DEPARTAMENTO DE ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

CARLOS RAFAEL SEIXAS MORGADO

Licenciado em Engenharia Eletrotécnica e de Computadores

Plataforma Web para análise e diagnóstico de centrais de cogeração

MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES
Universidade NOVA de Lisboa
Março, 2023

DEPARTAMENTO DE ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

PLATAFORMA WEB PARA ANÁLISE E DIAGNÓSTICO DE CENTRAIS DE COGERAÇÃO

CARLOS RAFAEL SEIXAS MORGADO

Licenciado em Engenharia Eletrotécnica e de Computadores

Orientador: Professor João Almeida das Rosas,
Professor Auxiliar, Universidade NOVA de Lisboa

Coorientador: Professor João Murta Pina,
Professor Auxiliar, Universidade NOVA de Lisboa

Júri:

Presidente: Professor Luís Augusto Bica Gomes de Oliveira,
Professor Associado com Agregação, Universidade NOVA de Lisboa

Arguentes: Professor André Teixeira Bento Damas Mora,
Professor Auxiliar, Universidade NOVA de Lisboa
Professor João Almeida das Rosas,
Professor Auxiliar, Universidade NOVA de Lisboa

Orientador: Professor João Almeida das Rosas,
Professor Auxiliar, Universidade NOVA de Lisboa

MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

Universidade NOVA de Lisboa
Março, 2023

Plataforma Web para análise e Diagnóstico de Centrais de Cogeração

AGRADECIMENTOS

Quero agradecer primeiramente ao meu orientador Professor João Almeida das Rosas pela orientação, critica construtiva e conselhos prestados na elaboração da dissertação. Quero, também agradecer à Faculdade Ciências e Tecnologia da Universidade Nova de Lisboa pela oportunidade de adquirir conhecimentos que com certeza serão essenciais para o meu futuro. Um agradecimento muito especial aos meus pais e à minha irmã pelo apoio, amor e carinho dado que me permitiu ganhar força para concluir esta etapa tão importante na minha vida. Um agradecimento especial aos meus amigos Miguel Pita e Luís Ferreira pelo apoio dado, compreensão e amizade.

Resumo

Este projeto visa o desenvolvimento de uma plataforma cujo objetivo principal consiste em automatizar a recolha de análises efetuadas em máquinas de cogeração e proceder à respetiva verificação dos dados, diagnóstico de potenciais falhas/divergências, despoletando os respetivos alertas. O objetivo consiste em desenvolver uma plataforma web, tendo em conta os seguintes requisitos: o upload de dados que permitirá guardá-los numa base de dados, desenvolvimento de ferramentas que permitirão ao utilizador visualizar os dados de modo a perceber o estado atual do equipamento, fazer o download destes dados e, também, a deteção de eventuais anomalias como, por exemplo, a presença de metais com valores muito elevados nas máquinas desencadeando os respetivos alertas. Este projeto destina-se a uma empresa cujo processo descrito acima é feito de forma manual, isto é, com a intervenção de humano que faz com que as deteções de falhas não sejam tão eficazes. A plataforma de manutenção preditiva desenvolvida no âmbito da dissertação comprova que a automatização destes processos permitirá à empresa seguir uma melhor organização estratégica conseguindo, assim, reduzir o tempo de paragem do equipamento e, também, os custos adicionais. Será demonstrado que, ao contrário da intervenção de humano, a plataforma de manutenção preditiva é muito mais eficiente e eficaz.

Palavras-chave: Plataforma web, Análise de dados, Automatização

Abstract

This project aims to develop a platform whose main objective is to automate the collection of analyzes carried out on cogeneration machines and proceed with the respective data verification, diagnosis of potential failures/anomalies, and triggering the respective alerts. The objective is to develop a web platform, taking into account the following requirements: uploading data that will allow them to be stored in a database, development of tools that will allow the user to visualize the data to understand the current state of the equipment, download this data and also detect any anomalies such as, for example, the presence of metals with very high values in the machines, triggering the respective alerts. This project is aimed at a company whose process described above is done manually, that is, with human intervention, which makes fault detection ineffective. The predictive maintenance platform developed within the scope of the dissertation proves that the automation of these processes will allow the company to follow a better strategic organization, thus reducing equipment downtime and additional costs. It will be demonstrated that, contrary to human intervention, the predictive maintenance platform is much more efficient and effective.

Keywords: Web platform, Data analysis, Automation

ÍNDICE

Índice de Figuras

Índice de Tabelas

SIGLAS

CSS	*Cascading Style Sheets*
HTTP	*Hipertext Transfer Protocol*
HTML	*HiperText Markup Language*
IBM	International Business Machines
IMAP	*Internet Message Access Protocol*
JSON	*JavaScript Object Natation*
NoSQL	*Not Only SQL*
PHP	*Hipertext Preprocessor*
SQL	*Structured Query Language*
SMTP	*Simple Mail Transfer Protocol*
URL	*Uniform Resource Locator*

1

INTRODUÇÃO

1.1 Enquadramento

Atualmente ainda existem empresas cuja manutenção dos seus equipamentos é feita de forma manual, isto é, a manutenção dos equipamentos é totalmente dependente da intervenção humana. Desta forma, é mais difícil usufruir dos benefícios da manutenção preditiva. No caso da empresa, onde este trabalho foi efetuado, era necessária uma abordagem mais expedita para tratar dos resultados das análises do estado de operação de máquinas de cogeração. Por exemplo, para se detetar se uma máquina de cogeração irá ter uma eventual falha deve-se verificar se o valor de ppm de alguns metais presentes no fluido é elevado ou se existe uma grande quantidade de água. Em algumas empresas quem acaba por fazer estas análises é um humano o que acaba por não ser eficaz, uma vez que devido à grande dimensão/escala dos dados acaba sempre por falhar na deteção das anomalias. Está prevista a integração dum módulo *Machine Learning*, a ser desenvolvido noutra dissertação.

1.2 Objetivo

Nesta tese de mestrado pretende-se o desenvolvimento de uma plataforma de manutenção preditiva para a empresa *Spinerg*, uma empresa especializada no negócio de lubrificantes. A solução para o problema descrito na secção 1.1 passa por automatizar este processo demoroso para o ser humano. Para isso, deve-se criar processos que permitem receber as análises automaticamente. Assim, deve-se primeiramente, armazenar os dados numa base de dados, permitindo que assim fiquem disponíveis para a análise correspondente. Depois, deve-se criar um processo que avalie a informação das análises recebidas com o intuito de detetar anomalias. Esse processo envolve indicação de ultrapassagem de limiares, ou seja, se o valor dos metais presentes no óleo lubrificante ultrapassou o limite estipulado e, depois, aplicar os dados das análises a algoritmos de *Machine Learning* para detetar tendências e padrões de comportamentos mais sofisticados que correspondam com anomalias mais difíceis de discernir como, por exemplo, a variação de vários parâmetros ao longo do tempo, que podem ser problemáticos. Por último, deve-se desenvolver a exibição das análises de uma forma intuitiva mediante a criação dos gráficos e tabelas. Como resultado, é de esperar que com o desenvolvimento desta plataforma a empresa opte por uma melhor estratégia de manutenção das máquinas dado com os processos descritos, se consegue detetar anomalias de forma mais eficiente e eficaz.

Para tal, será necessário desenvolver uma plataforma Web[1] para dar suporte à manutenção preditiva de máquinas de cogeração. Desta forma, deve conseguir detetar eventuais falhas nas diversas máquinas de cogeração distribuídas geograficamente.

Um *website* contém uma parte *backend* e uma parte *frontend*. Para o *backend* será usada a linguagem PHP para processar as requisições dos utilizadores ao servidor, enquanto no *frontend* será elaborado com HTML5 usando ainda CSS para o desenho das páginas e Javascript para permitir que os utilizadores interagem com a interface gráfica. O website foi colocado em modo produção num provedor de alojamento, serviço "Hosting Smart Linux" [2].

Para tal, será necessário desenvolver um *website*. Um *website* é, na verdade, um conjunto de páginas *web* que são acedidas através de um protocolo chamado HTTP na Internet. Para se construir estas páginas será necessário desenvolver a parte visual que permitirá ao utilizador

[1] Este tipo de sistemas também pode ser designados por aplicações Web, websites, etc.
[2] http://www.amen.pt

interagir com o *website* (ou também se pode chamar de *frontend*) e a parte *backend* que é referente ao código de um programa que permite operar, mas que os utilizadores não podem aceder. Para esta parte será usada a linguagem PHP para processar as requisições dos utilizadores ao servidor. Já a parte visual (*frontend*), a Interface gráfica que permite a Interação do utilizador, o HTML, CSS, Ajax e JavaScript serão as linguagens de programação utilizadas para esta vertente.

1.3 Estrutura da Tese

Esta dissertação está constituída por 4 capítulos com este incluído. No capítulo 1 inclui-se a introdução, onde está descrito os objetivos principais da tese, para além do enquadramento e, ainda, a organização da estrutura da estrutura da tese. O capítulo 2 é abrangido por estado da arte onde se inclui os trabalhos relacionados mais relevantes para o que será desenvolvido. No capítulo 3 será apresentado o desenvolvimento da plataforma, no capítulo 4 os testes e validação bem como a pesquisa de informação e um exemplo de importação de dados e, por fim o capítulo 5 apresentar-se-á a conclusão com os resultados obtidos e um possível trabalho a ser realizado no futuro.

2

ESTADO DA ARTE

Neste capítulo será feito um resumo da revisão bibliográfica, para além de estar descrita de forma fundamentada a teoria associada à plataforma colaborativa científica e as linguagens usadas para o desenvolvimento da tese.

2.1 Plataforma Manutenção Preditiva

A evolução das máquinas e equipamentos têm levado a que estas tenham uma maior sofisticação ao longo do tempo o que as torna mais eficientes desenvolvendo produtos com mais facilidade e qualidade. No entanto, nada disso impede que possam sofrer falhas e/ou anomalias. Por isso, é necessário realizar uma manutenção adequada para se conseguir corrigir essas falhas, ou até prevenir a sua ocorrência, de modo a garantir o funcionamento da máquina. Existem três tipos de manutenção: manutenção corretiva, manutenção preventiva e manutenção preditiva.

A manutenção corretiva (também pode ser chamado manutenção reativa) é uma manutenção que é realizada após a ocorrência de uma falha e com o objetivo de recuperar o funcionamento da máquina. Este tipo de manutenção não é ideal para máquinas críticas, isto é, para máquinas no qual se pretende que estejam em funcionamento contínuo pois uma falha pode levar a paragens inesperadas ou prolongadas e, com isso, provocar uma queda nas receitas da empresa.

A manutenção preventiva, como o próprio nome indica, tem como objetivo prevenir que uma falha aconteça. Este tipo de manutenção é feito de forma periódica independentemente do estado da máquina com o intuito de aumentar o tempo de vida das máquinas. Uma vez

que não depende da condição da máquina, este tipo de manutenção e feito de forma periódica o que pode resultar de manutenção desnecessária o que implica custos adicionais.

Já a manutenção preditiva é semelhante ao da manutenção preventiva com a diferença de que a manutenção ocorre quando uma avaria está a prestes a acontecer, conforme descrito mais abaixo. Na Figura 1, apresenta-se um esquema com as diversas modalidades de manutenção.

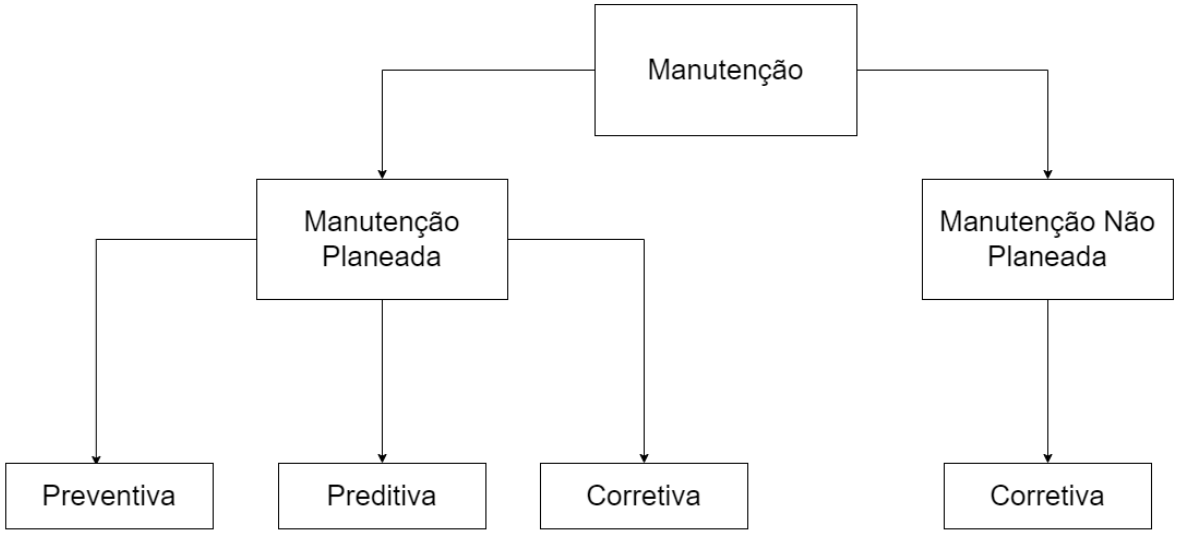


Figura 1 - Tipos de manutenção. [1].

2.1.1 Manutenção Preditiva

Como foi citado anteriormente, a manutenção preditiva avalia as condições do funcionamento do equipamento com o objetivo de prever falhas e determinar quando irá acontecer. Este tipo de manutenção consiste em analisar os dados dos equipamentos e extrair informações que indiquem quando a uma falha está para acontecer. Pode-se, então, dizer-se que, de forma muito geral, a manutenção preditiva contém três etapas: recolha de dados (*data collection*), mineração de dados (*data mining*) e aplicação de algoritmos como *Machine Learning* [1].

2.1.1.1 Recolha de dados

Se a ideia é prever quando a falha vai ocorrer, então é necessário recolher os dados para que a plataforma possa analisá-los e, assim, extrair as informações. Essa recolha feita instalando sensores nas máquinas para que se possam recolher os dados em tempo real. Nesta

dissertação pretende-se aplicar aos óleos lubrificantes. Portanto, os sensores devem recolher os dados de todos os metais presentes no óleo. Após recolher estes dados, segue-se para a próxima etapa em que ocorre a mineração de dados.

2.1.1.2 Mineração de dados

Nesta etapa, os dados são enviados para o *software* que ao processá-los, irá recolher as informações mais relevantes que lhe permita chegar a uma conclusão sobre o estado do equipamento. Com isto, é possível distinguir tendências e padrões ao usar um conjunto de dados. Pegando na técnica usada anteriormente, os dados a considerar devem ser os valores dos metais presentes no lubrificante, logo os valores destes dados devem ser recolhidos e usados para a análise onde serão usados na terceira e última etapa na aplicação dos algoritmos.

2.1.1.3 Aplicação de Algoritmos

Nesta etapa são feitos os cálculos com os dados relevantes para, depois, chegar a um prognóstico. No exemplo abordado o *software* está programado para verificar se existem certos metais presentes no óleo que indiquem desgaste da máquina e se os seus valores são altos. Caso exista, o *software* está programado para dar o alerta com a mensagem de que será necessário agendar uma intervenção preventiva pois estamos a falar de uma manutenção antes da avaria acontecer. Isto porque os critérios que definem que a máquina está para ter uma avaria é a presença de metais nomeadamente o Ferro (Fe), Alumínio (Al), Chumbo (Pb) e Estanho (Sn). As presenças destes metais no fluido indicam que existe um desgaste em peças que constituem o equipamento.

Como se pode perceber, a manutenção preditiva oferece muitas vantagens que valem o investimento como redução de custos da manutenção nos equipamentos, redução do número de falhas, maximização da produtividade das máquinas, aumento do tempo de vida útil das máquinas, permite uma melhor estratégia da parte da empresa de forma a aumentar as suas receitas.

2.2 Arquitetura Web

Qualquer site que se aceda, acaba-se por interagir com uma plataforma podendo ela ser de entretenimento, informativo ou outra plataforma *web* de qualquer conteúdo. Por exemplo, numa plataforma informativa, o utilizador pode pesquisar/consultar noticias da atualidade ou

até noticias mais antigas podendo, também, pesquisar noticias relacionado com temas do interesse do utilizador. Outro exemplo, é o portal das finanças que é uma plataforma online da autoridade tributária que permite ao utilizador, neste caso os contribuintes portugueses, tratar de assuntos fiscais relacionados com os seus impostos e/ou patrimónios, como por exemplo, validar faturas, fazer pesquisa de quanto já descontou nos impostos de um determinado intervalo de tempo. Pode-se, então, referir que as plataformas *web* revelam bastante utilidade uma vez que, por exemplo, permite a muitas pessoas tratar de certos assuntos sem terem que se deslocar a determinados sítios e, um exemplo disso é o que se acabou de referir no parágrafo anterior pois antigamente o contribuinte para tratar de assuntos como impostos e/ou patrimónios tinha que se deslocar a uma lógica física de finanças e isso era um problema pois nessa altura havia filas intermináveis o que levava as pessoas a estarem horas intermináveis à espera. Agora com o portal das finanças, os contribuintes podem tratar deste tipo de assuntos em casa o que torna muito mais prático e cómodo sem ter de lhes tomar demasiado tempo.
Existe uma plataforma chamada OpenML [2] que é uma plataforma de código aberto para aprendizado da máquina colaborativo. Foi criado para facilitar o compartilhamento e a colaboração de conjunto de dados de *Machine Learning*, algoritmos e experimentos entre pesquisadores e profissionais. O OpenML permite que os utilizadores importem um conjunto de dados, treinem e avaliem modelos de *Machine Learning* e partilhem os resultados com outros utilizadores. Outra plataforma um pouco idêntica à do OpenML é o SciServer que permite que os cientistas armazenem conjunto de dados das suas pesquisas/experiências e, também, partilhá-los com outros cientistas. Ao contrário do OpenML, o SciServer [3] não permite criar pesquisas em *Machine Learning*, somente permite a partilha de dados com outros utilizadores.
Existem plataformas *web* para todos os setores. Uma delas é a plataforma *Epik* [4], que é uma plataforma de desenvolvimento de jogos educativos.

Como se pode ver, estas plataformas têm revelado ter bastante impacto na sociedade. Isto porque nas empresas existem situações que podem indicar futuros incidentes que podem prejudicar o funcionamento do negócio sendo necessário arranjar a melhor estratégia para a resolução desse incidente ou até mesmo conseguir evitá-lo. Para isso as empresas necessitam que alguém analise dados e que chegue a uma conclusão e isto pode ser processo demoroso pois nestas empresas existe um enorme conjunto de dados. E é precisamente nestas situações que as plataformas se tornam uma ferramenta bastante conveniente e lucrativa para as empresas já que são estas aplicações que fazem o trabalho todo indicando só uma conclusão com base nos dados que a empresa contém.

E é neste contexto que se irá elaborar esta dissertação. Para esta dissertação será desenvolvida uma plataforma web com objetivo de automatizar a recolha de análises efetuadas em máquinas de cogeração para depois se proceder à respetiva análise dos dados, diagnóstico de falhas/divergências desencadeando os respetivos alertas.

Uma vez que se pretende desenvolver uma plataforma (ou aplicação) *web*, deve-se ter presente um conhecimento mais aprofundado sobre este tema que é saber o comportamento/funcionamento destas aplicações.

As aplicações *web* são muito semelhantes a aplicações normais. Sabe-se que uma aplicação é um *software* que funciona num determinado sistema, como por exemplo, num computador ou num smartphone pois as aplicações são executadas pelo utilizador localmente nestes dispositivos pelo utilizador. Ora, as aplicações *web* são ligeiramente diferentes pois estas são executadas num servidor *web*, isto é, quando se tenta aceder à aplicação, ela é executada num computador que está ligado à internet.

Então, para aceder à plataforma o utilizador faz um pedido ao servidor, este por sua vez ao receber a solicitação do utilizador, devolve a resposta ao utilizador. Estas solicitações e respostas trocadas entre o servidor *web* e o utilizador são feitas através de um protocolo chamado *Hypertext Transfer Protocol* ou abreviadamente HTTP. Ou seja, este protocolo é responsável pela transferência de dados na *web*. Portanto, pode-se constatar que existem duas camadas neste protocolo: o utilizador (também se pode designar de cliente) que faz uma solicitação e o servidor que responde à solicitação do cliente. O HTTP determina o esqueleto destas mensagens e a forma como estes dois intervenientes as trocam.

As páginas *web*, são compostas por vários arquivos incluindo texto, imagens e vídeos, arquivos esses que podem ser guardados em diversos formatos, como por exemplo GIF, JPEG, PNG, MP4, RMVB, entre outros. As páginas são desenvolvidas com uma linguagem de programação designada por HTML no qual se irá abordar mais à frente [5].

Por fim, falta abordar a forma de como o cliente acede à página. Existe, portanto, um endereço *web* que é uma referência a um recurso da web que especifica a localização de uma página web. Este endereço dá-se pelo nome de *Uniform Resource Locator*, ou URL de forma abreviada. Cada URL contem dois elementos: o nome do *hostname* do servidor ou também se pode chamar de domínio, e o local onde está o objeto anteriormente referido e o nome do caminho do objeto. Tome-se o seguinte URL como exemplo: https://www.fct.unl.pt/estudante/informacao-academica. No exemplo referido, o domínio é www.fct.unl.pt já o resto do link, /estudante/informacao-academica é o nome do caminho que contem o documento *web* [6].

Como foi referido anteriormente, o HTTP determina a estrutura e a forma de como é que as mensagens trocadas entre o cliente e o servidor. Sabe-se, então, que no servidor encontram-se várias páginas *web*. Para o utilizador ter acesso a elas, terá de enviar uma requisição ao servidor usando um URL através de um navegador *web* (Google Chrome, Microsoft Edge, Firefox). Essa requisição é feita através do protocolo HTTP e o servidor, através do endereço indicado pelo cliente, irá buscar o objeto referente a esse endereço e enviará, através de uma mensagem de resposta HTTP, o documento *web* solicitado pelo cliente (Figura 2).

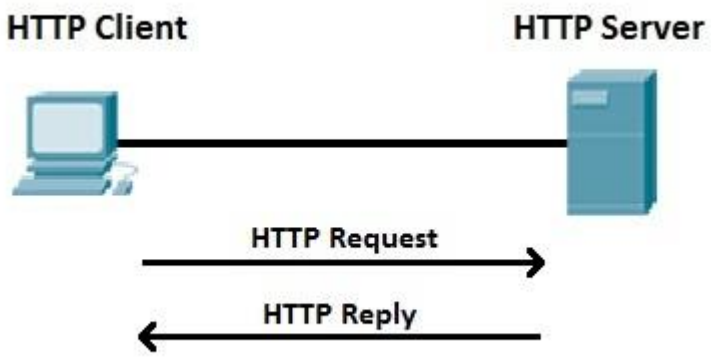


Figura 2 - Comunicação entre o utilizador e o servidor [20].

Já se sabe, portanto, que com o protocolo HTTP, o utilizador solicita ao servidor um documento *web*. Mas quando o cliente interage com uma plataforma, o protocolo HTTP utiliza vários métodos dependendo da ação por parte do utilizador. Por exemplo: quando um utilizador insere no navegador do browser o seguinte link https://www.fct.unl.pt/estudante/informacao-academica o método que está a ser usado é o método GET pois, neste caso, o utilizador está a solicitar ao servidor do domínio fct.unl.pt para aceder à página com aquele endereço. Os métodos existentes são: GET, HEAD, POST, OPTIONS, DELETE, TRACE, PUT, CONNECT [7].

Os métodos mais usados são o GET, POST, PUT e DELETE. Os métodos POST e PUT são comumente usados para enviar dados para um servidor. O método POST é usado para enviar novos dados ao servidor. Por exemplo, quando se está no site do facebook e quer-se criar uma conta. Neste caso preenche-se os campos com alguns dos nossos dados e uma password e, ao clicar no botão *Registar* está-se a enviar dados ao servidor permitindo, assim, ao utilizador usufruir desta plataforma. Quanto ao método PUT, é usado para atualizar dados já existentes. O método DELETE serve para remover um determinado objeto. Por exemplo, em certas plataformas, por vezes existem dados que, que por algum motivo, já não são necessários, portanto

não faz muito sentido continuar em exposição na plataforma. solução é remover esses dados e nessas plataformas existe obviamente a opção apagar. Quando o utilizador interage com essa opção, o protocolo HTTP recorre ao método DELETE permitindo, assim, a remoção do(s) dado(s) que o utilizador quis eliminar [7].

2.3 Componentes Tecnológicas

Para desenvolver uma plataforma de software, é possível utilizar diversas linguagens de programação. No entanto, é importante considerar os requisitos do sistema e escolher a linguagem de programação mais adequada para otimizar o desempenho e eficiência do *software*.

Softwares muito grandes ou que requerem muitos recursos podem apresentar lentidão devido à quantidade excessiva de código ou ao uso ineficiente da memória. Portanto, é fundamental escolher a linguagem de programação adequada e escrever um código eficiente para garantir um desempenho ideal.

Em certas linguagens de programação existem bibliotecas que podem ser usadas para tornar certos requisitos menos demorados no tempo de resposta. Ainda assim, qualquer *software*/aplicação, independentemente da complexidade no desenvolvimento do requisito, o *software* pode ser desenvolvido com qualquer linguagem de programação. Portanto, não há nenhuma "regra" que obrigue os programadores a desenvolver uma aplicação uma linguagem de programação especifica.

Então a questão que se coloca é, que linguagem deverá ser usada para elaborar uma plataforma? Deve-se escolher a linguagem com base na confiabilidade. Posto isso, com base no que que foi referido acima, para a presente dissertação, escolheu-se desenvolver a plataforma com as seguintes linguagens de programação:

- PHP;
- SQL;
- HTML;
- JavaScript

As linguagens referidas acima, são as mais utilizadas para desenvolver plataformas desta natureza. Nas secções seguintes vai-se abordar as linguagens que se vai utilizar para elaborar a plataforma.

2.3.1 PHP

O *Hypertext Preprocessor* ou mais abreviadamente PHP, é uma linguagem de programação criada por Rasmus Lerdof em 1994. Rasmus Lerdof desenvolveu o PHP para substituir vários scripts que ele usava na sua página pessoal [8]. Na figura 3 é apresentado em exemplo de um código e linguagem PHP.

```php
<?php

// Random PHP code snippet!

function create_category_feeds($categories = NULL) {

    global $wpdb, $title, $headcomments;

    if ($categories == NULL) {
        $sort_column = 'term_id';
        $query = "SELECT * FROM $wpdb->term_taxonomy
                  JOIN $wpdb->terms ON ( $wpdb->term_taxonomy.term_id = $wpdb->terms.term_id )
                  WHERE $wpdb->term_taxonomy.taxonomy = 'category' AND $wpdb->terms.term_id > 0 AND count
                  ORDER BY $wpdb->terms.name ASC";
        $categories = $wpdb->get_results($query);
    }

    $catsnum = count($categories);

    foreach ($categories as $category) {
        $link = '<link rel="alternate" type="application/rss+xml" title="';
        $link = $link . $title . ': ' . $category->name;
        $link = $link . '" href="' . get_category_rss_link(0, $category->term_id, $category->name) . '" />
        echo "\t" . $link . "\n";
    }

    $hcomlink = '<link rel="alternate" type="application/rss+xml" title="';
    $hcomlink = $hcomlink . $title . ': Comments';
```

Figura 3 - Exemplo de código em PHP [21].

O PHP é uma linguagem usada para desenvolver aplicações no lado do servidor. Muitas vezes o PHP é escolhido para desenvolver *websites* dinâmicos e interativos. O PHP foi uma das primeiras linguagens do lado do servidor que poderia ser incorporado em HTML o que torna mais fácil obter dados para, por exemplo, metê-los diretamente na página *web* (front-end) sem se necessitar de ficheiros externos para o fazer. O PHP tem evoluindo ao longo do tempo com várias atualizações. A última atualização até à data é a versão 8.1.0 que foi lançada em novembro de 2021. À medida que vão lançando novas versões, estas trazem sempre novas funcionalidades e o desbloqueiam novas capacidades [9].

Quando se vai elaborar uma plataforma *web* muitas das vezes escolhe-se o PHP pois é uma linguagem fácil de usar. Não necessita de se ter um vasto conhecimento em desenvolver páginas *web*. É também, uma linguagem muito versátil, ou seja, é independente da plataforma. Significa isto que se pode desenvolver qualquer aplicação/plataforma em qualquer sistema

operativo, ou seja, pode-se elaborar a dita aplicação/plataforma no Windows, MAC OS ou Linux.

O PHP é uma linguagem de programação do lado do servidor altamente versátil, que permite a criação de aplicações web compatíveis com diversos navegadores, como o Chrome, o Microsoft Edge e o Firefox, o que facilita a implementação de soluções em várias plataformas e sistemas, pois o conteúdo gerado pelo PHP pode ser exibido corretamente em diferentes navegadores pelos utilizadores finais.

Quando se quer elaborar uma aplicação ou uma plataforma, deseja-se que esta seja rápida e segura. O PHP corresponde positivamente a estes requisitos. O PHP usa a sua própria memória e, quando se usa a versão mais recente, é bastante rápido no que à velocidade diz respeito. A segurança é um requisito bastante importante de forma que se possa proteger os *websites* de ciberataques algo que são bastante frequentes atualmente.

O PHP consegue facilitar a ligação segura com qualquer base de dados. Existem bibliotecas com as quais o PHP consegue conectar-se a uma base dados de forma segura e seja de que tipo de base de dados for. Com essas bibliotecas, pode-se conectar-se a uma base dados.

O PHP é *open-source*, isto é, é um código que pode ser usado livremente para se poder utilizar e modificar sem qualquer tipo de restrições além de, também, ser gratuito. Com o PHP, é possível ter acesso a *frameworks* como *Symfony*, Laravel, entre outros, tornando essa funcionalidade bastante atrativa para as empresas, uma vez que ajuda a controlar seus custos. Uma vez que se vai usar o Laravel para este trabalho, na próxima secção vai abordar de forma mais detalhada esta *framework*.

2.3.1.1 Laravel

O Laravel é um *framework* PHP criado por Taylor Otwell em junho de 2011 com o objetivo de ser uma alternativa ao *CodeIgniter*. Ele é amplamente utilizado para desenvolver aplicações e plataformas web, e tem-se tornado uma das *frameworks* mais populares. Contudo, existem diferenças entre os dois. O Laravel é muito mais complexo do que o *Coldelgniter* pois existem bibliotecas que o Laravel possui que o *Coldelgniter* não tem. Por exemplo, o Laravel vem com funcionalidades de autenticação implementada algo que o *Coldelgniter* não tem. Outra é que o Laravel é orientado a objetos relacionais enquanto o *Coldelgniter* é orientado para objetos, ou seja, o Laravel pega nos dados e relaciona-os entre si com base em certas informações e em diferentes fatores, já o *Coldelgniter* tem uma organização mais isolada, isto é, os dados são agrupados sem se relacionarem uns com os outros.

O *ColdeIgniter* é uma *framework* simples e até muito fácil de aprender para quem ainda está a dar os primeiros passos na elaboração de plataformas *web* em PHP, ao contrário do Laravel pois este tem características mais avançadas, mais desenvolvidas e que requer uma maior atenção para o compreender. O Laravel é uma estrutura de desenvolvimento backend, ou seja, é executado no lado do servidor. É muito popular devido ao seu funcionamento em PHP, que é uma linguagem de programação frequentemente usada para desenvolvimento de backend, como mencionado anteriormente. O Laravel foca-se, essencialmente na manipulação de dados utilizando um padrão denominado *Model-View-Controller* ou abreviadamente MVC.

O MVC é uma arquitetura de *software* que é utilizado para desenvolver interfaces de utilizador e que dividem a lógica em três partes interligadas: *Model*, *View* e *Controller*. O *Model* é o componente central, o cérebro da aplicação por assim dizer. O *Model* gere os dados da aplicação/plataforma, a lógica de como eles se relacionam. O *Controller* é uma espécie de ponte que liga o *Model* ao *View*. O *Controller* interage com o *Model* obtendo informações através deste, muitas vezes através da base de dados, e depois vai dar essas informações ao View. O View, por sua vez, irá mostrar essas informações [10]-[12]. Por exemplo, se um utilizador solicitar alguma informação, ele envia um pedido HTTP ao Controller, este por sua vez pede os dados relativa a informação solicitada pelo utilizador. Assim que tiver os dados, o *Model* irá providenciar os dados ao *Controller* e, por último, este irá exibirá a informação pretendida pelo utilizador na *View*. Os dois primeiros componentes abordados serão programados com PHP pois como se percebeu operam no backend, já a *View* será com HTML e CSS (linguagens que se irá abordar mais à frente) pois esta componente é que irá providenciar ao utilizador informações que o pediu.

O Laravel usa esta estrutura para meter as plataformas a funcionar e desta forma o HTML seja gerido pelo *controller*. A arquitetura MVC é uma abordagem popular para o desenvolvimento de *software* que apresenta várias vantagens. Uma das principais vantagens é a separação clara das responsabilidades do software em três componentes distintos: o *Model*, que representa os dados e a lógica de negócios do sistema, a *View*, que é responsável pela interface do utilizador e pela apresentação dos dados, e o *controller*, que atua como intermediário entre o *model* e a *view*, tratando as solicitações do utilizador e atualizando o *model* ou a *view* conforme necessário.

Ao separar a aplicação em três componentes distintos, a arquitetura MVC torna o desenvolvimento mais modular e organizado, facilitando a manutenção e a evolução da aplicação. Cada componente é independente e pode ser alterado sem afetar os demais, desde que

a interface entre eles seja preservada. Esta separação também facilita a adição de novas funcionalidades e adaptação da aplicação a novos requisitos, sem a necessidade de alterar todo o código. A Figura 4 resume o que foi dito sobre a estrutura MVC.

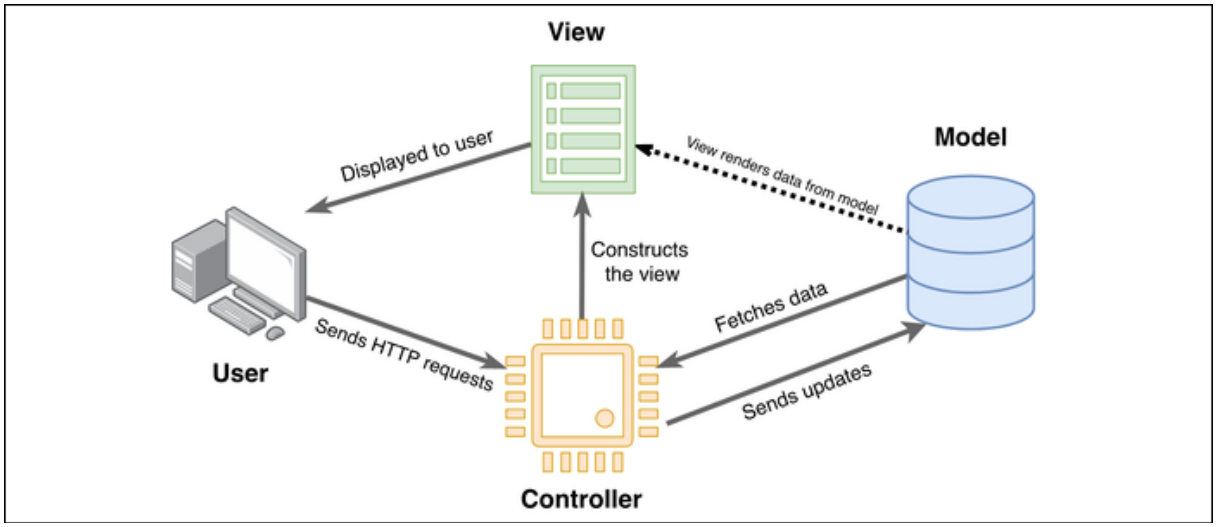


Figura 4 - Estrutura do MVC [22].

O Laravel beneficia-se da utilização da aplicação designada *composer*, que é uma ferramenta usada para gerenciar as dependências em PHP. Por outras palavras, o *composer* permite identificar quais são as dependências necessárias para que a aplicação funcione corretamente, além de instalá-las e atualizá-las automaticamente. Lançado em 2012, o *composer* foi criado por Nils Adermann e por Jordi Boggiano e é inspirado no *npm* e no *bundler*, só que a diferença é que o *composer* instala os pacotes numa diretoria chamada *vendor* e gere os pacotes a partir daí [11].

Outra ferramenta importante do Laravel é o *Artisan* que é uma linha de comando que contem um conjunto de ações, isto é, quando os comandos são executados pelo *developer* (nome usado à pessoa que está a desenvolver o projeto) decorrem ações como migração de dados, ou criação de controladores [13].

Apesar de a *framework* Laravel ter alguns mecanismos de proteção, existem algumas vulnerabilidades. Num artigo publicado por Felipe Pacheco, em que aborda os ataques a aplicações *web* desenvolvidas com o Laravel encontraram-se algumas vulnerabilidades. Por exemplo, um dos testes realizados foi utilizando um *cookie* de chamado *EditThisCookie* que, segundo o artigo, é uma extensão que gere os *cookies* no navegador e que permite procurar dados enquanto se navega pela *web*. Neste artigo foram descritas experiências utilizando este *cookie*

para testar a vulnerabilidade e detetou-se um roubo de dados provando, assim, a vulnerabilidade do Laravel perante este *cookie* [14].

2.3.2 SQL

A *Structured Query Language*, ou mais abreviadamente SQL é uma linguagem bastante usada para pesquisa e manipulação de bases de dados [8]. O SQL começou a surgir no início da década de 70 resultado de um projeto num laboratório da IBM cujo objetivo deste é demonstrar que a implementação do modelo relacional é muito mais viável, modelo este que foi proposto por Edgar Frank Codd. Inicialmente, esta linguagem dava-se pelo nome de *Structured English Query Language* (em português Linguagem de Consulta Estrutura) ou abreviadamente SEQUEL sendo que mais tarde foi mudado para SQL uma vez que SEQUEL era uma marca registada de uma empresa situada na Inglaterra chamada *Hawker Siddeley Dynamics Engineering Limited*.

Nos finais dos nãos 70 foi disponibilizado comercialmente o primeiro sistema de base de dados. Já na década de 80, ANSI e ISO trabalharam em conjunto na primeira versão padronizada da linguagem SQL e, de seguida, publicada. E, a partir daqui a linguagem foi evoluindo sendo criadas novas versões padrão, como por exemplo, SQL-92, SQL-99 tendo sido designadas desta forma de modo a fazer referência aos anos em que foram criadas. Existem vários modelos de base de dados, mas os mais comuns são o modelo Relacional e o modelo Não-Relacional no qual se irá agora abordar cada um deles.

2.3.2.1 Modelo Relacional

No modelo relacional, os dados são representados através de tabelas, como o próprio nome sugere. Cada tabela contém os registos e as colunas (também chamadas de atributos), cada uma com um domínio específico - ou seja, um conjunto de valores que podem ser armazenados nessa coluna. O foco principal desse modelo é a relação entre as tabelas. A tabela só pode ser modificada por comandos específicos. Por exemplo, veja-se a tabela 1, com o nome de *Estudante* e que tem como colunas/atributos *Nome*, *Cartão de Cidadão/Bi*, *Idade* e *Email* no qual os domínios destes atributos são, respetivamente *Integer*, *String*, *String*, *Integer* e *String*. Se se quiser alterar a tabela, seja apagar/adicionar colunas ou até mesmo apagar a tabela, ter-se-á de usar comandos específicos. No artigo *O Modelo Relacional*, são indicados os comandos para criar e modificar tabelas, assim como inserir as tuplas nelas [15].

Estudante
Id (Integer)
Nome (String)
Cartão de Cidadão/BI (String)
Idade (Integer)
Email (String)

Tabela 1 - Exemplo de uma tabela usada numa base de dados relacional

Nestas tabelas, como foi referido acima, a estrutura de dados também é constituída por instâncias que, também, se pode designar-se por tuplos e estas instâncias são um conjunto de linhas que estão relacionadas com a tabela [15]. Aproveita-se, agora, para se pegar na tabela ilustrada acima para que se possa exemplificar na prática os tuplos/linhas. Veja-se o seguinte exemplo:

Id	Nome	Cartão Cidadão	Idade	Email
1	João	789654123	26	joao@fct.unl.pt
2	Carlos	115599775	30	carlos@fct.unl.pt
3	Ana	147852369	25	ana@fct.unl.pt
4	Beatriz	541288974	26	bia@fct.unl.pt
5	Ricardo	157869125	24	ricky@fct.unl.pt

Tabela 2 - Tabela com tuplos/linhas

A tabela 2, é um exemplo de uma instância para a tabela *Estudante*, neste caso, esta tabela tem cinco tuplos. As tabelas são estruturas de dados fixas, enquanto as linhas representam informações específicas que podem ser modificadas ao longo do tempo. Essas modificações são realizadas através da interação do utilizador com a plataforma que armazena e gere os dados. A maioria das plataformas têm uma página no qual os utilizadores podem alterar os seus dados pessoais. Por exemplo, imagine-se que uma plataforma em que quando um utilizador se regista, a plataforma pede ao utilizador que este dê um *nickname*, isto é, uma alcunha para que possa ser visualizada num dos cantos superiores da plataforma. Se o utilizador desejar alterar o *nickname*, ele pode fazê-lo na página de *Definições*, que é comumente encontrada em diversas plataformas. Quando o utilizador acede a essa página e muda de *nickname*, é enviado a informação com o id que identifica o utilizador e o novo *nickname* que vai ser guardado numa tabela que contem um atributo/coluna designada, neste caso, aos *nicknames*. Ou

seja, a linha relativa ao utilizador será atualizada, sendo que, neste caso, o que atualizará será o atributo destinado aos *nicknames*.

Em cada tabela existe uma e uma só chave primária. A chave primária é um atributo, ou uma combinação deles cujo valor permite a identificação de forma unívoca dos registos/dados de uma tabela/relação de forma que um determinado valor só pode aparecer uma única vez em cada tabela tornando os registos únicos, isto é, diferentes entre si. Existe, também uma chave estrangeira que é um conjunto de atributos de uma tabela/relação que é usado para fazer uma referência a um registo de uma outra tabela/relação. Estas chaves servem para relacionar as tabelas entre si.

Existem casos em que é necessário fazer várias consultas e, para isso, é necessário unir dados que estão em tabelas diferentes. No modelo relacional existe um método para fazer essa junção de dados usando um comando JOIN.

Existem várias bases de dados que operam com este modelo, mas, segundo a plataforma *DB-engines* (https://db-engines.com/en/ranking), que é uma plataforma no qual mostra uma tabela com as bases de dados mais utilizadas, vê-se que até à data, são o *Oracle*, *MySQL*, *Microsoft SQL Server*. Figura 5 mostra esse *ranking*. Na imagem, são também apresentadas bases de dados de outro modelo (modelo não-relacional) que se irá abordar daqui a pouco.

383 systems in ranking, February 2022

Rank			DBMS	Database Model	Score		
Feb 2022	Jan 2022	Feb 2021			Feb 2022	Jan 2022	Feb 2021
1.	1.	1.	Oracle	Relational, Multi-model	1256.83	-10.05	-59.84
2.	2.	2.	MySQL	Relational, Multi-model	1214.68	+8.63	-28.69
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	949.05	+4.24	-73.88
4.	4.	4.	PostgreSQL	Relational, Multi-model	609.38	+2.83	+58.42
5.	5.	5.	MongoDB	Document, Multi-model	488.64	+0.07	+29.69
6.	6.	↑ 7.	Redis	Key-value, Multi-model	175.80	-2.18	+23.23
7.	7.	↓ 6.	IBM Db2	Relational, Multi-model	162.88	-1.32	+5.26
8.	8.	8.	Elasticsearch	Search engine, Multi-model	162.29	+1.54	+11.29
9.	9.	↑ 11.	Microsoft Access	Relational	131.26	+2.31	+17.09
10.	10.	↓ 9.	SQLite	Relational	128.37	+0.94	+5.20

Figura 5 - Bases de dados [23].

Até agora abordou-se o modelo relacional. Outro modelo bastante utilizado é o modelo não-relacionado que se irá abordar agora no qual, no final desta subsecção, irá também falar das vantagens, desvantagens e das diferenças de ambos os modelos.

2.3.2.2 Modelo Não-Relacional

Inicialmente, este tipo de modelo era conhecido como base de dados relacional, embora atualmente também seja denominado de NoSQL. Este termo foi usado por *Carlo Strozzi* pois dizia que o NoSQL era totalmente diferente do modelo relacional [17].

O modelo não-relacional é diferente do modelo relacional no sentido em que os atributos não têm uma tabela em comum. Viu-se nos exemplos apresentados que a tabela *Estudante* tinha como atributos: *Id*, *Nome*, *Cartão de Cidadão*, *Idade* e *Email*. No modelo não-relacional estes atributos não têm este tipo de relação. Neste modelo, uma das formas de armazenamento de dados é do tipo de *key-values*. Como o próprio nome indica os dados são apresentados em pares de chave-valor.

A Figura 6 mostra um exemplo deste tipo de base de dados não-relacional.

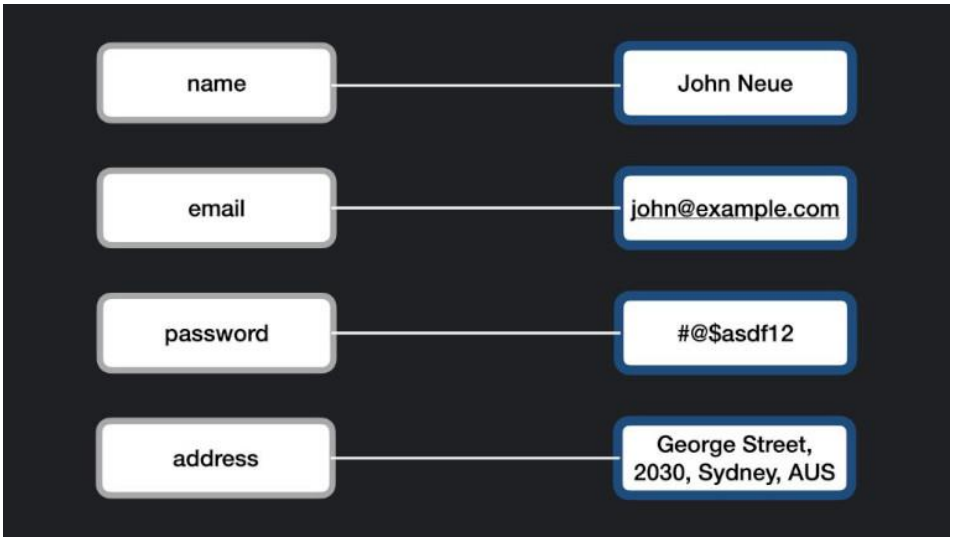


Figura 6 - Exemplo de organização de uma base de dados não-relacional [24].

Como se pode perceber, este modelo não armazena os dados/registos da mesma forma que o modelo relacional. Outro tipo de base dados que usa este modelo é quando os dados são armazenados em ficheiros do tipo JSON (*JavaScript Object Notation*). Ou seja, existe então, um documento com toda a informação sem ter a necessidade de se relacionar com quaisquer outros documentos, daí o nome ser modelo não-relacional uma vez que não necessita de relacionar outros documentos para obter o resto das informações, ou seja, toda a informação está no ficheiro JSON o que é bastante útil pois pode-se adicionar informação nova mais tarde e/ou até novos campos. Isto faz com que este modelo se torne mais flexível conforme as necessidades do utilizador sem haver preocupações quanto ao impacto nos dados já

armazenados. A imagem que se segue, permite ter uma perceção de como os dados são armazenados.

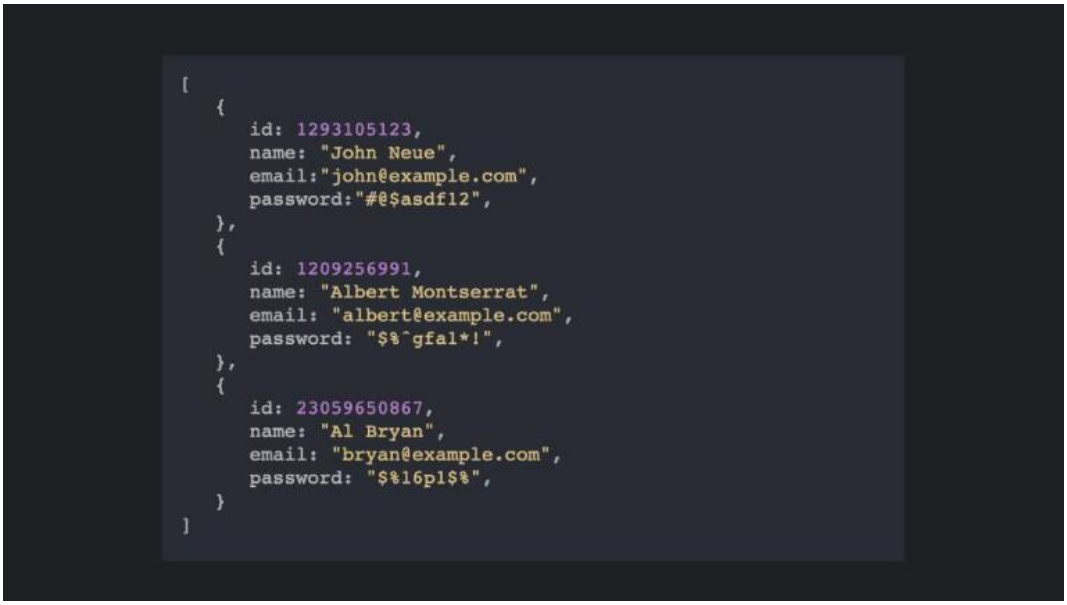


Figura 7 - Exemplo de um ficheiro JSON [24].

Ao contrário do modelo relacional, não existe o comando JOIN, ou seja, se se quiser fazer várias consultas ter-se-á de juntar os dados manualmente.

Bases de dados de modelo não-relacional como, por exemplo, NoSQL, apesar de ser uma base de dados bastante flexível e capaz de suportar uma enorme quantidade de dados, não significa que venha para substituir as bases de dados de modelo relacional. Apenas conseguem gerir os dados de forma mais eficiente algo que as bases de dados de modelo relacional não conseguem fazer. [18]. A tabela 3 permite ter uma melhor compreensão entre os dois modelos de base de dados abordados.

	Modelo Relacional	Modelo não-relacional
Base de dados	SQL	NoSQL
Armazenamento	Tabelas com atributos e tuplas fixas	Documentos JSON, *key-value*
Esquemas	Rigoroso	Flexível
Exemplos	Oracle, MySQL, PostgreSQL	MongoDB, Cassandra

Tabela 3 - Comparação entre modelos de base de dados

2.3.3 HTML

O *Hyper Text Markup Language* é uma das linguagens fundamentais da *Web*. O *Hyper Text Markup Language*, ou HTML de forma abreviada, é uma linguagem de programação padrão para documentos que são desenvolvidos para se exibir em páginas *web*. Originalmente desenvolvido nos anos 80 por um físico de seu nome Tim Berners-Lee, que foi um funcionário no CERN, o HTML foi sempre ao longo do tempo evoluindo para várias versões [8]. A última versão é o HTML5 que foi lançada em 2014.

Tim Berners-Lee criou esta tecnologia a partir de uma ideologia de forma que vários cientistas partilhassem documentos através da internet. Nessa altura, as comunicações eram feitas através de um protocolo de transferência de ficheiros ou abreviadamente protocolos FTP (*File Transfer Protocol*) que basicamente consiste em transferir ficheiros do computador de um servidor para um cliente numa rede de computadores. Esta é a tecnologia que se usa para se desenvolver páginas Web podendo, também, usar outras tecnologias. Os mais populares são *Cascading Style Sheets* (em português Folhas de Estilo em Cascata) ou mais abreviadamente CSS e também o JavaScript. O CSS serve para definir o estilo da página Web, como por exemplo, o tamanho e/ou tipo de letra, a cor desta e/ou da página, dividir conteúdos, adicionar animações e entre outras características que se pretende para decorar uma página. Já o JavaScript serve mais para desenvolver uma determinada funcionalidade/comportamento de uma página quando o utilizador interage com a página.

2.4 Trabalhos Relacionados

Nesta secção será citado um trabalho relacionado com plataformas preditivas. As plataformas preditivas têm surgido com o objetivo de ajudar as empresas a reduzir custos de manutenção. É o caso da Brightly Asset Essentials que é uma plataforma que permite fazer a gestão dos ativos de uma empresa. Com esta plataforma, a empresa monitoriza os seus ativos e, com isso, pode prever as suas falhas, prolongando o seu ciclo de vida, reduzindo custos adicionais e aumentando a rentabilidade [19]. Outra plataforma é a RapidMiner que à semelhança da Brightly Asset Essentials, também permite fazer análise de dados. No entanto, a RapidMiner, também permite criar modelos preditivos usando algoritmos de *machine learning* no qual podem ser usados para analisar, por exemplo, o estado de máquinas. [25]. Contudo, as plataformas acima mencionadas não contêm a funcionalidade de envio de alertas por email quando o estado de determinados equipamentos em estado critico.

3

PLATAFORMA WEB

3.1 Introdução

Nesta secção será feita uma apresentação da plataforma web desenvolvida no âmbito da presente dissertação. O desenvolvimento realizado e implementado ao longo desta tese, com a intenção de atingir os objetivos inicialmente propostos está descrito neste capítulo.

3.2 Requisitos da Plataforma

Como foi dito anteriormente o objetivo é desenvolver uma plataforma de manutenção preditiva que permita detetar anomalias que as máquinas possam vir a ocorrer no futuro. Tendo por base essas necessidades foram estabelecidos os seguintes requisitos:

1. Upload de dados
2. Upload de dados via Email
3. Envio de alertas
4. Visualização de dados
5. Download/Partilha de dados

3.2.1 Upload de Dados

Este requisito consiste em extrair os dados de um ficheiro de excel para os importar na base de dados. Para cumprir este requisito decidiu-se importar os dados através de duas vias, ou seja, o utilizador pode escolher a forma como quer importar os dados: importar através da plataforma ou através do envio de um email para um endereço específico. Para importar os

dados em ambos os métodos serão usados os ficheiros excel. A *Spinerg* lê os valores dos metais presentes no fluído e outras características (como viscosidade e quantidade de água) usando sensores cujos valores são depois exportados para um ficheiro do tipo excel.

Nesses ficheiros além dos valores das análises, também são guardadas informações acerca do cliente e da máquina e serão estes os dados que a plataforma terá de importar. Uma vez que o presente requisito requer guardar esses valores, então será necessário criar uma base de dados. Para fazer isso, primeiramente desenvolveu-se um modelo para a base de dados. Será necessário para guardar a informação dos clientes, os seus equipamentos e, claro, as análises realizadas para determinar o estado do equipamento. Também será imprescindível guardar os valores dos metais lidos para verificar a presença de metais nos lubrificantes.

No decorrer do desenvolvimento da tese, a empresa providenciou um exemplar de ficheiros em formato *Excel* onde estão presentes todos os dados da análise como os nomes dos clientes, os seus equipamentos, modelos destes últimos, e os dados sobre a análise em si.

	A
1	Client
2	ID Client
3	Machine
4	ID Machine
5	Description Machine
6	Sub Description Machine
7	ID Component Client
8	Manufacturer
9	Model
10	Code Machine
11	Condition
12	ID Sample
13	Start Date
14	Received Date
15	Date Finished
16	Oil
17	Oil in use
18	Machine Life (Unit)
19	Oil Life (Unit)
20	Volume Max (Unit)
21	Drain?
22	Comments
23	
25	**Metal Type 1**
26	Metal 1.1
27	Metal 1.2
28	Metal 1.3
29	Metal 1.4
30	**Metal Type 2**
31	Metal 2.1
32	Metal 2.2
33	Metal 2.3
34	Metal 2.4
35	Metal 2.5
36	Metal 2.6
37	Metal 2.7
38	Metal 2.8
39	Metal 2.9
40	Metal 2.10
41	Metal 2.11
42	**Metal Type 3**
43	Metal 3.1
44	Metal 3.2
45	Metal 3.3
46	Metal 3.4
47	Metal 3.5
48	Metal 3.6
49	Metal 3.7
50	Metal 3.8
51	Metal 3.9
52	Metal 3.10
53	**Metal Type 4**
54	Metal 4.1
55	Metal 4.2
56	Metal 4.3
57	Metal 4.4
58	Metal 4.5
59	Metal 4.6
60	Metal 4.7
61	**Metal Type 5**
62	Metal 5.1
63	Metal 5.2
64	Metal 5.3

Figura 8 - Exemplo de excel providenciado.

Deste modo irá abordar-se o desenvolvimento da base de dados. Da descrição da dissertação e, também, dos exemplares das análises no formato *excel* providenciadas pela empresa chegou-se ao seguinte modelo apresentado na imagem abaixo.

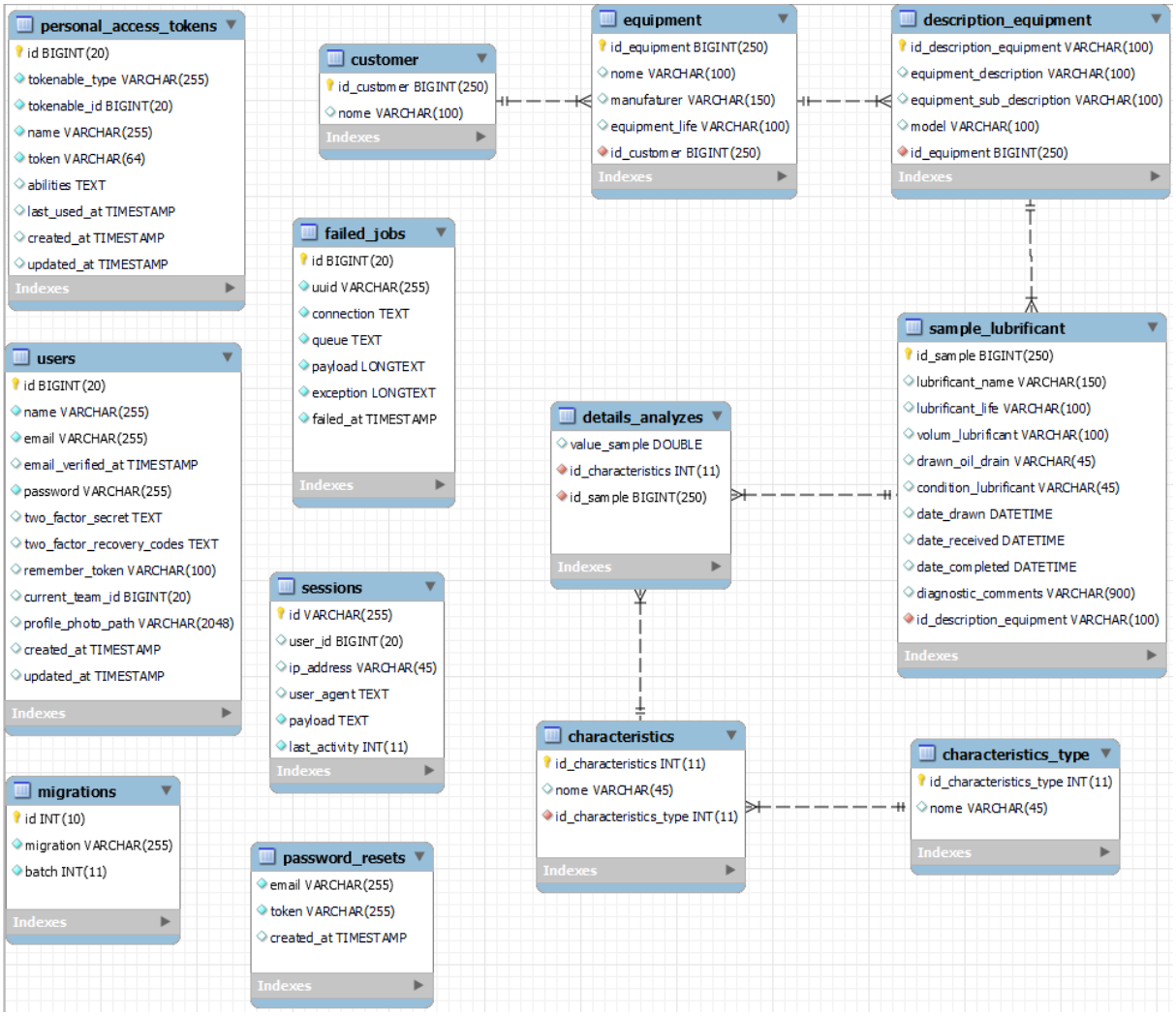


Figura 9 - Modelo da Base de dados do projeto.

A figura acima apresentada representa a base de dados do projeto. Quando um projeto é criado, o comando *php artisan migrate* é executado por padrão, criando diversas tabelas para autenticação, incluindo personal_access_tokens, users, migrations, failed_jobs, sessions e password_resets. Já as demais tabelas foram criadas de acordo com a análise feita à descrição e objetivos da dissertação. Como foi mencionado na introdução da tese, o objetivo é desenvolver uma plataforma colaborativa que permita analisar as amostras ao óleo lubrificante e classificar uma análise aos mesmos. Ao observar o *excel*, é possível extrair informações de

clientes, seus equipamentos, e informações relativas à análise da amostra. Portanto foi elaborada uma tabela para os clientes e para os seus equipamentos que contêm um id de identificação e nome. Nos ficheiros de *excel* providenciados pela empresa notou-se que os clientes tinham máquinas com o mesmo nome, mas o modelo destas eram diferentes e cada máquina tem um código associado, código esse que é representado na linha 10 do *excel*.

Deste modo, foi elaborada uma tabela chamada *description_equipment* que permite guardar o referido código como chave primária desta tabela. A tabela *sample_lubrificant* contém informações relativa à amostra do lubrificante, isto é, a datas de início e fim da análise, o resultado da amostra assim como o seu diagnóstico. Como se pode visualizar no *excel*, observa-se o nome do tipo de metais como por exemplo a linha 25 do *excel* e logo a seguir os metais de acordo com o seu tipo. Assim, foram elaboradas duas novas tabelas characteristics_type para guardar o tipo de metais e a tabela characteristics que servirá para guardar os seus nomes.

De modo que o metal fique relacionado com o seu tipo de metal adequado, e uma vez que um tipo de metal pode ter um ou mais metais, definiu-se uma relação de um para muitos entre as duas tabelas, criando uma chave estrangeira na tabela characteristics permitindo assim que o metal possa ter um tipo de metal associado. Já a tabela details_analyzes contem o valor das quantidades em ppm de cada metal detetado no óleo.

Vai-se agora abordar a forma como é feita a importação dos dados. Além de se ter a possibilidade de importar os dados através da plataforma, existe ainda mais um requisito que permite ao utilizador fazer o upload de dados através do envio de email. Este requisito será abordado mais à frente uma vez que a forma de importação dos dados para a base de dados é semelhante ao da importação através da plataforma. Independentemente do método, o utilizador usa o ficheiro excel dando início à importação dos dados.

Considere-se o excerto do ficheiro de excel apresentado anteriormente. A ideia é percorrer o excel linha a linha na primeira coluna e à medida em que vai percorrendo essas linhas os valores são recolhidos e guardados em variáveis para que depois possam ser guardados na base de dados voltando à primeira linha passando para a coluna seguinte terminando quando a primeira linha da coluna seguinte estiver vazia. No entanto, deve-se evitar guardar dados que já se encontrem na base de dados.

De acordo com os ficheiros de excel providenciados pela empresa, existem análises cujo nome cliente e a máquina são repetentes e considerando a base de dados criada anteriormente, deve-se evitar adicionar o mesmo nome e id do cliente assim como as suas máquinas sendo somente necessário estabelecer estes dados à análise feita à máquina, ou seja, se se

detetar que o nome do cliente e/ou a máquina deste já se encontra na base de dados, os dados que serão guardados serão só os referentes à análise como o id da análise e os valores lidos pelos sensores. Outro caso a ser evitado é importar as mesmas análises, ou seja, análises com o mesmo id. Basicamente dados cujo seu identificador já se encontra na base de dados devem ser desprezados a fim de evitar dados repetidos na base de dados passando à coluna seguinte do excel. O diagrama da figura seguinte explica de forma simplista o processo de importação dos dados.

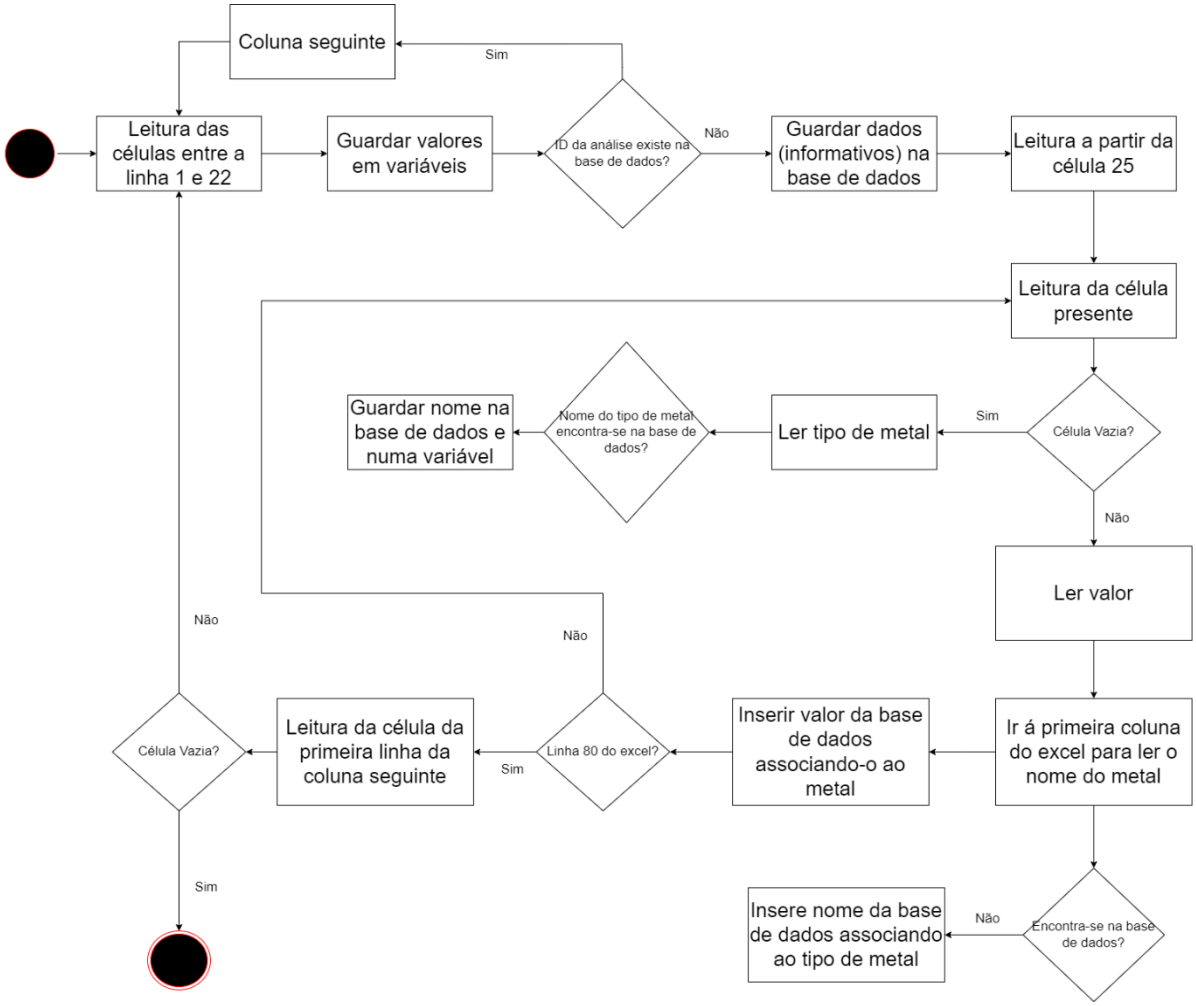


Figura 10 - Diagrama de atividade que explica o fluxo de importação de dados.

Após colocar o ficheiro para processamento, começa-se por ler a célula B1, isto é, a primeira linha da segunda coluna e vai lendo linha a linha dessa coluna até à linha 71 passando depois para a primeira linha da coluna seguinte. À medida que vai passando por estas linhas, os dados serão guardados em variáveis onde serão usadas em queries para meter na base de

dados. Os primeiros dados a serem guardados são aqueles que são de caracter informativo, ou seja, da linha 1 até à linha 22 do excel. O código escrito para extrair cada dado do excel foi o seguinte:

```
$customer_name = $spreadsheet->getActiveSheet()->toArray()[$i][$j];
$customer_number = $spreadsheet->getActiveSheet()->toArray()[$i + 1][$j];

$equipment_name = $spreadsheet->getActiveSheet()->toArray()[$i + 2][$j];
$equipment_number = $spreadsheet->getActiveSheet()->toArray()[$i + 3][$j];
$equipment_sub_description = $spreadsheet->getActiveSheet()->toArray()[$i + 4][$j];
$equipment_description = $spreadsheet->getActiveSheet()->toArray()[$i + 5][$j];
$equipment_manufacturer = $spreadsheet->getActiveSheet()->toArray()[$i + 7][$j];
$equipment_model = $spreadsheet->getActiveSheet()->toArray()[$i + 8][$j];

$lubrificant_name = $spreadsheet->getActiveSheet()->toArray()[$i + 15][$j];

$lubrificant_date_drawn = $spreadsheet->getActiveSheet()->toArray()[$i + 12][$j];
$lubrificant_date_received = $spreadsheet->getActiveSheet()->toArray()[$i + 13][$j];

$lubrificant_life = $spreadsheet->getActiveSheet()->toArray()[$i + 18][$j];
$lubrificant_analyst_code = $spreadsheet->getActiveSheet()->toArray()[$i + 9][$j];

$sample_id = $spreadsheet->getActiveSheet()->toArray()[$i + 11][$j];
$condition = $spreadsheet->getActiveSheet()->toArray()[$i + 10][$j];
$volum_lubrificant = $spreadsheet->getActiveSheet()->toArray()[$i + 19][$j];
$date_completed_sample = $spreadsheet->getActiveSheet()->toArray()[$i + 14][$j];
$drawn_oil_drain = $spreadsheet->getActiveSheet()->toArray()[$i + 20][$j];
$diagnostic_comments = $spreadsheet->getActiveSheet()->toArray()[$i + 21][$j];
```

Foi utilizada uma biblioteca de PHP que possibilita a leitura e a escrita de ficheiros excel e de outros for006Datos conforme pode ser observado. Uma vez que se pretende extrair os dados das células usou-se o código descrito acima. Considere-se o seguinte exemplo:

```
$customer_name = $spreadsheet->getActiveSheet()->toArray()[$i][$j];
```

Neste exemplo, a célula que está na linha *$i* na coluna *$j* será guardada numa variável denominada *$customer_name*. Após guardar estes dados, verificou-se se o *id* atual da análise já se encontra na base de dados. Em caso de afirmativo, passa-se para a coluna seguinte, caso comtrário envia-se os valores retirados do excel para a base de dados. Para se verificar se alguns dados como por exemplo, o *id* da análise já existe na base de dados, recorreu-se ao seguinte código:

```
$customer_found = \App\Models\Customer::where('id_customer', $customer_number)->first();
$equipment_found = \App\Models\Equipment::where('id_equipment', $equipment_number)->first();
$description_found = \App\Models\Description::where('id_description_equipment', $lubrificant_ana-
lyst_code)->first();
$sample_id_found = \App\Models\Sample::where('id_sample', '=', $sample_id)->first();
```

Este código permite verificar se os valores lidos no excel já se encontram na base de dados. Com estas variáveis basta fazer uma validação de que os dados já existem usando o ciclo *if else*. O código para esse efeito está apresentado abaixo.

```
if (empty($sample_id_found)) {

    if (empty($customer_found)) {
        $sqlQueryCustomer = "INSERT INTO customer(id_customer, nome) VALUES('$customer_number', '$cus-
tomer_name')";
        $connect->query($sqlQueryCustomer);
    }
    if (empty($equipment_found)) {
        $sqlQueryEquipment = "INSERT INTO equipment(id_equipment, nome, manufaturer, id_customer)
            VALUES('$equipment_number', '$equipment_name', '$equipment_manufacturer', '$customer_num-
ber')";
        $connect->query($sqlQueryEquipment);
    }

    if (empty($description_found)) {
        $sqlQueryDescription = "INSERT INTO description_equipment(id_description_equipment, equip-
ment_description, equipment_sub_description, model, id_equipment)
            VALUES('$lubrificant_analyst_code', '$equipment_description', '$equipment_sub_description',
'$equipment_model', '$equipment_number')";

        $connect->query($sqlQueryDescription);
    }

    $convert_date_drawn = strtotime($lubrificant_date_drawn);
    $date_drawn = date("Y-m-d H:i:s A", $convert_date_drawn);

    $convert_date_received = strtotime($lubrificant_date_received);
    $date_received = date("Y-m-d H:i:s A", $convert_date_received);

    $convert_date_completed = strtotime($date_completed_sample);
    $date_completed = date("Y-m-d H:i:s A", $convert_date_completed);
```

```
    $sqlQuerySample = "INSERT INTO sample_lubrificant(id_sample, lubrificant_name, lubrificant_life,
volum_lubrificant, drawn_oil_drain, condition_lubrificant, date_drawn, date_received, date_completed,
diagnostic_comments, id_description_equipment)
    VALUES('$sample_id', '$lubrificant_name', '$lubrificant_life', '$volum_lubrificant',
'$drawn_oil_drain', '$condition', '$date_drawn', '$date_received', '$date_completed', '$diagnostic_com-
ments', '$lubrificant_analyst_code')";
    $connect->query($sqlQuerySample);
} else {
    continue;
}
```

O código está dentro de um ciclo for que irá percorrer todas as linhas da coluna atual. A primeira condição a ser validada é se o *id* da análise atual já se encontra na base de dados. Em caso de afirmativo, ele passa para a coluna seguinte, caso contrário a análise é registada na base de dados. Durante o seu registo, é verificado se o cliente e a sua máquina também já se encontram na base de dados para que se evite repetição dos dados, no entanto estes estarão relacionados com a análise. Considere-se o seguinte código:

```
 $sqlQuerySample = "INSERT INTO sample_lubrificant(id_sample, lubrifi-
cant_name, lubrificant_life, volum_lubrificant, drawn_oil_drain, condi-
tion_lubrificant, date_drawn, date_received, date_completed, diagnostic_com-
ments, id_description_equipment)
VALUES('$sample_id', '$lubrificant_name', '$lubrificant_life', '$volum_lubrif-
icant', '$drawn_oil_drain', '$condition', '$date_drawn', '$date_received',
'$date_completed', '$diagnostic_comments', '$lubrificant_analyst_code')";
$connect->query($sqlQuerySample);
```

Ao executar o código acima descrito pode-se verificar que a máquina lida na coluna atual do excel é associada à análise. A variável denominada *$lubrificant_analyst_code* (esta variável contém um código de identificação de um equipamento específico) é inserida na tabela *sample_lubrificant* na coluna *id_description_equipment* que como se pode perceber na Figura 8 é uma coluna correspondente à tabela *description_equipment*.

Esta tabela contém uma coluna chamada *id_equipment* que é uma coluna correspondente à tabela *equipment*. Já esta tabela contém uma coluna chamada *id_customer* que é uma coluna correspondente à tabela *customer*. Desta forma consegue associar cada análise ao cliente e à sua máquina impedindo assim de se adicionar dados repetidos na base de dados. Após o tratamento destes dados, deve-se inserir os metais e os seus valores.

Pelo que se pode ver, a aplicação está implementada para saber o que cada linha de cada coluna significa, isto é, a aplicação sabe, por exemplo, que todas as primeiras linhas (linha

1 de todas as colunas) estão os nomes dos clientes, as segundas linhas (linha 2) estão o *id* do cliente e por aí adiante. Esta implementação foi inicialmente pensada para implementar o resto das informações das análises, contudo, o problema estaria em pegar nos valores referentes aos metais. Isto porque de acordo com os documentos *Excel* providenciados pela empresa, alguns continham determinados metais que outros ficheiros de excel não tinham o que se pode presumir que, no futuro a empresa opte por adicionar/remover um metal do excel. Significa isto que a implementação descrita anteriormente não serve uma vez que depois ter-se-ia a informação errada na base de dados relativamente aos valores dos metais.

Uma solução óbvia é adaptar a funcionalidade para capturar todos os valores, garantindo que apenas valores correspondentes ao metal correto sejam lidos. Isto pode ser feito lendo o valor na célula atual e, em seguida, ir buscar o nome do metal na primeira coluna da mesma linha. De seguida, os valores e os nomes dos metais podem ser inseridos na base de dados. Para isso será necessário validar duas situações: primeiro averiguar se a célula atual tem algum valor ou se tem o caracter "-" (este caracter significa que não se conseguiu ler o valor do metal).

A segunda situação a validar passa por verificar se tanto o metal como o seu tipo já existem na base de dados. Na primeira situação por vezes a célula atual pode encontrar-se vazia. Na Figura 8 verifica-se que existem essas linhas vazias como por exemplo as linhas 25, 30, 42 ou 56. Esta informação contém apenas o nome do tipo de metais, que são posteriormente colocadas na tabela *characteristics_type* e que, depois, usar-se-á a chave primária relativa ao tipo de metal. Como a tabela *characteristics_type* se relaciona com a da *characteristics* com uma relação de um para muitos (isto porque um tipo de metal tem vários metais), a chave primária do tipo de metais, será uma chave estrangeira na tabela *characteristics* no qual estarão os metais cuja chave estrangeira desses metais será igual à chave primária do seu tipo de metal de acordo com o excel.

Caso seja identificado o valor do metal na célula, este será inserido na tabela *details_analyzes* cujo a chave estrangeira será a *id_characteristics* de modo a identificar o metal. Como foi referido anteriormente, as células podem conter o caracter "-" sendo que este caracter significa que não foi possível ler o valor. Para este caso, a aplicação lê como se fosse zero pois o atributo que serve para guardar os valores dos metais é do tipo DOUBLE, ou seja, o atributo obriga a que a informação a ser guardada tenha de ser um número com casas decimais e, por isso, este caracter nunca será guardado no atributo da tabela *details_analyzes*. Depois de colocar toda esta informação nas respetivas tabelas e atributos/colunas, deve-se

garantir que, no futuro, não se deve inserir os mesmos tipos de metais e metais na base de dados de modo a não ficarem repetidos. O código abaixo apresentado importa os dados referentes aos metais e aos seus valores.

```
for ($line = 24; $line < 79; $line++) {
    if ($current_characteristics_type == 'Iron vs TBN') {
        continue;
    }

    if (empty($spreadsheet->getActiveSheet()->toArray()[$line][$col])) {
        if (!empty($spreadsheet->getActiveSheet()->toArray()[$line][0]) && $spreadsheet->getAc-
tiveSheet()->toArray()[$line][$col] != 0) {
            $characteristicType = $spreadsheet->getActiveSheet()->toArray()[$line][0];
            $current_characteristics_type = $characteristicType;
            if ($current_characteristics_type == 'Iron vs TBN') {
                continue;
            }
            $characteristicTypeFound = \App\Models\CharacteristicsType::where('nome', $characteristic-
Type)->first();
            if (empty($characteristicTypeFound)) {
                $characteristicTypeCount += 1;
                $sqlQueryCharacteristicType = "INSERT INTO characteristics_type(id_characteris-
tics_type, nome) VALUES('$characteristicTypeCount', '$characteristicType')";
                $connect->query($sqlQueryCharacteristicType);
            }
            continue;
        }

        if (!empty($spreadsheet->getActiveSheet()->toArray()[$line][0]) && $spreadsheet->getAc-
tiveSheet()->toArray()[$line][$col] == 0) {
            $characteristic = $spreadsheet->getActiveSheet()->toArray()[$line][0];
            $characteristicFound = \App\Models\Characteristics::where('nome', $characteristic)-
>first();
            if (empty($characteristicFound)) {
                $characteristicTypeFound = \App\Models\CharacteristicsType::where('nome', $cur-
rent_characteristics_type)->first();
                $characteristicCount += 1;
                $currenIdCharacteristic = $characteristicCount;
                $sqlQueryCharacteristic = "INSERT INTO characteristics(id_characteristics,
nome,  id_characteristics_type) VALUES('$characteristicCount', '$characteristic', '$characteristicType-
Found->id_characteristics_type')";
                $connect->query($sqlQueryCharacteristic);
            } else {
                $currenIdCharacteristic = $characteristicFound->id_characteristics;
            }
```

```
            $value = $spreadsheet->getActiveSheet()->toArray()[$line][$col];
            $sqlQueryValue = "INSERT INTO details_analyzes(value_sample, id_characteristics, id_sample)
            VALUES('$value', '$currenIdCharacteristic', '$sample_id')";
            $connect->query($sqlQueryValue);
        }

        continue;
    } else {
        $characteristic = $spreadsheet->getActiveSheet()->toArray()[$line][0];
        $characteristicFound = \App\Models\Characteristics::where('nome', $characteristic)->first();
        if (empty($characteristicFound)) {
            $characteristicTypeFound = \App\Models\CharacteristicsType::where('nome', $current_charac-
teristics_type)->first();
            $characteristicCount += 1;
            $currenIdCharacteristic = $characteristicCount;
            $sqlQueryCharacteristic = "INSERT INTO characteristics(id_characteristics, nome,  id_char-
acteristics_type) VALUES('$characteristicCount', '$characteristic', '$characteristicTypeFound->id_char-
acteristics_type')";
            $connect->query($sqlQueryCharacteristic);
        } else {
            $currenIdCharacteristic = $characteristicFound->id_characteristics;
        }
        $value = $spreadsheet->getActiveSheet()->toArray()[$line][$col];
        $sqlQueryValue = "INSERT INTO details_analyzes(value_sample, id_characteristics, id_sample)
            VALUES('$value', '$currenIdCharacteristic', '$sample_id')";
        $connect->query($sqlQueryValue);
    }
}
```

Primeiramente, verifica-se o conteúdo da célula atual e, se esta estiver vazia, tem-se duas condições a validar. A primeira é se a célula está mesmo sem conteúdo, e a segunda é se tem o valor zero no qual a aplicação deteta como se fosse vazio. Na primeira condição, é necessário recorrer à primeira coluna da linha da atual célula no qual contém o nome do tipo de metal. Esse nome será guardado numa variável e, através do model do *characteristicsType* (model criado para aceder à tabela *characteristicsType* na base de dados) irá verificar se o nome do tipo de metal já se encontra na respetiva tabela. Caso ainda não esteja, esse nome será inserido, caso contrário passa à célula seguinte. Na segunda condição, é porque simplesmente o metal cujo nome está situado na primeira coluna da linha da célula a ser lida tem valor embora seja zero. É necessário inserir o metal na base de dados, mas antes é necessário

saber qual o seu tipo de metal. Ao visualizar os documentos Excel, sabe-se que o nome do tipo de metal aparece sempre antes de um determinado metal.

Como foi dito na primeira condição o nome do metal é guardado na variável não só para depois fazer a validação se existe na base de dados, como também posteriormente para o associar aos metais nas células seguintes. Então, ao ler a célula atual, a aplicação está programada para ir à primeira coluna dessa célula buscar o nome do metal e caso este não exista, vai-se buscar o *id_characteristics_type* do tipo de metal para depois quando inserir o metal na respetiva tabela, associá-lo ao seu tipo de metal. Esta situação descrita também acontece para o caso em que a célula atual contém um determinado valor que não seja zero.

Nota ainda para a análise realizada nos documentos *Excel*, existem aproximadamente 74 linhas preenchidas, pois como foi dito, existem metais em determinados documentos que não existem em outros, daí o número de linhas ser diferente para cada ficheiro, contudo, a aplicação está implementada para ler até 79 para o caso de a empresa, se assim o entender, acrescentar mais metais lidos no equipamento. Se se quiser acrescentar mais para além dos 79, ter-se-á de alterar o número 79 que está no ciclo *for* para o número que se deseja.

3.2.2 Importação de Dados Via Email

Será abordada agora a forma com foi implementada o requisito da importação de dados via email. Este tipo de importação é ligeiramente diferente da descrita na importação via plataforma. A importação é feita através de um envio de email para um endereço específico enviando o ficheiro excel em anexo, verificando de seguida se o email contém o tal ficheiro excel e em caso de afirmativo, extrai-se o ficheiro para depois importar os dados do excel para a base de dados da mesma forma que se faz na importação via plataforma.

Para se implementar esta função precisar-se-á de criar um endereço de email e, também, criar duas tarefas de modo que uma possa fazer a verificação da existência do excel e, caso este exista, movê-la para uma pasta. Já a outra tarefa consiste na importação dos dados para a base de dados. Deve-se ter em conta que no futuro poderão existir imensos ficheiros Excel a verificar no email e, também, a importar, e ter milhares de ficheiros a importar, o que pode provocar congestionamentos na aplicação. Por isso, pensou-se na solução de criar *crons* para as tarefas descritas acima.

Uma *cron* é um comando do *Linux* que permite agendar tarefas a executar no futuro dependendo da periodicidade em que são configuradas como, por exemplo, programar uma das tarefas descritas acima de 30 em 30 minutos. Este comando é bastante útil na medida que permite automatizar funções sem ter a necessidade de interagir com a aplicação. Por isso, será

necessário criar uma *cron* para ler os emails no endereço e outra para a importação dos dados. A imagem a seguir apresentada mostra em formato de diagrama de atividades o funcionamento da *cron* que servirá para extrair os ficheiros de excel dos emails.

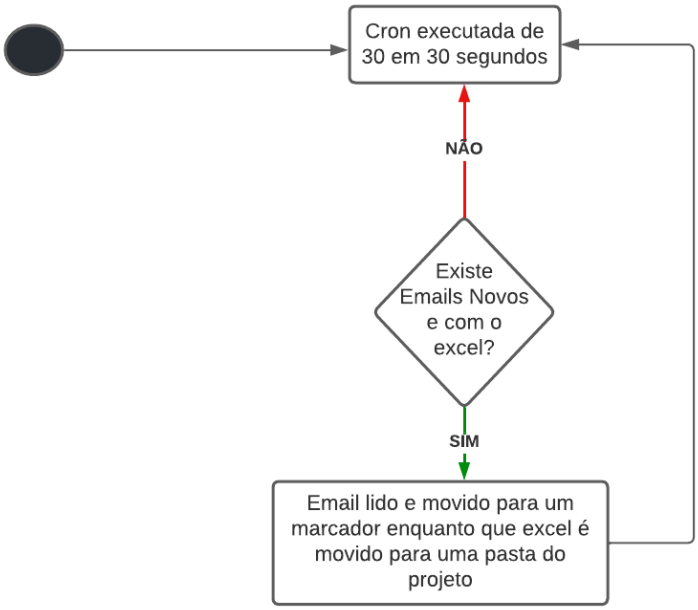


Figura 11 - Diagrama de atividade da cron que lê os emails e extrai os ficheiros de excel.

A primeira *cron* que será executada periodicamente irá verificar se o endereço de email contém emails que contenham o excel. Caso esses emails existam, o email é lido e movido para um marcador previamente criado a fim de evitar que esse email volte a ser lido. Já o excel será guardado numa pasta do projeto para ser processado mais tarde. A imagem abaixo mostra o funcionamento da segunda *cron* que tem como finalidade fazer o upload de dados dos ficheiros de excel presentes na pasta do projeto.

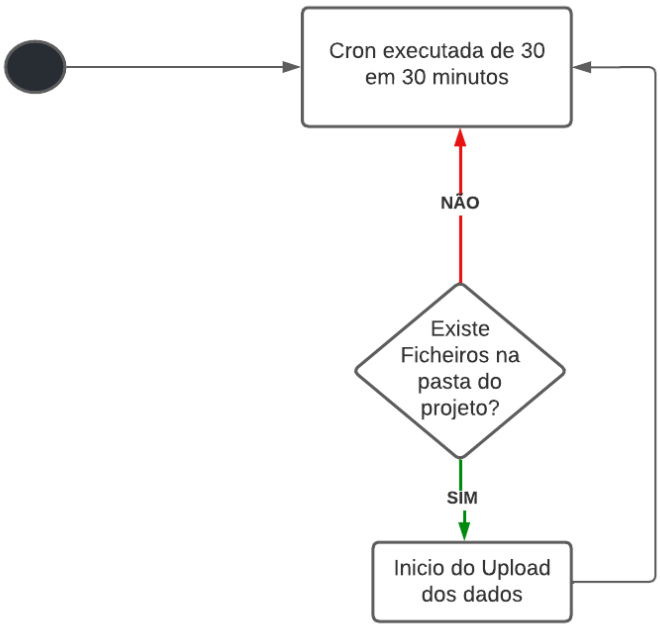


Figura 12 - Diagrama de atividade da cron que importa os dados.

A segunda *cron* consiste em iniciar o processo de importação de dados do ficheiro de excel. Inicialmente faz-se uma verificação se existem ficheiros de excel na pasta do projeto previamente definido e em caso de afirmativo, dá-se início ao processo de upload de dados, processo esse que foi descrito no requisito anterior. Vai-se agora abordar a forma como foi feita passo a passo o desenvolvimento deste requisito.

Necessita-se, primeiramente, de criar um endereço de email e, para efeitos de teste, criou-se o seguinte endereço: *carlos.teste.seixas@gmail.com*

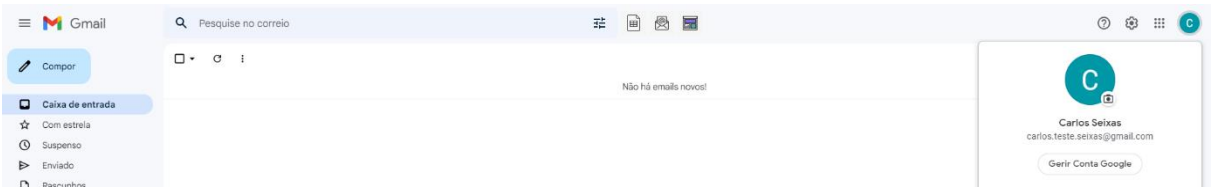


Figura 13 - Endereço de email que servirá para testar a importação via email

Após a criação do endereço, é necessário acedê-lo para que se possa recolher os ficheiros de excel. Deste modo e após algum estudo, chegou-se à conclusão de que o método IMAP é o que mais se adequa para a realização da importação via email. Para começar a usar este método deve-se ativá-lo no endereço e, para isso basta localizar o ícone das definições que está no canto superior direito (Figura 15) e clicar em "Ver todas as Definições".

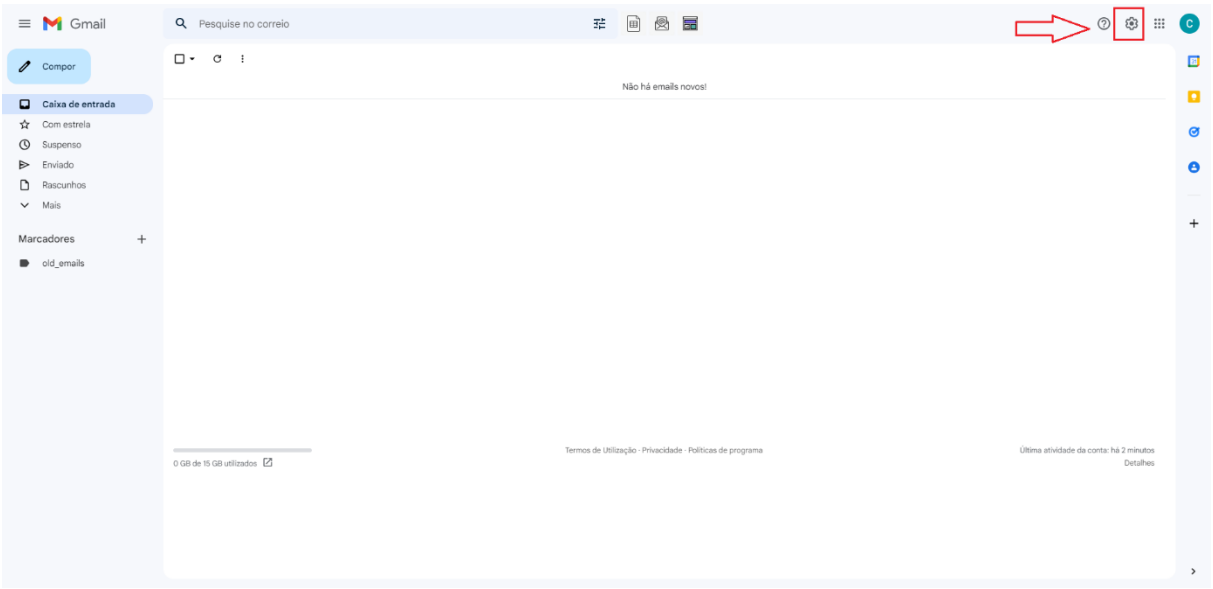


Figura 14 - Página do email para indicar onde se situa o ícone das definições

Uma vez na página das definições, ir ao separador do "Encaminhamento e POP/IMAP", ir a "Acesso IMAP" e, por fim, clicar em "Ativar IMAP" (Figura 15).

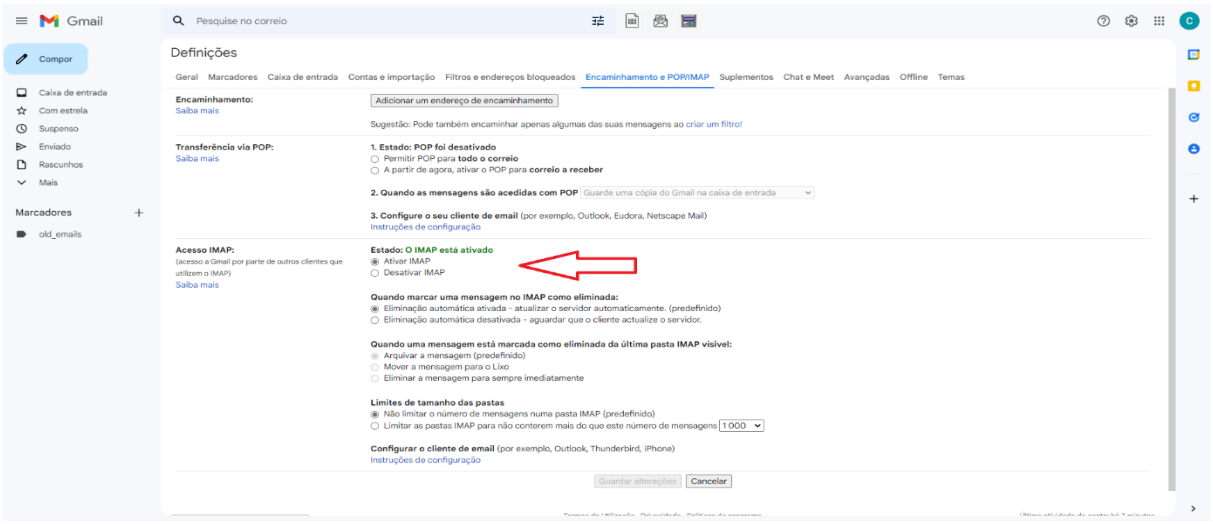


Figura 15 - Ativação do IMAP

Para acabar a configuração, necessita-se de criar as credenciais dos dispositivos para que estes possam aceder e enviar os ficheiros para o endereço. Para isso, na página principal do email clica-se no ícone do utilizador que está no canto superior direito e clica-se em "Gerir Conta Google".

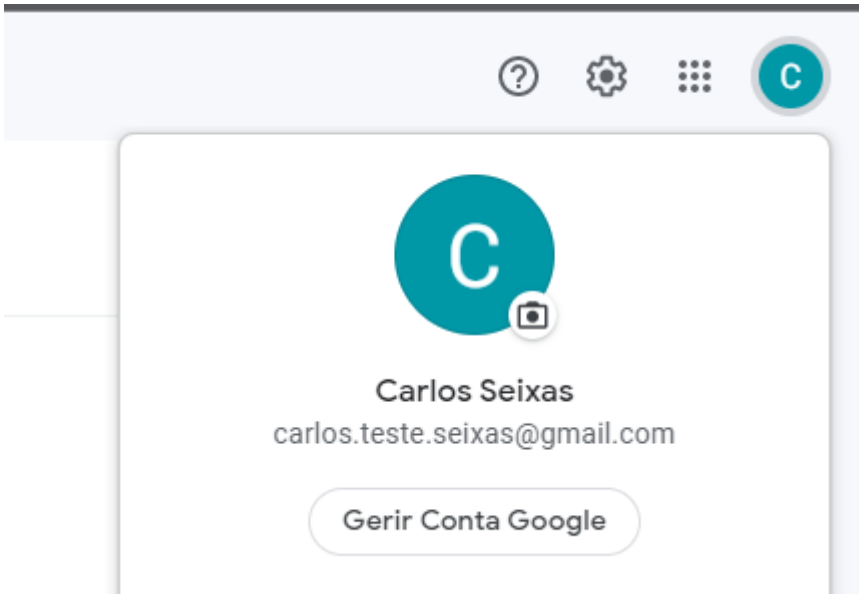


Figura 16 - Passo para permitir que os dispositivos acedam à conta

Após o clique, irá abrir uma nova janela onde se pode visualizar vários separadores no lado esquerdo. Clica-se no separador "Segurança", e selecionou-se a "Iniciar sessão no Google".

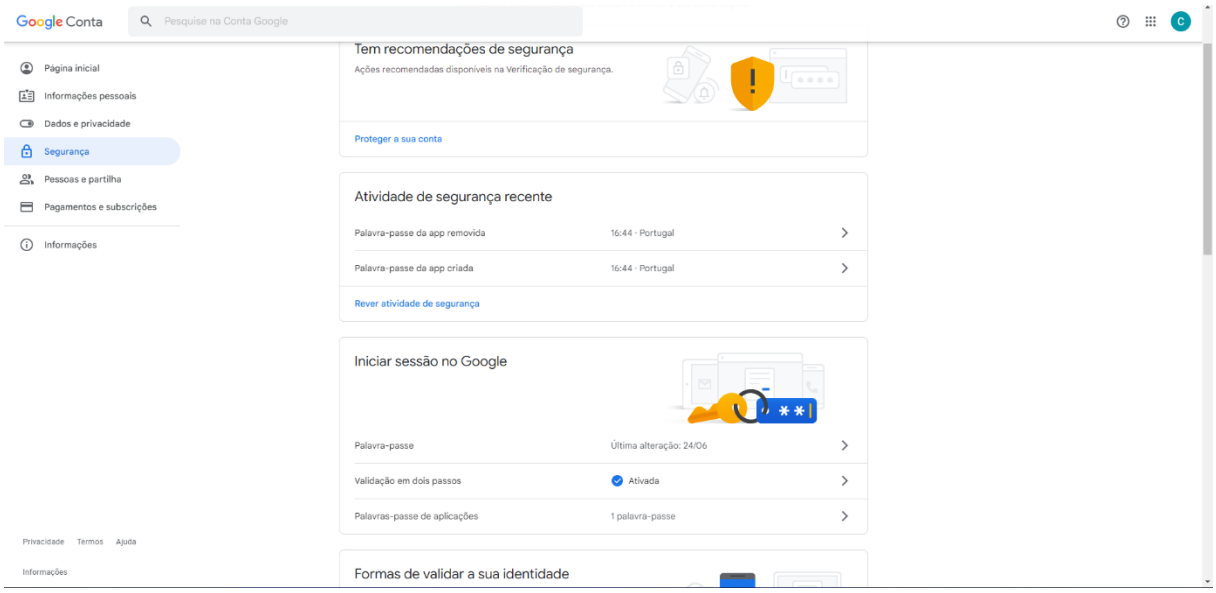


Figura 17 - Procedimento para iniciar IMAP

Em seguida, clica-se em "Palavra-passe da app criada" e, posteriormente, por motivos de segurança deve-se inserir a password do email. Após isso, é direcionado para outra página que vai permitir que outros dispositivos possam iniciar a sessão a partir da aplicação (Figura 18).

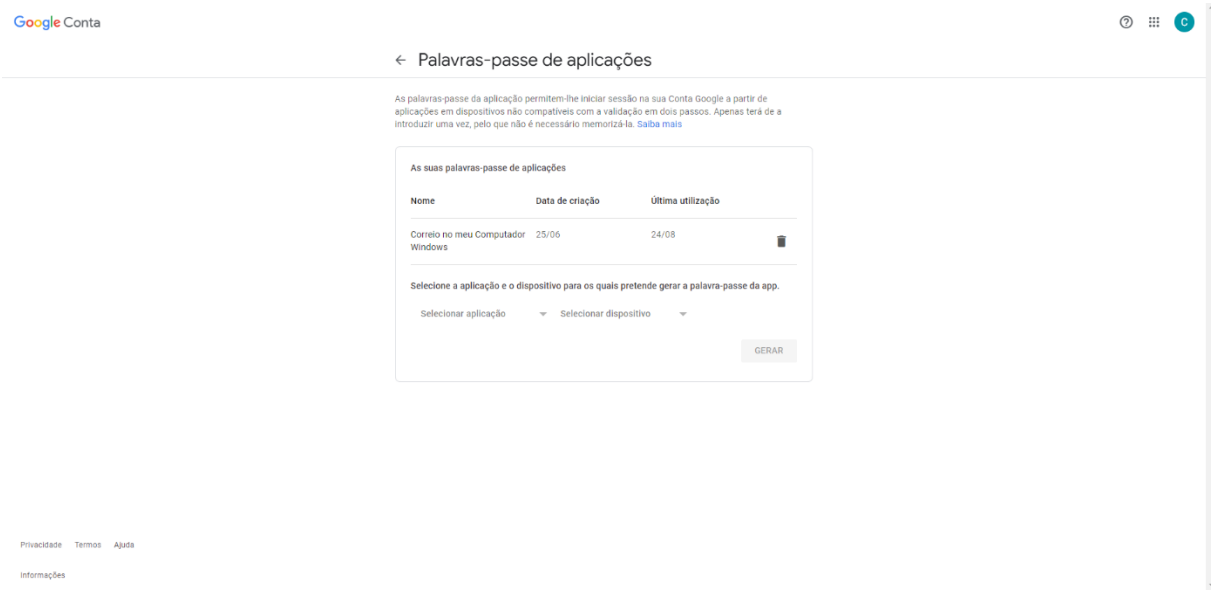


Figura 18 - Página em que se regista as passwords para se poder iniciar a sessão noutro dispositivo a partir de uma aplicação, neste caso o Correio

No *dropdown* onde diz "Selecionar aplicação" para o caso desta dissertação a aplicação será "Correio", já no *dropdown* "Selecionar dispositivo" será "Computador Windows" e, por fim, clicar em "Gerar". Ao clicar nesta opção, será exibida uma janela pop-up contendo uma palavra-passe de 16 caracteres, conforme mostrado na imagem abaixo. Essa palavra-passe será necessária para iniciar a sessão e ler os e-mails no código que será desenvolvido.

Uma vez configurado, pode-se começar por desenvolver um script que permita identificar os emails que continham os documentos Excel, recolher esses ficheiros e metê-los numa pasta. Como foi dito anteriormente, é necessário iniciar a sessão do endereço de email para que que se possa ler as suas mensagens. A melhor maneira é usando o *Internet Mail Access Protocol* ou abreviadamente IMAP.

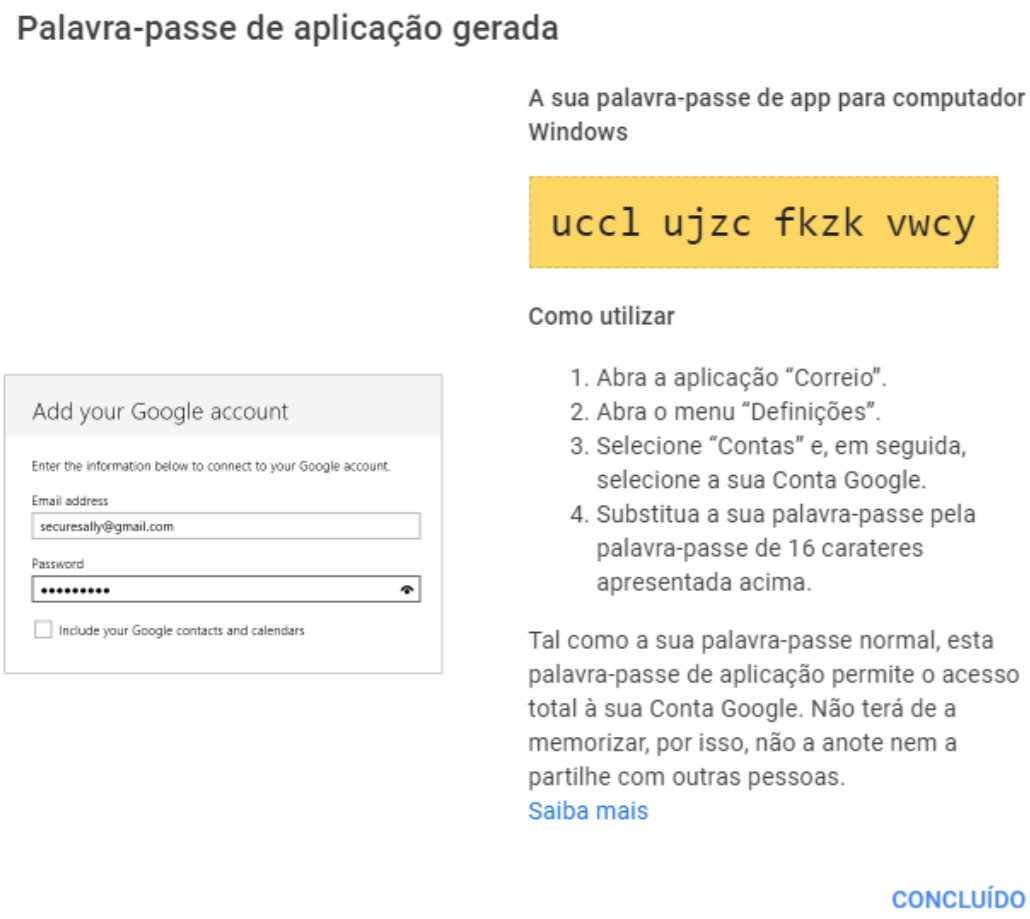


Figura 19 - Palavra-passe gerada que irá permitir iniciar a sessão.

O IMAP é um protocolo que permite aceder a uma conta de email [26]. Com o IMAP é possível procurar emails cujo estes tenham uma determinada palavra-chave no *subject* (Assunto em português). Como é hábito, para aceder a um email (por exemplo no Gmail) é necessário inserir o email e a palavra-passe.

```php
<?php
$host = '{imap.gmail.com:993/imap/ssl/novalidate-cert/norsh}INBOX';
        /* Your gmail credentials */
        $user = 'carlos.teste.seixas@gmail.com';
        $password = 'ucclujzcfkzkvwcy';

        /* Establish a IMAP connection */
        $conn = imap_open($host, $user, $password)
            or die('unable to connect Gmail: ' . imap_last_error());
```

Figura 20 - Credenciais para aceder ao endereço com o intuito de ler os emails que contém os ficheiros de excel

As linhas 8 e 9 do código apresentado na Figura 20 mostram como aceder à conta. A função *imap_open* permite iniciar a sessão da conta de email caso as três variáveis (*$host* que contém o servidor e o caminho de uma caixa de correio presente neste servidor, *$user* que é o endereço de email e a *$pass* que é uma palavra-passe definida que permite aceder ao email através de uma aplicação) estiverem corretas, a variável *$conn* irá retornar uma instância IMAP\Comnection permitindo, assim iniciar a sessão com sucesso. Caso contrário, o processo irá morrer (a função *die()* permite encerrar a aplicação) e retornará uma mensagem de erro a indicar motivo da falha de inicio de sessão. Referir ainda que a palavra-passe apresentada nas Figuras 19 e 20 são meramente ilustrativas e não correspondem ao endereço.

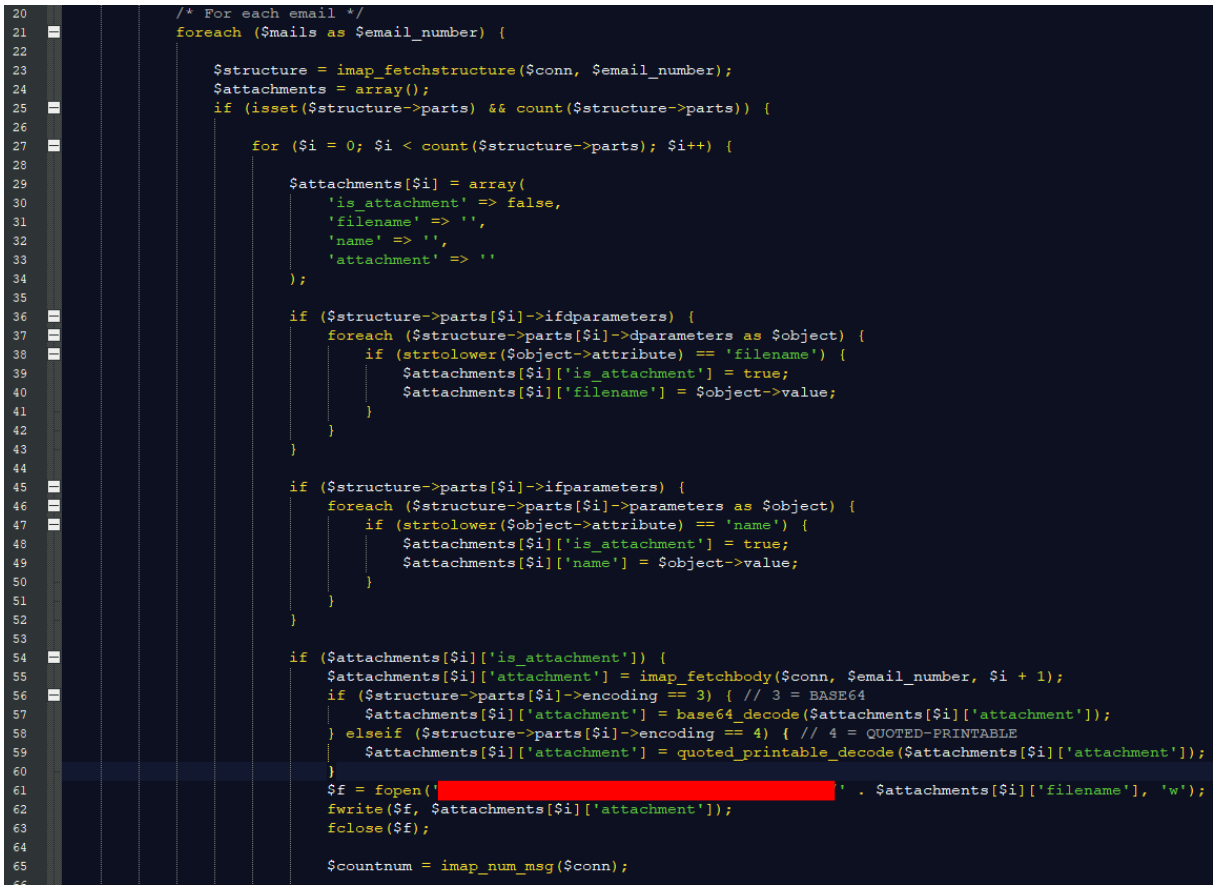
Após o acesso ao email, deve-se identificar quais os emails que contém os ficheiros a importar. De forma que os emails possam estar identificados na forma acima descrita deve-se ter em conta o seguinte código apresentado na figura abaixo.

```
11          /* Search emails from gmail inbox*/
12          $mails = imap_search($conn, 'SUBJECT "Import"');
13
14          /* loop through each email id mails are available. */
15          if ($mails) {
16
17              /* rsort is used to display the latest emails on top */
18              rsort($mails);
19
20              /* For each email */
21              foreach ($mails as $email_number) {
```

Figura 21 - Script que permite identificar os emails que contenham os ficheiros com os dados a importar.

A função apresentada na linha 12 da figura acima permite entrar na caixa de entrada atual e procurar por novos e-mails que contenham a palavra *Import* no assunto, permitindo, assim, identificar e-mails que contenham um ou mais arquivos Excel.

Depois da identificação dos emails, é altura de ler o corpo do email e identificar o(s) anexo(s) presentes na mensagem, recolhê-los e inseri-los numa pasta para depois importar os dados inseridos no excel.



```php
/* For each email */
foreach ($mails as $email_number) {

    $structure = imap_fetchstructure($conn, $email_number);
    $attachments = array();
    if (isset($structure->parts) && count($structure->parts)) {

        for ($i = 0; $i < count($structure->parts); $i++) {

            $attachments[$i] = array(
                'is_attachment' => false,
                'filename' => '',
                'name' => '',
                'attachment' => ''
            );

            if ($structure->parts[$i]->ifdparameters) {
                foreach ($structure->parts[$i]->dparameters as $object) {
                    if (strtolower($object->attribute) == 'filename') {
                        $attachments[$i]['is_attachment'] = true;
                        $attachments[$i]['filename'] = $object->value;
                    }
                }
            }

            if ($structure->parts[$i]->ifparameters) {
                foreach ($structure->parts[$i]->parameters as $object) {
                    if (strtolower($object->attribute) == 'name') {
                        $attachments[$i]['is_attachment'] = true;
                        $attachments[$i]['name'] = $object->value;
                    }
                }
            }

            if ($attachments[$i]['is_attachment']) {
                $attachments[$i]['attachment'] = imap_fetchbody($conn, $email_number, $i + 1);
                if ($structure->parts[$i]->encoding == 3) { // 3 = BASE64
                    $attachments[$i]['attachment'] = base64_decode($attachments[$i]['attachment']);
                } elseif ($structure->parts[$i]->encoding == 4) { // 4 = QUOTED-PRINTABLE
                    $attachments[$i]['attachment'] = quoted_printable_decode($attachments[$i]['attachment']);
                }
                $f = fopen('[redacted]' . $attachments[$i]['filename'], 'w');
                fwrite($f, $attachments[$i]['attachment']);
                fclose($f);

                $countnum = imap_num_msg($conn);
```

Figura 22 - Código de leitura do email e recolha dos anexos para uma pasta.

A figura acima, mostra como o email é lido e, também, como é feita a recolha dos anexos. Após a identificação dos anexos, estes são movidos para uma determinada pasta. Depois da leitura dos emails, é necessário garantir que estes não voltem a serem lidos. Portanto, a solução escolhida foi marcar o email como lido e, de seguida, este seria movido para um marcador. Estes marcadores são criados no gmail e são úteis em termos de organização de emails recebidos caso se queira voltar a lê-los/usá-los no futuro. Nesse sentido, criou-se um marcador chamado *old_emails* (para efeitos de teste) como se pode ver na Figura 23.

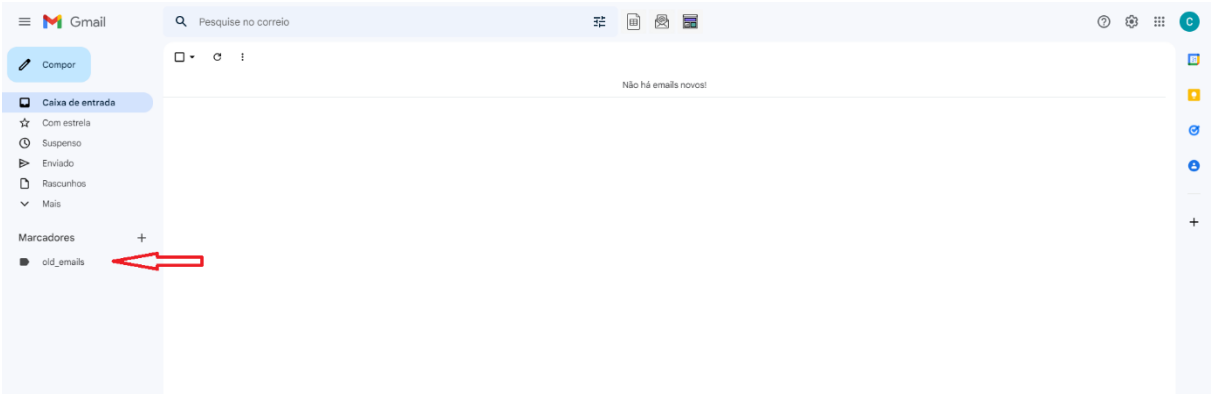


Figura 23 - Marcador Criado.

A figura a seguir apresentada mostra como mover o email para o marcador que se deseja guardar o email.

```
65                          $countnum = imap_num_msg($conn);
66
67                          //move the email to our saved folder
68                          $mail_port = "imap.gmail.com:993/imap/ssl";
69                          $mbox = imap_open("{imap.gmail.com:993/imap/ssl}INBOX", $user, $password) or die("Error opening mailbox: ".imap_last_error());
70                          $imapresult = imap_mail_move($mbox, '1:' . $countnum, 'old_emails');
71                          // dd($imapresult, $countnum);
72                          if ($imapresult == false) {
73                              die(imap_last_error());
74                          }
75                          imap_close($mbox, CL_EXPUNGE);
76                      }
77                  }
78              }
79
80              /* Retrieve specific email information*/
81              $headers = imap_fetch_overview($conn, $email_number, 0);
82
83          } // End foreach
84      } //endif
85
86
87
88      /* imap connection is closed */
89      imap_close($conn);
90
91
92
93  ?>
```

Figura 24 - Excerto de código de como mover um email para o marcador *old_emails*.

A linha 70 da figura acima mostra como é feita a transferência do email lido para o marcador *old_emails*. A função *imap_mail_move* permite mover um email para uma caixa de correio. Deste modo impede-se que os emails sejam lidos mais do que uma vez.

Uma vez recolhidos os ficheiros excel, estes já podem ser importados e para isso usou-se um processo semelhante ao usado na importação via plataforma só que desta vez o ficheiro está presente numa pasta e não numa página html da plataforma. Portanto, desenvolveu-se um script que permitisse aceder à pasta para ir buscar os ficheiros e dar início ao processo de importação.

```
22  $folderName = __DIR__ . '/public/filesExcel/';
23  echo $folderName    . '<BR/>';
24  $iterator = new DirectoryIterator($folderName);
25  echo "folder=" . $iterator;
26  foreach ($iterator as $fileinfo) {
27      $fileName = $fileinfo->getFilename();
28
29      echo "file=" . $fileName . PHP_EOL;
30      if (strpos($fileName, '.xlsx', 0) > 0) {
31          echo " Trata do ficheiro ";
32          app(App\Http\Controllers\ImportSampleController::class)->tratarAnalises($folderName, $fileName);
33          echo " Processou o ficheiro ";
34          unlink($folderName . '/' .  $fileName);
35          echo " Tratamento do ficheiro concluido ";
36      }
37  }
```

Figura 25 - Código que permite o tratamento dos ficheiros de excel.

Na Figura 25 é possível visualizar como é feito o processamento de importação dos dados. A linha 24 permite que se aceda à pasta onde estão inseridos os ficheiros. Depois disso recolha-se o primeiro ficheiro (linha 27) e verifica se o tipo de ficheiro é o excel através da condição *if* (linha 30) e, caso seja verdadeiro ele irá para uma função que irá importar os dados presentes no excel para a base de dados (linha 32).

Após finalizar a importação, começa o mesmo processo para o seguinte ficheiro caso exista mais algum. Como se pode observar pela linha 32 da figura acima, ele irá executar o código presente no ficheiro denominado *ImportSampleController*. O código que se desenvolveu neste ficheiro é o mesmo ao que foi apresentado no requisito anterior. Após o processo de importação estar completo, o ficheiro é apagado da pasta e caso exista mais ficheiros, volta a repetir-se o mesmo procedimento, caso contrário, a tarefa é dada por terminada voltando a ser executada depois de um determinado tempo definido pelo utilizador.

3.2.3 Envio dos Alertas

Após o upload dos dados, a plataforma está apta para indicar qual a condição de cada uma das máquinas. Existem três condições: a condição *Normal* que significa que a máquina está perfeitamente operacional, condição *Attention* que indica que o equipamento necessita de alguma supervisão, isto é, o óleo começa a ter determinadas características, ainda que mínimas, que estão a começar a seguir uma tendência de que a máquina terá uma falha em breve e a condição *Action* em que a máquina necessita mesmo de manutenção urgentemente uma vez que os metais presentes no fluido apresentam índices elevados. Nesse sentido, propôs-se a ideia de enviar um alerta por email quando se detetar máquinas se encontram na condição *Action*.

Para esse requisito usou-se o protocolo *SMTP* (*Simple Mail Transfer Protocol*) que é semelhante ao protocolo *IMAP* que foi abordado na secção anterior. Vai-se agora apresentar um exemplo de como é feito o envio de um email. Um exemplo de teste que se usou inicialmente para testar este protocolo foi através de um pedido ao servidor, retirar quantas máquinas estão em situação critica (condição *Action*) e de seguida, enviar o email. Primeiramente deve-se iniciar o servidor do endereço de email. De referir que o endereço de email usado é o mesmo que se usou na secção anterior (*carlos.teste.seixas@gmail.com*). O código seguinte mostra como se inicializa o servidor.

```
 //Server settings
        $mail->SMTPDebug = SMTP::DEBUG_SERVER;                      //Enable verbose debug output
        $mail->isSMTP();                                            //Send using SMTP
        $mail->Host       = 'smtp.gmail.com';                     //Set the SMTP server to send through
        $mail->SMTPAuth   = true;                                   //Enable SMTP authentication
        $mail->Username   = 'carlos.teste.seixas@gmail.com'; //'c.mor-
gado@fct.unl.pt';               //SMTP username
        $mail->Password   =   //SMTP password
        $mail->SMTPSecure = PHPMailer::ENCRYPTION_STARTTLS;   //PHPMailer::ENCRYPTION_SMTPS;      //En-
able implicit TLS encryption
        $mail->Port       = 587; //443;                                //TCP port to connect to; use 587
if you have set `SMTPSecure = PHPMailer::ENCRYPTION_STARTTLS`
```

Após a Inicialização, deve-se configurar o conteúdo do email, isto é, qual o assunto do email, e a mensagem em si. O código abaixo mostra como se faz essa configuração.

```
//Content
        $mail->isHTML(true);                  //Set email format to HTML
        $mail->Subject = 'EMERGENCY';
        $mail->Body    = "Existem $queryAction máquinas que necessitam de atenção imediata. É necessá-
rio marcar a manutenção dos equipamentos urgentemente.";
        $mail->AltBody = 'This is the body in plain text for non-HTML mail clients';

        $mail->send();
        echo 'Message has been sent';
        } catch (Exception $e) {
            echo "Message could not be sent. Mailer Error: {$mail->ErrorInfo}";
        }
```

Neste caso o que ficou definido foi que o email a ser enviado teria como assunto o texto *EMERGENCY*. Já o corpo é o que está definido na variável *$mail->Body*. Nessa variável pode-se visualizar uma outra variável denominada *$queryAction*. Esta variável foi previamente definida e, como se pode ver no código abaixo, retorna como resultado da query a quantidade de máquinas que têm como condição *Action*, ou seja, em situação crítica que necessitam de manutenção com urgência.

```
$queryAction = DB::table('sample_lubrificant')
        ->select('sample_lubrificant.condition_lubrificant')
        ->where('sample_lubrificant.condition_lubrificant', '=', 'Action')
        ->groupBy('sample_lubrificant.condition_lubrificant')->count();
```

Após definir um endereço que permite enviar o email para o endereço acima referido, este recebeu um email que está apresentado na figura seguinte.

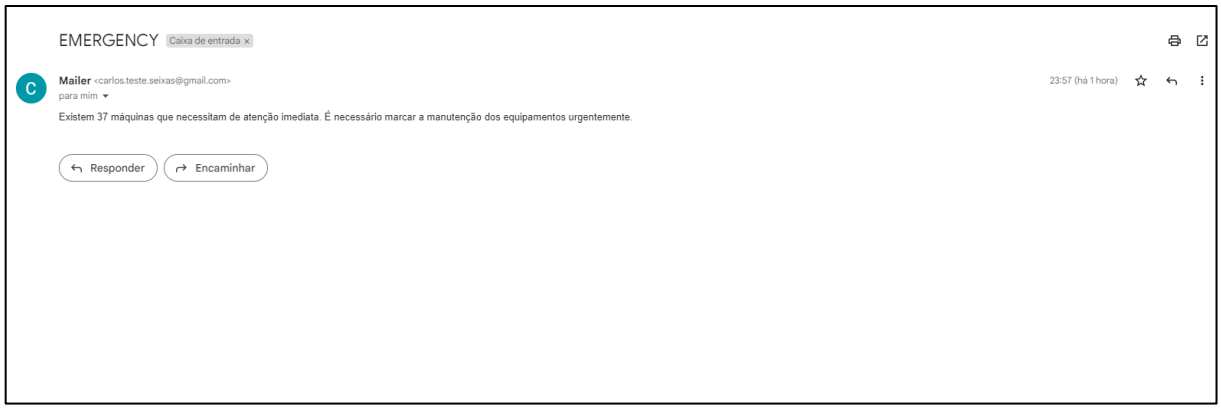


Figura 26 - Email recebido após a execução de um pedido ao servidor.

3.2.4 Visualização e Download de Dados

Este requisito consiste em apresentar dados ou informações na plataforma através de tabelas e gráficos com o intuito de exibi-los de forma simples e clara ao utilizador. A visualização de dados é importante no sentido na compreensão dos dados especialmente quando existe um grande volume de dados.

Este requisito é bastante útil pois ao analisar os dados consegue-se identificar padrões e tendências que possam indicar que se pode estar perante uma situação atípica. Nestes casos costuma-se usar gráficos de qualquer tipo para uma melhor compreensão e entender a causa da situação a analisar. Para a dissertação a visualização de dados será bastante útil

principalmente para verificar em que condição se encontra cada equipamento e analisar a evolução de um determinado metal de uma máquina. Portanto, os dados mais relevantes são os valores dos metais de uma determinada máquina de um cliente. A visualização de dados consiste em ir buscar os dados à base de dados (*backend*) e exibi-los numa página *web* (*front-tend*).

A abordagem usada para este requisito foi usar a arquitetura MVC que já foi abordada no capítulo 2. Para expor os dados na plataforma foi utilizada a arquitetura *MVC* que já foi abordada no capítulo 2 desta dissertação. A primeira página criada foi a listagem todos os clientes da empresa. Iniciou-se a criação do *Model* que servirá para fazer a ligação com a tabela *customer*, tabela referente aos clientes, cria-se o *controller* que fará a ligação entre o *Model* e o *View*. Por último, para o *View* bastará criar um ficheiro do tipo blade. Para criar este ficheiro basta criá-lo manualmente dentro da pasta *resources/views* que é onde serão criadas todas as páginas que serão desenvolvidas em html e também de *css* para definir estilos. O *blade* é um sistema de templates do Laravel que permite criar páginas de template em projetos PHP. Para criar um *Model* deve-se executar o seguinte comando:

php artisan make:model Customer

Deve-se associar à tabela *customer*. A figura abaixo mostra como se deve fazer essa associação (linha 12).

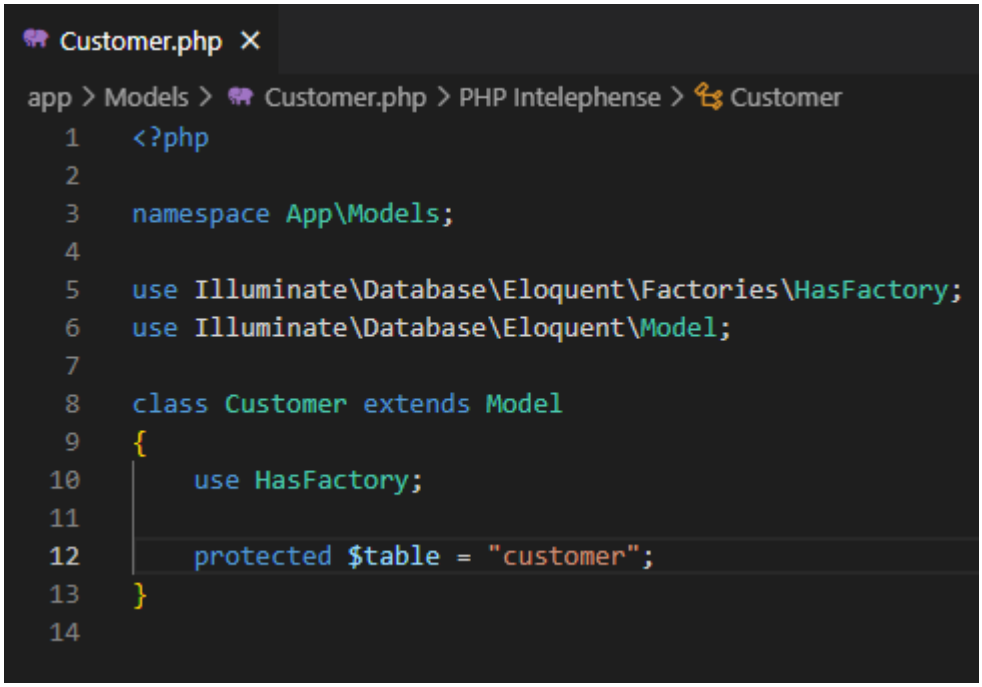


Figura 27 - Model que irá fazer a manipulação dos dados relativos à tabela customer.

Para criar o *controller,* é executar o mesmo procedimento que o *Model*. Corre-se o seguinte comando:

php artisan make:controller CustomerController

Uma vez que se quer listar todos os clientes na página html, optou-se por desenvolver o seguinte código no *Controller*.

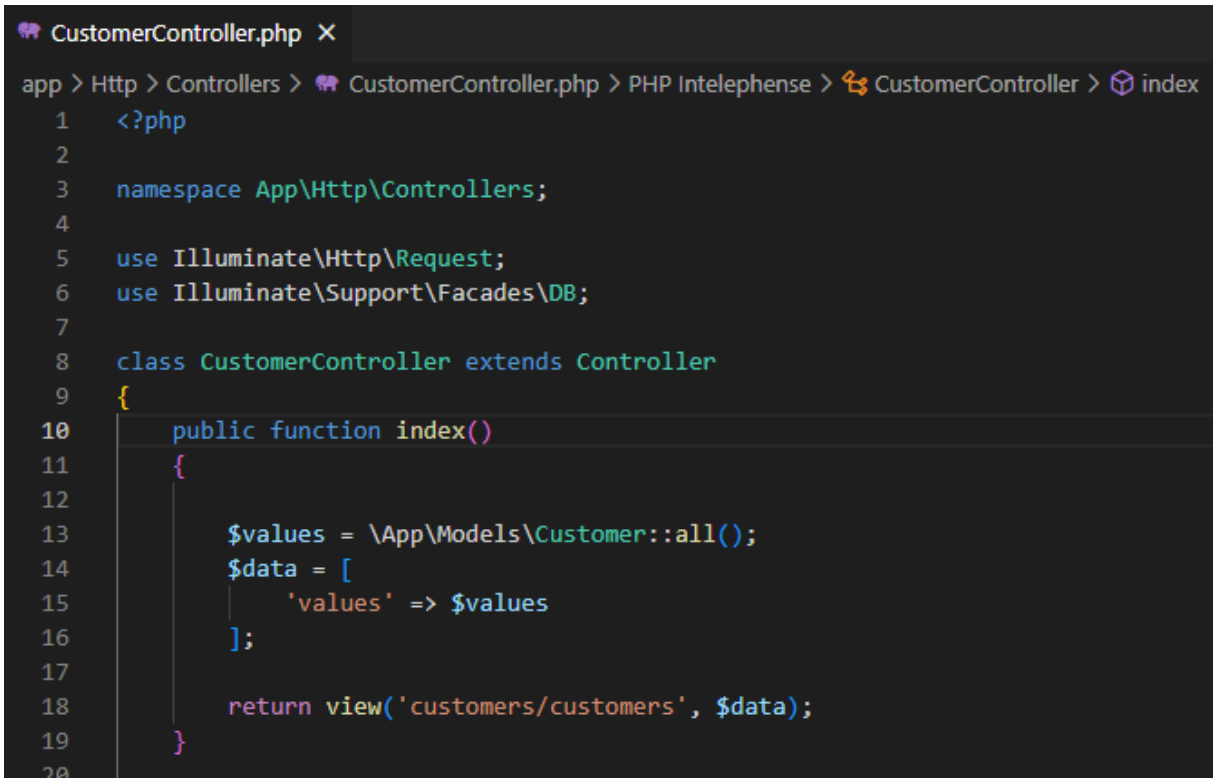


Figura 28 - Controlador relativo ao Cliente

A função *all()* na linha 13 da figura acima serve para passar todos os dados dos clientes para a variável *$values* (neste caso será o número do cliente (chave primária que permite identificar o cliente) e o nome). A função *view* na linha 18 servirá para usar a variável no ficheiro *customers.blade.php* que se situa a pasta *customers*. Deve-se definir no primeiro parâmetro para qual ficheiro blade se quer enviar a variável e no segundo parâmetro, mete-se a variável. Contudo, na Figura 28 vê-se que que se meteu a variável $values dentro de um *array* denominado *$data*. Isto é bastante útil na medida em que se quer enviar mais variáveis para o ficheiro blade.

Após a criação do *Controller*, é necessário definir uma rota no ficheiro *web.php*, uma vez que é necessário definir qual o *Controller* que será executado partir da rota indicada. Nesse sentido, criou-se a seguinte rota para a função *index* do controlador *CustomerController* como se pode verificar na imagem abaixo (linha 35).

```
35  Route::get('/customer', [App\Http\Controllers\CustomerController::class, 'index'])->name('customer');
36  Route::get('/customer/{id}', [App\Http\Controllers\CustomerController::class, 'find'])->name('customerId');
```

Figura 29 - Rota que irá executar a função index do CustomerController.

A linha 35 faz com que o que está na função *index* no *CustomerController* seja executado que, como se pode ver na Figura 28, irá retornar o ficheiro *customers* que está dentro da pasta chamada *customers* (por uma questão de organização uma vez que será necessário criar mais páginas relativas aos clientes da *Spinerg*) que por sua vez está na pasta *view*.

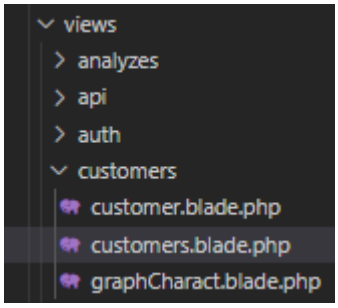


Figura 30 - Ficheiro customers.blade.php onde será executado o html onde serão listados todos os clientes na página.

Por fim, deve-se desenvolver o visual de como se pretende ficar listado, neste caso, todos os clientes na página. Decidiu-se apresentar os dados dos clientes numa tabela e, uma vez que a única informação que se tem sobre os clientes são o seu ID e o nome, então a decidiu-se optar por exibir estes dados. Para isso, usou-se uma tag chamada <*table*> que organizar os dados em linhas e colunas. Para saber distinguir quais as são as colunas e quais as linhas, dentro do <*table*>, existem duas tags: <*thead*> que é a tag usada para definir as colunas e a <*tbody*> que define as linhas. Para se definir cada coluna, usa-se a tag <*th*> dentro da tag <*thead*> e para definir cada linha coloca-se a tag <*td*> dentro da tag <*tbody*>. Antes

de se definir uma coluna ou linha, usa-se a tag <*tr*> que é uma tag que serve para definir estilos e comportamentos das colunas e linhas. O código está apresentado na figura seguinte.

```
42          <table align="center" class="display nowrap" id="table_data">
43              <thead>
44                  <tr class="text-left font-bold">
45                      <th class="border px-6 py-4" style="text-align: center;">Número Cliente</th>
46                      <th class="border px-6 py-4" style="text-align: center;">Cliente</th>
47                  </tr>
48              </thead>
49              <tbody>
50                  @foreach ($values as $value)
51                      <tr id="row-{{ $value->id_customer }}" class="td-buttons" style="cursor: pointer" data-id="{{ $value->id_customer }}">
52                          <td class="border px-6 py-4" style="text-align: center;">{{ $value->id_customer }}</td>
53                          <td class="border px-6 py-4" style="text-align: center;">{{ $value->nome }}</td>
54                      </tr>
55                  @endforeach
56              </tbody>
57          </table>
```

Figura 31 - Código html que irá listar os dados dos clientes numa tabela.

Na imagem apresentada, é possível observar o uso do ciclo foreach, que se encontra entre as linhas 50 e 55. O ciclo contém a variável *$values*, previamente definida no *Customer-Controller*, e é responsável por iterar sobre os valores de um array. Veja-se o que acontece ao acrescentar o código *dd($values)* (a função *dd* é uma função do Laravel que é uma mistura entre *var_dump* e *die* e permite que se escreva a(s) variável(eis) que se mete na função e cessar a aplicação) entre as linhas 13 e 14 na Figura 28. O resultado após a execução é apresentado na imagem abaixo.

A variável *$values* é um array que contém todos os dados da tabela *customer* e em cada posição tem-se o ID do cliente (id_customer) e o seu nome (nome). O *id_customer* e o *nome* que se visualiza na imagem são respetivamente as colunas *id_customer* e *nome* da tabela *customer*. Por essa razão é que nas linhas 52 e 53 na Figura 31 tem-se a variável $values a apontar para *id_customer* e, também para *nome* para que se possa apresentar os dados na tabela na página. Com isto conseguiu-se desenvolver a tabela em si faltando torná-la mais intuitiva. mais dinâmica, mais responsiva e, para isso, deve-se implementar em *Javascript*.

O *Javascript* será útil na medida em que se quer criar conteúdo que se atualiza de forma dinâmica. Neste caso, quer-se criar uma tabela responsiva que permita organizar e apresentar os dados com recursos como pesquisa, ordenação e filtro. Assim sendo, adicionou-se um plugin de *jQuery* chamado *Datatable*. Com este plugin pode-se organizar a apresentação dos dados da tabela por página e, também tornar a tabela responsiva à medida que se vai acrescentando caracteres no filtro. Para a sua implementação identificou-se a tabela como

table_data (Linha 42 da Figura 31) isto para que se possa identificar a tabela em que se quer usar o *datatable*. A imagem apresentada abaixo mostra como se deve implementar o *datatable* em *Javascript*.

```
7
8   <script>
9       jQuery(document).ready(function($) {
10        $('#table_data').DataTable();
11        $(document).on('click', '#table_data tbody tr button', function() {
12          $("#table_data tbody tr").html("");
13          $("#table_data tbody tr").html($(this).closest("tr").html());
14        });
15      });
16  </script>
```

Figura 32 - Código da implementação do Datatable.

A linha 9 da figura acima serve para começar a usar o plugin só quando a página html for totalmente carregada, A linha 10 serve para que a tabela com a identificação *table_data* pela propriedade *id* da tag *table* (linha 42 da Figura 31) comece a executar o *datatable*.

Com o *Javascript*, foi possível implementar uma funcionalidade em que ao clicar-se numa das linhas da tabela, o utilizador é direcionado para uma outra página onde aparece informações mais detalhadas acerca do cliente.

Nesse sentido, identificou-se a classe das linhas da tabela como *td-buttons* e o *data-id* como *{{ $value->id_customer }}* que serão usadas no *Javascript* e na identificação do cliente (ver Figura 31 na linha 51) da mesma forma que se fez para o *datatable*. O *data-id* irá definir cada linha da tabela como o *id* do cliente presente dessa linha. A imagem abaixo mostra a implementação desta funcionalidade (da linha 10 à 12).

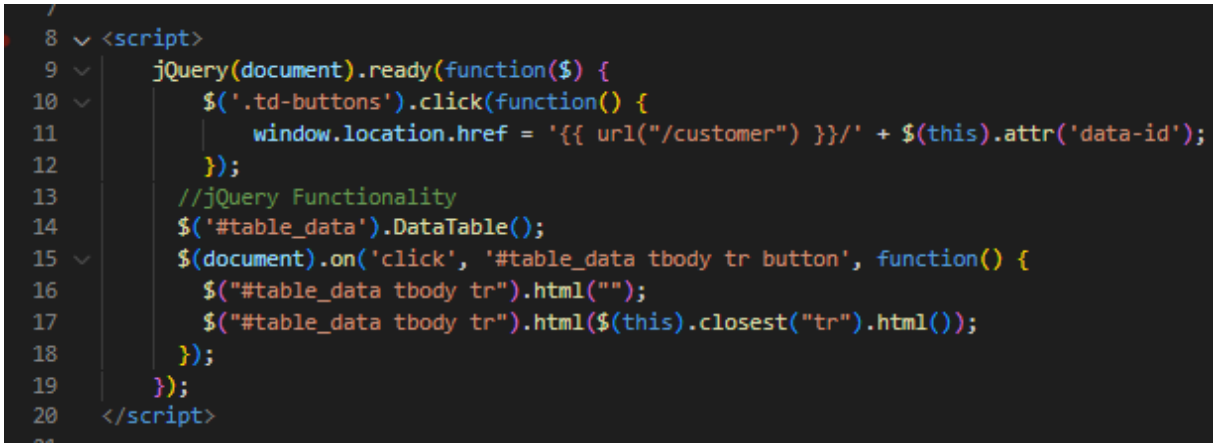


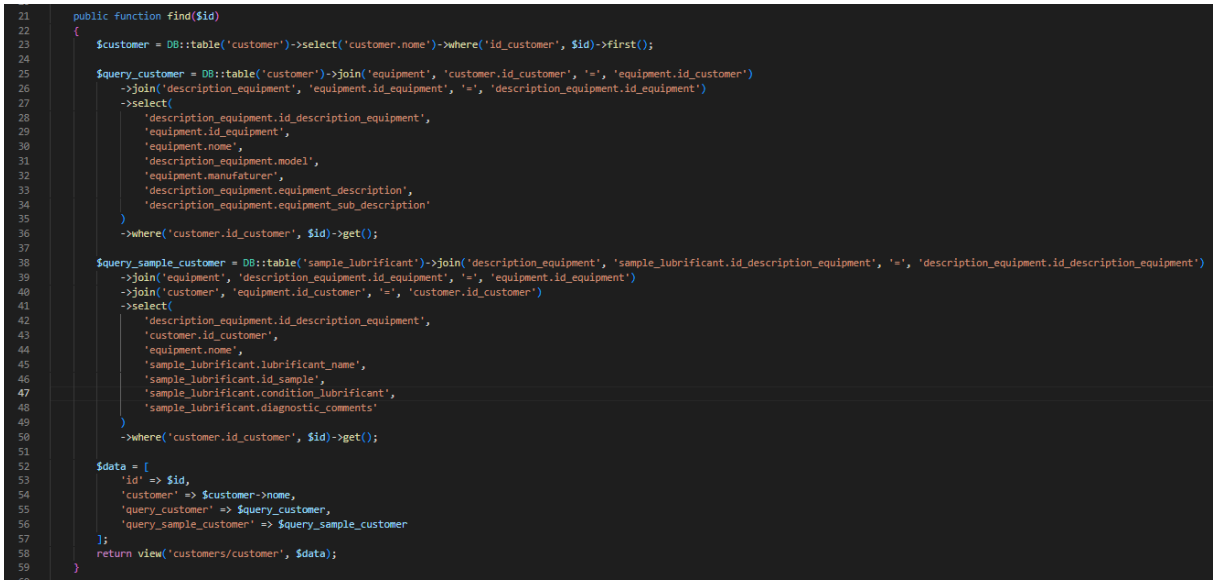
Figura 33 - Código em Javascript que ao clicar numa linha da tabela, é-se direcionado para uma outra página.

A linha 10 da figura acima mostra-nos que quando se clicar numa das linhas da tabela, ocorre-se um evento que neste caso irá para a página com a rota */customer/{id do cliente}*. Referir que na linha 11 está o seguinte código:

$(this).attr('data-id')

O que é passado neste código é o *id* da linha. Portanto, ao clicar-se numa linha da tabela, obtém-se o *id* da linha para, depois, ser-se direcionado para uma outra página, neste caso para a página de detalhe do cliente.

Para criar a referida página que contem informações acerca do cliente (e outras páginas semelhantes) foi feito um procedimento bastante semelhante como se fez para a criação da página da listagem dos clientes. Aproveitou-se o *CustomerController* para criar uma função



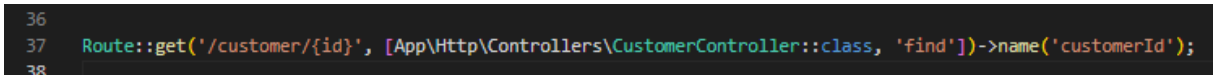
```php
public function find($id)
{
    $customer = DB::table('customer')->select('customer.nome')->where('id_customer', $id)->first();

    $query_customer = DB::table('customer')->join('equipment', 'customer.id_customer', '=', 'equipment.id_customer')
        ->join('description_equipment', 'equipment.id_equipment', '=', 'description_equipment.id_equipment')
        ->select(
            'description_equipment.id_description_equipment',
            'equipment.id_equipment',
            'equipment.nome',
            'description_equipment.model',
            'equipment.manufaturer',
            'description_equipment.equipment_description',
            'description_equipment.equipment_sub_description'
        )
        ->where('customer.id_customer', $id)->get();

    $query_sample_customer = DB::table('sample_lubrificant')->join('description_equipment', 'sample_lubrificant.id_description_equipment', '=', 'description_equipment.id_description_equipment')
        ->join('equipment', 'description_equipment.id_equipment', '=', 'equipment.id_equipment')
        ->join('customer', 'equipment.id_customer', '=', 'customer.id_customer')
        ->select(
            'description_equipment.id_description_equipment',
            'customer.id_customer',
            'equipment.nome',
            'sample_lubrificant.lubrificant_name',
            'sample_lubrificant.id_sample',
            'sample_lubrificant.condition_lubrificant',
            'sample_lubrificant.diagnostic_comments'
        )
        ->where('customer.id_customer', $id)->get();

    $data = [
        'id' => $id,
        'customer' => $customer->nome,
        'query_customer' => $query_customer,
        'query_sample_customer' => $query_sample_customer
    ];
    return view('customers/customer', $data);
}
```

Figura 34 - Função do controlador que permite recolher informações do cliente à base de dados.

que permita recolher toda a informação relativa ao cliente que se quer expor na página como se pode ver na figura a seguir apresentada.

Para a recolha dos dados desta vez usou-se o *QueryBuilder* que, tal como o nome indica, é um construtor de *queries* e permite executá-las na base de dados. Veja-se a *query* da variável *$customer* da linha 23 da imagem acima. O resultado da variável será o resultado da query que neste caso será o nome do cliente com o *id* definido no parâmetro da função.

As variáveis *$query_customer* e *$query_sample_customer* também são igualmente *queries,* mas que desta vez irão conter dados que estão noutras tabelas. Por isso são usadas várias vezes a função *join* pois permite que os dados de várias tabelas sejam combinados com base relação que existe entre elas. No caso da *$query_customer*, como se quer listar todos os equipamentos do cliente então necessita-se dos dados que estão na tabela *equipment* e na tabela *description_equipment* isto porque o cliente pode ter duas ou mais máquinas cujo *id_equipment* é o mesmo, mas os modelos são diferentes e esse modelo está identificado na tabela *description_equipment*. Já o *$query_sample_customer* os dados que importam mostrar na página estão nas tabelas *sample_lubrificant*, *description_equipment*, *equipment* e *customer*. Depois de desenvolver as *queries*, meteu-se todas as variáveis no *array $data* para enviar as variáveis para o *view customer.blade.php* (linha 58). Após o desenvolvimento da função, definiu-se uma rota no *web.php*.



```php

Route::get('/customer/{id}', [App\Http\Controllers\CustomerController::class, 'find'])->name('customerId');

```

Figura 35 - Rota definida para a página com informações de um determinado cliente.

De seguida foi desenvolver a página em html e também em *javascript* para implementar o *datatable* nas tabelas.

```
<div class="row" style=" padding-top: 50px">
    <table align="center" style="width: 100%; text-align:center" id="datas">
        <thead>
            <tr style="background-color: #004e79; color: white">
                <th class="border px-6 py-4" style="text-align: center;">Código</th>
                <th class="border px-6 py-4" style="text-align: center;">ID do Equipamento</th>
                <th class="border px-6 py-4" style="text-align: center;">Equipamento</th>
                <th class="border px-6 py-4" style="text-align: center;">Modelo</th>
                <th class="border px-6 py-4" style="text-align: center;">Manufatura</th>
                <th class="border px-6 py-4" style="text-align: center;">Descrição</th>
                <th class="border px-6 py-4" style="text-align: center;">Sub Descrição</th>
            </tr>
        </thead>
        <tbody>
            @foreach($query_customer as $customer)
            <tr style="background-color: white" class="buttons-equipment" style="cursor: pointer" data-id="{{ $customer->id_equipment }}">
                <td class="border px-6 py-4" style="text-align: center;">{{ $customer->id_description_equipment }}</td>
                <td class="border px-6 py-4" style="text-align: center;">{{ $customer->id_equipment }}</td>
                <td class="border px-6 py-4" style="text-align: center;">{{ $customer->nome }}</td>
                <td class="border px-6 py-4" style="text-align: center;">{{ $customer->model }}</td>
                <td class="border px-6 py-4" style="text-align: center;">{{ $customer->manufaturer }}</td>
                <td class="border px-6 py-4" style="text-align: center;">{{ $customer->equipment_description }}</td>
                <td class="border px-6 py-4" style="text-align: center;">{{ $customer->equipment_sub_description }}</td>
            </tr>
            @endforeach
        </tbody>
    </table>
</div>
```

Figura 36 - Código em html referente à listagem de todos os equipamentos que o cliente possui.

```
<div style=" padding-top: 50px">
    <table style="text-align:center" id="table_data">
        <thead>
            <tr style="background-color: #004e79; color: white">
                {{-- <th class="border px-6 py-4">ID</th> --}}
                <th class="border px-6 py-4" style="text-align: center;">ID</th>
                <th class="border px-6 py-4" style="text-align: center;">Código</th>
                <th class="border px-6 py-4" style="text-align: center;">Lubrificante</th>
                <th class="border px-6 py-4" style="text-align: center;">Condição</th>
                <th class="border px-6 py-4" style="text-align: center;">Diagnóstico</th>
            </tr>
        </thead>
        <tbody>
            @foreach($query_sample_customer as $sample)
            <tr id="row-{{ $sample->id_sample }}" class="buttons-sample" style="cursor: pointer" data-id="{{ $sample->id_sample }}">
                <td class="border px-6 py-4" style="text-align: center;">{{ $sample->id_sample }}</td>
                <td class="border px-6 py-4" style="text-align: center;">{{ $sample->id_description_equipment }}</td>
                <td class="border px-6 py-4" style="text-align: center;">{{ $sample->lubrificant_name }}</td>
                <td class="border px-6 py-4" style="text-align: center;">{{ $sample->condition_lubrificant }}</td>
                <td class="border px-6 py-4" style="text-align: center; width: 100%">{{ $sample->diagnostic_comments }}</td>
            </tr>
            @endforeach
        </tbody>
    </table>
</div>
```

Figura 37 - Código em html referente à listagem de todas as análises realizadas aos equipamentos do cliente.

Como se pode observar, a implementação da página foi muito semelhante com a diferença na recolha dos dados em que neste caso foi mais complexo uma vez que se necessitou de dados contidos noutras tabelas. Depois aplicou-se a mesma lógica que foi outrora aplicada

na página de listagem dos clientes, usando o ciclo *foreach* de modo a ir preenchendo na tabela os dados que estão presentes nos *arrays* *$query_customer* e *$query_sample_customer*.

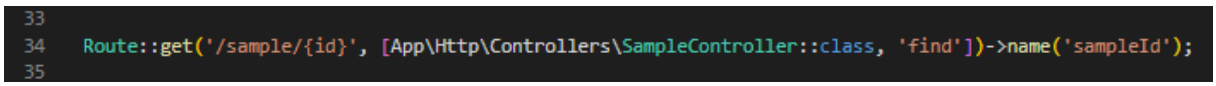
No caso dos dados relativos às análises, pretendia-se apresentar na página o número de identificação de cada uma (*id* da análise que na base de dados está definido como *id_sample*), o código da máquina (está na coluna *id_description_equipment* da tabela *description_equipment*), o lubrificante analisado (coluna *lubrificant_name* da tabela *sample_lubrificant*), a condição do lubrificante (coluna *condition_lubrificant*, tabela *sample_lubrificant*), ou seja, o resultado da análise e o comentário ao diagnóstico ao óleo (coluna *diagnostic_comments*, tabela *sample_lubrificant*). Já para o caso dos dados relativos às máquinas do cliente, quis-se exibir o código da máquina (coluna *id_description_equipment* da tabela *description_equipment*), a identificação desta (o código e identificação da máquina são diferentes), o nome (coluna *nome* da tabela *equipment*), o modelo da máquina (coluna *model* da tabela *equipment*), a manufatura do equipamento (coluna *manufaturer*, tabela *equipment*), descrição da máquina (coluna e*quipment_description,* tabela *equipment*) e uma descrição adicional da máquina (coluna *equipment_sub_description*, tabela *equipment*).

Com a conclusão da página que exibe as máquinas e as análises aos lubrificantes do cliente, deliberou-se desenvolver a página que exibe os metais e os valores de cada um de uma determinada análise. Mais uma vez que o procedimento foi o mesmo, ir buscar os dados à base de dados, metê-los num *array* para exibi-los na página. Portanto deve-se implementar, novamente em *javascript* o redireccionamento para uma nova página tal como se fez para a página de listagem dos clientes. Assim, desenvolveu-se o código apresentado na figura abaixo.

```
<script>
    jQuery(document).ready(function($) {
        $('.buttons-sample').click(function() {
            window.location.href = '{{ url("/sample") }}/' + $(this).attr('data-id');
        });
        $('#table_data').DataTable();
        $(document).on('click', '#table_data tbody tr button', function() {
            $("#table_data tbody tr").html("");
            $("#table_data tbody tr").html($(this).closest("tr").html());
        });

        $('.buttons-equipment').click(function() {
            window.location.href = '{{ url("/equipment/") }}/' + $(this).attr('data-id');
        });

        $('#datas').DataTable();
        $(document).on('click', '#datas tbody tr button', function() {
            $("#datas tbody tr").html("");
            $("#datas tbody tr").html($(this).closest("tr").html());
        });
    });
</script>
```

Figura 38 - Código em javascript no qual além de estar implementado o datatable, está implementado um a ocorrência de um evento após um clique de um elemento da tabela.

A linha 11 da imagem acima faz com que ocorra um evento ao clicar-se no elemento cujo classe dele é *buttons_sample* que neste caso será os elementos da tabela que mostram as análises (Ver Figura 37, linha 106) e que vai direcionar para uma nova página cuja rota é */sample/{id da análise}* pois o valor definido na variável *$(this).attr(data-id)* é *id_sample*. Já a linha 20 é semelhante com a diferença de que ao clicar num elemento da tabela que contem os equipamentos dos clientes, o utilizador é direcionado para uma nova página que cuja rota está definida como e*quipment/{id do equipamento}*, que é referente à página de detalhe das máquinas possuídas pelo cliente, página essa que será abordada mais à frente. Uma vez que a rota para a página das análises é */sample/{id da análise}*, então começou-se por definir essa rota no ficheiro *web.php* (ver figura abaixo).



```
Route::get('/sample/{id}', [App\Http\Controllers\SampleController::class, 'find'])->name('sampleId');
```

Figura 39 - Rota definida para a página que irá exibir os valores dos metais referente a uma determinada análise.

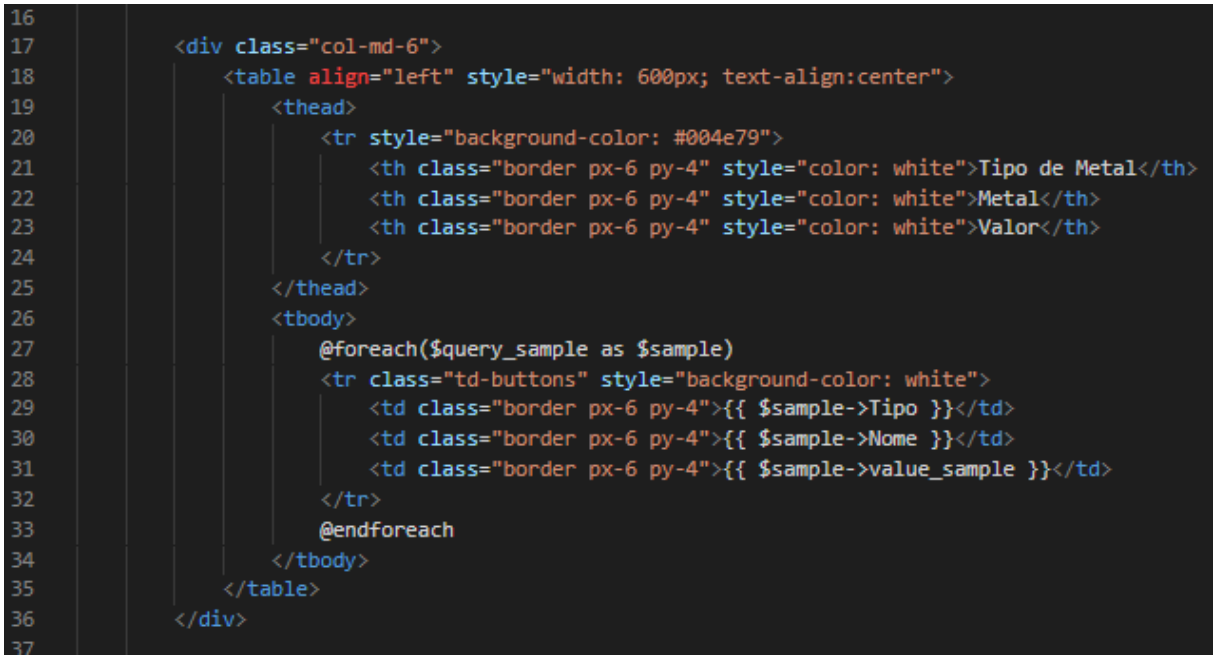
Como se pode ver pela imagem e seguindo o padrão descrito anteriormente, a função *find* que está no controlador *SampleController* é a função que irá recolher os metais e os seus respetivos valores da análise para depois apresentar na página. Em seguida é apresentada a imagem que mostra a função *find* do *SampleController*.

```php
public function find($id)
{
    $query_cust_equip = DB::table('sample_lubrificant')
        ->join('description_equipment', 'sample_lubrificant.id_description_equipment', '=', 'description_equipment.id_description_equipment')
        ->join('equipment', 'description_equipment.id_equipment', '=', 'equipment.id_equipment')
        ->join('customer', 'equipment.id_customer', '=', 'customer.id_customer')
        ->select('customer.nome as Cliente', 'equipment.nome as Equipamento', 'sample_lubrificant.id_sample', 'description_equipment.id_description_equipment')
        ->where('sample_lubrificant.id_sample', $id)->get()->first();

    $query_sample = DB::table('characteristics')
        ->join('characteristics_type', 'characteristics.id_characteristics_type', '=', 'characteristics_type.id_characteristics_type')
        ->join('details_analyzes', 'characteristics.id_characteristics', '=', 'details_analyzes.id_characteristics')
        ->join('sample_lubrificant', 'details_analyzes.id_sample', '=', 'sample_lubrificant.id_sample')
        ->select('sample_lubrificant.id_sample', 'characteristics_type.nome as Tipo', 'characteristics.nome as Nome', 'details_analyzes.value_sample')
        ->where('sample_lubrificant.id_sample', $id)->get();

    $data = [
        'id' => $id,
        'query_sample' => $query_sample,
        'query_cust_equip' => $query_cust_equip
    ];

    return view('analyzes/sample', $data);
}
```

Figura 40 - Função que recolhe os metais e os seus respetivos valores.

Esta função irá passar duas variáveis: uma com alguma informação relevante sobre a análise (*$query_cust_equip*) e outra com os metais e os seus respetivos valores (*$query_sample*). O fundamental a exibir na página são os tipos de metais, os metais e os seus respetivos valores e estes dados estão nas tabelas *characteristics* que contem os metais, *characteristics_type* que contem os tipos de metais e *details_analyzes* que é onde estão os valores lidos pelos sensores presentes nas máquinas. Nesse sentido criou-se a variável *$query_sample* (Figura 40), sendo ainda necessário fazer *join* com a tabela *sample_lubrificant* para identificar a análise que o utilizador que analisar.

Por fim, desenvolveu-se o código html criar a página e exibir os dados relativamente aos metais. A ideia é apresentar os dados numa tabela com três colunas: uma para o tipo de metal, uma para o metal e outra com o valor desse metal. Assim o código foi desenvolvido da seguinte forma.

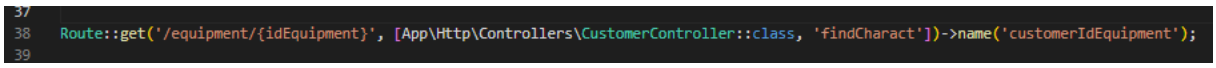


```
<div class="col-md-6">
    <table align="left" style="width: 600px; text-align:center">
        <thead>
            <tr style="background-color: #004e79">
                <th class="border px-6 py-4" style="color: white">Tipo de Metal</th>
                <th class="border px-6 py-4" style="color: white">Metal</th>
                <th class="border px-6 py-4" style="color: white">Valor</th>
            </tr>
        </thead>
        <tbody>
            @foreach($query_sample as $sample)
            <tr class="td-buttons" style="background-color: white">
                <td class="border px-6 py-4">{{ $sample->Tipo }}</td>
                <td class="border px-6 py-4">{{ $sample->Nome }}</td>
                <td class="border px-6 py-4">{{ $sample->value_sample }}</td>
            </tr>
            @endforeach
        </tbody>
    </table>
</div>
```

Figura 41 - Código em html que irá permitir exibir os valores dos metais numa tabela na página web.

Como se pode ver, foi implementado de forma muito semelhante às das outras páginas com listagens de elementos.

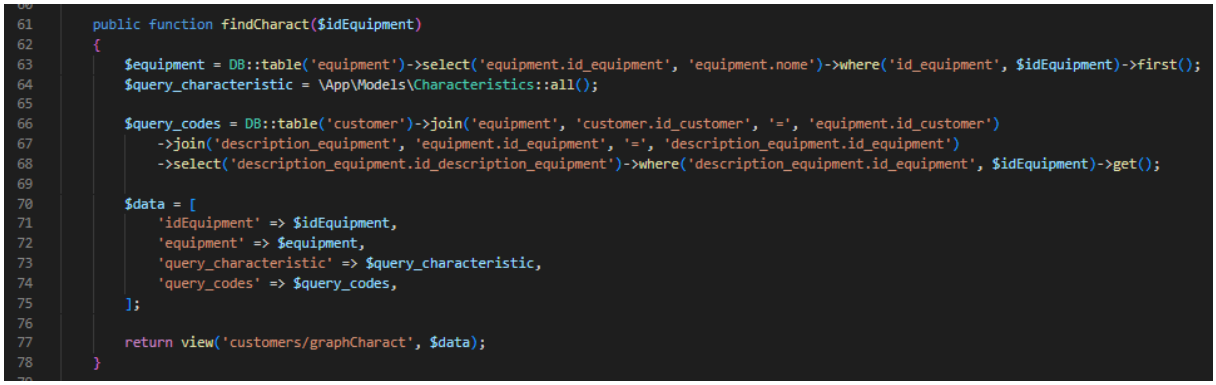
Por último, desenvolveu-se uma página que apresenta a evolução do metal. A ideia é que, na tabela onde é apresentada as máquinas do cliente, o utilizador possa clicar num elemento da tabela, seja direcionado para outra página e possa verificar a evolução de um determinado metal de uma máquina ao longo do tempo. Como foi indicado anteriormente em *javascript*, a rota a definir para a página a desenvolver é e*quipment/{id do equipamento}*, logo no ficheiro *web.php* definiu-se da seguinte forma. De referir que a identificação da máquina usada na rota é o *id* do equipamento e não o código. Optou-se por esta forma uma vez que os códigos estão identificados da seguinte maneira: *XXXXXXXX/YYYYY*. Como se pode ver o código apresenta uma barra oblíqua que é um caracter usado em rotas e, caso se usasse o código em rotas o utilizador seria direcionado para uma página que não existe.



```
Route::get('/equipment/{idEquipment}', [App\Http\Controllers\CustomerController::class, 'findCharact'])->name('customerIdEquipment');
```

Figura 42 - Definição da rota para a página html que exibirá a evolução de um metal de um determinado equipamento.

Seguindo o mesmo processo, pode-se constatar que se criou a função *findCharact* no *CustomerController* que irá devolver a página. A função foi criada da seguinte forma.



```php
public function findCharact($idEquipment)
{
    $equipment = DB::table('equipment')->select('equipment.id_equipment', 'equipment.nome')->where('id_equipment', $idEquipment)->first();
    $query_characteristic = \App\Models\Characteristics::all();

    $query_codes = DB::table('customer')->join('equipment', 'customer.id_customer', '=', 'equipment.id_customer')
        ->join('description_equipment', 'equipment.id_equipment', '=', 'description_equipment.id_equipment')
        ->select('description_equipment.id_description_equipment')->where('description_equipment.id_equipment', $idEquipment)->get();

    $data = [
        'idEquipment' => $idEquipment,
        'equipment' => $equipment,
        'query_characteristic' => $query_characteristic,
        'query_codes' => $query_codes,
    ];

    return view('customers/graphCharact', $data);
}
```

Figura 43 - Função do controlador referente à página relativa à evolução do metal.

Para se apresentar o gráfico que permite ver a evolução do metal é necessário que o utilizador escolha qual a máquina escolhendo o código e qual o metal a analisar e fá-lo-á através da página. Deste modo, criaram-se as variáveis a *$query_characteristics* que irá recolher todos os metais existentes e a *$query_codes* que irá recolher os códigos das máquinas relativas à identificação do equipamento que serão usadas num *dropdown* para que o utilizador possa fazer a sua escolha. As variáveis são, então, passadas para uma *view* do ficheiro *graphCharact.blade.php* como se pode observar na imagem abaixo.

```php
<div class="row" style=" padding-top: 50px">
    <table align="center" style="width: 100%; text-align:center">
        <thead>
            <tr>
                <th>Código da Máquina</th>
                <th>Caracteristica</th>
                <th></th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td><select name="code" id="code">
                        @foreach($query_codes as $characteristic)
                        <option value="{{ $characteristic->id_description_equipment }}">{{ $characteristic->id_description_equipment }}</option>
                        @endforeach
                    </select></td>
                <td><select name="charact" id="charact">
                        @foreach($query_characteristic as $characteristic)
                        <option value="{{ $characteristic->nome }}">{{ $characteristic->nome }}</option>
                        @endforeach
                    </select></td>
                <td><button id="new_button" type="submit" class="btn btn-primary">Analisar</button></td>
            </tr>
        </tbody>
    </table>
</div>
```

Figura 44 - Código referente ao dropdown dos códigos de cada máquina e outro dropdown dos metais.

Para se criar um *dropdown* deve-se usar a tag <*select*> que é usada em html para exibir uma lista de elementos para que o utilizador possa selecionar um deles. Para listar todos os elementos deve-se usar a tag <*options*> que é responsável por armazenar o valor correspondente do item no qual deve ser identificado no atributo *value* (como se pode ver nas linhas 52 e 57) e também o texto para uma melhor compreensão para o utilizador. Assim, escreveu-se o script em html que está exibido na Figura acima. Da linha 50 à linha 54 é apresentado o

dropdown com os códigos referente à identificação do equipamento enquanto da linha 55 à 59 é apresentado o *dropdown* dos metais. Foi acrescentado um botão na linha 60 para poder enviar os dois elementos da *dropdown* para o controlador de modo a obter os valores para apresentar no gráfico. Nota ainda para a identificação dos *dropdowns* nas linhas 50 e 55 que será útil no envio dos dois elementos para a base de dados. Após a escolha por parte do utilizador, este clica no botão *Analisar* e o seguinte código em *javascript* é executado.

```
79
80          $("#new_button").click(function() {
81              var code = $("#code").val();
82              var charact = $("#charact").val();
83
84              $.ajax({
85                  type: "GET",
86                  url: "/equipment/code/charact",
87                  data: {
88                      code: code,
89                      charact: charact,
90                  },
```

Figura 45 - Código que é executado após o clique no botão Analisar e que permite ao utilizador enviar um pedido para a base de dados.

A função deste código é que após o clique no botão *Analisar*, o item que está na *dropdown* dos códigos das máquinas é guardada na variável *code*, já o item que está na *dropdown* dos metais é guardada na variável *charact*.

Utilizando o método Ajax, é possível enviar variáveis para uma função criada no controlador e obter os valores do metal desejados. O *Ajax* (*Asynchronous JavaScript and XML*) é um método que permite o desenvolvimento de aplicações web mais interativas, permitindo a troca de dados com o servidor sem a necessidade de recarregar a página. O *Ajax* é programado em JavaScript e amplamente utilizado na construção de interfaces web dinâmicas. E é justamente uma troca de dados que acontece, pois, o utilizador irá enviar ao servidor um pedido com o código da máquina e o metal e receberá os valores que serão exibidos na página. A imagem seguinte mostra como se obtém os valores a exibir no gráfico.

```
80      public function graph(Request $request)
81      {
82          $codigo = $request->code;
83          $metal = $request->charact;
84
85          $query_graph_test = DB::table('sample_lubrificant')->join('description_equipment', 'sample_lubrificant.id_description_equipment', '=', 'description_equipment.id_description_equipment')
86              ->join('equipment', 'description_equipment.id_equipment', '=', 'equipment.id_equipment')
87              ->join('customer', 'equipment.id_customer', '=', 'customer.id_customer')
88              ->join('details_analyzes', 'sample_lubrificant.id_sample', '=', 'details_analyzes.id_sample')
89              ->join('characteristics', 'details_analyzes.id_characteristics', '=', 'characteristics.id_characteristics')
90              ->select(array('sample_lubrificant.date_completed as 0', 'details_analyzes.value_sample as 1'))
91              ->where('description_equipment.id_description_equipment', $codigo)->where('characteristics.nome', $metal)->get()->toArray();
92
93          for ($i = 0; $i < count($query_graph_test); $i++) {
94              $query_graph_test[$i] = (array) ($query_graph_test[$i]);
95              $query_graph_test[$i][0] = strtotime($query_graph_test[$i][0]) * 1000;
96          }
97
98          return $query_graph_test;
99      }
```

Figura 46 - Função que recolhe os valores de um metal para depois apresentar na página.

As linhas 82 e 83 guardam o código da máquina e o metal escolhidos pelo utilizador através dos *dropdowns* nas variáveis *$codigo* e *$metal* respetivamente. Com essas variáveis desenvolvemos uma query que permite obter todos os valores lidos do metal até então e esses valores são armazenados na variável *$query_graph_test* (linhas 85 à 91) e esses valores contêm a data da análise e o valor do metal. Uma vez que se usou o *Highchart* no desenvolvimento do gráfico (será abordado mais à frente) significa que a data deve estar no formato *timestamp*. Uma vez que a data não está nesse formato, então usou-se o ciclo *for* da linha 93 à 96 para percorrer a variável *$query_graph_test* e ir convertendo a data para o formato correto. Por fim, devolve-se os dados para exibi-los no gráfico.

Como foi dito, usou-se o *Highchart* para desenvolver o gráfico. O *Highchart* é uma biblioteca para desenvolvimento de gráficos e é totalmente programado em *javascript* podendo desenvolver diversos gráficos tais como gráficos de barras, de radar ou em linhas sem. Uma vez que se pretende visualizar a evolução do metal, faz sentido usar um gráfico de linhas. Fazer o download de dados é também um requisito útil na medida em que se pretende apresentar argumentos para corroborar de que alguma ação é necessária. Isto porque nunca é viável tomar certas ações sem apresentar factos que levam a custos adicionais desnecessários. Como já foi referido, quando existe metais no fluido e com valores bastante elevados, torna-se necessário fazer a manutenção da máquina.

Uma particularidade interessante nos *Highchart* é que permite fazer *download* do gráfico nos formatos PDF, PNG, JPEG e SVG, também permite exportar os dados para ficheiros do tipo excel além de se poder imprimir o gráfico, portanto o *Highchart* é uma ferramenta muito útil e vantajosa para aplicações web. Para desenvolver o gráfico no código escreveu-se o script que está apresentado nas duas imagens abaixo.

```
83
84              $.ajax({
85                  type: "GET",
86                  url: "/equipment/code/charact",
87                  data: {
88                      code: code,
89                      charact: charact,
90                  },
91                  success: function(res) {
92
93                      Highcharts.chart("container", {
94
95                          chart: {
96                              zoomType: 'x'
97                          },
98                          title: {
99                              text: 'Evolução de ' + charact
100                         },
101                         subtitle: {
102                             text: document.ontouchstart === undefined ?
103                                 'Click and drag in the plot area to zoom in' : 'Pinch the chart to zoom in'
104                         },
105                         xAxis: {
106                             type: 'datetime'
107                         },
108                         yAxis: {
109                             title: {
110                                 text: 'Valor em ppm'
111                             }
112                         },
113                         legend: {
114                             enabled: false
115                         },
```

Figura 47 - Pedido dos valores ao servidor web que em caso de sucesso, este devolve-os para apresentá-los no gráfico e também parte do código do Highchart.

```
    plotOptions: {
        area: {
            fillColor: {
                linearGradient: {
                    x1: 0,
                    y1: 0,
                    x2: 0,
                    y2: 1
                },
                stops: [
                    [0, Highcharts.getOptions().colors[0]],
                    [1, Highcharts.color(Highcharts.getOptions().colors[0]).setOpacity(0).get('rgba')]
                ]
            },
            marker: {
                radius: 2
            },
            lineWidth: 1,
            states: {
                hover: {
                    lineWidth: 1
                }
            },
            threshold: null
        }
    },
    series: [{
        name: 'Valor',
        data: res
    }]
});
```

Figura 48 - Outra parte do código do Highchart.

Se o pedido que o utilizador fez ao servidor for feito com sucesso o servidor responde com os valores que por sua vez são guardados na variável *res* (linha 91). De seguida, começa-se a correr o script relativo ao *highchart*. Nas figuras acima são apresentadas as configurações do gráfico. A primeira (*zoomType*) serve para que o utilizador possa ampliar o zoom ao longo do eixo x arrastando o rato, isso é, o utilizador, ao aumentar o zoom, o gráfico irá amplificar as datas pois estas pertencem ao eixo x o que acaba por ser bastante útil se, por exemplo o utilizador desejar analisar a evolução do metal de um determinado intervalo de tempo. A segunda (*title*) e terceira (*subtitle*) configuração apresentada é o título e subtítulo do gráfico. A quarta configuração (*xAxis*) serve para definir o eixo x, neste caso foi para definir o formato dos valores desse eixo. A configuração seguinte (*yAxis*) é parecida com a descrita do parágrafo anterior só que desta vez é para definir o eixo y. Neste caso definiu-se um texto que indica a unidade dos valores daquele eixo.

A seguir apresenta-se uma configuração para adicionar uma legenda ao gráfico, mas decidiu-se desativar uma vez que não fazia sentido. Depois é apresentado a configuração de parâmetro *plotOptions* que serve para definir os pontos dos dados do gráfico, como por exemplo se se deseja exibir os dados quando se passa o rato pelo ponto de um determinado dado.

Por fim, cabe destacar o parâmetro de configuração *series*, o qual inclui o conjunto de dados recuperados da base de dados e que será utilizado para exibi-los no gráfico. Esse conjunto é passado como parâmetro da opção *data*.

3.3 Publicação do Projeto

Após a finalização do desenvolvimento do projeto localmente, fez-se a instalação do projeto num painel de controlo para publicar *websites*. A interface gráfica de eleição foi o *cPanel*, que é disponibilizada pelo fornecedor do alojamento para a plataforma.

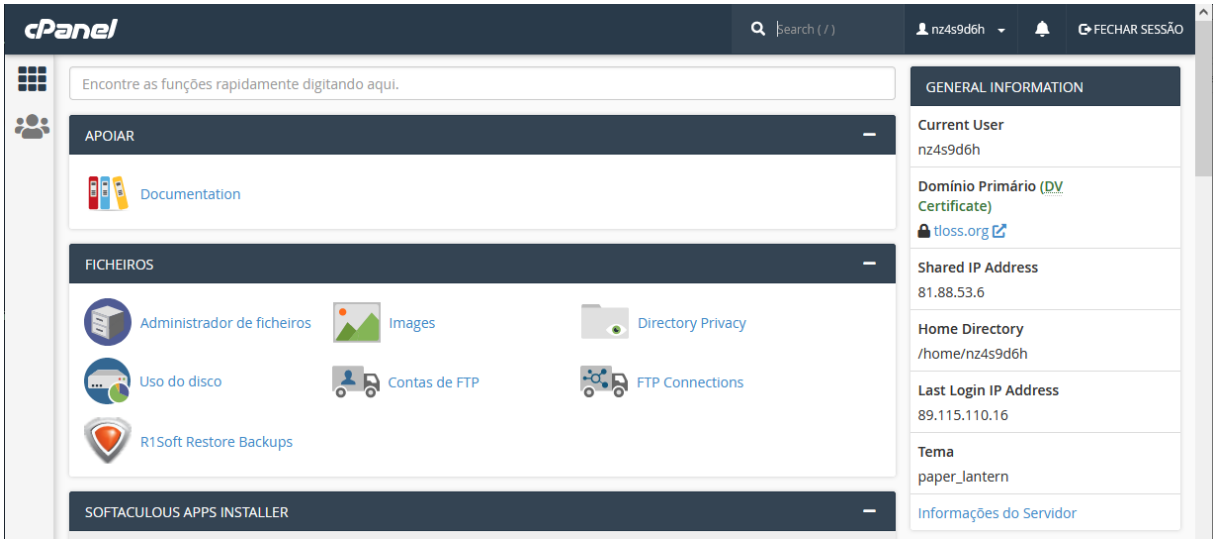


Figura 49 - Página inicial da interface cPanel.

Primeiramente começou-se por criar a base de dados e configurá-la criando um utilizador, uma palavra-passe um *host* e uma porta para aceder à base de dados. Deve-se então ir à secção MySQL *Databases* para definir os nomes para a base de dados, o utilizador e a senha para acedê-la. As duas imagens abaixo mostram essa secção.

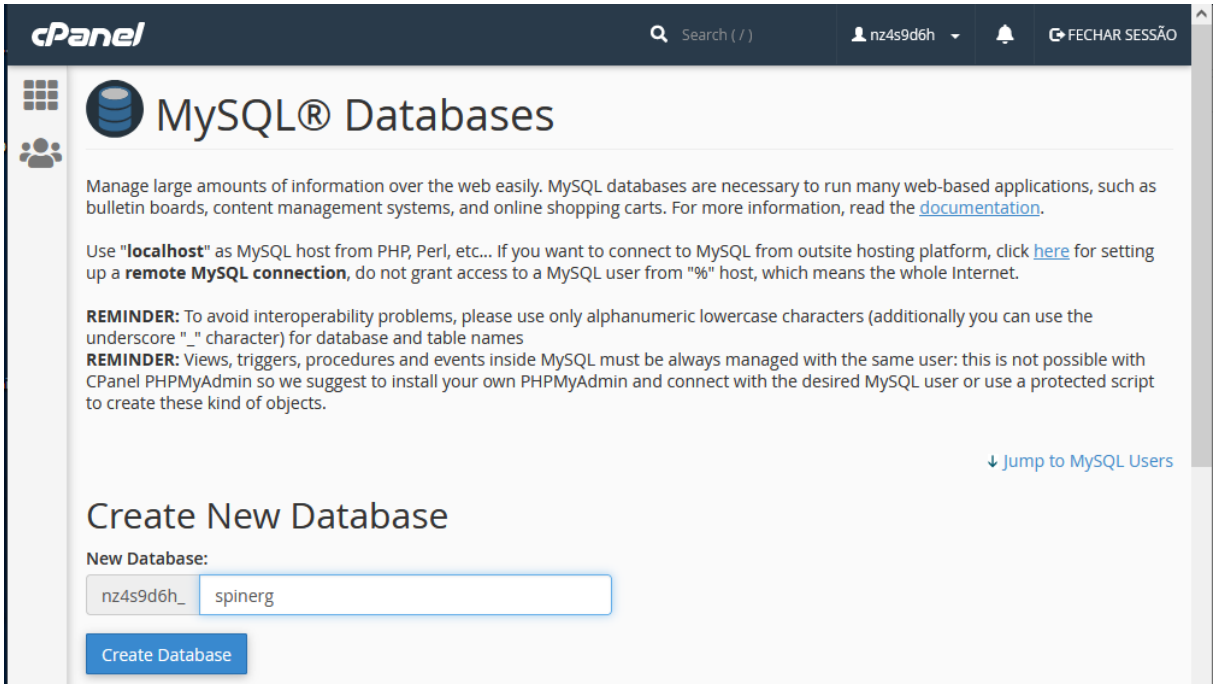


Figura 50 - Página da interface que permitirá criar a base de dados.

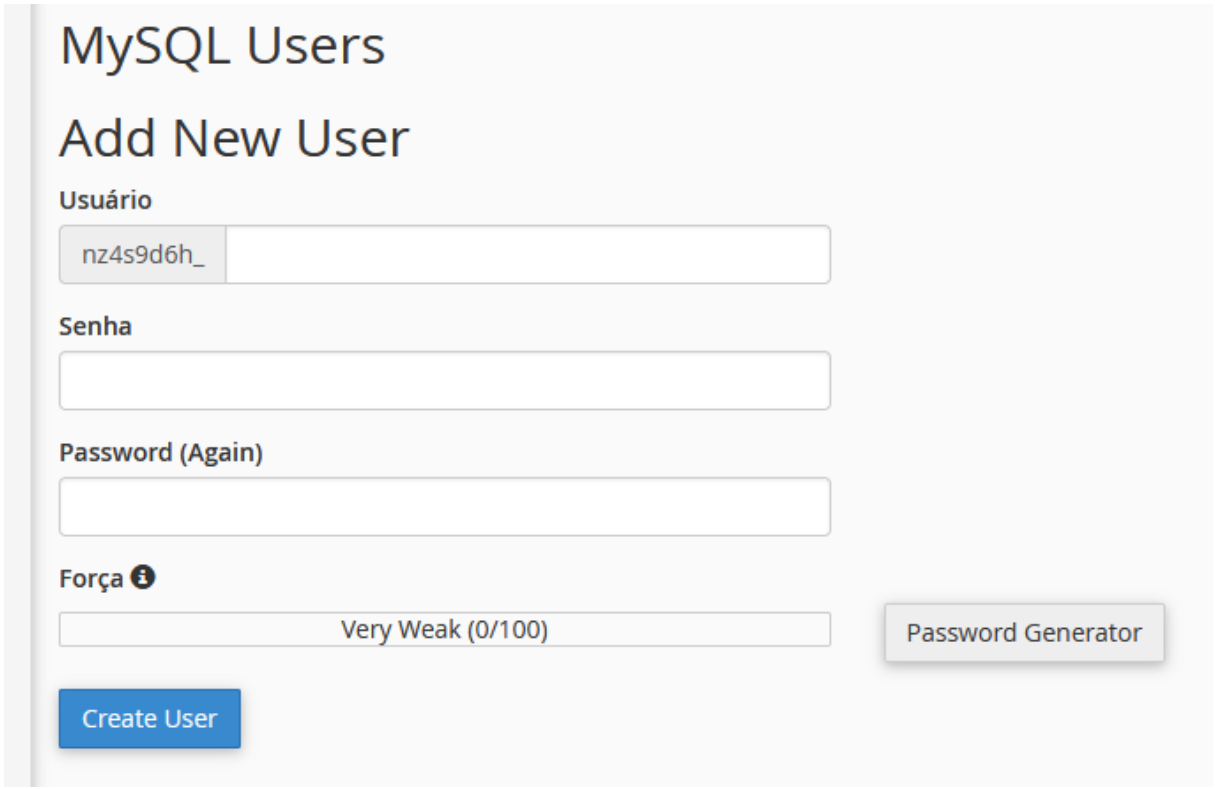


Figura 51 - Configuração da base de dados criando um utilizador e uma palavra-passe.

Para o nome da base de dados escolheu-se *nz4s9d6h_spinerg* assim como também para o nome do utilizador (*nz4s9d6h* é adicionado nos campos por defeito uma vez que esse é o nome utilizador da conta do *cPanel*).

Figura 52 - Adição de um utilizador à base de dados *nz4s9d6h_spinerg*.

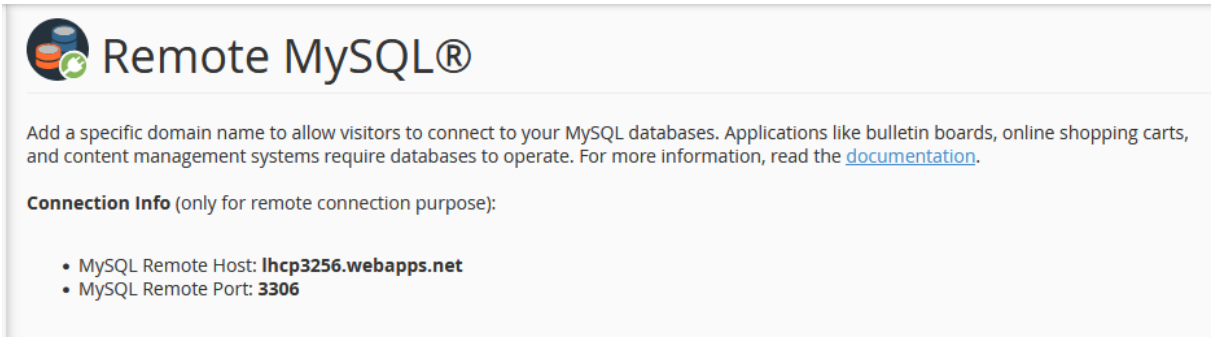


Figura 53 - Host e porta definidas para aceder à base de dados.

Após a criação da base de dados e adicionar um utilizador à base de dados, deu-se a este todos os privilégios para manusear a base de dados como mostra a figura abaixo.

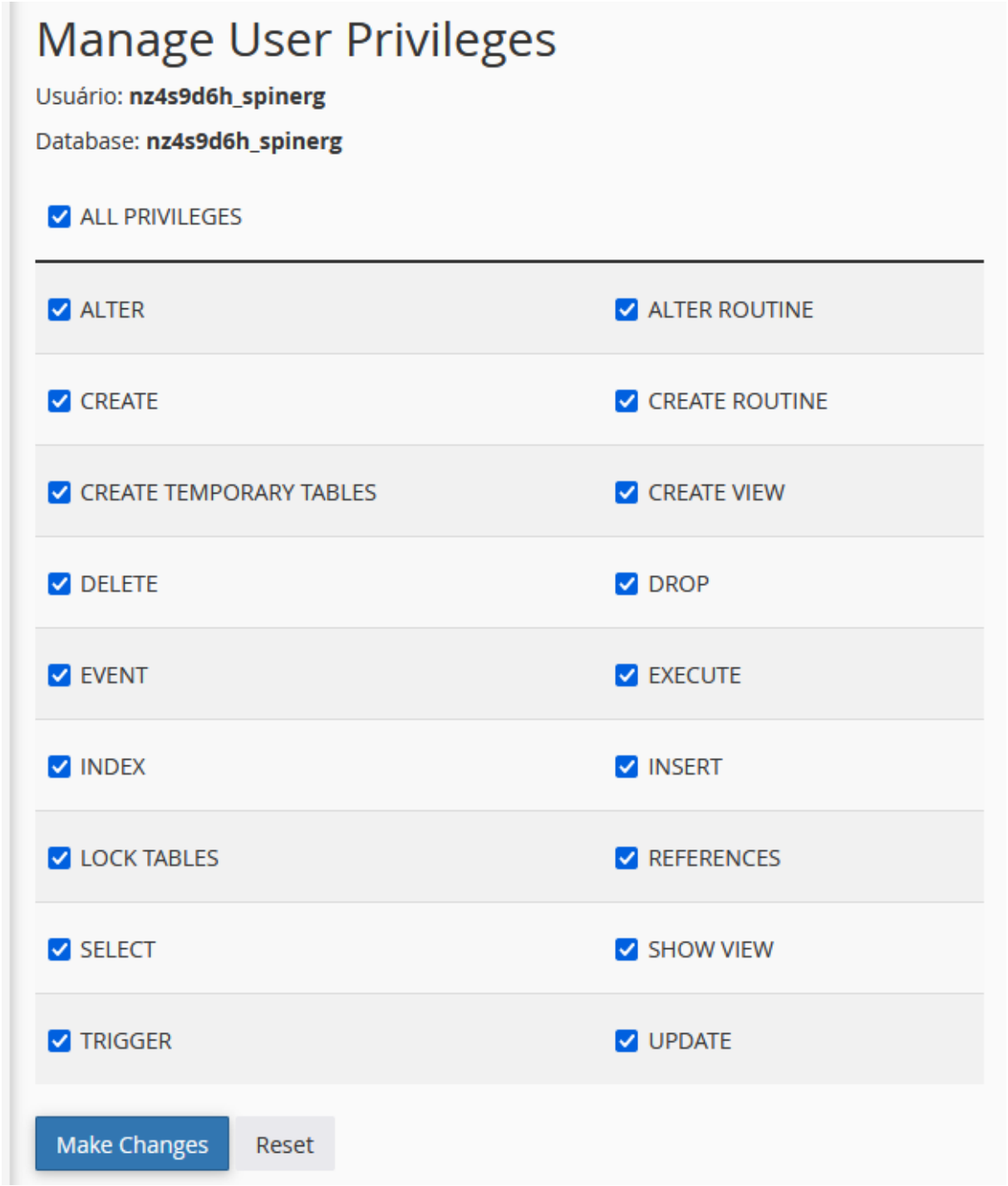


Figura 54 - Privilégios que permitem ao utilizador usar a base de dados.

Por fim foi necessário fornecer IP público que permite ao utilizador editar a base de dados.

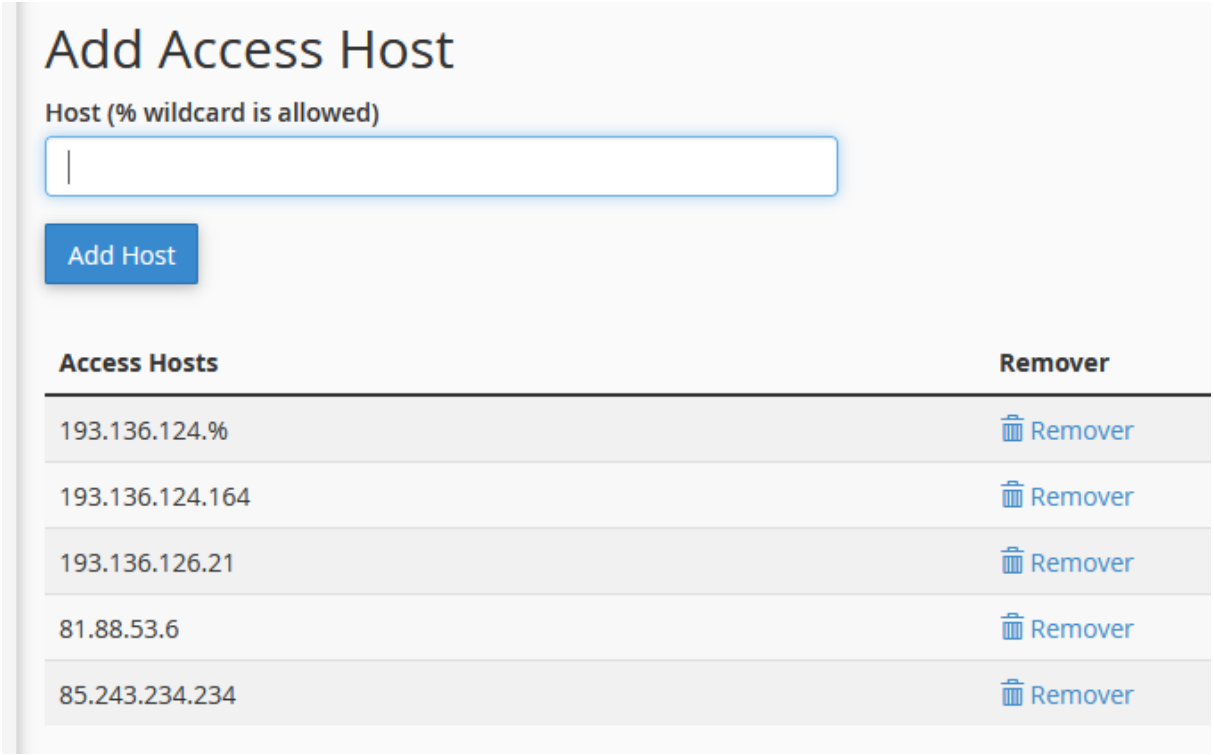


Figura 55 - IPs públicos com acesso à base de dados.

Após a criação da base de dados, as tabelas foram adicionadas utilizando o mySQL Workbench, uma ferramenta que possibilita a geração automática de código para criar tabelas a partir de um diagrama. Dessa forma, foi possível integrar as tabelas da base de dados local à nova base de dados.

De facto, já se tinha o diagrama, mas não com as tabelas que são usadas para a autenticação do utilizador (tabelas *users, failed_jobs, migrations, sessions, personal_access_tokens e password_resets*) pois são tabelas criadas pelo laravel por defeito. Ainda assim foi possível criar um diagrama com a base de dados local. Para isso deve-se ir a *Database -> Reverse Engineer* como mostra a figura seguinte.

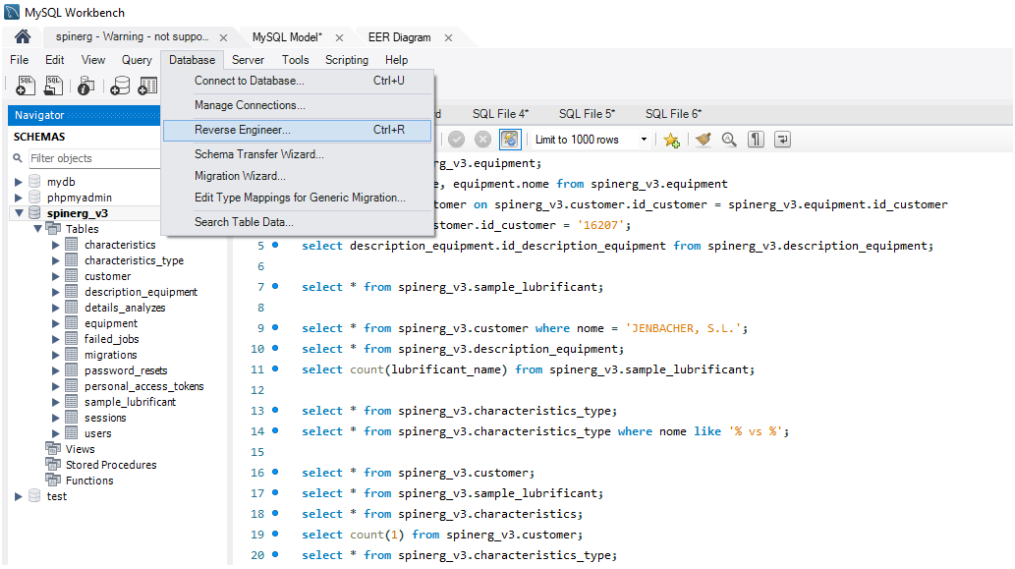


Figura 56 - Opção para gerar o diagrama da base de dados local que se usou para o desenvolvimento do projeto.

Após alguns procedimentos necessários deve-se escolher a base de dados.

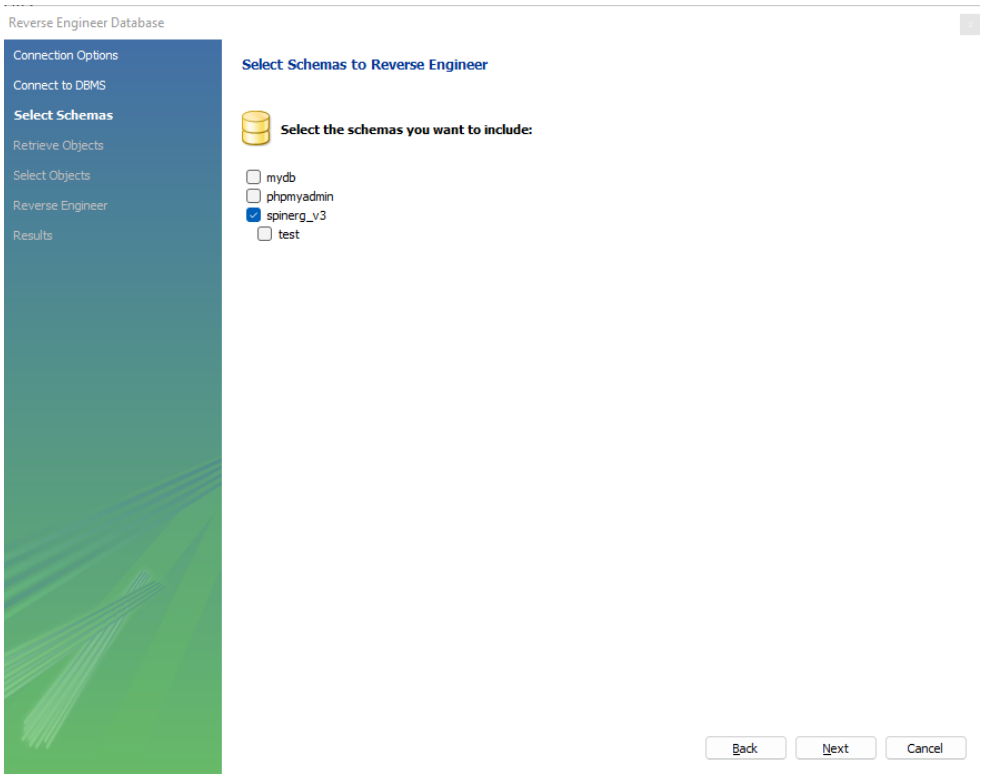


Figura 57 - Base de dados selecionada para gerar o diagrama.

Após mais uns passos, é gerado o diagrama.

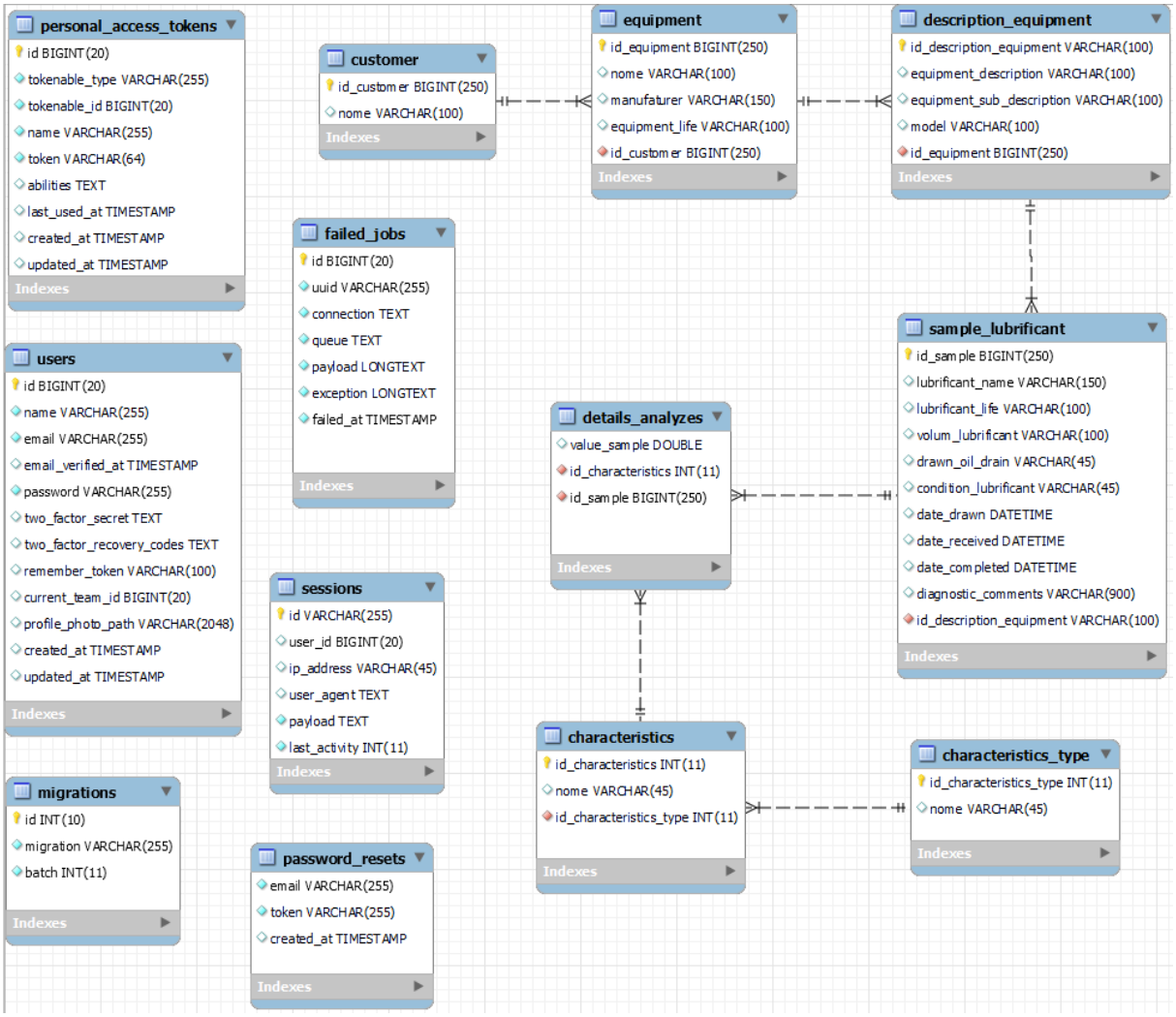


Figura 58 - Diagrama gerado no qual será usado para gerar um código em SQL para criar as tabelas na base de dados *nz4s9d6h_spinerg*.

Para gerar o código é ir ao menu *Database* e selecionar ao submenu *Forward Engineer*. Após uma série de procedimentos, é gerado um código SQL como aparece na imagem abaixo apresentada.

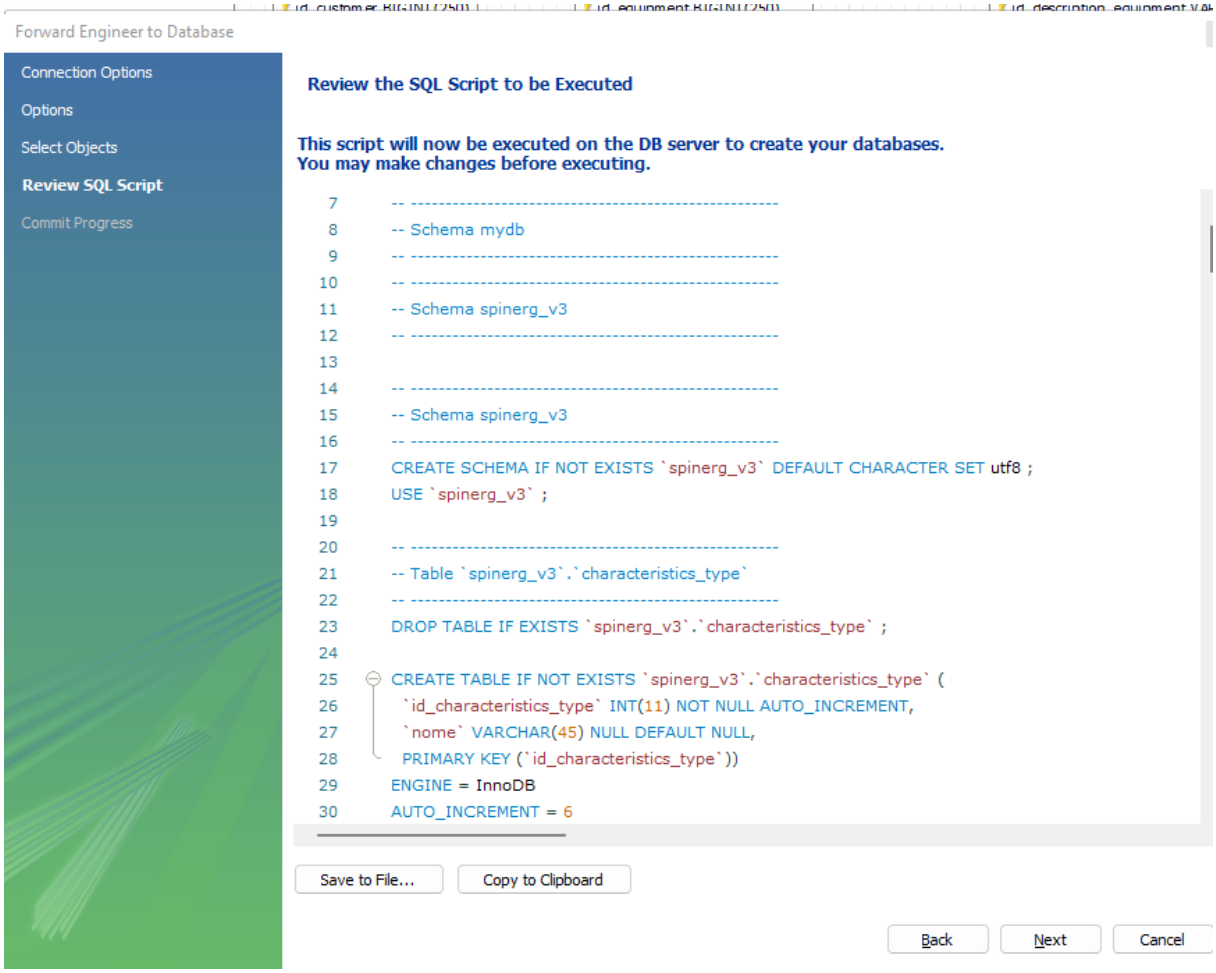


Figura 59 - Código sql gerado para executar na base de dados *nz4s9d6h_spinerg*.

Como se pode visualizar, a base de dados usada é *spinerg_v3*. Tem este nome pois foi dado à base de dados local para se testar o projeto à medida que foi desenvolvido. Portanto, onde está *spinerg_v3*, deve-se substituir por *nz4s9d6h_spinerg* para que as tabelas sejam criadas nesta base de dados.

Finalmente, adicionaram-se os ficheiros referentes ao projeto no *ftp account*. Os ficheiros da aplicação web ficam na pasta determinada pelo utilizador.

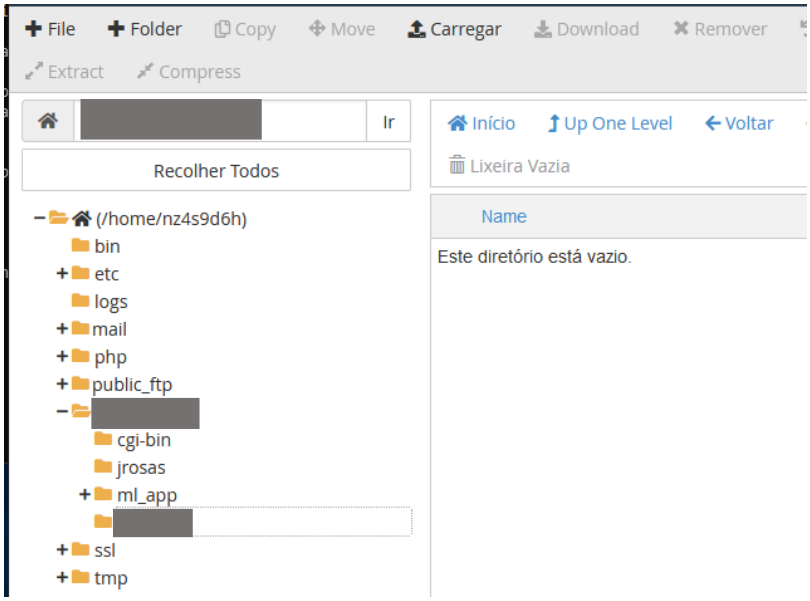


Figura 60 - Pasta do ftp onde irão ficar os ficheiros da aplicação web.

De seguida adicionou-se uma conta ftp como mostra a figura seguinte.

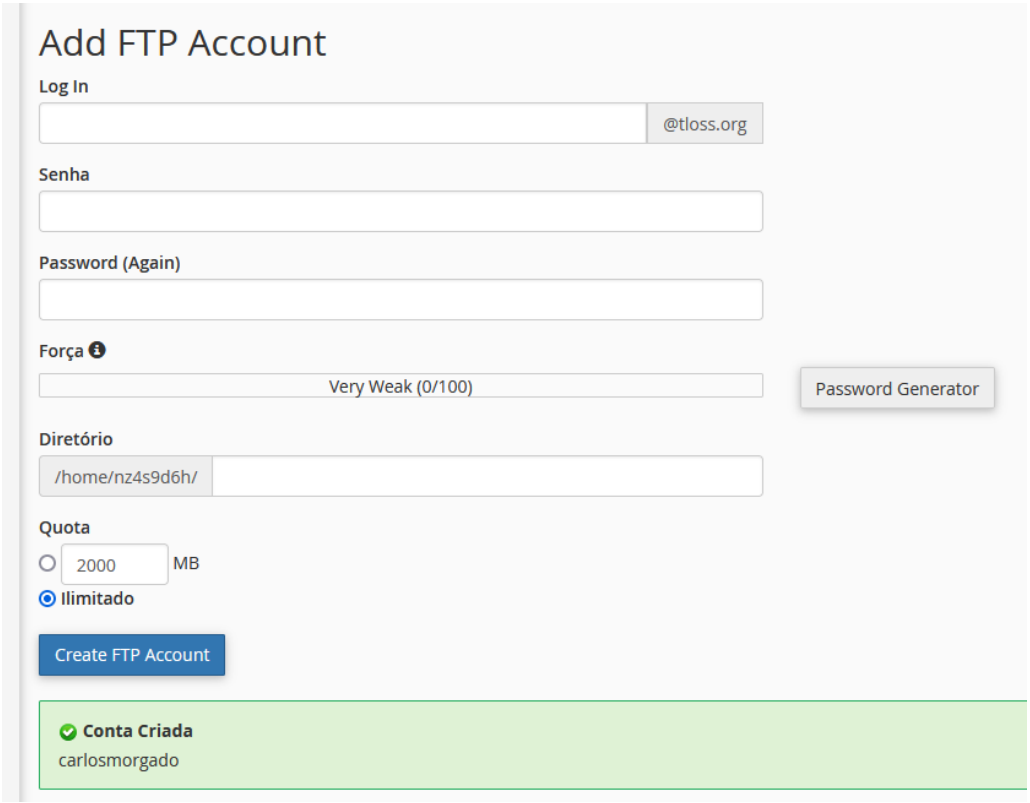


Figura 61 - Campos necessários para criar uma conta ftp.

Para adicionar a conta definiu-se um site (campo Log In) como tloss.org, criou-se uma palavra-passe para aceder ao ftp de modo a fazer o *upload* dos ficheiros da aplicação web e, por fim, definir uma diretoria. Para aceder ao ftp instalou-se uma ferramenta denominada *WinSCP* que é um *ftp client* que permite transferir ficheiros de forma segura. Inicialmente deve-se preencher os campos *Nome de Anfitrião* com o que foi definido no campo *Log In* (conforme mostrado na referência à figura), e preencher o nome do utilizador e a respetiva *password*.

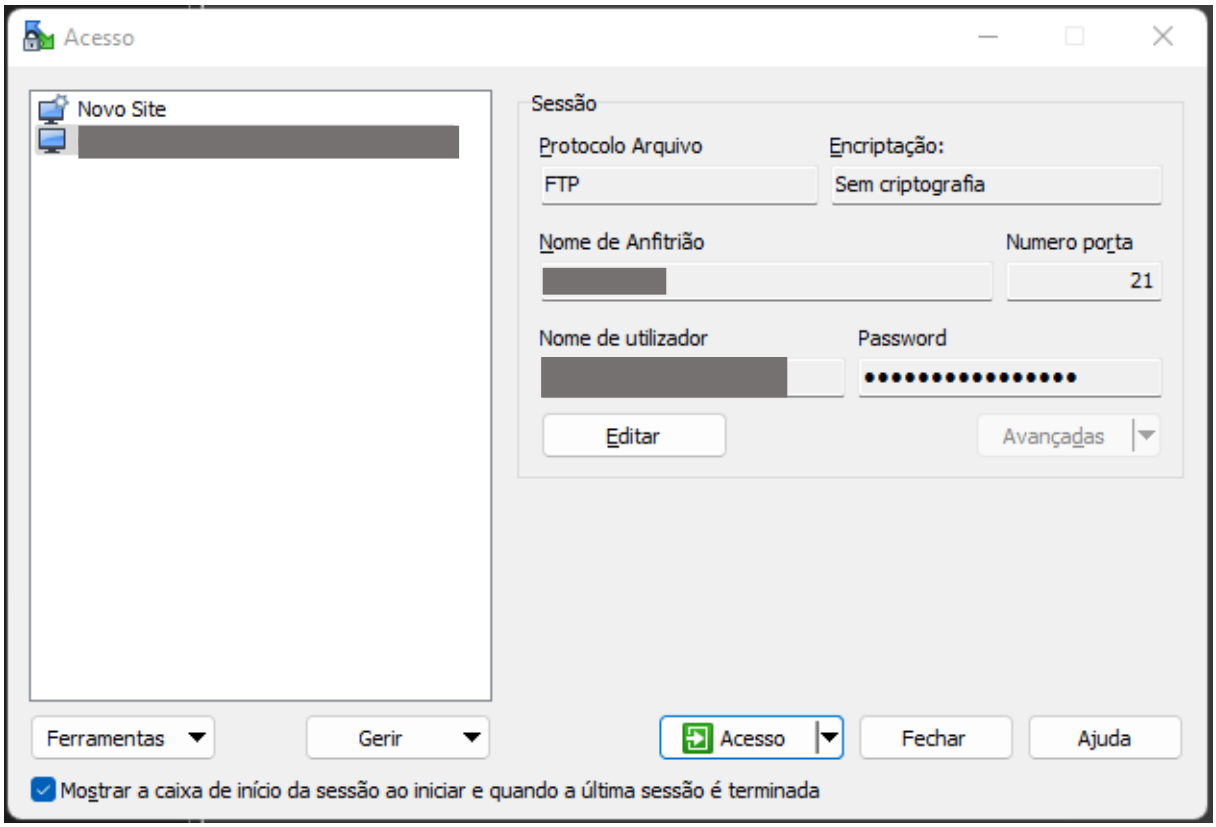


Figura 62 - Inicio da sessão do ftp da ferramenta WinSCP.

Após aceder ao ftp com sucesso pode-se, por fim, arrastar os ficheiros da aplicação para o ftp em que neste caso arrastou-se os ficheiros da esquerda para a direita.

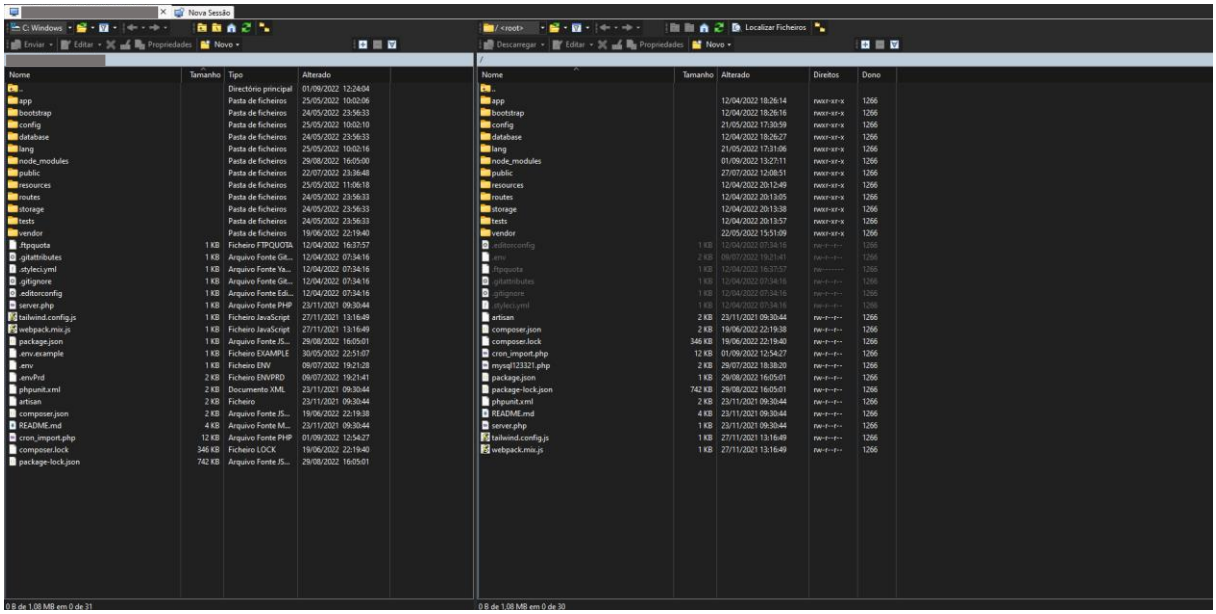


Figura 63 - Transferência de ficheiros da aplicação web para o ftp.

4

TESTES E RESULTADOS

Neste capítulo irá abordar-se os testes feitos ao longo do desenvolvimento do projeto assim como o resultado. Para correr a aplicação *web* é necessário executar o comando *php artisan serve* que é um comando que tem como propósito correr a aplicação no servidor de desenvolvimento. Ao iniciar a aplicação e acedendo ao link acima descrito o utilizador acede à página inicial que está apresentada na imagem abaixo.

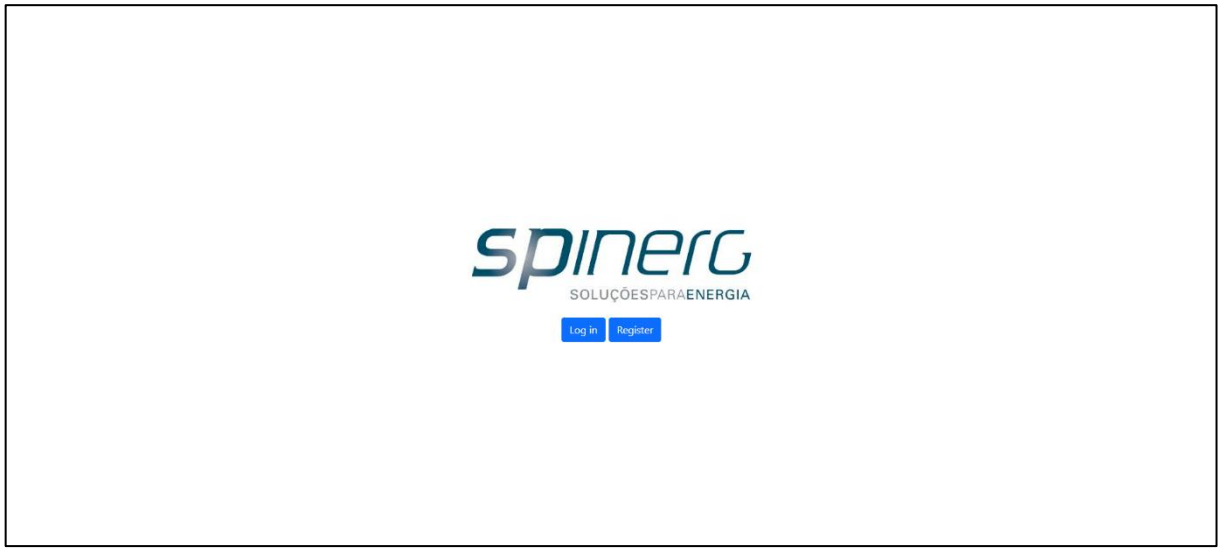


Figura 64 - Página inicial da plataforma.

A página apresenta a imagem da empresa para qual se desenvolveu a aplicação *web* e dois botões: um para a pessoa iniciar a sessão (Botão *Log In*) e outro para se registar caso ainda não tenha conta (Botão *Register*).

Figura 65 - Página do login que permitirá ao utilizador iniciar sessão (esta funcionalidade está desativada para efeitos de confidencialidade).

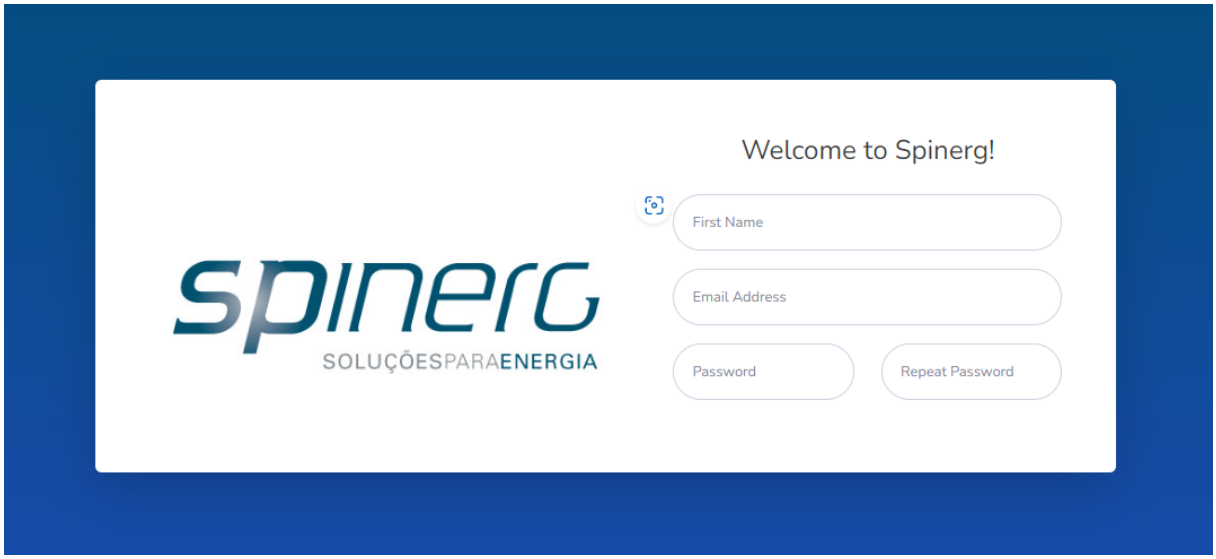


Figura 66 - Página que permitirá ao utilizador registar-se caso ainda não tenha conta (esta funcionalidade está desativada para efeitos de confidencialidade).

4.1 Importação dos dados

Nesta secção vai-se apresentar o procedimento da importação de dados. De referir que a importação é feita de duas formas: importação via plataforma ou importação via email como já foi abordado anteriormente. É importante destacar que os dados das análises foram

alterados assim como algumas imagens terem dados ocultados em virtude de um acordo de confidencialidade estabelecido pela empresa, a fim de preservar a privacidade das informações.

4.1.1 Importação Via Plataforma

Foi dito anteriormente que se desenvolveu uma página com um campo que permite selecionar um ficheiro para importar os dados. A página em questão é apresentada na imagem a seguir.

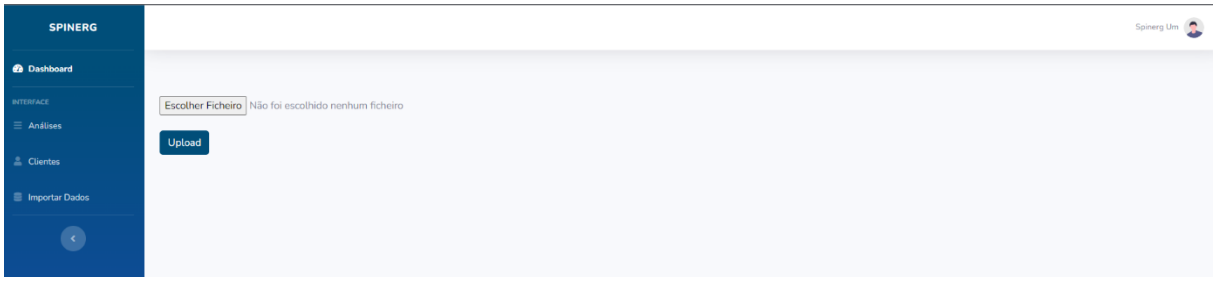


Figura 67 - Página que irá permitir ao utilizador selecionar um ficheiro excel para importar os dados.

Ao carregar no botão *Escolher Ficheiro* uma caixa de diálogo é aberta para que se possa escolher o ficheiro excel para importação dos dados. Após escolher o ficheiro com os dados que se quer importar, o nome deste é exibido na página como mostra a imagem seguinte.

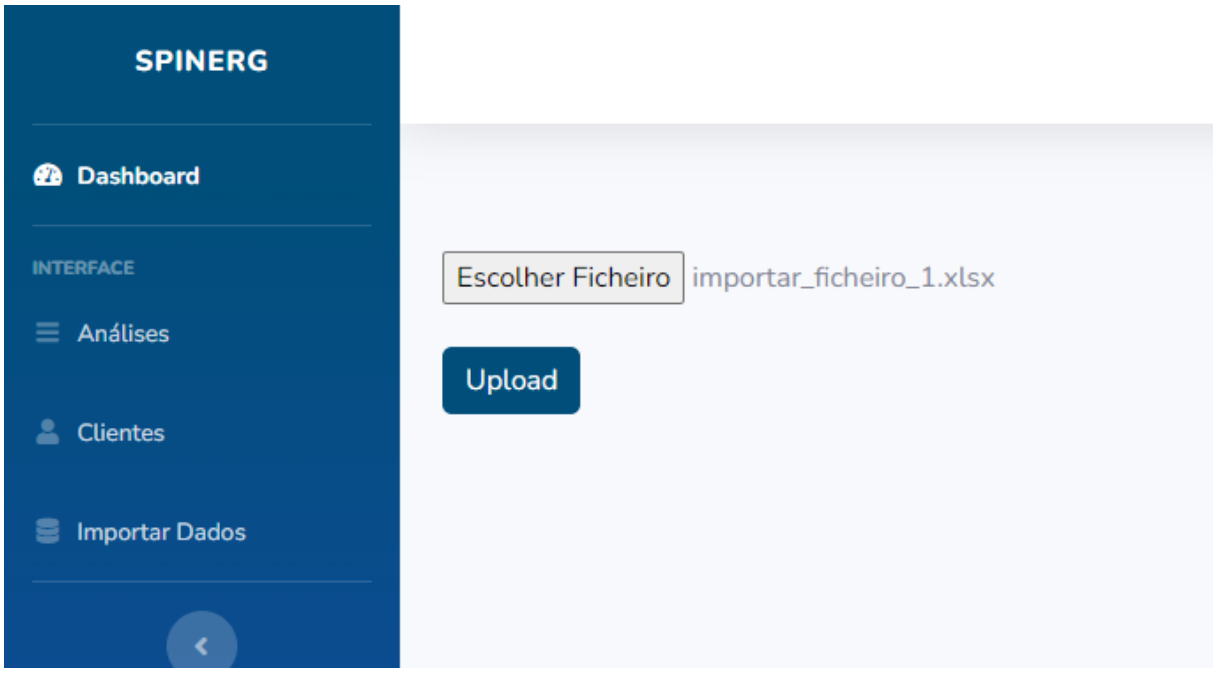


Figura 68 - Depois que o ficheiro excel é selecionado, o seu nome é exibido na página.

Por fim, clica-se no botão *Upload* para dar início à importação de dados e após o clique no botão, é apresentado um *spinner* de modo a indicar que a importação de dados está em processamento.

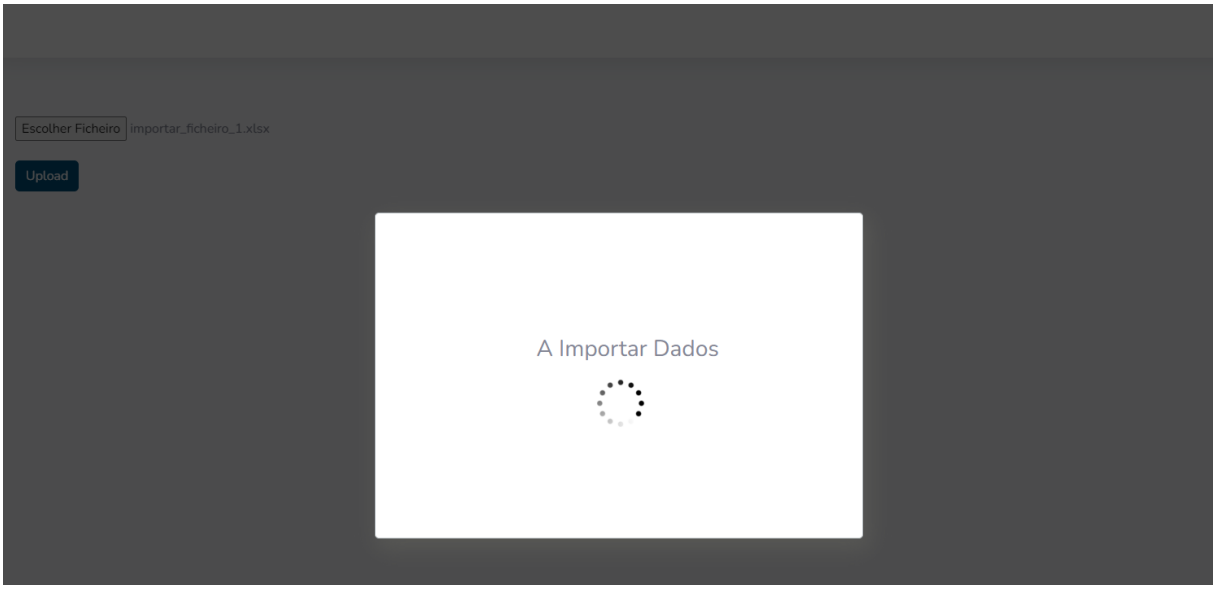


Figura 69 - Spinner que é exibido após o clique no botão *Upload*.

Quando a importação é finalizada, a página é recarregada podendo assim voltar a importar se o utilizador assim o desejar. Ver-se-á agora se a importação ocorreu com sucesso. Para isso vai ser apresentada uma imagem do excel com os valores dos metais de uma determinada análise.

25	**Metal Type 1**	
26	Metal 1.1	202
27	Metal 1.2	45
28	Metal 1.3	17
29	Metal 1.4	7.90
30	**Metal Type 2**	
31	Metal 2.1	13
32	Metal 2.2	17
33	Metal 2.3	19
34	Metal 2.4	20
35	Metal 2.5	0
36	Metal 2.6	0
37	Metal 2.7	0
38	Metal 2.8	9
39	Metal 2.9	9
40	Metal 2.10	9
41	Metal 2.11	4
42	**Metal Type 3**	
43	Metal 3.1	-
44	Metal 3.2	2.09
45	Metal 3.3	6
46	Metal 3.4	-
47	Metal 3.5	0
48	Metal 3.6	0
49	Metal 3.7	0
50	Metal 3.8	0
51	Metal 3.9	1
52	Metal 3.10	0
53	**Metal Type 4**	
54	Metal 4.1	2098
55	Metal 4.2	500
56	Metal 4.3	12
57	Metal 4.4	14
58	Metal 4.5	24
59	Metal 4.6	7
60	Metal 4.7	1
61	**Metal Type 5**	
62	Metal 5.1	17.9
63	Metal 5.2	0
64	Metal 5.3	-

Figura 70 - Valores dos metais relativo à análise 38.

ID	Tipo	Metal	Valor
38	Metal Type 1	Metal 1.1	202
38	Metal Type 1	Metal 1.2	45
38	Metal Type 1	Metal 1.3	17
38	Metal Type 1	Metal 1.4	7.9
38	Metal Type 2	Metal 2.1	13
38	Metal Type 2	Metal 2.2	17
38	Metal Type 2	Metal 2.3	19
38	Metal Type 2	Metal 2.4	20
38	Metal Type 2	Metal 2.5	0
38	Metal Type 2	Metal 2.6	0
38	Metal Type 2	Metal 2.7	0
38	Metal Type 2	Metal 2.8	9
38	Metal Type 2	Metal 2.9	9
38	Metal Type 2	Metal 2.10	9
38	Metal Type 2	Metal 2.11	4
38	Metal Type 3	Metal 3.1	NULL
38	Metal Type 3	Metal 3.2	2.09
38	Metal Type 3	Metal 3.3	6
38	Metal Type 3	Metal 3.4	NULL
38	Metal Type 3	Metal 3.5	0
38	Metal Type 3	Metal 3.6	0
38	Metal Type 3	Metal 3.7	0
38	Metal Type 3	Metal 3.8	0
38	Metal Type 3	Metal 3.9	1
38	Metal Type 3	Metal 3.10	0
38	Metal Type 4	Metal 4.1	2098
38	Metal Type 4	Metal 4.2	500
38	Metal Type 4	Metal 4.3	12
38	Metal Type 4	Metal 4.4	14
38	Metal Type 4	Metal 4.5	24
38	Metal Type 4	Metal 4.6	7
38	Metal Type 4	Metal 4.7	1
38	Metal Type 5	Metal 5.1	17.9
38	Metal Type 5	Metal 5.2	0
38	Metal Type 5	Metal 5.3	NULL

Figura 71 - Valores dos metais extraídos do ficheiro excel para a base de dados.

Na Figura 70 estão apresentados os valores lidos dos metais no ficheiro *Excel*. Na figura seguinte é apresentada o resultado de uma *query* que mostra os valores dos metais. Se se observar as duas imagens e comparar o resto dos metais e os seus valores pode-se perceber que os valores apresentados na base de dados correspondem aos valores apresentados no excel pelo que se pode concluir que a implementação que foi apresentada no capítulo 3 está correta.

4.1.2 Importação Via Email

Para a importação via email, a abordagem consiste em criar duas crons e que possam ser executadas de trinta em trinta segundos: uma para ler emails que contêm os ficheiros de excel para extrair o ficheiro excel e guardá-lo numa pasta do projeto e outra cron é para extrair os ficheiros de excel nesta pasta e dar início à importação dos dados eliminando-o da pasta após a conclusão da importação. Uma vez que localmente não é possível criar *crons*, usou-se

ficheiros com a extensão *bat*. Este tipo de ficheiros permite executar um conjunto de comandos sequencialmente que estão presentes nesse ficheiro simulando assim uma *cron*. Para a primeira simulação da cron criou-se o primeiro ficheiro da seguinte forma.

```
1  @echo off
2  cd [CAMINHO_DA_PASTA_ONDE_ESTÁ_INSERIDO_A_CRON]
3
4  :the_loop
5  timeout /t 20
6  echo 'a executar o cron'
7  php ler_emails.php
8
9  goto the_loop
```

Figura 72 - Simulação de uma cron através de um ficheiro do tipo bat que extrai os ficheiros excel dos emails e os coloca numa pasta.

O ficheiro é executado na pasta indicada na linha dois da figura acima e entrará num ciclo infinito em que a cada vinte segundos irá executar o código presente no ficheiro *ler_emails.php*, código esse que foi apresentado no capítulo 3 na subsecção Importação Via Email em que irá ler os emails para extrair os ficheiros de excel e irá guardar estes numa pasta do projeto.

Figura 73 - Envio de um email com o ficheiro excel para o email de teste.

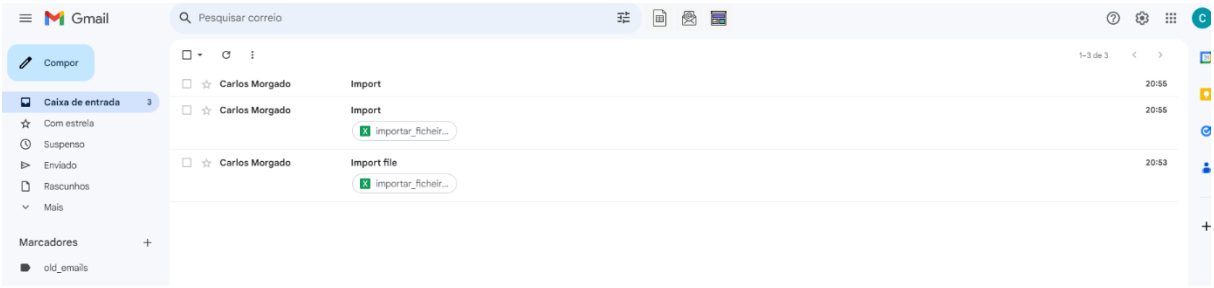


Figura 74 - Emails recebidos.

Para executar o ficheiro deve-se abrir uma linha de comandos na pasta onde se encontra o ficheiro, escrever o nome do ficheiro e executá-lo.

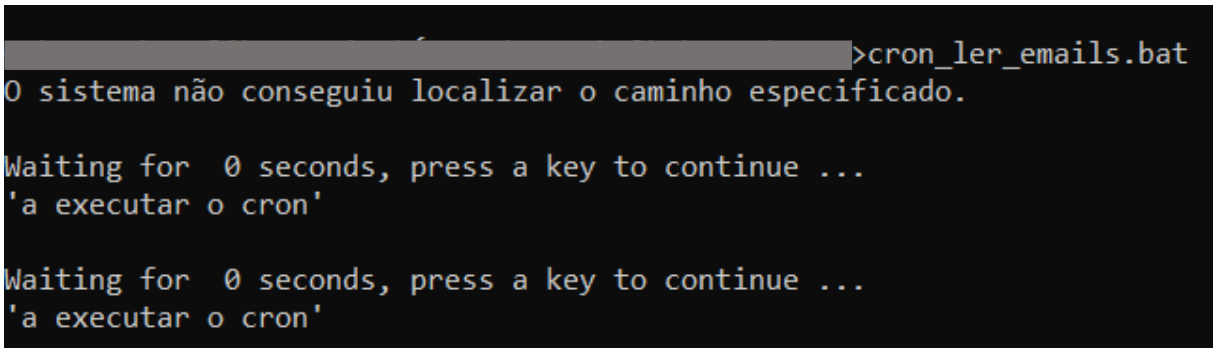


Figura 75 - Execução do ficheiro que simula a cron de leitura de emails.

Como se pode visualizar pela imagem acima, de vinte em vinte segundos o ficheiro *ler_emails.php* é executado. Portanto, os emails são lidos cujo *subject* tem a palavra *Import* contido são considerados, extraem-se os ficheiros de excel, estes são movidos para uma pasta do projeto desses emails e, por fim, os emails são movidos para o marcador *old_emails* para evitar que voltem a serem lidos aquando de uma nova execução da cron.

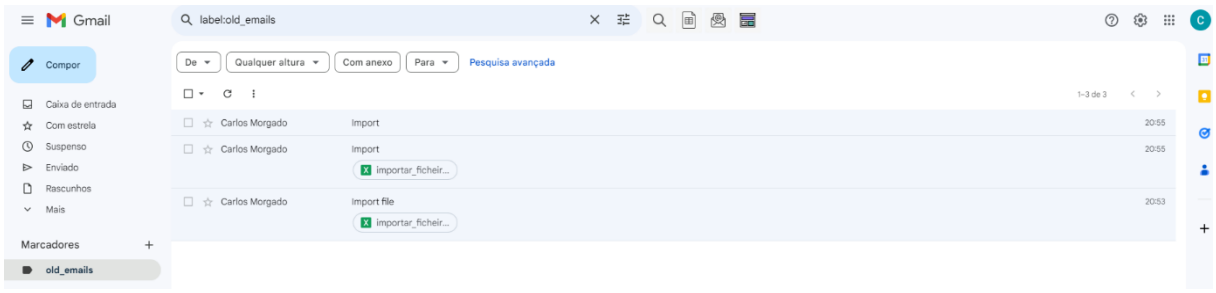


Figura 76 - Emails no marcador old_emails após terem sido lidos.

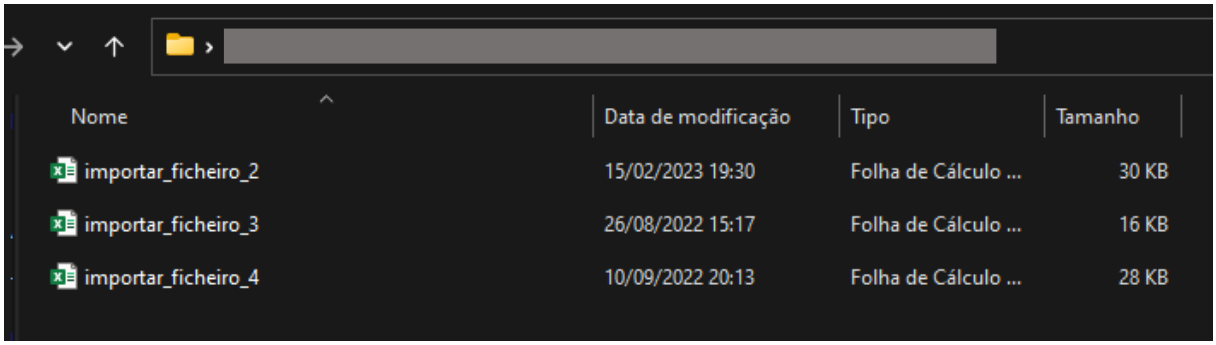


Figura 77 - Ficheiros extraídos dos emails.

Para importar os ficheiros acima exibidos, criou-se outro ficheiro da extensão bat que permite pegar nestes ficheiros um a um, iniciar o processo de importação e depois eliminá-los da pasta.

```
1  @echo off
2  cd [CAMINHO_DA_PASTA_ONDE_ESTÁ_INSERIDO_A_CRON]
3
4  :the_loop
5  cd [CAMINHO_DA_PASTA_ONDE_ESTÁ_INSERIDO_O_FICHEIRO_DA LINHA 8]
6  timeout /t 20
7  echo 'a executar o cron de importar dados...'
8  php cron_import.php
9
10 goto the_loop
```

Figura 78 - Simulação de uma cron através de um ficheiro do tipo bat importará os dados presentes no excel e os importará na base de dados.

Tal como no primeiro, também entrará num ciclo infinito em que de trinta em trinta segundos o ficheiro *cron_import.php* é executado.

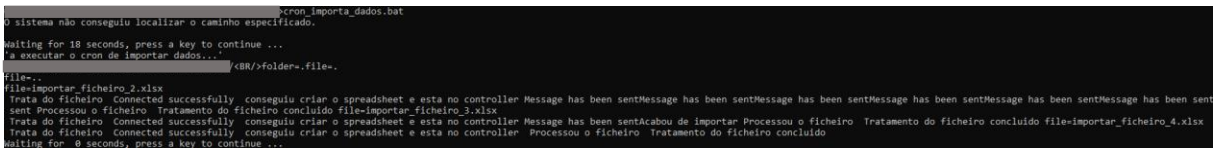


Figura 79 - Execução do ficheiro cron_importa_dados.bat que simula uma cron que permite importar os dados através de um ficheiro presente na pasta.

Como se pode ver, ao executar recolhe-se o ficheiro excel na pasta onde o ficheiro está inserido e começa o processo de importação de dados na base de dados seguindo-se a eliminação do ficheiro.

Nas imagens abaixo são apresentadas evidências de que os dados foram importados com sucesso. Na Figura 80 estão apresentados os valores lidos dos metais no ficheiro excel. Na figura seguinte é apresentado o resultado de uma *query* que mostra os valores dos metais. Se se observar as duas imagens e comparar o resto dos metais e os seus valores pode-se perceber que os valores apresentados na base de dados correspondem aos valores apresentados no excel pelo que se pode concluir que a implementação que foi apresentada no capítulo 3 está correta.

25	**Metal Tyoe 1**	
26	Metal 1.1	303
27	Metal 1.2	26
28	Metal 1.3	7.9
29	Metal 1.4	4
30	**Metal Tyoe 2**	0
31	Metal 2.1	2
32	Metal 2.2	3
33	Metal 2.3	4
34	Metal 2.4	11
35	Metal 2.5	1
36	Metal 2.6	8
37	Metal 2.7	5
38	Metal 2.8	6
39	Metal 2.9	1
40	Metal 2.10	0
41	Metal 2.11	-
42	**Metal Tyoe 3**	
43	Metal 3.1	700
44	Metal 3.2	-
45	Metal 3.3	0
46	Metal 3.4	-
47	Metal 3.5	8
48	Metal 3.6	5
49	Metal 3.7	90
50	Metal 3.8	87
51	Metal 3.9	54
52	Metal 3.10	-
53	Metal 3.11	22
54	Metal 3.12	-
55	Metal 3.13	98
56	**Metal Tyoe 4**	
57	Metal 4.1	11
58	Metal 4.2	1
59	Metal 4.3	2
60	Metal 4.4	6
61	Metal 4.5	88
62	Metal 4.6	69
63	Metal 4.7	334
64	**Metal Tyoe 5**	
65	Metal 5.1	2134
66	Metal 5.2	23.8
67	Metal 5.3	-
68	Metal 5.4	123
69	Metal 5.5	0
70	Metal 5.6	9.54

Figura 80 - Valores dos metais relativo à análise 54.

ID	Tipo	Metal	Valor
54	Metal Type 1	Metal 1.1	303
54	Metal Type 1	Metal 1.2	26
54	Metal Type 1	Metal 1.3	7.9
54	Metal Type 1	Metal 1.4	4
54	Metal Type 2	Metal 2.1	2
54	Metal Type 2	Metal 2.2	3
54	Metal Type 2	Metal 2.3	4
54	Metal Type 2	Metal 2.4	11
54	Metal Type 2	Metal 2.5	1
54	Metal Type 2	Metal 2.6	8
54	Metal Type 2	Metal 2.7	5
54	Metal Type 2	Metal 2.8	6
54	Metal Type 2	Metal 2.9	1
54	Metal Type 2	Metal 2.10	0
54	Metal Type 2	Metal 2.11	NULL
54	Metal Type 3	Metal 3.1	700
54	Metal Type 3	Metal 3.2	NULL
54	Metal Type 3	Metal 3.3	0
54	Metal Type 3	Metal 3.4	NULL
54	Metal Type 3	Metal 3.5	8
54	Metal Type 3	Metal 3.6	5
54	Metal Type 3	Metal 3.7	90
54	Metal Type 3	Metal 3.8	87
54	Metal Type 3	Metal 3.9	54
54	Metal Type 3	Metal 3.10	NULL
54	Metal Type 4	Metal 4.1	11
54	Metal Type 4	Metal 4.2	1
54	Metal Type 4	Metal 4.3	2
54	Metal Type 4	Metal 4.4	6
54	Metal Type 4	Metal 4.5	88
54	Metal Type 4	Metal 4.6	69
54	Metal Type 4	Metal 4.7	334
54	Metal Type 5	Metal 5.1	2134
54	Metal Type 5	Metal 5.2	23.8
54	Metal Type 5	Metal 5.3	NULL
54	Metal Tyoe 1	Metal Tyoe 2	0
54	Metal Tyoe 3	Metal 3.11	22
54	Metal Tyoe 3	Metal 3.12	NULL
54	Metal Tyoe 3	Metal 3.13	98
54	Metal Tyoe 5	Metal 5.4	123
54	Metal Tyoe 5	Metal 5.5	0
54	Metal Tyoe 5	Metal 5.6	9.54

Figura 81 - Valores dos metais extraídos do ficheiro excel para a base de dados.

4.2 Apresentação da Plataforma

No capítulo três abordou-se a forma como foram desenvolvidas as páginas em que são exibidos os dados em tabelas. O primeiro foi a página em que se exibem os dados relativos aos clientes. Nesse sentido, irá por se começar a apresentar essa página.

Como se pode observar, são apresentados o número e o nome de cada cliente assim como as funcionalidades do *datatable* em que são apresentados dois campos em cima da tabela (*dropdown* que permite ao utilizador escolher quantos dados é que pretende exibir na tabela e um campo de filtro que permite ao utilizador encontrar um determinado cliente escrevendo o nome ou até mesmo o número) e, também, a paginação no fundo da tabela o que é muito útil uma vez que a aplicação eventualmente poderá demasiados dados. Na imagem seguinte está apresentada o funcionamento da paginação no qual se pode ver os outros dados na página seguinte da tabela.

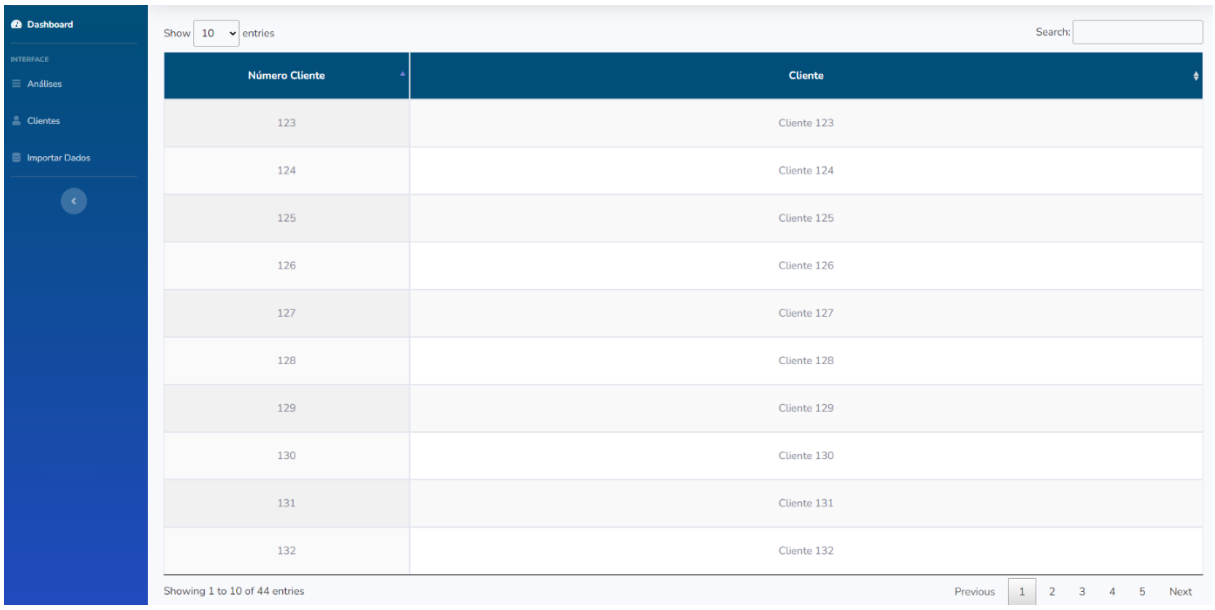


Figura 82 - Página em que se exibe os dados dos clientes.

O utilizador coloca no campo o nome do cliente e, automaticamente, somente a coluna que contem o nome do cliente é exibido na tabela. Caso se queira repor todos os dados, basta que o utilizador elimine os caracteres inseridos no filtro que os dados vão sendo repostos na tabela. As linhas da tabela são clicáveis e permitem direcionar o utilizador para uma nova página no qual, como se abordou anteriormente, serão apresentadas as máquinas do cliente e as análises das máquinas. A seguir serão apresentadas duas figuras referentes à dita página.

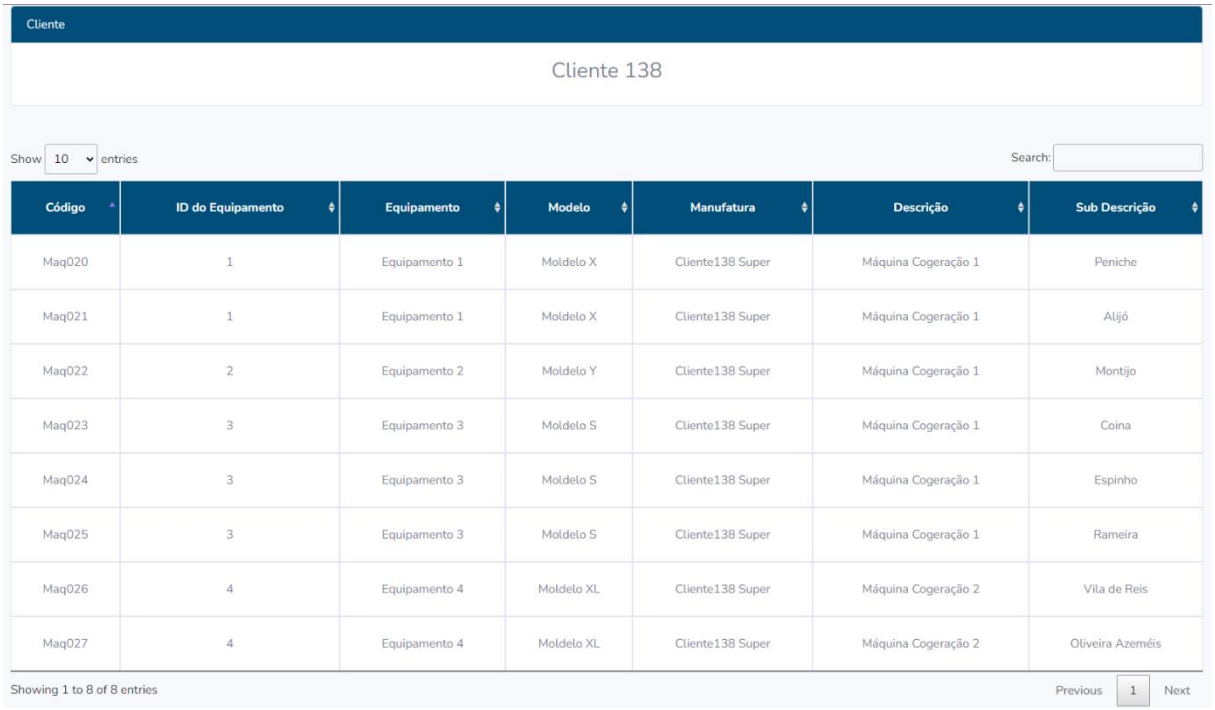


Figura 83 - Exibição das máquinas de um cliente numa tabela.

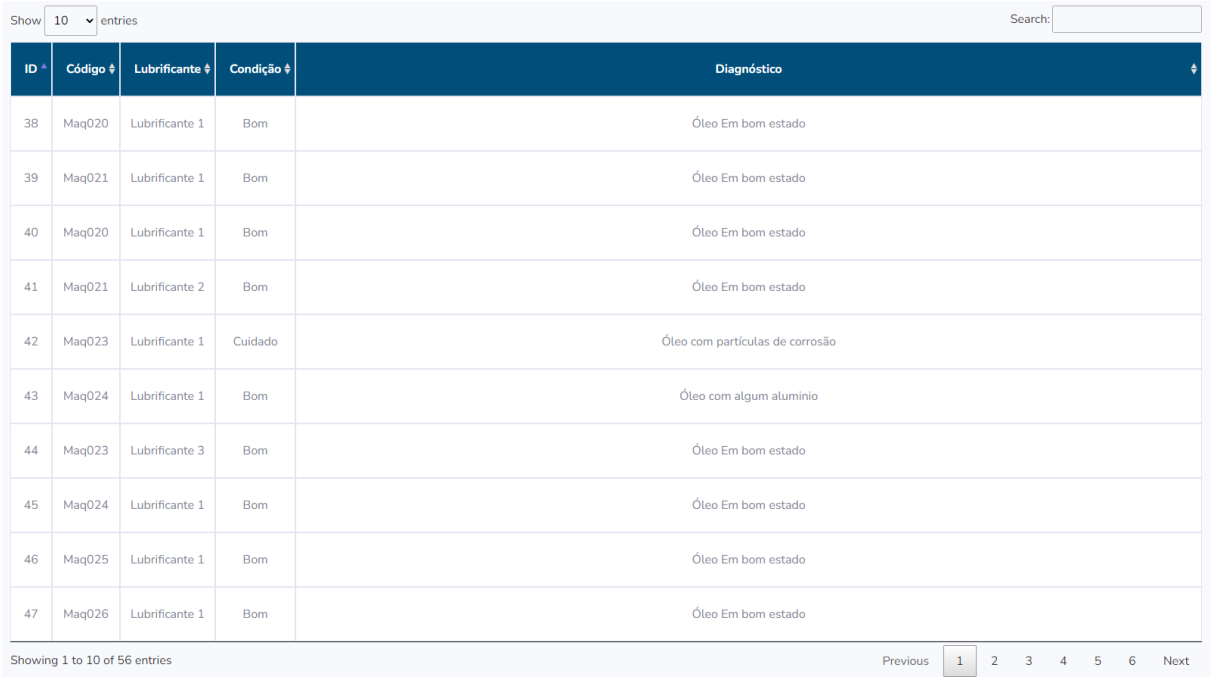
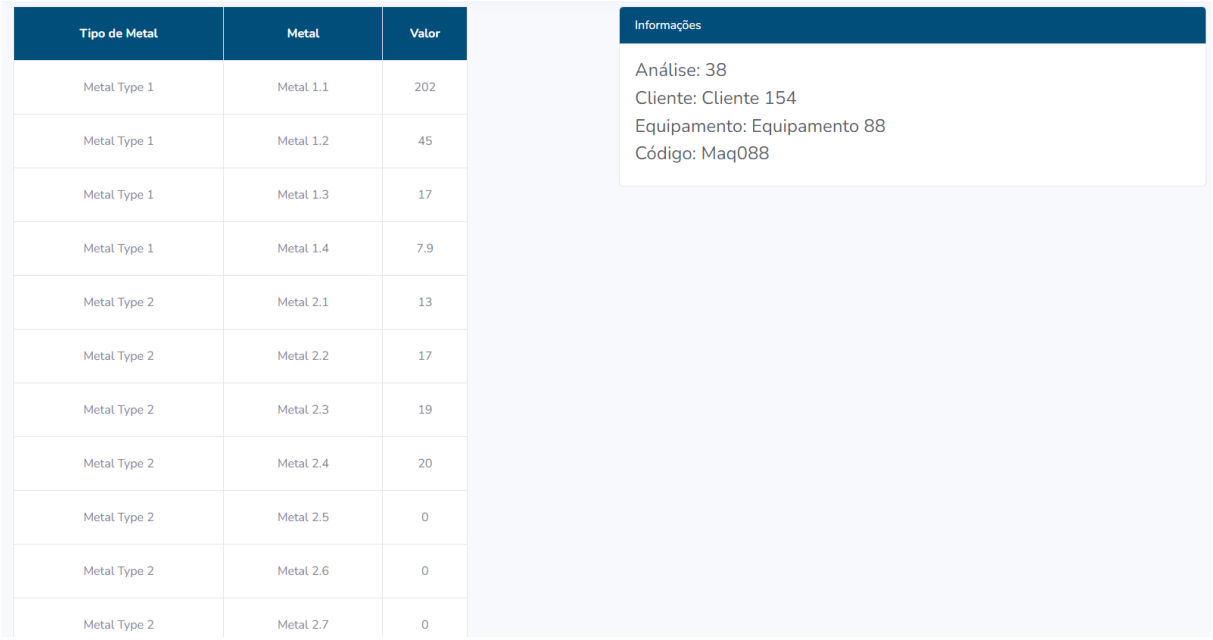


Figura 84 - Exibição das análises às máquinas de um cliente.

Na primeira figura apresentada, visualiza-se as máquinas que o cliente possui numa tabela bastante semelhante à tabela da página que lista todos os clientes enquanto na segunda

imagem, são apresentadas as análises feitas às máquinas do cliente. Tal como na página anterior, as linhas de ambas as tabelas são clicáveis direcionando o utilizador para outra página. Ao clicar-se na linha da tabela da Figura 84 o utilizador será direcionado para uma página em que mostra os valores dos metais numa tabela assim como informação adicional acerca da análise. Já na tabela apresentada na Figura 85 o utilizador será direcionado para uma nova página no qual poderá analisar a evolução de um dos metais. Ao clicar numa das linhas linha o utilizador é direcionado para a seguinte página.



Tipo de Metal	Metal	Valor
Metal Type 1	Metal 1.1	202
Metal Type 1	Metal 1.2	45
Metal Type 1	Metal 1.3	17
Metal Type 1	Metal 1.4	7.9
Metal Type 2	Metal 2.1	13
Metal Type 2	Metal 2.2	17
Metal Type 2	Metal 2.3	19
Metal Type 2	Metal 2.4	20
Metal Type 2	Metal 2.5	0
Metal Type 2	Metal 2.6	0
Metal Type 2	Metal 2.7	0

Figura 85 - Página que exibe os valores de uma análise.

Esta página é composta por dois *dropdowns* no qual um contem o código da máquina escolhida e outro que contem os nomes dos metais. No segundo *dropdown* o utilizador escolhe qual o metal que quer analisar e clica no botão *Analisar* que irá enviar um pedido ao servidor para que este devolva os valores desse metal para serem apresentados no gráfico.

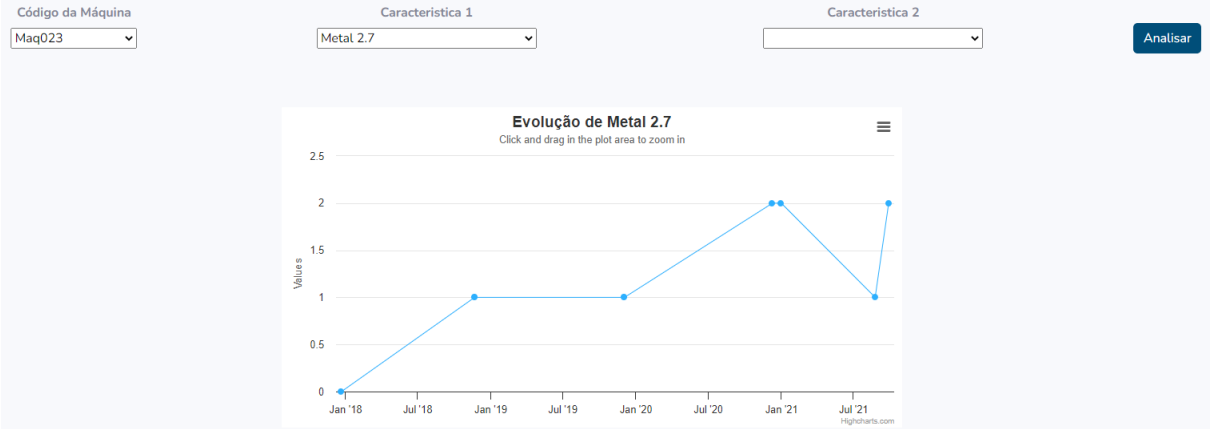


Figura 86 - Visualização do gráfico após o utilizador escolher o metal e o clique no botão.

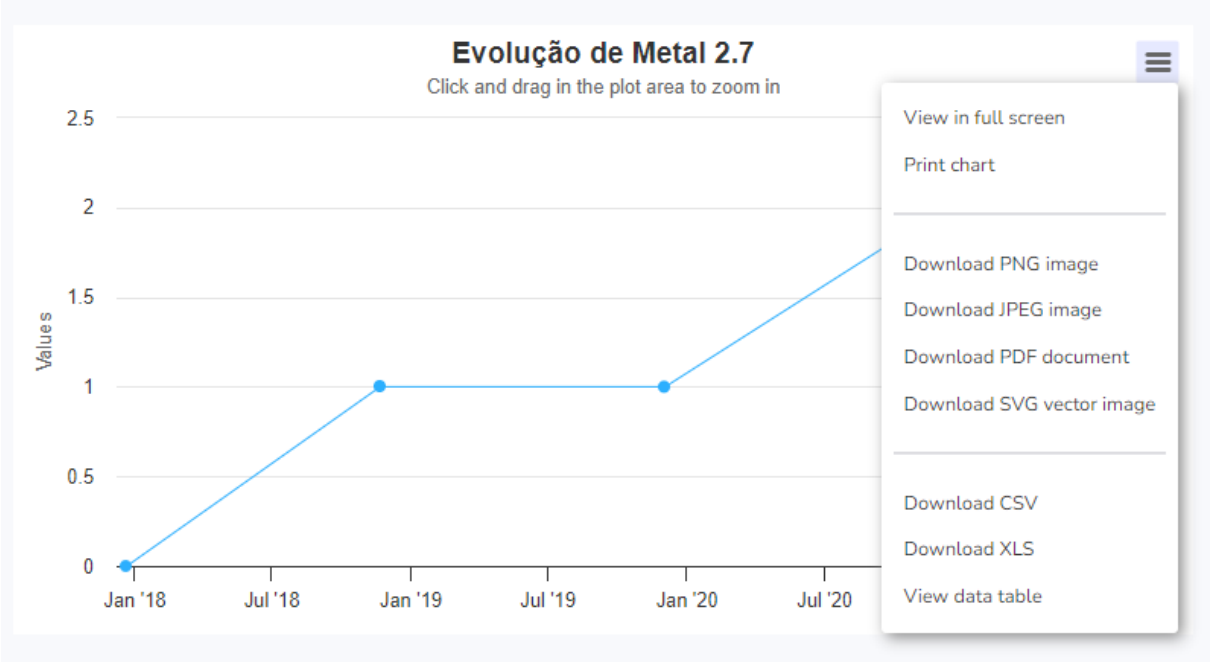


Figura 87 - Opções do gráfico.

Na imagem acima é possível observar as várias opções do gráfico que podem ser úteis ao utilizador, como por exemplo, fazer o download do gráfico ou extrair os dados para um ficheiro do tipo excel.

Por último foi também desenvolvida uma página que lista todas as análises importadas e a sua implementação foi feita à semelhança do que feita no desenvolvimento da página que lista todos os clientes. As informações presentes na tabela são a identificação da análise (*ID* da análise), código da máquina, o lubrificante, a condição da máquina e a data de conclusão da

análise. Tal como as outras tabelas apresentadas, esta também tem linhas clicáveis que permitem direcionar o utilizador para uma nova página e nela são exibidos os valores referentes ao metal. A página em questão está apresentada na figura seguinte.

Show 10 entries Search:

ID	Código da Máquina	Lubrificante	Condição	Data de Conclusão
38	Maq088	Lubrificante 1	Bom	2023-01-02
54	Maq081	Lubrificante 1	Bom	2023-02-12
3325991	Maq054	Lubrificante 1	Perigo	2023-06-02
3326077	Maq0867	Lubrificante 2	Perigo	2023-09-28
3326498	Maq040	Lubrificante 1	Perigo	2023-10-03
3326622	Maq019	Lubrificante 4	Bom	2023-10-09
3326995	Maq004	Lubrificante 6	Cuidado	2023-02-28
3326996	Maq0234	Lubrificante 10	Cuidado	2023-12-09
3326997	Maq087	Lubrificante 1	Cuidado	2023-09-09
3326998	Maq069	Lubrificante 1	Bom	2023-06-20

Showing 1 to 10 of 542 entries Previous 1 2 3 4 5 ... 55 Next

Figura 88 - Página que lista todas as análises numa tabela.

5

CONCLUSÃO

5.1 Conclusões

Ao longo deste trabalho, procedeu-se ao desenvolvimento duma plataforma de manutenção preditiva com o objetivo de automatizar a recolha de análises efetuadas em equipamentos de cogeração e proceder à verificação dos dados, diagnosticando potenciais falhas, despoletando os respetivos alertas. Pelos testes e validações apresentados no capítulo 4, o objetivo foi atingido. A plataforma recolhe os dados tanto através da plataforma como através de um envio para um email.

As páginas apresentam um aspeto visual simples e interativo, assim como funcionalidades que permitem dar ao utilizador informações relevantes acerca dos metais presentes no óleo lubrificante e assim decidir se se deve proceder ou não à manutenção da máquina. Isto faz com que evite situações em que uma eventual falha repentina que leva à necessidade de intervenção humana para descobrir a causa e também a correção da mesma no qual leva tempo o que significa que a máquina fica sem produzir óleo que, por consequência, faz com que a empresa não gere receitas. Uma vez que a empresa conhece um certo padrão, isto é, conhece o comportamento que origina as falhas nas máquinas, a plataforma permite mostrar ao utilizador a evolução do metal presente nos óleos e este pode identificar o tal padrão de uma eventual falha, isto permite que a empresa possa gerir melhor a produção dos óleos e a manutenção das máquinas minimizando os custos.

Um dos maiores desafios durante o desenvolvimento do projeto foi a importação dos dados via email uma vez que a dificuldade estava em ler os emails recebidos e garantir que estes só seriam lidos uma vez. Inicialmente não se sabia qual o melhor método para realizar esta funcionalidade. No entanto após alguma pesquisa descobriu-se os dois protocolos já

mencionados nesta dissertação sendo o *IMAP* o protocolo mais apropriado para ler os emails recebidos.

Os resultados mais relevantes para a dissertação foram a importação das análises na base de dados pois mostra que a plataforma consegue automatizar os processos mencionados no início da dissertação de uma maneira mais eficiente e eficaz.

5.2 Trabalhos Futuros

Apesar da utilidade da plataforma e das suas funcionalidades existem espaços para melhorias. Na implementação dos dados a aplicação está programada para ler 79 linhas do ficheiro de excel. Aqui existe uma potencial melhoria no sentido de se poder configurar esta funcionalidade caso a empresa meter informação adicional uma vez que este valor só dá para ser alterado no código. Outra melhoria que pode ser desenvolvida na plataforma é a interoperabilidade da plataforma com um modulo de Machine Learning para a deteção de padrões mais sofisticados.

BIBLIOGRAFIA

[1] Auvo Tecnologia SA, 2022, "Manutenção preditiva, preventiva e corretiva", https://www.blog.auvo.com/post/manutencao-preditiva-preventiva-e-corretiva, visto em 2022/09/28

[2] RIJN, Jan N. van; BISCHL, Bernd; TORGO, Luis; GAO, Bo; UMAASHANKAR, Venkatesh; FISCHER, Simon; WINTER, Patrick; WISWEDEL, Bernd; BERTHOLD, Michael R.;, and VANSCHOREN, Joaquin BERTHOLD, OpenML: A Collaborative Science Platform, 2013

[3] RIPPIN, Mike; LEMSON, Gerard; THAKAR, Ani; MEDVEDEV, Dmitry; POPP, Manuchehr Taghizadeh, SciServer: Collaborative Science Platform, 2018

[4] SANTOS, Daniela Gomes Plataforma Epik - Desenho e Desenvolvimento de Questionários Interativos como Atividades Educacionais, 2019

[5] KUROSE, James. F. & ROSS, Keith W. Redes de Computadores e a internet: Uma abordagem top-down, 3ª Edição, Editora Pearson, São Paulo– SP, 2006

[6] TANEMBAUM, Andrew S. Redes de Computadores, 7ª Edição, Editora Campus, Rio de Janeiro – RJ, 2003.

[7] RODRIGO PREIS BEOCK, Leandro; DE FREITAS CONSONE, Cibele; RODRIGUES LIMA, Leandro; PETRICA, Eder, Protocolo HTTP

[8] NASCIMENTO, Bruno Miguel Marques, Plataforma de apoio à decisão para análise e solução de erros de cravamento, 2016

[9] LERDORF, Rasmus; PHP Link: https://www.php.net/, 1995

[10] POPE Glenn E. Krasner and Stephen T., A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System, 1988

[11] ADERMANN, N.; BOGGIANO, J. Composer. Link: https:// getcomposer.org/

[12] GORLA, João Paulo Ferro, FOSCHINI, Ivan João, Arquitetura para Desenvolvimento Web Basseado em JSF 2.0 utilizando Padrões de Projeto, 2013

[13] MCCOOL, Shawn, Laravel Starter, 2012

[14] PACHECO, Felipe Guimarães, Análise das Técnicas de Segurança do Framework Laravel Contra Ataques a Aplicações Web, 2019

[15] MACÁRIO, Carla Geovana do N., Baldo, Stefano Monteiro, O Modelo Relacional, 2005

[16] CALDEIRA, Carlos Pampulim, Introdução ao Modelo de Dados Relacional, 2004

[17] DE SOUZA, Fernando José, DE SANTANA, Paulo Henrique Alves, Estudo de Caso: Análise Entre Banco de dados Relacional e Não Relacional, 2017

[18] LÓSCIO, Bernadette Farias; de OLIVEIRA, Hélio Rodrigues, PONTES, Jonas César de Sousa Pontes, NoSQL no desenvolvimento de aplicações web, 2011

[19] Brightly 2022, "Brightly Asset Essentials" https://www.brightlysoftware.com/lp/asset-essentials-cmms-software?utm_source=g2&utm_medium=reviewsite&utm_campaign=asset+essentials visto em 2022/09/29

[20] Cisco Certified Network Associate, "HTTP and HTTPS explained" https://study-ccna.com/http-https/ visto em 2022/09/29

[21] Starr Jeff 2022, Perishable Pres, 5 Easy Ways to Syntax Highlight PHP Code https://perishablepress.com/5-easy-ways-to-display-syntax-highlighted-php-code/ visto em 2022/09/29

[22] How-To Geek 2020, "What is Laravel, and how do you get started whit It?" https://www.cloudsavvyit.com/1535/what-is-laravel-and-how-do-you-get-started-with-it/ visto em 2022/09/29

[23] DB-Engines 2022, "DB-Engines Ranking" https://db-engines.com/en/ranking visto em 2022/09/29

[24] Debug Everything 2021, " Qual a diferença entre base de dados relacional e não relacional?" https://blog.debugeverything.com/pt/diferenca-base-de-dados-relacional-e-nao-relacional/ visto em 2022/09/29

[25] RapidMiner 2001, https://rapidminer.com/ visto em 28/04/2023

[26] How to get emails using PHP and IMAP ? - GeeksforGeeks How to get emails using PHP and IMAP, 2020