Instituto Politécnico de Tomar

Escola Superior de Tecnologias de Tomar

Ricardo António da Rocha da Cruz

Relatório de Estágio na Equipa RD da Compta Emerging Business

Relatório de Estágio

Orientado por:

Doutora Sandra Jardim – Instituto Politécnico de Tomar
Engenheiro Ricardo Tavares – Compta Emerging Business

Relatório de Estágio
apresentado ao Instituto Politécnico de Tomar
para cumprimento dos requisitos necessários
à obtenção do grau de Mestre
em Produção de Conteúdos Digitais

Dedico este esforço a todos os professores, colegas, amigos e familiares que contribuíram para a execução deste Estágio e Relatório

RESUMO

Hoje em dia todas as empresas se deparam com um problema relacionado com a gestão de faturas. Sem melhores hipóteses, a maior parte destas empresas encontra-se numa situação em que é necessário gerir e organizar quantidades enormes de informação, sem grande suporte a nível digital. Este suporte é necessário para que as empresas possam ter fácil acesso à informação contida nas faturas de forma rápida e eficiente. Para além disso, é muito difícil conseguir extrair informação conjunta (tal como custos de dados consumos a uma dada data, consumos gerais, custos gerais, etc.) de dadas faturas em papel de modo a poder analisar consumos ou gerir custos.

Este estágio foi executado no âmbito da criação de uma aplicação *web* que pudesse responder a esta necessidade das empresas. Esta aplicação tem de ser capaz de importar faturas em formato digital e extrair toda a informação que lhes diga respeito. Adicionalmente a aplicação tem de ser capaz de processar e transformar essa informação de modo a promover uma melhor compreensão por parte do utilizador.

Estes objetivos foram conseguidos através da implementação da aplicação especificada por meio da utilização de tecnologias *web* como *AngularJS* e *Ruby on Rails*.

Palavras-chave: Faturas, data, mining, gestão, empresas

ABSTRACT

Nowadays all companies are faced with an issue related to invoice management. Without a better choice or option, most of these companies are in a situation where it is necessary to manage and organize huge amounts of information without digital support. This support is necessary so that companies can have easy access to the information contained in the invoices quickly and efficiently. Moreover, it is very difficult to extract joint information (such as consumption costs at a given date, general power or resources consumption, general costs, etc.) of given paper invoices to analyze consumption or manage costs.

This internship was carried out with the intent of creating a web application that could respond to these needs. This application must be able to import invoices in digital format and extract all the information that concerns them. Additionally, the application must be able to process and transform this information in order to promote a better understanding on the part of the user.

These goals were achieved by implementing said application through the use of web technologies such as AngularJS and Ruby on Rails.

Keywords: Invoices, data, mining, companies, management

AGRADECIMENTOS

Este relatório e estágio concretizou-se graças à grande ajuda dos meus orientadores Professora Sandra Jardim e Engenheiro Ricardo Tavares que, incansáveis, se mostraram sempre disponíveis para me ouvir e/ou aconselhar.

Sem nunca esquecer todos os meus colegas de trabalho, membros da equipa *R&D*, que me ajudaram desde o inicio a crescer como profissional.

Deixo também uma palavra de consideração a todos os professores do mestrado, que ao longo desta caminhada, me apoiaram e prepararam de forma a conseguir superar os obstáculos encontrados ao longo da minha, ainda curta, carreira profissional.

Gostaria também de deixar um especial agradecimento a todos os colegas que contribuíram e me ajudaram ao longo da composição deste relatório. A estes, e a todos os que indireta ou diretamente também contribuíram para a concretização do relatório, um forte e sentido obrigado.

Índice

Índice de Figuras

Índice de Tabelas

Índice de Códigos

1. Introdução

O presente relatório de estágio é elaborado no âmbito da unidade curricular de Estágio com vista à aquisição do grau de Mestre em Produção de Conteúdos Digitais.
Este estágio decorreu no *Tagus Valley* Abrantes com a minha integração na Equipa de *Research and Development* da empresa *Compta Emerging Business*, num período de 3 meses.

Ao longo destes 3 meses foi-me proposto que, juntamente com parte da equipa RD, desenvolvesse uma aplicação *web* para uso empresarial cujo objetivo seria a gestão e monitorização de gastos e consumos através da leitura digital de faturas. Ao conseguir ler, analisar e armazenar dados dessas mesmas faturas, tornou-se possível desenhar mapas com vários **Pontos de Instalação,** desenhar gráficos com informações sobre rácios de gastos/consumos e tornar o trabalho de análise do utilizador numa tarefa muito mais fácil e prática de ser executada.

2. A empresa *Compta Emerging Business*

Fundada em 1972, a empresa ***Compta*** iniciou o seu funcionamento com serviços de processamento de dados para o sector TI emergente, rapidamente se virando mais para a área de Sistemas de Informação para Empresas, Setores Públicos e Telecomunicações. Sendo que logo nos primeiros anos da sua existência se tornou a referência no que toca a Integração de Sistemas no mercado TI Nacional.

Em 1983 a empresa iniciou a sua expansão orientando-se mais para as áreas de *Telecom* e Setores de Sistemas de Comunicação. Entre 1983 e 1992 teve também a sua integração no ***Grupo Compta*** criado em 1987. Até ao ano 2000 o ***Grupo Compta*** atingiu metas de liderança na área de Integração de Sistemas e criou parcerias com líderes a nível mundial das áreas e indústrias do *ICT* (*Information and Communication Technologies)*.

Em 2001 a ***Compta Emerging Business*** é fundada, criando assim a empresa dedicada à produção de produtos *Software* para Indústrias e *Smart Cities*. Com os seus primeiros anos de atividade concentrados apenas na área de *R&D* (*Research and Development)* e implementação de projetos, a empresa ***CEB*** entregou as suas primeiras soluções para Câmaras Municipais e Industrias, na área de **Eficiência Energética** e **Gestão de Resíduos**.

Entre 2010 e 2015 a empresa ***Compta Emerging Business*** continuou a efetuar com sucesso a entrega de soluções para *Smart Cities* nas áreas de **Gestão de Resíduos, Ambiente, Transportes e Logísticas e Eficiência Energética.** É-lhe atribuído o prémio de prestígio "*Green Project Award*" pela melhor solução TI no que toca à gestão de **Resíduos Urbanos**. Mais para o final deste período são fundados os primeiros polos de operação da equipa *R&D* em Lisboa e Porto.

Entre 2016 e a presente data são fundados mais dois polos de operação em Évora e Abrantes e são registadas utilizações dos produtos ***CEB*** em mais de 40 Cidades em áreas diferentes. Para além disso os produtos ***CEB*** passam também a ser utilizados a nível mundial por um total de 27 Países através de Parcerias e Acordos com Integradores de Sistemas locais.

3. *Framework AngularJS*

AngularJS diz respeito a uma *framework* de *Javascript open-source*, mantida pela empresa *Google*, que tem como objetivo o desenvolvimento e execução de *aplicações de* Front-end. Esta *framework* comtempla também uma outra chamada ***Apache Cordova***, responsável pelo desenvolvimento de aplicações multiplataforma. O principal propósito do ***AngularJS*** consiste na simplificação de ambos o desenvolvimento e testes de aplicações *web* por meios de uma *framework client-side* que se rege pelas hierarquias **MVC** (*Model-View-Controller)* e **MVVM** (*Model-View-View-Model*) [3].

O seu funcionamento baseia-se na leitura de páginas *HTML* compostas de tributos e *tags* adicionais ao *HTML* nativo. O ***AngularJS*** interpreta estas *tags* como diretivas de modo a ligar partes da página *HTML* a elementos de *input* e *output* representados sob a forma de variáveis *Javascript* normais. Estas variáveis podem ser manipuladas diretamente por código (dentro de um controlador) ou através de elementos *HTML* que lhes estejam ligados. Este modo de atuação permite a criação de páginas *web* dinâmicas capazes de adaptar o seu conteúdo de acordo com estados / conteúdos de variáveis que as controlam [2].

Figura 1 - Logotipo da framework AngularJS

3.1. Estrutura de uma aplicação AngularJS

Como já referi anteriormente esta *framework* baseia-se nas arquiteturas *MVC e MVVM.* Para tal existem três componentes essenciais no que toca à estrutura de uma aplicação desenvolvida em ***AngularJS*.** Estes componentes são, nada mais, nada menos, que as ***Modules***, ***Views*** e os ***Controllers*** [2].

3.1.1. Módulos

Os ***Módulos*** funcionam como que contentores para as diversas seções de uma aplicação desenvolvida em *AngularJS*. Cada aplicação desenvolvida nesta *Framework* tem de ter pelo menos um ***Módulo*** [2]**.** Como já referi acima, *AngularJS* atua sobre a linguagem *Javascript*, como tal a declaração de um ***Módulo*** dá-se da seguinte forma:

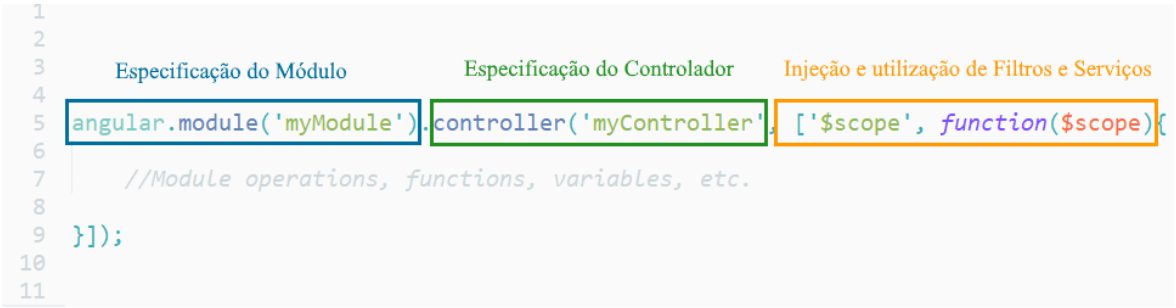
```
3  var myModule = angular.module('myApp', []);
```

Código 1 - Declaração de um AngularJS Module

Os ***Módulos*** incluem ***Controladores, Serviços, Filtros*** e ***Diretivas,*** sendo que estes conteúdos só podem ser acedidos dentro do *Scope* do módulo onde foram declarados. Por outras palavras e exemplificando, os **Controladores,** e suas funções, incluídos no módulo declarado no excerto de código acima, só poderão ser utilizados ou acedidos em elementos *HTML* que integrem a diretiva ***ng-app=”myApp”*** ou sejam filhos de um elemento que a integre.

3.1.2. Controladores

Os **Controladores** podem ser considerados o cérebro de uma aplicação desenvolvida em *AngularJS*. A construção e declaração de um controlador pode ser dividida em três partes, estando estas representadas na figura seguinte.



Código 2 - Exemplo da declaração de um controlador

O primeiro passo consiste, tal como o nome indica, na especificação do **Módulo** ao qual o **Controlador** estará associado. O segundo passo consiste apenas na declaração do **Controlador** em si. Os **Controladores** e todos os outros componentes de um Módulo (até mesmo o **Módulo** em si) são sempre referenciados pelo seu nome que deve ser único em toda a aplicação. Por último, dá-se a **Injeção de Dependências** (declaração em *String*) e posteriormente utilização (parâmetros da função) das dependências injetadas.

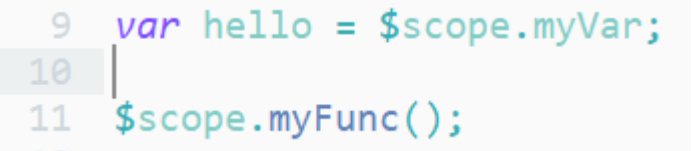
Como já referi anteriormente, os controladores são responsáveis por todas as operações de uma ou mais ***Views*** através da interação com funções e variáveis especiais. Estas funções ou variáveis podem ser designadas por **Funções e Variáveis de *Scope***. Em *AngularJS*, o *Scope* representa todos os modelos da Aplicação. No fundo o ***Scope*** é a ponte que liga a parte visual da página *web* à parte lógica. Posto isto, apenas funções e variáveis de ***Scope*** podem ser acedidas na estrutura *DOM* do *HTML*.

A declaração de funções ou variáveis de ***Scope*** é feita como se estivéssemos a falar de uma variável *Javascript* normal, com a exceção de esta apresentar o prefixo "$scope".

```
$scope.myVar = 'Hellow World!';

$scope.myFunc = function(){
    Console.log('Hello World!');
};
```

Código 3 - Exemplo da declaração de variáveis e funções de Scope

Seguindo a mesma filosofia, a utilização destas variáveis ou funções procede da mesma maneira, adotando apenas o prefixo "$scope".

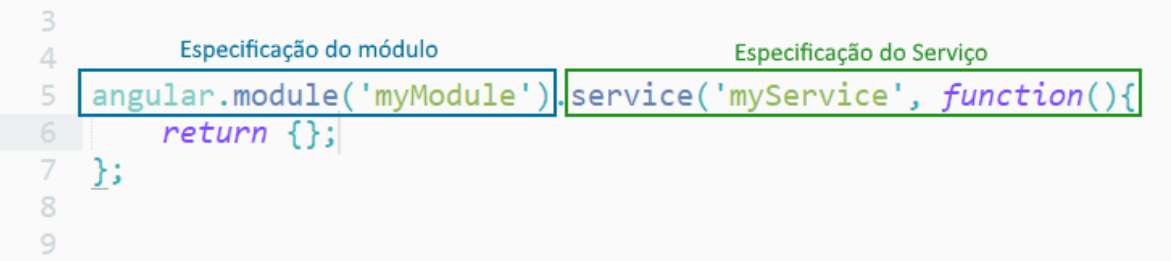


Código 4 - Exemplo de utilização de variáveis e funções de Scope

Por outro lado, aquando da sua utilização em Views, o prefixo "$scope" não é utilizado (mais informações no capitulo relativo às *Views*). Os Controladores também podem conter variáveis e funções nativas de *Javascript*, a única diferença é que estas não podem ser acedidas ou alteradas através da interação direta com as *Views*.

3.1.3. Serviços

Os **Serviços** são objetos partilhados por toda a aplicação por meio de **Injeção de Dependências.** Estes permitem criar *snipets* de código que pode ser usado em qualquer parte da aplicação, evitando assim a sua repetição. Em específico, no projeto em que este relatório se baseia, os **Serviços** são muito utilizados para guardar e reutilizar função de chamadas às *APIs*.



Código 5 - Exemplo de declaração de um Serviço

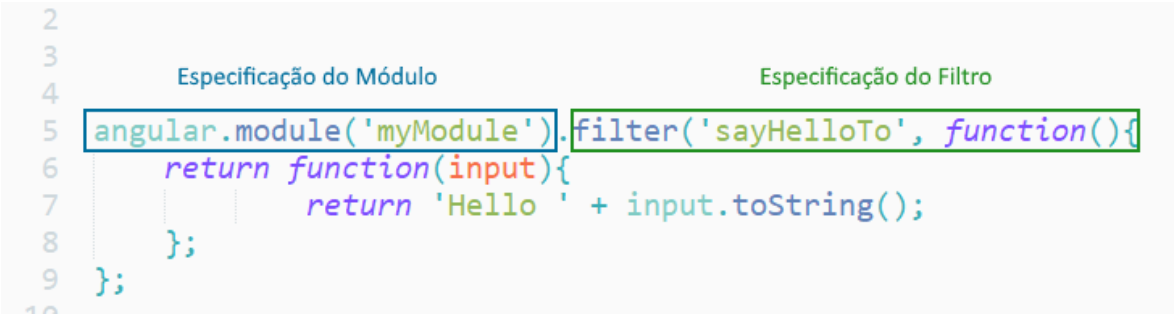
Os **Serviços** são de domínio restrito ao dos controladores, ou seja, só podem ser utilizado na parte de controlo da aplicação (*Javascript)*. De modo a utilizar quaisquer funções ou propriedades de um **Serviço** é necessário, primeiro injetá-lo no Controlador pretendido. Esta operação é feita como demonstra o *Código 2* apresentado na secção respeitante aos **Controladores**. Após injetar o **Serviço** pretendido, este pode ser utilizado normalmente como se tratasse de uma variável ou função de *Javascript*.

```
angular.module('myModule').service('myService', function(){
    return {
        sayHello: function(){
            console.log('Hello');
        }
    };
};

angular.module('myModule').controller('myController', ['$scope', 'myService', function($scope, myService){

    function init(){
        myService.sayHello();
    }

    init();

}]);
```

Código 6 – Exemplo de utilização de um Serviço

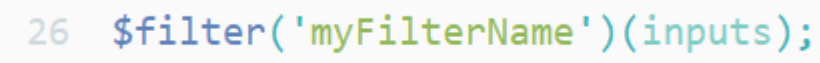
3.1.4. Filtros

Os **Filtros**, tal como o nome indica, servem para filtrar informação que lhes é passada de modo a troná-la mais "apresentável". Os **Filtros** podem ser usados em Controladores, Views, Diretivas ou Serviços.



Código 7 - Exemplo da declaração de um Filtro

Tal como nos **Serviços** é necessário primeiro injetar um **Filtro** para o poder utilizar no âmbito de um Controlador, no entanto, isto é feito de maneira diferente. Como os filtros podem ser utilizados em diversos sítios, não faria muito sentido estes só serem acedidos caso o seu identificador fosse injetado. Como tal, em vez de injetar o **Filtro** propriamente dito, injeta-se antes o serviço "*$filter*". Este serviço é nativo do *AngularJS* e a sua utilização a nível de um Controlador encontra-se exemplificada na figura seguinte:



Código 8 - Exemplo da utilização do Serviço $filter

Deste modo a utilização de um filtro a nível de um Controlador (ou qualquer outra componente escrita em *Javascript)* poderá ser feita da seguinte forma:

```
4
5  angular.module('myModule').filter('sayHelloTo', function(){
6      return function(input){
7          return 'Hello ' + input.toString();
8      };
9  };
10
11
12 angul                                                    $filter){
13
14     f
15
16         console.log(greeting);
17     }
18
19     init();
20
21 }]);
```

Código 9 - Exemplo da utilização de filtros a nível do Controlador

```
5
6  <div>{{'Ricardo' | sayHelloTo}}</div>
7
```

Código 10 - Exemplo da utilização de um Filtro no âmbito de uma View

Por outro lado, a nível de um *View* a utilização de um filtro é mais simplificada e representada pelo caracter "|". Nota: Falarei da utilização dos caracteres "{{" e "}}" na secção das *Views.*

3.1.5. Diretivas

As **Diretivas** em *AngularJS* podem ser divididas em quatro tipos: **Diretivas de Elemento**, **Diretivas de Atributo**, **Diretivas de Classe** e **Diretivas de Comentário**.

As **Diretivas de Elemento** apresentam-se sob a forma de um elemento *HTML* e representam-se exatamente da mesma forma, como demonstra a figura seguinte:

```
10
11  <my-directive></my-directive>
12
```

Código 11 - Exemplo da representação de uma Diretiva de Elemento

As **Diretivas de Atributo** apresentam-se sob a forma de atributos presentes em elemento *HTML* ou até mesmo em outras **Diretivas de Elemento**:

```
14
15  <div my-directive=""></div>
16
```

Código 12 -Exemplo da representação de uma Diretiva de Atributo

As **Diretivas de Classe** seguem exatamente a mesma filosofia das anteriores e apresentam-se sob a forma de classes *CSS* de um elemento *HTML*:

```
18
19  <div class="my-directive"></div>
20
```

Código 13 - Exemplo da representação de uma Diretiva de Classe

Por fim, as **Diretivas de Comentário** apresentam-se sob a forma de comentários presentes numa página *HTML*:

```
23
24  <!-- directive: my-directive -->
25
```

Código 14 - Exemplo da representação de uma Diretiva de Comentário

Tal como os **Filtros** e **Serviços** as **Diretivas** têm de ser declaradas e construídas a partir de um **Módulo.** Esta declaração é feita de forma muito idêntica às outras

```
3
4          Declaração do Módulo                    Declaração da Diretiva
5  angular.module('myModule').directive('myDirective', function(){
```

Código 15 - Exemplo da declaração de uma Diretiva

componentes que referi anteriormente e encontra-se exemplificada na figura seguinte:

Na declaração de uma **Diretiva** é possível também definir alguns parâmetros relativos ao seu funcionamento. Estes parâmetros encontram-se explicados na tabela seguinte:

Parâmetros	Descrição

scope	Parâmetro que especifica as variáveis que vão compor o *Scope* da diretiva aquando da sua inicialização. Estas variáveis podem tomar valores provenientes de *inputs* dados à diretiva ou serem inicializadas com um valor à escolha.
	`scope: {`
	`  info: '=info'`
	`},`

templateUrl	Parâmetro que define a *template* que a diretiva irá utilizar, caso esta necessite. Este parâmetro requer um caminho direto para um ficheiro *HTML*, ficheiro esse que será importado para a página *web* aquando do carregamento da diretiva que o solicita. `templateUrl: 'my-directive.html'`
Restrict	Parâmetro que restringe a associação de variáveis do *Scope* a certos componentes do *DOM*. Este parâmetro pode tomar os valores: 'A', 'E', 'C' e 'M'. Este parâmetro é o responsável pela distinção de uma diretiva entre os seus quatro tipos. `restrict: 'A',`
Controller	Parâmetro que permite especificar um controlador dedicado para a diretiva. Este parâmetro aceita não só uma variável que aponte para a declaração do controlador, mas também aceita apenas o nome do controlador pretendido. `controller: 'myDirectiveController'`

Tabela 1- Parâmetros aceites na declaração de uma Diretiva

Posto isto, o resultado final da declaração de uma **Diretiva** acabará com um aspeto muito idêntico ao seguinte:

```
14  angular.module('myModule').directive('myDirective', function(){
15      return {
16          restrict: 'A',
17          scope: {
18            info: '=info'
19          },
20          templateUrl: 'my-directive.html',
21          controller: 'myDirectiveController'
22      };
23  });
```

Código 16 - Exemplo da declaração completa de uma Diretiva

3.1.6. Views

As *Views* representam qualquer tipo de ficheiro *HTML (.html)* que diga respeito a uma página ou parte de uma página *web*. Estes ficheiros *HTML*, regra geral, estarão sempre associados a um ficheiro *Javascript (.js)* responsáveis pelo seu funcionamento aquando da sua utilização numa página *web*.

Tal como foi explicado anteriormente, *AngularJS* baseia-se no princípio da interação *View – Controller* através de ligações feitas entre elementos ou *tags HTML* e variáveis de *Javascript* nativo. Estas ligações são conseguidas através de ***angular-directives.*** Estas diretivas dizem respeito a *tags* (idênticas às " ", "", etc) ou atributos de *tags* (idênticos a "src", "class", "style", etc) que serão embutidos no código *HTML* das páginas de uma aplicação e que, posteriormente, irão ser lidos pela *framework* de modo a processar as suas ligações com os *Controladores*.

Para além de proporcionar a comunicação *View-Controller* estas diretivas também possibilitam a inclusão de *Views* parciais dentro de outras *Views*.

A principal diferença entre *tags* nativas e as diretivas do *AngularJS* consiste no fato de as diretivas do *AngularJS* serem sempre iniciadas com o prefixo "ng-".

Diretivas	**Descrição**
ng-app	Esta diretiva tem como objetivo designar qual o módulo registado irá ser usado para a página *web* em questão. Ex. * *
ng-controller	Esta diretiva permite definir qual o controlador responsável pelas operações que ocorram no elemento *HTML*, à qual está ligada, e aos seus elementos filhos. Só é possível ligar um controlador a um elemento *HTML* caso este elemento ou possíveis pais do mesmo estejam ligados ao módulo *AngularJS* (através da diretiva *ng-app*) ao qual o *controller* pertence. Ex. * *
ng-init	Diretiva que serve para executar uma função ou expressão de *Scope* assim que o elemento *HTML*, ao qual está associada, é carregado para a página *web*. Ex. * *
ng-click	Diretiva que executa uma função ou expressão de *Scope* assim que o *AngularJS* detetar um evento de clique no elemento *HTML* ao qual esta diretiva se encontra associada. Ex. * *
ng-model	Diretiva que tem o propósito de “ligar” uma variável de *Scope* a um elemento *HTML* de *Input/Output* Ex. * *
ng-bind	Diretiva responsável por ligar uma variável de *Scope* a qualquer elemento *HTML*, alterando assim o conteúdo desse elemento (*Inner HTML)* para corresponder ao conteúdo da *variável* que lhe foi associada.

	Ex. * *
ng-include	Diretiva que “importa” os conteúdos de um ficheiro *HTML* e o insere dentro de uma *View* sob a forma de *View* parcial. Ex. * *
ng-if	Diretiva que adiciona / remove da página web não só o elemento *HTML*, em que está embutida, mas também os seus filhos, de acordo com o resultado da expressão *boleana* que lhe é passada. Ex. *<div ng-if=” true”>* - adicionado Ex. * * - removido
ng-show	Esta diretiva funciona de maneira idêntica à diretiva *ng-if*, com a diferença de que esta apenas esconde o elemento *HTML* (e seus filhos) através da propriedade css “*visibility: hidden*”. Ex. *<div ng-show=” true”>* - visível Ex. * * - escondido

Tabela 2 - Exemplos de diretivas AngularJS

Uma outra possível interação entre um Controlador e uma *View* dá-se através do uso direto de variáveis de *Scope* no *HTML*. Esta operação é possível através do uso de ‘{{’ e ‘}}’. Estes dois conjuntos de caracteres aceitam variáveis, funções ou até mesmo operações lógicas. Na figura seguinte estão representados três exemplos distintos da sua utilização:

```
 8
 9  <span>{{myVar}}</span>
10
11  <span>{{myFunction()}}</span>
12
13  <span>{{myVar == 0 ? 'Hello' : 'World!'}}</span>
14
```

Código 17 - Exemplos das diferentes utilizações de variáveis de Scope numa View

4. Framework Rails

Ruby on Rails ou simplesmente ***Rails*** é uma *web framework server-side* desenvolvida em ***Ruby*** que respeita a arquitetura *MVC (Model View Controller)*. Esta *framework* tem como principais objetivos a facilidade de utilização de transferências de dados através de *JSON* e *XML* [3].

Figura 2 - Logotipo da framework Rails

4.1.1. Controladores

Os **Controladores,** tal como em *AngularJS* são responsáveis pelo funcionamento de toda a aplicação *server-side*. A declaração de um controlador em *Rails* é bastante simples e assemelha-se muito a linguagens de programação orientadas a objetos visto que, um controlador, é no fundo uma classe. Esta classe (Controlador) irá herda sempre, direta ou indiretamente, uma outra classe nativa da *framework*: *ApplicationController* [2].

```
5
6     Nome do Controlador          Herança do Controlador
7   class myController < ApplicationController
8
9
10
11  end
12
```

Código 18 - Exemplo da declaração de um Controlador em Rails

Para fins deste relatório, irei abordar a introdução a esta *framework* de uma forma mais específica de modo a ir ao encontro dos desenvolvimentos efetuados durante o meu estágio. Como tal, a estrutura dos Controladores criados neste processo respeita algumas regras impostas à equipa que integrei.

- Todos os controladores regras de nomenclatura que vão ao encontro das suas funções. Por exemplo, um Controlador que efetue operações sob um dado modelo terá o seu nome baseado nesse modelo adicionado do prefixo "*Controller*". **Ex. *class modelNameController***
- Todos os métodos de um Controlador são escritos com *snake case*, ou seja, quaisquer palavras que componham o nome do método serão separadas por *underscores*. **Ex. *def my_method***
- Métodos que não representem *endpoints* de rotas (mais informações no Capítulo respeitante às rotas), deverão ser declarados com privados.

```
5
6  #Nome do Controlador respeita o nome do Modelo sobre o qual atua
7  class myModelController < ApplicationController
8
9      #Métodos declarados com snake_case
10     #Método respeitante a um endpoint de rotas
11     def my_endpoint_method
12
13         puts say_hello
14
15     end
16
17     #Secção de métodos privados
18     private
19
20     #Método privado
21     def say_hello
22
23         puts 'Hello World!'
24
25     end
26
27 end
```

Código 19 - Exemplo das práticas usadas na estrutura de Controladores em Rails

Na figura seguinte encontram-se exemplificados os três pontos que referi acima.

4.1.2. Modelos

Em *Ruby on Rails* os modelos representam uma interface entre os Controladores e as tabelas da Base de Dados. Deste modo cada Modelo diz respeito a uma tabela na Base de Dados. O *Rails* é capaz de fazer esta ligação automaticamente desde que a declaração do Modelo seja feita seguinte regras de nomenclatura (isto deve-se ao facto do *Rails* usar muito associações de sintaxe para correlacionar modelos ou variáveis com outros elementos sem que a sua ligação ou relação se encontre explicita). Os Modelos tal como os Controladores são representados por classes em *Ruby*. A sua declaração é feita da seguinte forma:

```
6  class myModel
7
8
9
10 end
```

Código 20 - Exemplo da declaração de um modelo em Rails

Os modelos no *Rails* são usados para executar *queries* à base de dados mantendo os resultados num formato de ***ActiveRecord*** (mais informação adiante), para tal estes possuem métodos que correspondem a determinados comandos de base de dados. Consideremos a tabela de base de dados representada na figura seguinte:

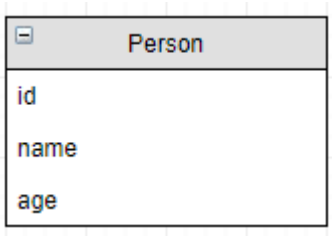


Figura 1 - Exemplo de Tabela para a Entidade "Person"

A tabela seguinte exemplifica os principais métodos de *queries*:

Método	Descrição

all	Método que executa uma *query* que vai buscar todos os resultados possíveis de uma tabela.
	
	`#ActiveRecord`
	`Person.all`
	
	`#SQL Query`
	`SELECT * FROM 'persons'`

where	Método que executa uma *query* de pesquisa à base de dados. Este método recebe qualquer número de parâmetros sendo que estes dizem respeito a colunas da tabela pesquisa na base de dados, seguidos do valor pelo qual a pesquisa se deve de basear.
	
	Ex. Pesquisar a tabela "persons" por uma pessoa cuja idade seja "21" e o nome seja "João".
	
	`#ActiveRecord`
	`Person.where(age: 21, name: 'João')`
	
	`#SQL Query`
	`SELECT * FROM 'persons' p`
	`WHERE p.age = 21 AND p.name = 'João'`

find_by	Este método atua da mesma forma que o método *where* com a única diferença de que retorna apenas o primeiro registo encontrado.
	
	Ex. Seguindo o mesmo exemplo do método *where*
	
	`#ActiveRecord`
	`Person.find_By(age: 21, name: 'João')`
	
	`#SQL Query`
	`SELECT * FROM 'persons' p`
	`WHERE p.age = 21 AND p.name = 'João'`
	`LIMIT 1`

first	Este método tanto pode ser usado em resultados de outras *queries* como sozinho e tem como objetivo limitar os resultados a um dado número de registos tendo em conta a sua ordem. Ex. No seguinte exemplo encontra-se demonstrada uma *query* que vai ```17 #ActiveRecord 18 Person.first(3) 19 20 #SQL Query 21 SELECT * FROM persons ORDER BY clients.id ASC LIMIT 1``` buscar os três primeiros registos da tabela.
last	Este método funciona exatamente da mesma maneira que o método *first* com a única diferença de que em vez de ir buscar os primeiros n resultados, vai buscar os ultimos n. Ex. No seguinte exemplo encontra-se demonstrada uma *query* que vai ```17 #ActiveRecord 18 Person.last(3) 19 20 #SQL Query 21 SELECT * FROM persons ORDER BY clients.id DESC LIMIT 1``` buscar os três ultimos registos da tabela.
order	Este método permite ordenar os resultados de uma *query* de acordo com uma ou mais colunas da tabela pesquisada. Ex. Ordenar os registos da tabela 'persons' pela sua idade, de forma ascendente ```17 #ActiveRecord 18 Person.order(age: asc) 19 20 #SQL Query 21 SELECT * FROM persons ORDER BY clients.age ASC```
select	Este método permite selecionar colunas especificas de um tabela. Pode ser utilizado sozinha caso em que irá retornar todos os registos existentes ou pode ser utilizado após outras *queries*.

	Ex.
	`#ActiveRecord`
	`Person.select(:name, :age)`
	
	`#SQL Query`
	`SELECT name, age FROM persons`
	
	Selecionar apenas as colunas “name” e “age” da tabela “persons”

limit	Este método limita o número de resultados retornados por uma *query*.
	
	`#ActiveRecord`
	`Person.select(:name, :age).limit(3)`
	
	`#SQL Query`
	`SELECT name, age FROM persons LIMIT 3`
	
	Ex. Limitar o número de resultados para três

Tabela 3 - Métodos de queries em Ruby on Rails

Como referido anteriormente, o principal objetivo destes métodos revolve à volta da geração de resultados, provenientes de *queries* à base de dados, que permitam o uso de ***ActiveRecords***. **ActiveRecord** é um tipo de variável que pode ser trabalhado de maneira idêntica a um ***Object***. Através deste tipo de variável é possível aceder aos atributos de um dado **Modelo** que, por sua vez, dizem respeito a colunas de um registo de uma tabela na base de dados.

Considerando a tabela representada na *Figura 2*, através de uma variável ***ActiveRecord*** seria possível aceder aos atributos "Id", "Name" e "Age" como se estes se tratassem de atributos de uma variável do tipo *Object*. Isto pode ser verificado na figura seguinte:

```
5
6   def showcase_person
7
8       _person = Person.find_by(id: 1)
9
10      puts _person.id
11      puts _person.name
12      puts _person.age
13
14  end
15
```

Figura 2 – Exemplo da utilização de variáveis do tipo ActiveRecord

Os **Modelos** refletem a estrutura integral da base de dados no que tocas há suas tabelas, como tal é também possível especificar a relação existente entre M**odelos** diferentes. Estas relações são declaradas na classe representante do **Modelo.** Estas relações têm de ser especificadas em ambos os modelos relacionados, seguindo a respetiva perspetiva. Na tabela seguinte estão exemplificadas as possíveis relações entre **Modelos** declaradas em *Rails*.

Relação	**Descrição**

has_one	Representa uma relação de um para um (1-1) no que toca à
	`class Person`
	`    has_one :identification_card`
	`end`
	`class IdentificationCard`
	`    belongs_to :person`
	`end`
	estrutura de uma base de dados.

has_many	Representa um relação de um para muitos (1-n) no que toca à estrutura de uma base de dados.
	`class Person`
	`    has_many :dogs`
	`end`
	`class Dog`
	`    belongs_to :person`
	`end`

belongs_to	Representa o lado dependente da relação. Tanto é utilizado em relações de um para um (1-1) ou de um para muitos (1-n) mas sempre do lado
	`class Person`
	`    has_many :dogs`
	`end`
	`class Dog`
	`    belongs_to :person`
	`end`
	dependente.

has_and_belongs_to	Representa uma relação de muitos para muitos (n-n) no que toca à estrutura de uma base de dados.
	`class Person`
	`    has_and_belongs_to :first_name`
	`end`
	`class FirstName`
	`    has_and_belongs_to :person`
	`end`

Tabela 4 - Tipos de relações entre modelos em Rails

4.1.3. Rotas

As rotas dizem respeito a um serviço que escuta chamadas *http* a determinados *endpoints* especificados no desenvolvimento da aplicação. As rotas são geridas por uma componente nativa do *Rails* chamada ***Rails Router*.** Esta componente reconhece *URLs* e redireciona as suas chamadas para **Controladores** e **Métodos** específicos, de acordo com a respetiva rota. No código seguinte encontra-se exemplificada a declaração de uma rota.

```
namespace :routes do

    get 'persons/get_person_name'

end
```

Controlador

Método

Código 21 - Exemplo da declaração de rotas em Rails

Em termos de redireccionamento as rotas funcionam muito à base de coerência de sintaxe, ou seja, se possível o *Rails Routes* irá associar segmentos do *URL* da rota a controladores e métodos existentes. No exemplo mostrado acima, pode-se constatar uma rota que quando chamada irá ser redirecionada para o controlador responsável pelo modelo "Person" e mais especificamente para o método "get_person_name". Isto só é possível se o controlador se encontrar declarado e estruturado de acordo com as regras que especifiquei anteriormente. Caso eventualmente se deseje declarar uma rota que não siga estas diretivas, então esta terá de especificar o sítio específico para onde deve ser redirecionada (controlador e método).

```
19  namespace :routes do
20
21      get 'persons/get_person_name', to: 'persons#get_person_name'
22
23  end
```

Controlador

Método

Código 22 - Exemplo da declaração de uma rota com redirecionamento específico

Como qualquer chamada *http*, estas podem carregar parâmetros. Estes parâmetros são também passados pelas rotas do *Rails*. Após a rota redirecionar o pedido para o controlador e o seu método, os parâmetros passados na chamada podem ser acedidos através de um objetivo nativo do *Rails*, chamado *"params"* da seguinte forma:

```
6   class Person
7
8       def get_person_name
9
10          _id = params[:id]
11
12          _person = ::Person.find_by(id: _id)
13
14      end
15
16  end
```

5. Plataforma Phoenix

A Phoenix é a *framework* desenvolvida pela unidade de Desenvolvimento do Grupo *Compta* (Equipa *RD*) com o objetivo de suportar todos os desenvolvimentos de produtos próprios. É um acelerador de produtização de aplicações, contribuindo para a simplificação no desenvolvimento de produtos. É composta por um conjunto de bibliotecas, funções e métodos que simplificam, facilitam e mantêm o código limpo, organizado e reutilizável, dando estabilidade, fiabilidade e uniformização ao software e produtos desenvolvidos.

Esta plataforma, já em utilização, tem diversos componentes transversais que são reutilizados em todas as aplicações, entre os quais *APIs* de dados, janelas modais, gráficos e *widgets* etc. Estes componentes permitem a reutilização de código, facilitando o desenvolvimento do produto, quer este seja de pequenas ou grandes dimensões.

Foi utilizada a metodologia *AGILE*, organizada em processo *SCRUM*, que garante o desenvolvimento e comercialização de produtos uniformes e sustentáveis. A aplicação concentra em si um *product backlog* que representa o conjunto de funcionalidades que os produtos desenvolvidos devem apresentar. A Phoenix garante que os diversos desenvolvimentos, que recorrem a tecnologias de *HTML5*, *AJAX* (*Asynchronous JavaScript* and *XML*), *CSS3*, *JQuery*, *JSON* (*Javascript Object Notation*), *SOAP*, *Java*, *SQL*, *JasperReports*, entre outras, sejam reutilizáveis e garante a coerência de desenvolvimento e passagem a modo de produção dos produtos Grupo *Compta*. Permite, ainda, que a empresa se torne independente das plataformas onerosas de terceiros que vinha utilizando até agora para realizar as mesmas tarefas, estando preparada para comercialização a terceiros. Como a base é partilhada, as alterações são comuns a todas as aplicações. Para além disso, como a estratégia de produtização permite o desenvolvimento em camadas, esta torna mais simples e célere a entrega ao mercado de produtos próprios, facto comprovado pelos produtos já lançados, com base nesta *framework*. Esta capacidade afeta positivamente a estratégia de comercialização de novos produtos pela empresa, conferindo-lhe maior velocidade no lançamento, robustez no desenvolvimento e um conjunto de funcionalidades base, verticalizadas que rapidamente podem adicionar valor aos diversos segmentos de mercado que estão a ser abordados pela empresa, como o setor do Ambiente, Energia, Logística Pesada, Agricultura e Mar.

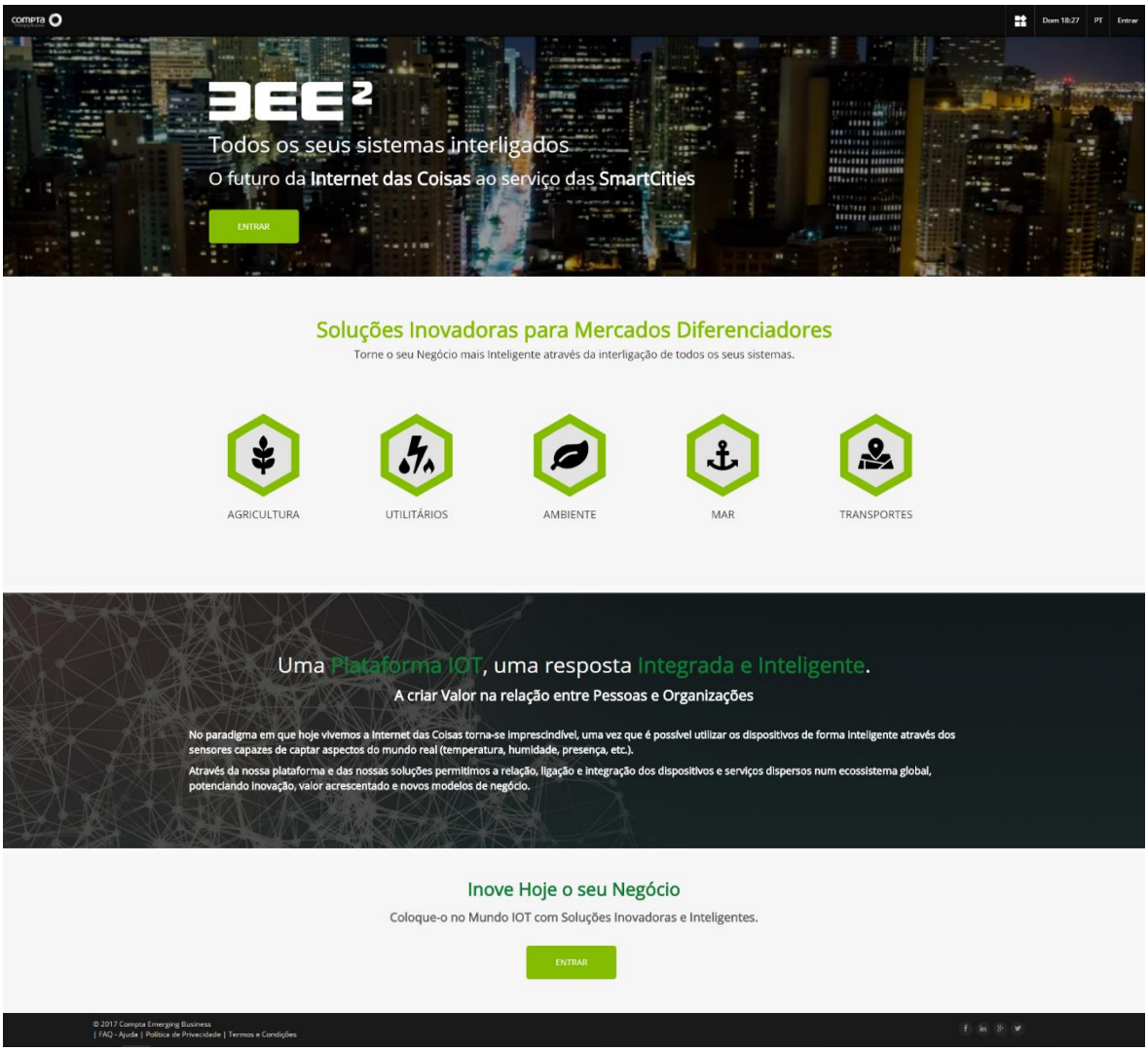


Figura 3 - Página Inicial Plataforma Phoenix

6. Gestão de Faturas

A **Gestão de Faturas** diz respeito a um módulo de uma aplicação existente na plataforma *Phoenix* respeitante à gestão e monitorização de consumos e gastos energéticos. Este módulo, por sua vez, também relacionado com gestão energética, possibilita também a importação e manuseamento de faturas de eletricidade a partir da plataforma. Esta ferramenta permite ao utilizador da plataforma visualizar com mais detalhe e facilidade todos os aspetos das suas faturas, bem como detalhar aspetos respeitantes aos seus pontos de instalação e até mesmo tirar conclusões relativas aos seus gastos e consumos. É neste módulo aplicacional que este relatório se vai focar no âmbito de desenvolvimento.

6.1. Faturas

Como já referi anteriormente este módulo aplicacional funciona através do *upload* de faturas de eletricidade para a plataforma. Estas faturas podem ser apresentadas em formato ***ZIP***, no então é também possível importar faturas em massa através da sua compactação e junção num único arquivo ***ZIP***.

Existem vários tipos de faturas sendo que cada tipo pode conter informações e campos diferentes. No entanto, a forma através da qual lhes é extraída a informação é equivalente para todos estes tipos. Estes tipos são:

- **A0** - Faturas de Média Tensão (MT)
- **B0 –** Faturas de Baixa Tensão Especial (BTE)
- **B1 –** Faturas de Baixa Tensão Normal (BTN)
- **C0 –** Faturas Iluminação Pública (IP)
- **D0 –** Juros de Pagamentos em Atraso
- **E0 –** Faturas *Business to Business (B2B)*
- **F0 –** Faturas *Universal Business Language (UBL)*

6.2. *Upload* e Importação de Faturas

Como já referi anteriormente é possível importar faturas para a plataforma de modo a poder usufruir das suas informações. Para tal é necessário fazer o *upload* de uma ou mais faturas num dos formatos suportados. O processo de importação pode ser dividido em quatro etapas, sendo estas as seguintes:

- *Upload*
- Validação de Sintaxe
- Validação de Erros
- Cálculo de Consumos

6.2.1. Upload

Esta etapa, tal como o nome indica, consiste no processo de envio do ficheiro ou pasta compactada do cliente para o servidor. Todo este processo pode ser separado em duas partes. A parte que diz respeito à seleção do ficheiro pretendido através do uso de janelas modais e do ***FileSystem*** nativo do sistema operativo que o utilizador possuir, trabalho realizado pela Aplicação Cliente. Por outro lado, a receção e armazenamento, do ficheiro, trabalho este realizado pelo servidor.

1) <u>Cliente</u>

O processo de *upload* tem início na página de registo da aplicação, respeitante à listagem e gestão das faturas importadas. Esta página tem o seguinte aspeto:

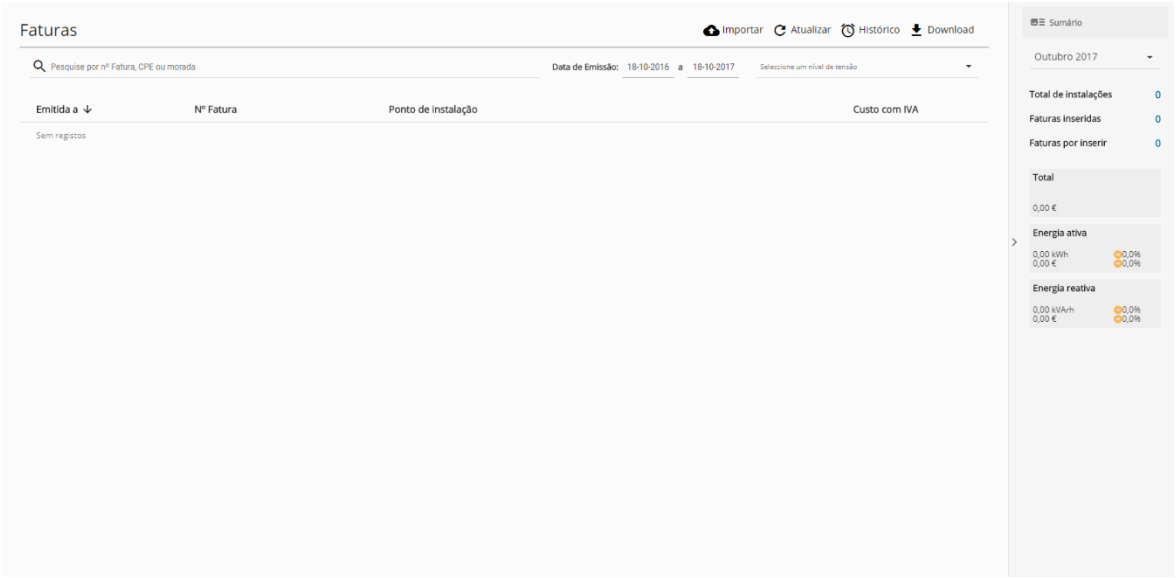


Figura 4 - Página de Listagem e Gestão de Faturas Importadas

Todos os elementos que compõem esta e todas as seguintes páginas demonstradas neste relatório, encontram-se exemplificados e explicados no manual de utilizador da plataforma que incluí como anexo no **Capítulo 5** deste relatório.

O processo de *upload* tem início ao premir o botão 

presente à direita do título da listagem. Ao premir este botão, será invocado o método de *Scope* seguinte:

```
$scope.onTableImport = function () {
    var
        _locals_data = {
            user: $scope.user,
            app_id: $state.current.data.appId,
            user_system_id: $stateParams.system_id
        };

    $scope.$emit('preloader', true);
    $mdDialog.show({
        locals: {
            localsData: _locals_data
        },
        templateUrl: 'modules/ezenergy/views/dialogs/import-invoices-file.dialog.tmpl.html',
        controller: 'EzenergyImportInvoicesFileDialogController',
        clickOutsideToClose: true,
        resolve: {},
        onComplete: function () {
            $scope.$emit('preloader', false);
        }
    }).then(function (answer) {
    }, function () {
        //Do nothing
    });

};
```

Código 23 - Método de Scope responsável pela inicialização da janela de upload de faturas

Este método é responsável pela inicialização da janela modal que, por sua vez, é responsável pela seleção da(s) faturas que o utilizador deseja carregar para a plataforma e, consequentemente, o seu envio para o servidor. Esta janela é bastante simples em termos de aspeto, apresentando apenas um campo de texto, um botão para a seleção de ficheiros e

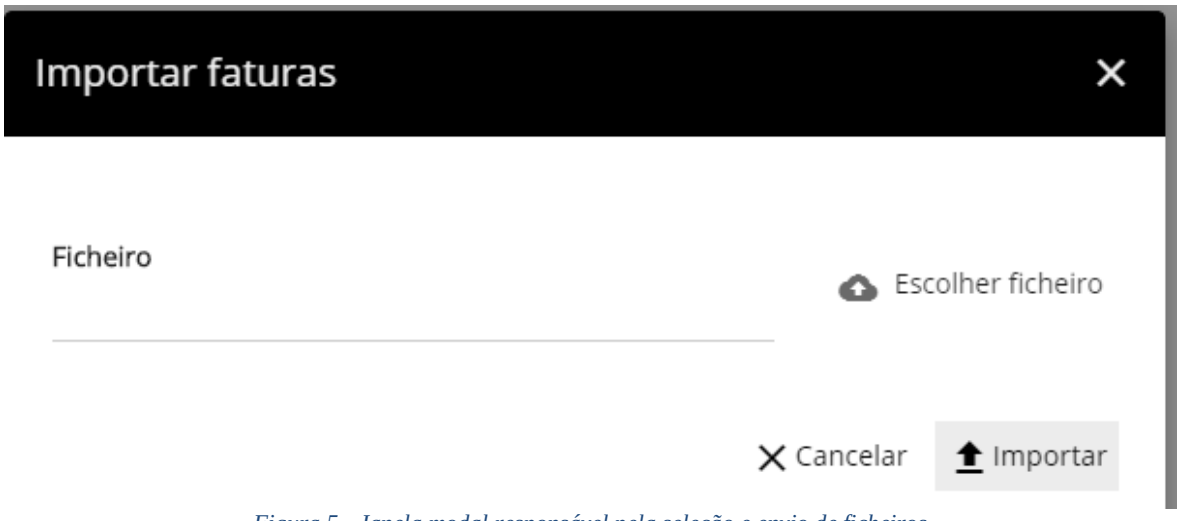


Figura 5 - Janela modal responsável pela seleção e envio de ficheiros

dois botões para cancelar ou gravar as ações feitas na mesma.

Para escolher o ficheiro a carregar, o utilizador deve clicar no botão Escolher ficheiro . Clicar neste botão irá fazer com que seja apresentada ao utilizador, uma janela de seleção de ficheiros nativa do Sistema Operativo que o utilizador possuir. Neste caso, dá-se como exemplo o

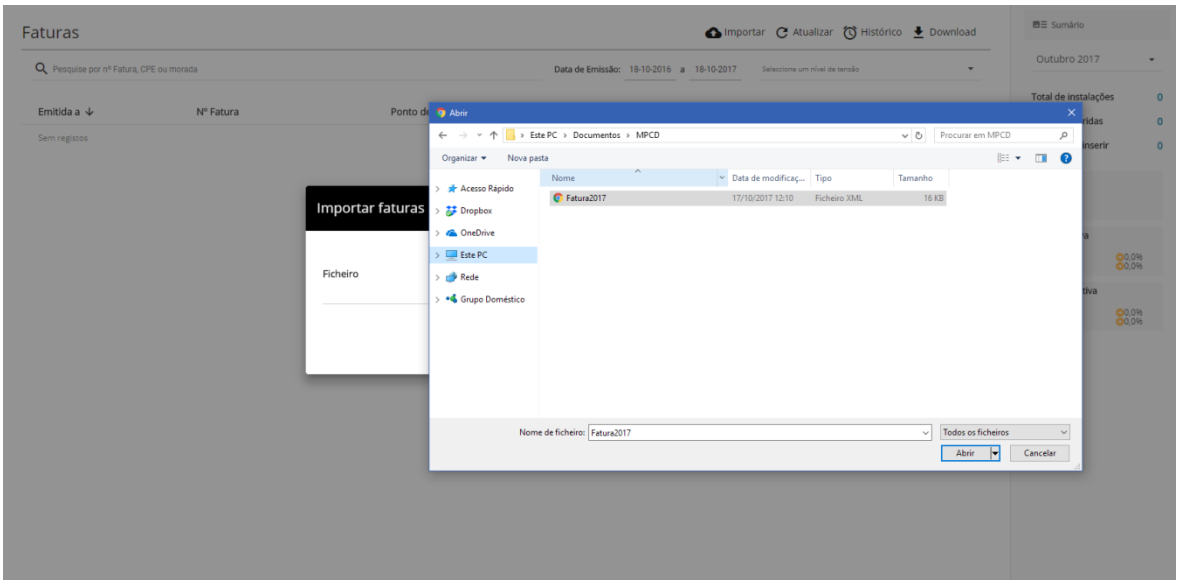


Figura 6 - Exemplo da janela de seleção de ficheiros Windows 10

Windows 10:

Esta operação é possível através da utilização de uma diretiva *third-party* chamada ***nv-file-select***. Esta diretiva permite não só a seleção de ficheiros através do Sistema Operativo, mas também disponibiliza ao programador acesso a vários detalhes do ficheiro, de entre os quais o seu tamanho e o seu formato. Tal permite disponibilizar informações apelativas ao utilizador e, mais importante, verificar se o formato do ficheiro é suportado pela plataforma.

Quando o ficheiro estiver pronto para ser carregado, o utilizador deve clicar no botão Importar, iniciando assim o processo de envio do ficheiro carregado, do cliente para o servidor onde este será armazenado e, posteriormente, processado. O envio deste ficheiro é feito através do ***endpoint*** "/api/ezenergy/invoice_uploads/upload_invoice" da *API* disponibilizada pelo servidor.

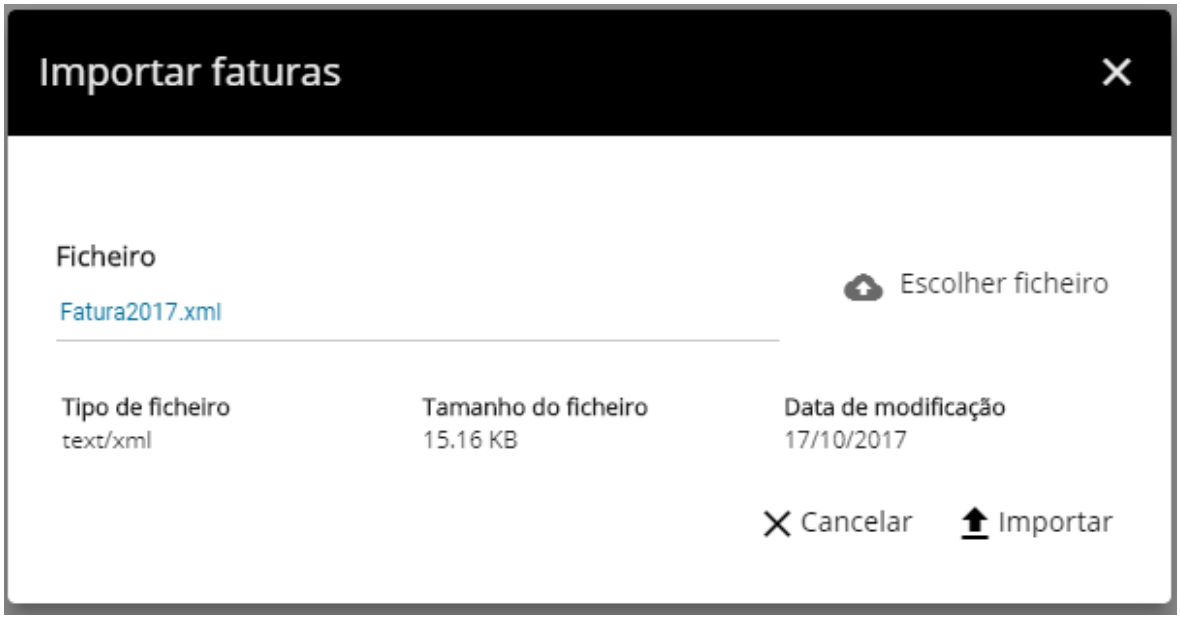


Figura 7 - Exemplo de um ficheiro pronto a ser carregado

2) Servidor

O trabalho do lado do servidor só começa aquando da receção do ficheiro carregado através do *endpoint* especificado anteriormente. Este *endpoint* é representado pela seguinte rota:

```
post 'invoice_uploads/upload_invoice'
```

Código 24 - Rota responsável pelo upload de faturas

Como referido na secção dedicada à *framework Rails* no **Capítulo 4,** esta rota aponta para o Controlador "*InvoiceUploadsController*", mais especificamente, para o seu método "*upload_invoice*". Este Controlador é responsável por todo o processo de *upload* de faturas, desde a etapa de carregamento, à etapa de cálculo de consumos. O método "upload_invoice" serve apenas como ponto de resposta da *API*. Este método é responsável por gerar uma resposta à chamada efetuada por parte do cliente, seja esta num estado de sucesso ou erro. Para tal, é necessário recorrer a uma classe responsável pelo armazenamento do ficheiro carregado.

```ruby
# STEP 1 (status 100 -> 101)
def upload_invoice

    # System params
    # :user_id
    # :user_system_id
    # :file
    _it = ::Ezenergy::UploadInvoices.call( context params: params)

    if _it.success?

        build_response( data file: _it.invoice_upload, message: t('messages.upload_success'))


    else
        build_response( data {
                            errors: (_it.errors || [_it.message]),
                            message: _it.message,
                            error_code: _it.error_code
                        },
                         success false,  code Api::RETURN_CODE_PARAM_MISSING,  message t('errors.messages.invalid'),  status 404)
    end

end
```

Código 25 - Método de resposta à chamada do endpoint

Esta classe, denominada "*Ezenergy::UploadInvoices*", quando invocada verifica se os argumentos apresentados pela chamada à *API* são aceitáveis e, de seguida, procede à criação de uma instância de um Modelo do tipo "*InvoiceUpload*" responsável pela inserção dos dados do ficheiro na tabela da base de dados respeitante a este modelo (*invoice_upload*).

```
def call()

    if !valid_params?
        context.fail!({message: I18n.t('errors.messages.invalid'), error_code: Api::RETURN_CODE_PARAM_MISSING })
    else

        @invoice_upload = ::Ezenergy::InvoiceUpload.new(
            file_path: params[:file],
            user_system_id: params[:user_system_id],
            invoice_file_type_id: params[:invoice_file_type_id])

        if is_zip? || is_xml?
            save()
        else
            context.fail!(
                message: I18n.t('errors.ezenergy.invoices.unknown_file_format'),
                invoice_upload_error_type:3,
                error_code: ::Ezenergy::RETURN_CODE_UNKNOWN_INVOICE
            )
        end

    end
end
```

Código 26 - Método de chamada da classe InvoiceUploads

Os argumentos aceites encontram-se definidos no método "*valid_params?*" da classe que pode ser visualizado no excerto de código seguinte:

```
def valid_params?
    # !(params.blank? || params[:file].blank? || params[:user_system_id].blank? || params[:user_id].blank?)
    # TODO
    !(params.blank? || params[:file].blank? || params[:user_system_id].blank? )
end
```

Código 27 - Método responsável pela definição dos parâmetros aceites pela API de upload de ficheiros

O método descrito procede a uma verificação de modo a descobrir se os parâmetros passados pela chamada à ***API*** possuem tanto um ficheiro `params[:file].blank?` como um id do sistema ao qual este *upload* deve ser associado `params[:user_system_id].blank?`

Caso estes parâmetros se encontrem presentes, será então criada a instância do modelo ***InvoiceUpload*** para dar entrada do registo na base de dados.

```
@invoice_upload = ::Ezenergy::InvoiceUpload.new(
    file_path: params[:file],
    user_system_id: params[:user_system_id],
    invoice_file_type_id: params[:invoice_file_type_id])
```

Código 28 - Criação da instância do modelo InvoiceUpload, método call, classe InvoiceUploads

Previamente à inserção dos novos valores na tabela da base de dados respeitante a este modelo, ocorre a verificação do formato do ficheiro carregado. Este ficheiro deve ser do formato ***ZIP*** ou do formato ***XML.*** Esta verificação é efetuada pelos métodos "*is_zip?*" e "*is_xml?*" respetivamente, através da extensão presente no nome do ficheiro carregado.

```
def is_zip?
    File.extname(@invoice_upload.file_path.path) == '.zip'
end

def is_xml?
    File.extname(@invoice_upload.file_path.path) == '.xml'
end
```

Código 29 - Métodos de verificação do formato do ficheiro carregado, classe InvoiceUploads

Finalmente, após esta verificação o método grava os conteúdos na base de dados, caso não tenham ocorrido exceções ou então retorna uma mensagem de erro. No caso da gravação esta é realizada por um outro método denominado "*save*".

```
def save

    _message = nil

    @invoice_upload.to_uploaded
    if @invoice_upload.save
        context.message ||= I18n.t('messages.create_successful')

        @invoice_upload.to_syntax_validation!

        context.invoice_upload = @invoice_upload
    else
        context.invoice_upload = @invoice_upload
        _message = context.invoice_upload.errors.full_messages.join(', ')

        context.fail!({errors: context.invoice_upload.errors,
                       message: (_message || I18n.t('errors.ezenergy.invoices.unable_to_save_invoice'))})

    end
end
```

Código 30 - Método de gravação de conteúdos na base de dados, classe InvoiceUploads

Este método, para além de ser responsável pela gravação de dados resultantes do processo de *upload* de ficheiros, é também responsável pela mudança de estado do *"InvoiceUpload"*. O Modelo "*InvoiceUpload*" pode encontrar-se nos seguintes estados:

- ***Uploaded*** *(101)* – O ***InvoiceUpload*** é dado como carregado para a plataforma
- ***Syntax Validation*** *(102)* – O ***InvoiceUpload*** encontra-se na fase de validação sintática
- ***Error Validation*** *(103)* – O ***InvoiceUpload*** encontra-se na fase de validação de erros
- ***Processed*** *(104)* – O ***InvoiceUpload*** encontra-se processado e pronto para o cálculo de consumos.
- ***Invalid*** – O ***InvoiceUpload*** é inválido e não poderá prosseguir o processo de importação.

Neste caso específico, o estado do *"InvoiceUpload"* criado irá ser definido como ***Uploaded*** aquando da sua criação e, caso os dados respeitantes ao ficheiro sejam guardados com sucesso, o seu estado será alterado novamente para ***Syntax Validation*.**

O armazenamento dos dados respeitantes ao InvoiceUpload criado são guardados através de um outro método, também denominado "*save*". A diferença entre este método e o outro especificado mais acima é que este representa um método nativo da *framework Rails* pertencente a qualquer modelo. No fundo o que este método faz é um *insert* de todos os seus atributos, como colunas da respetiva tabela na base de dados.

```
@invoice_upload.save
```

Código 31 - Método nativo do rails utilizado para a gravação de dados relativos à tabela invoice_uploads

Após todo este processo, é devolvida uma resposta de falha ou sucesso ao cliente, indicando a este o resultado da tentativa de *upload*. Esta resposta é gerada no método inicial "*upload_invoices*" através do método *build_response*.

```
if _it.success?
    build_response( data file: _it.invoice_upload, message: t('messages.upload_success'))

else
    build_response( data {
                        errors: (_it.errors || [_it.message]),
                        message: _it.message,
                        error_code: _it.error_code
                    },
                    success false, code Api::RETURN_CODE_PARAM_MISSING, message t('errors.messages.invalid'), status 404)
end
```

Código 32 - Excerto de código onde é gerada e enviada a resposta ao cliente

6.2.2. Validação de Sintaxe e Importação de Dados

O processo de validação de sintaxe consiste na verificação da coerência de campos na fatura carregado. Como as faturas se apresentam no formato ***XML*** é necessário verificar se todas as *tags* ***XML*** se encontram abertas e fechadas convenientemente de modo a não existirem erros de estrutura. Para além desta verificação, este processo é responsável também pela inserção de todos os valores, presentes na(s) fatura(s) importadas, nas tabelas da base de dados.

1) <u>Cliente – Pedido ao Servidor</u>

Após a conclusão do processo de *upload* do(s) ficheiro(s) selecionados, o servidor é responsável pelo envio de uma resposta ao cliente onde vem especificado o estado e resultado do *upload* efetuado. Ao receber esta resposta, a janela modal de *upload* irá redirecionar o utilizador para a página de estado da importação.

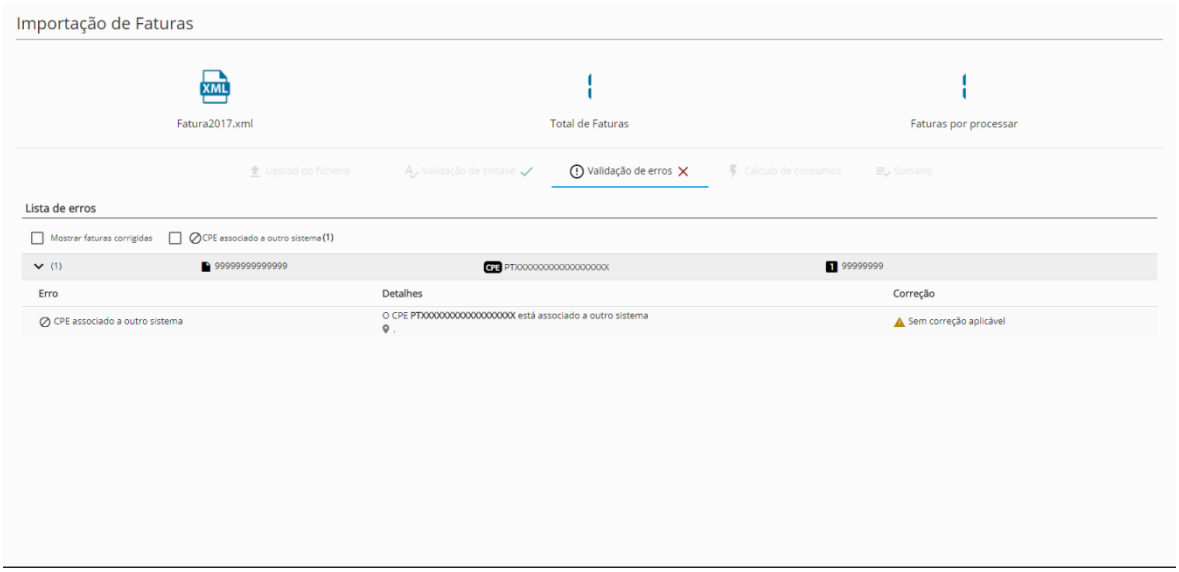


Figura 8 - Página de Estado da Importação

Após o *upload* de um ou mais ficheiros, esta será sempre a página de consulta de estado desse mesmo *upload*. Esta página pode-se dividir em três secções: **Detalhes do *Upload***, **Etapa da Importação** e **Detalhes da Etapa**.

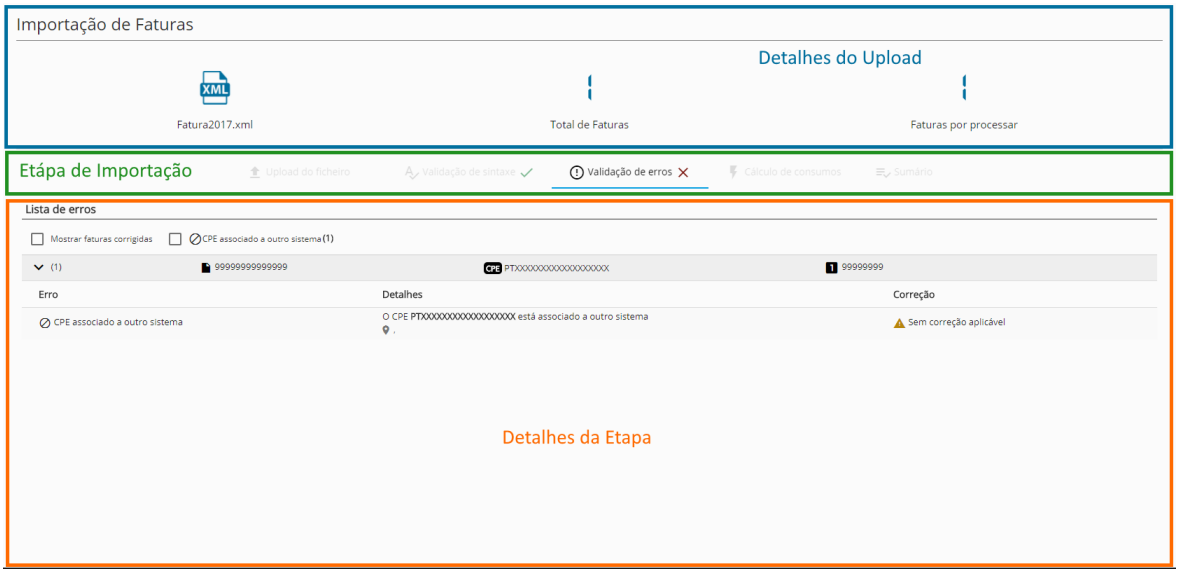


Figura 9 - Secções da página de Estado da Importação

- **Detalhes do *Upload*** – Local onde se encontram especificados os pormenores referentes ao(s) ficheiro(s) carregados para a plataforma deste *upload* em específico.
- **Etapa da Importação** – Secção composta por quatro separadores indicativos da etapa em que a importação encontra.
- **Detalhes da Etapa** – Secção composta por uma tabela onde se encontram visíveis todos os erros ou avisos respeitantes à etapa em que a importação se encontra.

Ao iniciar esta página, o cliente irá realizar um novo pedido ao servidor através do ***endpoint*** "*/api/ezenergy/invoice_uploads/upload_synthatic_validation*".

```
EzEnergyInvoiceUploadsService.uploadSyntaticValidation({
    id: $scope.uploadInfo.id,
    user_system_id: $stateParams.system_id
}, onUploadSyntaticValidation, onUploadSyntaticValidationError);
```

Código 33 - Chamada do cliente ao endpoint de validação de sintaxe

2) **Servidor**

O servidor começa o seu trabalho ao receber o pedido proveniente do cliente, no ***endpoint*** "*/api/ezenergy/invoice_uploads/upload_synthatic_validation*". Este ***endpoint*** é especificado pela rota seguinte:

```
post 'invoice_uploads/upload_syntatic_validation'
```

Código 34 - Rota responsável pela validação de sintaxe

Esta rota irá recorrer ao método "***upload_syntatic_validation***" do Controlador "**InvoiceUploadsController**".

```
# STEP 2 (status 101 -> 102)
def upload_syntatic_validation

    # System params
    # :user_id
    # :user_system_id
    # :file
    _invoice_upload = ::Ezenergy::InvoiceUpload.includes(:invoices).find(params[:id])

    _it = ::Ezenergy::ValidateUploadSyntaxAndStoreData.call( context invoice_upload_id: _invoice_upload.id,
                                                             params: {user_system_id: params[:user_system_id],
                                                            user_id: current_user.id})

    _data = serialize_invoice_upload( invoice_upload _it.invoice_upload)

    if _it.success?

        build_response( data file: _it.invoice_upload, invoice_upload: _data, message: t('messages.upload_success'))

    else


        build_response( data {
                            invoice_upload: _data,
                            errors: (_it.errors || [_it.message]),
                            message: _it.message,
                            error_code: _it.error_code,
                            invoice_upload_errors: _it.try(:invoice_upload_errors)
                        },
                         success false,  code Api::RETURN_CODE_PARAM_MISSING,  message t('errors.messages.invalid'),  status 404)
    end

end
```

Código 35 - Método upload_syntatic_validation do Controlador InvoiceUploadsController

O pedido feito pelo cliente ao servidor é feito com apenas um parâmetro, sendo este o *id* do *upload* a ser processado. Como tal, o método chamado pela rota é responsável por procurar a instância do modelo ***InvoiceUpload*** através da pesquisa pelo *id* que lhe foi passado.

```
_invoice_upload = ::Ezenergy::InvoiceUpload.includes(:invoices).find(params[:id])
```

Código 36 - Pesquisa através do campo id pela instância do Modelo InvoiceUpload

Após encontrar a instância do modelo pretendida, este método irá passar o id da mesma para uma outra classe. Esta classe, denominada "**ValidateUploadSyntaxAndStoreData**", em conjunto com outras, a referir no decorrer deste documento, é responsável por todo o processo de validação de sintaxe e também pelo armazenamento de todos os dados presentes nas faturas do *upload* em questão, na base de dados.

```
_it = ::Ezenergy::ValidateUploadSyntaxAndStoreData.call( context invoice_upload_id: _invoice_upload.id,
                                                         params: {user_system_id: params[:user_system_id],
                                                       user_id: current_user.id})
```

Código 37 - Chamada do método call da classe ValidateUploadSyntaxAndStoreData

Ao ser chamado, o método "***call***" da classe "**ValidateUploadSyntaxAndStoreData**" será responsável por algumas verificações a respeito do "***InvoiceUpload***" que lhe é passada.

```
def call()

    if !valid_params?
        context.fail!({message: I18n.t('errors.messages.invalid'), error_code: Api::RETURN_CODE_PARAM_MISSING})
    else

        context.invoice_upload = ::Ezenergy::InvoiceUpload.find( *ids context.invoice_upload_id) if context.invoice_upload.blank?
        @invoice_upload = context.invoice_upload

        invalid_file_response! if !@invoice_upload.syntax_validation?

        if is_zip?
            unzip_invoice && save()
        elsif is_xml?
            load_file( invoice_file_path @invoice_upload.file_path.path) && save()
        else
            unknown_file_format_response!
        end

        invalid_file_content_response! if @invoice_upload.invoice_upload_errors.size > 0

        @invoice_upload.to_content_validation!

    end

end
```

Código 38 - Método call da classe ValidateUploadSyntaxAndStoreData

A primeira parte deste método tem por objetivo verificar se os conteúdos que lhe foram passados são válidos, ou seja, neste caso se são existentes e não vêm a "*nil*" (*nil* é o equivalente ao *null* da linguagem *Javascript*, em *Ruby*). Esta verificação é feita através da chamada do método "*valid_params*". Caso este método retorne um valor falso, será enviada uma resposta de volta ao método inicial com uma mensagem de erro.

```
if !valid_params?
    context.fail!({message: I18n.t('errors.messages.invalid'), error_code: Api::RETURN_CODE_PARAM_MISSING})
else
```

```
def valid_params?
    context.invoice_upload.present? || context.invoice_upload_id.present?
end
```

Código 39 – Chamada e declaração do método de verificação de parâmetros

A segunda parte deste método tem por objetivo verificar se o "***InvoiceUpload***" de contexto se encontra realmente na etapa de "Validação de Sintaxe". Caso este não se encontre nesta etapa, não faz sentido realizar a validação logo, é retornado ao método inicial uma resposta de erro e este "***InvoiceUpload***" é dado como inválido.

```
def invalid_file_response!
    @invoice_upload.to_invalid!
    context.fail!({message: I18n.t('errors.messages.invalid'),
                   invoice_errors: [I18n.t('errors.messages.invalid')],
                   :error_code => ::Api::RETURN_CODE_FORBIDDEN_URL})
end
```

Código 40 - Método de resposta de erro de upload inválido

Após estas duas partes do método, é então feita a verificação do formato em que os ficheiros correspondentes a este "***InvoiceUpload***" se encontram. Para tal são usados dois métodos: "*is_zip?*" e "*is_xml?*". Ambos estes métodos já foram usados na primeira etapa da importação de faturas e, como tal, já foram explicados na secção anterior.

Desta verificação podem resultar três situações:

- Os ficheiros são do formato ***ZIP***
- Os ficheiros são do formato ***XML***
- Os ficheiros são de outro formato não aceite pela aplicação (Formato Desconhecido).

a) Ficheiros de Formato Desconhecido

Caso os ficheiros pertencentes ao "***InvoiceUpload***" sejam de formato não válido ou desconhecido pela aplicação, será retornada uma resposta de erro de formato de ficheiro para o método inicial. Esta mensagem de erro é gerada pelo método "*unknown_file_format_response!*". Neste método o "***InvoiceUpload***" é também definido como inválido.

```
def unknown_file_format_response!
    @invoice_upload.to_invalid!
    context.fail!(errors:[{message: I18n.t('errors.ezenergy.invoices.unknown_file_format'),
                    errors: I18n.t('errors.ezenergy.invoices.unknown_file_format'),
                   invoice_upload_error_type_id: 3,
                   error_code: ::Ezenergy::RETURN_CODE_UNKNOWN_INVOICE}])
end
```

Código 41 - Método de mensagem de erro respeitantes ao formato de ficheiros

b) **Ficheiros em formato *XML***

Os ficheiros que se encontram em formato ***XML*** estão prontos para serem processados,

```
def load_file(invoice_file_path)

    @user_system_id = params[:user_system_id]
    @user_id = params[:user_id]
    @creator = User.find( *ids @user_id)

    _invoice_processor = ::Ezenergy::Invoice::InvoiceParser.call( context file_path: invoice_file_path,
                                                                  user_system_id: @user_system_id,
                                                                  user_id: @user_id)
    # TODO delete  invoice_upload: @invoice_upload,
    if _invoice_processor.failure?

        if context.invoice_upload.invoice_upload_errors.blank? && _invoice_processor.errors.present?
            (_invoice_processor.errors || []).uniq.each do |error|
                context.invoice_upload.invoice_upload_errors << ::Ezenergy::InvoiceUploadError.new(
                    invoice_upload_error_type_id: error[:invoice_upload_error_type_id],
                    description: error[:errors]
                )
            end

            #context.invoice_upload.without_versioning :save

        end
        context.fail!({
                          errors: _invoice_processor.errors,
                          message: (_invoice_processor.message),
                          error_code: _invoice_processor.error_code
                      })
    else
        @invoice_upload.invoices += _invoice_processor.response

        context.message = _invoice_processor.message
    end

end
```

Código 42 - Método load_file da classe ValidateUploadSyntaxAndStoreData

como tal, o seu *URL* é passado para o método "*load_file*".

O método "*load_file*" tem como objetivo a criação de uma instância do "***InvoiceParser***". Esta classe é a responsável por determinar qual o tipo de fatura presente no ficheiro a ser processa e pela seleção do processador indicado para esse mesmo tipo. De modo a poder selecionar qual o processador a executar para a fatura atual, é necessário primeiro verificar o tipo de fatura. Esta verificação é feita através dos *headers* e *subheaders* do ficheiro ***XML***. Estes *headers* e *subheaders* são definidos no seguinte bloco de código:

```
#Xml Structure
_xml_type_a0 = ['INI', 'FMT000', 'CAB000'].freeze
_xml_type_b0 = ['TOP', 'FC0000', 'BTEORC'].freeze
_xml_type_b1 = ['TOP', 'FC0000', 'BTNORC'].freeze
_xml_type_c0 = ['INI', 'FCT001', 'CAB000'].freeze
_xml_type_d0 = ['TOP', 'FC0000', 'NDEORC'].freeze
_xml_type_e0 = ['TOP', 'FC0000', 'B2BINI'].freeze
_xml_type_f0 = ['', 'Invoice', ''].freeze
```

Código 43 - Declaração dos headers e subheaders dos ficheiros XML para cada tipo de fatura

A leitura de ficheiros ***XML*** é executada através do uso de uma *gem* para o *Rails* chamada "*Nokogiri*". Esta *gem* permite ler ficheiros ***XML***, verificar se estes se encontram mal formatados, procurar *tags* de ***XML*** específicas, etc. A utilização desta *gem* é bastante simples sendo que a sua inicialização é feita no seguinte bloco de código:

```
_invoice_file_xml = Nokogiri::XML(_invoice_file) do |config|
    config.strict; config.noblanks
end
```

Código 44 - Inicialização da gem Nokogiri para leitura e processamento de ficheiros XML

Ao inicializar a *gem* passando-lhe o caminho para o ficheiro ***XML*** que se pretende processar, podem também ser definidas algumas configurações especiais. Neste caso foram definidas duas: *strict* e *noblanks*.

- ***Strict* –** Esta configuração da *gem* ***Nokogiri*** permite que esta emita uma mensagem de erro à primeira ocorrência de um ficheiro ***XML*** mal formatado. Por exemplo, caso existam *tags* por fechar.
- ***Noblanks* –** Esta configuração da *gem* ***Nokogiri*** permite que esta ignore todas as *tags* que se encontrem vazias, ou seja, que não contenham valores.

Com estas duas configurações definidas à priori, é possível descartar imediatamente faturas ou atributos de faturas que podem ser considerados como "lixo", ou seja, que não apresentam qualquer interesse para a importação.

Após a declaração e inicialização desta *gem* é então possível começar a diferenciação entre os vários tipos de faturas. Como já referi anteriormente, isto é conseguido através da verificação dos cabeçalhos das mesmas. Para tal a *gem* ***Nokogiri*** possui um método de procure que permite localizar e devolver (caso existam) todas as ocorrências das *tags* ***XML*** que lhe são passadas. Este processo encontra-se implementado no seguinte bloco de

```
_invoice_file_xml.search(''+_xml_type_a0[1]+
                         ','+_xml_type_b0[1]+','+
                         _xml_type_b1[1]+','+
                         _xml_type_e0[1]+ ',' +
                         _xml_type_f0[1] +','+
                         _xml_type_c0[1]+'').each do |single_xml_invoice|
```

Código 45 - Execução de uma pesquisa pelos cabeçalhos responsáveis pela diferenciação entre os vários tipos de faturas

código:

É de notar que faturas de tipos diferentes podem iniciar-se com o mesmo *header*, daí ser também necessário verificar o *subheader*. Sirva de exemplo uma fatura do tipo *B0 e E0*; esta teria um início semelhante ao seguinte:

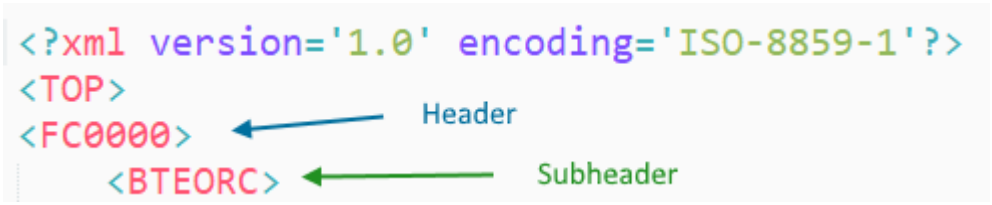


Figura 10 - Exemplo de cabeçalho respeitante a uma fatura do tipo B0

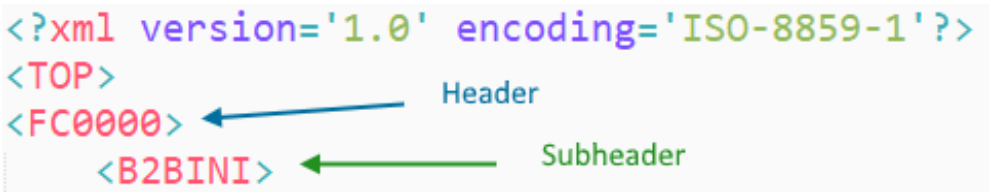


Figura 11 - Exemplo de cabeçalho respeitante a uma fatura do tipo E0

Ambas as faturas exemplificadas nas figuras acima se apresentam com o mesmo *header*, no entanto, os seus *subheaders* são diferentes. É também de notar que a primeira *tag* é ignorada na diferenciação das faturas. Isto deve-se ao facto de simplesmente não valer o esforço verificar esta primeira *tag* pois, tal como o *header*, também esta se encontra

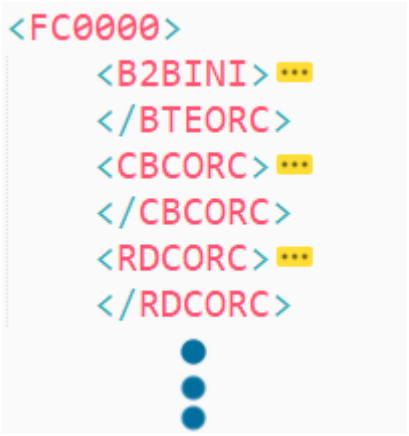
presente em mais do que um tipo de fatura. Logo, para fazer a diferenciação basta comparar apenas a segunda e terceira *tag* do ***XML***.

```
if (single_xml_invoice.name == _xml_type_b0[1] && !single_xml_invoice.css(_xml_type_b0[2]).empty?)
 Rails.logger.info 'XML-Type-B0'


 _xml_type_b_processor = ::Ezenergy::Invoice::Xml::TypeBProcessor.call( context xml_file: single_xml_invoice,
                                                                        user_system_id: context.user_system_id || '0',
                                                                        invoice_type_id: INVOICE_TYPE_B0)

 if _xml_type_b_processor.failure?

    _errors += _xml_type_b_processor.errors

 else

    _invoices_result = _invoices_result + _xml_type_b_processor.response

 end
```

Código 46 - bloco de código responsável pela verificação do tipo de fatura

A nível de código, estas verificações são feitas da seguinte forma:

Sabemos que a variável "*single_xml_invoice*" representa o nó ***XML*** resultante da pesquisa feita pelos *headers* dos vários tipos de faturas, bem como todos os nós seus filhos. Por outras palavras, se eu pesquisar pelo *header* "FC0000" e este existir no ficheiro, ser-me-á retornada uma variável instância da *gem* ***Nokogiri*** que representa um ***XML*** com uma possível estrutura semelhante à da figura seguinte:



Código 47 - Exemplo do XML resultante após uma possível pesquisa pelo header "FC0000"

As variáveis resultantes da pesquisa feita pela *gem* ***Nokogiri*** recebem também algumas propriedades específicas. Uma dessas propriedades (a qual se encontra a ser usada no excerto de código acima) é a propriedade "*name*". Esta propriedade, quando chamada, retorna uma *String* com o nome da *tag* pela qual a pesquisa foi feita. Ou seja, caso a pesquisa tenha sido feita pela *tag* "FC0000", então a propriedade "*name*" vai tomar o valor de *"FC0000"*.

Uma outra propriedade destas variáveis encontra-se sob a forma de um método chamado "*css*". Este método permite a execução de *queries CSS* com o propósito de encontrar ocorrências de uma dada *String* como nome de uma ou mais *tags*.

No caso do excerto de código mostrado acima, a verificação será feita para faturas do tipo *B0*. Por extenso, se a propriedade "*name*" possuir o valor de "*FC0000*" e o método *"css"* retornar ocorrências de *tags* cujo nome seja "*BTEORC*" então o presente tipo é *B0* ou **Fatura de Baixa Tensão Especial**.

Após feita esta verificação, é então chamada o processador indicado para o tipo de fatura em questão onde, finalmente, serão extraídos os dados presentes na fatura.

```
_xml_type_b_processor = ::Ezenergy::Invoice::Xml::TypeBProcessor.call( context xml_file: single_xml_invoice,
                                                                       user_system_id: context.user_system_id || '0',
                                                                       invoice_type_id: INVOICE_TYPE_B0)
```

Código 48 - Chamada do processador responsável por faturas do grupo B

Como referido anteriormente, o processo de processamento e tratamento de dados presentes nas faturas, segue uma lógica única para todos os tipos de faturas. A única diferença entre os diferentes processadores de faturas tem a ver com os nomes das *tags* ***XML*** usadas para pesquisa e extração de dados. No final do presente documento será detalhado e explicado o processador das faturas do tipo *B0 (Faturas Baixa Tensão Especial)*.

Antes de avançar com os procedimentos efetuados dentro da classe do processador, neste caso, a classe "*TypeBProcessor*" é necessário primeiro referir e abordar uma outra classe. A classe "*XmlProcessorHelper*" tem como objetivo a junção de todos os métodos responsáveis pela extração e tratamento de dados contidos no ficheiro ***XML***. O propósito e funcionamento destes métodos encontram-se descritos de seguida.

- ***valid_invoice?*** **–** Método cujo o objetive se baseia em verificar se a fatura a ser processado é válida. Este processo consiste na verificação das *tags* que constituem o ***XML*** da fatura de modo a excluir as *tags* que devem de ser ignoradas nesta verificação e, após essa exclusão, verificar se ainda existem *tags* na fatura. Caso ainda existam *tags* após a exclusão, então significa que a fatura é válida e possui conteúdos de interesse à importação. Caso contrário a fatura pode ser descartada.

```
def valid_invoice?(invoice_xml, invoice_type, invoice_tags, invoice_number, document_number, ignore_nodes = [])

    _fields = []

    invoice_tags[invoice_type].each do |key, invoice_fields|
        invoice_fields.each do |field|
            _to_ignore = false
            ignore_nodes.each do |ignore_node|
                if ignore_node == field

                    _to_ignore = true

                end
            end

            if invoice_xml.css(field).empty? && !_to_ignore

                _fields << field

            end
        end
    end

    if _fields.length > 0
        _inv_number = get_invoice_number( inv invoice_xml, invoice_type, invoice_tags, invoice_number, document_number)

        _fields.each do |f|
            @errors << {message: I18n.t('errors.ezenergy.invoices.unable_to_process_invoice'),
                        errors: I18n.t('errors.ezenergy.invoices.missing_field_invoice',
                                       node: f, invoice_number: _inv_number), invoice_upload_error_type_id:1,
                        error_code: ::Ezenergy::RETURN_CODE_MISSING_FIELD}
        end

        return false #Is not Valid

    else

        return true #Is Valid

    end

end
```

Código 49 - Método valid_invoice da classe XmlProcessorHelper

- ***extract_attribute*** – Método responsável pela extração de informações de um nó específico. Para o bom funcionamento deste método, é necessário que lhe sejam passados três parâmetros. O primeiro parâmetro diz respeito ao ***XML*** completo o qual será usado como base de dados aquando da pesquisa por *tags*. O segundo método diz respeito a um possível "caminho" predefinido de *tags*. Este "caminho" é utilizado para chegar a *tags* que se encontrem em níveis inferiores no ***XML***. Por fim, o último parâmetro diz respeito ao nome da *tag* que se pretende encontrar e da qual se pretende extrair a informação.

```ruby
def extract_attribute(xml, xml_path, node_with_value)
    if xml_path.nil?
        return xml.css(node_with_value).map(&:text)[0]
    else
        _value = xml
        if xml_path.include? '/'
            xml_path.split('/').each do |parent_node|
                _value = _value.css(parent_node)
            end
            _value = _value.css(node_with_value).map(&:text)[0]

        else
            _value = xml.css(xml_path).css(node_with_value).map(&:text)[0]
        end
        return _value
    end
end
```

Código 50 - Método extract_attribute da classe XMLProcessorHelper

- ***parse_nif*** – Este método, tal como o nome indica, permite filtrar um dado conteúdo que lhe é passado por parâmetros, de modo a extrair um valor de NIF válido. Este processo é feito através de uma *Regular Expression*.

```
def parse_nif(nif, nif_regex)
    if !nif.blank? && !nif.nil?
        if nif =~ nif_regex[0]
            return $1
        else
            return nil
        end
    end
end
```

Código 51 - Método parse_nif da classe XMLProcessorHelper

- ***parse_date*** **–** Este método é responsável pela transformação de uma data num dado formato, para outro pré-definido na aplicação. O formato para o qual as datas são convertidas é o seguinte: "*xx-mm-yyyy*"

```
def parse_date(dt)
    dt.blank? ? '' : Date.parse(dt).strftime(I18n.t('date.formats.default'))
end
```

Código 52 - Método parse_date da classe XMLProcessorHelper

- ***split_date_and_parse*** **–** Este método tem como objetivo a formatação de datas num formato específico. Existem várias ocorrências de datas nas faturas compostas por um inicio e um fim, no entanto em base de dados, não é viável guardar períodos de tempo numa só coluna. Como tal, é necessário separar a data de inicio da data de fim, de modo a poder gravá-las separadamente. O formato em que estes períodos de tempo se encontram é o seguinte: "*xx-mm-yyyy* a *xx-mm-yyyy*". Este método recebe um parâmetro adicional, de modo a decidir qual das duas datas este deve retornar (inicio ou fim). Em termos de formatação da data depois de separada, este método utiliza o método de formatação referido no ponto anterior.

```
# Expected string (date) -> "yyyy-mm-dd a yyyy-mm-dd"
def split_date_and_parse(date, is_launch)
    if !date.blank? && !date.nil?
        if date.include? ' a '
            if is_launch
                return parse_date( dt date.split(' a ')[0])
            else
                return parse_date( dt date.split(' a ')[1])
            end
        else
            return nil
        end
    else
        return nil
    end
end
```

Código 53 - Método split_date_and_parse da classe XMLProcessorHelper

- ***which_currency?*** **–** Este método é utilizado para definir qual a moeda utilizado num dado campo da fatura. Esta decisão é tomada com base em *Regular Expressions*. No fundo o método vai correr todas as *Regular Expressions* definidas na inicialização no processador e vai verificar se alguma corresponde com o conteúdo do campo da fatura a ser verificado. Este método é necessário pois o que se pretende armazenar na base de dados é uma referência (id) à entrada numa tabela que diga respeito à moeda utilizada na fatura. As *Regular Expressions* estão definidas juntamente com o *id* registado na tabela "currencies", de modo a que o método "*which_currency"* possa ter acesso a tal *id* para posteriormente o retornar como resultado da verificação.

```
#-------------------------------------------------------------
# CURRENCY

# 1st position is the currency_id
# 2nd position is array of regex's to that currency
#[ id, [regex's] ]

CURRENCY_EURO = [1, [/^.*EUR.*$/]].freeze

CURRENCIES = [CURRENCY_EURO]
```

Código 54 - Definição das Regular Expressions para as possíveis moedas

```
def which_currency?(str, currencies)
    if !str.blank? and !str.nil?
        currencies.each do |curren|
            curren[1].each do |regex|
                if str =~ regex
                    return curren[0]
                end
            end
        end
        return nil
    else
        return nil
    end
end
```

Código 55 - Método which_currency? da class XMLProcessorHelper

- ***which_component_is?*** – Este método permite decidir qual dos componentes de uma fatura está a ser usado no campo atual. Este método funciona de forma semelhante ao método explicado no ponto anterior, pelo que recorre também ao uso de *Regular Expressions*. As *Regular Expressions* definidas para este método também são compostas de uma referência à tabela "*invoice_components*" e, claro, a *Regular Expression* em si.

```
#------------------------------------------------------------
# INVOICE_COMPONENTS

# 1st position is the component_id
# 2nd position is the array of regex's to that component
#[ id, [regex's] ]

ACTIVE_ENERGY = [1, [/^.*Energia\sActiva.*$/]].freeze
REACTIVE_ENERGY = [2, [/^.*Energia\sReactiva.*$/]].freeze
AUDIOVISUAL = [3, [/^.*Contribui.*udio-Visual.*$/]].freeze
ELECTRIC_TAX = [4, [/^.*Imposto\ssobre\sConsumo\sElectricidade.*$/]].freeze
PEAK_HOUR_POWER_NETWORK = [5, []].freeze
CONTRACTED_POWER_NETWORK = [6, []].freeze
POWER_NETWORKS = [7, [/^Redes\sPot.?ncia$/]].freeze

INVOICE_COMPONENTS = [ACTIVE_ENERGY,
                      REACTIVE_ENERGY,
                      AUDIOVISUAL,
                      ELECTRIC_TAX,
                      PEAK_HOUR_POWER_NETWORK,
                      CONTRACTED_POWER_NETWORK,
                      POWER_NETWORKS
]
```

Código 56 - Definição das Regular Expressions para os diversos componentes de uma fatura

```
def which_component_is?(str, components)
    if !str.blank? and !str.nil?
        components.each do |inv_comp|
            inv_comp[1].each do |regex|
                if str =~ regex
                    return inv_comp[0]
                end
            end
        end
        return nil
    else
        return nil
    end
```

Código 57 - Método which_currency? da classe XMLProcessorHelper

- ***which_daily_period?*** – Método responsável pela verificação de qual o período diário ao qual um campo da fatura está associado. Tal como os dois métodos anteriores, este também se baseia no uso de *Regular Expressions* associadas a um *id*, neste caso da tabela "*invoice_daily_periods*", de modo a conseguir estabelecer uma correspondência.

```
#-------------------------------------------------------------
# INVOICE_DAILY_PERIODS

# 1st position is the daily_period_id
# 2nd position is array of regex's to that daily_period
#[ id, [regex's] ]

DAILY_PERIOD_PEAK = [1, [/^.*Ponta\s\(P\).*$/, /^.*Ponta.*$/]].freeze #Ponta
DAILY_PERIOD_SHOULDER = [2, [/^.*Cheia\s\(C\).*$/, /^.*Cheia.*$/]].freeze #Cheio
DAILY_PERIOD_OFF_PEAK = [3, [/^.*Vazio\sNormal\s\(Vn\).*$/,
                             /^.*Vazio\sNormal.*$/,
                             /^.*vazio\s\(Vz\).*$/,
                             /^Vazio\s-.*$/,
                             /^Redes\sVazio.*$/,
                             /^.*Activa\sVazio$/]].freeze #Vazio
DAILY_PERIOD_SUPER_OFF_PEAK = [4, [/^.*Super\sVazio\s\(SV\).*$/, /^.*Super\sVazio.*$/]].freeze #Super Vazio
DAILY_PERIOD_OUT_OFF_PEAK = [5, [/^.*\.cons\.FV$/]].freeze #Fora de Vazio

INVOICE_DAILY_PERIODS = [DAILY_PERIOD_PEAK,
                         DAILY_PERIOD_SHOULDER,
                         DAILY_PERIOD_OFF_PEAK,
                         DAILY_PERIOD_SUPER_OFF_PEAK,
                         DAILY_PERIOD_OUT_OFF_PEAK
]
```

Código 58 - Definição das Regular Expressions para os períodos diários de uma fatura

```
def which_daily_period?(str, daily_periods)
    if !str.blank? and !str.nil?
        daily_periods.each do |inv_daily_p|
            inv_daily_p[1].each do |regex|
                if str =~ regex
                    return inv_daily_p[0]
                end
            end
        end
        return nil
    else
        return nil
    end
end
```

Código 59 - Método which_daily_period da classe XMLProcessorHelper

- ***which_tension_is?*** – Este método tem como objetivo a verificação de qual o nível de tensão ao qual dado campo da fatura se refere. Mais uma vez, também este método se encontra na mesma categoria dos três métodos anteriores. Utiliza *Regular Expressions* associadas a *ids* de entradas na tabela "*tension_levels*".

```
-
#------------------------------------------------------------
# TENSION_LEVELS

# 1st position is the tension_level_id
# 2nd position is array of regex's to that tension_level
#[ id, [regex's] ]

TENSION_LEVEL_BTE = [1, [/^BTE/]].freeze
TENSION_LEVEL_BTN = [2, [/^BTN$/]].freeze
TENSION_LEVEL_MT = [3, [/^MT$/]].freeze

TENSION_LEVELS = [TENSION_LEVEL_BTE,
                  TENSION_LEVEL_BTN,
                  TENSION_LEVEL_MT
]
#------------------------------------------------------------
```

Código 60 - Definição das Regular Expressions para os possíveis níveis de tensão de uma fatura

```
def which_tension_is(str, tension_levels)
    if !str.blank? and !str.nil?
        tension_levels.each do |t_l|
            t_l[1].each do |regex|
                if str =~ regex
                    return t_l[0]
                end
            end
        end
        return nil
    else
        return nil
    end
end
```

Código 61 - Método which_tension_is? da classe XMLProcessorHelper

- ***extract_vat_value*** **–** Este método permite verificar e retornar um valor de NIF caso este seja válido. Este processo é feito através da comparação de um valor (passado como parâmetro) a uma *Regular Expression*.

```
#REGEX
EXTRACT_VAT_VALUE_REGEX=[/^.*?(\d*?)%.*?$/].freeze
```

Código 62 - Definição de Regular Expression para um NIF

```
def extract_vat_value(str, extract_vat_regex)
    if !str.blank? && !str.nil?
        if str =~ extract_vat_regex[0]
            return $1
        end
    end
end
```

Código 63 - Método extract_vat_value da classe XMLProcessorHelper

Tal como referido anteriormente, os métodos descritos anteriormente funcionam como auxiliares aos processadores de faturas são usados no processamento de dados provenientes das mesmas.

Os processadores de faturas podem ser divididos em duas secções. A primeira secção diz respeito à especificação de campos presentes numa fatura de dado tipo e à especificação de *Regular Expressions* utilizadas nos métodos auxiliares de modo a efetuar a extração e armazenamento de dados. A Segunda secção diz respeito aos métodos de construção dos detalhes da fatura, para posterior armazenamento.

A declaração dos campos das faturas é realizada sob a forma de *Arrays* de *Strings* ou Objetos de moda a que a sua utilização seja mais fácil. No caso de *Arrays* com apenas *Strings*, estes são usados para obter dados de uma forma mais direta, como por exemplo Localizar Campo e Retirar Informação.

```
INVOICE_TAGS = {
    'B0' => {
        'header' => ['CBCORC', 'BTEORC', 'BTEG02', 'BTEMSG'],
        'details' => ['BTEDFI', 'BTEDF0', 'BTEDFF'],
        'vat' => ['BTETTF', 'BTEDGE'],
        'consumption' => ['BTEQCD'],
    },
    'B1' => {
        'header' => ['CBCORC', 'BTNORC', 'BTNG02', 'BTNMSG'],
        'details' => ['BTNDFI', 'BTNDF0', 'BTNDFF'],
        'vat' => ['BTNTTF', 'BTNDGE'],
        'consumption' => ['BTNQCD'],
    }
}.freeze
```

Código 64 - Exemplo da declaração dos campos recorrendo apenas a Strings (Recolha de informação direta)

No caso de *Arrays* compostos por objetos, estes são usados também para associar determinadas informações a entradas já existentes em tabelas da base de dados.

```
TENSION_LEVEL_BTE = [1, [/^BTE/]].freeze
TENSION_LEVEL_BTN = [2, [/^BTN$/]].freeze
TENSION_LEVEL_MT = [3, [/^MT$/]].freeze
```

Código 65 - Exemplo da declaração dos campos recorrendo a objetos (Recolha de Informação e associação a entradas já existentes na BD)

Quando o processador é chamado, através do método "*call*", este irá executar duas funções principais. A primeira função consiste na verificação da validade da fatura mediante o método "*valid_invoice?*" da classe "*XMLProcessorHelper*". A segunda função consiste na construção do objeto correspondente à fatura importada. Este objeto terá vários atributos, sendo que cada um deles corresponderá a uma propriedade da fatura importada. Atendendo a que este objeto será utilizado mais tarde para armazenar os dados na base de dados através do modelo "*Invoice*", cada um dos seus atributos é nomeado de acordo com a coluna correspondente da tabela "*invoices*".

O método "*call*" do processador pode ser visualizado no excerto de código seguinte:

```
def call

    _xml_obj = context.xml_file
    _invoice_type_id = context.invoice_type_id
    _invoice_type = invoice_type_id_to_invoice_type( invoice_type_id _invoice_type_id)
    _invoices = []

    @errors = []

    if valid_invoice?( invoice_xml _xml_obj,
                       invoice_type _invoice_type,
                       invoice_tags INVOICE_TAGS,
                       invoice_number INVOICE_NUMBER,
                       document_number DOCUMENT_NUMBER,
                       ignore_nodes IGNORE_VALIDATION_IN_THIS_NODES)

        _header_tags = INVOICE_TAGS[_invoice_type]['header']
        _details_tags = INVOICE_TAGS[_invoice_type]['details']
        _vat_tags = INVOICE_TAGS[_invoice_type]['vat']
        _consumption_tags = INVOICE_TAGS[_invoice_type]['consumption']

        _invoices << build_invoice( invoice_xml _xml_obj,
                                    header_tags _header_tags,
                                    details_tags _details_tags,
                                    vat_tags _vat_tags,
                                    consumption_tags _consumption_tags,
                                    invoice_type_id _invoice_type_id)
    else
        context.fail!(errors: @errors)
    end
    context.response = _invoices

end
```

Código 66 - Método call do processado de faturas em xml

O método responsável pela criação do objeto utilizado na criação de uma nova entrada na tabela de faturas da base de dados é denominado por "*build_invoice*". Este método recebe como parâmetros a referência para *XML* propriamente dito, a definição dos *headers* que funcionam como "caminho" até nós inferiores no *XML*, entre outras definições de *tags*. No fundo este método funciona como um mapeamento de campos do *XML* em ordem a atributos de um objeto. Neste método é então criada uma instância para o modelo "*Invoice*", através da qual serão armazenados os dados nas tabelas correspondentes. Este processo é feito através dos métodos nativos "*new*" e "*save*", presentes em qualquer

```
_invoice = ::Ezenergy::Invoice.new(
    cpe: extract_attribute( xml invoice_xml, xml_path header_tags[0], node_with_value CPE[0]),
    user_system_id: context.user_system_id,
    currency_id: which_currency?( str extract_attribute( xml invoice_xml, xml_path header_tags[0],
```

Código 67 - Exemplo do mapeamento de campos de XML para atributos do modelo

modelo do *Rails.*

Após o objeto ser construído e a instância do Modelo ser criada, estes serão gravados e armazenados na base de dados. Caso ocorra algum erro aquando da gravação, este erro será retornado para o método chamado originalmente pela *API:* "*upload_syntatic_validation*". Este método irá então retornar a resposta ao cliente, seja esta uma resposta de sucesso ou uma resposta de erro.

```
if _it.success?

    build_response( data file: _it.invoice_upload, invoice_upload: _data, message: t('messages.upload_success'))

else

    build_response( data {
                        invoice_upload: _data,
                        errors: (_it.errors || [_it.message]),
                        message: _it.message,
                        error_code: _it.error_code,
                        invoice_upload_errors: _it.try(:invoice_upload_errors)
                    },
                    success false,  code Api::RETURN_CODE_PARAM_MISSING,  message t('errors.messages.invalid'),
                    status 404)
end
```

Código 68 - Resposta enviada ao cliente com o resultado da validação sintática

c) **Ficheiros em formato *ZIP***

Quando os ficheiros se encontram no formato ***ZIP*** estes primeiro terão de ser descompactados em memória para depois serem analisados e processados um a um. O processamento dos ficheiros descompactados é exatamente o mesmo referido anteriormente. Todo este processo é executado dentro do método *"unzip_invoice"* da classe "*ValidateUploadSyntaxAndStoraData*" que pode ser visualizado no excerto de código seguinte.

```
def unzip_invoice

    begin

        _at_least_processed_1_invoice = false

        Zip::File.open( fileName @invoice_upload.file_path.path) do |zip_file|
            # Handle entries one by one

            zip_file.each do |entry|

                ZIP_XML_NAME_REGEX.each do |regex|

                    if (entry.to_s =~ regex)

                        _at_least_processed_1_invoice = true
                        _inv_file_path = remove_zip_from_file_path + entry.to_s
                        entry.extract(_inv_file_path)
                        load_file( invoice_file_path _inv_file_path)
                        break

                    end

                end

            end

        end

    rescue Zip::ZipError => e

        Rails.logger.error(e.message)
        unknown_file_format_response!
    end

end
```

Código 69 - Método unzip_invoice da classe ValidateUploadSyntaxAndStoreData

3) Cliente – Processamento da resposta do Servidor

Ao receber a resposta proveniente do servidor, o cliente irá então processá-la. Dependendo ou não do tipo da resposta, seja de erro ou de sucesso, o cliente irá reagir de maneiras diferentes. Caso a resposta seja de sucesso, o cliente irá passar à fase seguinte da importação de faturas sendo esta a **Validação de Erros**. No caso de a resposta chegar com erros então estes serão exibidos ao utilizador sob a forma de linhas de uma tabela com o

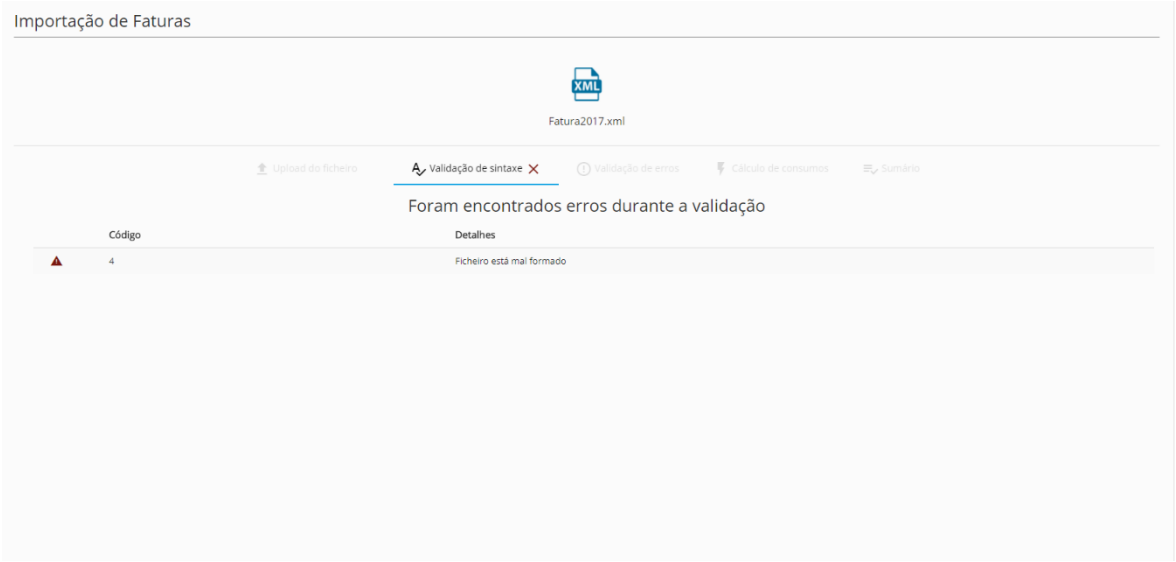


Figura 12 - Exemplo da exibição de erros ocorridos durante a Validação de Sintaxe de uma fatura

seguinte aspeto:

Ao receber a resposta com erros do servidor, o controlador responsável pelo separador da **Validação de Sintaxe** irá passar os erros recebidos para uma variável de *scope* para que esta seja acessível do lado do *HTML*.

```
function onUploadSyntaticValidationError(error) {
    $scope.isRunning = false;
    $scope.hasErrors = true;
    var stepState = {
        step: 2,
        isRunning: false,
        hasErrors: true,
        isSuccess: false
    };

    if (error.data.errors) {
        $scope.errors = error.data.errors.map(function (error) {
            return {
                description: error.errors,
                invoice_upload_error_type_id: error.invoice_upload_error_type_id
            };
        });
    }


    $scope.$emit('Invoices.Import.SetStepState', stepState);
    $scope.$emit('Invoices.Import.ValidationsReady');
}
```

Código 70 - Associação dos erros recebidos do servidor a uma variável de scope

Do lado do *HTML* a variável *"$scope.errors"* será iterada através da diretiva "*ng-repeat*" de modo a formar uma linha da tabela por cada erro e preencher as suas colunas com os diversos detalhes de cada erro.

```
<!-- Table Content -->
<md-data-table-container style="..." ng-show="hasErrors">
    <table md-data-table class="ezwaste-theme md-primary md-hue-3">
        <thead md-head md-order="tableOptions.query.order">
        <tr md-row>
            <th style="..."></th>
            <th name="{{'ezenergy.invoices.error_code' | translate }}"></th>
            <th name="{{'ezenergy.invoices.details' | translate }}"></th>
        </tr>
        </thead>
        <tbody md-body>
        <tr md-row
            ng-repeat="error in errors"
            ng-class-odd="'odd'" >
            <td style="..."><i class="material-icons" style="...">warning</i> </td>
            <td>{{error.invoice_upload_error_type_id}}</td>
            <td>{{error.description}}</td>
        </tr>
        </tbody>
    </table>
</md-data-table-container>

<md-data-table-pagination ng-if="detailsList.length > 0"
                          md-label="tableOptions.paginationTranslation"
                          md-limit="tableOptions.query.limit"
                          md-row-select="tableOptions.query.select"
                          md-page="tableOptions.query.page"
                          md-total="{{ tableOptions.total }}"
>
</md-data-table-pagination>
```

Iteração dos Erros

Código 71 - Preenchimento da tabela de erros através da utilização da diretiva ng-repeat

6.2.3. Validação de Erros

Após concluída a etapa da **Validação de Sintaxe** é então iniciada a etapa de **Validação de Erros.** Esta verificação consiste na examinação de todos os dados importados de modo a que estes sejam coerentes e não se encontrem repetidos num só **Sistema** da plataforma. Esta verificação é sempre feita em ordem aos *uploads*. Existem vários tipos de possíveis erros que poderão ocorrer durante esta verificação, sendo que algumas podem ser corrigidos através da aplicação. De seguida encontram-se listados os vários tipos:

- **Tentativa de Importação de Faturas com CPEs não registados no Sistema** – Não é permitida a importação de faturas cujos **CPEs** (Códigos Ponto de Entrega) não se encontrem previamente registados no sistema.

- **Tentativa de Importação de Faturas com *CPEs* já existentes em outros sistemas –** Tal como o nome indica não é permitida a importação de faturas para um sistema que contêm **CPEs** (Códigos Ponto de Entrega) já existentes/importados noutros sistemas.

- **Tentativa de Importação de Faturas já previamente importadas** – Não é permitida a importação de faturas que já tenham sido importadas previamente neste sistema.

- **Tentativa de Importação de Faturas cujos períodos de faturação se sobreponham a outras faturas já importadas** – Não é permitida a importação de faturas cujos períodos de faturação sejam iguais ou se sobreponham aos de outras faturas previamente importadas. Não faz sentido haver duas faturas diferentes registadas no mesmo período de tempo.

- **Tentativa de Importação de Faturas cujos números de contadores associados aos seus CPEs tenham sido alterados** – Na aplicação cada **CPE** tem um número de contador associado. Caso este número não seja coerente com o número da fatura a ser importada, ocorrerá este erro.

Como já referi anteriormente alguns destes erros podem ser corrigidos através da aplicação na mesma página da verificação. Estas correções passam desde a criação de **Pontos de Instalação** de modo a registar o **CPE** em falta no sistema, etc. Por outro lado, os erros que não são possíveis de correção necessitam de uma abordagem externa à aplicação em si, seja esta relacionada com a verificação das faturas em questão.

1) Cliente – Pedido ao Servidor

Tal como nas outras etapas, também esta se inicia do lado do Cliente. Após a verificação de sintaxe o cliente será redirecionado para o separador correspondente à verificação de erros e será então enviado um pedido ao Servidor para que este inicie então o processo.
Este pedido é feito através do seguinte excerto de código:

```
EzEnergyInvoiceUploadsService.processUpload({
        id: $scope.uploadInfo.id
    },onUploadErrorValidation, onUploadErrorValidationError);
```

Código 72 - Excerto de código responsável pelo pedido ao servidor para iniciar a verificação de erros

2) Servidor

A receção como sempre, feita através de uma dada rota que representa o *endpoint* ao qual o cliente faz o pedido. Este *endpoint*, neste caso é representado pela seguinte rota:

```
post 'invoice_uploads/process_import'
```

Código 73 - Rota representante do endpoint de inicio da validação de erros de uma fatura

Da mesma forma que todas as outras rotas, esta aponta para um método de um controlador específico. Este controlador é o "InvoiceUploadsController" e o método é o "process_import".

O método “process_import” é responsável por iniciar o processo de validação de erros. Esta inicialização é feita através da classe “ContentValidationsProcessor”.

```
def process_import

    _success = false
    _invoice_upload = ::Ezenergy::InvoiceUpload.includes(:invoices).find(params[:id])
    _user_system = ::Common::UserSystem.find_by(id: _invoice_upload.user_system_id)
    if _invoice_upload.validated? || _invoice_upload.processed?
        _success = true
    else
        _process_consumptions = ::Ezenergy::Invoice::ContentValidationsProcessor
                                    .call( context user_id: current_user.id,
                                           invoice_upload: _invoice_upload,
                                           invoice_id: params[:invoice_id],
                                           user_system_id: params[:user_system_id])

        _success = _process_consumptions.success?
    end
    _data = serialize_invoice_upload( invoice_upload _invoice_upload.reload)

    if _success
        build_response( data invoice_upload: _data, file: _invoice_upload, message: t('messages.upload_success')
    else

        #::InvoiceAlertsMailer.invoice_upload_failed_email( current_user.email, _invoice_upload.id).deliver_lat

        build_response( data {
                            invoice_upload: _data,
                            invoice_errors: serialize_invoice_errors( invoice_upload _invoice_upload),
                            user_system: serialize_user_system( user_system _user_system),
                            import_result: false,
                            error_code: _process_consumptions.error_code,
                            message: _process_consumptions.message
                        }, success false, code _process_consumptions.error_code,
                         message _process_consumptions.message, status 404)
    end
end
```

Chamada ao método call da classe ContentValidationsProcessor

Código 74 - Método process_import da classe InvoiceUploadsController

Ao ser chamada esta classe vai chamar um outro método presente no modelo “*InvoiceUpload*”. Este método denominado “*invoices_validation_by_upload*” vai então, por sua vez, chamar a função *SQL* presente na base de dados.

```
_success = ::Ezenergy::InvoiceUpload.invoices_validation_by_upload( user_id context.user_id,
                                                                    invoice_upload_id context.invoice_upload.id)

def self.invoices_validation_by_upload(user_id, invoice_upload_id )

    _result = DbConn.execute_scalar( procedureName 'ezenergy.invoices_validation_by_upload',
                                     *paramsValues user_id, invoice_upload_id )
    return _result.to_boolean\

end
```

Código 75 - Processo de chamada da função SQL "invoices_validation_by_upload"

Na figura seguinte encontra-se a criação e início do funcionamento da função responsável pela junção de todos os erros ocorrentes no *upload* em questão:

```
CREATE OR REPLACE FUNCTION ezenergy.invoices_validation_by_upload(
    i_user_id integer,
    i_invoice_upload_id integer)

    RETURNS BOOLEAN AS
$BODY$

DECLARE
    _server_date timestamp without time zone;

    _counter integer;
    _r record;
    _json json;

BEGIN

    -- marcar erros pre-existentes
    UPDATE ezenergy.invoice_errors set flg_active=FALSE
    WHERE invoice_id IN(select id from ezenergy.invoices as i
    where i.invoice_upload_id = i_invoice_upload_id and i.flg_deleted = false)
    AND flg_deleted = FALSE AND solver_id IS NULL;

    _counter = 0;
```

Código 76 - Criação e início do funcionamento da função SQL invoices_validation_by_upload

De modo a validar todos os tipos de erros referidos, esta função recorre à utilização de outras funções auxiliares, sendo que cado uma valida um tipo de erro específico. Cada uma destas funções auxiliares é executada uma vez por cada fatura pertencente ao *upload* em questão. Estas funções auxiliares são todas chamadas da mesma maneira e o seu resultado é sempre retornado no mesmo formato. Isto permite o armazenamento em massa de todos os erros ocorridos neste *upload*.

```
for _r in ( SELECT * FROM ezenergy.get_upload_cpes_in_other_system(i_invoice_upload_id) )
loop
    -- raise notice 'get_upload_cpes_in_other_system _r: %', _r;
    _json = to_json(_r);
    perform ezenergy.add_error_to_invoice(i_user_id, _r.new_invoice_id, 3, _json);
    _counter = _counter + 1;
end loop;
```

Código 77 - Exemplo da chamada de uma função auxiliar para a validação de um erro específico

De seguida é descrito o funcionamento de cada uma das funções auxiliares e também a que tipo de erros estas estão associadas.

a. CPEs não existentes no sistema

Este erro ocorre quando a aplicação deteta uma tentativa de importação de uma ou mais faturas cujos **CPEs** ainda não se encontram registados no sistema onde estas estão a ser importadas. Este tipo de erros é processado pela função *SQL* "*get_upload_unexisting_cpes*". Esta verificação funciona através da iteração de todas as faturas relacionadas com o *upload* em questão, de modo a verificar se os seus **CPEs** se encontram presentes no sistema onde estas foram importadas. Esta relação entre **CPEs** e sistemas é feita através dos **Pontos de Instalação** (cada ponto pode ter um ou mais **CPEs** associados).

```
-- Outup: lista de cpe's de um certo invoice_upload que não existe num certo sistema
SELECT DISTINCT
    i.id             AS new_invoice_id,
    -1               AS invoice_id,
    -1               AS old_invoice_id,
    i.cpe            AS cpe,
    -1               AS installation_point_id,
    ''::VARCHAR      AS previous_meter_number,
    i.user_system_id AS user_system_id

FROM ezenergy.invoices AS i
WHERE i.invoice_upload_id = i_invoice_upload_id
      AND i.flg_deleted = FALSE
      AND (i_invoice_id IS NULL OR i.id = i_invoice_id)
      AND COALESCE(i.workflow_state_id, -1) NOT IN(7,8)
      AND i.cpe NOT IN (SELECT DISTINCT ipc.cpe
                        FROM ezenergy.installation_point_cpes AS ipc
                            JOIN ezenergy.installation_points AS ip
                            ON ipc.installation_point_id = ip.id
                            AND ip.user_system_id = i.user_system_id
                        where ipc.flg_deleted = FALSE AND ip.flg_deleted = FALSE
                        GROUP BY ipc.cpe)
      AND ezenergy.check_invoice_error_above_ranks(i.id, 3) = true;
```

Código 78 - Função SQL para verificar se os CPEs importados existem no sistema

b. CPEs existentes noutros sistemas

Este erro ocorre quando a aplicação deteta uma tentativa de importação de uma ou mais faturas em que um ou mais **CPEs** já se encontram registados noutros sistemas que não aquela onde a importação está a ser feita. Este tipo de erros é processado pela função *SQL* "*get_upload_cpes_in_other_system*". Esta verificação funciona através da iteração de todos os **CPEs** presentes nas faturas do *upload* em questão de modo a verificar se existe uma ligação entre eles e algum **Ponto de Instalação** que pertença a um sistema que não aquele onde a importação está a ser efetuada.

```sql
--Output: lista de cpes, id de ponto de instalação e nome do sistema
-- de um certo invoice_upload que existem noutros sistemas que não o sistema que foi feito o upload
SELECT DISTINCT
    i.id  AS new_invoice_id,
    -1    AS invoice_id,
    -1      AS old_invoice_id,
    i.cpe,
    ip.id AS installation_point_id,
           '':: VARCHAR AS previous_meter_number,
    ip.user_system_id AS user_system_id

FROM
    ezenergy.invoices AS i
    RIGHT JOIN ezenergy.installation_point_cpes AS ipc
        ON ipc.cpe = i.cpe AND ipc.flg_deleted = FALSE
    JOIN ezenergy.installation_points AS ip
        ON ip.id = ipc.installation_point_id AND ip.flg_deleted = FALSE
WHERE
    i.invoice_upload_id = i_invoice_upload_id
    AND i.flg_deleted = FALSE
    AND i.user_system_id = _user_system_id AND ip.user_system_id <> _user_system_id
    AND COALESCE(i.workflow_state_id, -1) NOT IN (8, 7);
```

Código 79 - Função SQL para verificar a utilização de CPEs da importação atual, em outros sistemas

c. Importação de Faturas já previamente importadas

Este erro ocorre quando a aplicação deteta uma tentativa de importação de uma ou mais faturas previamente importadas. Este tipo de erros é processado pela função *SQL* "*get_duplicated_invoices*". Esta verificação é realizada através da iteração de todas as faturas a serem importadas no presente *upload* e da sua comparação com outras faturas importadas noutros *uploads* direcionados ao mesmo sistema. De modo a decidir se uma fatura é igual a outra, é feita uma comparação dos seus números de documento. Estes números são únicos para cada fatura permitindo assim uma certeza na sua distinção.

```
-- Outup: lista de factura já inseridas previamente
SELECT
new_invoices.inv_id,
    old_invoices.inv_id AS invoice_id,
    old_invoices.inv_id AS old_invoice_id,
    '' :: VARCHAR     AS cpe,
    -1                AS installation_point_id,
    '' :: VARCHAR     AS previous_meter_number,
    new_invoices.usi   AS user_system_id
FROM
(
    SELECT i.id as inv_id, i.user_system_id as usi, *
    FROM ezenergy.invoices AS i
    LEFT JOIN ezenergy.invoice_uploads AS iu ON i.invoice_upload_id = iu.id
    WHERE i.invoice_upload_id = i_invoice_upload_id AND i.flg_deleted = FALSE
    and i.workflow_state_id != 8
) AS new_invoices
INNER JOIN
(
    SELECT i2.id as inv_id, *
    FROM ezenergy.invoices AS i2
    LEFT JOIN ezenergy.invoice_uploads AS iu ON i2.invoice_upload_id = iu.id
    WHERE i2.flg_deleted = false
    AND i2.workflow_state_id = 7

) AS old_invoices ON new_invoices.cpe = old_invoices.cpe
AND new_invoices.inv_id <> old_invoices.inv_id
WHERE new_invoices.document_number = old_invoices.document_number
AND ezenergy.check_invoice_error_above_ranks(new_invoices.inv_id, 2) = true;
```

Código 80 - Função SQL para verificar a existência de faturas duplicadas num dado sistema

d. Importação de faturas com períodos de faturação sobrepostos

Este erro ocorre quando a aplicação deteta uma tentativa de importação de uma ou mais faturas cujos períodos de faturação se sobreponham aos de outras faturas previamente importadas. Este tipo de erros é processado pela função *SQL* "*get_uploaded_inv_ids_with_same_dates*". Esta verificação é feita através da iteração de todas as faturas importadas e a comparação das suas datas de inicio e fim relativamente às datas das faturas já existentes no sistema.

```
SELECT
 q1.id  new_invoice_id,
 q2.id  invoice_id,
 q2.id  old_invoice_id,
 q1.cpe cpe,
 -1     installation_point_id,
 ''::VARCHAR AS previous_meter_number,
 -1     user_system_id
 FROM
 (
     SELECT *
     FROM ezenergy.invoices AS i
     WHERE i.invoice_upload_id = i_invoice_upload_id and i.flg_deleted = false
         and i.workflow_state_id != 8
 ) AS q1
 INNER JOIN
 (
     SELECT *
     FROM ezenergy.invoices AS i
     WHERE i.invoice_upload_id <> i_invoice_upload_id
     AND i.workflow_state_id = 7 and i.flg_deleted = false
 ) AS q2 ON q1.id <> q2.id AND q1.cpe = q2.cpe
 WHERE ((q1.launch_date_uts > q2.launch_date_uts AND q1.end_date_uts < q2.end_date_uts)
           OR (q1.launch_date_uts < q2.launch_date_uts AND q1.end_date_uts > q2.end_date_uts)
           OR (q1.launch_date_uts > q2.launch_date_uts AND q1.end_date_uts > q2.end_date_uts
             AND q1.launch_date_uts < q2.end_date_uts)
           OR (q1.launch_date_uts < q2.launch_date_uts AND q1.end_date_uts < q2.end_date_uts
             AND q1.end_date_uts > q2.launch_date_uts)
           OR (q1.launch_date_uts = q2.launch_date_uts AND q1.end_date_uts < q2.end_date_uts)
           OR (q1.launch_date_uts > q2.launch_date_uts AND q1.end_date_uts = q2.end_date_uts)
           OR (q1.launch_date_uts = q2.launch_date_uts AND q1.end_date_uts = q2.end_date_uts))
           AND ezenergy.check_invoice_error_above_ranks(q1.id, 3) = true;
```

Código 81 - Função SQL para verificar a existência de faturas cujos períodos de faturação se sobrepõem

e. Importação de faturas cujos números de contador tenham sido alterados

Este erro ocorre quando a aplicação deteta uma tentativa de importação de uma ou mais faturas cujos números de contadores tenham sido alterados desde a ultima importação de faturas referentes a um dado **CPE**. Este tipo de erros é processado pela função *SQL* "*get_new_uploaded_invoices_where_meter_number_changed*". Esta verificação é feita através da iteração de todas as a serem importadas e a sua comparação com as faturas já importadas previamente. Caso exista uma diferença no número de um dos contadores, isto

```sql
SELECT
    data.invoice_id       AS new_invoice_id,
    i_invoice_id          AS invoice_id,
                data.previous_id        AS old_invoice_id,
    data.cpe              AS cpe,
    data.ip_id            AS installation_point_id,
    data.previous_meter_number as previous_meter_number,
    data.user_system_id   AS user_system_id
FROM (
    SELECT i.id AS invoice_id,
        i.invoice_upload_id,
        il.meter_number,
        i.launch_date,
        lag(il.meter_number) over distance_window AS previous_meter_number,
        lag(i.id) over distance_window AS previous_id,
        i.cpe,
        ipc.installation_point_id AS ip_id,
        i.user_system_id
    FROM ezenergy.invoices AS i
    LEFT JOIN ezenergy.invoice_uploads AS iu ON iu.id = i.invoice_upload_id
    LEFT JOIN ezenergy.installation_point_cpes AS ipc ON i.cpe = ipc.cpe
    LEFT JOIN
      (SELECT il.invoice_id, il.meter_number
       FROM ezenergy.invoice_lines AS il
       WHERE il.flg_deleted = false
        AND coalesce(il.meter_number, '') != ''
        AND il.component_type = 'InvoiceConsumption'
       ) AS il ON il.invoice_id = i.id
    WHERE
       ( (i.workflow_state_id < 7 AND i.flg_deleted = false AND iu.status IN(102)
       AND iu.id = i_invoice_upload_id ) OR
        (i.workflow_state_id = 7 AND i.flg_deleted = false AND iu.status IN(103, 104)
        AND iu.id <> i_invoice_upload_id) )

    WINDOW distance_window AS (PARTITION BY i.cpe ORDER BY i.launch_date)
) AS data
WHERE data.invoice_upload_id = i_invoice_upload_id AND coalesce(data.previous_meter_number, '') != ''
      AND (coalesce(data.previous_meter_number, '') != coalesce(data.meter_number, '')
          AND coalesce(data.meter_number, '') != '')
AND ezenergy.check_invoice_error_above_ranks(data.invoice_id, 3) = true
ORDER BY data.previous_id DESC;
```

Código 82 - Função SQL para verificar a ocorrência de alterações no número de contador de faturas, entre importações

significa uma incoerência entre faturas.

Após estas verificações serem executadas os erros existentes serão então introduzidos e armazenados na base de dados como registos da tabela "*invoice_errors*" com ligação à fatura a que dizem respeito.

```
UPDATE ezenergy.invoice_errors SET
    flg_active = TRUE,
    updated_at = _server_date,
    updater_id = i_user_id
WHERE id = _invoice_error_id;
```

Código 83 - Armazenamento dos erros e sua associação à fatura a que dizem respeito

Ao terminar todo este processo, será então enviada uma resposta ao cliente com o resultado de todas estas verificações.

```
if _success
    build_response( data invoice_upload: _data,
                    file: _invoice_upload,
                    message: t('messages.upload_success'))
else

    build_response( data {
                        invoice_upload: _data,
                        invoice_errors: serialize_invoice_errors( invoice_upload _invoice_upload),
                        user_system: serialize_user_system( user_system _user_system),
                        import_result: false,
                        error_code: _process_consumptions.error_code,
                        message: _process_consumptions.message
                    }, success false, code _process_consumptions.error_code,
                    message _process_consumptions.message, status 404)
end
```

Código 84 - Envio da resposta ao cliente após a validação de erros

3) Cliente – Processamento da resposta do Servidor

Ao receber a resposta do Servidor, o Cliente reage de diferentes formas, consoante o resultado da validação de erros. Caso a validação tenha sido corrida com sucesso (sem a ocorrência de erros nas faturas importadas) o cliente irá passar à próxima etapa da importação, sendo estas o **Cálculo de Consumos**. Caso tenham ocorrido erros durante esta etapa, o cliente irá iterá-los e disponibilizá-los sob a forma de linhas de uma tabela, tal como ocorre na etapa anterior. Ao receber a listagem de erros ocorridos durante esta etapa, o Cliente vai copiar essa listagem para uma variável de *scope* onde esta poderá ser acedida do lado do *HTML*. Esta listagem é então conseguida através da utilização da diretiva "*ng-repeat"* que irá iterar a variável de *scope* de modo a listar cada erro numa linha da tabela.

```
<tr md-row
    ng-repeat="error in invoice_error.errors"
    ng-if="error.selected || no_ge_toggle"
    ng-class-odd="'odd'"

>
```

Código 85 - Construção das linhas da tabela de listagem de erros através da diretiva ng-repeat

Cada linha da tabela é composta por três colunas, sendo que as duas primeiras dizem respeito a informação relativa ao erro e a terceira diz respeito às possíveis soluções para o mesmo. Nem todos os erros têm resolução, pelo que existem apenas quatro tipos de possíveis resoluções a aplicar.

- **Criação de um Ponto de Instalação Novo** – Esta resolução permite ao utilizador criar no momento um ponto de instalação novo e associar-lhe um ou mais **CPEs** das faturas a serem importadas. Esta resolução só pode ser aplicada a erros relacionados com a inexistência de **CPEs** importados registados no sistema.

- **Mudança de Número de Contador** – Esta resolução permite ao utilizador alterar um dado número de contador para o que está a ser importado no momento. Esta resolução só pode ser aplicada a erros relacionados com alterações no número de contador.

- **Forçar a Importação** – Esta resolução permite ao utilizador forçar a importação de dadas faturas independentemente de estas conterem erros. Esta resolução só pode ser aplicada no caso da ocorrência de erros relacionados com datas sobrepostas e existência de faturas duplicadas.

- **Ignorar Fatura** – Esta resolução permite ao utilizador ignorar a importação de uma dada fatura que contenha erros. Esta resolução só pode ser aplicada no caso da ocorrência de erros relacionados com a existência de faturas duplicadas.

6.2.4. Cálculo de Consumos

Após concluída a etapa da **Validação de Erros** é iniciada a etapa de **Cálculo de Consumos.** Esta etapa consiste na examinação de todos os consumos importados de modo a somá-los e associá-los aos seus respetivos **CPEs** para depois serem armazenados sob a forma de dados possíveis de serem trabalhados facilmente. Esta etapa é especialmente importante pois funciona como base no que toca ao manuseamento de dados de modo a que estes sejam mais fáceis de ler e trabalhar na **Análise de Consumos**.

1) **Cliente – Pedido ao Servidor**

Da mesma forma que nas outras etapas, também esta se inicia do lado do Cliente. Após a **Validação de Erros**, o cliente será redirecionado para o separador correspondente ao **Cálculo de Consumos** e será então enviado um pedido ao Servidor para que este inicie então o processo. Este pedido é feito através do seguinte excerto de código:

```
EzEnergyInvoiceUploadsService.calculateConsumptions({
    id: $scope.uploadInfo.id
}, onUploadConsumptionCalculation, onUploadConsumptionCalculationError);
```

Código 86 - Serviço usados para iniciar o pedido de Calculo de Consumos ao Servidor

2) **Servidor**

A receção como sempre, feita através de uma dada rota que representa o *endpoint* ao qual o cliente faz o pedido. Este *endpoint*, neste caso é representado pela seguinte rota:

```
post 'invoice_uploads/calculate_consumptions'
```

Código 87 - Rota correspondente ao endpoint responsável pela receção de pedidos relativos ao processo de calculo de consumos

Ao receber um pedido da parte do cliente, esta rota irá redirecionar tal pedido para o método "calculate_consumptions" do controlador "InvoiceUploadsController". Neste

método será então iniciado todo o processo relativo ao cálculo de consumos. Este cáculo é efetuado por meio de uma função *SQL* denominada "*update_cpe_consumptions*".

Esta função pode dividir-se em duas partes sendo que a primeira tem como propósito a limpeza de todos os consumos pré-existentes e a segunda tem como objetivo o processo de cálculo propriamente dito.

- **Limpeza de Consumos Pré-existentes** – Esta etapa do **Cálculo de Consumos** é especialmente importante de modo a manter a coerência entre os dados exibidos ao utilizador. Como tal, é necessário apagar quaisquer cálculos feitos sob versões obsoletas dos dados presentes no sistema. Isto vai permitir à função de cálculo recalcular todos os consumos relacionados com um dado *upload* de modo a não ocorrerem "colisões de dados" ou incoerências.

- **Cálculo de Consumos** – Esta etapa, tal como o nome indica, efetuará todos os cálculos de consumos globais relacionados com o presente *upload*. Isto significa que a função irá correr todos os **CPEs** e respetivos consumos de modo a processá-los matematicamente e gerar uma listagem de consumos globais e dados de estatísticos relacionados com os mesmos. Estes cálculos são sempre efetuados em ordem aos **CPEs** para um dado sistema.

Os consumos calculados pela função podem apresentar-se como dois tipos, sendo estes **Energia Ativa e Energia Reativa**.

- **Energia Ativa** – Diz respeito à parte da energia elétrica, medida em *Killowatt-Hora* (*kWh)* que executa o trabalho propriamente dito. No caso de um motor, por exemplo, é a **Energia Ativa** que faz com que o motor mova as suas peças.

- **Energia Reativa** – Diz respeito à parte da energia elétrica, medida em *Killovolt-Amper-Reativo-Hora (kVARh)* que não executa trabalho. Esta é utilizada apenas para gerar os campos eletromagnéticos necessários para o funcionamento de um equipamento.

Após serem realizados todos os cálculos de consumo, é então retornada uma resposta ao cliente. Tal como nas outras etapas, caso tenham ocorrido erros durante esta etapa, a resposta será acompanhada de uma mensagem de erro. Caso contrário, a resposta será acompanhada de uma mensagem de sucesso.

```
if _import_consumption_data_result

    build_response( data invoice_upload: _data, message: t('messages.upload_success'))
else
    _message = I18n.t('errors.unable_to_create')

    build_response( data {
                        invoice_upload: _data,
                        errors: [_message],
                        import_result: false,
                        error_code: ::Api::RETURN_CODE_MODEL_SAVE,
                        message: _message
                    }, success false, code Api::RETURN_CODE_MODEL_SAVE, message _message, status 404)
end
```

Código 88 - Excerto de código onde é enviada a resposta com o resultado do cálculo de consumos

3) **<u>Cliente – Processamento da resposta do Servidor</u>**

Aquando da receção da resposta proveniente do Servidor, o Cliente irá processar a mesma de modo a reagir e altura o seu aspeto afim de refletir o resultado da mesma. Caso a resposta seja dada como sucesso o Cliente irá exibir uma página com o seguinte

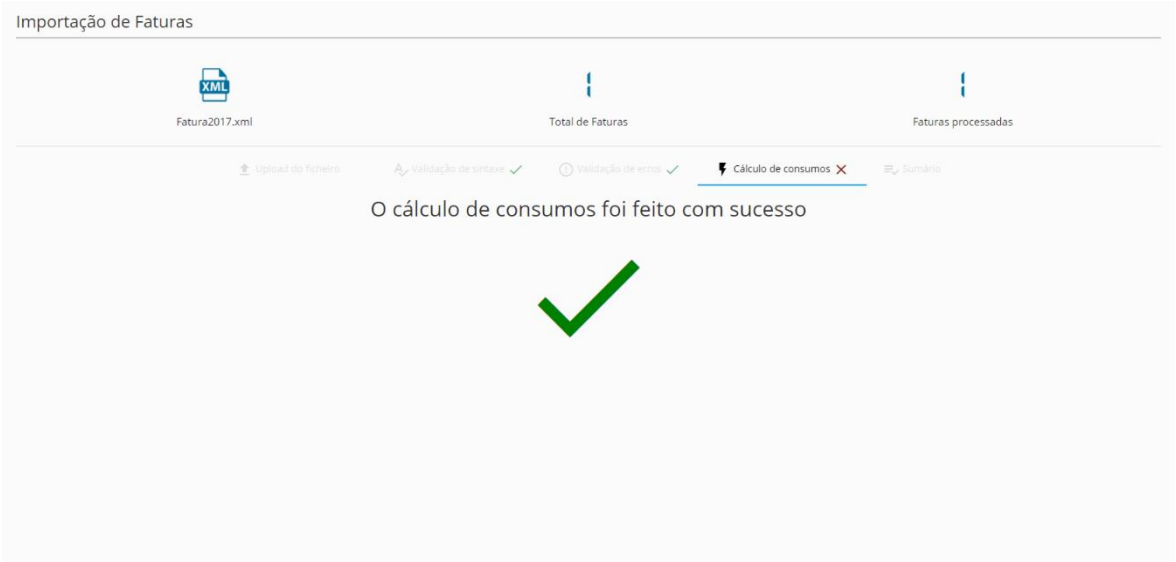


Figura 13 - Aspeto da página de importação após receção do resultado da etapa de Cálculo de Consumos

aspeto:

Com isto é dado como concluído todo o processo de importação de faturas. Após todo este processo a página de importação redireciona o utilizador para o separador de “Sumário” onde poderá verificar todos os alertas gerados ou erros corrigidos ao longo deste processo.

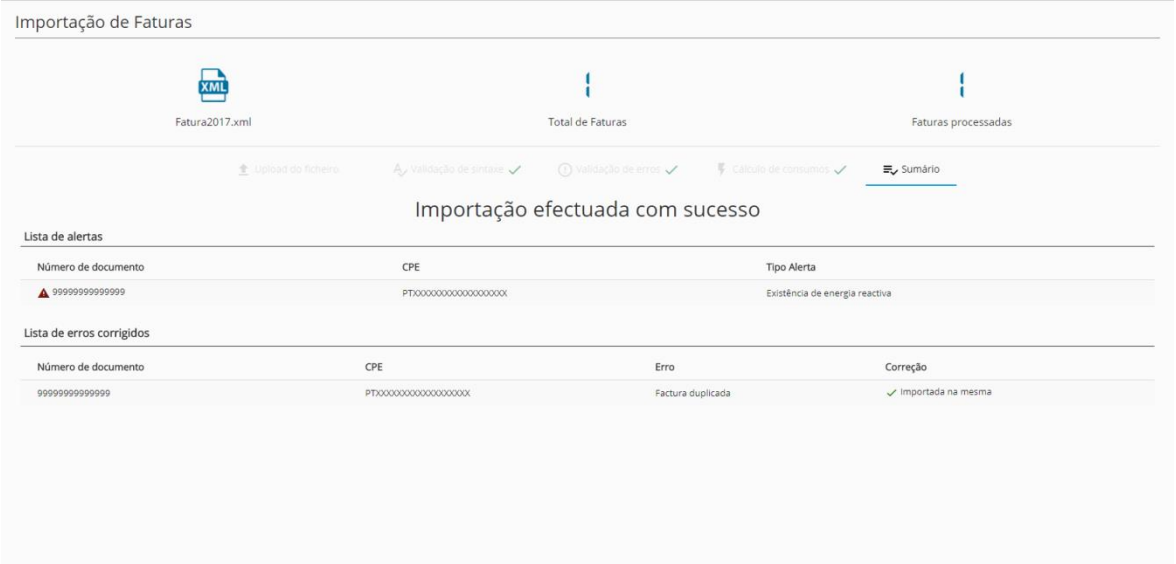


Figura 14 - Aspeto do separador de Sumário exibido ao utilizador após a conclusão da importação de faturas

6.3. Análise de Consumos

A **Análise de Consumos** diz respeito a uma página de visualização detalhada dos consumos referentes a um dado **Ponto de Instalação**. Esta página reflete o resultado de todo o processo de importação, com especial ênfase na etapa de **Cálculo de Consumos**. Esta página de análise permite ao utilizador entender de forma mais gráfica e simplista todos os dados pertencentes às faturas que importou para o sistema. A página pode dividir-se em três secções: **Visão Geral, Monitorização e Detalhes do Ponto de Instalação.**

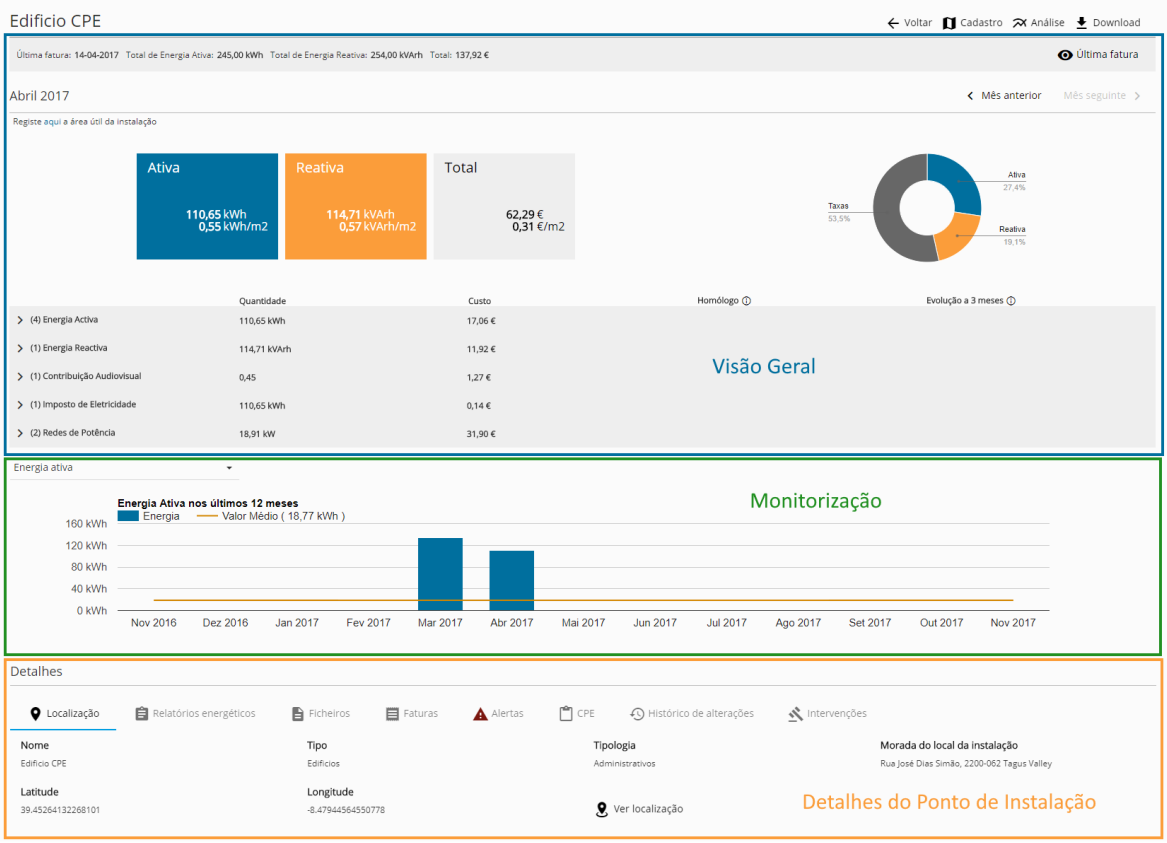


Figura 15 - Aspeto geral da página de Análise de Consumos

1) <u>Visão Geral</u>

Esta secção tem como objetivo exibir ao utilizador detalhes gerais relativos aos consumos e preços gerais do **Ponto de Instalação** selecionado. Esta informação encontra-se presente nesta seção em três formas distintas: **Cartões de Energia, Gráfico de Custos Percentuais e Acordeão de Gastos e Consumos Detalhados.**

- **Cartões de Energia** – Representa os três quadrados à esquerda no topo da página. Cada um destes quadrados apresenta um total de dados diferentes, sendo que o primeiro apresenta o total de Energia Ativa Consumido seguido do total de Energia Ativa Consumida por Metro Quadrado; O segundo quadrado apresenta o total de Energia Reativa Consumida seguido do total de Energia Reativa Consumida por Metro Quadrado; E o terceiro quadrado apresenta o Custo Total Registado seguido do Custo Total Registado por Metro Quadrado;

Figura 16 - Cartões de Energia; Gastos e Consumos totais e por metro quadrado

- **Gráfico de Custos Percentuais** – Tal como o nome indica, apresenta o total de Custos efetuados neste **Ponto de Instalação** subdivididos percentualmente pelo tipo de consumo ao qual se encontram associados (Energia Ativa, Energia Reativa ou Taxas).

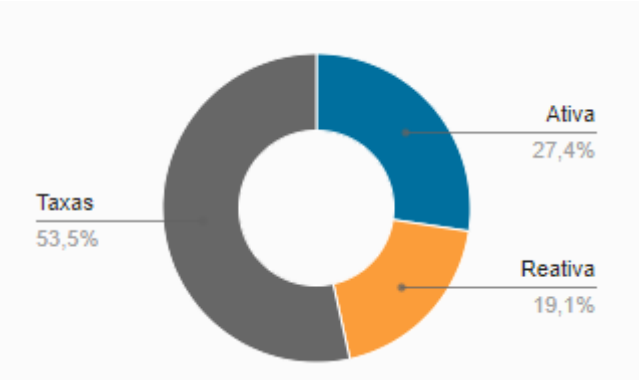
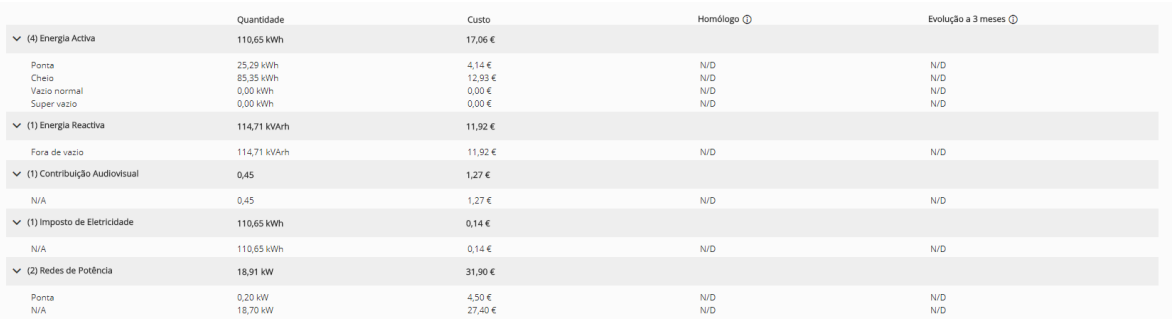


Figura 17 - Gráfico de Custos Percentuais; Registo das percentagens de gastos efetuados no Ponto de Instalação

- **Acordeão de Gastos e Consumos Detalhados** – Esta secção é denominada por "acordeão" dada a sua propriedade de expandir e colapsar os seus conteúdos de acordo com divisões por grupos. Nesta secção estão exibidos em detalhe todos os consumos e respetivos gastos de todos os componentes das faturas importadas (Energia Ativa e Reativa, Contribuição Audiovisual, Imposto de Eletricidade, etc.). Para além disso esta secção apresenta também uma comparação dos gastos/consumos relativamente ao mesmo período do ano anterior (Homologo) e a Tendência de Evolução para os próximos três meses.



	Quantidade	Custo	Homólogo ⓘ	Evolução a 3 meses ⓘ
(4) Energia Activa	110,65 kWh	17,06 €		
Ponta	25,29 kWh	4,14 €	N/D	N/D
Cheio	85,35 kWh	12,93 €	N/D	N/D
Vazio normal	0,00 kWh	0,00 €	N/D	N/D
Super vazio	0,00 kWh	0,00 €	N/D	N/D
(1) Energia Reactiva	114,71 kVArh	11,92 €		
Fora de vazio	114,71 kVArh	11,92 €	N/D	N/D
(1) Contribuição Audiovisual	0,45	1,27 €		
N/A	0,45	1,27 €	N/D	N/D
(1) Imposto de Eletricidade	110,65 kWh	0,14 €		
N/A	110,65 kWh	0,14 €	N/D	N/D
(2) Redes de Potência	18,91 kW	31,90 €		
Ponta	0,20 kW	4,50 €	N/D	N/D
N/A	18,70 kW	27,40 €	N/D	N/D

Figura 18 -Acordeão de Gastos e Consumos Detalhados

2) **Monitorização**

A secção da monitorização permite ao utilizador consultar uma espécie de histórico relativo a Energias ou Potências registadas no **Ponto de Instalação**. Este histórico é apresentado sob a forma de gráfico.

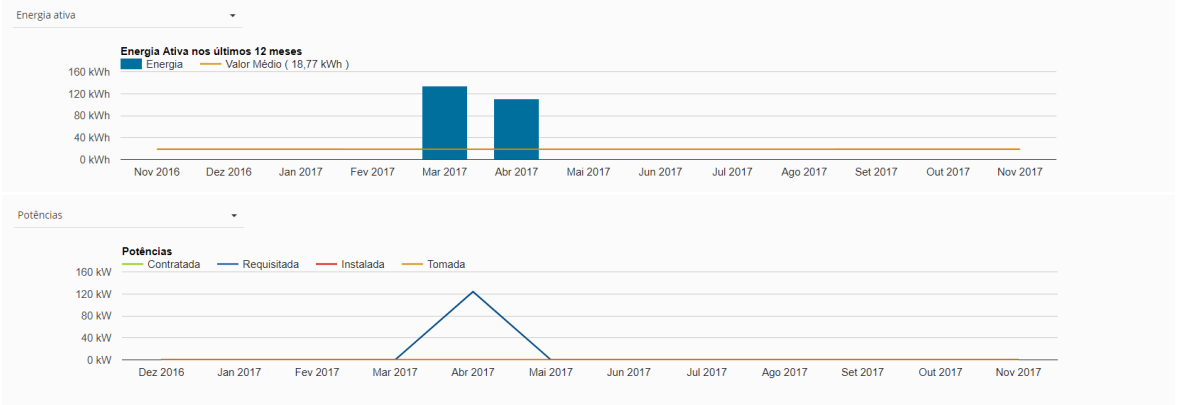


Figura 19 - Gráficos de Histórico de Energia e Pontência

3) **Detalhes do Ponto de Instalação**

Esta secção, tal como o nome indica, tem como objetivo a disponibilização ao utilizador de todos os detalhes relativos ao **Ponto de Instalação**. Para tal, estes são exibidos e organizados sob a forma de separadores, sendo que cada separador representa um determinado grupo de detalhes.

- **Localização** – Este separador apresenta todos os detalhes relativos à localização do **Ponto de Instalação** bem como o seu nome e tipologia. Está presente também neste separador um botão que permite ao utilizador visualizar a localização geográfico do **Ponto de Instalação** num mapa.

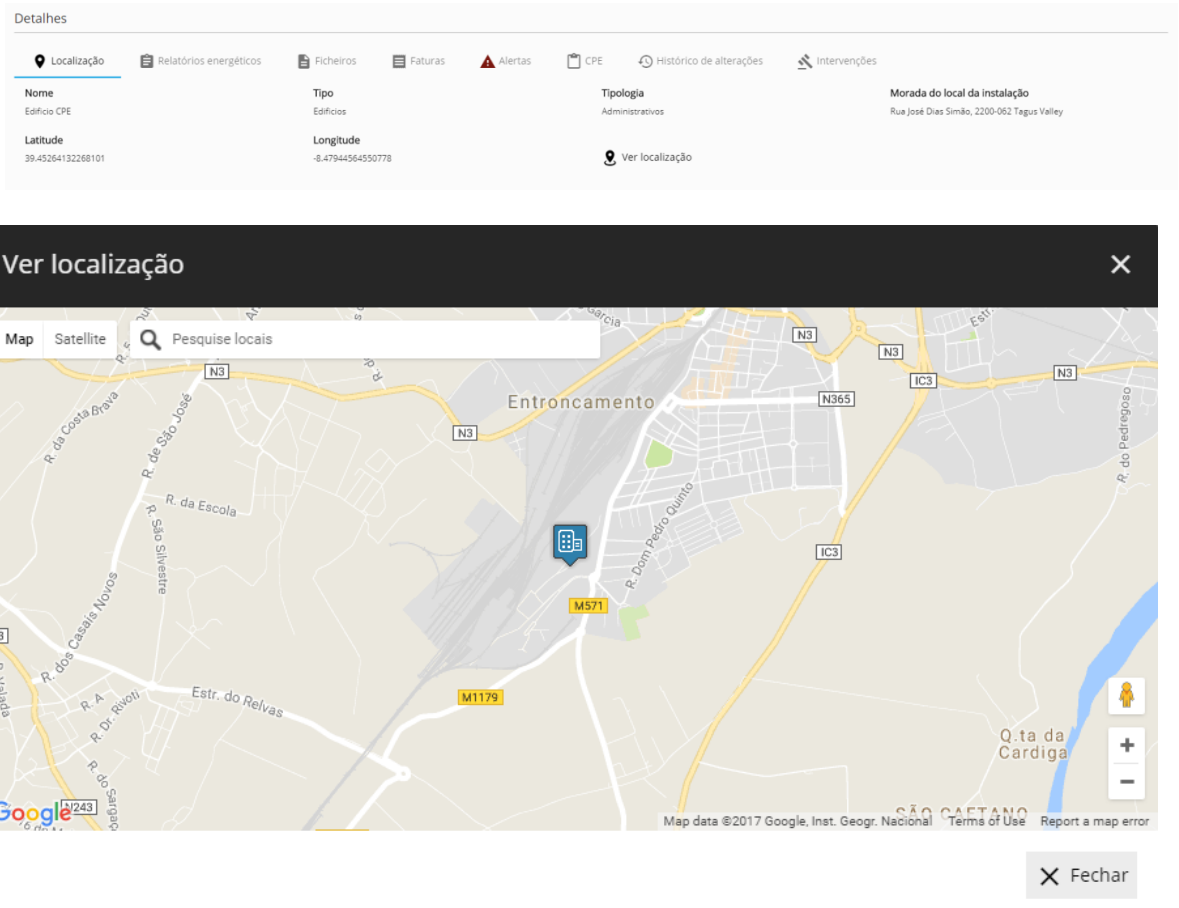


Figura 20 - Separador de Localização de um Ponto de Instalação e visualização da mesma num mapa

- **Relatórios Energéticos** – Este separador permite ao utilizador gerar Relatórios referentes ao **Ponto de Instalação** e fazer o seu *download* em PDF. Este processo funciona como uma exportação de dados no formato apresentado na plataforma.

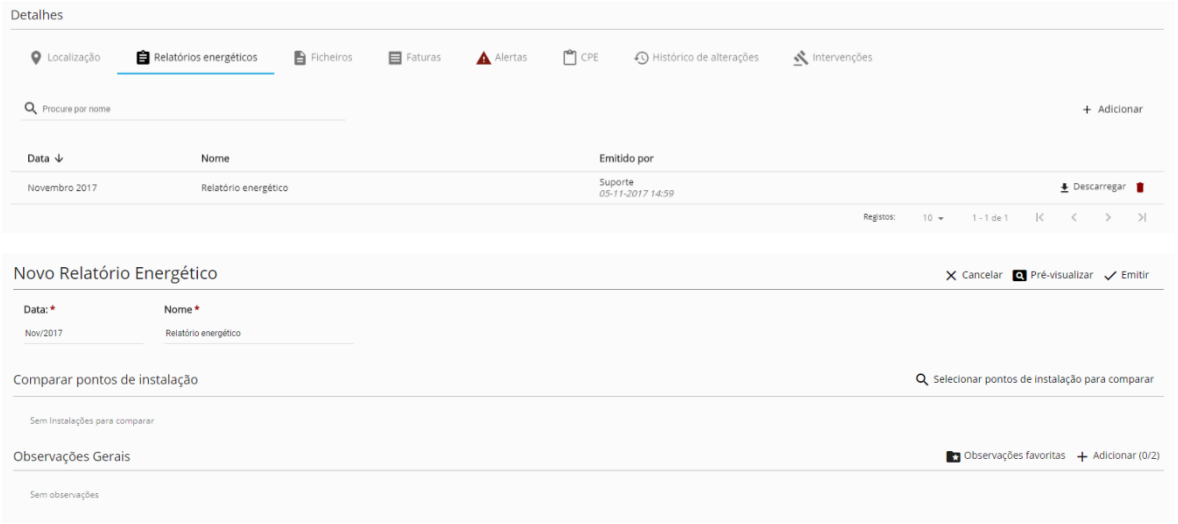


Figura 21 - Separador para listagem de Relatórios Energéticos respeitantes ao Ponto de Instalação e página de adição de novo relatório

- **Ficheiros** – Neste separador estão listados todos os ficheiros *XML* carregados para a plataforma que se relacionaram com o **Ponto de Instalação**. É também visível a data de *upload* e o estado em que a sua importação terminou.

Figura 22 - Separador de Ficheiros onde estão exibidos todos os ficheiros importados para este Ponto de Instalação

- **Faturas** – Neste separador encontram-se listadas todas as Faturas importadas para este **Ponto de Instalação.** Nesta listagem é mostrado o **nº de Documento**, a **Designação**, o **Consumo** e o **Custo** totais de cada Fatura.

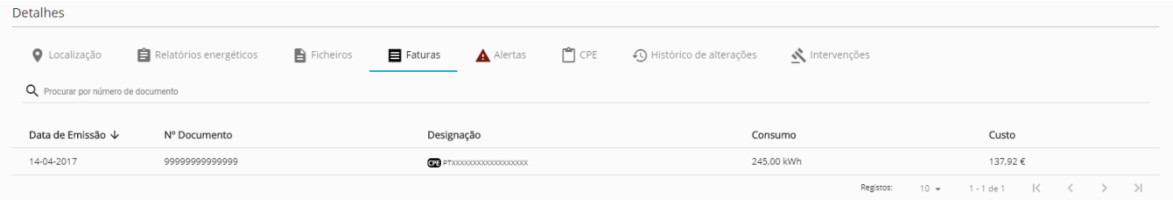


Figura 23 - Separador de listagem de todas as faturas importadas para o Ponto de Instalação

- **Alertas** – Neste separador encontram-se listados todos os alertas gerados para cada **CPE** associado ao **Ponto de Instalação** após efetuadas as importações de faturas. Nesta listagem encontram-se detalhados o **nº de Documento** da Fatura da qual o **Alerta** é originário, o **Tipo de Alerta** e os **Detalhes** do mesmo. É também possível adicionar comentários meramente informativos, por parte do utilizador.

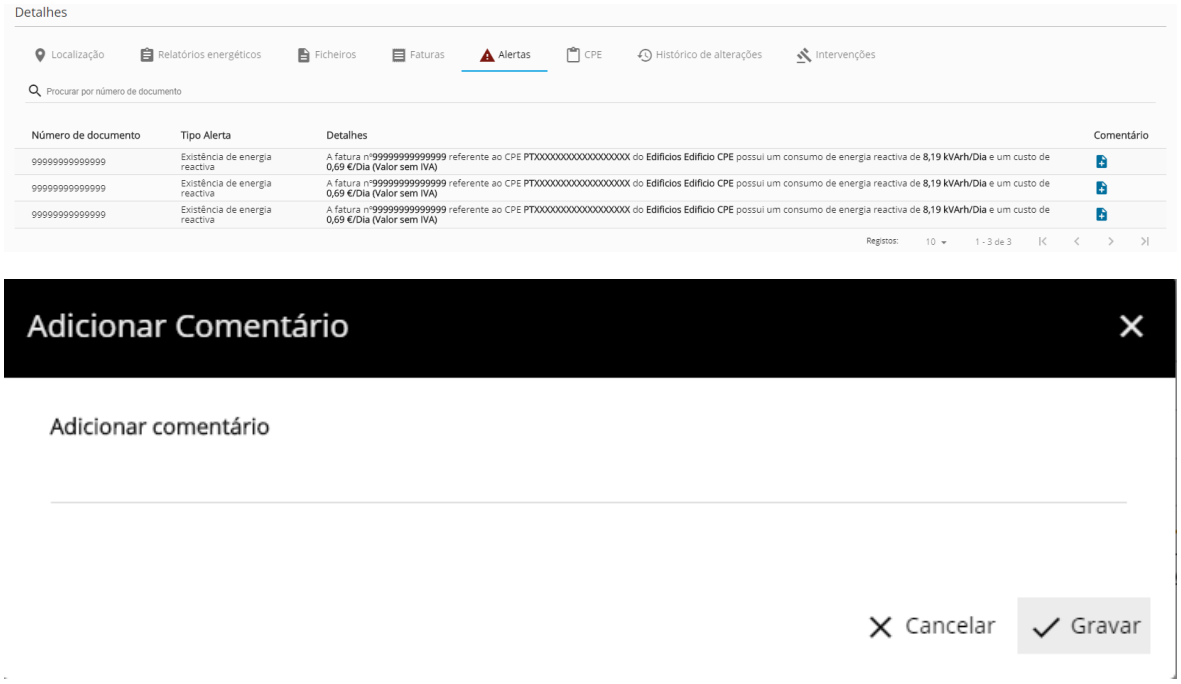


Figura 24 - Separador de alertas gerados aquando das importações de faturas e janela modal para adição de comentários aos mesmos

- **CPE** – Este separador funciona como uma listagem meramente informativa de todos os **CPEs** registados no **Ponto de Instalação.**

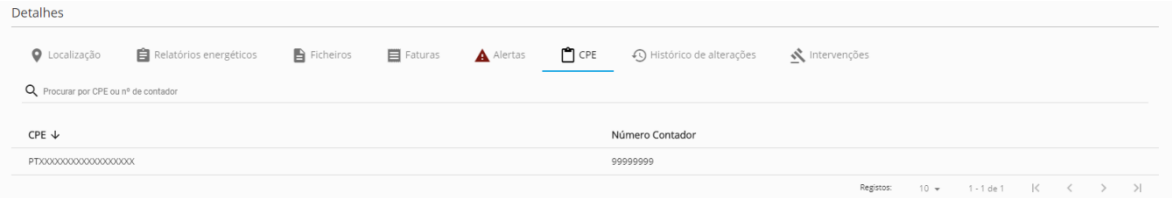


Figura 25 - Separador CPE onde se encontram listados todos os CPEs pertencentes ao Ponto de Instalação

- **Histórico de Alterações** – Tal como o nome indica, este separador lista todas as

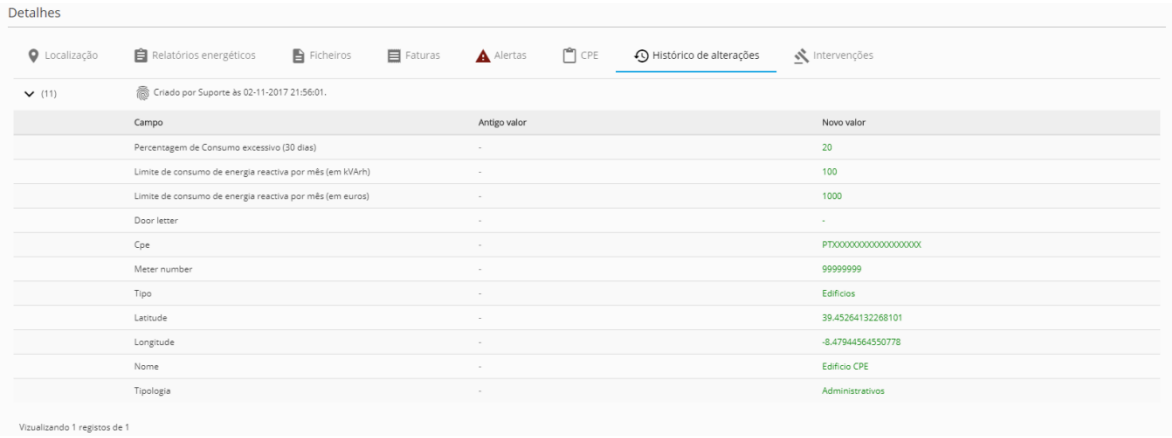


Figura 26 - Separador Histórico de Alterações onde são listadas todas as alterações feitas ao Ponto de Instalação

alterações feitas ao **Ponto de Instalação.**

- **Intervenções** – Este separador tem como objetivo a listagem de todas as **Intervenções** registadas no **Ponto de Instalação**. Para além disso é também possível editar ou apagar qualquer registo de **Intervenção**.

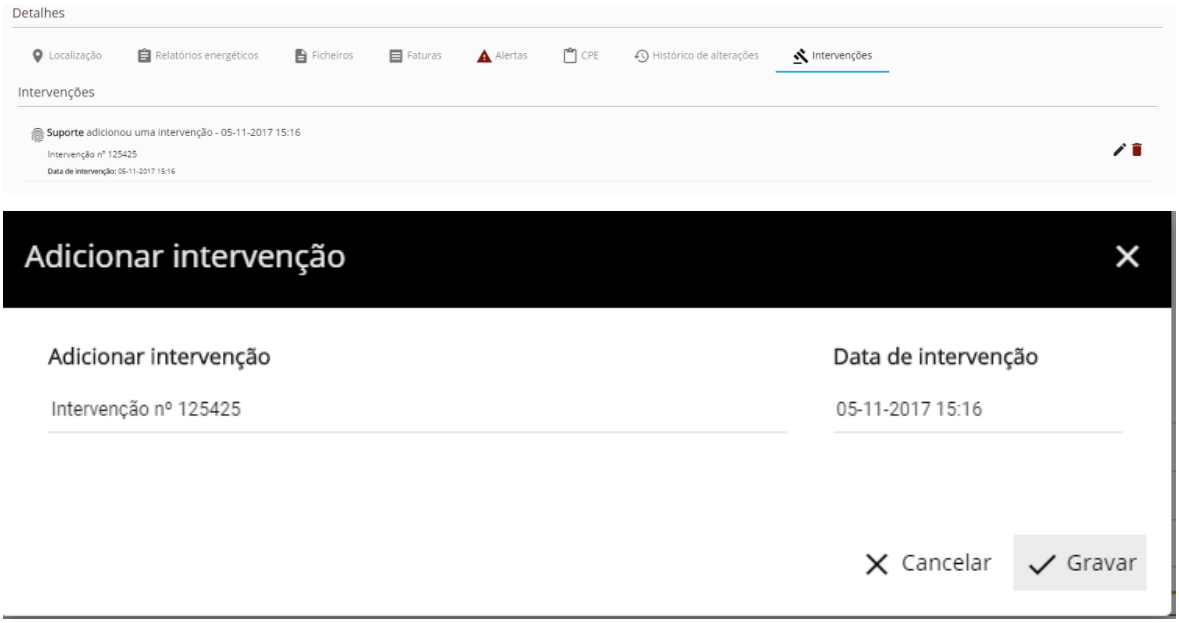


Figura 27 - Separador de Intervenções onde são listadas todas as intervenções registadas no Ponto de Instalação e Janela Modal da sua edição.

7. Conclusão

O estágio curricular é de grande importância na integração de qualquer estudante na vida profissional, na medida em que representa, na maioria dos casos, o primeiro contacto com o mercado de trabalho. A empresa ou organização em que um estagiário é acolhido é um local onde este deverá ser integrado e os seus conhecimentos colocados em prática durante o período de contacto.

Apesar do meu estágio curricular ter começado em outubro de 2016, eu já mantinha contacto com a empresa *Compta Emerging Business* há cerca de um ano, pois foi nesta empresa que realizei o meu Estágio Profissional após a conclusão da Licenciatura em Engenharia Informática. No entanto, o facto de já estar integrado e familiarizado com o ambiente e ferramentas utilizadas nos desenvolvimentos feitos pela equipa *R&D*, não invalida todo o conhecimento e técnicas que vim a adquirir e dominar, ao longo destes nove meses de estágio.

Relativamente ao tema abrangido por este relatório, considero que os objetivos propostos aquando do início do projeto para a aplicação de **Gestão de Faturas** foram atingidos, tanto por mim como por todos os elementos da equipa da qual fiz parte.

De todas as tarefas e desenvolvimentos que realizei, foi a etapa de importação de faturas que mais dificuldades trouxe à conclusão desta aplicação, devido à complexidade do processo em si e à complexidade que este exige em termos de implementação. Foi durante esta etapa do desenvolvimento que houve mais comunicação e troca de ideias, pelo que, foi adquirido um maior nível de cumplicidade para com os meus colegas de trabalho que, apesar de pertencentes a outros polos, se demonstraram sempre disponíveis. Foi também durante este estágio e todo o tempo que passei com a equipa de *R&D* que vim a constatar e compreender melhor a importância do trabalho em equipa no que respeita a desenvolvimentos de aplicações de grande escala. Escusado será dizer que sem tais características, o cumprimento e conclusão deste projeto não seriam possíveis.

Por fim, resta-me apenas referir que, apesar de concluída esta etapa do projeto, já existem planos e ideias para o levar mais além do seu estado atual, nomeadamente através da

aplicação de Inteligência Artificial. Esta futura implementação irá permitir à aplicação ser mais inteligente e eficaz na disponibilização e processamento de informação, providenciando assim uma melhor experiência ao utilizador.

8. Referências Bibliográficas

[1] *Kehoe, D. (2014). Learn Ruby on Rails: Book One. RailsApps.*

[2] *Laurent, S. Dumbill, E. Locklear, M. (2016). Learning Rails 5: Rails from the Outside In. O'Reilly Media.*

[3] *Ruebbelke, L. (2015) AngularJS in Action. Manning Publications.*

[4] *Seshadri, S. (2014). AngularJS: Up and Running: Enhanced Productivity with Structured Web Apps. O'Reilly Media.*