**Escola Secundária de Silves**

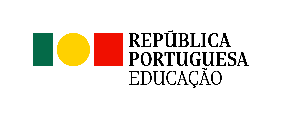
**Curso Profissional de Técnico de Informática - sistemas**

**Relatório da Loja Online de Peças Automotivas e Tratores**



Nº 12285 Ricardo Leandro Medina Cabrita

**08/07/2024**



**Escola Secundária de Silves**

**Curso Profissional de Técnico de Informática - Sistemas**

**08/07/2024**

**Relatório da Loja Online de Peças Automotivas e Tratores**

**Relatório Final**

**da**

**Prova de Aptidão Profissional**

Nº 12285 Ricardo Leandro Medina Cabrita

Orientado pelos Professores: Mónica Franco

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Nome do Aluno) (Professores responsáveis)

**Agradecimentos**

Queria primeiramente agradecer ao meu tutor de estágio que eu acompanhei durante dois anos por ter me ajudado no desenvolvimento do projeto de aptidão profissional, ele que me ajudou na construção da aplicação.

Também queria agradecer aos meus pais pelo apoio deles.

E também queria agradecer a minha namorada por sempre me animar e de dar motivação para continuar a construir o projeto.

E queria agradecer a minha professora orientadora Mónica Franco pela ajuda e compreensão da parte dela.

**Resumo**

Este relatório vem ao abrigo do projeto desenvolvido para validação de conhecimentos do meu processo escolar, durante os anos em que participei no curso promovido pelo agrupamento escolar.

O projeto realizado consiste num site de uma loja online para uma oficina de mecânica automóvel e agrícola sendo a função dela, alem de fazer publicidade, demostra os serviços na oficina como é cada serviço, para também otimizar vendas assim um utilizador fazer a encomenda que deseja e a encomenda pode ser levantada na loja.

**Índice**

[Introdução 10](#_Toc171507349)

[Desenvolvimento 10](#_Toc171507350)

[Descrição técnica 11](#_Toc171507351)

[Ferramentas 12](#_Toc171507352)

[Linguagens de programação 13](#_Toc171507353)

[Desenvolvimento do projeto 13](#_Toc171507354)

[Área do Utilizador 14](#_Toc171507355)

[Área do Administrador 36](#_Toc171507356)

[Análise do Sistema 50](#_Toc171507357)

[Dificuldades 50](#_Toc171507358)

[Avaliação 51](#_Toc171507359)

[Conclusão 52](#_Toc171507360)

[Bibliografia 53](#_Toc171507361)

[Anexo A - Listagem do material utilizado 54](#_Toc171507362)

[Figura 1 – Tree do projeto em desenvolvimento 13](#_Toc171497737)

[Figura 2 - inicio da sessão 14](#_Toc171497738)

[Figura 3 – ficheiro .htaccess 14](#_Toc171497739)

[Figura 4 – init.php (app) 15](#_Toc171497740)

[Figura 5 – .htaccess (app) 15](#_Toc171497741)

[Figura 6 – classe app 15](#_Toc171497742)

[Figura 7 – classe app(continuação) 16](#_Toc171497743)

[Figura 8 – Controlador da home 16](#_Toc171497744)

[Figura 9 – função para fazer debug da aplicação no browser 16](#_Toc171497745)

[Figura 10 – debug do que foi escrito na url 17](#_Toc171497746)

[Figura 11 – parâmetros da url (classe app(continuação)) 17](#_Toc171497747)

[Figura 12 -Exemplo de view 18](#_Toc171497748)

[Figura 13 – Função de controlo de carregamento das views 18](#_Toc171497749)

[Figura 14 – Função de erro de página 18](#_Toc171497750)

[Figura 15 – Controlador da home atualizado 19](#_Toc171497751)

[Figura 16 – página teste no browser 19](#_Toc171497752)

[Figura 17 – Atualização da classe app.php(core) 20](#_Toc171497753)

[Figura 18 – Atualização da classe app.php(core) continuação 20](#_Toc171497754)

[Figura 19 – vista da tree pasta controladores separados 21](#_Toc171497755)

[Figura 20 – tree da pasta publica com os assets 22](#_Toc171497756)

[Figura 21 – exemplo dos includes 22](#_Toc171497757)

[Figura 22 – como os includes são chamados dentro das páginas 23](#_Toc171497758)

[Figura 23 – ficheiro configurações globais do sistema 23](#_Toc171497759)

[Figura 24 – xampp 24](#_Toc171497760)

[Figura 25 – phpmyadmin 24](#_Toc171497761)

[Figura 26 – tabela users 25](#_Toc171497762)

[Figura 27 – conexão PDO do php 26](#_Toc171497763)

[Figura 28 – instancia de conexão à base d e dados 26](#_Toc171497764)

[Figura 29 – função ler 27](#_Toc171497765)

[Figura 30 – função escrever 27](#_Toc171497766)

[Figura 31– registo do utilziador 28](#_Toc171497767)

[Figura 32 – view do registo 28](#_Toc171497768)

[Figura 33– controlador do registo 29](#_Toc171497769)

[Figura 34- função model 29](#_Toc171497770)

[Figura 35 – class model user registo 30](#_Toc171497771)

[Figura 36- função model registo (continuação) 30](#_Toc171497772)

[Figura 37 - função model registo (continuação) 31](#_Toc171497773)

[Figura 38 - função model registo (continuação) 31](#_Toc171497774)

[Figura 39 – registo na base de dados 31](#_Toc171497775)

[Figura 40– exemplo de erro promovido pelo modelo 32](#_Toc171497776)

[Figura 41 – função de erro criada na pasta de funções globais 32](#_Toc171497777)

[Figura 42 – função de controlo de autenticação 33](#_Toc171497778)

[Figura 43 – exemplo função de controlo de autenticação utilizador regular 34](#_Toc171497779)

[Figura 44 – exemplo função de controlo de autenticação do administrador 34](#_Toc171497780)

[Figura 45 – exemplo função de controlo de autenticação utilizador administrador 35](#_Toc171497781)

[Figura 46- menu com acesso ao administrador 35](#_Toc171497782)

[Figura 47 – admin dashboard 36](#_Toc171497783)

[Figura 48 – admin dashboard 36](#_Toc171497784)

[Figura 49 – ajax logo 37](#_Toc171497785)

[Figura 50– registo de categoria 37](#_Toc171497786)

[Figura 51 – código do modal do registo de categoria 38](#_Toc171497787)

[Figura 52– função get data 38](#_Toc171497788)

[Figura 53 – exemplo de erro 39](#_Toc171497789)

[Figura 54 – função send\_data 39](#_Toc171497790)

[Figura 55 – função category no controlador categories 40](#_Toc171497791)

[Figura 56 – modelo categories 40](#_Toc171497792)

[Figura 57 – função create no modelo category 41](#_Toc171497793)

[Figura 58– função category no controlador categories tratamento da response 41](#_Toc171497794)

[Figura 59 – função handle\_result, para dar a resposta do resultado ao utilizador 42](#_Toc171497795)

[Figura 60 – exemplo da resposta do resultado ao utilizador 42](#_Toc171497796)

[Figura 61 – carregar os dados da tabela categories no index 43](#_Toc171497797)

[Figura 62 – função no modelo category para obter os dados da tabela categories 43](#_Toc171497798)

[Figura 63 – exemplo de mostrar os dados organizados numa tabela em html 44](#_Toc171497799)

[Figura 64 – tabela com os dados organizados 44](#_Toc171497800)

[Figura 65 – exemplo de categoria desativada 44](#_Toc171497801)

[Figura 66 – exemplo de remover categoria, mas o statua enabled 45](#_Toc171497802)

[Figura 67 – exemplo de editar categoria 45](#_Toc171497803)

[Figura 68 – exemplo de apagar categoria 46](#_Toc171497804)

[Figura 69 – exemplo de adicionar produto 47](#_Toc171497805)

[Figura 70 – exemplo de editar produto 47](#_Toc171497806)

[Figura 71 – exemplo de remover produto 48](#_Toc171497807)

[Figura 72 – exemplo de tabela de produtos 48](#_Toc171497808)

[Figura 73– exemplo shop no utilizador 49](#_Toc171497809)

**Notação** e **Glossário**

|  |  |
| --- | --- |
|  | World Wide Web  FrontEnd- O Front-end está muito relacionado com a **interface gráfica** do projeto. Ou seja, é onde se desenvolve a aplicação com a qual o utilizador irá interagir diretamente, seja em softwares, sites, aplicativos, etc.  BackEnd- As linguagens de programação backend lidam com os bastidores das funcionalidades das aplicações web. É o código que conecta a internet com a base de dados, gerencia as conexões dos utilizador e alimenta a aplicação web. O backend trabalha em conjunto com o frontend para entregar o produto para o utilizador.  AJAX- AJAX significa Asynchronous JavaScript and XML, ou JavaScript e XML Assíncronos. é um conjunto de técnicas de desenvolvimento voltado para a web que permite que aplicações trabalhem de modo assíncrono, processando qualquer requisição ao servidor em segundo plano.  CSS- CSS é a sigla para o termo em inglês Cascading Style Sheets que, traduzido para o português, significa Folha de Estilo em Cascatas. O CSS é fácil de aprender e entender e é facilmente utilizado com as linguagens de marcação HTML ou XHTML.  HTML- A Linguagem de Marcação de Hipertexto (HTML) é uma linguagem de programação que compõe a maior parte das páginas da internet e das aplicações online. Um hipertexto é um texto usado para fazer referência a outros textos, enquanto uma linguagem de marcação é composta por uma série de marcações que dizem para os servidores da web qual é o estilo e a estrutura de um documento.  PHP-O PHP (um acrônimo recursivo para PHP: Hypertext Preprocessor) é uma linguagem de script open source de uso geral, muito utilizada, e especialmente adequada para o desenvolvimento web e que pode ser embutida dentro do HTML.  CRUD- Trata-se de um grupo de comandos de linguagem SQL que é usado para recuperar, incluir, remover e modificar informações dentro de uma base de dados  JSON- O formato JSON é usado para estruturar dados em formato de texto e permitir a troca de dados entre aplicações de forma simples, leve e rápida.  Xmlhttprequest- XMLHttpRequest é um objeto que fornece funcionalidade ao cliente para transferir dados entre um cliente e um servidor. Ele fornece uma maneira fácil de recuperar dados de um URL sem ter que fazer uma atualização de página inteira. Isso permite que uma página da Web atualize apenas uma parte do conteúdo sem interromper o que o utilizador esteja a fazer. XMLHttpRequest é usado constantemente na programação de [AJAX](https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Client-side_web_APIs/Fetching_data). |
|  |  |
|  |  |
|  |  |

«Esta secção apresenta os conceitos (glossário de termos) ordenados alfabeticamente, símbolos matemáticos e acrónimos **utilizados** no corpo do texto do relatório.»

# Introdução

O desenvolvimento do projeto de aptidão profissional foi inspirado na oficina de carros e máquinas agrícolas do meu pai, com o objetivo de otimizar as vendas da oficina. Durante o processo, surgiram várias ideias, já que atualmente acompanho meu pai no seu trabalho e tinha uma noção clara do que era necessário.

Decidi criar uma loja online para alinhar a oficina com as tendências atuais de marketing, permitindo não apenas vendas na loja, mas também encomendas online e publicidade pela internet. A minha motivação para o projeto foi ajudar meu pai na sua empresa, criando uma plataforma que mostra como os serviços funcionam e facilita o acesso a eles. A oficina é especializada em mecânica agrícola, uma área que me interessa bastante.

Meu objetivo com este projeto é, um dia, tornar a loja online pública na internet, contribuindo para o crescimento e modernização da oficina.

# Desenvolvimento

Para dar início ao ao projecto, considerei importante envolver o meu pai, tratando-o de forma profissional, como cliente, para entender de fato suas expectativas. Começamos com reuniões para levantar todos os requisitos necessários para desenhar o sistema. No início, esta fase foi a mais desafiadora, pois envolveu a recolha e organização de muitas informações.

A ideia original era ter um sistema que permitisse passar de forma clara todas as informações relativas ao negocio e para isso entende-mos que a pagina principal deveria ter apenas referencias ao tipo de serviço que eram propostos pela empresa.

Optamos por colocar imagens simples, pois consideramos que o sistema irá ser visto por várias pessoas de várias idades, logo, a nossa preocupação com a usabilidade do sistema foi sempre o ponto que consideramos importante para facilitar a pesquisa do nosso produto ao publico em geral.

Do tipo tradicional, consideramos que as cores deveriam ser em tons de azul, pois os grandes negócios utilizam estas cores, pois o ser humano responde de forma natural e é a cor mais atrativa em sistemas informáticos atuais.

A página principal seria também composta por um menu, onde o cliente poderia pesquisar várias ofertas do nosso sistema, nomeadamente a loja, serviços ou mesmo um wishlist.

Todo o sistema terá nas suas páginas um cart (carrinho de compras) onde os utilizadores poderão, imagine-se assim que entrem na loja, adicionar algum produto e passar depois para uma área especifica para finalizar a compra.

Consideramos importante o utilizador ter uma área pessoal onde pudesse alterar os seus dados de perfil, bem como consultar o seu histórico de compras realizadas, bem como o método de pagamento.

Como todo o sistema necessita de uma área em que seja feita a sua administração, por exemplo, a colocação de artigos na página principal, consulta ou manutenção da área utilizador, sugeri ao meu cliente uma área CRM (Gestão de Relacionamento com o Cliente) para facilitar o manuseamento do sistema.

# Descrição técnica

Para o desenvolvimento da aplicação em questão utilizei um paradigma de desenvolvimento OOP-programação orientada a objetos, construindo uma framework tipo MVC ( Model, View, Controller).

A utilização deste tipo de desenvolvimento possibilita atribuição de responsabilidades a cada parte do projeto. Numa breve descrição podemos identificar os seguintes pontos que fazer parte deste acrónimo:

* Modelo(M): manipula as conexões do base de dados e passa informações para o controlador.
* View(V): este manipulador das visualizações do utilizador, como templates no front-end.
* Controlador(C): este manipulador controla toda a lógica negociável entre todas as partes, o Modelo e as Visualizações.

O MVC, resumidamente, são frameworks que seguem o padrão de, ao aceder a uma URL, a framework direciona a solicitação a um controlador específico, que então chama um método que executa uma ação correspondente, como exibir uma página ou processar um formulário.

* Controlador: A parte que manipula a lógica de fluxo da aplicação.
* Método: Função dentro do controlador que responde a uma ação específica.
* Ação: A tarefa específica que o método do controlador executa, geralmente em resposta a uma solicitação do utilizador.

# Ferramentas

Para a realização do projeto, comecei por instalar as seguintes ferramentas:

* Visual Studio Code (VSC);
* Xammp;
* Repositório GitHub
* Git

O visual studio code é uma ferramenta de desenvolvimento (IDE) [1], que serve para desenvolvimento de código utilizando várias linguagens de programação. Para o desenvolvimento do meu projeto, vou utilizar a linguagem de desenvolvimento web PHP.

A princípio é uma ferramenta muito simples, mas possui uma loja de extensões imensa, e que continua crescer.

Ou seja, com essa enorme coleção de extensões, podemos adicionar diversas funcionalidades ao VS Code de forma bem simples.

Como citado na referência [2], o xammp é um pacote de software livre que consiste em um servidor web Apache, um banco de dados MySQL, PHP e Perl. O nome XAMPP é um acrônimo para as iniciais dos componentes que o compõem: X (para qualquer sistema operacional), Apache, MySQL, PHP e Perl. Ele foi desenvolvido para ser uma solução fácil de instalar e configurar, permitindo que aos utilizadores que tenham um ambiente de desenvolvimento web completo em seu computador local.

O repositório GitHub: [3] serviu para dar resposta ao alojamento das várias versões que eu ia construindo no meu projeto. Esta plataforma oferece um desenvolvimento colaborativo que aloja projetos na nuvem utilizando o sistema de controle de versões chamado Git.

A tecnologia Git é um sistema de controle de versões para guardar as versões do projeto e quando ligado a rede enviar para a cloud como já havia mencionado.

# Linguagens de programação

No desenvolvimento do meu projeto utilizei várias linguagens de programação e scripting, nomeadamente:

* Scritting:
* HTML
* CSS
* PHP
* Javascript

O Html, também conhecido por ser uma linguagem de programação mais básico na construção de Web.

O CSS é igualmente útil para a estética do trabalho aplicando um estilo ao conteúdo do html. Pode ser implementado um template, podemos deixar o front-end mais percetível para ter um aspeto mais versátil para o utilizador.

Já o JavaScript como descrito na referência [4] é uma linguagem de programação que permite a você criar conteúdo que se atualiza dinamicamente, controlar a multimédia, imagens animadas, e tudo o mais que há de interessante.

Por fim a utilização do PHP, permitiu me contruir a minha framework de raiz, onde pude estabelecer relações entre as várias partes do projeto, sendo nas views, como o controlador e no modelo. Resumidamente, foi a linguagem principal que utilizei para estabelecer a comunicação necessária entre todas as partes do meu projeto.

# Desenvolvimento do projeto

Inicialmente, comecei por instalar todas as ferramentas necessárias identificadas anteriormente.

Criei uma conta no pessoal no GitHub [3] para puder alojar todo o meu projeto de uma maneira segura e puder sempre verificar todos os passos que ia dando sempre que adicionava alguma atualização ao meu projeto.

Após o repositório público criado, que pode ser consultado em [5], comecei por puxar a versão ‘main’, para o meu computador local e dar inico ao desenvolvimento do projeto.

Toda a informação, ou documentação que ia gerando, foi sempre sido atualizada através do sistema disponibilizado de ‘commits’, que permitia assim ir documentando todo o meu projeto de uma forma automática.

Ao iniciar o desenvolvimento, criei um branch(ramo) do meu projeto no repositório, porque as boas práticas indicam que nunca se deve trabalhar no repositório ‘main’, pois este pode ser considerado o repositório de produção.

De referir que o meu projeto vai ser construído em duas partes distintas como já mencionado anteriormente, nomeadamente a parte do utilizador e a parte do administrador, assim optei igualmente por dicidi o meu relatório, por considerar importante esta descrição.

# Área do Utilizador

Assim sendo, iniciei um novo branch, com um nome de ‘develop’ e comecei por criar alguns ficheiro e pastas, que passarei de momento a explicar.

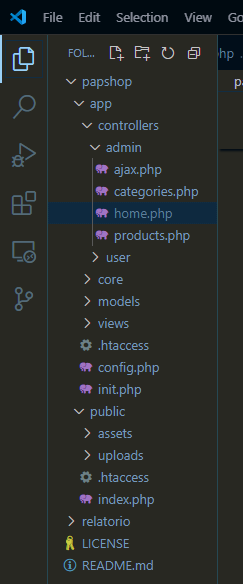


Figura 1 – Tree do projeto em desenvolvimento

A pasta com o nome “PAP” e dentro dessa pasta criei o meu projeto com o nome “papshop”. Dentro do projeto criei duas pastas “app” (backend) e “public” (frontend).

Dentro da pasta publica criei mais dois arquivos “índex.php”,”.htaccess”.

No índex.php” é onde se inicia a sessão em nosso navegador e incluímos na pasta “app” para instanciar um objeto da nova função APP () que é onde vai arrancar a nossa framework.

Irá também iniciar uma sessão no browser que utilizar-mos.

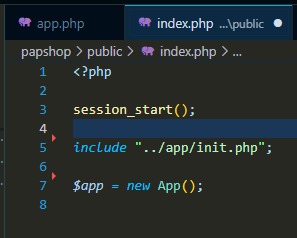


Figura 2 - inicio da sessão

No arquivo “.htaccess” é onde um conjunto de regras do manipulador direciona todos os URLs solicitações que não correspondem a um arquivo ou diretório existente ao arquivo index.php, permitindo que o script PHP processe a URL e decida como lidar com a solicitação.

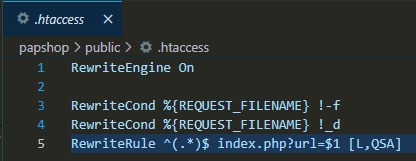


Figura 3 – ficheiro .htaccess

Como podemos observar na tree do projeto, dentro da pasta “APP” criei 2 arquivos “init.php”,”.htaccess” e quatro pastas “controllers”,”core”,”models”,”views”.

Dentro do arquivo “init.php” é onde identifico todos os caminhos para as principais pastas do meu projeto.

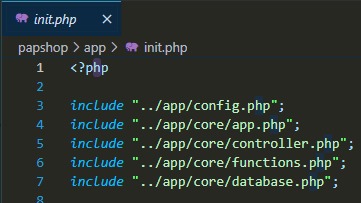


Figura 4 – init.php (app)

No arquivo “.htaccess” a opção Options -Indexes estiver presente, o servidor web não mostrará uma lista de arquivos quando alguém aceder a um diretório no navegador e não houver nenhum arquivo de índice presente. Esta configuração é uma medida de segurança comum para impedir que visitantes vejam a estrutura de diretórios do site e tenham acesso aos arquivos que não deveriam ser públicos.

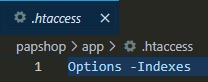


Figura 5 – .htaccess (app)

Com o propósito de iniciar os testes de conectividade e funcionalidade na minha framework, comecei por criar dentro da pasta core um ficheiro app.php. Este ficheiro irá fazer toda a manipulação de conteúdos que é escrita na url do nosso browser, como já identificado anteriormente, na explicação do MVC (Model, View, Controller).

Assim sendo, a fim de facilitar a explicação irei proceder a uma breve explicação por partes.

A instanciação da classe APP é onde todo o projeto arranca.

Comecei por definir 3 variáveis, que irão ajudara manipular a url.

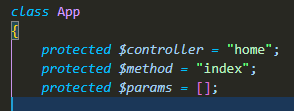


Figura 6 – classe app

No código seguinte é onde irá ser feita a leitura do controlador, ou seja, inicialmente o utilizador irá carregar em um href no html e ele irá verificar se o controlador, para onde o href esta apontado existe e irá colocar em letras pequenas e agarrar em um array a primeira posição do índex da variável $url.

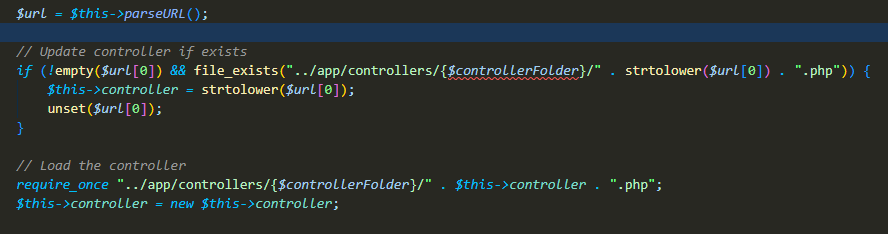


Figura 7 – classe app(continuação)

Com o objetivo de testar a primeira parte do código gerado, criei um controlador como nome” Home” para conseguir fazer debug do que estava a fazer no meu projeto. Este controlador, irá servir para no futuro carregar a minha view do índex.php.

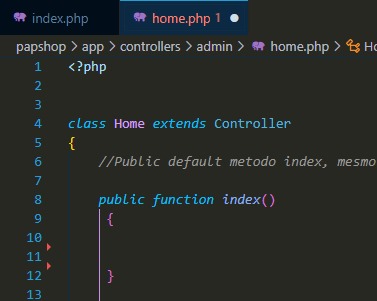


Figura 8 – Controlador da home

Para facilitar o debug, criei uma função show () dentro da pasta functions do meu projeto, que irá servir para todas as funções globais da framework.

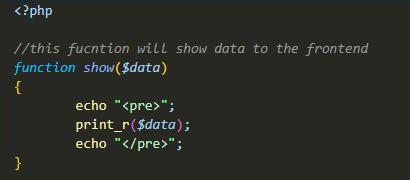


Figura 9 – função para fazer debug da aplicação no browser

O próximo passo, foi abrir o xammp e iniciar o servidor apache para testar o desenvolvido até ao momento.

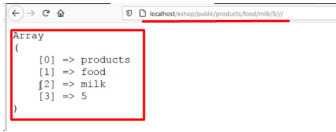
Ao inserir alguns dados na url, este foi o resultado obtido:  
  


Figura 10 – debug do que foi escrito na url

Assim, como eu tinha identificado, anteriormente, neste momento os dados da url estão a ser colocados dentro de um array e são separados em cada índex, cada vez que é identificado que existe um barra (/) ele automaticamente faz o split e insere os dados em cada posição.

O próximo passo é apanhar o método. Para isso adicionei ao código o seguinte excerto:

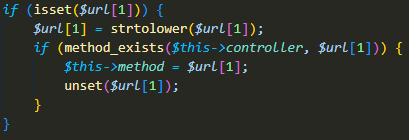


Figura 11 – parâmetros da url (classe app(continuação))

Neste excerto de código é onde verifico se o método existe e, se existir, ler o método, caso não exista, enviará o utilizador para uma página de erro. Para isso, é criado um sistema que verifica se a URL contém um segundo segmento, que geralmente é usado para especificar um método dentro do controlador. Se o método existir no controlador, ele será definido como o método atual e o segmento correspondente da URL será removido. Isso permite chamar métodos específicos dentro de controladores com base na URL

Com todo o sistema testado e a funcionar, foi hora de introduzir as views para testar se tudo estava conectado.

Para isso dentro da pasta views, criei um ficheiro índex.php, e inseri algum texto aleatório para teste.

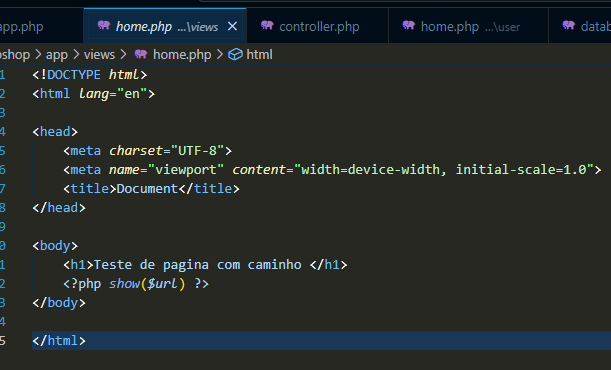


Figura 12 -Exemplo de view

Então, seguindo o paradigma de construção, todos os controladores devem se iniciar com o método índex, é neste método onde vai ser carregada na view.

Para isso, foi criado uma função publica, que irá tratar desse processo:

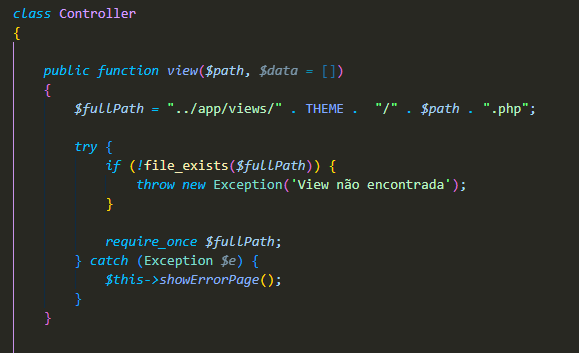


Figura 13 – Função de controlo de carregamento das views

Esta função esta contruída para carregar as views do projeto. Simplesmente, ela está apontando para o diretório das views e se encontrar o solicitado, mostra, caso a view não seja encontrada, irá disparar um erro, que vai chamar a pagina 404.php. No escopo da função já estou a contemplar o array data que ira enviar os dados para as views no futuro.

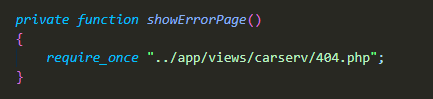


Figura 14 – Função de erro de página

E o controlador foi então atualizado com a função a chamar a view de test



Figura 15 – Controlador da home atualizado

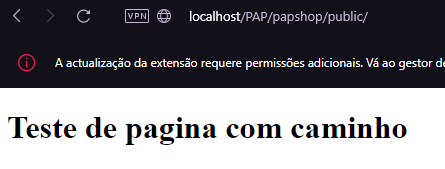


Figura 16 – página teste no browser

Com todo o sistema de carregamento de views testado e funcional, e como boa prática e seguindo o paradigma, visto que o projeto consiste em um webstore, necessito de adicionar ao meu projeto uma área de backend onde somente o utilizador autorizado a aceder para fazer a manipulação do site, como por exemplo a introdução na secção de vendas dos produtos e categorias.

Assim tive de manipular a class app, de forma a dividir o conceito para, que um dia quem viesse a agarrar neste projeto, tenha facilidade em perceber o que esta construído e como esta contruído o mais rapidamente possível.

Então, tive a necessidade de alterar o código na classe app, para proceder ao carregamento, fora dos controladores do user, ou do admin em pastas separadas nos controladores. As alterações foram as seguintes:



Figura 17 – Atualização da classe app.php(core)

No inico desta atualização é muito idêntica com o que existia anteriormente, a alteração mais suscetível de reparo é somente a condição de fixar a url do controlador user numa variável, e fazer a verificação se existir, caso o controlador exista, ele carrega a pasta user controller, caso contrário, vai ao controlador admin pesquisar. Caso não exista em lado nenhum, dispara o 404.php como havíamos visto na função da view.

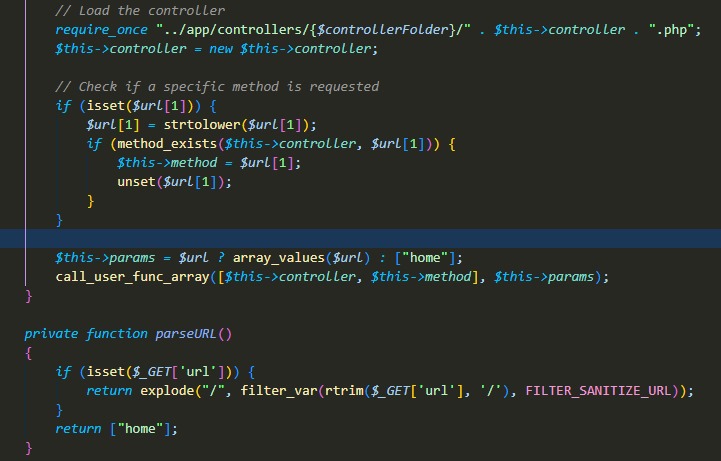


Figura 18 – Atualização da classe app.php(core) continuação

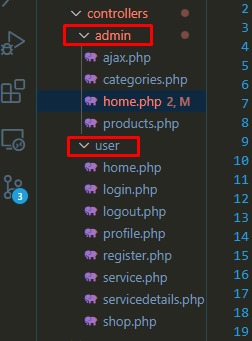


Figura 19 – vista da tree pasta controladores separados

O passo seguinte foi encontrar um template adequado para utilizar no projeto. Escolhi um template que utiliza a tecnologia Bootstrap para facilitar a implementação. Para evitar cópias não autorizadas, optei por fazer um pequeno investimento e adquiri um template no site [inserir nome do site]. A escolha do template seguiu os padrões de usabilidade que foram abordados anteriormente.

Visto que vou utilizar um CRM, tambem optei por aplicar um template [8] que fosse de encontro as espectativas necessárias e me pudesse facultar todas as ferramentas necessárias para auxiliar na contrução do sistema.

Visto que os templates possuem características diferentes, tive de adaptar o código na pasta publica para que os caminhos de carregamento dos assets podesse ser assegurada para que todas as librarias necessárias fossem carregadas.

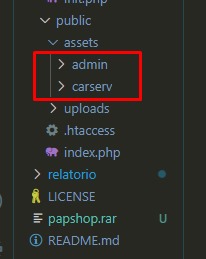


Figura 20 – tree da pasta publica com os assets

Dentro das pastas publicas, será o que vai estar disponível para os utilizadores comuns do sistema, caso optem por inspecionar a source do código, a restante logica esta protegida no backend.

Dentro do arquivo “controller.php” inseri também a seguinte função para manipular as ligações com os modelos.

Após a colocação dos templates, com o propósito de agilizar todo o processo e renderização do sistema, visto que vão existir partes comuns, decidi separar a parte do header, footer e menu de ambos templates, para facilitar a leitura e escrita do projeto e ter um código mais limpo e visível:

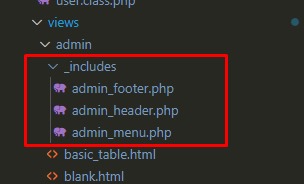


Figura 21 – exemplo dos includes

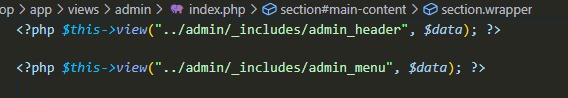


Figura 22 – como os includes são chamados dentro das páginas

O passo seguinte foi testar a conexão com a base de dados.

Para isso comecei por criar algumas variáveis publicas para facilitar todo o processo, visto que este tipo de implentação já esta definido para no futuro, dependente do servidor, tenha um acesso rápido a poder fazer todas as alterações necessárias num so local.

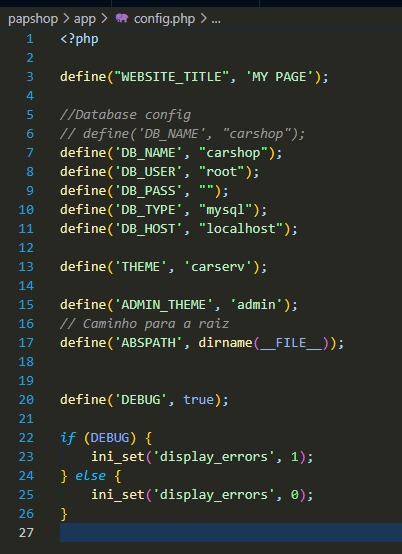


Figura 23 – ficheiro configurações globais do sistema

Na figura anterior podemos verificar que foram definidas várias variáveis consideradas globais, como a ligação à nossa base de dados, a definição automática do caminho dos assets e um caminho absoluto, sempre com a vertente de uma futura implementação em produção.

Concluído o processo de ligação à minha base de dados, eis que veio a hora de testar a ligação, e nada melhor do que isso, do que abrir no servidor xampp o mysql e através do phpmyadmin iniciar uma nova base de dados.

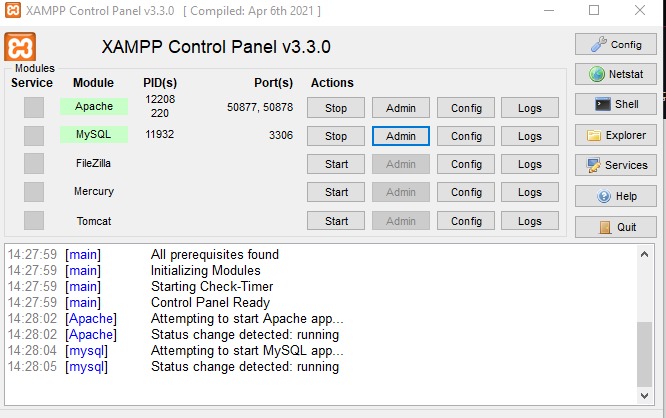


Figura 24 – xampp

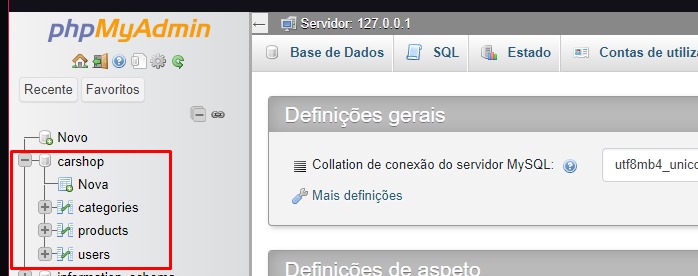


Figura 25 – phpmyadmin

Já com a base de dados criada, iniciei com a tabela utilizador (users) com os seguintes campos:

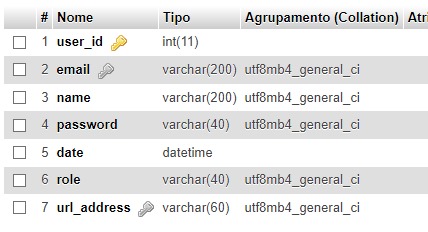


Figura 26 – tabela users

Na estrutura da bse de dados existe a tabela dos “users” que consiste em guardar os dados dos utilizadores dados esses que são:

* user\_id – o id vai ser a identificação do utilizador na base de dados e é preenchida automaticamente assim que é realizado um novo registo;
* email – servirá para guardar o email do utilizador registado;
* nome – servirá para guardar o nome do utilizador registado;
* password - servirá para guardar o palavra-passe do utilizador registado;
* data - servirá para guardar a data do registo do utilizador;
* role - varchar porque recebe uma frase como valor neste caso costumer ou admin para definir o papel do utilizador;
* url\_address – servirá para guardar um identificador único que vai ser usado para identificar o utilizador em todo o processo de manipulação de dados, em vez de utilizar o user\_id, tipo token;

Optei por iniciar com a introdução da tabela utilizadores, para dar resposta ao requisito de registo de utilizador e autenticação, bem como de logout do sistema.

Assim sendo, para dar resposta ao proposto, comecei por criar dentro do core, database algumas funções que me vao permitir realizar algumas operações dentro da base de dados.

Comecei por criar uma função onde vou realizar a ligação à base de dados utilizando o PDO.

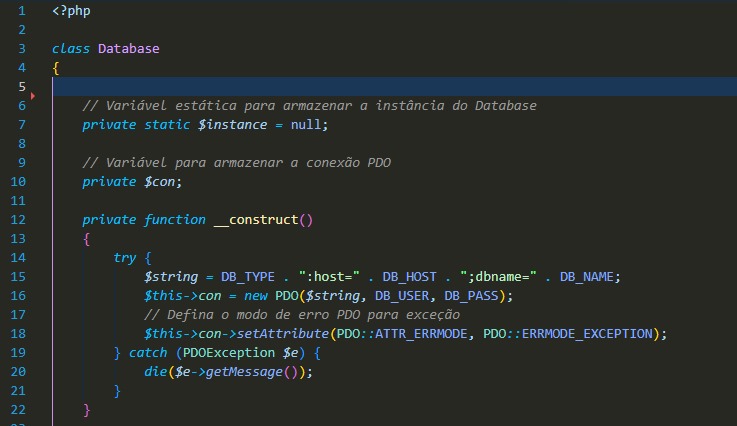


Figura 27 – conexão PDO do php

Para obter múltiplas conexões, tive de criar um sistema de instanciação de conexões, como podemos verificar no trexo seguinte:

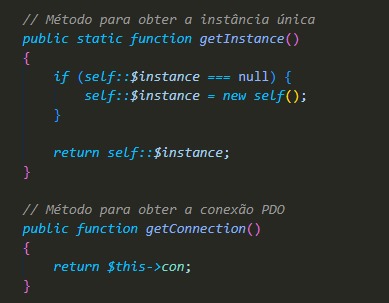


Figura 28 – instancia de conexão à base d e dados

Por fim, as funções que vão ler e escrever na minha base de dados:

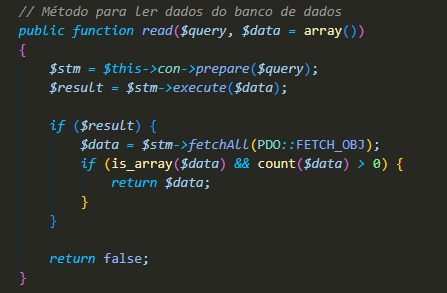


Figura 29 – função ler

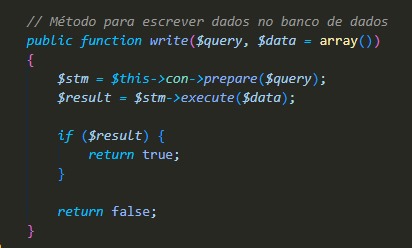


Figura 30 – função escrever

Proximo passo foi testar a ligação com um formulário do registo de utilizador:

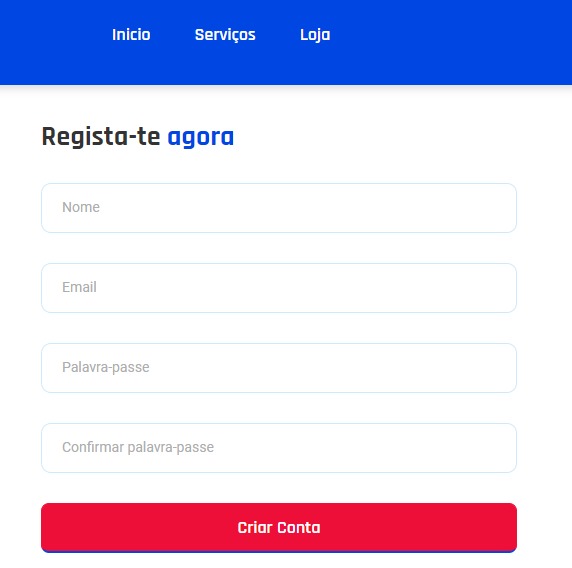


Figura 31– registo do utilziador

Numa primeira fase utilizei o método post, para realizar as requisições dos dados para o controlador, para fazer o registo e autenticação do utilizador.



Figura 32 – view do registo

O placeholder sublinhado já construído para que o utilizador em caso de erro, não tenha que voltar a escrever os dados novamente.

No controlador podemos observar que tenho uma nova função criada para chamar o modelo:

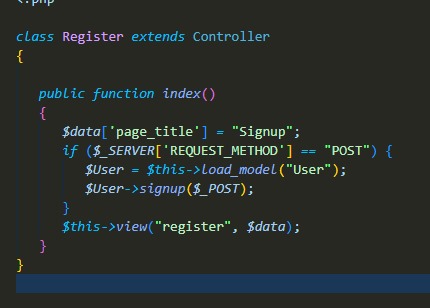


Figura 33– controlador do registo

A função load\_model, vai sempre que solicitada para recorrer a conexão do modelo onde os dados vão ser tratado antes de mandar para a base de dados.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 34- função model

Assim no modelo, podemos tratar todos os dados recebidos do formulário:



Figura 35 – class model user registo

Após recolher os dados do POST através da variável $data, passei a tratar os dados e a exibir alguns erros:

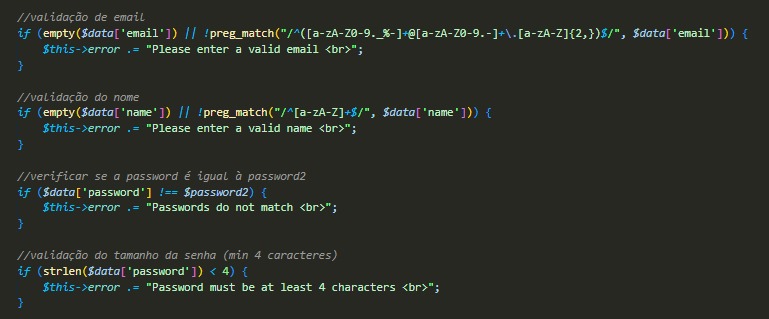


Figura 36- função model registo (continuação)

Aqui são feitos algumas validações, para que não sejam introduzidos dados que não devam na base de dados.

Assim sendo temos no primeiro ponto a validação do email, depois do nome, para somente serem introduzidos caracteres, e a password 1 deve ser igual à segunda e no minimop deve ter 4 caracteres. Poderia aqui explorar mais este campo, mas ficara para uma próxima atualização do sistema.

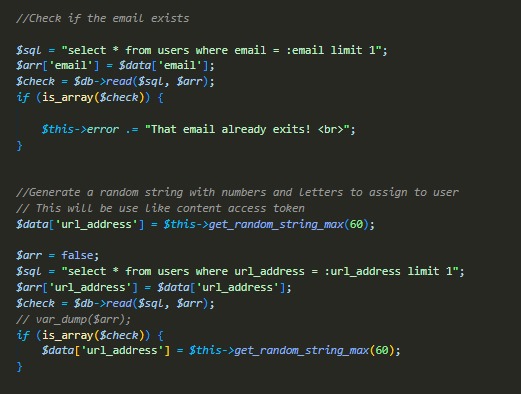


Figura 37 - função model registo (continuação)

Neste excerto, será feito uma validação que não existe um email repetido na base de dados e também o url\_address. Na questão da url\_address, caso exista, irá criar um novo.



Figura 38 - função model registo (continuação)

Por fim caso esteja tudo correto, os dados serão inserido na base de dados e um novo utilizador é criado e o reload da pagina é feito para o login.

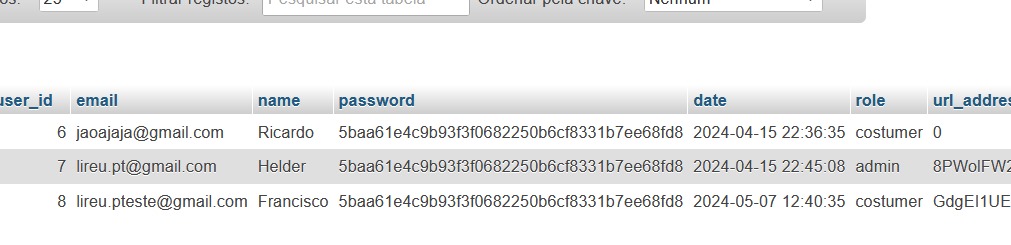


Figura 39 – registo na base de dados

Caso exista algum erro, criei uma função error para lidar com o alerta no frontend ao utilizador.

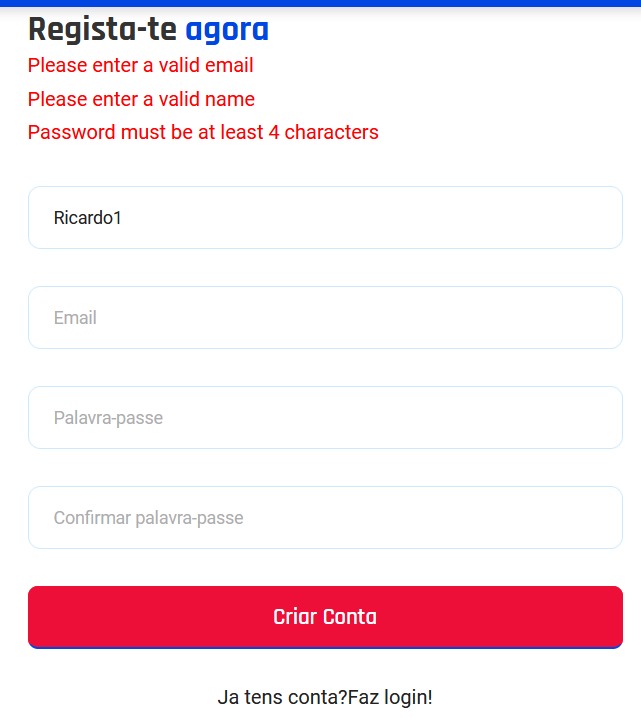


Figura 40– exemplo de erro promovido pelo modelo

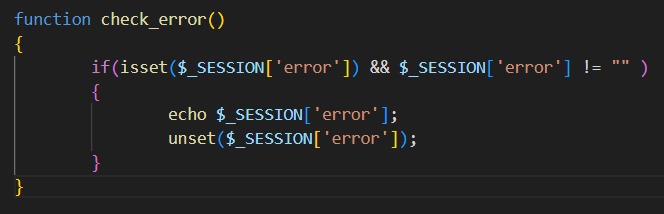


Figura 41 – função de erro criada na pasta de funções globais

Na sequência do registo, o modo de autenticação foi criado exatamente da mesma forma.

Seguidamente, na necessidade de promover segurança no sistema, foi criado uma função no modelo do user, para validar se o utilizador esta autorizado a aceder à uma área reservada.

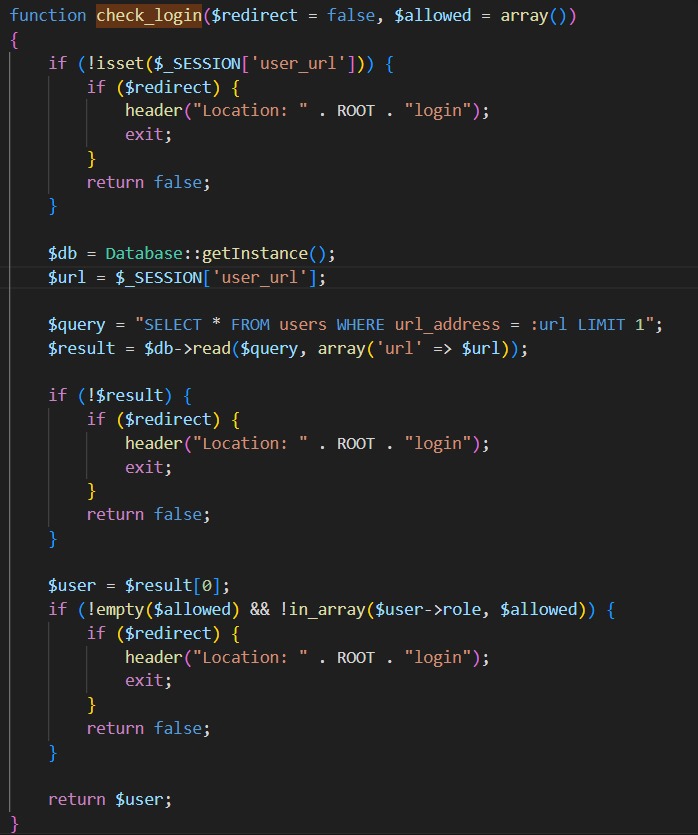


Figura 42 – função de controlo de autenticação

Caso o utilizador esteja registado, terá acesso, caso contrário voltar para a página de autenticação.

Foi introduzido também no escopo da função, um valor booleano, para validar a informação e também é possível receber um valor array que vai servir para definir o controlo de acesso entre utilizador regular, ou administrador.



Figura 43 – exemplo função de controlo de autenticação utilizador regular

No array user\_data servirá para enviar os dados para a view, como por exemplo o nome do utilizador, entres outros dados que sejam necessários.

Neste caso não utilizo a sessão para manipular os dados, pois os dados podem facilmente serem apanhados no browser por pessoas mal-intencionadas.



Figura 44 – exemplo função de controlo de autenticação do administrador

O passo seguinte, foi criar um acesso a área de administração do sistema. Assim, através de uma condição criada na view, podemos verificar em se o utilizador pode ou não aceder à área reservada.

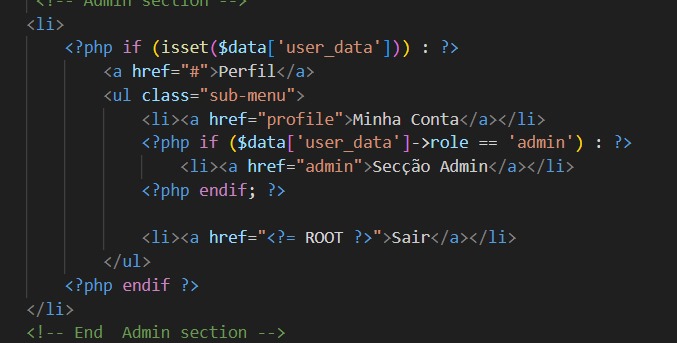


Figura 45 – exemplo função de controlo de autenticação utilizador administrador

Neste código, verificamos a existência de dados no user\_data e caso exista, vamos ler o campo do role, e caso seja admin, ou costumer, será mostrado o menu de acesso à área reservada.

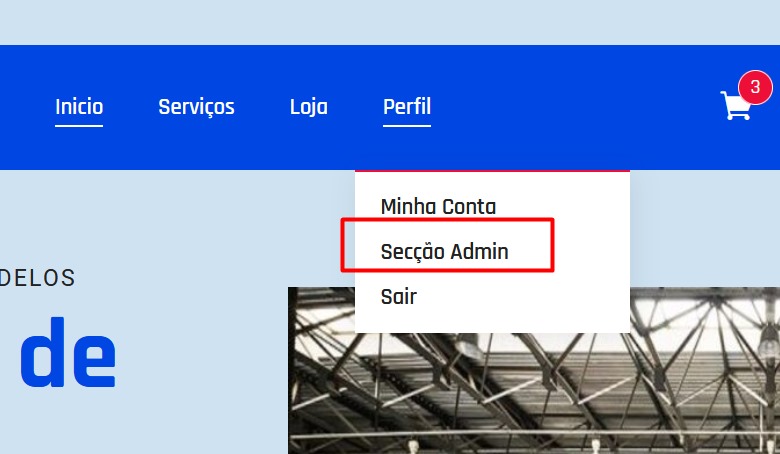


Figura 46- menu com acesso ao administrador

# Área do Administrador

Na secção do administrador, inicialmente limpei o template e comecei por recriar o menu, com as funcionalidades que considero necessárias para o meu projeto.

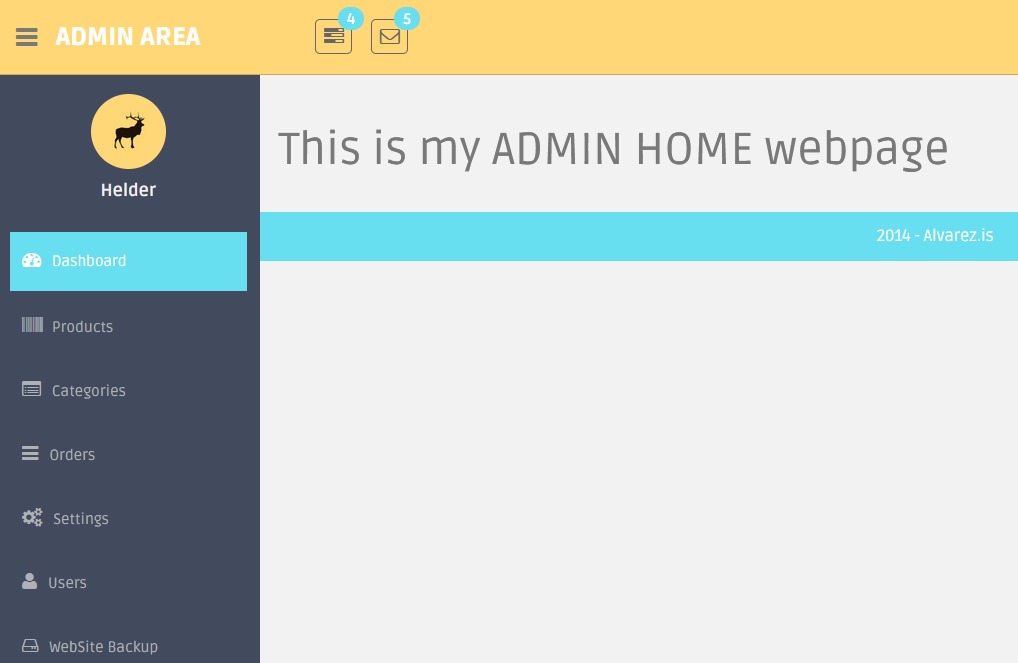


Figura 47 – admin dashboard

A imagem acima, podemos verificar, que utilizarei o menu para aceder às várias funcionalidades que pretendo implementar.

Comecei por implementar as categorias, devido ao relacionamento com a tabela produtos, visto que uma categoria, terá vários produtos.

Toda a demonstração a nível de relatório, será para exemplificar o CRUD implementado nas categorias, pois nos produtos, a diferença será na imagem que será adicionada ao produto.

Comecei então por adicionar a tabela categorias na base de dados.



Figura 48 – admin dashboard

À imagem dos utilizadores, na tabela categorias tenho:

* id – o id vai servir para a identificação da categoria na base de dados e é preenchida automaticamente assim que é realizado um novo registo;
* category – servirá para guardar o nome da categoria;
* disabled – servirá para gerir se a categoria esta ativa ou inativa;

Para fazer a manipulação os dados recolhidos do formulário, utilizei uma tecnologia diferente do registo e login, com o nome AJAX.



Figura 49 – ajax logo

Ajax é o uso metodológico de tecnologias como Javascript e XML, providas por navegadores, para tornar páginas Web mais interativas com o utilizador, utilizando-se de solicitações assíncronas de informações.

Este tipo de implementação permite utilizar as requisições dentro do mesmo ficheiro html, utilizando como componentes, em sem parado, como por exemplo modais, e a resposta do controlador será sempre de acordo com o solicitado, dando assim uma maior utilidade ao controlador, quando o passaremos a utilizar, não somente para o index, mas agora sim para os métodos quando solicitados.

Comecei então por criar o registro de uma categoria.

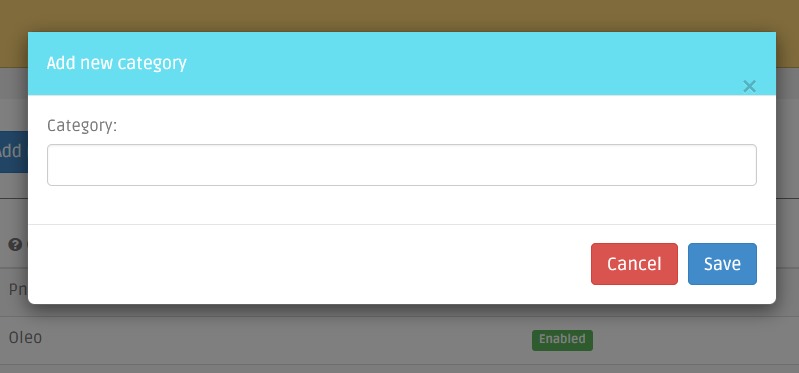


Figura 50– registo de categoria



Figura 51 – código do modal do registo de categoria

Utilizei o onclick para apanhar os dados dos campos com a função criada do get\_data().

Esta função encontra-se no fundo do htmal dentro das tags de script e utilizamos o javascript para fazer a manipulação de todo o processo.

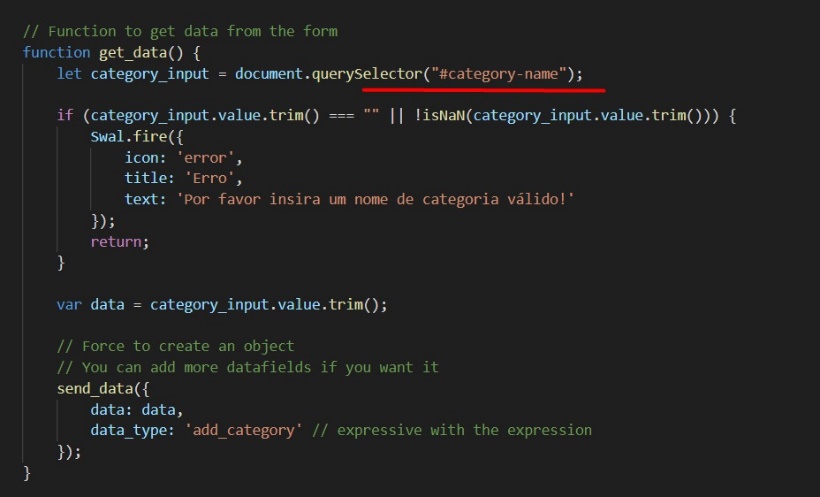


Figura 52– função get data

Numa breve explicação, esta função quando é chamada no onclick, vai ao campo com o id = category-name e vai ler o valor do campo. Após recolher os dados para um array, vai adicionar um data\_type, para fazer o controlo no controlador das categorias, neste caso estamos a adicionar uma nova categoria. Após este processo vai verificar se esta tudo correto, ou seja se não estiver o campo vazio e se não é um número. Caso seja erro, utilizamos a biblioteca do sweet alert para disparar um erro a dizer que algo não está bem.

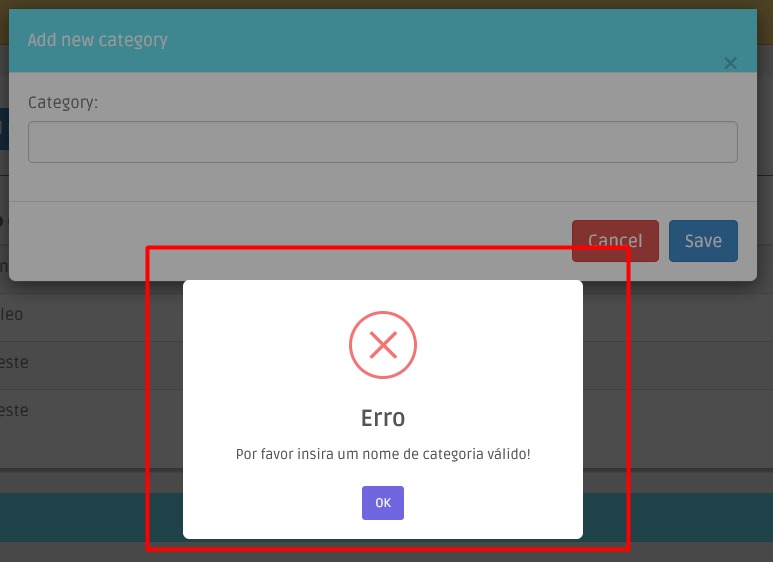


Figura 53 – exemplo de erro

Caso esteja tudo direito, passamos os dados através do AJAX para o controlador utilizando o formato JSON através recorrendo a funcionalidade xmlhttprequest fazendo o [7]decode aos dados.



Figura 54 – função send\_data

Já no controlador, recebemos os dados no array



Figura 55 – função category no controlador categories

Inicialmente, criamos um modelo, porque a atribuição de responsabilidades, é o paradigma mais importante desta construção.

Pós esse processo chamamos através do load\_model o modelo category, onde tratamos os dados antes de inseri-los na base de dados. Fazemos o encode dos dados, ou seja, abrimos os dados do array, vamos verificar o campo da data\_type, e caso esteja tudo correto, a variável $check irá estar a espera para receber o resultado enviado do modelo após chamar a função create.

Passaremos então para o modelo das categorias.

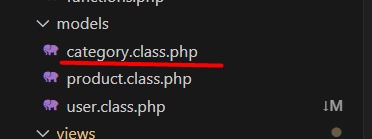


Figura 56 – modelo categories

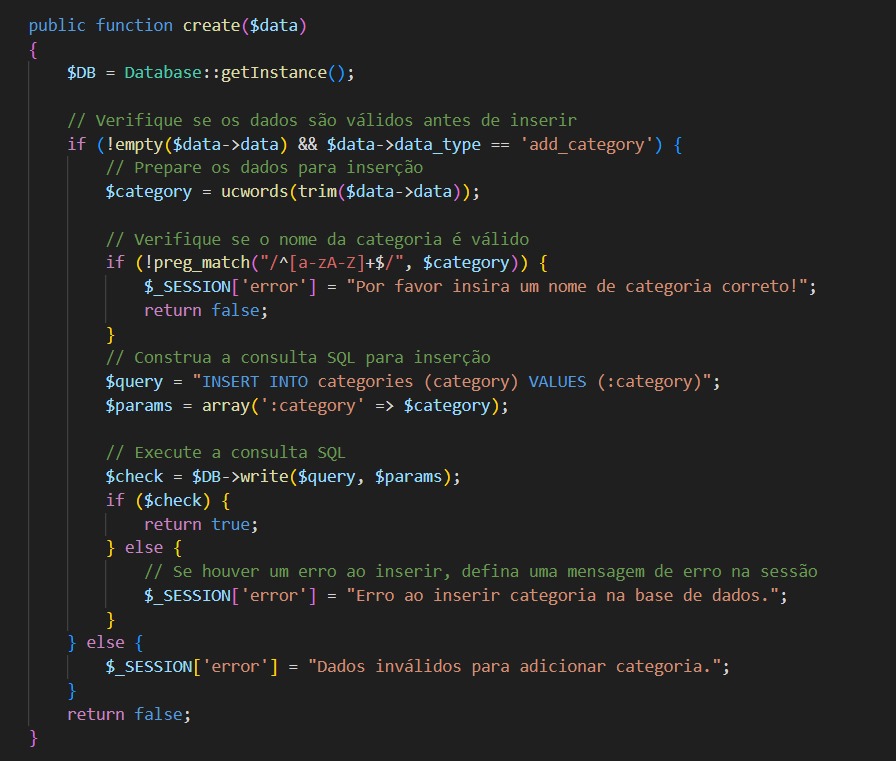


Figura 57 – função create no modelo category

Nesta função, recebemos os dados, e se existir um erro caso o dado a ser inserido não esteja correto, será enviado através da $check, caso esta tudo ok, será enviado também na mesma variável. Os erros de restantes serão enviados através da sessão, por ter um trato diferente.

Voltando ao controlador, recebemos a resposta e após o veredico, caso positivo, ou erro, assim o encode da resposta é feito para enviar para o lado da view:

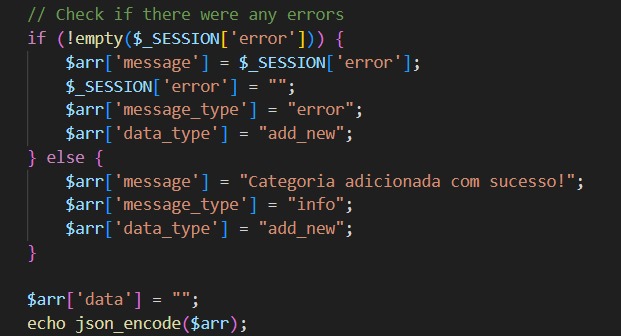


Figura 58– função category no controlador categories tratamento da response

Enviado para a view, terá uma um afunção de apoio com o nome handle\_response que vai soltar o swall do sweet alert com o sucesso ou o erro.

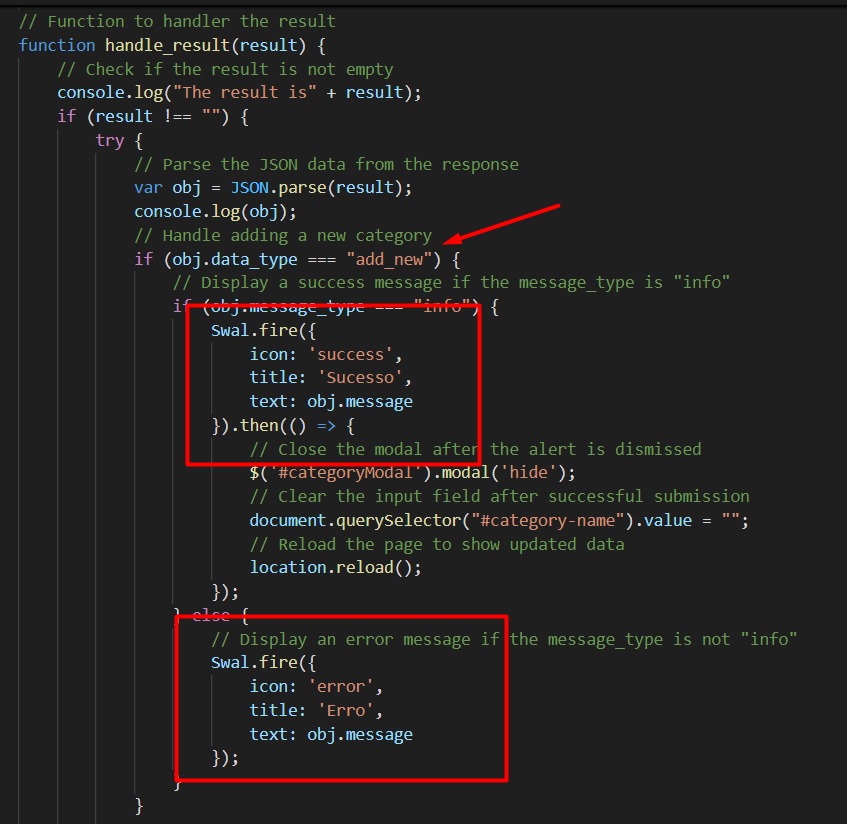


Figura 59 – função handle\_result, para dar a resposta do resultado ao utilizador

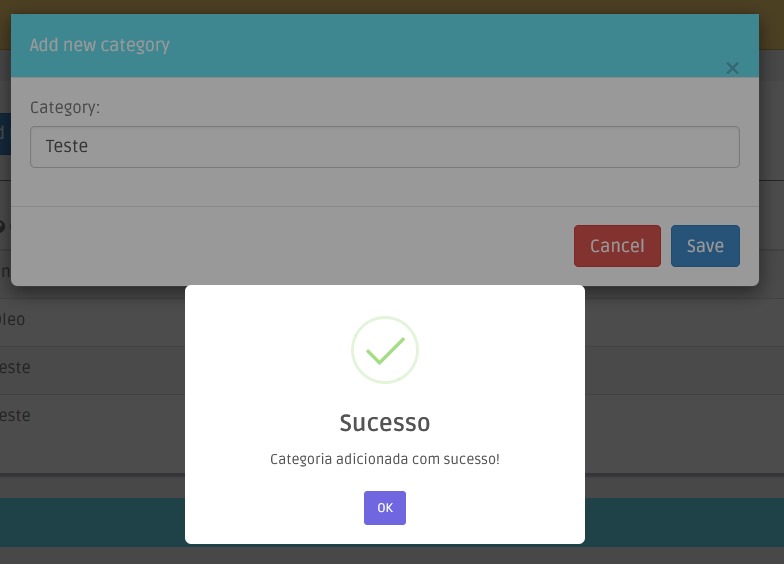


Figura 60 – exemplo da resposta do resultado ao utilizador

Após este processo, a tabela é automaticamente atualizada, através do dados chamados, com a função get\_categories que é carregada no index da página.

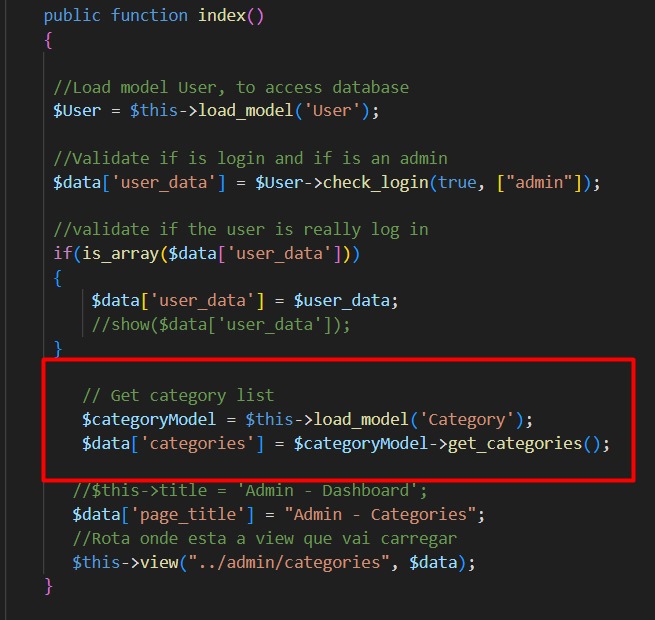


Figura 61 – carregar os dados da tabela categories no index



Figura 62 – função no modelo category para obter os dados da tabela categories

Para mostrar os resultados na view, utilizamos um foreach para iterar os dados obtidos atravaes do array data[‘categories’], e assim conseguimos dar um echo e ver os dados através do php.



Figura 63 – exemplo de mostrar os dados organizados numa tabela em html

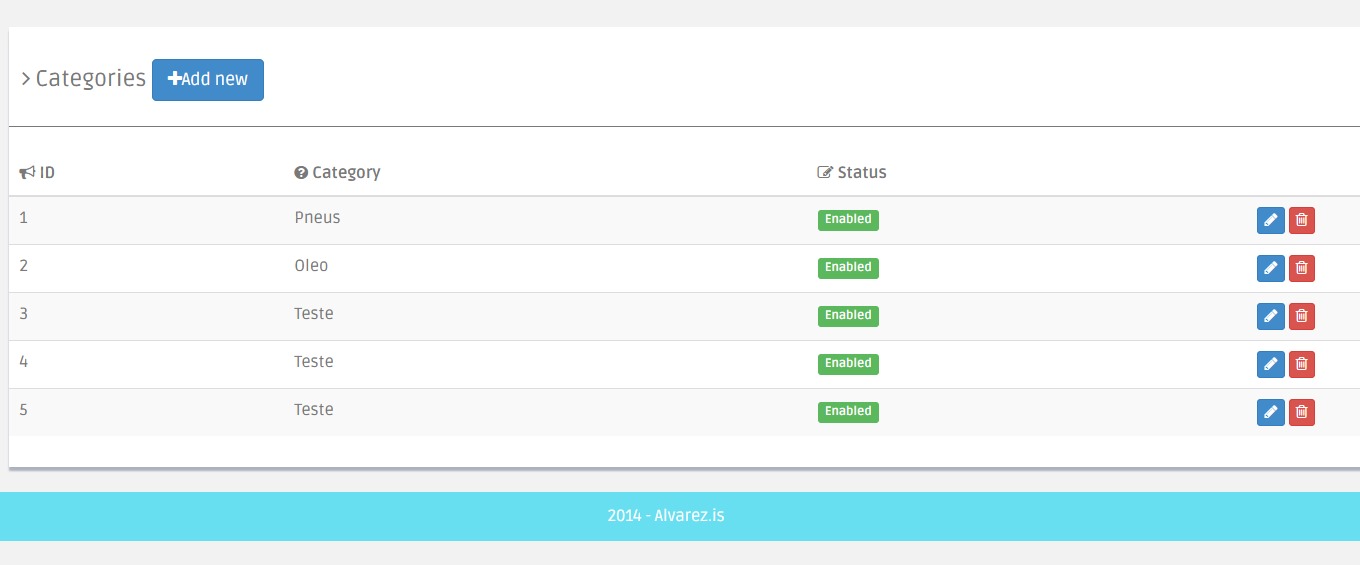


Figura 64 – tabela com os dados organizados

Algumas funcionalidades foram aplicadas, como por exemplo tentar remover uma categoria, quando ela esta ativa, vai soar um erro a dizer que não é permitido.

Editar a categoria nas ações e inclusive a troca de estado da categoria no status.



Figura 65 – exemplo de categoria desativada

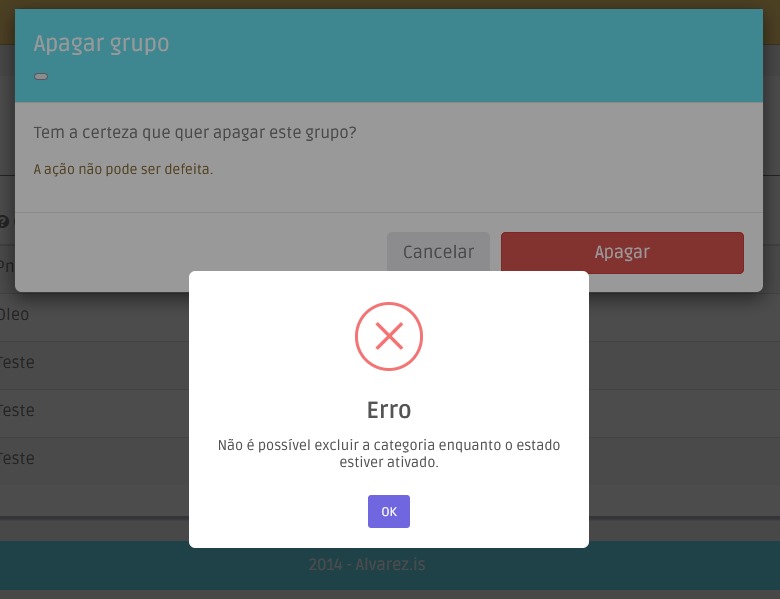


Figura 66 – exemplo de remover categoria, mas o statua enabled

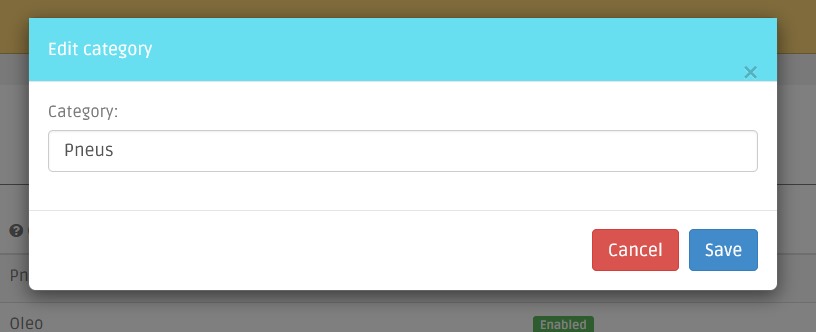


Figura 67 – exemplo de editar categoria

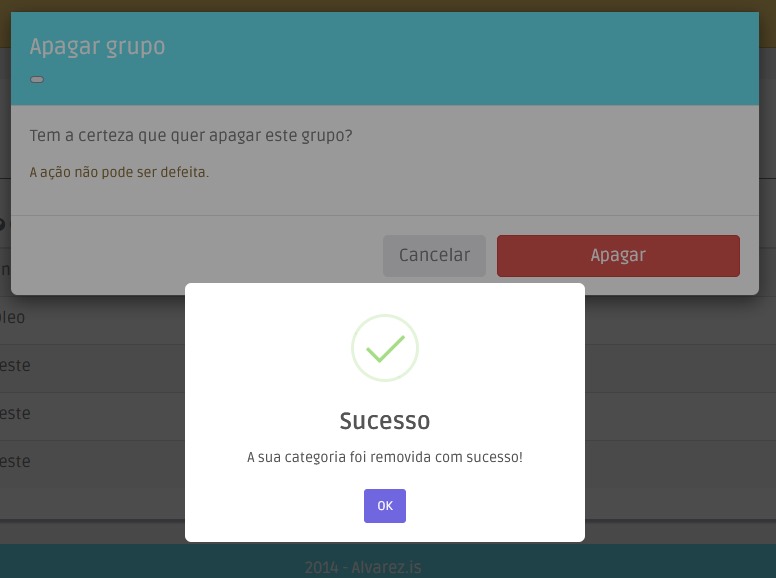


Figura 68 – exemplo de apagar categoria

Todo o processo de implementação, seguio o mesmo padrão do adicionar categoria, recorrendo sempre ao mesmo paradigma.

Recolher os dados do utilizador, trat-los a uma primeira vez na função get\_data, e depois passa-los para o controlador, então no controlador, verificar se está tudo bem com o array recebido, e passa-los para o modelo. Já no modelo, fazer nova verificação dos dados e se tudo tivre correto, enviar o ok para o controlador, que por sua vez irá gerar a resposta e fazer o encode da mesma e JSON para enviar para a função handle\_result que mostrar atrave´s do sweet alert a resposta ao utilizador.

Nos produtos a diferença é que adicionei a imagem e tive de alterar a recolha dos dados para tratar os dados como um objecto enviado em $\_POST devido a utilizar a variável reservada de $\_FILES do php para manipular as imagens.

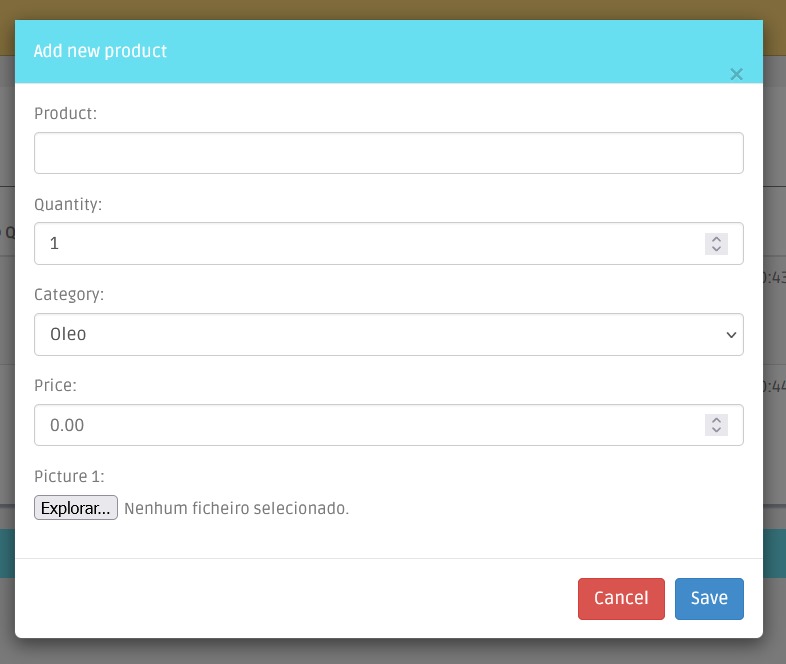


Figura 69 – exemplo de adicionar produto

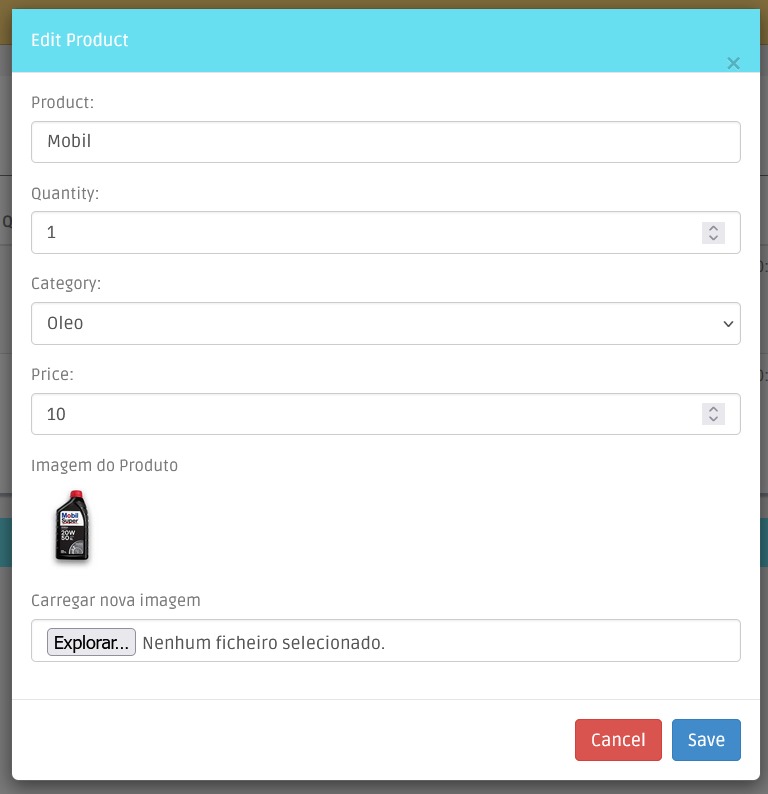


Figura 70 – exemplo de editar produto

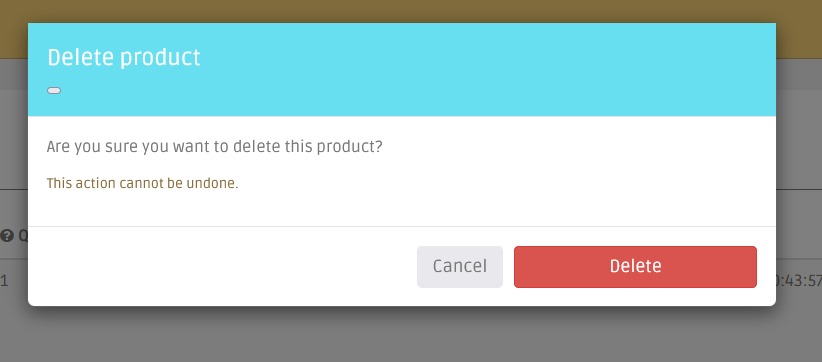


Figura 71 – exemplo de remover produto

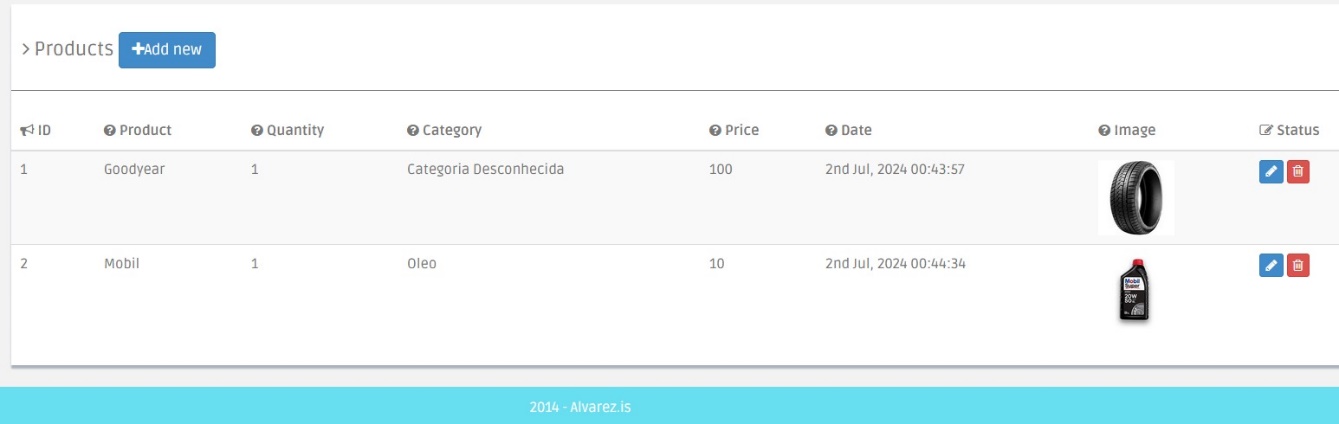


Figura 72 – exemplo de tabela de produtos

Por sua vez, na secção do utilizador, basta chamar a tabela com os produtos na shop.

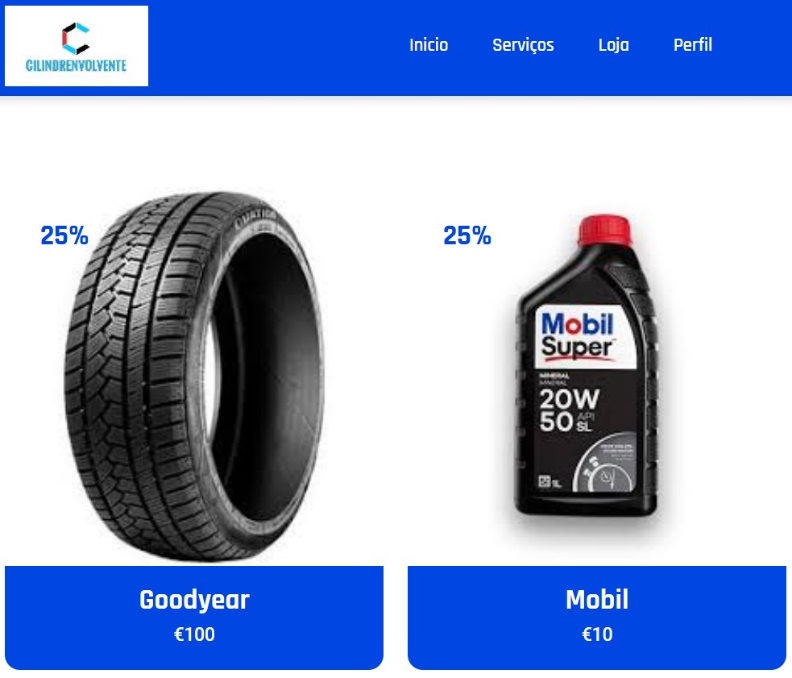


Figura 73– exemplo shop no utilizador

# Análise do Sistema

O sistema ainda se encontra uma fase muito embrionária de desenvolvimento, mas posso constatar com todo o mecanismo preparado, relativamente a controladores, de views e modelos, facilmente o projeto pode ser escalonável para um nível muito superior.

# Dificuldades

Tive algumas dificuldades em perceber algumas funcionalidades, pois os conteúdos a que me propus aprender são demasiado evoluídos para o que atualmente lecionamos no curso, contudo admito que a estratégia adotada enquadra-se dentro do espectável para que no futuro possa, em caso de trabalho ou aprofundamento de estudos, estar mais á vontade para discutir alguns pontos, que até ao momento desconhecia por completo.

# Avaliação

O sistema encontra-se num repositório público do GitHub em [6], e poderá ser feito o download e podemos utilizar o sistema, recorrendo a um servidor xampp.

# Conclusão

Ao realizar este projeto pude ampliar mais os meus conhecimentos sobre programação de sites, aprendi mais sobre a segurança a ter ao construir um site online para impedir vazamento de dados, organização a ter no programa facilitando a localização de uma linha de código, tive dificuldades de inicio ao implementar algo que não foi ensinado no curso então tive dificuldade a aprender e a aplicar o conhecimento no código e realizando a construção do projeto.

O sistema ainda está longe de estar concluído, contudo, podemos observar que o caminho para o desenvolvimento futuro está encaminhado.

# Bibliografia

Esta secção lista as referências bibliográficas online citadas no texto.

**[1]VS Code - O que é e por que você deve usar?** Disponível em: <https://www.treinaweb.com.br/blog/vs-code-o-que-e-e-por-que-voce-deve-usar>.

**[2] O que é: XAMPP - Blog Escola Ninja WP**. Disponível em: <https://blog.escolaninjawp.com.br/glossario/o-que-e-xampp/>.

**[3]**LENCINA, W. **O que é GitHub: para que serve, como funciona e como utilizar**. Disponível em: <https://ebaconline.com.br/blog/o-que-e-github>.

**[4]O que é JavaScript? - Aprendendo desenvolvimento web | MDN**. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/First\_steps/What\_is\_JavaScript>.

**[5]** *SweetAlert2*. (n.d.). Sweetalert2.Github.io. <https://sweetalert2.github.io/>

**[6]**<https://github.com/cabritaXgamer/PAP>

**[7]**<https://www.dicio.com.br/decodificar/>

**[8]** [**https://themeforest.net/item/carserv-auto-repair-bootstrap-5-template/32805423?\_ga=2.110565381.1454711771.1720523569-1569350184.1714396334&\_gac=1.88294121.1718788295.CjwKCAjwg8qzBhAoEiwAWagLrApLqM-YBXY7dtK548XY1hGfv8kQNZN5DWlOGfm\_1ugiEY6Jf6E5lhoC3hgQAvD\_BwE**](https://themeforest.net/item/carserv-auto-repair-bootstrap-5-template/32805423?_ga=2.110565381.1454711771.1720523569-1569350184.1714396334&_gac=1.88294121.1718788295.CjwKCAjwg8qzBhAoEiwAWagLrApLqM-YBXY7dtK548XY1hGfv8kQNZN5DWlOGfm_1ugiEY6Jf6E5lhoC3hgQAvD_BwE)

‌

# Anexo A - Listagem do material utilizado

Para a construção do meu projeto de aptidão profissional utilizei:

* repositório no GitHub;
* kit digital escolar;
* computador pessoal.