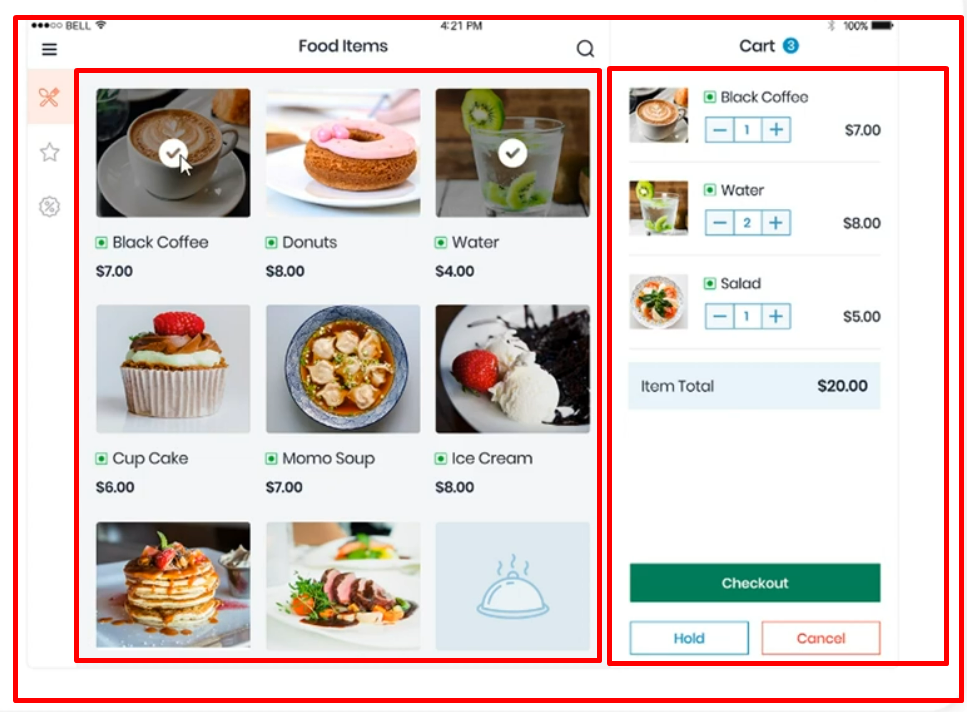
**Ponto de venda on-line**

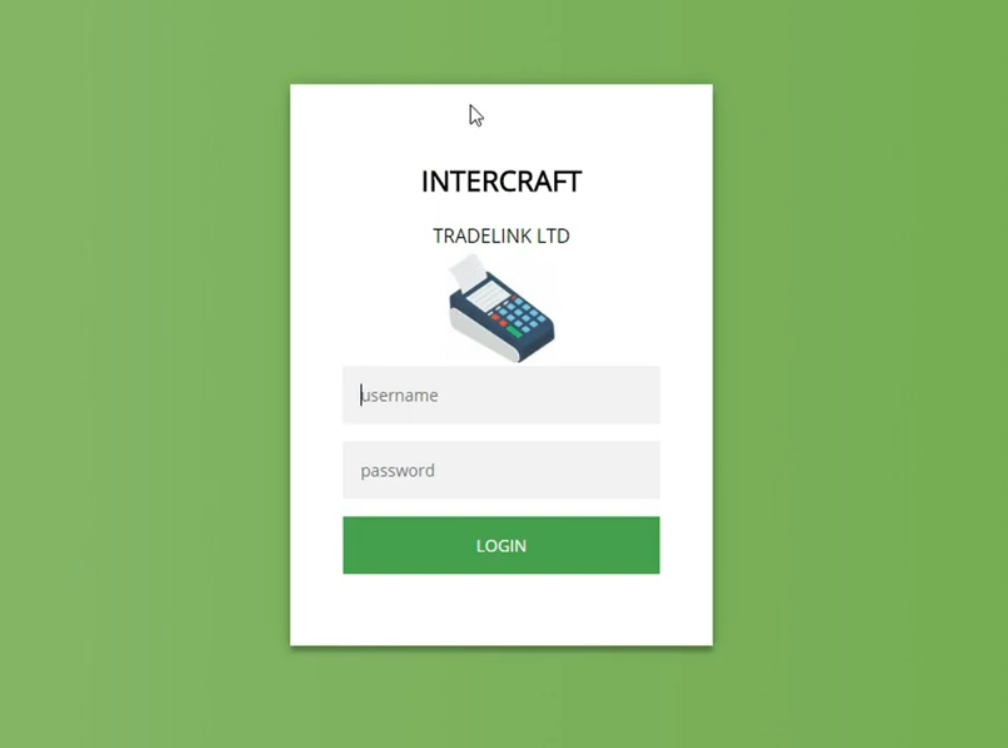
Manual de instrução e código incorporado

**INTRODUÇÃO**

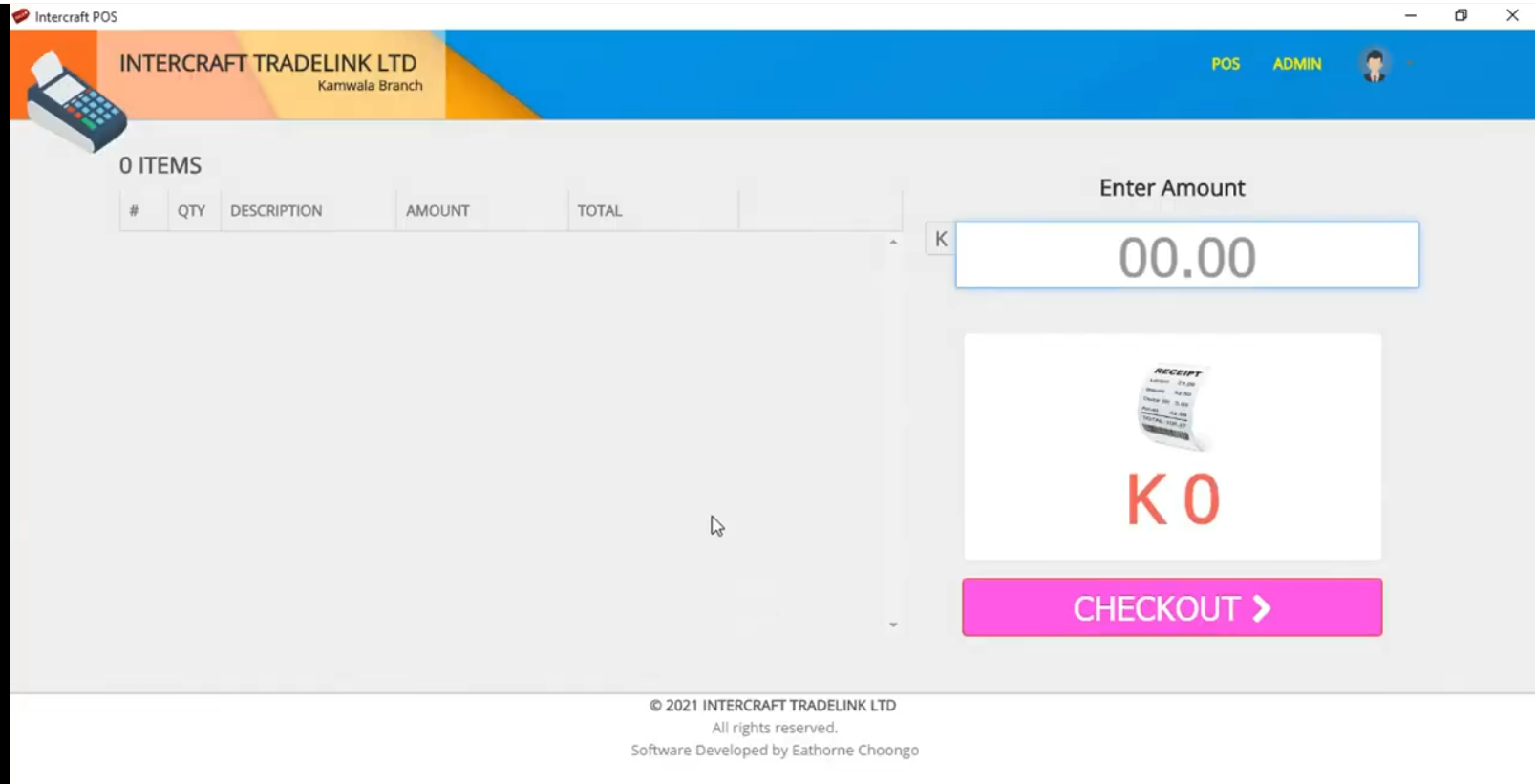
Este sistema foi desenvolvido na linguaguem **PHP** e **Javascript** afim de atender restaurantes e hoteis de forma rápida e simplificada. Na UI conforme é ilustrada abaixo, o cliente poderá visualizar o produto que na qual pretendera adquirir e sistematicamente é adicionado na lista de compras na coluna a direita, toda vez que for clicado no produto será contabilizado no painel do quadro a direita.



Teremos um layout para atender o login de entrada para que os utilizadores possam aceder ao sistema e acederem com as suas respectivas contas.

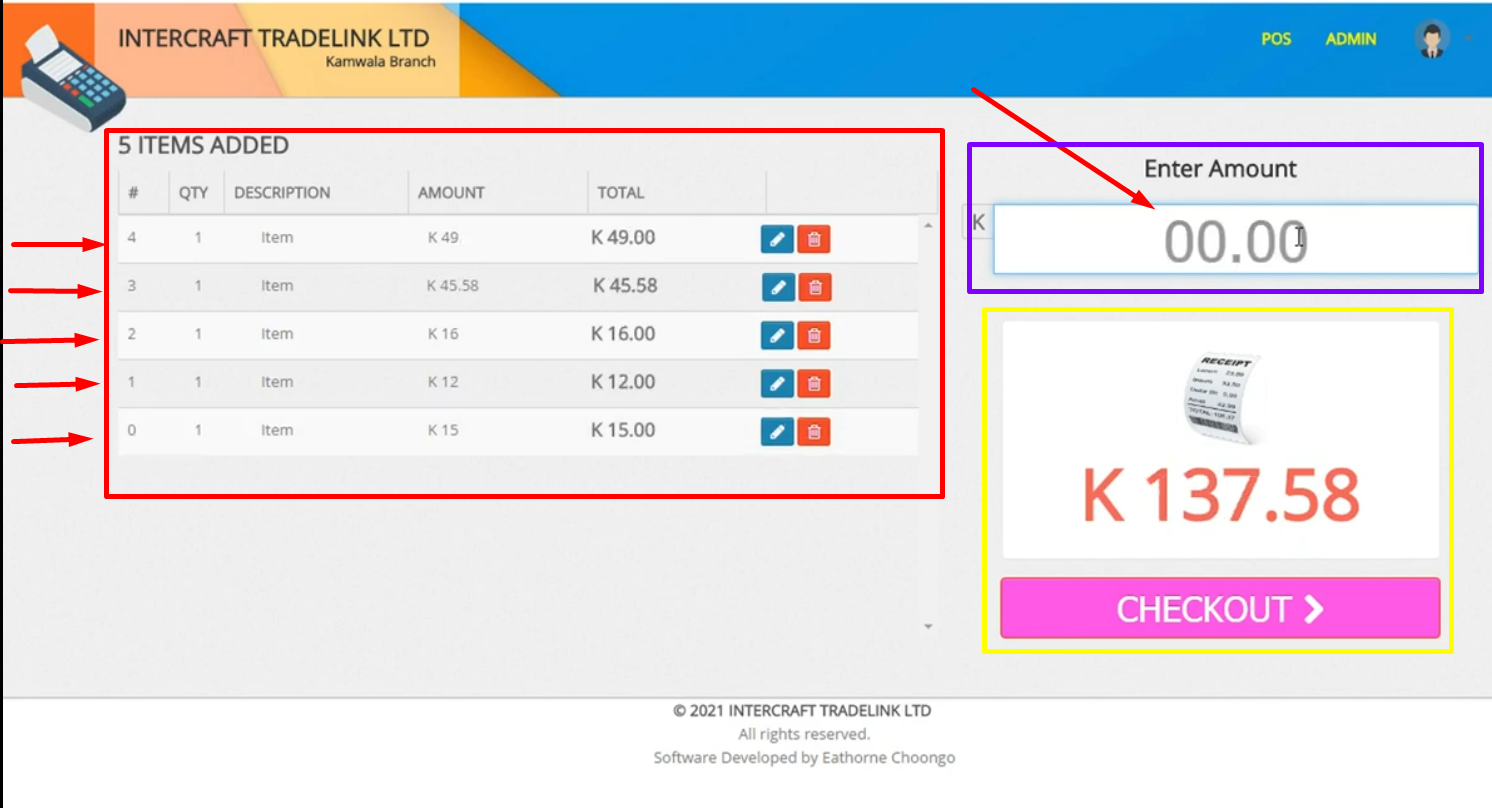


Apóis efetuar o login, caso as suas credencias esteja validada na **BD** então lhe é permitido visualizar o 2º formulario layout do sistema onde lhe será visivel os pedidos de entrada e saída dos produtos.

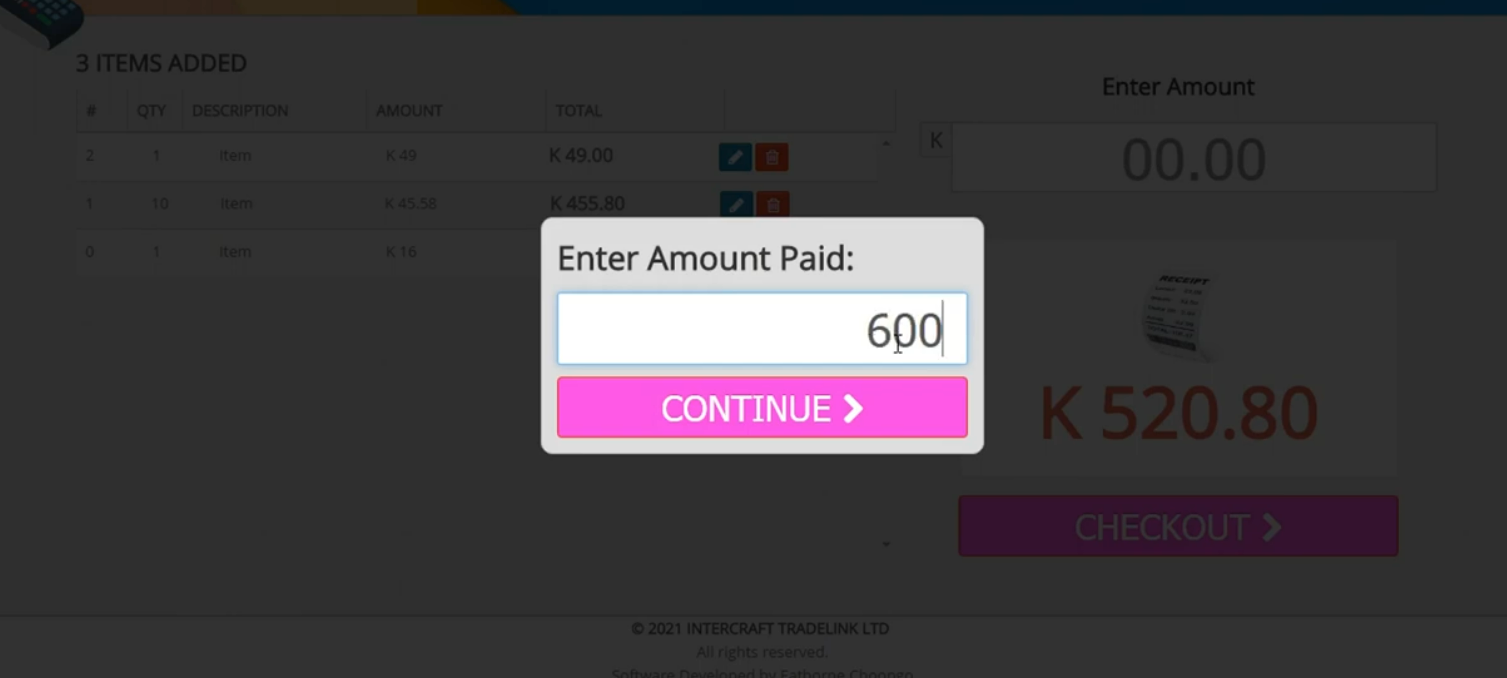


Por cada registro lhe será apresendao os itens selecionados para a sua compra e no final é apresentado o total aguardando a sua confirmação para o fecho da compra dos produtos.

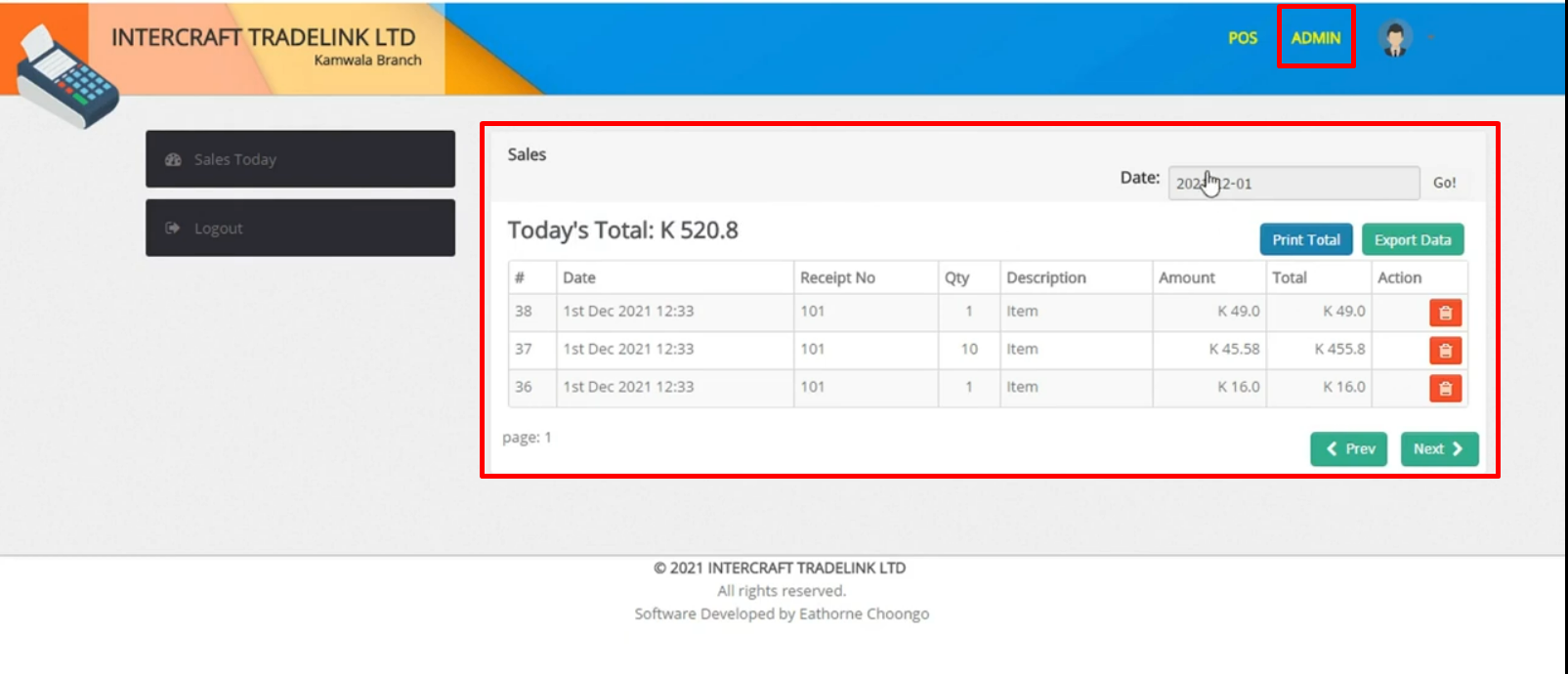
**Atenção**: antes de finalizar a compra o utilizador pode editar ou excluir um produto que esteja incorreto ou não queira adquirir.



O montante de pagamento lhe será pedido no final afim de se efetivar a compra dos produtos e lhe será emitido uma fatura comprovante para futuras reclamações ou mesmo o asseguramento da loja em beneficios fiscais.



No formulario **admin** é visualizado todas as vendas geradas em tempo real em todas as lojas instaladas o sistema **PDV**. Termos neste formulario layout os menus de: Exportação, impressão de fatura, data de filtro, exclusão, avançar e recuar paginas.



**O QUE VAMOS APRENDER**?

* PHP DESKTOP
* PHP
* CONTROLO DE ACESSO
* SQLITE
* JAVASCRIPT
* AJAX
* CHARTS
* BOOTSTRAP CSS

**PHP DESKTOP**

O PHP Desktop é uma aplicação open-source que permite executar aplicações web PHP como aplicações desktop nativas. Sua finalidade principal é simplificar o desenvolvimento de aplicações desktop usando a tecnologia web que os desenvolvedores já conhecem, como PHP, HTML, JavaScript e CSS. Aqui estão alguns pontos sobre suas funcionalidades e usos:

1. **Desenvolvimento de Aplicações Desktop**: Permite criar aplicações desktop usando PHP, sem a necessidade de aprender novas linguagens ou frameworks específicos para desktop.
2. **Ambiente Integrado**: PHP Desktop integra um servidor web (baseado em Chromium) e um interpretador PHP, facilitando o desenvolvimento e a distribuição de aplicações desktop.
3. **Interface Web para Desktop**: Utiliza tecnologias web para criar a interface do usuário, o que possibilita o uso de frameworks de frontend como Angular, React, Vue.js, entre outros.
4. **Distribuição Simples**: As aplicações podem ser distribuídas como executáveis autônomos, simplificando a instalação e uso pelo usuário final.
5. **Acesso a Recursos Locais**: Permite acessar recursos locais do sistema operacional, como o sistema de arquivos e dispositivos de hardware, através de extensões PHP e APIs JavaScript.

Em resumo, o PHP Desktop facilita a transição de desenvolvedores web para o desenvolvimento de aplicações desktop, reutilizando conhecimentos e tecnologias já dominadas.

**PHP**

O PHP (Hypertext Preprocessor) é uma linguagem de programação de uso geral voltada principalmente para o desenvolvimento web. Sua finalidade principal é permitir a criação de sites dinâmicos e interativos. Aqui estão algumas das principais funcionalidades e usos do PHP:

1. **Desenvolvimento de Sites Dinâmicos**: PHP permite criar páginas web que interagem com o usuário e atualizam seu conteúdo de maneira dinâmica, em vez de simplesmente exibir conteúdo estático.
2. **Interação com Bancos de Dados**: PHP oferece suporte para uma ampla variedade de bancos de dados, como MySQL, PostgreSQL, Oracle, entre outros, facilitando a criação de aplicações web que armazenam e recuperam dados.
3. **Geração de Páginas HTML**: PHP é embutido em código HTML e pode gerar HTML dinamicamente, permitindo personalizar páginas web com base em dados ou preferências do usuário.
4. **Formulários e Processamento de Dados**: PHP é amplamente utilizado para processar dados enviados por formulários, validar entradas do usuário e interagir com serviços web.
5. **Desenvolvimento de APIs**: Com PHP, é possível desenvolver APIs RESTful, permitindo a comunicação entre diferentes aplicações e serviços.
6. **Autenticação e Controle de Acesso**: PHP fornece ferramentas para implementar sistemas de autenticação de usuários, gerenciamento de sessões e controle de acesso a recursos protegidos.
7. **Manipulação de Arquivos**: PHP permite criar, abrir, ler, escrever e manipular arquivos no servidor, o que é útil para tarefas como upload de arquivos, geração de relatórios e armazenamento de logs.
8. **Integração com Outras Tecnologias**: PHP pode ser integrado com diversas tecnologias e serviços, como sistemas de email, bibliotecas de gráficos, bibliotecas de processamento de imagens, entre outros.
9. **Desenvolvimento de CMSs**: Muitos sistemas de gerenciamento de conteúdo (CMS), como WordPress, Joomla e Drupal, são escritos em PHP, permitindo a criação e gerenciamento de sites de maneira eficiente e flexível.
10. **Comunidade e Suporte**: PHP possui uma grande comunidade de desenvolvedores e uma vasta quantidade de recursos, como documentação, tutoriais, frameworks (como Laravel, Symfony) e bibliotecas, que facilitam o desenvolvimento e resolução de problemas.

Em resumo, PHP é uma linguagem poderosa e versátil para o desenvolvimento web, oferecendo uma ampla gama de funcionalidades que suportam a criação de aplicações web complexas e interativas.

**CONTROLO DE ACESSO**

O controle de acesso na programação PHP é um conjunto de técnicas e práticas que garantem que apenas usuários ou processos autorizados possam acessar certos recursos ou executar certas ações em uma aplicação web. Aqui estão algumas das finalidades e benefícios do controle de acesso em PHP:

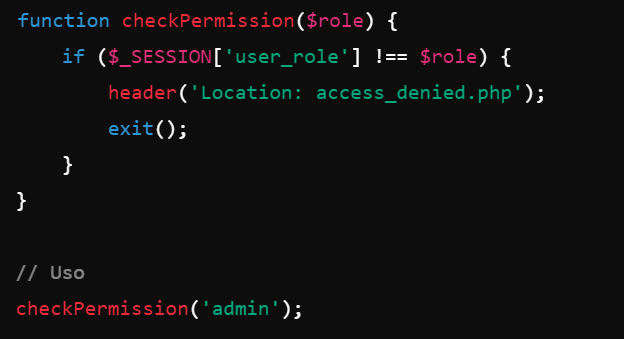
1. **Segurança**: Previne acessos não autorizados a informações sensíveis, funcionalidades restritas e dados privados. Isso ajuda a proteger a aplicação contra invasões e uso indevido.
2. **Proteção de Dados**: Garante que dados confidenciais, como informações de usuários e registros financeiros, sejam acessíveis apenas a pessoas autorizadas, protegendo-os contra roubo e exposição.
3. **Integridade do Sistema**: Restringe quem pode modificar dados e configurações do sistema, evitando alterações maliciosas ou não intencionais que possam comprometer a integridade da aplicação.
4. **Gerenciamento de Usuários**: Permite diferenciar os níveis de acesso entre diferentes tipos de usuários, como administradores, moderadores e usuários comuns. Isso ajuda a definir claramente as permissões e responsabilidades de cada grupo.
5. **Conformidade com Normas e Regulamentações**: Muitas regulamentações de segurança de dados (como GDPR, HIPAA, PCI-DSS) exigem mecanismos de controle de acesso para garantir a privacidade e segurança dos dados.
6. **Experiência do Usuário**: Melhora a experiência do usuário ao apresentar apenas os recursos e informações relevantes para seu nível de acesso, evitando confusão e sobrecarga de informações.

**Exemplos de Controle de Acesso em PHP**

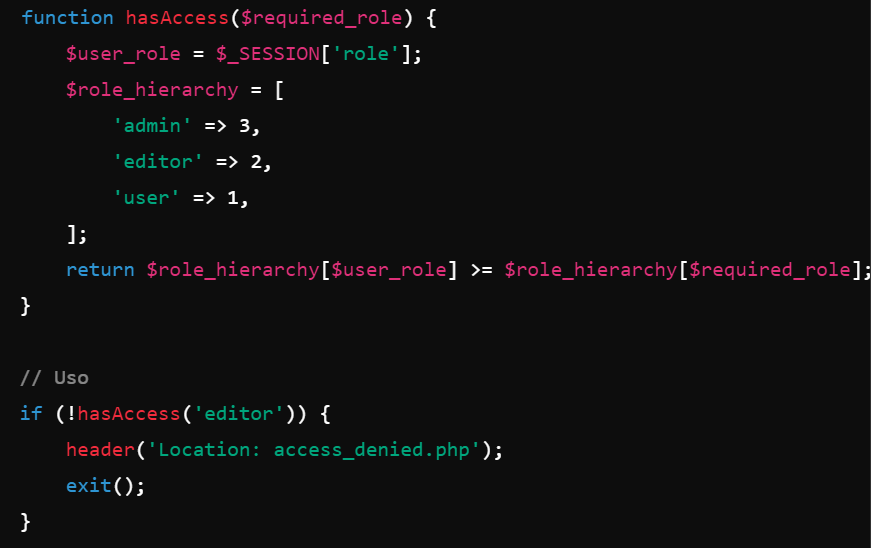
1. **Autenticação de Usuário**:
   * Verificação de credenciais de login (usuário e senha) para garantir que apenas usuários registrados possam acessar a aplicação.
2. **Gerenciamento de Sessões**:
   * Uso de sessões PHP para manter o estado do login do usuário e verificar suas permissões em diferentes partes da aplicação.



1. **Autorização**:
   * Definição de permissões específicas para diferentes níveis de usuário. Por exemplo, somente administradores podem acessar páginas de administração e realizar operações críticas.



1. **Controle Baseado em Funções (RBAC)**:
   * Implementação de um sistema de controle de acesso baseado em funções, onde os usuários são atribuídos a funções específicas, e cada função tem um conjunto de permissões associadas.



1. **Controle Baseado em Atributos (ABAC)**:
   * Utilização de atributos específicos (como tempo de acesso, localização, tipo de dispositivo) para tomar decisões de acesso mais granulares.

Em resumo, o controle de acesso é essencial para garantir a segurança, integridade e usabilidade de uma aplicação PHP, protegendo tanto os dados quanto a experiência do usuário.

**SQLITE**

O SQLite é um banco de dados embutido que é amplamente usado em PHP por várias razões, devido à sua simplicidade e eficiência. Aqui estão algumas das principais finalidades do SQLite no PHP:

1. **Banco de Dados Local e Leve**: O SQLite é um banco de dados leve e embutido que não requer um servidor de banco de dados separado. Isso o torna ideal para aplicações locais, desenvolvimento e pequenos projetos web.
2. **Fácil Configuração**: Diferente de outros sistemas de gerenciamento de banco de dados, como MySQL ou PostgreSQL, o SQLite não requer configuração complexa. Basta incluir o arquivo de banco de dados no seu projeto.
3. **Alta Performance para Aplicações Pequenas a Médias**: Para aplicações que não necessitam de um banco de dados distribuído ou extremamente escalável, o SQLite oferece excelente performance com menor overhead.
4. **Desenvolvimento e Teste**: Durante o desenvolvimento de aplicações PHP, o SQLite é muito útil para testes rápidos e desenvolvimento local, sem a necessidade de instalar e configurar um banco de dados completo.
5. **Aplicações Mobile e Desktop**: Devido à sua natureza leve, o SQLite é amplamente usado em aplicativos mobile e desktop que precisam de um banco de dados local.
6. **Portabilidade**: O banco de dados SQLite é armazenado em um único arquivo, o que facilita a portabilidade de dados entre diferentes ambientes e sistemas.
7. **Compliant com ACID**: O SQLite suporta transações ACID (Atomicidade, Consistência, Isolamento, Durabilidade), garantindo a integridade dos dados, mesmo em casos de falhas.
8. **Suporte a Recursos SQL**: Ele oferece suporte a muitos recursos SQL padrão, permitindo que você use a maior parte do seu conhecimento SQL sem muitas modificações.

Para utilizar o SQLite em PHP, você pode usar a extensão PDO (PHP Data Objects) com SQLite.

**JAVASCRIPT**

O JavaScript e o PHP são linguagens de programação usadas frequentemente juntas no desenvolvimento web, mas servem a propósitos diferentes. Aqui estão as principais finalidades do JavaScript no contexto de uma aplicação PHP:

1. **Interatividade e Dinamismo no Lado do Cliente**: JavaScript é usado para adicionar interatividade e dinamismo às páginas web. Ele permite a criação de elementos interativos, como menus dropdown, sliders de imagem, validação de formulários em tempo real, etc., que são executados no navegador do usuário sem a necessidade de recarregar a página.
2. **Manipulação do DOM**: JavaScript pode manipular o Document Object Model (DOM) da página web, permitindo a adição, remoção ou modificação de elementos HTML dinamicamente, sem a necessidade de interação com o servidor.
3. **Comunicação Assíncrona (AJAX)**: JavaScript é fundamental para implementar AJAX (Asynchronous JavaScript and XML), que permite a comunicação assíncrona com o servidor. Isso significa que é possível enviar e receber dados do servidor em segundo plano sem recarregar a página. Em uma aplicação PHP, isso é frequentemente usado para buscar dados de uma API, enviar formulários, ou atualizar partes da página de forma dinâmica.
4. **Melhoria da Experiência do Usuário (UX)**: Ao permitir atualizações de página rápidas e respostas interativas, JavaScript melhora significativamente a experiência do usuário. Ele pode proporcionar feedback imediato, animações suaves, e interfaces de usuário (UI) mais intuitivas e responsivas.
5. **Validação do Lado do Cliente**: Embora a validação do lado do servidor (usando PHP) seja essencial para a segurança, a validação do lado do cliente (usando JavaScript) melhora a usabilidade ao fornecer feedback imediato aos usuários, evitando a necessidade de submeter formulários com dados inválidos e esperar uma resposta do servidor.
6. **Desenvolvimento de Aplicações de Página Única (SPA)**: Com frameworks e bibliotecas JavaScript modernas como React, Angular, e Vue.js, é possível criar aplicações de página única que carregam uma única página HTML e dinamicamente atualizam o conteúdo conforme necessário, enquanto o backend PHP fornece os dados e lógica de negócio.

**Explicação**

* **HTML e JavaScript**: A página HTML contém um botão que, quando clicado, chama a função **loadData().** Esta função usa XMLHttpRequest para fazer uma solicitação AJAX ao servidor PHP.
* **PHP**: O script server.php retorna uma resposta simples com a data e hora atuais, que é então exibida na div com o id content.

Este exemplo básico demonstra como usar JavaScript para fazer uma solicitação AJAX a um script PHP, permitindo a atualização dinâmica do conteúdo da página sem recarregar.

**Nota:**

JavaScript é essencial para criar uma experiência de usuário rica e interativa em aplicações web PHP, permitindo manipulação de DOM, validação do lado do cliente, comunicação assíncrona e muito mais. Enquanto PHP lida com a lógica de negócios e interações com o banco de dados no servidor, JavaScript torna a interface do usuário mais dinâmica e responsiva.

**AJAX**

AJAX (Asynchronous **JavaScript** and **XML**) é uma técnica de desenvolvimento web que permite a atualização assíncrona de partes de uma página da web sem precisar recarregar a página inteira. Em PHP, AJAX é frequentemente utilizado para:

1. **Atualização Dinâmica de Conteúdo**: Permite que partes da página sejam atualizadas sem a necessidade de recarregar a página completa. Por exemplo, você pode carregar novos dados de um banco de dados e atualizar uma lista ou tabela na página.
2. **Formulários Dinâmicos**: Pode ser usado para enviar dados de formulários para o servidor e receber respostas sem precisar enviar o formulário completo, o que melhora a experiência do usuário.
3. **Validação de Dados em Tempo Real**: Permite que a validação de dados do lado do servidor seja realizada enquanto o usuário está preenchendo um formulário, oferecendo feedback imediato sem precisar submeter o formulário.
4. **Carregamento Assíncrono de Dados**: Carrega dados adicionais conforme necessário, por exemplo, em um site de notícias que carrega mais artigos conforme o usuário rola para baixo.
5. **Interação em Tempo Real**: Facilita interações em tempo real, como chat ao vivo ou atualizações de status de atividades em um painel de controle.

AJAX usa JavaScript para enviar requisições HTTP assíncronas ao servidor e PHP para processar essas requisições e retornar os dados desejados, que são então manipulados no lado do cliente e atualizados dinamicamente na interface do usuário.

**CHARTS**

**Finalidade**: Gráficos e visualizações de dados.

CHARTS se refere geralmente a bibliotecas ou ferramentas que permitem a criação de gráficos e representações visuais de dados em páginas web. No PHP, isso pode ser integrado através de bibliotecas específicas ou APIs de terceiros que geram gráficos a partir de dados. Exemplos de bibliotecas populares incluem Chart.js, Highcharts e Google Charts. Essas bibliotecas permitem criar gráficos de barras, linhas, pizza, etc., para ajudar a visualizar dados de maneira mais intuitiva.

**Bootstrap CSS**

**Finalidade**: Design responsivo e estilização.

Bootstrap é um framework front-end popular que fornece uma série de estilos e componentes pré-definidos para construir interfaces de usuário responsivas e atraentes. Ele inclui CSS e JavaScript para criar layouts flexíveis, botões, formulários, barras de navegação e muito mais. No PHP, Bootstrap pode ser usado para estilizar as páginas web geradas pelos scripts PHP, melhorando a aparência e a funcionalidade das aplicações web sem a necessidade de escrever CSS do zero.

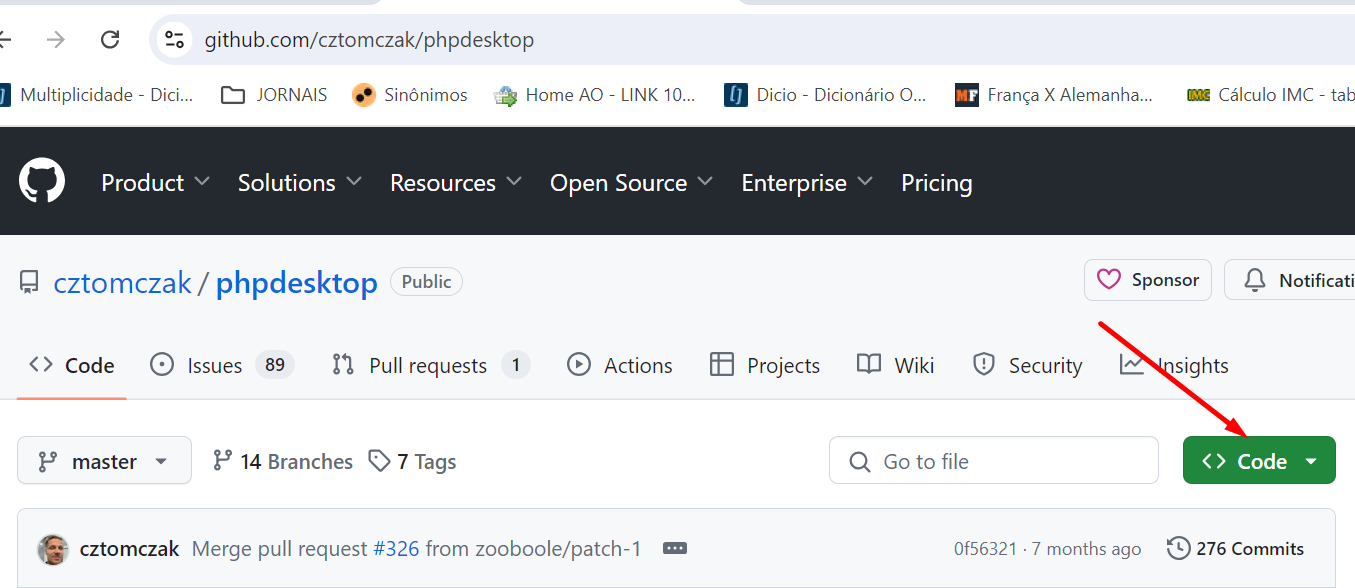
**RECOMENDAÇÕES**

Este manual é recomendavel para todas as pessoas de diversas áreas do saber pois não basta apenas ler quem deseja aprender a programar, mesmo aqueles que interagem diretamente com os sistemas pois é fundamental ter o conhecimento por detrás da ferramenta, assim como no cimena o ator deve saber o seu posicionamento para que a camera possa captar as emoções ou a imagem do belo nos diversos angulos da sua faceta, assim é com o usuario comum tem um computador.

**PREPARAÇÃO**

1. Baixar o PHP Desktop o seu zip file

<https://github.com/cztomczak/phpdesktop>



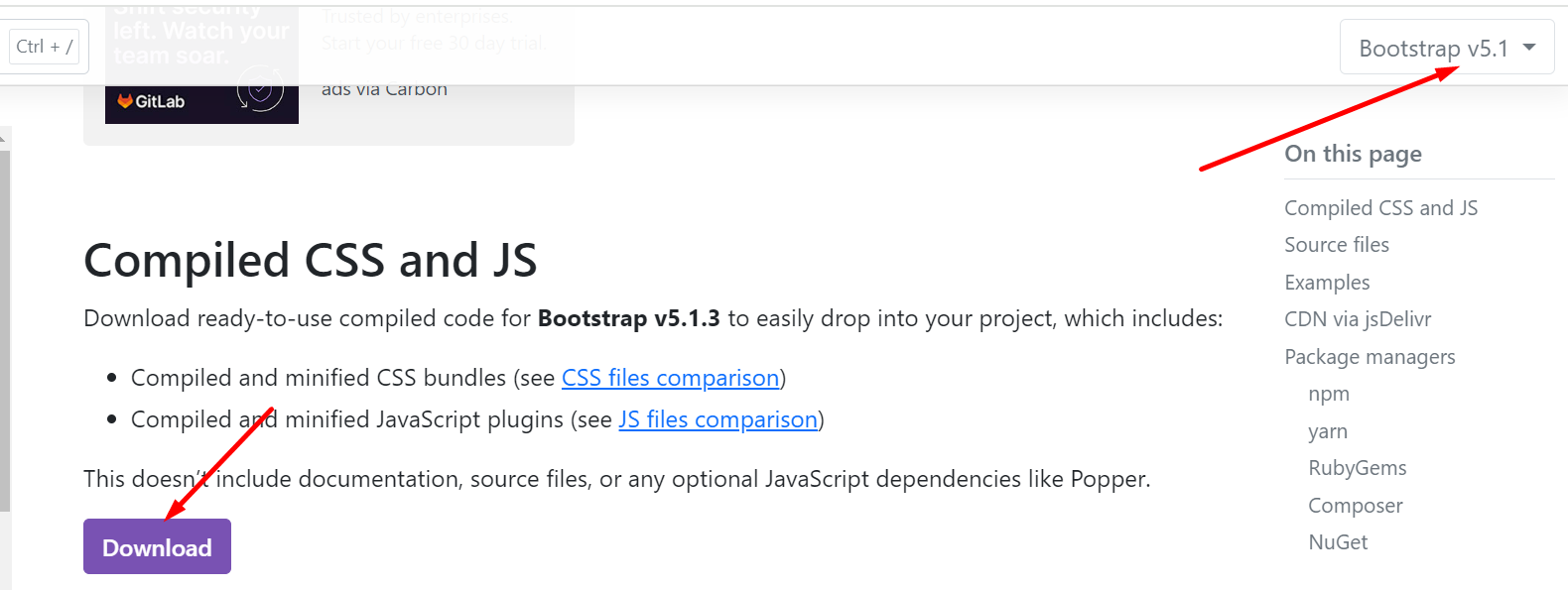
1. Baixar e instalar o Xampp

<https://www.apachefriends.org/>

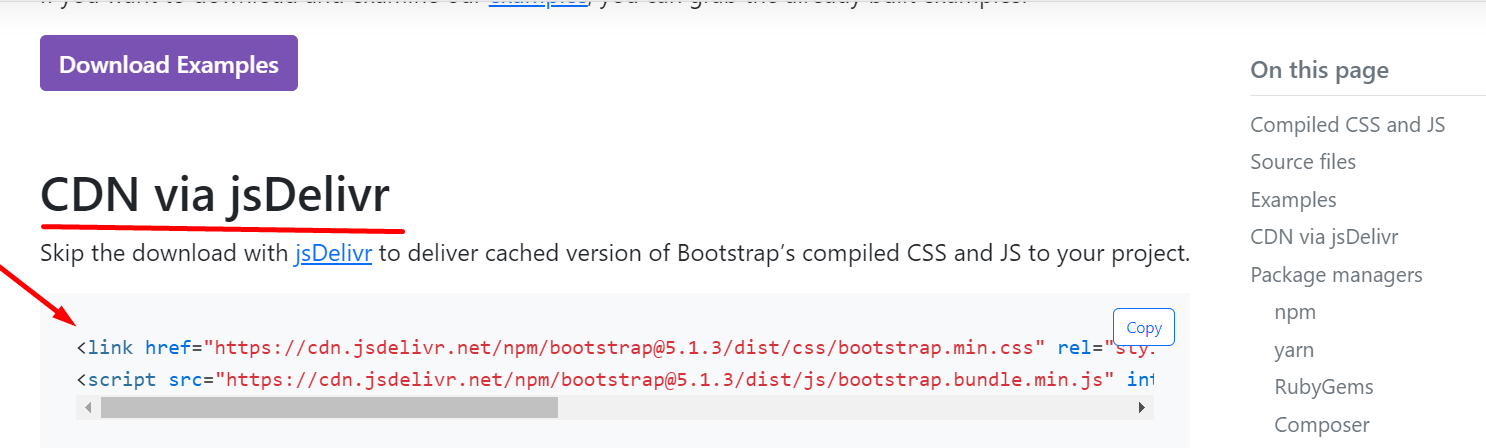


1. Baixar o bootstrap

<https://getbootstrap.com/>



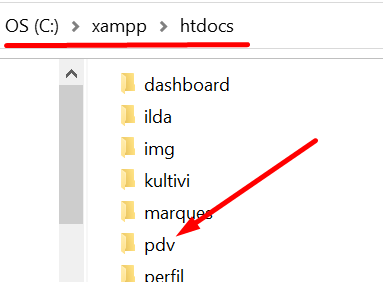
1. Copiar os links do CDN



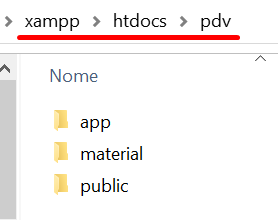
**Nota**: O jsDelivr é uma rede de distribuição de conteúdo (CDN, do inglês *Content Delivery Network*) que fornece uma maneira eficiente de entregar arquivos JavaScript, CSS e outros recursos estáticos para websites e aplicações web.

**Principais finalidades:**

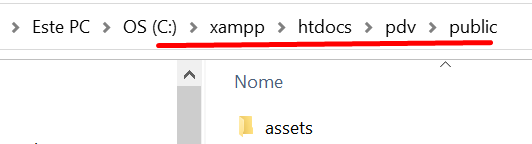
1. **Desempenho Melhorado**: O jsDelivr ajuda a acelerar o tempo de carregamento das páginas da web, armazenando em cache e distribuindo arquivos estáticos através de uma rede global de servidores. Isso significa que os arquivos são servidos a partir do servidor mais próximo do usuário, reduzindo a latência e melhorando a velocidade de carregamento.
2. **Alta Disponibilidade e Escalabilidade**: Como o jsDelivr usa uma rede de servidores distribuídos globalmente, ele pode lidar com altos volumes de tráfego e garantir que os arquivos estejam sempre disponíveis, mesmo em caso de falhas de servidor.
3. **Facilidade de Uso**: Oferece URLs para bibliotecas populares e frameworks JavaScript, CSS e fontes, o que facilita a inclusão desses recursos em projetos sem a necessidade de hospedá-los localmente. Isso é útil para garantir que você sempre esteja usando a versão mais recente e segura de uma biblioteca.
4. **Redução de Carga no Servidor**: Ao usar um CDN como o jsDelivr, você reduz a carga no seu servidor de origem, já que os arquivos estáticos são servidos pelo CDN em vez do seu servidor principal.
5. **Segurança**: O jsDelivr pode fornecer HTTPS, ajudando a proteger a integridade dos arquivos e garantindo que eles não sejam alterados ou corrompidos durante a transmissão.
6. Criar a pasta principal [**pdv**] para alocar os arquivos dentro do **xampp**



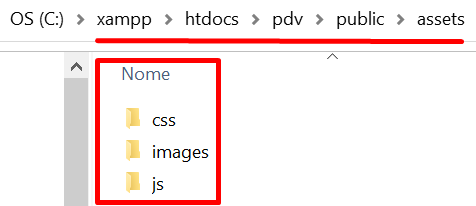
1. Criar as subpastas: **app**, **public, material** para a montagem do MVC:



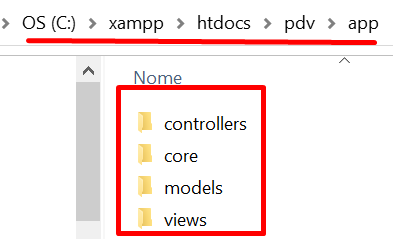
1. A pasta **public** vai alocar a subpasta **assets**



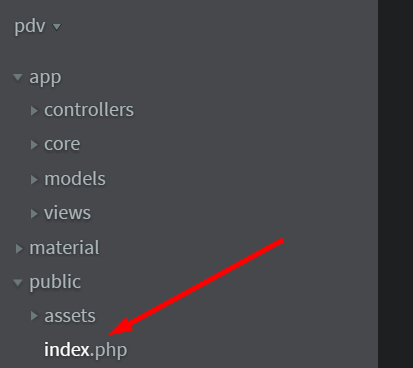
1. A subpasta **assets** vai alocar as pastas: **js**, **images**, **css**



1. Dentro da pasta **app** criamos as pastas que daram sentido ao modelo **MVC:** controllers, core, models, views.

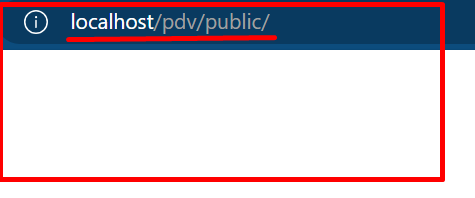


1. Na pasta **public** devemos criar o primeiro ficheiro **index.php**

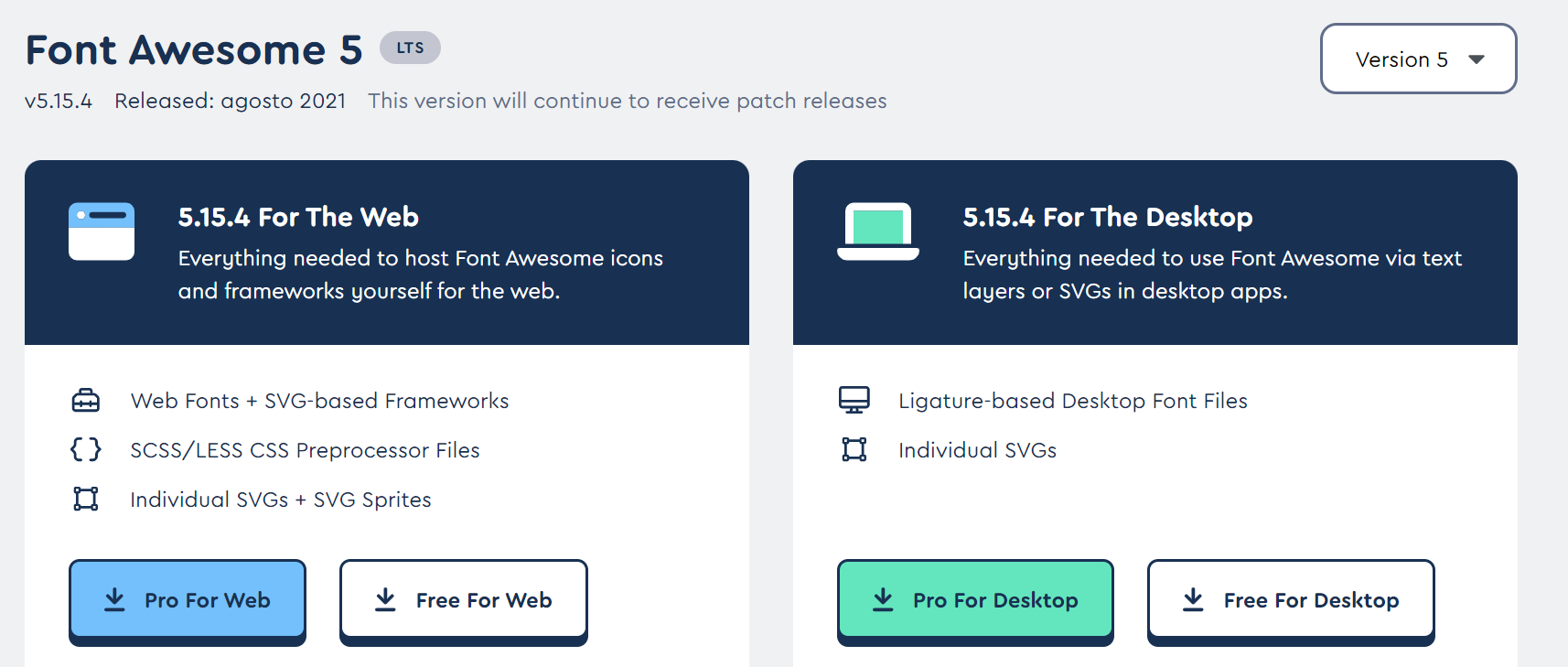


Nota: O ficheiro **index.php** na pasta public é geralmente a entrada principal para uma aplicação web PHP. Aqui estão alguns dos principais propósitos desse ficheiro:

1. **Roteamento de Requisições**: Muitas aplicações PHP, especialmente as que usam frameworks como Laravel ou Symfony, utilizam o **index.php** para rotear todas as requisições HTTP para o controlador apropriado. Isso é feito através do uso de um arquivo **.htaccess** que redireciona todas as requisições para **index.php**.
2. **Inicialização da Aplicação**: O **index.php** é responsável por carregar as dependências da aplicação, inicializar o ambiente e configurar as variáveis globais. Normalmente, ele inclui ou requer arquivos essenciais, como o **autoloader do Composer**.
3. **Segurança**: Ao colocar **index.php** na pasta public, você garante que outros arquivos sensíveis da aplicação (como configurações e scripts de backend) não sejam diretamente acessíveis a partir da web. Somente os arquivos na pasta public são expostos.
4. **Configuração de Ambiente**: Pode definir configurações específicas de ambiente (desenvolvimento, produção, etc.) para ajustar o comportamento da aplicação de acordo com o ambiente em que está sendo executada.
5. O proximo passo será testar o roteamento, mas não temos nenhum ficheiro dentro da pasta nos será apresentado como vazio, pois ainda não foi definido as rotas e adicionado o ficheiro <**index.php**> para a captura dos dados.

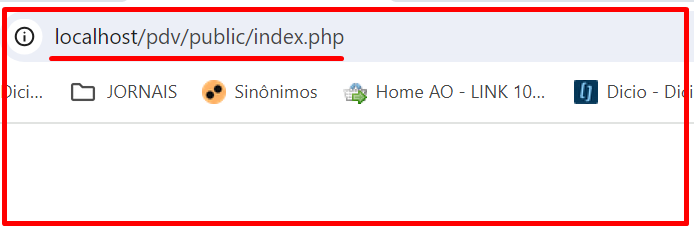


1. Após o teste da pasta vamos verificar o fontawesome <https://fontawesome.com/v5/search?o=r&m=free> onde podemos pegar todos os icons do sistema sem qualquer custo adicional ou <https://fontawesome.com/v5/download>

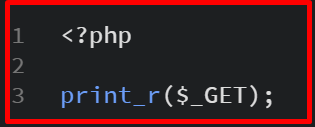


1. Testar os valores pegos da URL atraves da **variavel** **global**

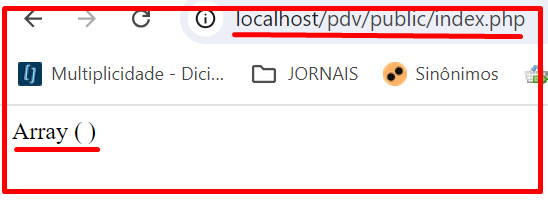
Ex:



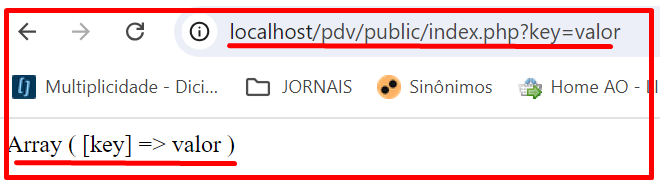
Agora devemos analisar como obter os valores da **URL,** dentro da pasta **public** existe o ficheiro **index.php** ficheiro responsavel por pegar os dados da **URL**.



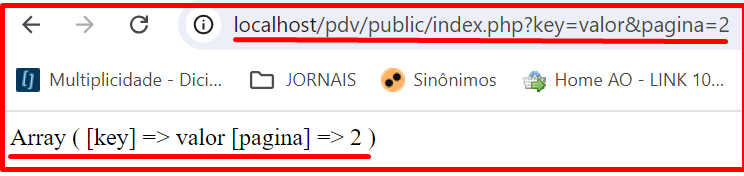
Resultado desta captura da URL



Todos os dados que são filtrados na URL deve ser capturado para o rediressionamento é filtrado como **metodo** por pegar as requisições.



E temos dentro do metodo o **controlador** das páginas internas

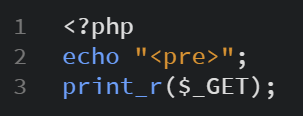


E dentro da página temos o **visualizar**

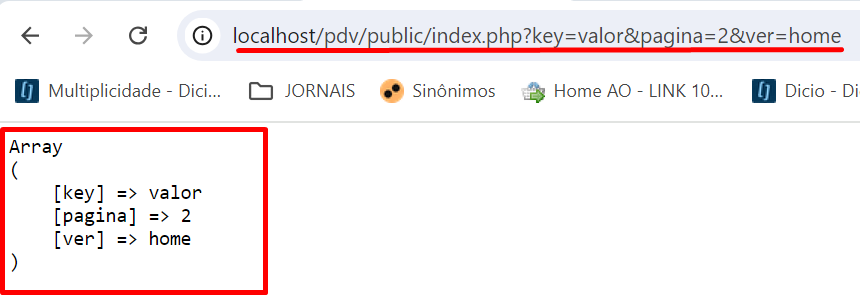


Para podermos ver de uma forma mais estruturada podemos filtrar estruturalmente e visualizar a atribuição como arrays.

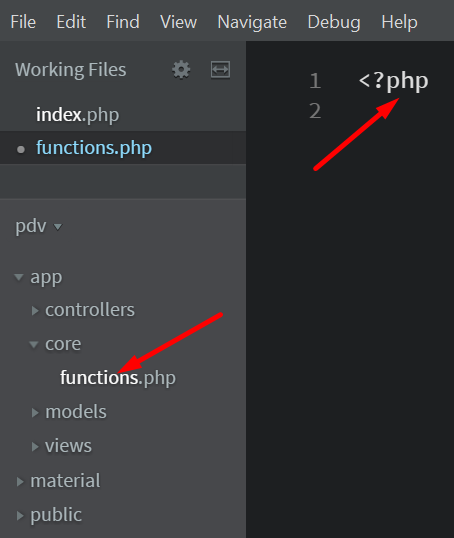
[**codigo**]



[**Resultado**]

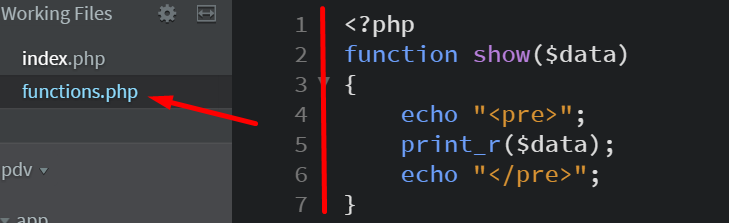


1. Dentro da pasta <**core**> criamos o ficheiro <**function.php**>

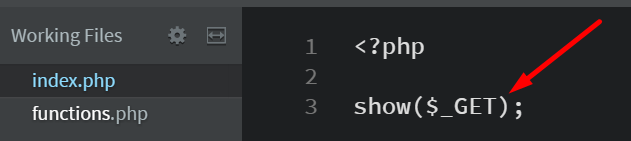


O **function.php** tem como proposito alocar as funcionalidades especiais, como **configuração da data & hora**, **funções monetarias** e outras regras como formatações monetaria, etc.

[**codigo:** definindo o function.php]



[**codigo:** chamando o **function.php** no **index.php**]



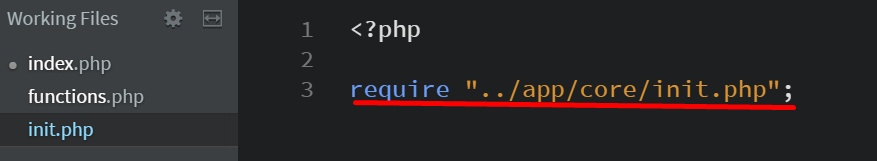
**[saida:** erro fatal pois não sabe aonde esta localizado o **metodo show()** por isso deve-se definir o **autoloader** para poder achar o metodo pretendido**.**

**Fatal error: Uncaught Error: Call to undefined function show() in C:\xampp\htdocs\pdv\public\index.php:3 Stack trace: #0 {main} thrown in C:\xampp\htdocs\pdv\public\index.php on line 3**

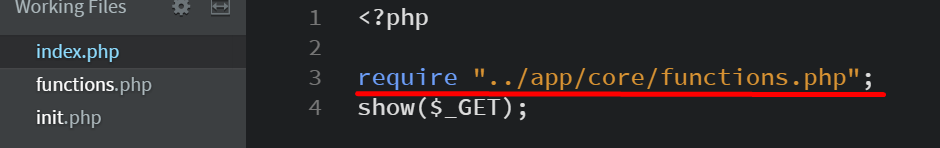


Nesse caso devemos criar dentro da pasta <**core**> o ficheiro **init.php** para poder inicialiazr ou fazero **autoloader (carregamento)** das funções definidas

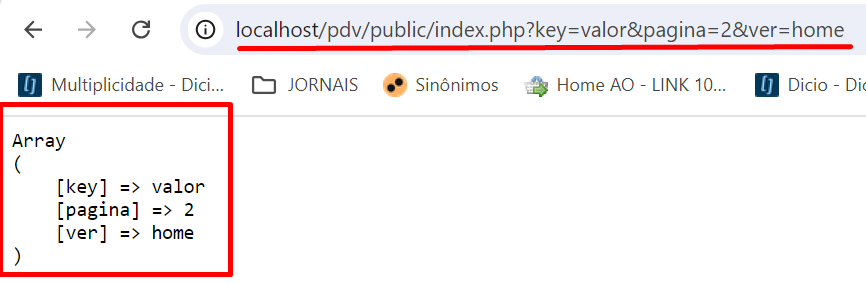
Path no **init.php**



Path no **function.php**



**Resultado:** Testadoexiste resposta no autoloader definido



**Nota**:

o contexto de um projeto PHP, a pasta <**core>** é frequentemente usada para armazenar arquivos fundamentais que implementam a lógica principal do sistema ou framework.

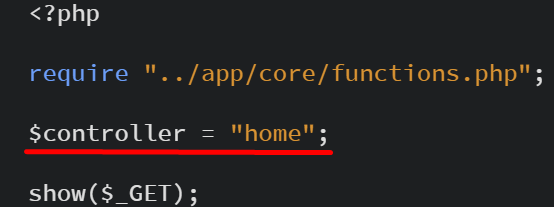
O arquivo **functions.php** dentro dessa pasta geralmente tem as seguintes finalidades:

1. **Funções Utilitárias**: Contém funções auxiliares e utilitárias que são usadas em várias partes do sistema. Essas funções podem realizar tarefas comuns, como manipulação de strings, operações de data e hora, ou outras operações que são frequentemente necessárias.
2. **Funções de Inicialização**: Pode incluir funções responsáveis por inicializar configurações e carregar recursos essenciais para o funcionamento da aplicação.
3. **Funções de Configuração**: Pode definir funções que configuram ou ajustam parâmetros do sistema, como conexões com o banco de dados, parâmetros de configuração, ou variáveis globais.
4. **Funções de Integração**: Facilita a integração entre diferentes componentes do sistema ou framework. Por exemplo, funções que ajudam a carregar classes, gerenciar sessões, ou lidar com requisições HTTP.
5. **Abstrações e Helpers**: Fornece funções de "helper" que abstraem operações complexas, tornando o código mais limpo e fácil de manter.

Geralmente, o **functions.php** serve como um ponto central para funções que precisam estar disponíveis globalmente em diferentes partes do código. A ideia é manter o código modular e reutilizável, e centralizar funções que são usadas em diversos locais do projeto.

1. No ficheiro **<index.php>** da pasta **public** deve-se definir o metodo ou variavel **$controller = “home”;** pois serve para especificar qual controlador (controller) será carregado por padrão quando uma requisição é feita ao servidor web.

**Razões para definir o método $controller = "home";:**



* 1. **Controlador Padrão:**

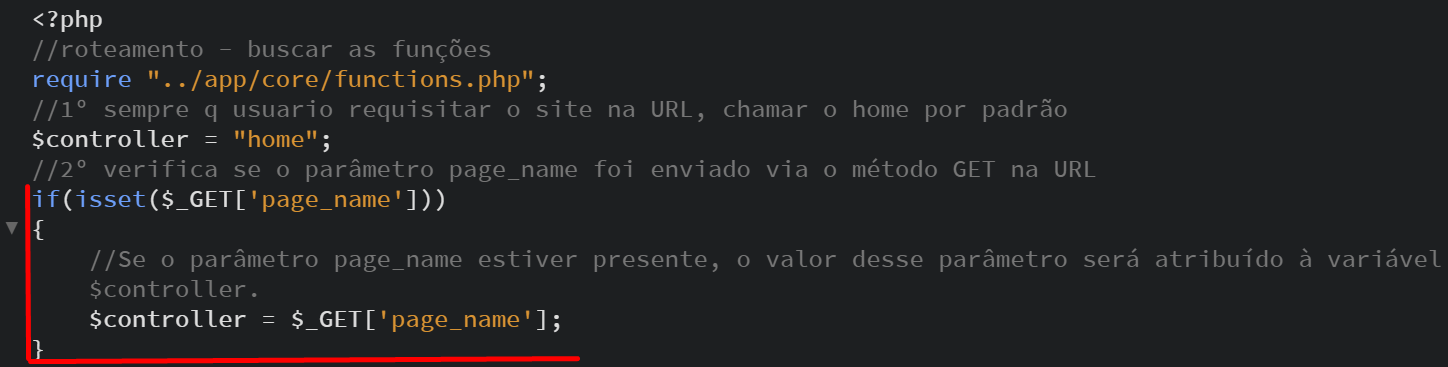
O controlador "**home**" pode ser configurado como o controlador padrão, que é chamado quando o usuário acessa a raiz do site ou uma URL sem especificar um controlador.

**Exemplo:** Se o usuário acessar **http://seudominio.com/,** o controlador **"home"** será chamado para processar a requisição**.**

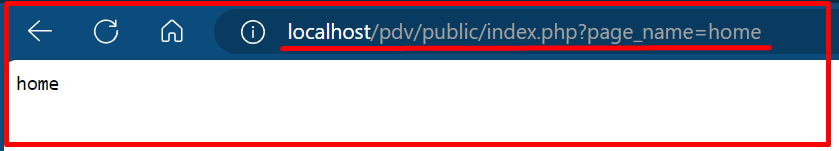
* 1. Verificar o parametro **<page\_name>** foi enviadona **URL** ex: <http://exemplosite.com/index.php?page_name=home>

Se **page\_name** estiver presente o valor desse parametro será atribuido a variavel **$controller** logo **$controller** será igual a "**home**".

[**codigo1**]: pegar a variavel da URL

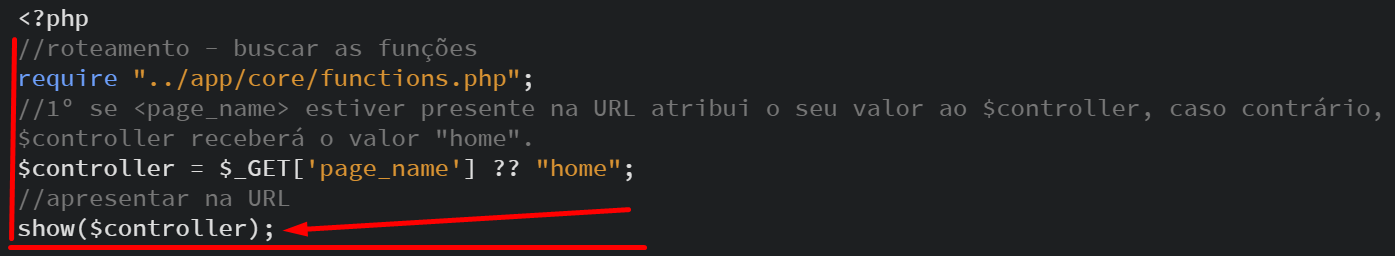


[**Resultado**]

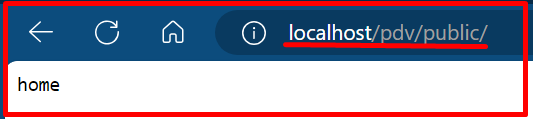


* 1. Mas para uma melhor organização do **código**, vamos simplificar numa unica linha. E caso remova somente a URI com este **codigo2** por padrão sempre que aceder a página ela redireciona para o “**home**”.

[**codigo2**]: pegar a variavel da URL

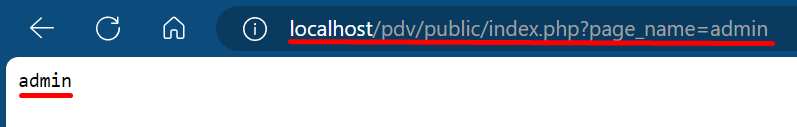


[**Resultado**]

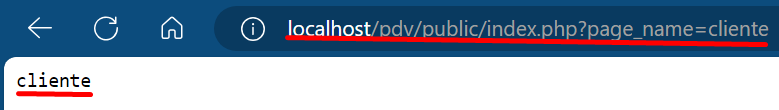


* 1. Agora se definirmos uma variavel na URL diferente de **home** ela atribuira o valor da **page\_name** ao **$controller**, mas se estiver vazio como no exemplo acima ela atribui ao “**home**” seu metodo padrão.

**Exemplo:1**



**Exemplo:2**



Exemplo:3



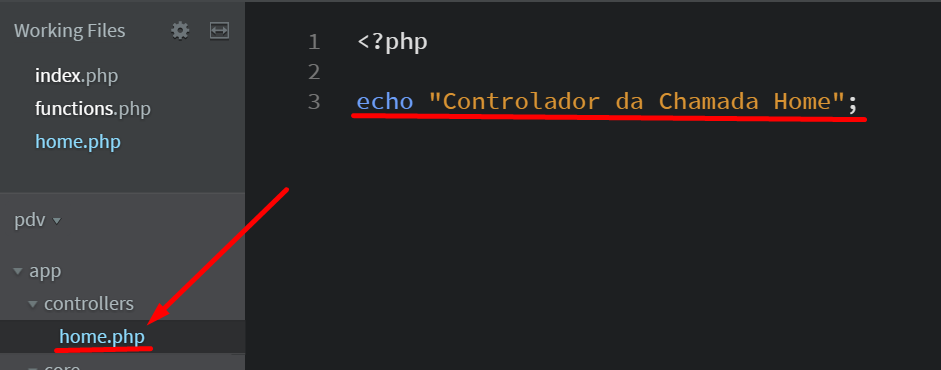
**Nota**: Tendo o sistema de rota funcional agora devemos criar **os ficheiros controladores** na pasta controler, que respondera, ou seja, será o responsavel por coordenar todo metodo atribuido a page\_name no link da **URL.**

**Exemplo:**

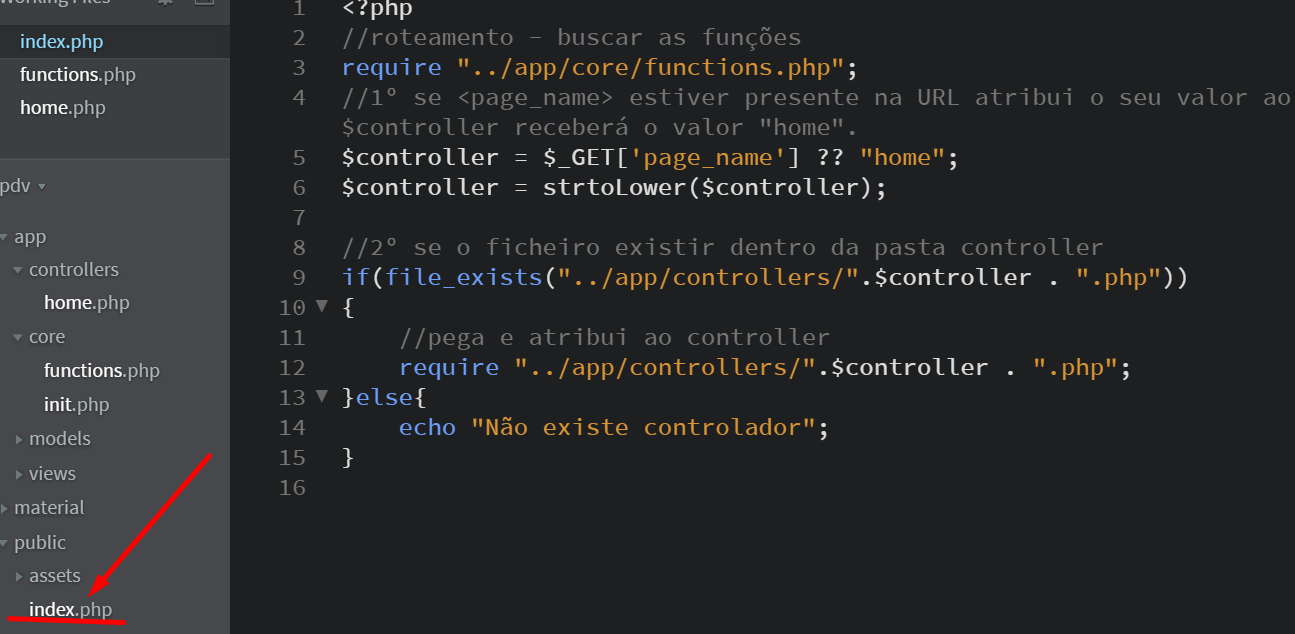
o **page\_name=home** terá o seu ficheiro dentro da pasta controller <**home.php**>, o **page\_name=admin** terá o seu ficheiro dentro da pasta controller <**admin.php**>, o page**\_name=cliente** terá o seu ficheiro dentro da pasta controller <**cliente.php**>, o **page\_name=financeiro** terá o seu ficheiro dentro da pasta controller <**financeiro.php**>.

1. Criando os metodos da **page\_name** dentro da pasta controller para passar a atribuição aos seus controladores.

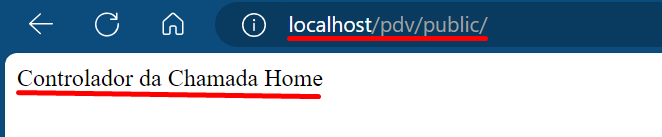
[codigo]: **home.php**



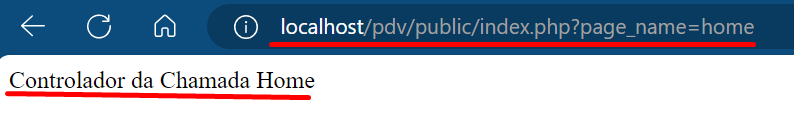
[codigo]: **index.php**



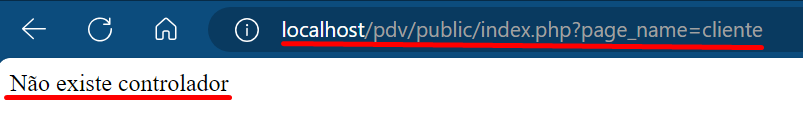
[**Resultado**]: controlador **home** padrão



[[**Resultado**]: controlador **home** atributo **page\_name**]

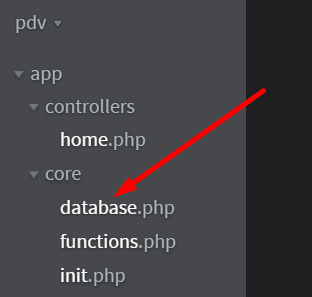


**OBS**: Agora vamos atribuir ao page\_name um metodo sem controlador para verificar o seu resultado.

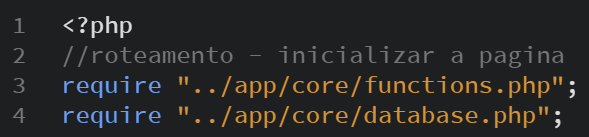


**Nota**: A partir deste momento é visivel ver a importancia dos ficheiros controladores dentro da pasta **controllers** pois todas as intruções **$\_GET** na **URL** é entrega aos ficheiros para avaliar e controlar toda a operação do sistema.

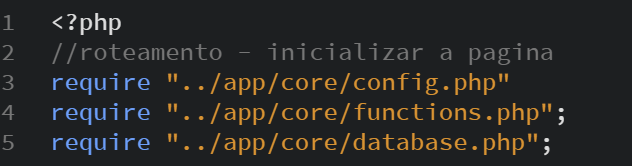
1. Agora na pasta <**core**> temos que criar o ficheiro **database** para criação das regras de entrada de dados diretamente a base de dados.

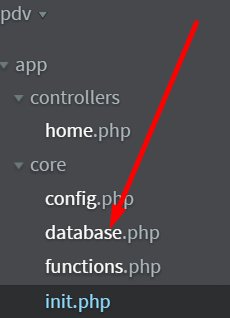


**Nota**: todos novos ficheiros que forem adicionados dentro da **pasta** **core** devem ser roteados no ficheiro **init.php** da mesma pasta.

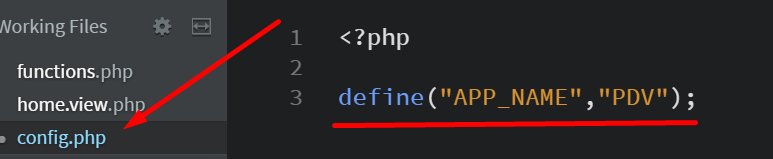


* 1. Depois disso temos que criar o ficheiro **config.php** dentro da pasta **core**. E adicionar o novo ficheiro para dentro da rota **init.php**

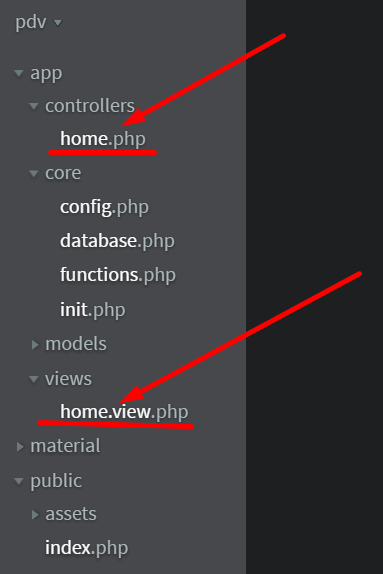




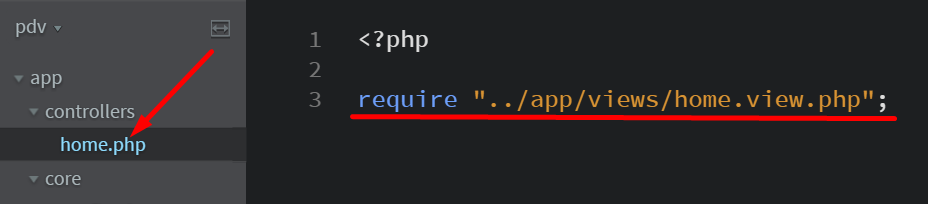
* 1. O ficheiro **config.php** será usado para definir as constantes do negócio, ou seja, toda chamada de acesso a base de dados, passa por ele.



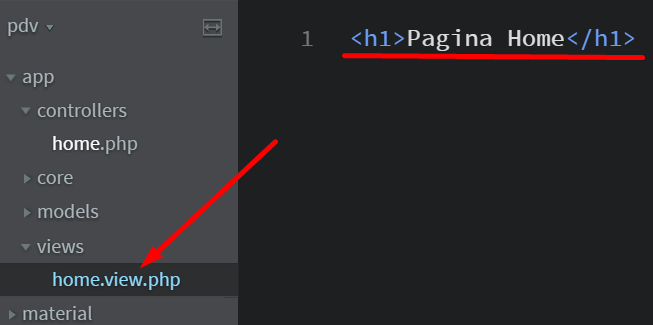
* 1. Todo **controlador** carece de uma **view** e vice-versa, pois cada controlador é resposável pela sua view.



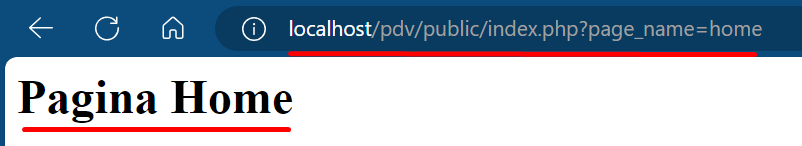
* 1. Agora faremos um pequeno teste em que o **controlador** toma o controle pela **view**, de forma a trabalharem ambos em conjunto. Aqui o controlador busca a **view do home**



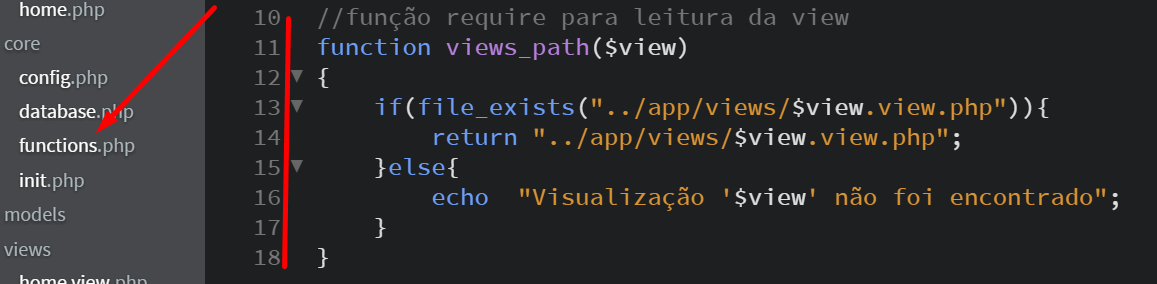
* 1. Aqui é definido a **view do home**



* 1. Aqui é feito a saída do resultado no browser



* 1. Para uma questão de estetica e controle dos roteamentos podemos mover o **require** do controller **home.php** e move-lo para dentro da **function.php**

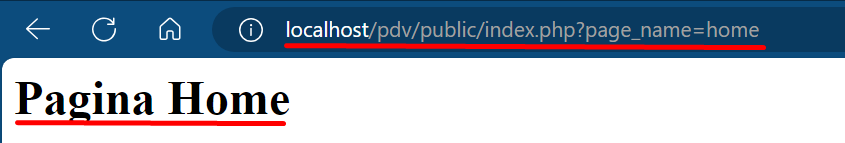


* 1. E atualizamos o código do controller **home.php**

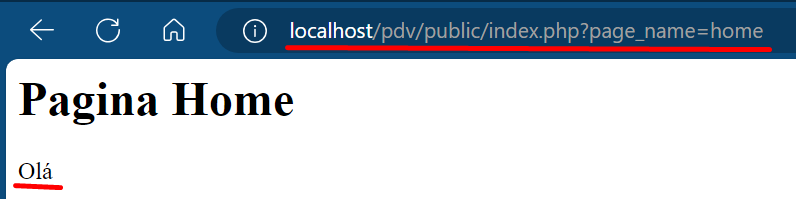
**Nota1**: O dado atribuido a variavel **$myvar** é passado para o frm **view.home** para que seja apresentado na pagina.



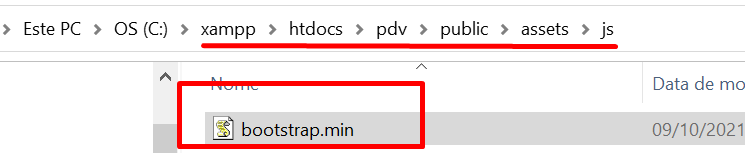
* 1. Atualizando a página no browser e o resultado continua sendo o mesmo.



* 1. Agora podemos verificar a saida do **$myvar** = “**hey**” definido um echo na view do home.



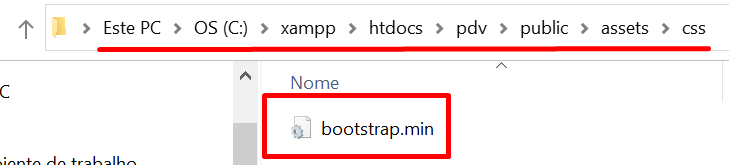
1. Tendo o preparado o controler e a view referente ao home.php agora vamos nos focar ana estetica da pagina ou seja **o bootstrap** pertencente ao **bootstrap-5.1.3-dist.**



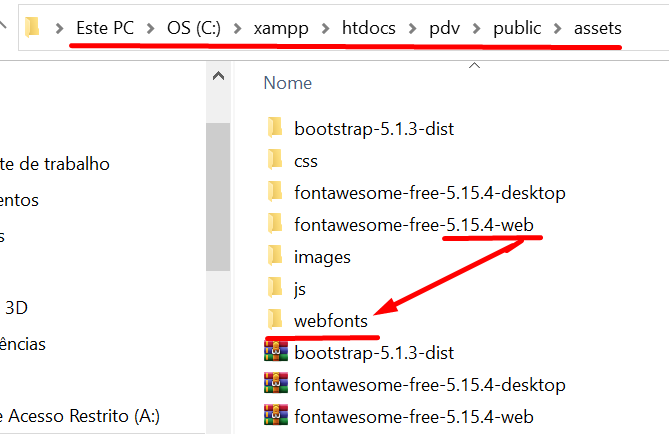
Retiramos da **pasta** abaixo:



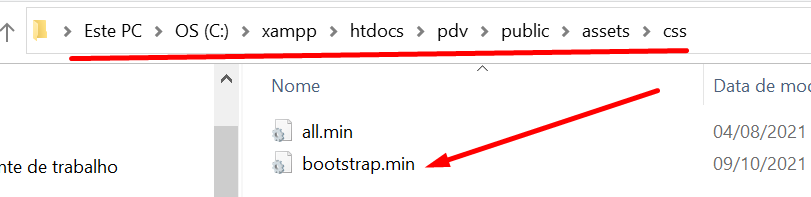
Assim como foi feito no **js** também deverá ser feito no **css**



E após isso vamos ao **fontawesome** e copiamos o **webfonts** e colamos simplemente dentro da pasta assets.



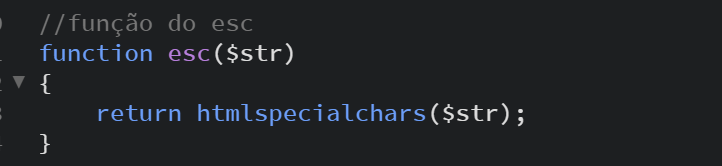
Após isso na mesma pasta do **fontawesome** devemos copiar o ficheiro para o **css**.



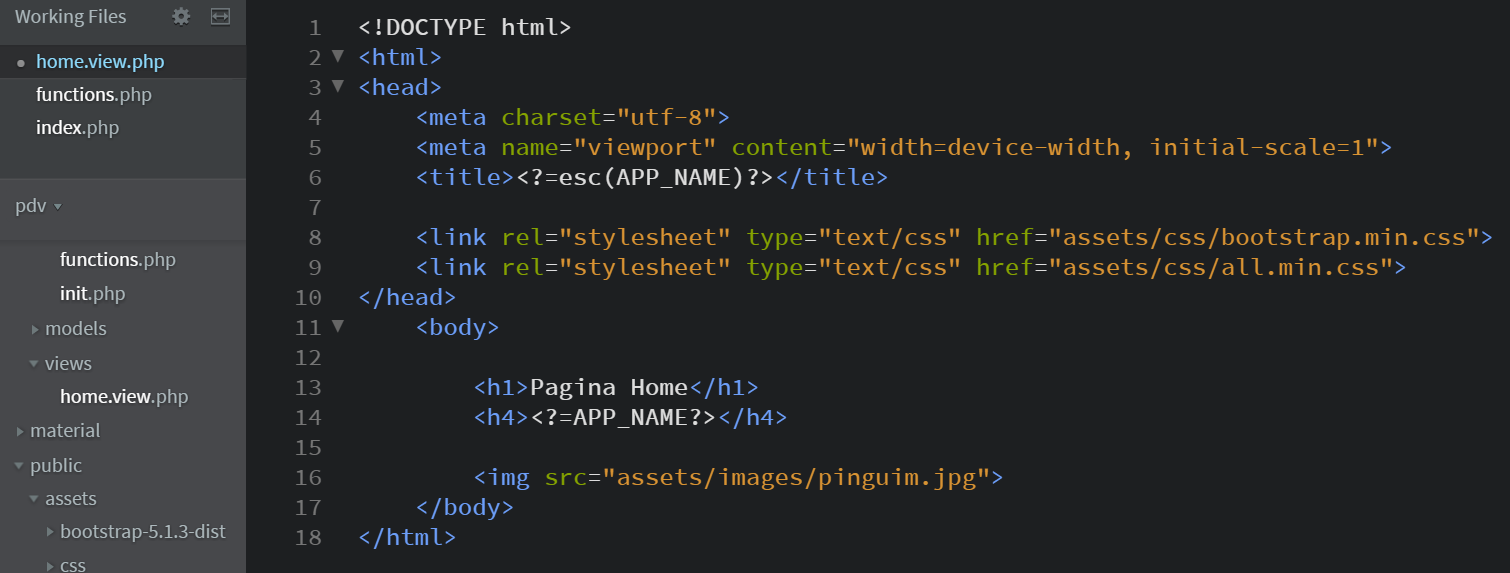
1. Tendo todos os ficheiros nas referidas pastas vamos agora começar a trabalhar no [**home.view.php**](http://home.view.php)



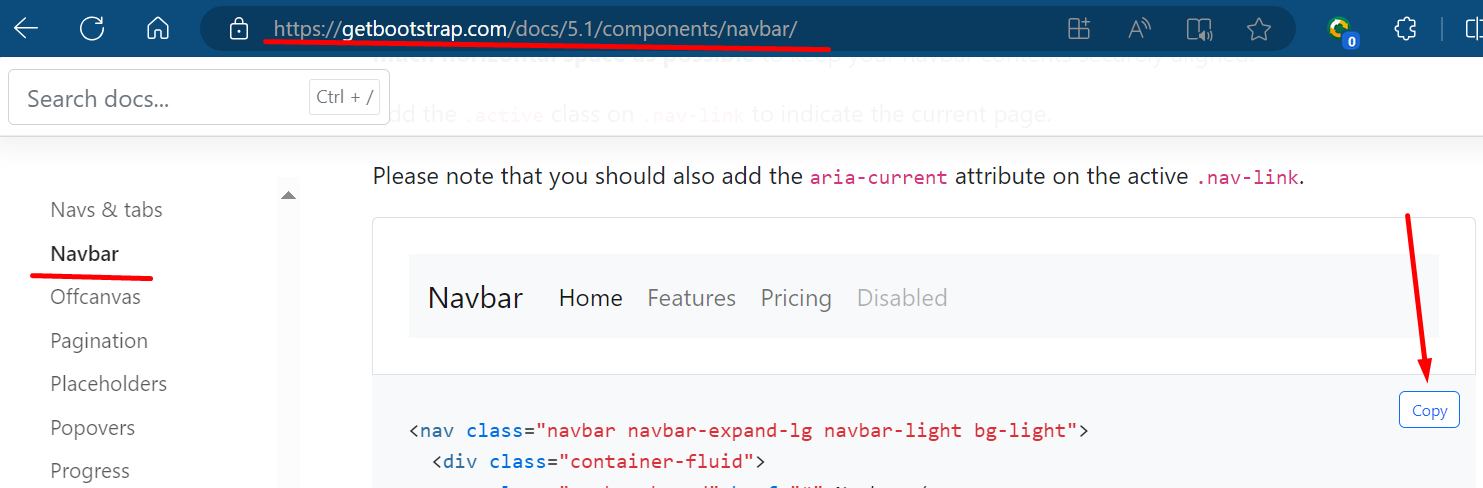
Uma vez que temos o **esc** ativo no **home.view** devemos definir a sua chamada a partir do **function.php**



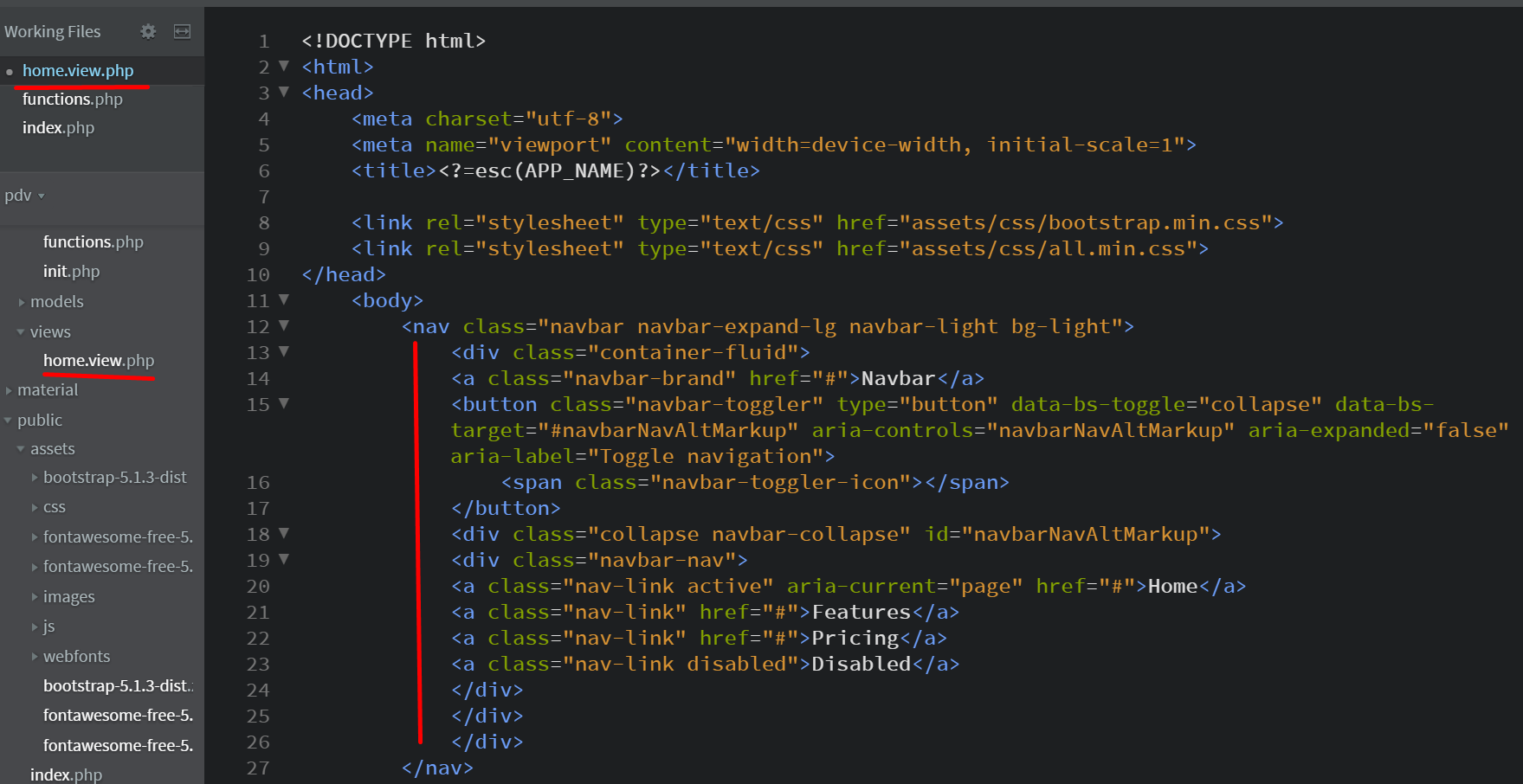
Aqui estamos testando o roteamento e a chamada da image



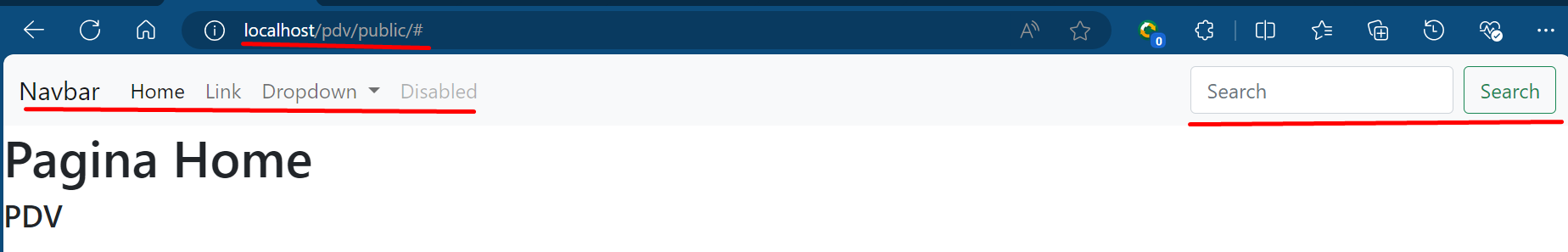
Após isso devemos atualizar a página view do menu home, vamos **copiar** o mesmo para colocação do menu no sistema.



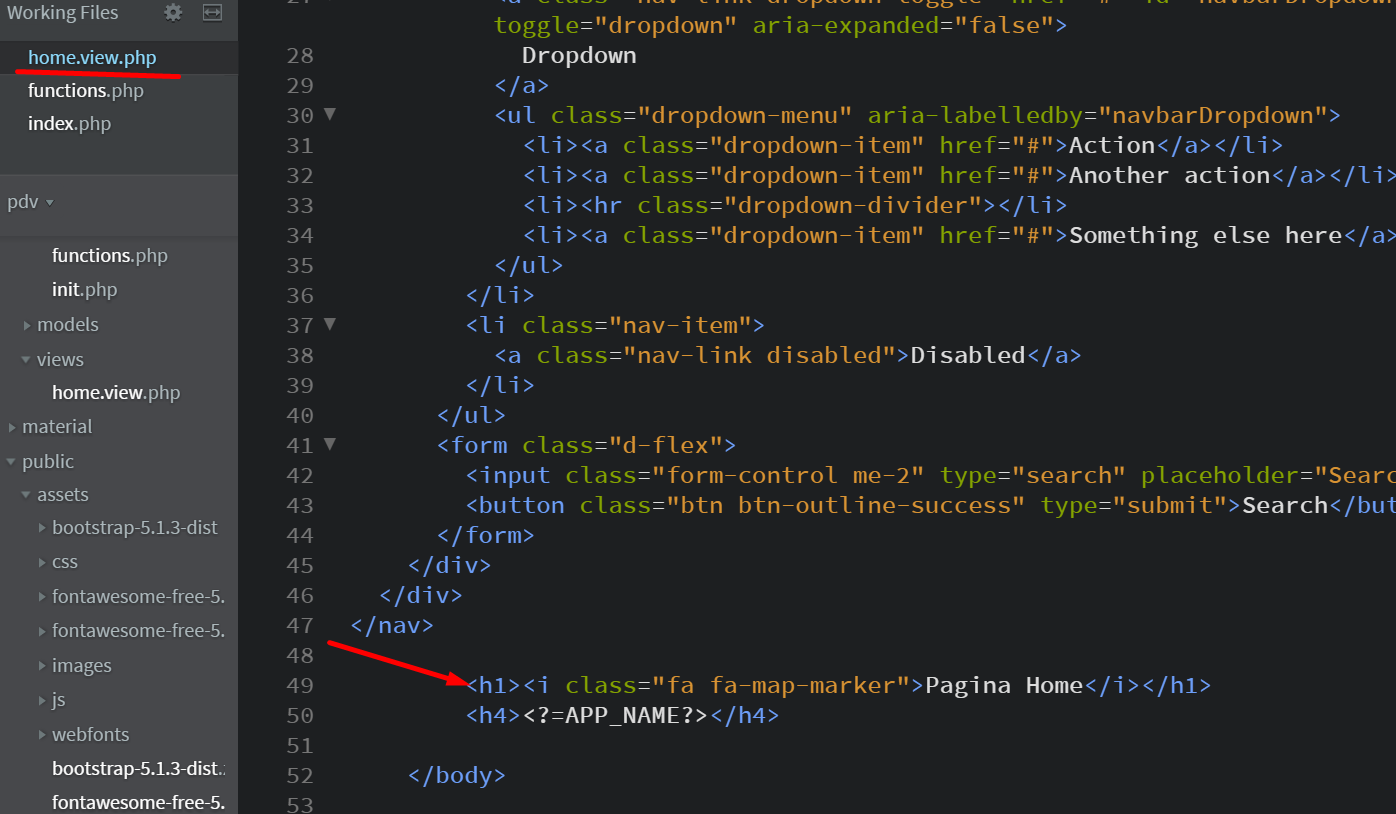
E desta fica deste jeito a colocação do código



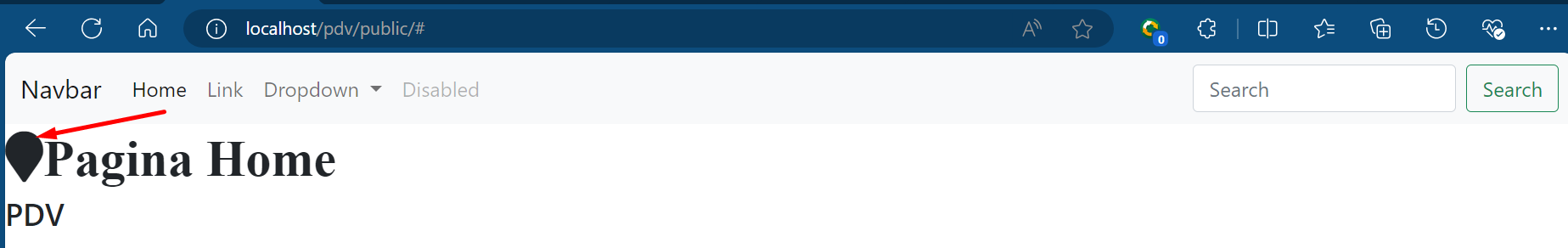
E com o código que conseguimos montar a navegação



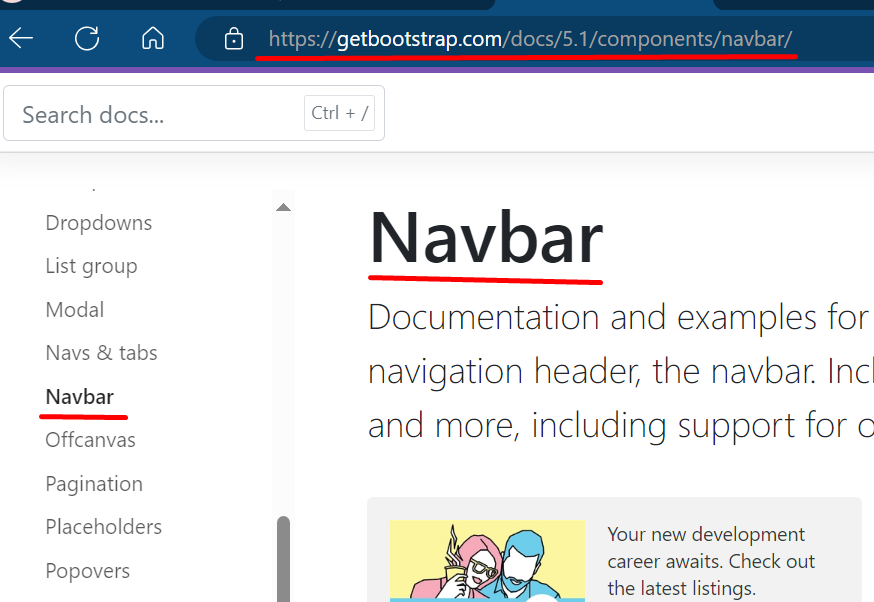
E definimos no **home.view** o icone das imagens



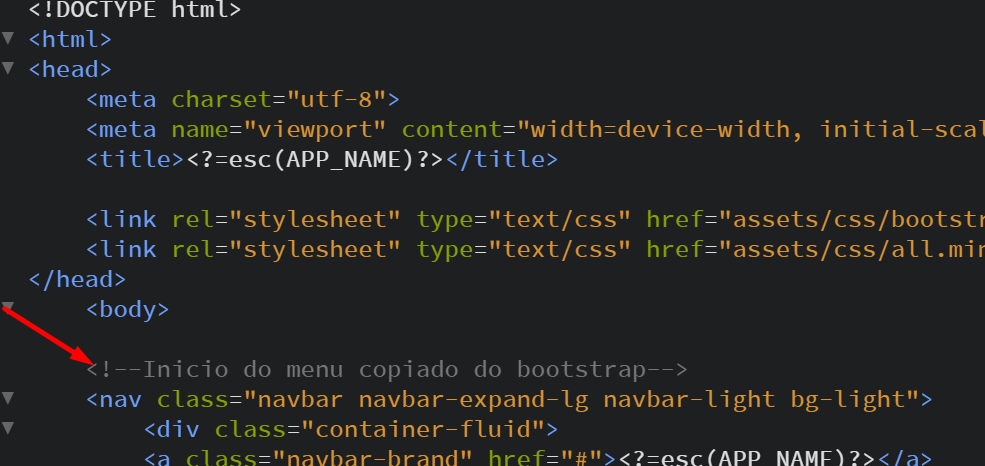
Resultado no browser

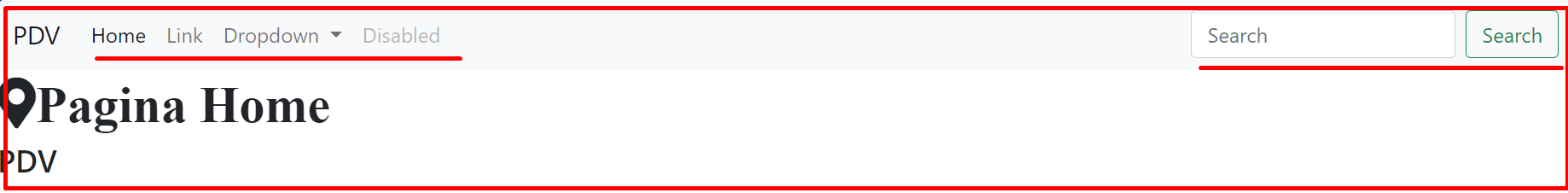


1. **Links de navegação**
2. **Navbar** copiado do bootstrap para chamada dos menus, ou seja, sempre que for preciso um menu devemos procura-lo dentro do bootstrap.

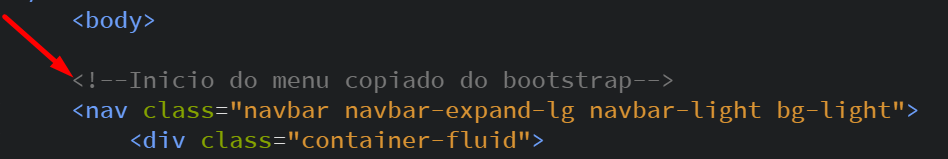


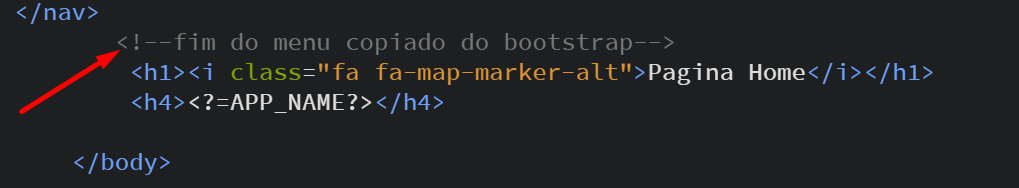
[Navbar · Bootstrap v5.1 (getbootstrap.com)](https://getbootstrap.com/docs/5.1/components/navbar/)



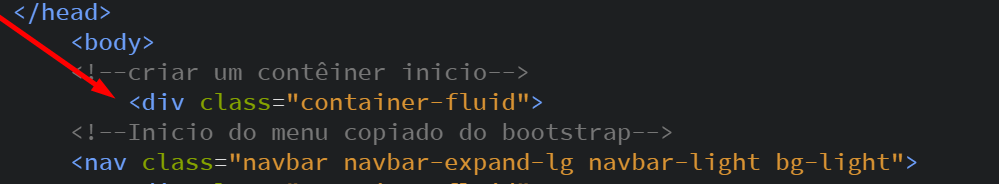


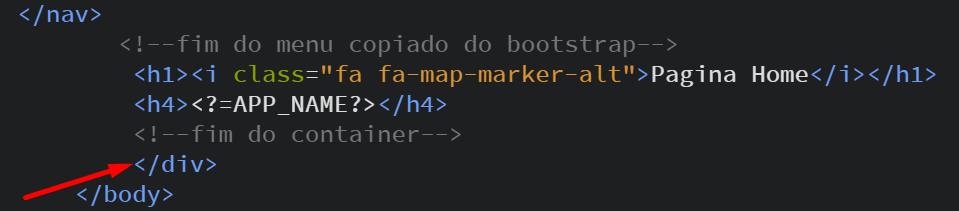
1. No entanto o **navbar** cópiado para o código carece de uma referência de controlo, desse modo vamos adicinar entre o início e o fim dentro do body o **container** de controlo.





1. Agora temos a navbar incorporada para dentro do container que será o responsavel pelo seu gerenciamento.

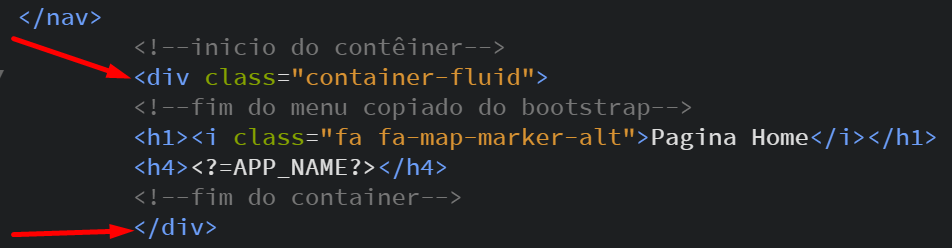




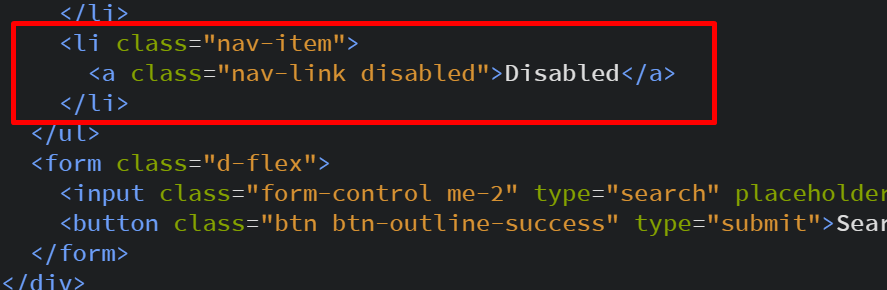
1. Com a entrada dp container temos as laterais do menus ajustado ao site.

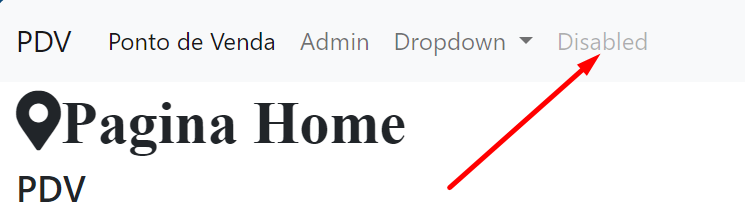


1. Como também podemos recorta-lo e deixa-lo controlar somente a barra a esquerda da navegação.



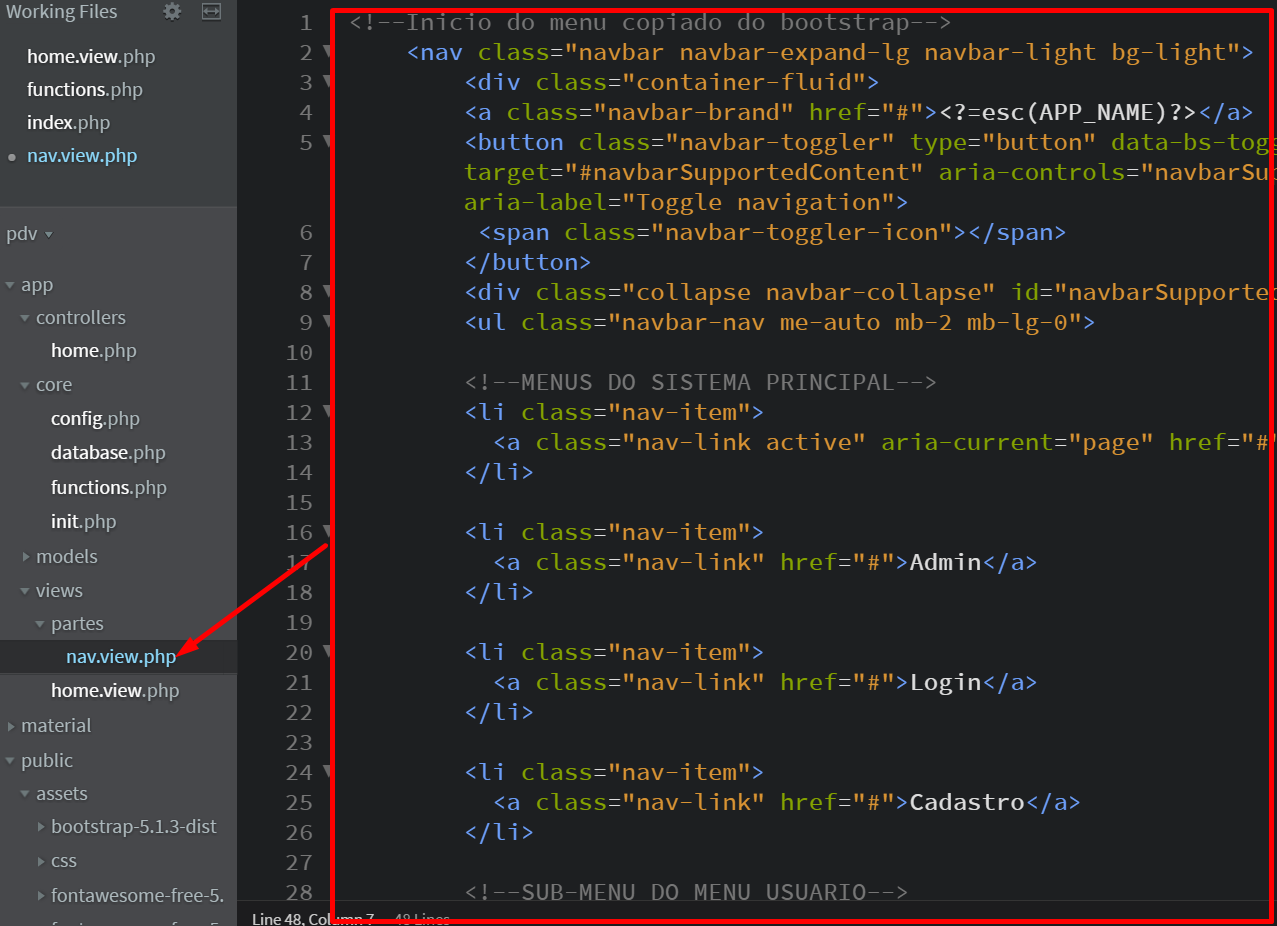
1. Removemos o menu disabled pois não precisamos da sua chamada.



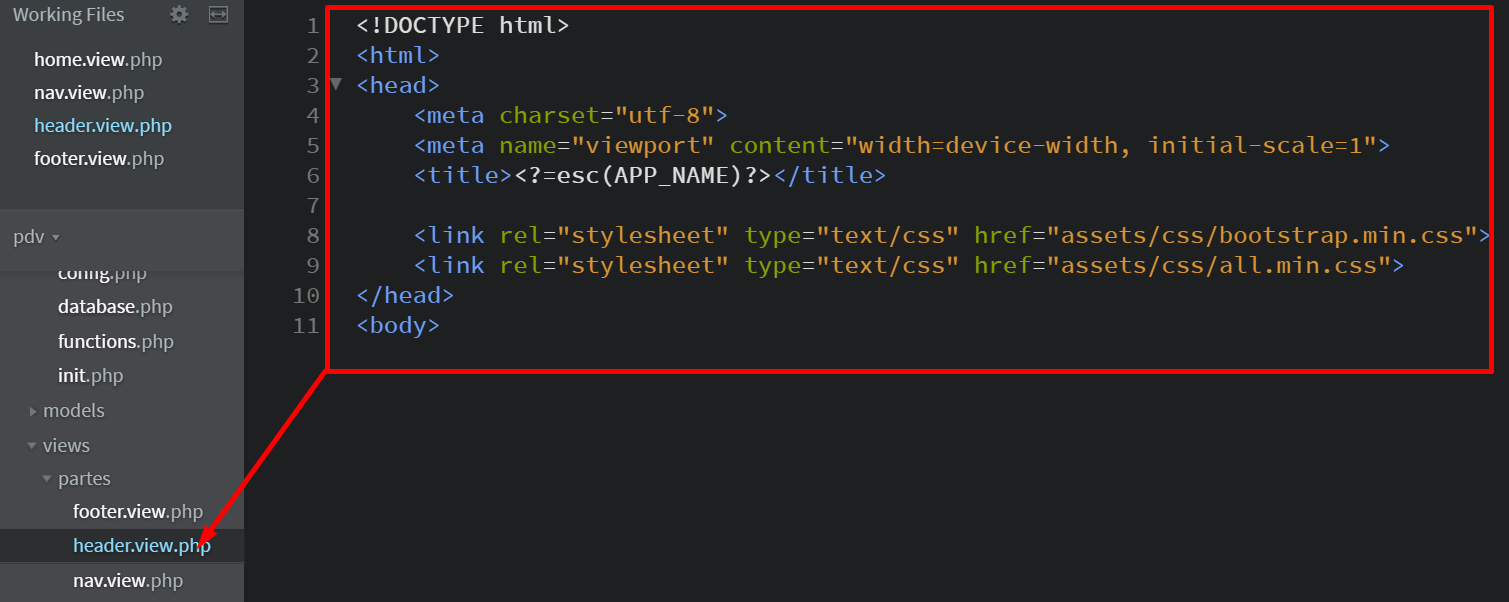


1. Para um melhor controlo do código vamos repartir o codigo de **navegação do menu** por partes para que seja compreensivel.

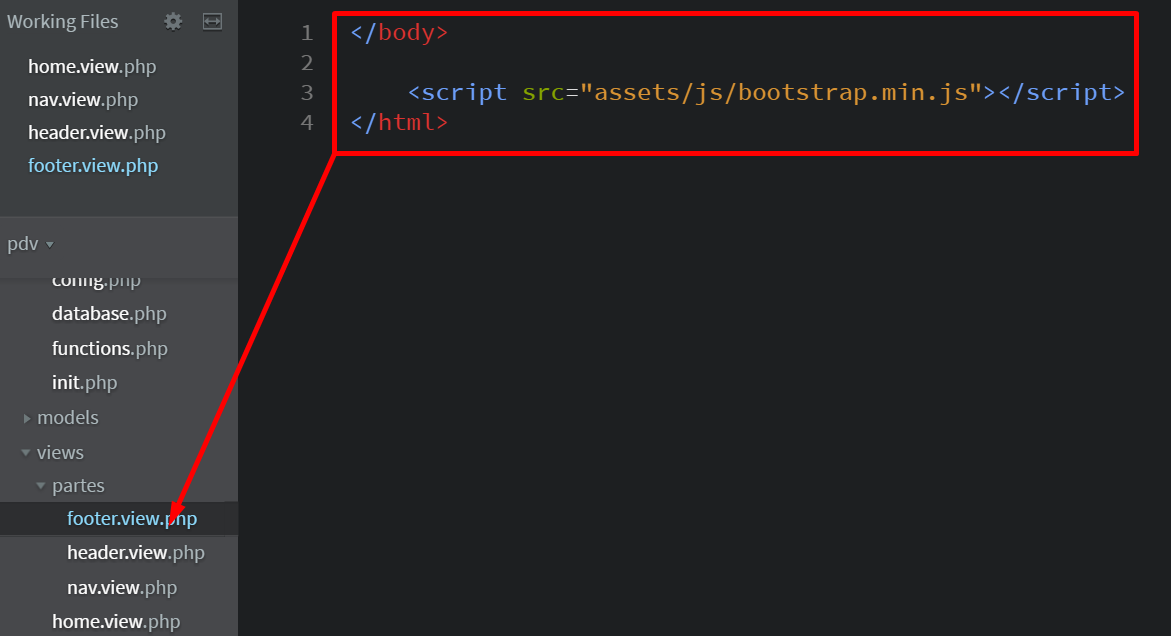




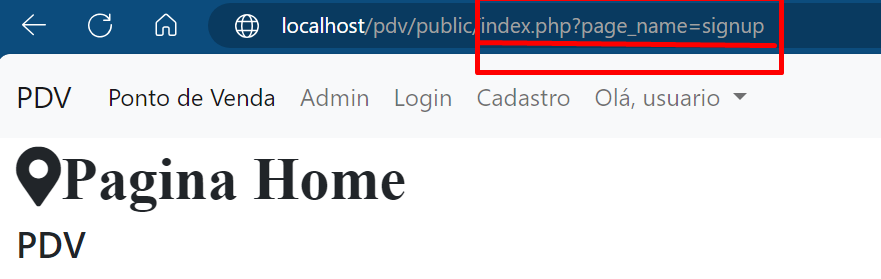
1. Vamos separar também o cabeçalho



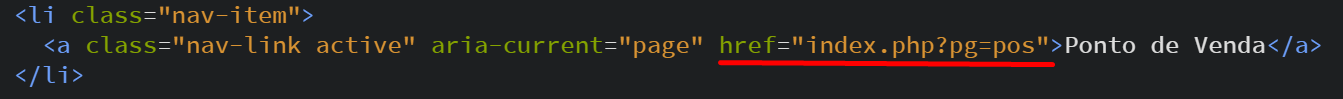
1. Vamos separar também o rodápe

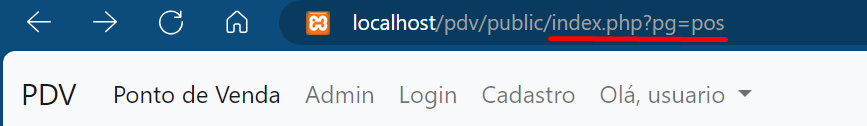


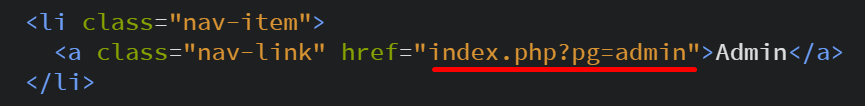
1. Copiamos o link para a chamada da referida página do sistema e substitumos pela a ancora que naqual ela faz referencia.

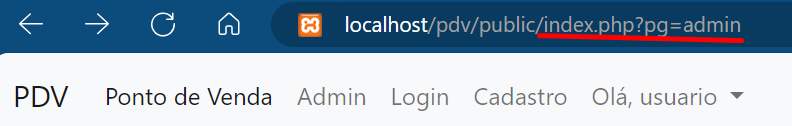


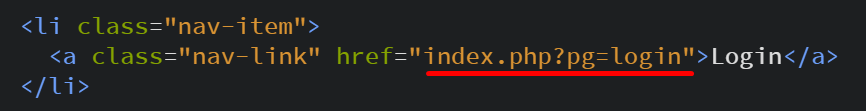
Ou seja, cada link atribuido a uma ancora deve ter seu proprio controlador de página.

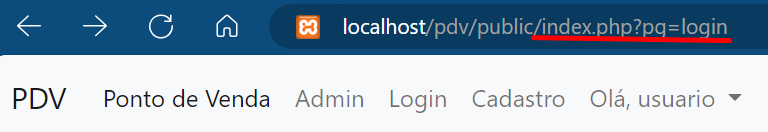


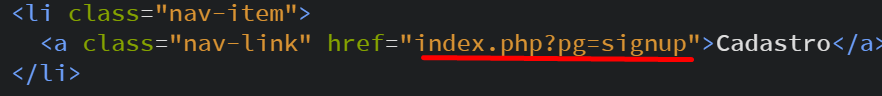


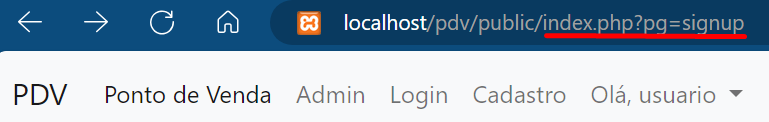




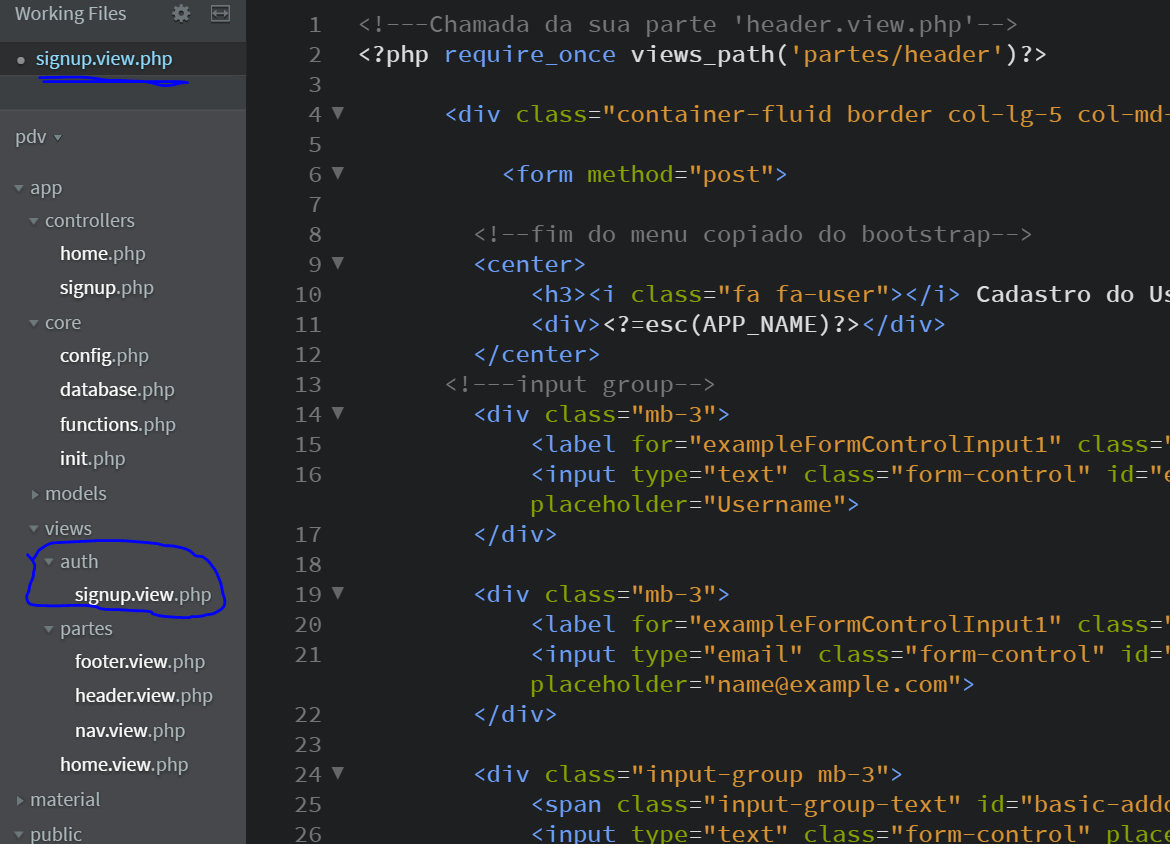








1. Trabalharemos agora no formulario de cadastro, ou seja, inscrição.



1. ...
2. ....

**ARQUITETURA – MVC**

No padrão de arquitetura Model-View-Controller (**MVC**) aplicado ao PHP, cada pasta tem uma função específica que ajuda a organizar o código de forma modular e facilitar a manutenção e desenvolvimento. Aqui está a finalidade de cada uma delas:

1. **Controllers**:
   * **Função**: Os controladores (controllers) atuam como intermediários entre os modelos (**models**) e as visões (**views**). Eles recebem as requisições HTTP, processam os dados (geralmente obtendo-os dos modelos), e decidem qual visão será apresentada ao usuário.
   * **Exemplo**: Um controlador de usuário (UserController) pode ter métodos para **criar**, **editar**, **atualizar** e **deletar** usuários.
2. **Models**:
   * **Função**: Os modelos (**models**) representam a lógica de dados e as regras de negócios da aplicação. Eles lidam com a interação com o banco de dados, como a **recuperação**, **inserção**, **atualização** e **exclusão** de dados.
   * **Exemplo**: Um modelo de usuário (**User**) pode ter métodos para encontrar um usuário por ID, validar dados de usuário, e salvar um usuário no banco de dados.
3. **Core**:
   * **Função**: A pasta core geralmente contém classes e funcionalidades essenciais que suportam o framework MVC. Isso pode incluir o roteamento, classes base para controladores e modelos, gerenciadores de sessão, e outras utilidades que são usadas em toda a aplicação.
   * **Exemplo**: Um roteador (**Router**) que decide qual controlador e método chamar com base na URL da requisição.
4. **Views**:
   * **Função**: As visões (**views**) são responsáveis por apresentar a interface do usuário. Elas contêm a marcação HTML e, às vezes, lógica simples de exibição. As visões recebem os dados dos controladores e geram a saída HTML para o navegador.
   * **Exemplo**: Uma visão de perfil de usuário (profile.php) que exibe os dados do usuário em uma página web.

**Resumo da Estrutura MVC no PHP:**

* **Controllers**: Processam as requisições e coordenam a interação entre modelos e visões.
* **Models**: Gerenciam os dados e regras de negócio.
* **Core**: Contém as funcionalidades e classes essenciais do framework.
* **Views**: Apresentam a interface do usuário.

Cada pasta e seu conteúdo trabalham juntos para separar as responsabilidades da aplicação, facilitando a manutenção, o desenvolvimento colaborativo e a escalabilidade do projeto.