tipo == 'senhorio') {
                return redirect('to: /Senhorio');
            }

            if ($user->Tipo == 'gestor') {
                return redirect('to: /gestor');
            }
        } elseif ($user->Estado == "Desativo") {
            return back()->withErrors(['message' => 'Conta não está ativa']);
        } elseif ($user->Estado != "Desativo" && $user->Estado != "Ativo") {
            return redirect('to: /validation')->with('Email', '$Email');
        }
    }
}
```

Figure 16-controller de login

Este trecho de código PHP é responsável por lidar com o processo de login de utilizadores no sistema. Aqui está uma explicação passo a passo do que o código faz:

Obtenção dos Dados de Login:

O método login recebe uma solicitação (Request) que contém os campos de entrada Email e password, representando o e-mail e a senha inseridos pelo utilizador no formulário de login.

Verificação do Utilizador:

É feita uma consulta ao banco de dados para buscar um utilizador com o e-mail fornecido. Se não for encontrado nenhum utilizador correspondente, ou se a senha fornecida não coincidir com a senha armazenada no banco de dados (após ser criptografada com password\_hash), uma mensagem de erro é retornada, informando que o e-mail ou a senha estão incorretos.

Verificação do Estado do Utilizador:

Se o utilizador for encontrado no banco de dados e estiver com o estado "Ativo", um token de ativação é gerado aleatoriamente usando Str::random(60) e é atualizado no registro do utilizador no banco de dados. Em seguida, esse token é armazenado na sessão do utilizador.

Além disso, algumas informações do utilizador, como tipo de utilizador e e-mail, são armazenadas na sessão.

Dependendo do tipo de utilizador (aluno, senhorio ou gestor), o redireccionamento é feito para a página apropriada após o login.

Tratamento de Contas Desativadas:

Se o utilizador estiver com o estado "Desativo", é retornado um redireccionamento com uma mensagem informando que a conta não está ativa.

Tratamento de Outros Estados de Conta:

Se o estado do utilizador não for nem "Desativo" nem "Ativo", é feito um redireccionamento para uma página de validação com o e-mail do utilizador na sessão, possivelmente para ativação ou confirmação de e-mail.

Em resumo, esse código gerencia o processo de login de utilizadores, garantindo que apenas utilizadores com contas ativas accedem ao sistema e redirecionando-os para a página apropriada após o login. Ele também lida com casos em que a conta está desativada ou requer validação adicional.

## 3.4 Registar

### 3.4.1 FRONT END

```
<div class="container-main">
  <div class="container-login">
    <div class="form-box-login">
      
      <form action="/registrar" method="post" >
        @csrf
        <div class="input-group-login">
          <label for="username">Username</label>
          <input type="text" id="username" name="username" required>
        </div>
        <div class="input-group-login">
          <label for="email">Email</label>
          <input type="email" id="email" name="email" required>
        </div>
        <div class="input-group-login">
          <label for="password">Password</label>
          <input type="password" id="password" name="password" required>
        </div>
        <div class="input-group-login">
          <label for="password_confirmation">Re-password</label>
          <input type="password" id="password_confirmation" name="password_confirmation" required>
        </div>
        <div style="text-align: center">
          <button type="submit">Registar</button>
        </div>
      </form>
    </div>
  </div>
</div>
```

Figure 17-form registrar

Este trecho de código HTML representa um formulário de registo de utilizador. Aqui está uma explicação de cada parte do código:

Estrutura do Formulário:

O formulário está contido numa estrutura de divs com as classes container-main, container-login e form-box-login, que podem ser usadas para aplicar estilos de layout específicos.

Imagen:

Uma imagem é exibida acima do formulário. A URL da imagem é definida como <https://comum.rcaap.pt/retrieve/104938>.

Campos do Formulário:

Existem quatro campos no formulário:

Username: Um campo de texto onde os utilizadores podem inserir o seu nome de utilizador.

Email: Um campo de texto onde os utilizadores podem inserir o seu endereço de e-mail.

Password: Um campo de senha onde os utilizadores podem inserir a sua senha.

Repassword. Um campo de senha onde os utilizadores devem confirmar a sua senha, digitando-a novamente.

### Botão de Envio:

Um botão "Registar" é fornecido para enviar o formulário. Quando clicado, o formulário será submetido para a rota /registrar usando o método HTTP POST.

### Validação dos Campos:

Todos os campos do formulário são marcados com o atributo required, o que significa serem obrigatórios. Isso garante que os utilizadores não possam enviar o formulário sem preencher todos os campos obrigatórios.

Este formulário é usado para coletar informações de registo de utilizadores, como nome de utilizador, e-mail e senha. Uma vez preenchido e enviado, os dados do formulário podem ser processados por um servidor “web” para criar uma conta de utilizador no sistema.

## 3.4.2 BACK-END

```
1 usage  ~ MuaiadHadad
public function registrar(Request $request){
    // 200 -> foi registado
    // 400 -> já existe o email
    $estado = strtoupper(Str::random( length: 6));

    $emailExists = DB::table( table: 'user')->where( column: 'Email', $request->email)->exists();

    if($emailExists) {
        return view( view: 'User\Register')->with('Code', '400');
    }
    $tipoUtilizador = strpos($request->email, needle: '@alunos.estgoh.ipc.pt') !== false ? 'aluno' : 'senhorio';
    DB::table( table: 'user')->insert([
        'UserName' => $request->username,
        'Email' => $request->email,
        'Password' => bcrypt($request->password),
        'Estado' => $estado,
        'Tipo' => $tipoUtilizador,
        'Avatar'=> 'avatars/User-avatar.svg.png'
    ]);

    $this->enviarEmail( titulo: 'Codigo de ativação de sua conta', $request->username , $estado,$request->email);

    return view( view: 'User\Register')->with('Code', '200');
}
```

Figure 18-controler registrar

Este trecho de código PHP está relacionado ao processo de registo de utilizadores a sistema. Aqui está uma explicação das principais partes:

### Geração de Estado Aleatório:

A variável \$estado é definida como uma sequência aleatória de 6 caracteres em maiúsculas. Isso é feito usando o método Str::random(6) para gerar uma string aleatória e strtoupper para convertê-la para maiúsculas.

### Verificação de E-mail Existente:

O código verifica se o e-mail fornecido já existe na tabela de utilizadores. Isso é feito consultando o banco de dados usando o método where e o método exists. Se o e-mail já existir, o código retorna uma visão específica com o código de status '400', indicando um erro.

#### Determinação do Tipo de Utilizador:

Com base no domínio do e-mail fornecido, o código determina se o usuário é um aluno ou um senhorio. Se o e-mail contiver @alunos.estgoh.ipc.pt, o utilizador será considerado um aluno; caso contrário, será considerado um senhorio.

#### Inserção no Banco de Dados:

Se o e-mail não existir na base de dados, os detalhes do utilizador (nome de utilizador, e-mail, senha, estado, tipo e avatar padrão) são inseridos na tabela de utilizadores usando o método insert do Laravel.

#### Envio de E-mail de Ativação:

Após a inserção bem-sucedida no banco de dados, um e-mail de ativação é enviado para o novo utilizador. O método enviar Email é chamado com os detalhes necessários para enviar o e-mail de ativação. Presumivelmente, esse método é responsável por enviar um e-mail para o utilizador com um código de ativação.

#### Retorno da Resposta:

Finalmente, uma visão específica é retornada com o código de status '200', indicando um registo bem-sucedido.

Esse código controla o processo de registo de utilizadores, garantindo que apenas e-mails únicos sejam registados, gerando um estado aleatório para a conta e enviando um e-mail de ativação para o utilizador recém-registado.

## 2.5 Email enviados

### 2.5.1 BACK-END

```
class EstadoEmail extends Mailable
{
    use Queueable, SerializesModels;

    2 usages
    public $subject;
    public $contentData;
    2 usages
    public $viewName;

    /**
     * Create a new message instance.
     *
     * @return void
     */
    4 usages ▲ MuaiadHadad
    public function __construct($subject, $contentData, $viewName)
    {
        $this->subject = $subject;
        $this->contentData = $contentData;
        $this->viewName = $viewName;
    }

    /**
     * Build the message.
     *
     * @return $this
     */
    no usages ▲ MuaiadHadad
    public function build()
    {
        return $this->subject($this->subject)
            ->view($this->viewName)
            ->with($this->contentData);
    }
}
```

```
/**  
 * Build the message.  
 *  
 * @return $this  
 */  
no usages  ↳ MuaiadHadad  
public function build()  
{  
    return $this->subject($this->subject)  
        ->view($this->viewName)  
        ->with($this->contentData);  
}
```

Figure 19-controlar email

Esta classe EstadoEmail é uma Mailable do Laravel, o que significa que ela é responsável por construir e enviar e-mails no framework Laravel. Daremos uma olhada nas partes principais da classe:

Propriedades Públicas:

\$subject: Armazena o assunto do e-mail.

\$contentData: Armazena os dados de conteúdo do e-mail.

\$viewName: Armazena o nome da visão (template) a ser usada para renderizar o e-mail.

Método Construtor:

**\_\_construct(\$subject, \$contentData, \$viewName):** Recebe o assunto do e-mail, os dados de conteúdo e o nome da visão como parâmetros e os atribui às propriedades correspondentes da classe.

Método build():

Este método é chamado para construir o e-mail.

**subject(\$this->subject):** Define o assunto do e-mail.

**view(\$this->viewName):** Define a visão (template) a ser usada para renderizar o e-mail.

**with(\$this->contentData):** Passa os dados de conteúdo para a visão do e-mail, permitindo que sejam acessados na visão como variáveis.

Essa classe é uma forma conveniente de estruturar e-mails no Laravel. Ela permite separar a lógica de construção e envio de e-mails do restante da aplicação, seguindo o conceito de responsabilidade única. Isso facilita a manutenção e o desenvolvimento de funcionalidades relacionadas ao envio de e-mails.

## 3.6 Validação código

### 3.6.1 BACK-END



```
1 usage  ~ MuaiadHadad
public function validateCode(Request $request){
    $codigo = $request->input('CODIGO');
    $email = $request->input('Email');

    $user = DB::table('user')->where('Estado', $codigo)->where('Email', $email)->first();

    if ($user) {
        DB::table('user')->where('Email', $email)->update(['Estado' => 'Ativo']);
        return redirect('/Login')->with('success', 'Código validado com sucesso! Você pode fazer login agora.');
    } else {
        return back()->with([
            'Error' => 'Código de validação inválido.',
            'Email' => $email
        ]);
    }
}
```

Figure 20-controlar ativação de utilizador

Este método validateCode é responsável por validar o código de ativação enviado por e-mail durante o processo de registo. Aqui está uma explicação detalhada do que ele faz:

Parâmetros do Método:

\$request: objeto que contém os dados da requisição HTTP, como os dados enviados pelo formulário.

Recuperação dos Dados:

\$codigo = \$request->input('CODIGO'): Obtém o código de ativação enviado pelo usuário.

\$email = \$request->input('Email'): Obtém o endereço de e-mail associado ao código de ativação.

Consulta ao Banco de Dados:

DB::table('user')->where('Estado', \$codigo)->where('Email', \$email)->first(): Consulta a tabela de usuários procurando por um registro que corresponda ao código de ativação e ao endereço de e-mail fornecido.

Validação do Resultado:

Se um utilizador for encontrado com o código de ativação e o endereço de e-mail correspondentes, isso significa que o código é válido.

Atualiza o estado do utilizador para "Ativo" no banco de dados.

Redireciona o utilizador para a página de login com uma mensagem de sucesso.

Se nenhum utilizador for encontrado com os dados fornecidos, significa que o código de ativação é inválido.

Retorna o utilizador à página anterior (provavelmente a página de registo) com uma mensagem de erro.

Esse método desempenha um papel importante no processo de registo de utilizadores, garantindo que apenas códigos de ativação válidos sejam aceites para ativar as contas dos utilizadores.

```
public function ReenviarCode(Request $request, $email){
    $estado = strtoupper(Str::random(6));
    $update=DB::table('user')->where('Email', $email)->update(['Estado' => $estado]);
    if($update){
        $this->enviarEmail( titulo: 'Codigo de ativação de sua conta', $estado,$email);
        return back()->with([
            'success'=> 'Código reenviado com sucesso para ' . $email,
            'Email' => $email
        ]);
    }else{
        return back()->with([
            'Error' => 'Não foi possível reenviar',
            'Email' => $email
        ]);
    }
}
```

Figure 21-controlar reenviar código de ativação

Este método ReenviarCode é responsável por reenviar o código de ativação para um determinado endereço de e-mail. Aqui está uma explicação passo a passo do que ele faz:

Parâmetros do Método:

\$request: objeto que contém os dados da requisição HTTP.

\$email: O endereço de e-mail para o qual o código de ativação será reenviado.

Geração de um Novo Código de Ativação:

\$estado = strtoupper(Str::random(6)): Gera um novo código de ativação aleatório com 6 caracteres em maiúsculas.

Atualização do Código de Ativação no Banco de Dados:

DB::table('user')->where('Email', \$email)->update(['Estado' => \$estado]): Atualiza o código de ativação para o novo valor gerado no banco de dados para o utilizador correspondente ao endereço de e-mail fornecido.

Envio do E-mail com o Novo Código:

Se a atualização for bem-sucedida:

Chama o método enviarEmail para enviar um e-mail com o novo código de ativação.

Retorna para a página anterior (provavelmente a página de registro) com uma mensagem de sucesso, indicando que o código foi reenviado com sucesso para o endereço de e-mail especificado.

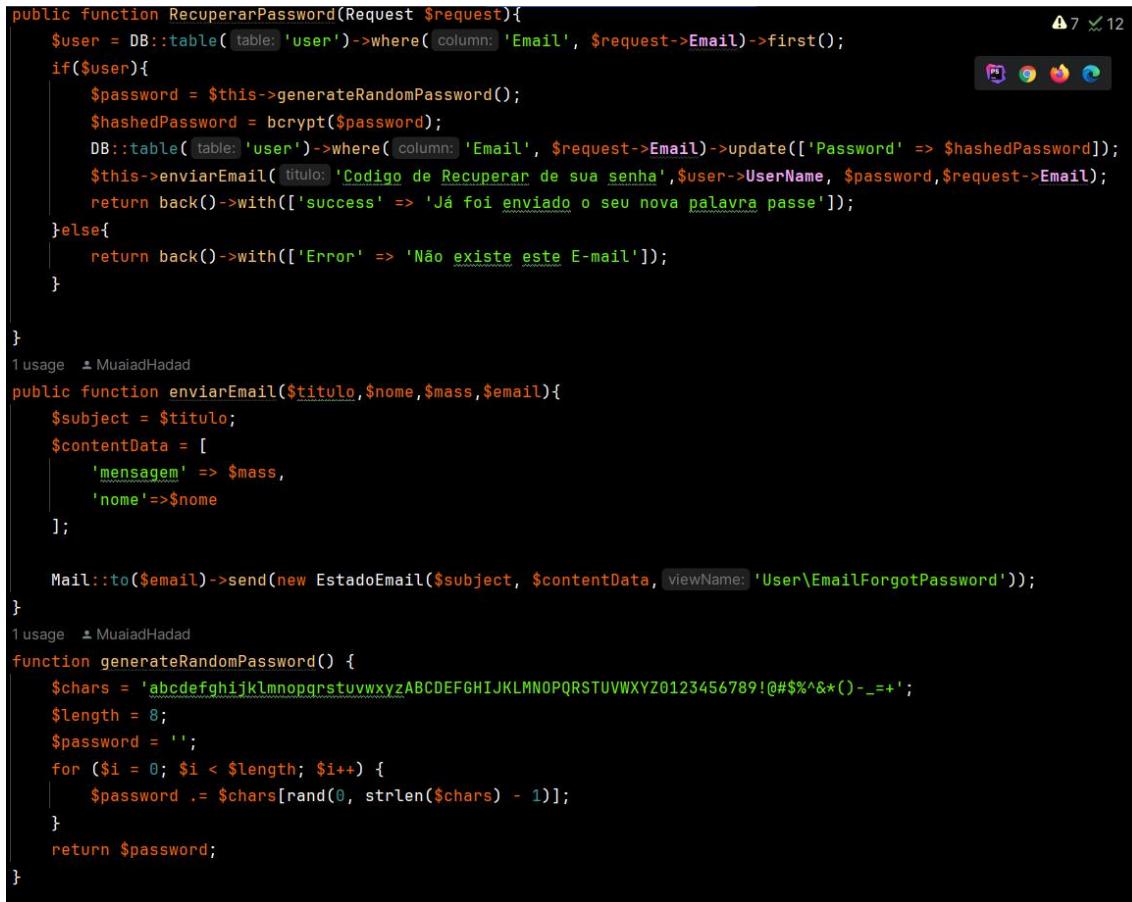
Se a atualização falhar:

Retorna para a página anterior com uma mensagem de erro informando que não foi possível reenviar o código.

Este método permite que os utilizadores solicitem um novo código de ativação caso o anterior tenha sido perdido ou não recebido. Isso garante uma experiência suave durante o processo de registo.

## 3.7 Recuperar a Password

### 3.7.1 BACK-END



The screenshot shows a code editor with a dark theme. The code is written in PHP and implements a password recovery feature. It includes methods for generating random passwords, updating user records, and sending emails. The code uses various frameworks and libraries, as indicated by the syntax highlighting and comments. The editor interface includes tabs for different files, a search bar, and a toolbar with icons for file operations.

```
public function RecuperarPassword(Request $request){
    $user = DB::table('user')->where('Email', $request->Email)->first();
    if($user){
        $password = $this->generateRandomPassword();
        $hashedPassword = bcrypt($password);
        DB::table('user')->where('Email', $request->Email)->update(['Password' => $hashedPassword]);
        $this->enviarEmail('Código de Recuperar de sua senha', $user->UserName, $password, $request->Email);
        return back()->with(['success' => 'Já foi enviado o seu nova palavra passe']);
    }else{
        return back()->with(['Error' => 'Não existe este E-mail']);
    }
}

1 usage  ↳ MuaiadHadad
public function enviarEmail($titulo, $nome, $mass, $email){
    $subject = $titulo;
    $contentData = [
        'mensagem' => $mass,
        'nome'=>$nome
    ];

    Mail::to($email)->send(new EstadoEmail($subject, $contentData, viewName: 'User\EmailForgotPassword'));
}
1 usage  ↳ MuaiadHadad
function generateRandomPassword() {
    $chars = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&*()_-+=';
    $length = 8;
    $password = '';
    for ($i = 0; $i < $length; $i++) {
        $password .= $chars[rand(0, strlen($chars) - 1)];
    }
    return $password;
}
```

Figure 22-controlar Forgot Password

Este código implementa a funcionalidade de recuperação de senha. Aqui está o que cada parte faz:

Método RecuperarPassword:

Primeiro, ele procura no banco de dados por um utilizador com o e-mail fornecido.

Se encontrar o utilizador:

Gera uma nova senha aleatória usando o método generateRandomPassword.

Hashes a nova senha.

Atualiza a senha do utilizador no banco de dados com a nova senha gerada.

Chama o método enviarEmail para enviar um e-mail ao utilizador com a nova senha.

Retorna para a página anterior (provavelmente a página de recuperação de senha) com uma mensagem de sucesso.

Se não encontrar o utilizador, retorna para a página anterior com uma mensagem de erro informando que o e-mail não existe.

Método enviarEmail:

Este método recebe um título, um nome, uma mensagem e um endereço de e-mail como argumentos.

Constrói o conteúdo do e-mail com o título, a mensagem e o nome.

Usa o objeto Mail para enviar o e-mail para o endereço fornecido, utilizando a classe EstadoEmail para renderizar o e-mail.

Método generateRandomPassword:

Este método gera uma senha aleatória de 8 caracteres, utilizando letras maiúsculas, minúsculas, números e alguns caracteres especiais.

Geralmente, esse código permite que os utilizadores solicitem a recuperação de senha e recebam um e-mail com uma nova senha temporária, permitindo que eles cessem a sua conta e posteriormente a alterem para uma senha da sua escolha.

## 3.8 Gestor área pessoal

### 3.8.1 FRONT-END

interface de utilizador para um sistema de gestão de utilizadores, anúncios e pedidos. Aqui está uma visão geral do que ele faz:

Toast Messages:

As mensagens de toast são exibidas para fornecer feedback ao utilizador sobre operações realizadas, como erros ou sucesso.

As mensagens de erro e sucesso são exibidas em elementos <div> posicionados à direita da tela.

A toast messages contêm ícones para indicar o tipo de mensagem (erro ou sucesso), bem como o texto da mensagem.

Seleção de Lista:

Uma seção de seleção de lista permite que o utilizador escolha entre diferentes tipos de listas, como lista de utilizadores, lista de anúncios ou lista de pedidos.

A seleção é feita via um menu suspenso.

Listas:

Existem três seções de listas representadas por <section> com classes específicas (module--list).

Cada seção de lista corresponde a um tipo específico de lista: lista de utilizadores, lista de anúncios ou lista de pedidos.

Cada seção de lista contém uma tabela (<table>) que exibe os dados relevantes.

Para cada tipo de lista, as colunas da tabela exibem informações como ID, nome, endereço, proprietário, data de expiração, preço, estado e opções de ação (por exemplo, remover, ativar/desativar, consultar).

As linhas da tabela são preenchidas com os dados obtidos da variável \$utilizadores, \$Casa, \$Quartos, \$DataQuartoPending e \$DataCasaPending, dependendo do tipo de lista sendo exibida.

Para cada item na lista, são fornecidas opções de ação, como ativar/desativar um utilizador ou anúncio, remover um utilizador ou anúncio, ou consultar detalhes adicionais.

Funcionalidade Interativa:

As seções de lista e as mensagens de toast ter funcionalidades interativas ativadas em resposta a ações do utilizador, como selecionar uma lista específica, pesquisar itens ou executar ações nas listas (como ativar/desativar ou remover utilizadores/anúncios).

Geralmente, “interface” de utilizador funcional para um sistema de gestão administrativa. Ele fornece uma maneira intuitiva para os administradores interagirem com e gerenciarem utilizadores, anúncios e pedidos no sistema.

### 3.8.2 BACK-END

```
2 usages ▾ MuaiadHadad
public function GetPageGestor(){
    if(session( key: 'tipo_usuario')=="gestor"){
        $token=session( key: 'ActivationToken');
        $user = DB::table( table: 'user')->where( column: 'ActivationToken', $token)->get();
        $utilizadores = DB::table( table: 'user')->get();
        $QuartosPending =DB::table( table: 'quarto')
            ->join( table: 'banho', first: 'banho.id_quarto', operator: '=', second: 'quarto.id')
            ->join( table: 'contato', first: 'contato.id_quarto', operator: '=', second: 'quarto.id')
            ->join( table: 'cozinha', first: 'cozinha.id_quarto', operator: '=', second: 'quarto.id')
            ->join( table: 'endereco', first: 'endereco.id_quarto', operator: '=', second: 'quarto.id')
            ->join( table: 'quartos_de_casa', first: 'quartos_de_casa.id_quarto', operator: '=', second: 'quarto.id')
            ->join( table: 'sala', first: 'sala.id_quarto', operator: '=', second: 'quarto.id')
            ->join( table: 'servicos', first: 'servicos.id_quarto', operator: '=', second: 'quarto.id')
            ->join( table: 'outros', first: 'outros.id_quarto', operator: '=', second: 'quarto.id')
            ->join( table: 'user', first: 'user.id', operator: '=', second: 'quarto.id_user')
            ->where( column: 'quarto.estado', operator: 'pending')
            ->select( columns: 'user.*', 'quarto.id as idnow', 'quarto.*', 'banho.*', 'contato.*', 'cozinha.*', 'endereco.*'
                , 'quartos_de_casa.*'
                , 'sala.*', 'servicos.*', 'outros.*')->get();
        $CasaPending =DB::table( table: 'casa_completa')
            ->join( table: 'banho', first: 'banho.id_casa', operator: '=', second: 'casa_completa.id')
            ->join( table: 'contato', first: 'contato.id_casa', operator: '=', second: 'casa_completa.id')
            ->join( table: 'cozinha', first: 'cozinha.id_casa', operator: '=', second: 'casa_completa.id')
            ->join( table: 'endereco', first: 'endereco.id_casa', operator: '=', second: 'casa_completa.id')
            ->join( table: 'sala', first: 'sala.id_casa', operator: '=', second: 'casa_completa.id')
            ->join( table: 'servicos', first: 'servicos.id_casa', operator: '=', second: 'casa_completa.id')
            ->join( table: 'outros', first: 'outros.id_casa', operator: '=', second: 'casa_completa.id')
            ->join( table: 'user', first: 'user.id', operator: '=', second: 'casa_completa.id_user')
            ->where( column: 'casa_completa.estado', operator: 'pending')
            ->select( columns: 'user.*', 'casa_completa.id as idnow', 'casa_completa.*', 'banho.*', 'contato.*', 'cozinha.*', 'endereco.*'
```

```

        , 'sala.*', 'servicos.*', 'outros.*')->get();
$Quartos =DB::table( table: 'quarto')
->join( table: 'banho', first: 'banho.id_quarto', operator: '=', second: 'quarto.id')
->join( table: 'contato', first: 'contato.id_quarto', operator: '=', second: 'quarto.id')
->join( table: 'cozinha', first: 'cozinha.id_quarto', operator: '=', second: 'quarto.id')
->join( table: 'endereco', first: 'endereco.id_quarto', operator: '=', second: 'quarto.id')
->join( table: 'quartos_de_casa', first: 'quartos_de_casa.id_quarto', operator: '=', second: 'quarto.id')
->join( table: 'sala', first: 'sala.id_quarto', operator: '=', second: 'quarto.id')
->join( table: 'servicos', first: 'servicos.id_quarto', operator: '=', second: 'quarto.id')
->join( table: 'outros', first: 'outros.id_quarto', operator: '=', second: 'quarto.id')
->join( table: 'user', first: 'user.id', operator: '=', second: 'quarto.id_user')
->select( columns: 'user.*', 'quarto.id as idnow', 'quarto.*', 'banho.*', 'contato.*', 'cozinha.*', 'endereco.*'
        , 'quartos_de_casa.*'
        , 'sala.*', 'servicos.*', 'outros.*')->get();
$Casa =DB::table( table: 'casa_completa')
->join( table: 'banho', first: 'banho.id_casa', operator: '=', second: 'casa_completa.id')
->join( table: 'contato', first: 'contato.id_casa', operator: '=', second: 'casa_completa.id')
->join( table: 'cozinha', first: 'cozinha.id_casa', operator: '=', second: 'casa_completa.id')
->join( table: 'endereco', first: 'endereco.id_casa', operator: '=', second: 'casa_completa.id')
->join( table: 'sala', first: 'sala.id_casa', operator: '=', second: 'casa_completa.id')
->join( table: 'servicos', first: 'servicos.id_casa', operator: '=', second: 'casa_completa.id')
->join( table: 'outros', first: 'outros.id_casa', operator: '=', second: 'casa_completa.id')
->join( table: 'user', first: 'user.id', operator: '=', second: 'casa_completa.id_user')
->select( columns: 'user.*', 'casa_completa.id as idnow', 'casa_completa.*', 'banho.*', 'contato.*', 'cozinha.*', 'endereco.*'
        , 'sala.*', 'servicos.*', 'outros.*')->get();
return view( view: 'Page_Gestor\Gestor', [ 'utilizadores' => $utilizadores, 'Data'=>$user, 'DataCasaPending'=>$CasaPending,
    'DataQuartoPending'=>$QuartosPending, 'Quartos'=>$Quartos, 'Casa'=>$Casa]);
} else{
    abort( code: 403, message: 'Acesso não autorizado.');
}
}

```

**Figure 23-controlar de página principal de gestor**

Essa função GetPageGestor é responsável por renderizar a página do gestor, exibindo informações relevantes para utilizadores com permissão de gestor. Aqui está uma visão geral do que ela faz:

Verificação de Permissão:

A função verifica se o tipo de utilizador na sessão é "gestor".

Se não for um gestor, retorna um erro 403 (Acesso não autorizado).

Recuperação de Dados:

Se o utilizador for um gestor, a função continua a recuperar os dados relevantes para exibição na página do gestor.

O token de ativação do utilizador é recuperado da sessão.

Os dados do utilizador, incluindo os dados do utilizador com base no token de ativação, são recuperados do banco de dados.

Os dados de diferentes tipos de propriedades (quartos e casas) são recuperados separadamente do banco de dados. Existem três conjuntos de dados diferentes:

QuartosPending: Quartos que estão pendentes de aprovação.

CasaPending: Casas completas que estão pendentes de aprovação.

Quartos: Todos os quartos disponíveis.

Casa: Todas as casas completas disponíveis.

Renderização da Página:

Os dados recuperados são passados para a visão Page\_Gestor\Gestor para renderização.

Os dados de utilizadores, quartos pendentes, casas pendentes, quartos e casas são passados como variáveis para a visão.

Retorno de Erro para Acesso Não Autorizado:

Se o utilizador não for um gestor, a função chama abort(403, 'Acesso não autorizado.'), que retorna um erro 403 (Acesso não autorizado).

Essa função é responsável por garantir que apenas gestores autorizados tenham acesso à página do gestor e exibam as informações relevantes para eles.

## 3.9 Adicionar

### 3.9.1 BACK-END

```
1 usage  ± MuaiadHadad
public function Adcionargestor(Request $request){
    $username = $request->username;
    $email = $request->Email;
    $userExists = DB::table('user')->where('Email', $email)->exists();
    if ($userExists) {
        return redirect()->back()->with('error', 'Este e-mail já está em uso. Por favor, escolha outro.');
    }
    $password = $this->generateRandomPassword();
    $hashedPassword = bcrypt($password);
    DB::table('user')->insert([
        'UserName' => $request->username,
        'Email' => $request->Email,
        'Password' => $hashedPassword,
        'Estado' => 'Ativo',
        'Tipo' => 'gestor',
        'Avatar' => 'avatars/User-avatar.svg.png'
    ]);
    $this->enviarEmail(título: 'Bem-vindo à Plataforma de Alojamento da ESTGOH!', $request->username, $password, $request->Email);
    return redirect()->back()->with('success', 'Gestor adicionado com sucesso!');
}
```

Figure 24-controlar add gestor

Esta função Adcionargestor é responsável por adicionar um novo gestor ao sistema. Aqui está uma visão geral do que ela faz:

Recebimento dos Dados:

A função recebe os dados do novo gestor por meio do objeto Request.

Os dados incluem o nome de utilizador, o email e outros detalhes necessários.

Verificação de Existência de Utilizador:

Verifica se já existe um utilizador com o email fornecido no banco de dados.

Se o email já estiver em uso, redireciona de volta à página anterior com uma mensagem de erro.

Geração de Senha Aleatória:

Gera uma senha aleatória para o novo gestor usando a função generateRandomPassword.

### Hashing da Senha:

A senha gerada é então hashada usando a função bcrypt antes de ser armazenada no banco de dados.

### Inserção no Banco de Dados:

Insere os detalhes do novo gestor, incluindo nome de utilizador, email, senha hashada, estado, tipo e avatar, no banco de dados.

### Envio de Email de Boas-Vindas:

Um email de boas-vindas é enviado para o novo gestor contendo o seu nome de utilizador e senha gerada aleatoriamente.

### Redirecionamento com Mensagem de Sucesso:

Após adicionar o gestor com sucesso e enviar o email, o utilizador é redirecionado de volta à página anterior com uma mensagem de sucesso.

Essa função garante que novos gestores possam ser adicionados ao sistema de forma segura e eficiente, além de fornecer feedback adequado ao utilizador sobre o resultado da operação.

## 3.10 Profile

### 3.10.1 BACK-END

```
2 usages  ↗ MuaiadHadad
public function GetPageProfGestor(){
    if(session( key: 'tipo_usuario')=="gestor"){
        $token=session( key: 'ActivationToken');
        $utilizadores = DB::table( table: 'user')->where( column: 'ActivationToken', $token)->get();
        if ($utilizadores) {
            return view( view: 'Page_Gestor\Profile', ['utilizadores' => $utilizadores]);
        } else {
            abort( code: 403, message: 'Bad request');
        }
    }else{
        abort( code: 403, message: 'Acesso não autorizado.');
    }
}
```

Figure 25-dados do utilizador

```
public function updateProfile(Request $request, $id){  
    if($request->passwordnovo!=null){  
        $hashedPassword = bcrypt($request->passwordnovo);  
        if($request->passwordold!=null ){  
            $password=$request->input( key: 'passwordold');  
            $utilizadores = DB::table( table: 'user')->where( column: 'id', $id)->first();  
            if (! $utilizadores || ! password_verify($password, $utilizadores->Password)) {  
                return back()->with(['error' => 'Palavra passa está incorreta!']);  
            }else{  
                DB::table( table: 'user')->where( column: 'id', $id)->update([  
                    'Password' => $hashedPassword  
                ]);  
            }  
        }  
    }  
    if($request->file( key: 'avatar')!=null){  
        $avatarPath = $request->file( key: 'avatar')->store( path: 'avatars', options: 'public');  
        DB::table( table: 'user')->where( column: 'id', $id)->update([  
            'Avatar' => $avatarPath,  
        ]);  
    }  
    DB::table( table: 'user')->where( column: 'id', $id)->update([  
        'UserName' => $request->username,  
        'Email' => $request->Email,  
    ]);  
    return redirect()->back()->with('success', 'Perfil do usuário atualizado com sucesso!');  
}
```

Figure 26-update dados de utilizador

Este trecho de código HTML representa um formulário de registo de utilizador. Aqui está uma explicação de cada parte do código:

Estrutura do Formulário:

O formulário está contido numa estrutura de divs com as classes container-main, container-login e form-box-login, que podem ser usadas para aplicar estilos de layout específicos.

Imagen:

Uma imagem é exibida acima do formulário. A URL da imagem é definida como <https://comum.rcaap.pt/retrieve/104938>.

Campos do Formulário:

Existem quatro campos no formulário:

Username: Um campo de texto onde os utilizadores podem inserir o seu nome de utilizador.

Email: Um campo de texto onde os utilizadores podem inserir o seu endereço de e-mail.

Password: Um campo de senha onde os utilizadores podem inserir a sua senha.

Repassword. Um campo de senha onde os utilizadores devem confirmar a sua senha, digitando-a novamente.

#### Botão de Envio:

Um botão "Registar" é fornecido para enviar o formulário. Quando clicado, o formulário será submetido para a rota /registar usando o método HTTP POST.

#### Validação dos Campos:

Todos os campos do formulário são marcados com o atributo required, o que significa serem obrigatórios. Isso garante que os utilizadores não possam enviar o formulário sem preencher todos os campos obrigatórios.

Este formulário é usado para coletar informações de registo de utilizadores, como nome de utilizador, e-mail e senha. Uma vez preenchido e enviado, os dados do formulário podem ser processados por um servidor “web” para criar uma conta de utilizador no sistema.

### 3.11 Back-end remover utilizador

```
1 usage  ± MuaiadHadad
public function removerUser($id){
    $user = DB::table('user')->where('id', $id)->first();

    if (!$user) {
        return redirect()->back()->with('error', 'Não foi possível remover o utilizador!');
    }
    DB::table('user')->where('id', $id)->delete();

    return redirect()->back()->with('success', 'utilizador foi removido com sucesso!');
}

1 usage  ± MuaiadHadad
public function GetPageAddGestor(){
    if(session('tipo_usuario')=="gestor"){
        $token=session('ActivationToken');
        $user = DB::table('user')->where('ActivationToken', $token)->get();
        return view('Page_Gestor\Adicionar_Gestor',[ 'Data'=>$user]);
    }else{
        abort(403, 'Acesso não autorizado.');
    }
}
```

Figure 27-remover utilizador

Aqui está uma análise das funções removerUser e GetPageAddGestor:

removerUser(\$id)

Recuperação do Utilizador:

A função tenta recuperar o utilizador com o ID fornecido do banco de dados.

Verificação da Existência do Utilizador:

Verifica se o utilizador existe. Se não existir, redireciona de volta com uma mensagem de erro.

Remoção do Utilizador:

Se o utilizador existir, remove-o do banco de dados.

Redirecionamento com Mensagem de Sucesso:

Após a remoção bem-sucedida, redireciona de volta com uma mensagem de sucesso.

GetPageAddGestor()

Verificação de Permissões:

Verifica se o tipo de utilizador armazenado na sessão é "gestor". Se não for, gera um erro 403 (Acesso não autorizado).

Recuperação de Dados do Utilizador:

Se o tipo de utilizador for "gestor", recupera os detalhes do utilizador (gestor) atual com base no token de ativação armazenado na sessão.

Renderização da Página de Adicionar Gestor:

Se os detalhes do utilizador (gestor) forem recuperados com sucesso, renderiza a página de adicionar gestor, passando os detalhes do utilizador como dados para a visualização.

Caso contrário, retorna um erro 403 (Solicitação inválida).

Essas funções garantem que apenas gestores possam remover utilizadores e acessar a página para adicionar gestores. Além disso, fornecem feedback apropriado ao utilizador sobre o resultado das operações.

### 3.12 Back-end consultar anúncio

```

1 usage  ~ MuaiadHadad
public function GetPageDetalheQuarto($id)
{
    if (session('ActivationToken') != null) {
        $token = session('ActivationToken');
        $user = DB::table('user')->where('column: ActivationToken', $token)->get();
        $QuartosAtive = DB::table('quarto')
            ->join('banho', first: 'banho.id_quarto', operator: '=', second: 'quarto.id')
            ->join('contato', first: 'contato.id_quarto', operator: '=', second: 'quarto.id')
            ->join('cozinha', first: 'cozinha.id_quarto', operator: '=', second: 'quarto.id')
            ->join('endereco', first: 'endereco.id_quarto', operator: '=', second: 'quarto.id')
            ->join('quartos_de_casa', first: 'quartos_de_casa.id_quarto', operator: '=', second: 'quarto.id')
            ->join('sala', first: 'sala.id_quarto', operator: '=', second: 'quarto.id')
            ->join('servicos', first: 'servicos.id_quarto', operator: '=', second: 'quarto.id')
            ->join('outros', first: 'outros.id_quarto', operator: '=', second: 'quarto.id')
            ->join('user', first: 'user.id', operator: '=', second: 'quarto.id_user')
            ->where(column: 'quarto.id', $id)
            ->select(columns: 'contato.Email as EmailQuarto', 'user.*', 'cozinha.Micro-ondas as Micro', 'quarto.id as id'
                , 'quartos_de_casa.*'
                , 'sala.*', 'servicos.*', 'outros.*', 'servicos.Wi-Fi as wifi')->get();
        if (!$QuartosAtive){
            abort(code: 404, message: 'Bad Request.');
        }
        $PhotoQuarto =DB::table('quarto')
            ->join('midia_de_casa', first: 'midia_de_casa.id_quarto', operator: '=', second: 'quarto.id')
            ->where(column: 'quarto.id', $id)
            ->select(columns: 'midia_de_casa.*')->get();

        return view(view: 'Page_Gestor\detalhe_quarto_Gestor', ['DadosUser' => $user, 'DadosQuarto'=>$QuartosAtive, 'PhotoQuarto'=>$PhotoQuarto]);
    }else{
        $QuartosAtive =DB::table('quarto')
            ->join('banho', first: 'banho.id_quarto', operator: '=', second: 'quarto.id')
            ->join('contato', first: 'contato.id_quarto', operator: '=', second: 'quarto.id')
            ->join('cozinha', first: 'cozinha.id_quarto', operator: '=', second: 'quarto.id')
            ->join('endereco', first: 'endereco.id_quarto', operator: '=', second: 'quarto.id')
            ->join('quartos_de_casa', first: 'quartos_de_casa.id_quarto', operator: '=', second: 'quarto.id')
            ->join('sala', first: 'sala.id_quarto', operator: '=', second: 'quarto.id')
            ->join('servicos', first: 'servicos.id_quarto', operator: '=', second: 'quarto.id')
            ->join('outros', first: 'outros.id_quarto', operator: '=', second: 'quarto.id')
            ->join('user', first: 'user.id', operator: '=', second: 'quarto.id_user')
            ->where(column: 'quarto.id', $id)
            ->select(columns: 'contato.Email as EmailQuarto', 'user.*', 'cozinha.Micro-ondas as Micro', 'quarto.id as id'
                , 'quartos_de_casa.*'
                , 'sala.*', 'servicos.*', 'outros.*', 'servicos.Wi-Fi as wifi')->get();
        if (!$QuartosAtive){
            abort(code: 404, message: 'Bad Request.');
        }
        $PhotoQuarto =DB::table('quarto')
            ->join('midia_de_casa', first: 'midia_de_casa.id_quarto', operator: '=', second: 'quarto.id')
            ->where(column: 'quarto.id', $id)
            ->select(columns: 'midia_de_casa.*')->get();

        return view(view: 'Page_Gestor\detalhe_quarto_Gestor', ['DadosUser' => null, 'DadosQuarto'=>$QuartosAtive, 'PhotoQuarto'=>$PhotoQuarto]);
    }
}

```

Figure 28-consultar anúncio

Aqui está uma análise da função GetPageDetalheQuarto:

### GetPageDetalheQuarto(\$id)

Verificação da Sessão:

Verifica se há um token de ativação armazenado na sessão.

Recuperação dos Dados do Utilizador:

Se houver um token de ativação na sessão, a função recupera os detalhes do utilizador com base nesse token.

Recuperação dos Dados do Quarto:

A função busca os detalhes do quarto com o ID fornecido no banco de dados.

Os dados incluem informações sobre o quarto, como detalhes de contrato, comodidades, localização, etc.

Se nenhum quarto for encontrado com o ID fornecido, a função aborta com um erro 404 (Solicitação inválida).

Recuperação das Fotos do Quarto:

A função também busca as fotos associadas ao quarto no banco de dados.

Renderização da Página:

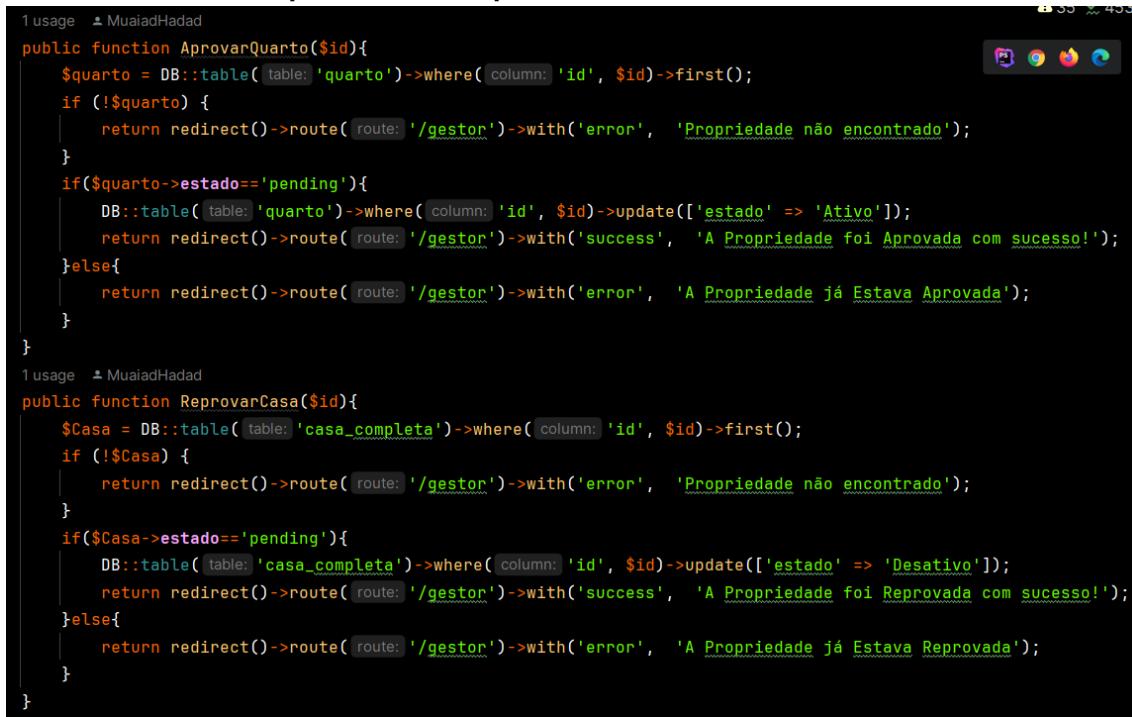
Com base nos dados recuperados, a função renderiza a página de detalhes do quarto.

Se houver um utilizador na sessão, os detalhes do utilizador são passados para a visualização, com os detalhes e fotos do quarto.

Se não houver utilizador na sessão, apenas os detalhes e fotos do quarto são passados para a visualização.

Essa função permite visualizar os detalhes de um quarto específico e as fotos associadas a ele. Dependendo da presença ou ausência de um utilizador na sessão, a página renderizada pode exibir ou não informações adicionais sobre o utilizador.

### 3.13 Back-end aprovar e reprovar anúncio



```
1 usage  ~ MuaiadHadad
public function AprovarQuarto($id){
    $quarto = DB::table('quarto')->where('id', $id)->first();
    if (!$quarto) {
        return redirect()->route('/gestor')->with('error', 'Propriedade não encontrado');
    }
    if($quarto->estado=='pending'){
        DB::table('quarto')->where('id', $id)->update(['estado' => 'Ativo']);
        return redirect()->route('/gestor')->with('success', 'A Propriedade foi Aprovada com sucesso!');
    }else{
        return redirect()->route('/gestor')->with('error', 'A Propriedade já Estava Aprovada');
    }
}

1 usage  ~ MuaiadHadad
public function ReprovVarCasa($id){
    $Casa = DB::table('casa_completa')->where('id', $id)->first();
    if (!$Casa) {
        return redirect()->route('/gestor')->with('error', 'Propriedade não encontrado');
    }
    if($Casa->estado=='pending'){
        DB::table('casa_completa')->where('id', $id)->update(['estado' => 'Desativo']);
        return redirect()->route('/gestor')->with('success', 'A Propriedade foi Reprovada com sucesso!');
    }else{
        return redirect()->route('/gestor')->with('error', 'A Propriedade já Estava Reprovada');
    }
}
```

Figure 29-Aprovar e reprovar anúncio

Aqui está uma análise das funções AprovarQuarto e ReprovVarCasa:

AprovarQuarto(\$id)

Recuperação do Quarto:

A função tenta recuperar os detalhes do quarto com o ID fornecido no banco de dados.

Verificação da Existência do Quarto:

Se o quarto não for encontrado, a função redireciona de volta à página do gestor com uma mensagem de erro.

Verificação do Estado do Quarto:

Verifica se o estado do quarto é 'pending' (pendente).

Se o estado for 'pending', atualiza o estado do quarto para 'Ativo' e redireciona de volta à página do gestor com uma mensagem de sucesso.

Se o estado do quarto não for 'pending', significa que já está aprovado, então a função redireciona de volta à página do gestor com uma mensagem de erro.

ReprovVarCasa(\$id)

Recuperação da Casa:

A função tenta recuperar os detalhes da casa completa com o ID fornecido no banco de dados.

#### Verificação da Existência da Casa:

Se a casa completa não for encontrada, a função redireciona de volta à página do gestor com uma mensagem de erro.

#### Verificação do Estado da Casa:

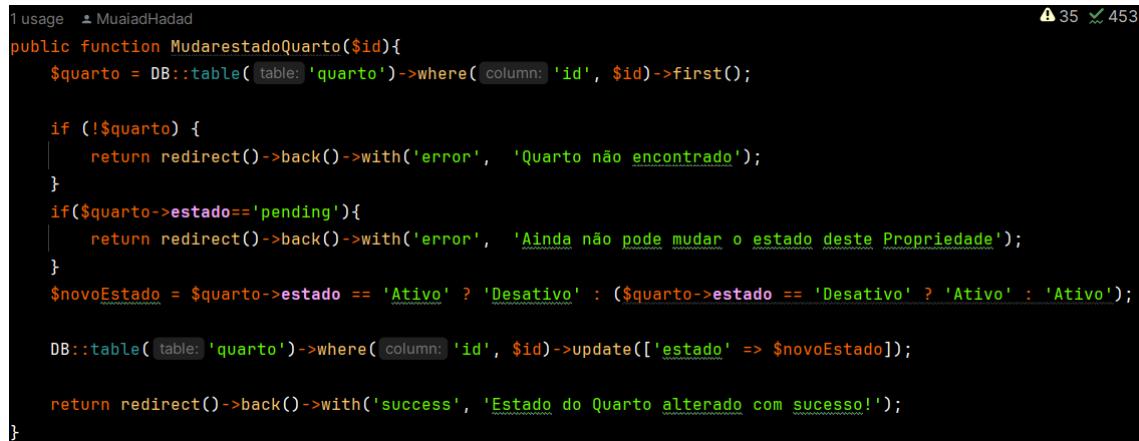
Verifica se o estado da casa é 'pending' (pendente).

Se o estado for 'pending', atualiza o estado da casa para 'Desativo' e redireciona de volta à página do gestor com uma mensagem de sucesso.

Se o estado da casa não for 'pending', significa que já está reprovada, então a função redireciona de volta à página do gestor com uma mensagem de erro.

Ambas as funções garantem que a aprovação ou reprovação seja feita apenas se o item estiver no estado adequado e fornecem feedback apropriado ao utilizador após a operação.

## 3.14 Back-end mudar estado de um anúncio



The screenshot shows a GitHub code review interface. At the top right, there are statistics: 35 reviews and 453 comments. The code itself is a PHP function named `MudarestadoQuarto($id)`. It starts by querying the database for a room with the given ID. If no room is found, it redirects back with an error message. If the room's state is 'pending', it also redirects back with an error message. Otherwise, it updates the room's state to 'Ativo' if it was 'Desativo', or to 'Desativo' if it was 'Ativo'. Finally, it redirects back with a success message indicating the state was changed successfully.

```
Usage  ↗ MuaiadHadad
public function MudarestadoQuarto($id){
    $quarto = DB::table('quarto')->where('id', $id)->first();

    if (!$quarto) {
        return redirect()->back()->with('error', 'Quarto não encontrado');
    }
    if($quarto->estado=='pending'){
        return redirect()->back()->with('error', 'Ainda não pode mudar o estado deste Propriedade');
    }
    $novoEstado = $quarto->estado == 'Ativo' ? 'Desativo' : ($quarto->estado == 'Desativo' ? 'Ativo' : 'Ativo');

    DB::table('quarto')->where('id', $id)->update(['estado' => $novoEstado]);

    return redirect()->back()->with('success', 'Estado do Quarto alterado com sucesso!');
}
```

Figure 30-mudar estado de um anúncio

Aqui está uma análise da função `MudarestadoQuarto($id)`:

`MudarestadoQuarto($id)`

Recuperação do Quarto:

A função tenta recuperar os detalhes do quarto com o ID fornecido no banco de dados.

Verificação da Existência do Quarto:

Se o quarto não for encontrado, a função redireciona de volta à página anterior com uma mensagem de erro.

Verificação do Estado do Quarto:

Verifica se o estado do quarto é 'pending' (pendente).

Se o estado for 'pending', significa que ainda não pode mudar o estado deste quarto, então a função redireciona de volta à página anterior com uma mensagem de erro.

Atualização do Estado do Quarto:

Determina o novo estado do quarto com base no estado atual.

Se o estado atual for 'Ativo', o novo estado será 'Desativo'.

Se o estado atual for 'Desativo', o novo estado será 'Ativo'.

Atualiza o estado do quarto no banco de dados com o novo estado.

Redirecionamento com Mensagem de Sucesso:

Após a atualização bem-sucedida, a função redireciona de volta à página anterior com uma mensagem de sucesso informando que o estado do quarto foi alterado com sucesso.

Essa função permite alternar entre os estados 'Ativo' e 'Desativo' do quarto e fornece feedback adequado ao utilizador após a operação.

## 3.15 Senhorio área pessoal

### 3.15.1 BACK-END

```
public function GetPageSenhorio(){
    if(session( key: 'tipo_usuario')=="senhorio"){
        $token=session( key: 'ActivationToken');
        $user = DB::table( table: 'user')->where( column: 'ActivationToken', $token)->get();
        $userqu = DB::table( table: 'user')->where( column: 'ActivationToken', $token)->first();
        $quartos =DB::table( table: 'quarto')
            ->join( table: 'banho', first: 'banho.id_quarto', operator: '=', second: 'quarto.id')
            ->join( table: 'contato', first: 'contato.id_quarto', operator: '=', second: 'quarto.id')
            ->join( table: 'cozinha', first: 'cozinha.id_quarto', operator: '=', second: 'quarto.id')
            ->join( table: 'endereco', first: 'endereco.id_quarto', operator: '=', second: 'quarto.id')
            ->join( table: 'quartos_de_casa', first: 'quartos_de_casa.id_quarto', operator: '=', second: 'quarto.id')
            ->join( table: 'sala', first: 'sala.id_quarto', operator: '=', second: 'quarto.id')
            ->join( table: 'servicos', first: 'servicos.id_quarto', operator: '=', second: 'quarto.id')
            ->join( table: 'outros', first: 'outros.id_quarto', operator: '=', second: 'quarto.id')
            ->where( column: 'id_user', $userqu->id)
            ->select( columns: 'quarto.id as idnow', 'quarto.*', 'banho.*', 'contato.*', 'cozinha.*', 'endereco.*'
                , 'quartos_de_casa.*'
                , 'sala.*', 'servicos.*', 'outros.*')->get();
        $Casa =DB::table( table: 'casa_completa')
            ->join( table: 'banho', first: 'banho.id_casa', operator: '=', second: 'casa_completa.id')
            ->join( table: 'contato', first: 'contato.id_casa', operator: '=', second: 'casa_completa.id')
            ->join( table: 'cozinha', first: 'cozinha.id_casa', operator: '=', second: 'casa_completa.id')
            ->join( table: 'endereco', first: 'endereco.id_casa', operator: '=', second: 'casa_completa.id')
            ->join( table: 'sala', first: 'sala.id_casa', operator: '=', second: 'casa_completa.id')
            ->join( table: 'servicos', first: 'servicos.id_casa', operator: '=', second: 'casa_completa.id')
            ->join( table: 'outros', first: 'outros.id_casa', operator: '=', second: 'casa_completa.id')
            ->where( column: 'id_user', $userqu->id)
            ->select( columns: 'casa_completa.id as idnow', 'casa_completa.*', 'banho.*', 'contato.*', 'cozinha.*', 'endereco.*'
                , 'sala.*', 'servicos.*', 'outros.*')->get();
        return view( view: 'Page_Senhorio\Senhorio_Principal_page', ['Data'=>$user, 'Quarto'=>$quartos,'Casa'=>$Casa]);
    }else{
        abort( code: 403, message: 'Acesso não autorizado.')
    }
}
```

Figure 31-controlar área pessoal senhorio

Aqui está uma análise da função GetPageSenhorio():

### GetPageSenhorio()

Verificação do Tipo de Usuário:

Verifica se o tipo de usuário na sessão é "senhorio". Se não for, aborta a solicitação com um código de status 403 (Acesso não autorizado).

Recuperação do Usuário Atual:

Obtém o token da sessão.

Recupera os detalhes do utilizador com base no token de ativação da sessão.

Recuperação dos Quartos e Casas do Senhorio:

Recupera os quartos do senhorio do banco de dados, com os seus detalhes de casa de banho, contato, cozinha, endereço, sala, serviços e outros.

Recupera as casas completas do senhorio do banco de dados, com os seus detalhes de casa de banho, contrato, cozinha, endereço, sala, serviços e outros.

Renderização da Página:

Renderiza a página 'Senhorio\_Principal\_page' e passa os seguintes dados para a visualização:

Detalhes do utilizador.

Quartos do senhorio.

Casas completas do senhorio.

Essa função é responsável por exibir a página principal para os senhorios após verificar se o tipo de utilizador na sessão é um senhorio. Ele fornece uma visão geral dos quartos e casas do senhorio, permitindo que eles gerenciem as suas propriedades.

## 3.16 Editar anúncio

### 3.16.1 BACK-END

```

public function EditCasa($id){
    if(session('key','tipo_usuario')=="senhorio") {
        $token = session('key','ActivationToken');
        $user = DB::table('table: user')->where(column: 'ActivationToken', $token)->get();
        $Casativate =DB::table('table: casa_completa')
            ->join('table: banho', first: 'banho.id_casa', operator: '=', second: 'casa_completa.id')
            ->join('table: contato', first: 'contato.id_casa', operator: '=', second: 'casa_completa.id')
            ->join('table: cozinha', first: 'cozinha.id_casa', operator: '=', second: 'casa_completa.id')
            ->join('table: endreco', first: 'endereco.id_casa', operator: '=', second: 'casa_completa.id')
            ->join('table: sala', first: 'sala.id_casa', operator: '=', second: 'casa_completa.id')
            ->join('table: servicos', first: 'servicos.id_casa', operator: '=', second: 'casa_completa.id')
            ->join('table: outros', first: 'outros.id_casa', operator: '=', second: 'casa_completa.id')
            ->join('table: user', first: 'user.id', operator: '=', second: 'casa_completa.id_user')
            ->where(column: 'casa_completa.id', $id)
            ->select(columns: 'contato.Email as EmailQuarto','user.*','cozinha.Micro-ondas as Micro','casa_completa.id as idnow','casa_completa.*','banho.*','contat
            , 'sala.*', 'servicos.*', 'outros.*','servicos.Wi-Fi as wifi')->get();
        if(!$Casativate){
            abort( code: 404, message: 'Bad Request.');
        }
        $PhotoCasa =DB::table('table: casa_completa')
            ->join('table: midia_de_casa', first: 'midia_de_casa.id_casa', operator: '=', second: 'casa_completa.id')
            ->where(column: 'casa_completa.id', $id)
            ->select(columns: 'midia_de_casa.*')->get();
        $QuartosCasa =DB::table('table: casa_completa')
            ->join('table: quartos_de_casa', first: 'quartos_de_casa.id_casa', operator: '=', second: 'casa_completa.id')
            ->where(column: 'casa_completa.id', $id)
            ->select(columns: 'quartos_de_casa.*')->get();
        return view('Page_Senhorio>EditCasa', ['Data' => $user,'DadosCasaAtive'=>$Casativate,'PhotoCasa'=>$PhotoCasa,'DadosQuartoCasa'=>$QuartosCasa]);
    }else{
        abort( code: 403, message: 'Acesso não autorizado.');
    }
}

```

**Figure 32-enviar dados de anúncio para página de edit**

Aqui está uma análise da função EditCasa(\$id):

EditCasa(\$id)

Verificação do Tipo de Utilizador:

Verifica se o tipo de utilizador na sessão é "senhorio". Se não for, aborta a solicitação com um código de status 403 (Acesso não autorizado).

Recuperação do Utilizador Atual e Detalhes da Casa:

Obtém o token da sessão.

Recupera os detalhes do utilizador com base no 'token' de ativação da sessão.

Recupera os detalhes da casa completa com o ID fornecido, com os seus detalhes de casa de banho, contrato, cozinha, endereço, sala, serviços e outros.

Se não encontrar a casa com o ID fornecido, aborta a solicitação com um código de status 404 (Solicitação inválida).

Recuperação das Fotos da Casa:

Recupera as fotos da casa completa com o ID fornecido.

Recuperação dos Quartos da Casa:

Recupera os quartos da casa completa com o ID fornecido.

Renderização da Página de Edição da Casa:

Renderiza a página 'EditCasa' e passa os seguintes dados para a visualização:

Detalhes do utilizador.

Detalhes da casa completa.

Fotos da casa completa.

Quartos da casa completa.

Essa função permite que os senhorios editem os detalhes de uma casa completa específica, incluindo os seus quartos, e visualizem as fotos associadas a ela.

```
public function SubEditCasa($id,Request $request){
    if(session(key:'tipo_usuario')=="senhorio") {
        if (empty($request->input(key:'title')) || empty($request->input(key:'descricao')) || empty($request->input(key:'Tipo')) || empty($request->input(key:'sexo')) )
            return redirect()->back()->with('error','Por favor, preencha todos os campos obrigatórios.');
    }

    $token=session(key:'ActivationToken');
    $user = DB::table(table:'user')->where(column:'ActivationToken', $token)->first();
    DB::table(table:'casa_completa')->where(column:'id', $id)
        ->update([
            'Titulo' => $request->input(key:'title'),
            'description' => $request->input(key:'descricao'),
            'Tipo' => $request->input(key:'Tipo'),
            'Genero' => $request->input(key:'sexo'),
            'Preco' => $request->input(key:'preco'),
            'area' => $request->input(key:'area'),
            'estado' => 'pending',
            'id_user'=>$user->id,
        ]);

    DB::table(table:'cozinha')->where(column:'id_casa', $id)
        ->update([
            'Forno' => $request->has(key:'Forno') ? true : false,
            'Fogao' => $request->has(key:'Fogao') ? true : false,
            'Cadeirinha' => $request->has(key:'Cadeirinha') ? true : false,
            'Mac_cafe' => $request->has(key:'Mac_cafe') ? true : false,
            'Placa' => $request->has(key:'Placa') ? true : false,
            'Micro-ondas' => $request->has(key:'Micro-ondas') ? true : false,
            'Pratos' => $request->has(key:'Pratos') ? true : false,
            'Utensilios' => $request->has(key:'Utensilios') ? true : false,
            'Frigorifico' => $request->has(key:'Frigorifico') ? true : false,
        ]);

    DB::table(table:'banho')->where(column:'id_casa', $id)
        ->update([
            'Chuveiro' => $request->has(key:'Chuveiro') ? true : false,
            'Toalhas' => $request->has(key:'Toalhas') ? true : false,
        ]);
    DB::table(table:'contato')->where(column:'id_casa', $id)
        ->update([
            'Nome' => $request->input(key:'nome'),
            'Email' => $request->input(key:'email'),
            'Telefone' => $request->input(key:'title'),
        ]);
    DB::table(table:'endereco')->where(column:'id_casa', $id)
        ->update([
            'Endereco' => $request->input(key:'Endereco'),
            'N_andar' => $request->input(key:'N_andar'),
            'Codigo_postal' => $request->input(key:'Codigo_postal'),
            'Distancia' => $request->input(key:'Distancia'),
            'Lat' => $request->input(key:'lat'),
            'Lon' => $request->input(key:'lon'),
        ]);
    DB::table(table:'midia_de_casa')->where(column:'id_casa', $id)->delete();
    if (!is_null($request->file(key:'photos'))) {
        foreach ($request->file(key:'photos') as $photo) {
            $filename = $photo->store(path:'upload1', options:'public');
            DB::table(table:'midia_de_casa')
                ->insert([
                    'Path' => $filename,
                    'id_casa'=> $id
                ]);
        }
    }
}
```

```

foreach ($request->input('key:existing_photos') as $photo) {
    DB::table('midia_de_casa')
        ->insert([
            'Path' => $photo,
            'id_casa'=> $id
        ]);
}

DB::table('outros')->where('column: id_casa', $id)
->update([
    'Maquina_lavar_roupa' => $request->has('key: Maquina_lavar_roupa') ? true : false,
    'Maquina_sacar_roupa' => $request->has('key: Maquina_sacar_roupa') ? true : false,
    'Aquecimento_central' =>$request->has('key: Aquecimento_central') ? true : false,
    'passar_Ferro' => $request->has('key: passar_Ferro') ? true : false,
    'Aquecedor_eletrico' => $request->has('key: Aquecedor_eletrico') ? true : false,
]);
DB::table('sala')->where('column: id_casa', $id)
->update([
    'estar_partilhada' => $request->has('key: estar_partilhada') ? true : false,
    'Sofas' => $request->has('key: Sofas') ? true : false,
    'Televisao' => $request->has('key: Televisao') ? true : false,
    'Mesa_jantar' => $request->has('key: Mesa_jantar') ? true : false,
]);
DB::table('servicos')->where('column: id_casa', $id)
->update([
    'Wi-Fi' => $request->has('key: Wi-Fi') ? true : false,
    'Elevador' => $request->has('key: Elevador') ? true : false,
    'Despesas' => $request->has('key: Despesas') ? true : false,
    'Recibo' => $request->has('key: Recibo') ? true : false,
    'limpeza' => $request->has('key: limpeza') ? true : false,
]);
return redirect()->back()->with('success', 'A sua propriedade foi Editada com sucesso!');

```

Figure 33-submeter editar

Aqui está uma análise da função SubEditCasa(\$id, Request \$request):

SubEditCasa(\$id, Request \$request)

Verificação do Tipo de Utilizador:

Verifica se o tipo de utilizador na sessão é "senhorio". Se não for, aborta a solicitação com um código de status 403 (Acesso não autorizado).

Validação dos Campos Obrigatórios:

Verifica se os campos obrigatórios estão preenchidos. Se algum estiver vazio, redireciona de volta com uma mensagem de erro.

Atualização dos Detalhes da Casa Completa:

Recupera o utilizador atual com base no token de ativação da sessão.

Atualiza os detalhes da casa completa com o ID fornecido usando os dados fornecidos no formulário, incluindo título, descrição, tipo, género, preço, área e estado (definido como "pending").

Atualiza também o ID do utilizador associado à casa.

Atualização dos Detalhes da Cozinha, Casa de banho, Contrato, Endereço, Outros, Sala e Serviços:

Atualiza os detalhes da cozinha, casa de banho, contrato, endereço, outros, sala e serviços associados à casa completa com base nos dados fornecidos no formulário.

Atualização das Fotos da Casa:

Exclui todas as fotos associadas à casa com o ID fornecido.

Se novas fotos forem enviadas, as armazena e as associa à casa.

Também adiciona quaisquer fotos existentes selecionadas para permanecer.

Redireccionamento de Volta com Mensagem de Sucesso:

Redireciona de volta para a página anterior com uma mensagem de sucesso após a edição bem-sucedida da propriedade.

Essa função permite que os senhorios editem os detalhes de uma casa completa, incluindo as suas configurações de cozinha, casa de banho, contrato, endereço, outros, sala, serviços e fotos associadas.

## 3.17 Adicionar anúncio

### 3.17.1 FRONT-END

```
<div id="Porquarto" style="...>
  <form action="/Senhorio/Adicionar/AddQuarto" method="POST" enctype="multipart/form-data">
    @csrf
    <div class="submit_1 clearfix">
      <h4 class="mgt col_1">Descrição e preço do Quarto</h4>
      <hr>
      <h5>Título da propriedade</h5>
      <input name="title" class="form-control" placeholder="Property Title" type="text">
      <h5>Descrição da Propriedade</h5>
      <textarea name="descricao" placeholder="Property Description" class="form-control form_e"></textarea>
      <div class="col-sm-6 space_all">
        <div class="submit_111 clearfix">
          <h5>Tipo</h5>
          <select class="form-control" name="Tipo" required>
            <option value="Apartamento">Selecione Tipo</option>
            <option value="Casa">Casa</option>
            <option value="Apartamento">Apartamento</option>
            <option value="Estúdio">Estúdio</option>
          </select>
        </div>
      </div>
      <div class="col-sm-6 space_right">
        <div class="submit_111 clearfix">
          <h5>Género de residência</h5>
          <select class="form-control" name="sexo" required>
            <option value="M/F">Selecione Sexo</option>
            <option value="Masculino">Masculino</option>
            <option value="Feminino">Feminino</option>
            <option value="M/F">Masculino e Feminino</option>
          </select>
        </div>
      </div>
    </div>
  </form>
</div>
```

```
<div class="submit_1i clearfix">
    <div class="col-sm-6 space_left">
        <div class="submit_1i1 clearfix">
            <h5>Preço</h5>
            <input name="preco" class="form-control" placeholder="EUR" type="text" required>
        </div>
    </div>
    <div class="col-sm-6 space_right">
        <div class="submit_1i1 clearfix">
            <h5>Área M<sub>sub</sub>2 do quarto</h5>
            <input name="area" class="form-control" placeholder="m²" type="text" required>
        </div>
    </div>
</div>
<hr>
<br>
<h4><i class="fa fa-area-chart" aria-hidden="true"></i> características</h4>
<div class="submit_2i clearfix">
    <h5><input name="roupa_de_cama" type="checkbox"> <span class="span_1">roupa de cama</span></h5>
    <h5><input name="cama" type="checkbox"> <span class="span_1">cama</span></h5>
    <h5><input name="mesa_cabeceira" type="checkbox"> <span class="span_1">mesa cabeceira</span></h5>
    <h5><input name="Candeeiro_de_mesa_do_estudo" type="checkbox"> <span class="span_1">Candeeiro de mesa do estudo</span></h5>
    <h5><input name="Mesa_do_estudo" type="checkbox"> <span class="span_1">Mesa do estudo</span></h5>
    <h5><input name="Janelas" type="checkbox"> <span class="span_1">Janelas</span></h5>
    <h5><input name="Varanda" type="checkbox"> <span class="span_1">Varanda</span></h5>
    <h5><input name="Armario" type="checkbox"> <span class="span_1">Armário</span></h5>
    <h5><input name="Casa_de_banho_privativa" type="checkbox"> <span class="span_1">Casa de banho privativa</span></h5>
</div>
</div>
```

```
<h4 class="mgt col_1">Midia de propriedade</h4>
<hr>
<div class="submit_1i1 clearfix" id="uploadArea">
    <span class="span_1"><i class="fa fa-cloud-upload"></i></span>
    <h5>Clique aqui ou solte os arquivos para fazer upload</h5>
</div>
<input type="file" name="photos[]" id="fileInput" multiple="multiple" style="display: none;" required>
<hr>
<div id="fileListPorQuarto"></div>
</div>
<div class="submit_1i clearfix">
    <h4 class="mgt col_1">Localização da propriedade</h4>
    <hr>
    <div class="submit_1i1 clearfix">
        <div class="col-sm-6 space_left">
            <div class="submit_1i1 clearfix">
                <h5 class="mgt">Endereço</h5>
                <input name="Endereco" class="form-control" placeholder="Digite seu endereço" type="text" required>
            </div>
        </div>
        <div class="col-sm-6 space_right">
            <div class="submit_1i1 clearfix">
                <h5 class="mgt">Nº Andar</h5>
                <input name="N_andar" class="form-control" placeholder="Digite seu Nº Andar" type="text" required>
            </div>
        </div>
    </div>
    <br>
    <div class="submit_1i clearfix">
        <div class="col-sm-6 space_left">
            <div class="submit_1i1 clearfix">
                <h5 class="mgt">Código-postal</h5>
                <input name="Codigo_postal" class="form-control" placeholder="Digite seu Código-postal" type="text" required>
            </div>
        </div>
    </div>
</div>
```

```
<div class="submit_1 clearfix">
    <h4 class="mgt col_1">Informações de contato</h4>
    <hr>
    <div class="submit_1i clearfix">
        <div class="col-sm-6 space_left">
            <div class="submit_1ii clearfix">
                <h5 class="mgt">Nome</h5>
                <input name="nome" class="form-control" placeholder="Digite seu nome" type="text" required>
            </div>
        </div>
        <div class="col-sm-6 space_right">
            <div class="submit_1ii clearfix">
                <h5 class="mgt">Email</h5>
                <input name="email" class="form-control" placeholder="Digite seu e-mail" type="text" required>
            </div>
        </div>
    </div>
    <div class="submit_1i clearfix">
        <div class="col-sm-6 space_left">
            <div class="submit_1ii clearfix">
                <h5>Telefone</h5>
                <input name="tel" class="form-control" placeholder="Digite seu telefone" type="text" required>
            </div>
        </div>
    </div>
    <div class="submit_3 clearfix">
        <button style="..." type="submit" class="mgt"><a class="button mgt">Adicionar propriedade</a></button>
    </div>
</form>
</div>
```

Figure 34-Html adicionar anúncio

Este formulário parece ser destinado a adicionar um novo quarto a uma propriedade. Aqui está uma análise das seções do formulário:

#### Seção Descrição e Preço do Quarto

Campos para inserir o título e a descrição do quarto.

Seleção do tipo de propriedade (Casa, Apartamento, Estúdio).

Seleção do gênero da residência (Masculino, Feminino, M/F).

Inserção do preço do quarto em EUR.

Inserção da área do quarto em metros quadrados.

#### Seção Características do Quarto

Checkboxes para selecionar as características do quarto, como roupa de cama, cama, mesa de cabeceira, etc.

#### Seção Mídia de Propriedade

Área para fazer upload de fotos do quarto.

Botão para selecionar arquivos para upload.

#### Seção Localização da Propriedade

Inserção do endereço do quarto.

Inserção do número do andar.

Inserção do código postal.

Seleção da distância por meio de um mapa ou manualmente.

#### Seção Cozinha

Checkboxes para selecionar as características da cozinha, como forno, fogão, micro-ondas, etc.

#### Seção Sala

Checkboxes para selecionar as características da sala, como área de estar compartilhada, sofás, televisão, etc.

#### Seção Casa de Banho

Checkboxes para selecionar as características do casa de banho, como chuveiro, toalhas, etc.

#### Seção. Outros

Checkboxes para selecionar outras características, como máquina de lavar roupa, aquecimento central, etc.

#### Seção Serviços

Checkboxes para selecionar os serviços oferecidos, como Wi-Fi, elevador, limpeza, etc.

#### Seção Informações de Contato

Inserção do nome do contato.

Inserção do endereço de e-mail do contato.

Inserção do número de telefone do contato.

#### Botão de Submissão

Botão para enviar o formulário e adicionar a propriedade.

Parece ser um formulário abrangente para adicionar um novo quarto a uma propriedade, permitindo que os utilizadores forneçam detalhes específicos sobre o quarto e as suas características.

### 3.17.2 BACK-END

```

public function AddCasa(Request $request){
    if (empty($request->input('title')) || empty($request->input('descricao')) || empty($request->input('area')) {
        return redirect()->back()->with('error', 'Por favor, preencha todos os campos obrigatórios.');
    }
    if (is_null($request->file('photos'))) {
        return redirect()->back()->with('error', 'Por favor, adicione pelo menos uma foto.');
    }
    for ($i = 1; $i <= $request->input('n_quartos_cont'); $i++) {
        if (empty($request->input('area_'.$i))) {
            return redirect()->back()->with('error', 'Por favor, preencha a área de todos os quartos.');
        }
    }
    $token=session('ActivationToken');
    $user = DB::table('user')->where('column: ActivationToken', $token)->first();
    $currentDate = Carbon::now();
    $currentDate->addYear();
    $formattedDate = $currentDate->format('Y-m-d');
    $propertyId = DB::table('casa_completa')->insertGetId([
        'Titulo' => $request->input('title'),
        'description' => $request->input('descricao'),
        'Tipo' => $request->input('Tipo'),
        'Genero' => $request->input('sexo'),
        'Preco' => $request->input('preco'),
        'area' => $request->input('area'),
        'estado' => 'pending',
        'data_fim'=>$formattedDate,
        'id_user'=>$user->id,
        'N_quartos'=>$request->input('n_quartos'),
    ]);
}

for ($i = 1; $i <= $request->input('n_quartos_cont'); $i++) {
    DB::table('quartos_de_casa')->insert([
        'area_quarto' => $request->input('area_'.$i),
        'roupa_de_cama' => $request->has('roupa_de_cama_'.$i) ? true : false,
        'cama' => $request->has('cama_'.$i) ? true : false,
        'mesa_cabeceira' => $request->has('mesa_cabeceira_'.$i) ? true : false,
        'Candeeiro_de_mesa_do_estudo' => $request->has('Candeeiro_de_mesa_do_estudo_'.$i) ? true : false,
        'Mesa_do_estudo' => $request->has('Mesa_do_estudo_'.$i) ? true : false,
        'Janelas' => $request->has('Janelas_'.$i) ? true : false,
        'Varanda' => $request->has('Varanda_'.$i) ? true : false,
        'Armario' => $request->has('Armario_'.$i) ? true : false,
        'Casa_de_banho_privativa' => $request->has('Casa_de_banho_privativa_'.$i) ? true : false,
        'id_casa' => $propertyId,
    ]);
}

DB::table('cozinha')->insert([
    'Forno' => $request->has('Forno') ? true : false,
    'Fogao' => $request->has('Fogao') ? true : false,
    'Caldeira' => $request->has('Caldeira') ? true : false,
    'Mag_cafe' => $request->has('Mag_cafe') ? true : false,
    'Placa' => $request->has('Placa') ? true : false,
    'Micro-ondas' => $request->has('Micro-ondas') ? true : false,
    'Pratos' => $request->has('Pratos') ? true : false,
    'Utensilios' => $request->has('Utensilios') ? true : false,
    'Frigorifico' => $request->has('Frigorifico') ? true : false,
    'id_casa' => $propertyId,
]);

```

```
DB::table( table: 'banho')->insert([
    'Chuveiro' => $request->has( key: 'Chuveiro') ? true : false,
    'Toalhas' => $request->has( key: 'Toalhas') ? true : false,
    'id_casa' => $propertyId,
]);
DB::table( table: 'contato')->insert([
    'Nome' => $request->input( key: 'nome'),
    'Email' => $request->input( key: 'email'),
    'Telefone' => $request->input( key: 'title'),
    'id_casa' => $propertyId,
]);
DB::table( table: 'endereco')->insert([
    'Endereco' => $request->input( key: 'Endereco'),
    'N_andar' => $request->input( key: 'N_andar'),
    'Codigo_postal' => $request->input( key: 'Codigo_postal'),
    'Distancia' => $request->input( key: 'Distancia'),
    'let' => $request->input( key: 'letLag'),
    'id_casa' => $propertyId,
]);
foreach ($request->file( key: 'photos') as $photo) {
    $filename = $photo->store( path: 'upload', options: 'public');
    DB::table( table: 'midia_de_casa')->insert([
        'Path' => $filename,
        'id_casa' => $propertyId,
    ]);
}
```

```
DB::table('outros')->insert([
    'Maquina_lavar_roupa' => $request->has('Maquina_lavar_roupa') ? true : false,
    'Maquina_sacar_roupa' => $request->has('Maquina_sacar_roupa') ? true : false,
    'Aquecimento_central' => $request->has('Aquecimento_central') ? true : false,
    'passar_Ferro' => $request->has('passar_Ferro') ? true : false,
    'Aquecedor_eletrico' => $request->has('Aquecedor_eletrico') ? true : false,
    'id_casa' => $propertyId,
]);
DB::table('sala')->insert([
    'estar_partilhada' => $request->has('estar_partilhada') ? true : false,
    'Sofas' => $request->has('Sofas') ? true : false,
    'Televisao' => $request->has('Televisao') ? true : false,
    'Mesa_jantar' => $request->has('Mesa_jantar') ? true : false,
    'id_casa' => $propertyId,
]);
DB::table('servicos')->insert([
    'Wi-Fi' => $request->has('Wi-Fi') ? true : false,
    'Elevador' => $request->has('Elevador') ? true : false,
    'Despesas' => $request->has('Despesas') ? true : false,
    'Recibo' => $request->has('Recibo') ? true : false,
    'limpeza' => $request->has('limpeza') ? true : false,
    'id_casa' => $propertyId,
]);
return redirect()->back()->with('success', 'A sua propriedade foi adicionada com sucesso!');
```

Figure 35-controlar adicionar anúncio

Este método AddCasa parece lidar com a adição de uma nova propriedade (casa). Aqui está uma análise passo a passo do que acontece no método:

**Validação de Campos Obrigatórios:** Verifica se todos os campos obrigatórios do formulário são preenchidos. Se algum estiver vazio, redireciona de volta com uma mensagem de erro.

**Verificação de Fotos:** Verifica se pelo menos uma foto foi enviada. Se não houver nenhuma foto, redireciona de volta com uma mensagem de erro.

**“Loop” para Adicionar Quartos:**

Itera sobre o número de quartos fornecidos no formulário.

Insere os detalhes de cada quarto na tabela quartos\_de\_casa.

**Inserção de Detalhes da Casa:**

Insere os detalhes principais da casa na tabela casa\_completa.

Insere os detalhes da cozinha na tabela cozinha.

Insere os detalhes da casa de banho na tabela banho.

Insere os detalhes de contacto na tabela contacto.

Insere os detalhes de endereço na tabela endereço.

Insere as fotos na tabela midia\_de\_casa.

Insere os outros detalhes na tabela outros.

Insere os detalhes da sala na tabela sala.

Insere os serviços na tabela servicos.

Redirecionamento: Finalmente, redireciona de volta à página anterior com uma mensagem de sucesso.

Este método parece ser bastante abrangente, lidando com a inserção de múltiplos detalhes de quartos, características da casa, informações de contacto, serviços e muito mais.

## 3.18 ControllerChat

A classe ControllerChat é uma classe controladora que contém métodos para lidar com a lógica de negócios relacionada a chats entre alunos e senhorios.

### 3.18.1 MÉTODOS

**CriarChatQuarto(\$id):** Este método é responsável por criar ou obter um chat relacionado a um quarto específico. Ele verifica se o utilizador é um aluno, obtém informações do utilizador atualmente autenticado, verifica se já existe um chat para o quarto especificado e, em seguida, redireciona para a página de conversa com os dados necessários.

**CriarChatCasa(\$id):** Similar ao método anterior, mas para criar ou obter um chat relacionado a uma casa completa.

**GetChat(\$id):** Este método é usado para obter as mensagens de um chat específico. Ele verifica se o utilizador é um aluno ou senhorio, obtém informações sobre o chat e as mensagens relacionadas a ele e atualiza o estado do chat para indicar que foi visualizado.

**sendMessage(Request \$request, \$id):** Este método é responsável por enviar uma mensagem para um chat específico. Ele recebe a mensagem do formulário enviado pelo utilizador, insere a mensagem no banco de dados e redireciona de volta para a página de conversa.

## 2.18.2 BACK END

```
$verfiy=DB::table( table: 'chat')
->where( column: 'id_aluno', $userqu->id)
->where( column: 'id_senhorio', $idsenh->id_user)
->first();
if($verfiy){
    $conversas = DB::table( table: 'chat')
        ->join( table: 'user', first: 'user.id', operator: '=', second: 'chat.id_senhorio')
        ->orderBy( column: 'chat.id', direction: 'desc')
        ->select( columns: 'chat.*','chat.id as idchat','user.id as userid','user.*')
        ->where( column: 'id_aluno', $userqu->id)->get();
    $con = DB::table( table: 'chat')
        ->join( table: 'user', first: 'user.id', operator: '=', second: 'chat.id_senhorio')
        ->where( column: 'chat.id_aluno', $userqu->id)
        ->where( column: 'chat.id', $verfiy->id)
        ->orderBy( column: 'chat.id', direction: 'desc')
        ->select( columns: 'chat.*','chat.id as idchat','user.*')
        ->first();
    $sinal= DB::table( table: 'chat')
        ->join( table: 'chat_senhorio', first: 'chat_senhorio.id_chat', operator: '=', second: 'chat.id')
        ->join( table: 'chat_aluno', first: 'chat_aluno.id_chat', operator: '=', second: 'chat.id')
        ->join( table: 'user', first: 'user.id', operator: '=', second: 'chat.id_senhorio')
        ->select( columns: 'user.*','chat.*','chat.id as idchat',
            'chat_senhorio.masseg as senhmasseg','chat_aluno.masseg as alunomasseg'
            , 'chat_senhorio.Estado as massestado','chat_senhorio.*')
        ->limit( value: 1)
        ->where( column: 'chat.id', $con->idchat)->get();

$mensagens_combinadas = DB::table(function ($query) use ($con) {
    $query->select(DB::raw( value: "'aluno' AS origem", 'id AS id_origem', 'id_chat', 'TimeAlun as time_envio', 'masseg as alu', 'Estado as estado')
        ->from('chat_aluno')
        ->where('id_chat', $con->idchat)
        ->unionAll(
            DB::table('chat_senhorio')
                ->select(DB::raw( value: "'senhorio' AS origem", 'id AS id_origem', 'id_chat', 'TimeSenh as time_envio', 'masseg as sen', 'Estado as estado')
                ->where( column: 'id_chat', $con->idchat)
        );
}, as: 'mensagens_combinadas')
->orderBy( column: 'time_envio', direction: 'ASC')
->get();

return view( view: 'Page_aluno\conversation', [ 'Data'=>$user, 'chats'=>$conversas, 'chat'=>$mensagens_combinadas, 'sinal'=>$sinal]);
```

Figure 36-Chat

**\$verfiy :** Esta parte do código realiza uma consulta à tabela de chats para verificar se já existe um chat entre o aluno (id\_aluno) e o senhorio (id\_senhorio) específicos. Ele usa o objeto DB do Laravel para interagir com o banco de dados e a função first() para obter apenas o primeiro resultado da consulta.

**\$conversas:** Esta parte da lógica é para recuperar todas as conversas associadas ao aluno logado. Realiza uma junção com a tabela user para obter informações sobre o senhorio, seleciona os campos relevantes e filtra pela identificação do aluno.

Aqui, é feita uma subconsulta para combinar todas as mensagens relacionadas ao chat em uma única coleção de mensagens. São selecionados os campos relevantes e as mensagens são ordenadas por tempo de envio.

**\$mensagens\_combinadas:** aqui, é feita uma subconsulta para combinar todas as mensagens relacionadas ao chat numa única coleção de mensagens. São selecionados os campos relevantes e as mensagens são ordenadas por tempo de envio.

```

public function sendMessage(Request $request,$id){
    if(session( key: 'tipo_usuario')=="aluno"){
        $message = $request->input( key: 'message');
        DB::table( table: 'chat_aluno')->insert([
            'id_chat'=> $id,
            'massag' =>$message,
            'Estado'=>'0',
            'TimeAlun'=>Carbon::now()
        ]);
        return redirect()->back()->with('success' , 'Casa adicionada a Propriedade com sucesso');
    }else {
        abort( code: 403, message: 'Acesso não autorizado.');
    }
}

```

Figure 37-enviar chat

Esta parte do código insere a mensagem no banco de dados. A mensagem é associada ao chat específico através do campo id\_chat. O campo Estado é definido como '0' para indicar que a mensagem ainda não foi vista. O campo TimeAlun recebe a data e hora atual.

## 3.19 Adicionar para os favoritos

### 3.19.1 BACK END

```

public function AddfevorQuarto($id){
    if(session( key: 'tipo_usuario')=="aluno"){
        $token=session( key: 'ActivationToken');
        $userqu = DB::table( table: 'user')->where( column: 'ActivationToken', $token)->first();
        DB::table( table: 'feveritos')->insert([
            'id_user'=>$userqu->id,
            'id_quarto'=>$id
        ]);
        return redirect()->back()->with('success' , 'Casa adicionada a Propriedade com sucesso');
    }else{
        return redirect( to: '/Login');
    }
}

```

Figure 38- favoritos

Este método é responsável por adicionar um quarto aos favoritos de um utilizador. Ele primeiro verifica se o utilizador está autenticado como um aluno, obtém o utilizador correspondente ao token de ativação na sessão e, em seguida, insere um novo registo na tabela de favoritos associando o utilizador ao quarto específico. Se o utilizador não estiver autenticado como aluno, ele será redirecionado para a página de login.