EMANUEL FILIPE CORREIA LOURENÇO

PLATAFORMA INTELIGENTE DE ORÇAMENTAÇÃO

UNIVERSIDADE DO ALGARVE

INSTITUTO SUPERIOR DE ENGENHARIA

DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA

2019

EMANUEL FILIPE CORREIA LOURENÇO

PLATAFORMA INTELIGENTE DE ORÇAMENTAÇÃO

Mestrado em Engenharia Elétrica e Eletrónica

Tecnologias da Informação e Telecomunicações

Trabalho efetuado sob a orientação de:

Professor Doutor Pedro J. S. Cardoso

Professor Doutor Roberto Lam

UNIVERSIDADE DO ALGARVE

INSTITUTO SUPERIOR DE ENGENHARIA

DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA

2019

PLATAFORMA INTELIGENTE DE ORÇAMENTAÇÃO

Declaração de autoria de trabalho

Declaro ser o autor deste trabalho, que é original e inédito. Autores e trabalhos consultados estão devidamente citados no texto e constam da listagem de referências incluída.

Faro, ____ de _____ de 2019

_____

(Emanuel Filipe Correia Lourenço)

AGRADECIMENTOS

Gostaria de agradecer aos meus orientadores, Professor Doutor Pedro J. S. Cardoso e Professor Doutor Roberto Lam por toda a disponibilidade e ajuda que me deram durante esta etapa. Quero também agradecer à minha família e amigos pelo apoio que me deram, em especialmente à Adriana Santos pelas revisões incansáveis ao longo da elaboração deste trabalho.

RESUMO

Existem diversos programas de faturação no mercado, no entanto, na sua generalidade apenas mostram o nome, uma breve descrição do produto, quantidades e valores dos produtos, sendo essa informação para um cliente normal pouco percetível. Desta forma, e face à necessidade de uma empresa em obter uma ferramenta que permita elucidar os clientes sobre os detalhes dos produtos que lhes estão a propor, nasceu a ideia deste projeto, uma aplicação de orçamentação que permita agilizar todo o processo referente à elaboração de propostas/orçamentos, ao mesmo tempo que permite apresentar uma descrição mais ampla dos produtos propostos.

Esta plataforma irá permitir detalhar mais pormenorizadamente a informação dos produtos propostos ao cliente, e agilizar o sector da orçamentação. Deste modo a aplicação permite efetuar mais facilmente as propostas, obtendo ao mesmo tempo as suas margens de lucro, evitando com isto as folhas de cálculo utilizadas pela grande maioria das empresas e a criação/edição de documentos de texto que seriam posteriormente utilizadas para transmitir a proposta ao cliente. Ao evitar que os funcionários da empresa tenham que efetuar os cálculos das margens e posteriormente elaborar a proposta, permite a elaboração de mais propostas no mesmo período de tempo.

O facto da aplicação ser desenvolvida numa plataforma web, permite a realização dos orçamentos e consultas das diversas informações, sejam efetuados em qualquer lugar, desde que o equipamento do utilizador contenha acesso à internet. Além das habituais informações das propostas, fornecedores, clientes e produtos, esta aplicação irá permitir a visualização do histórico das propostas efetuadas.

Ao longo deste relatório será descrito o funcionamento de uma empresa particular, estando o seu negócio relacionado com o fornecimento e montagem de sistemas de instalações técnicas, as ferramentas utilizadas na implementação da aplicação de orçamentação e a forma como foi implementada.

A elaboração dos orçamentos é auxiliada por um sistema de recomendação de modo a agilizar a elaboração das propostas. Para se compreender o ser funcionamento, serão expostos alguns dos métodos existentes e o método adotada no projeto.

Palavras chave: Orçamentação, Sistema de Recomendação, Laravel, Base de Dados.

ABSTRACT

There are several billing programs in the market, however, overall they only show the name, a brief description of the product, quantities and values of the products, thought this sort of information is barely perceptible by a normal customer. Thus, given the need for a company to acquire a tool to enlighten customers about the details of the products they are being proposed, the idea of this project was formulated: a budgeting application that allows to streamline the whole process related to the elaboration of proposals / budgets, while allowing a broader description of the proposed products.

This platform will allow for a more in-depth specification of the information on the products offered to the client and streamline the budgeting sector. Therefore, this application allows to elaborate proposals in an easier manner, obtaining at the same time profit margins, thus avoiding not only the worksheets used by the great majority of companies but also the creation / editing of text documents later used to convey the proposal to the client. By preventing the company employees from having to perform margin calculations and subsequently write the proposal, allows for the development of more proposals in the same period of time.

The fact that the application is developed in a web platform, allows the elaboration of the budgets and consultations of the diverse information, to be carried out in any place, as long as the equipment of the user has access to an internet connection. In addition to the usual information of proposals, suppliers, customers and products, this application will allow the visualization of the history of the proposals that were carried out.

Throughout this report it will be addressed the operation of a particular company in which its business is related to the supply and assembly of technical installations systems. There will also be a focus on the tools used in the budgeting and how it was implemented.

Budgeting is assisted by a recommendation system to expedite the preparation of proposals. In order to understand how it works, some of the existing methods and the method adopted in the project will be exposed.

Key words: Budgeting, Recommendation System, Laravel, Database.

ÍNDICE GERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

ÍNDICE DE EQUAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

ACID Atomicidade, Consistência, Isolamento e Durabilidade (*Atomicity, Consistency, Isolation and Durability*)

AJAX Javascript Assíncrono e XML (*Asynchronous Javascript and XML*)

CASE *Computer-Aided Software Engineering*

CSS *Cascading Style Sheets*

DFD Diagrama de Fluxo de Dados (*Data Flow Diagram*)

DER Diagrama de Entidade e Relacionamento (*Entity Relationship Diagram*)

FDD *Functional Decomposition Diagram*

FN Forma Normal

HTML *HyperText Markup Language*

MVC *Model – View – Controller*

MVP *Model – View – Presenter*

MVVM *Model – View – View-Model*

NIF Número de Identificação Fiscal

PHP *PHP: Hypertext Preprocessor*

RegEx Expressão Regular (*Regular Expression*)

SQL *Structured Query Language*

POO Programação Orientada a Objetos (*Object-oriented programming*)

1. INTRODUÇÃO

1.1. CONTEXTO

Este projeto enquadra-se na necessidade de uma empresa, e à qual por razões de privacidade não irá ser revelada, em obter uma ferramenta que permita agilizar o processo de orçamentação. Ir-se-á nesse enquadramento desenvolver uma aplicação que permita facilitar o trabalho dos funcionários do departamento de orçamentação, e o tempo de resposta aos clientes. Com esta aplicação de orçamentação quer-se ainda permitir aos funcionários do departamento comercial e do técnico a obtenção de informações relacionadas com determinados clientes e fornecedores, nomeadamente os seus contactos, informações contidas nas propostas bem como informações técnicas dos produtos.

Atualmente esta empresa utiliza uma ferramenta obsoleta que apenas permite a criação de números de propostas, e da respetiva revisão, para o orçamento de forma automática. Consoante o *template* selecionado, essa ferramenta cria um documento de texto ou de cálculo previamente definido, estando estes *templates* bastante desatualizados. Para a criação dessa mesma proposta e consequentemente obter o referido número de proposta, é necessário o utilizador inserir manualmente todos os dados do cliente, como o nome do cliente, número de identificação fiscal – NIF, morada, dados de contacto e a que categoria a proposta se enquadra, sempre que cria uma proposta, independentemente se o cliente for novo ou existente. Deste modo os dados do cliente poderão estar inseridos de diversas maneiras, dificultando uma futura consulta de uma determinada proposta, ou até mesmo do histórico de propostas realizadas para o mesmo cliente.

1.2. OBJETIVO

Tendo em conta que o objetivo deste projeto é a implementação de uma aplicação que permita facilitar e agilizar todo o processo referente à orçamentação, esta aplicação deverá permitir: fazer a gestão de contactos de clientes e fornecedores; elaborar orçamentos e respetivas revisões suportados em dados de preços de mão-de-obra e matérias-primas existentes na base de dados, exportando depois a proposta no formato PDF; permitir o acesso a múltiplos utilizadores à aplicação, restringindo o acesso a determinada informação dependendo do cargo/função dos utilizadores; dar a possibilidade de os utilizadores obterem cronogramas e relatórios referentes aos orçamentos efetuados. Ainda de forma a auxiliar o orçamentista no momento de efetuar os orçamentos, será

implementado um sistema de recomendação, para recomendar novos produtos a incluir na proposta, e possivelmente poderá recomendar também produtos que existam em *stock,* desde que estes se enquadrem nos sistemas propostos. Devido às diferentes tecnologias disponíveis atualmente, a aplicação será desenvolvida para ser utilizada tanto em dispositivos fixos como computadores e nos dispositivos móveis como tablets e/ou smartphones.

1.3. ESTRUTURA

Este relatório encontra-se estruturado em seis capítulos. Neste capítulo foi abordado o tema do relatório, os problemas que originaram este projeto e os seus objetivos.

No entanto, antes de implementar a aplicação é necessário compreender o funcionamento dos departamentos da empresa, e posteriormente passar à implementação das funcionalidades requeridas, evitando com isto erros de implementação ou até a criação funcionalidades desajustadas às reais necessidades.

No Capítulo 2 será abordado o funcionamento do departamento de vendas, através de algumas das metodologias CASE – *Computer-Aided Software Engineering* [1, 2, 3], para análise e especificação do sistema de informação. O funcionamento será descrito em primeiro lugar pelo modelo de negócio, passando depois para o modelo funcional e terminando no modelo de relacionamento.

O Capítulo 3 será iniciado com uma breve explicação sobre o modelo de dados e a normalização da base de dados, e posteriormente a sua implementação. Devido à dimensão da base dados a implementação será abordada faseadamente.

Ao longo do Capítulo 4 serão exemplificadas as ferramentas utilizadas para a implementação do projeto, forma como foram aplicadas e qual a sua função.

No Capítulo 5 existirá uma breve abordagem a alguns dos diferentes tipos de sistemas de recomendação existentes. Serão apresentados alguns exemplos do sistema adotado e o motivo pelo qual foi adotado.

Por último, no Capítulo 6, será efetuada uma reflexão sobre o trabalho efetuado, bem como das funcionalidades que ainda poderão ser implementados no futuro, de modo a melhorar a aplicação.

2. ANÁLISE DO SISTEMA

Neste capítulo será efetuado uma análise ao sistema de gestão de vendas, com a finalidade de estudar o funcionamento da empresa de fornecimento e montagem de sistemas de instalações técnicas, para a qual a aplicação será desenvolvida. Com base na informação obtida através dessa análise, esta será organizada e posteriormente irá dar origem às funcionalidades que serão implementadas na aplicação de orçamentação, seguindo as regras de negócio. Esta informação será complementada com recurso às metodologias de análise utilizando as ferramentas CASE, através dos modelos de negócio, modelo funcional e modelo de relacionamento, de modo a que a informação obtida seja mais facilmente analisada e interpretada.

2.1. MODELO DE NEGÓCIO

No início desta análise é imprescindível entender o funcionamento do sistema de gestão de vendas da empresa. Para simplificar a sua análise deverá começar-se por ilustrar o modelo de negócio. Este modelo consiste em decompor os processos da empresa em processos mais simples e de melhor perceção [1, 2, 3]. Este processo é normalmente associado a uma abordagem do tipo *Top-Down*, isto é, inicia-se com uma abordagem mais ampla do sistema retratado, passando depois sucessivamente para processos mais detalhados.

Na Figura 2.1 encontra-se ilustrado o diagrama de objetos referente à estrutura do sistema de gestão de vendas. Nesta figura é possível reparar que este modelo começa por ilustrar o processo do sistema de Gestão de Vendas de forma mais ampla passando posteriormente para os subsistemas que o constituem.

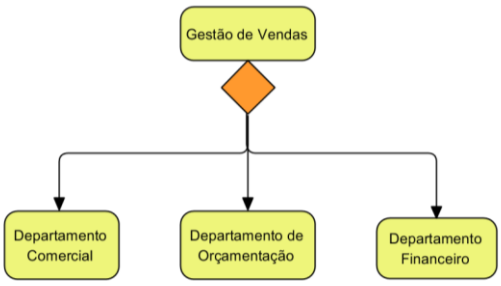


Figura 2.1: Estrutura organizativa do sistema de gestão de vendas

Da análise ao sistema de gestão de vendas foi possível observar a existência de três unidades organizativas pelo qual este é constituído, nomeadamente pelo Departamento Comercial, o Departamento de Orçamentação e o Departamento Financeiro.

Para compreender o funcionamento de cada uma destas unidades será efetuado um breve resumo das funções de cada departamento.

O Departamento Comercial é responsável por angariar possíveis novos clientes e negociar com os atuais. Este departamento faz de elo entre o cliente e a empresa, transmitindo aos restantes departamentos toda a informação necessária para a elaboração das propostas de equipamentos e/ou assistência técnica, instalação dos equipamentos e faturação da proposta.

O Departamento de Orçamentação é responsável por efetuar as propostas, mediante as solicitações dos clientes, podendo este ser o cliente final ou revendedor. Os conteúdos para as propostas solicitadas podem chegar através de levantamentos efetuados pelo Departamento Técnico, informação transmitida pelo Departamento Comercial, de cadernos de encargos ou conforme solicitação dos clientes. É igualmente da responsabilidade dos elementos do Departamento de Orçamentação procurar novos fornecedores nas situações em que os atuais não forneçam os produtos necessários, ou quando o valor dos produtos não se enquadra nos orçamentos dos clientes;

Por último, o Departamento Financeiro é responsável por toda a parte financeira da empresa, deste pagamento de salários aos funcionários da empresa, a fornecedores, bem como verificar a situação se os clientes e se estes não contêm nenhuma divida à empresa. Em caso de existirem situações por regularizar, este departamento transmite essa informação aos elementos do Departamento Comercial.

2.2. MODELO FUNCIONAL

O modelo funcional inicia-se retratando de forma abrangente o sistema de gestão de vendas, a troca de informação entre o sistema de gestão de vendas e as entidades externas. O diagrama de fluxo de dados, DFD, de nível zero e que representa todo o sistema em um único processo é denominado de diagrama de contexto.

Uma das vantagens do processo de análise é perceber o funcionamento do sistema, ilustrando as trocas de informação e assegurando que o sistema seja de fácil compreensão.

Neste DFD existe dois tipos de entidades: as entidades internas que se denominam por processos, que neste caso serão considerados o Departamento Comercial, Departamento de Orçamentação e o Departamento Financeiro, e as entidades externas tais como clientes, fornecedores e Departamento Técnico. Apesar destas entidades serem consideradas como

externas, não necessitando que tenham de ser externas à empresa como, por exemplo, o Departamento Técnico.

O DFD é composto por quatro componentes: a) os processos; b) os fluxos de dados que representam a troca de informação; c) os arquivos que são a informação guardada, podendo ser em arquivo físico em formato papel ou em arquivo virtual numa base de dados; e d) as entidades externas.

Existem diversas metodologias disponíveis para representação dos diagramas, tais como *Yourdon Systems Method, STRADIS e SSADM,* estando estas metodologias descritas por exemplo em [2]. A metodologia SSADM será a adotada neste trabalho para a representação dos diagramas, utilizando a simbologia na Figura 2.2.

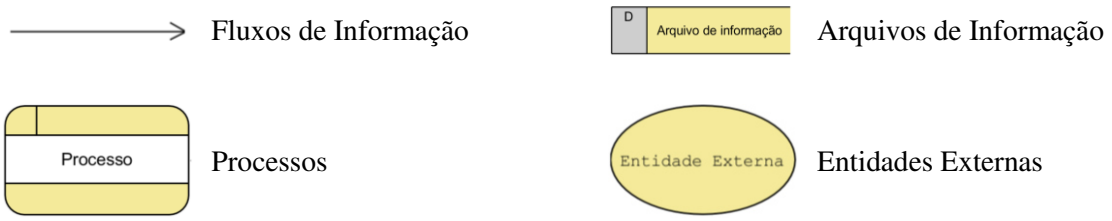


Figura 2.2: Símbolos da metodologia SSADM

Na Figura 2.3 pode-se observar a troca de informação entre o processo do sistema de gestão de vendas e as unidades externas. Como unidades externas observa-se a existência de fornecedores e clientes, que são elementos externos ao sistema de gestão de vendas e à empresa. Já o Departamento Técnico apenas é externo ao sistema de gestão de vendas, uma vez que se trata de um departamento interno da empresa em análise.

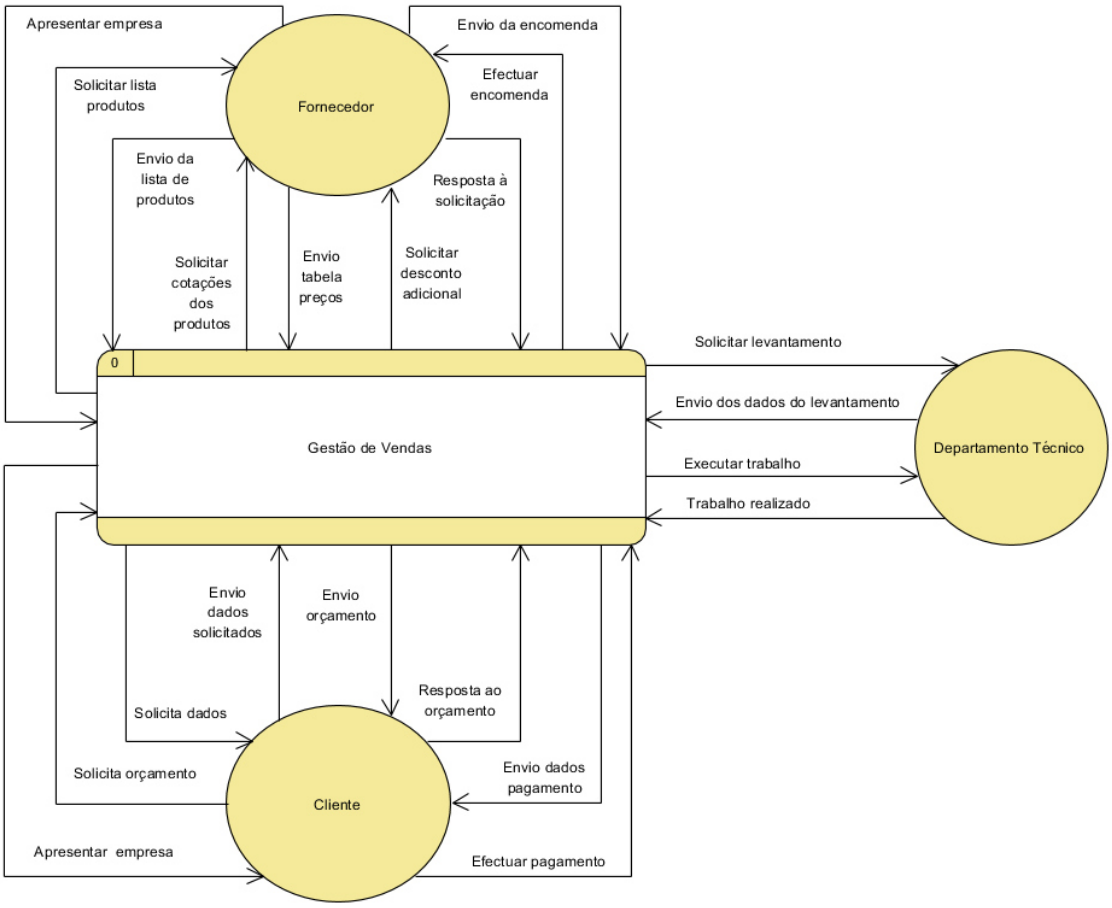


Figura 2.3: Diagrama de Contexto

De seguida serão detalhados os próximos níveis do DFD, demostrando o funcionamento interno do sistema de gestão de vendas. Existem os mesmos elementos externos do diagrama de contexto, ou seja, clientes, fornecedores e Departamento Técnico. No entanto, em substituição do processo do sistema de gestão de vendas surgem os três departamentos que o constituem, Departamento Comercial, Departamento Financeiro e Departamento de Orçamentação.

Na Figura 2.4 encontra-se exemplificado o primeiro nível do DFD, mostrando de forma global como a informação é trocada entre os elementos internos do sistema.

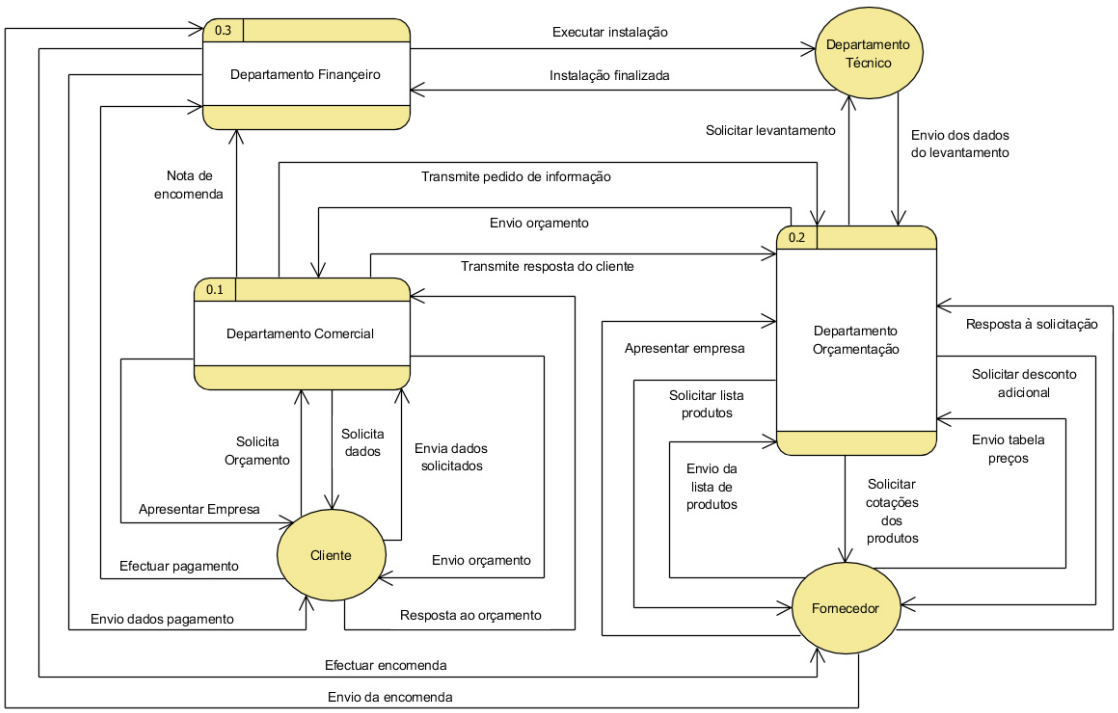


Figura 2.4: DFD – Primeiro Nível

Após finalizado a análise do primeiro nível do diagrama, ver Figura 2.4, é possível decompor outros níveis para cada um dos subsistemas internos. No entanto, não é recomendado a utilização de demasiados níveis, pois tal situação poderia por em causa a perceção do sistema devido à excessiva abstração, uma vez que a pessoa poderá perder a noção de qual o processo que está a ser analisado. Como tal, será apenas ilustrado mais um nível para cada um dos subprocessos.

Na Figura 2.5, o segundo nível do DFD, pode-se visualizar a troca de informação relativo ao Departamento Comercial. Este departamento faz a ligação entre a empresa e o cliente. Encontrando-se o Departamento de Orçamentação e o Departamento Financeiro representados como entidades externas, pelo facto de o diagrama da Figura 2.5 corresponder ao Departamento Comercial.

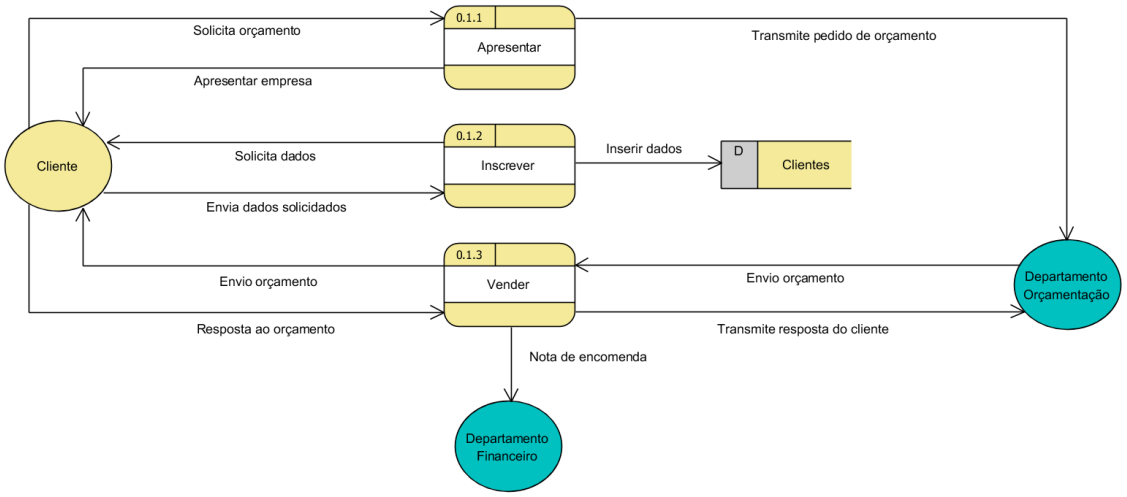


Figura 2.5: DFD – Segundo Nível – Departamento Comercial

No diagrama da Figura 2.5, observa-se a existência de três processos. No processo 0.1.1 o Departamento Comercial vai ao encontro de potenciais clientes, exibindo a carta de apresentação da empresa onde constam as áreas de negócio onde a empresa trabalha. O cliente caso esteja interessado poderá solicitar um orçamento nas áreas apresentadas.

Quando os clientes solicitam uma cotação e antes de ser possível satisfazer as suas necessidades, é requerido aos clientes os seus dados para ser criada a ficha, como representado no processo 0.1.2.

Por fim, no processo 0.1.3, o Departamento Comercial procede ao reencaminhamento do orçamento para o cliente, ficando posteriormente a aguardar por uma resposta deste. O cliente por sua vez, e após visualizar a proposta, poderá decidir por avançar tal como esta foi apresentada, rejeitar ou pedir para renegociar. Neste último caso, e se for possível satisfazer as necessidades do cliente, é lhe enviada uma nova proposta.

Após ser informado da decisão do cliente, o Departamento Comercial informa o Departamento de Orçamentação sobre a decisão deste. Em caso de aceitação do orçamento, o Departamento Comercial procede ao envio da nota de encomenda para o Departamento Financeiro, para que este último possa dar seguimento ao procedimento.

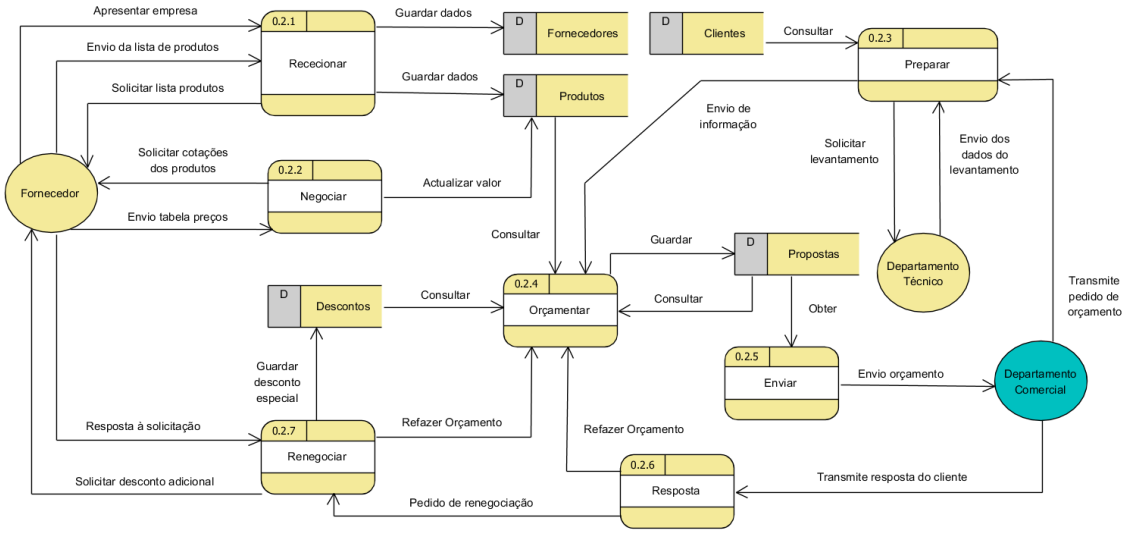


Figura 2.6: DFD – Segundo Nível – Departamento de Orçamentação

O segundo nível do diagrama referente ao Departamento de Orçamentação, Figura 2.6, é o mais complexo dos três. Este nível conta com sete processos, em que os processos 0.2.4 a 0.2.7 poderão ser executados mais do que uma vez.

O processo 0.2.1 é iniciado com a apresentação dos fornecedores, facultando os seus dados e produtos. Caso estes se enquadrem nas mesmas áreas da empresa, é solicitado uma lista de produtos. A informação referente aos produtos enviada pelo fornecedor será guardada para posteriormente ser consultada. No processo 0.2.2, os produtos serão alvo de negociação com os respectivos fornecedores. Esta negociação incide no valor de venda dos produtos e suas margens de venda.

Após o Departamento de Orçamentação receber a listagem dos produtos, estes são guardados, sendo actualizados sempre que existir uma alteração de valor.

O processo 0.2.3 deste DFD é iniciado após o Departamento de Orçamentação receber informação do Departamento Comercial. Esta informação transmitida servirá para a elaboração da proposta solicitada pelo cliente. O Departamento de Orçamentação em caso de dúvida, e de forma a evitar potênciais erros ou orçamentar equipamentos que não sejam adquados à situação, poderá solicitar ao Departamento Técnico uma análise do equipamento a propor e as suas quantidades.

Depois de reunir toda a informação relativa aos dados do cliente, produtos a orçamentar e respectivos valores, é iniciado o orçamento da proposta, processo 0.2.4. Após o orçamento estar concluido é enviado para apreciação do Departamento Comercial,

processo 0.2.5. Caso este departamento aprove o orçamento, este será reencaminhado para o cliente, como demostrado na Figura 2.5, processo 0.1.3.

Após a obtenção da resposta do cliente, o Departamento Comercial informa o Departamento de Orçamentação sobre a decisão do cliente, processo 0.2.6. No caso do cliente requerer uma renegociação da proposta, e em casos excepcionais, poderá ser necessário recorrer ao fornecedor para que este atribua um desconto adicional, como ilustrado no processo 0.2.7. Sempre que o cliente solicitar uma nova proposta, repetem-se os processos 0.2.4 a 0.2.7 e, em certos casos, poderá ser necessário repetir igualmente o processo 0.2.3.

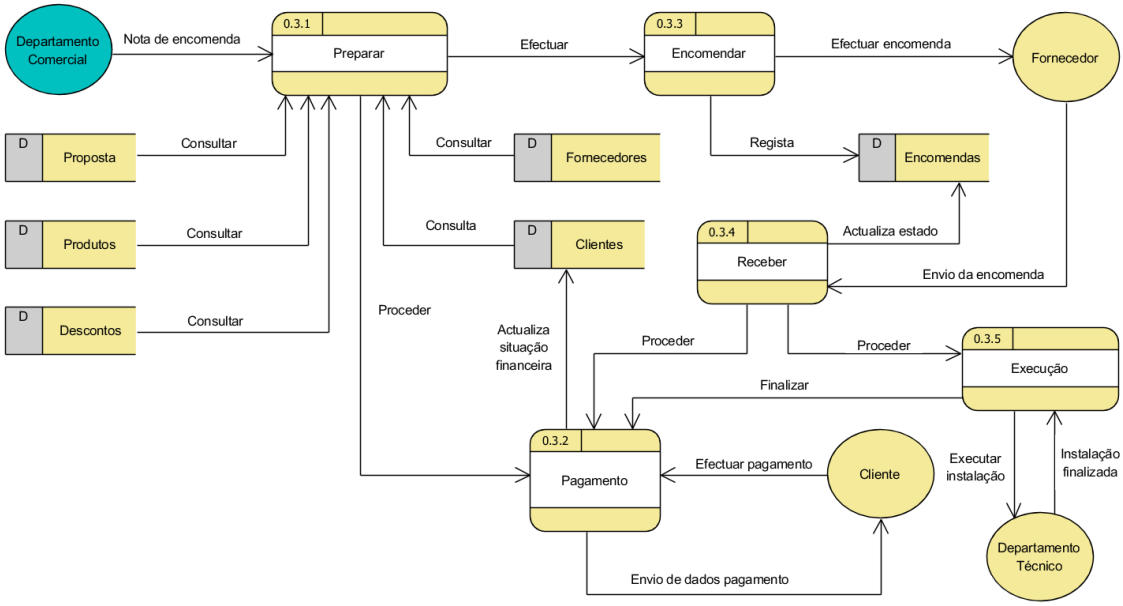


Figura 2.7: DFD – Segundo Nível – Departamento Financeiro

Finalmente, observa-se o segundo nível do DFD referente ao Departamento Financeiro, Figura 2.7. Este começa pelo processo de preparação 0.3.1, que é quando este recebe a nota de encomenda por parte do Departamento Comercial. Este processo inicia com o Departamento Financeiro a consultar os dados da proposta, selecionando o melhor fornecedor para cada um dos produtos em questão. Depois de analisado o melhor fornecedor será possível emitir a nota de encomenda. Ao mesmo tempo, serão vistos os termos da proposta referentes às fases de pagamento. Existindo alguma parcela referente à adjudicação da proposta, a encomenda aos fornecedores só será efectuada após liquidação desses mesmos valores, processo 0.3.2.

Após a conclusão das fases de preparação e pagamento, processos 0.3.1 e 0.3.2, é então processada a encomenda dos produtos aos respectivos fornecedores, processo 0.3.3, ficando depois a aguardar o envio dos mesmos.

Com a recepção das encomendas, processo 0.3.4, e depois de conferido todo o material, é iniciado o processo 0.3.5. Este processo consiste em informar o Departamento Técnico que o material necessário para a execução dos trabalhos já está disponível, e o mesmo deverá informar quando o trabalho já se encontrar realizado.

A proposta pode conter várias condições de pagamento como, por exemplo, pagamanto faseado com o decorrer da instalação. Desta forma o processo de pagamento será efectuado diversas vezes.

2.3. MODELO RELACIONAL

Na maioria dos sistemas de informação/aplicações requer uma base de dados para armazenar toda a informação, e como tal será imprescindível efetuar um estudo prévio da forma como esta deve ser implementada.

Uma das vantagens do modelo relacional é a representação simples dos dados, bem como a facilidade com que permite efetuar consultas complexas na base dados.

Com base na análise efetuada nos modelos anteriores é possível observar a relação existente entre as principais entidades, representadas no diagrama da Figura 2.8, Diagrama de Entidades e Relacionamentos – DER [4].

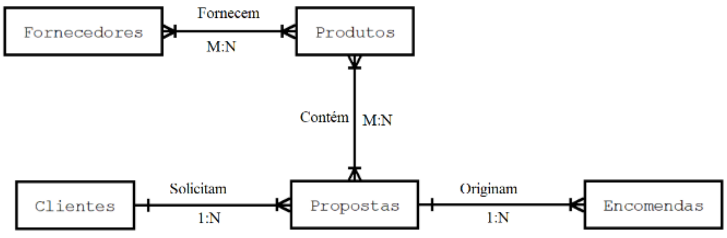


Figura 2.8: Relacionamentos entre as principais entidades

Neste modelo relacional serão descritas as entidades [4, 5], podendo estas ser classificadas de físicas ou lógicas, e as suas relações. As entidades físicas são geralmente associadas a objetos existentes e visíveis, tais como, um cliente, fornecedor ou um

produto. Já as entidades lógicas são aquelas não são objetos físicos, tal como uma venda ou uma classificação.

Num modelo relacional, o relacionamento existente entre as tabelas permitem restringir a existência de dados redundantes/informação duplicada. Para a realização da correspondência entre entidades, os atributos não necessitam de conter o mesmo nome, no entanto os tipos de dados desses campos devem permanecer os mesmos, podendo, contudo, existir exceções no campo chave, ou seja, nas situações em que este é do tipo inteiro com numeração automática. Por exemplo, o atributo "tipo_id" da tabela "Produtos", Tabela 2.2, que faz referência ao atributo "id" da tabela "Tipos de Produto", Tabela 2.3.

O relacionamento entre entidades podem ser classificados de três formas [5], um para um – 1:1, um para muitos – 1:N e muitos para muitos – M:N.

Diz-se que existe um relacionamento de 1:1, representado na Figura 2.9, quando um registo na "Tabela A" corresponde apenas a um registo na "Tabela B" e vice-versa. Nas Tabelas 2.1 e 2.2 encontra-se representado um exemplo do tipo de relacionamento 1:1, onde cada cliente contém uma única identificação – Tabela 2.1, e cada identificação corresponde a apenas um cliente – Tabela 2.2. No entanto poderá existir situações em que não existe qualquer correspondência dos registos inclusos na "Tabela A" com os da "Tabela B".

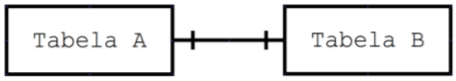


Figura 2.9: Representação de Relacionamento 1:1

id	nome	identificação_id
1	Cliente A	1
2	Cliente B	2
3	Cliente C	3
4	Cliente D	4

Tabela 2.1: Clientes – Relacionamento 1:1

id	identificacao
1	NIF A
2	NIF B
3	NIF C
4	NIF D

Tabela 2.2: Identificações – Relacionamento 1:1

Já para existir um relacionamento 1:N, um registo da "Tabela A" deverá corresponder a um ou mais registos na "Tabela B", representado na Figura 2.10, no entanto, tal como relacionamento anterior, poderá existir situações em que não existe qualquer correspondência dos registos inclusos na "Tabela A" com os da "Tabela B". Na situação

inversa, um registo na “Tabela B” apenas poderá corresponder a um registo na “Tabela A”.

Na Tabela 2.3 é possível visualizar uma listagem de produtos e das respetivas descrições, originada pela união da tabela de produtos, Tabela 2.4, com a tabela de descrições, Tabela 2.5. Ainda na Tabela 2.3 verifica-se a existência de diversos produtos com mais do que uma descrição, no entanto cada descrição apenas se refere a um produto.

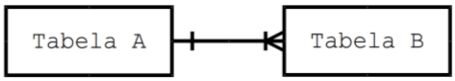


Figura 2.10: Representação de Relacionamento 1:N

id	nome	descricao
1	Produto A	Descrição A
2	Produto A	Descrição B
3	Produto B	Descrição C
4	Produto B	Descrição D
5	Produto C	Descrição E
6	Produto C	Descrição F

Tabela 2.3: Exemplos de tabelas com relacionamento de 1:N

id	nome
1	Produto A
2	Produto B
3	Produto C
4	Produto D

Tabela 2.4: Tabela de Produtos - Relacionamento 1:N

id	descricao	produto_id
1	Descrição A	1
2	Descrição B	1
3	Descrição C	2
4	Descrição D	2
5	Descrição E	3
6	Descrição F	3

Tabela 2.5: Tabela Tipos de Produto - Relacionamento 1:N

Por último existe o relacionamento M:N, representado na Figura 2.11, que significa que um registo na “Tabela A” corresponde a vários registos na “Tabela B”, e um registo na “Tabela B” também corresponde a vários registos na “Tabela A”, podendo contudo existir situações em que os registos presentes nas tabelas A e/ou B não contenham qualquer correspondência na outra tabela. Neste tipo de relacionamento é necessário proceder à criação de uma tabela intermédia, representado na Figura 2.12, onde serão relacionados os registos pertencentes à tabela A com os registos da tabela B.

Analisando o exemplo ilustrado na Tabela 2.6, originada pelas Tabelas 2.8 e 2.9, observa-se que existem vários produtos com diferentes tipos e o mesmo tipo com vários produtos.

Deste modo, para evitar a duplicação de registos, é necessário criar a tabela de relação que faz a correspondência dos produtos com os seus tipos, Tabela 2.7.

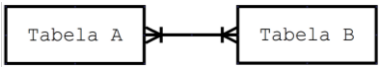


Figura 2.11: Representação de Relacionamento M:N

Figura 2.12: Representação de Relacionamento M:N com tabela intermédia

id	nome	nome_tipo
1	Produto A	Tipo A
2	Produto B	Tipo A
3	Produto B	Tipo B
4	Produto B	Tipo C
5	Produto C	Tipo C
6	Produto C	Tipo D
7	Produto D	Tipo A
8	Produto D	Tipo D

Tabela 2.6: Exemplos de tabelas – Relação M:N

produto_id	**tipo_id**
1	1
2	1
2	2
2	3
3	3
3	4
4	1
4	4

Tabela 2.7: Tabela de Relação entre Produtos e Tipos de Produto – Relação M:N

id	nome
1	Tipo A
2	Tipo B
3	Tipo C
4	Tipo D

Tabela 2.8: Tabela Tipos de Produto – Relação M:N

id	nome
1	Produto A
2	Produto B
3	Produto C
4	Produto D

Tabela 2.9: Tabela Produtos – Relação M:N

Concluída a análise ao sistema ao sistema de gestão de vendas, verificou-se que este era composto por três unidades organizativas, sendo estas o Departamento Comercial, o Departamento Financeiro e o Departamento de Orçamentação. Existindo diversas interpretações derivadas do mesmo sistema, iniciar-se-á no próximo capítulo o desenho da base dados para guardar toda a informação necessária com base no estudo efetuado neste capítulo.

3. DESENHO DO SISTEMA

Neste capítulo será abordado a forma como foi implementado o sistema de base dados da aplicação. Para que esta implementação seja efetuada de forma correta, será necessário conhecer as regras referentes à normalização [6].

Assim neste capítulo começar-se-á por referir algumas das formas de normalização. Após se conhecer as formas de normalização, será efetuado uma análise do sistema implementado, ilustrado o seu resultado através do desenho do sistema – Figura A.1, que derivado à sua dimensão estará ilustrado em blocos.

3.1. NORMALIZAÇÃO

Para a construção do diagrama de entidades e relacionamentos existem diversas formas de normalização, que para este projeto considerou-se até à 3ª Forma Normal (FN). Na Figura 3.1 encontra-se ilustrado a ordem com que é efetuada a normalização das tabelas, existindo mais formas para além da 3ª FN. No entanto as restantes formas normais não foram consideradas neste projeto, por se aplicarem a situações específicas.

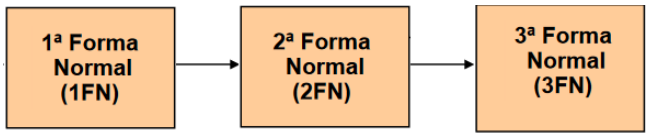


Figura 3.1: Normalização de Tabelas

Para se poder dizer que uma tabela se encontrar na 1ª FN é necessário que todos os atributos da tabela sejam atómicos, ou seja, cada atributo apenas deva conter um valor. Por exemplo, na Tabela 3.1 é possível observar que o atributo "contacto" contêm mais do que um valor, não cumprindo com isto a 1ª Forma Normal. Assim de forma a estar na 1ª FN é necessário proceder à divisão do conteúdo como, por exemplo, ilustrado na Tabela 3.2.

cliente_id	nome	contacto
1	Cliente A	{Contacto 1, Contacto 2}
2	Cliente B	{Contacto 3}
3	Cliente C	{Contacto 4, Contacto 5, Contacto 6}
4	Cliente D	{Contacto 7, Contacto 8}

Tabela 3.1: Exemplo de tabela não normalizada

cliente_id	nome	contacto
1	Cliente A	Contacto 1
1	Cliente A	Contacto 2
2	Cliente B	Contacto 3
3	Cliente C	Contacto 4
3	Cliente C	Contacto 5
3	Cliente C	Contacto 6
4	Cliente D	Contacto 7
4	Cliente D	Contacto 8

Tabela 3.2: Exemplo de tabela na 1ª FN

Nas tabelas existem alguns atributos considerados de chaves, que dependendo das suas funções podem ser considerados de "Chaves Primárias", "Super-Chaves", "Chaves Candidatas" ou "Chaves Estrangeiras".

Considera-se Super-Chave quando esta é capaz de identificar de forma única um tuplo da relação.

Para ser Chave Candidata é imprescindível que o conjunto de atributos que definem de forma única um tuplo na tabela seja mínimo, ou seja o conjunto de atributos não pode ser reduzido sem que este deixe de identificar de forma única um tuplo.

Chave Primária tem como objetivo identificar de forma única um tuplo da tabela, sendo escolhida de entre as Chaves Candidatas. A chave primária poderá ser constituída por mais do que um atributo.

No relacionamento entre tabelas, o atributo que identifica o registo de outra tabela denomina-se de Chave Estrangeira.

Depois de confirmado que os registos presentes na tabela se encontram na 1ª FN, pode-se passar para a verificação se está na 2ª FN.

Para se verificar que se encontra na 2ª FN é necessário que todos os atributos descritores, ou seja elementos que não pertencem à chave primária, dependam da totalidade da chave primária e não de parte dela.

Considere-se o diagrama de dependências funcionais apresentado na Figura 3.2 e um exemplo na Tabela 3.3, verifica-se que os atributos "nome_produto e "nome_idioma" não depende da totalidade da chave que é composta pelos atributos "produto_id" e "idioma_id", mas de parte desta, não cumprindo com isto a 2ª FN.

produto_id	nome_produto	idioma_id	nome_idioma	descricao
1	Produto A	1	Idioma A	Descricao A
1	Produto A	2	Idioma B	Descricao B
2	Produto B	1	Idioma A	Descricao C
2	Produto B	2	Idioma B	Descricao D
3	Produto C	2	Idioma B	Descricao E
4	Produto D	1	Idioma A	Descricao F
5	Produto E	1	Idioma A	Descricao G

Tabela 3.3: Exemplo de tabela não normalizada

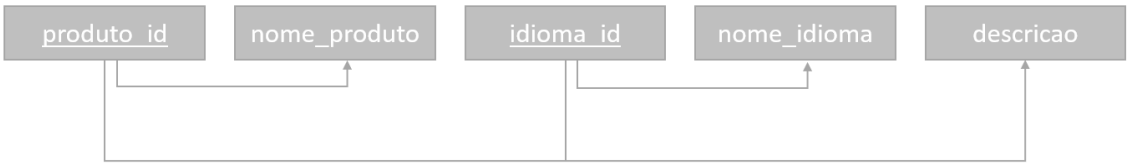


Figura 3.2: 2ª FN – Exemplo de diagrama de dependências

Para cumprir a 2ª FN é necessário separar da tabela os campos que não dependem completamente da chave, ficando com o equivalente às Tabelas 3.4, 3.5 e 3.6.

produto_id	nome_produto
1	Produto A
2	Produto B
3	Produto C
4	Produto D
5	Produto E

Tabela 3.4: Tabela de produtos

produto_id	idioma_id	descricao
1	1	Descricao A
2	2	Descricao B
3	1	Descricao C
4	2	Descricao D
5	2	Descricao E
6	1	Descricao F
7	1	Descricao G

Tabela 3.5: Tabela de descrições na 2ª FN

idioma_id	nome_produto
1	Idioma A
2	Idioma B

Tabela 3.6: Tabela de idiomas

Depois de confirmada a 2ª FN, passa-se para a 3ª FN. De modo a que uma tabela se encontre na 3ª Forma Normal, esta deve já estar na 2ª FN e todos os seus atributos apenas podem depender de atributos que sejam chaves primárias.

Observando o exemplo da Tabela 3.7 confirmar-se que todos os atributos ficam definidos pela chave "id", cumprindo a 2ª FN. No entanto, analisando o diagrama das dependências

funcionais associado à Tabela 3.7, Figura 3.3, verificar-se que o atributo "descricao" também depende funcionalmente do atributo "idioma_id", não cumprindo a 3ª FN.

id	referencia	descricao	idioma_id	estado_id	tipo _id
1	HAVR-0472H1	Servidor tríbrido de 4 canais	1	1	1
2	HAVR-0472H1	4-Channel hybrid server	2	1	1
3	AVN-71SMVR	Câmara analógica tipo dome para exterior	1	1	2
4	AWP-72SMVR	Câmara analógica tipo bullet para exterior	1	1	2
5	AU-S12	Fonte de alimentação de 2A	1	2	3
6	AU-S12	2A Power supply	2	2	3

Tabela 3.7: Exemplo de tabela não normalizada

Figura 3.3: 3ª FN – Exemplo de diagrama de dependências

Deste modo para cumprir a 3ª Forma Normal é necessário proceder à divisão da Tabela 3.7 em duas novas tabelas, obtendo-se então as tabelas "Produtos" e "Descrições de produto", Tabelas 3.8 e 3.9.

id	referencia	estado_id	tipo_id
1	HAVR-0472H1	1	1
3	AVN-71SMVR	1	2
4	AWP-72SMVR	1	2
5	AU-S12	2	3

Tabela 3.8: Tabela de produtos na 3ª FN

id	idioma_id	descricao
1	**1**	Servidor tríbrido de 4 canais
1	**2**	4-Channel hybrid server
2	**1**	Câmara analógica tipo dome para exterior
3	**1**	Câmara analógica tipo bullet para exterior
4	**1**	Fonte de alimentação de 2A
5	**2**	2A Power supply

Tabela 3.9: 3ª FN –Tabela descrições de produto

3.2. DESCRIÇÃO DO MODELO RELACIONAL

Com base nas regras de normalização será o momento de proceder ao desenho do sistema. Face à dimensão do modelo de dados da aplicação de orçamentação, ilustrado no Figura A.1, o mesmo seria extremamente complexo de ser analisado de forma global. Por este motivo, a sua descrição será efetuada em várias secções.

Para verificar a data de criação e da edição da informação em todo o sistema foram criados dois atributos, sendo estes o atributo "created_at" que indica quando a informação foi criada e o atributo "updated_at" que indicar a data da última alteração efetuada na informação. De modo a evitar problemas ao apagar registos com relações em outras tabelas ou por engano, foi implementada a função "*Soft Deleting*" [7] em todo o sistema, criando com isto o atributo "deleted_at". Esta função evita que o registo seja mostrado na aplicação, mantendo-se o registo na base dados, com a exceção dos casos em que se pretenda mostrar essa informação como por exemplo uma "reciclagem".

Tendo em conta que os clientes, fornecedores e as sucursais contém uma morada que irá corresponder a uma localidade, optou-se por criar um registo de localidades. A primeira secção do sistema correspondente às tabelas relacionadas com as localidades. Sendo visível na Figura 3.4 as relações entre as tabelas países – distritos, distritos – concelhos e concelhos – localidades, é uma relação de um para muitos (1:N), ou seja, um país tem vários distritos e um distrito tem apenas um país; um distrito tem vários concelhos e um concelho apenas um distrito; um concelho contém várias localidades e uma localidade diz respeito a um concelho.

Para evitar cruzamento de informação na tabela de países, Tabela 3.10, foram adicionadas algumas condições, como é possível observar nos "*indexes*" da Figura 3.4, os atributos "pais", "indicativo" e "sigla" devem ser únicos, Tabela 3.11.

id	nome	indicativo	sigla
1	Portugal	+351	PT
2	Portugal	+34	PT
3	Espanha	+351	ES
4	Espanha	+34	ES
5	França	+33	FR
6	Eslovénia	+386	ES

Tabela 3.10: Tabela de países sem restrição

id	nome	indicativo	Sigla
1	Portugal	+351	PT
3	Espanha	+34	ES
5	França	+33	FR
6	Eslovénia	+386	SI

Tabela 3.11: Tabela de países com restrição

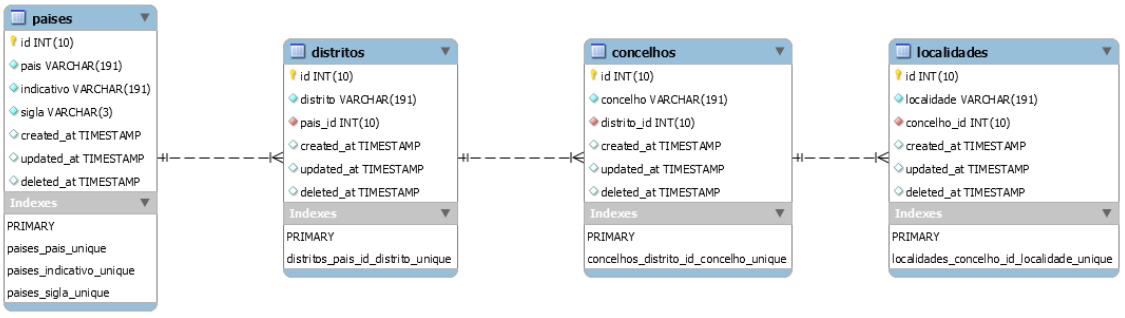


Figura 3.4: DER – Localidades

Antes de ser possível registar seja o que for, é necessário criar os registos da empresa, estando na Figura 3.5 indicado o conjunto de tabelas onde será guardada toda a informação referente às empresas. Para permitir que futuramente a aplicação possa ser utilizada por diversas empresas, foi criada uma tabela extra denominada de **empresas**, onde será armazenada toda a informação genérica correspondente a cada empresa. Uma vez que cada empresa pode ser constituída por um agrupamento de lojas/sucursais, é essencial criar a tabela **sucursais** para armazenar os dados relativamente às lojas de cada empresa, tal como o nome e a morada. Cada empresa deverá ser composta pelo menos uma sucursal.

Para tornar um orçamento mais apelativo e não apenas uma folha em branco com um conjunto de tabelas e respetivos valores, foi criada uma tabela extra, **empresas_folhas_propostas**, que relaciona as empresas com as folhas de rosto/marca d'água. Cada empresa pode conter várias folhas de rosto, mas apenas uma pode estar ativa.

Para garantir que não existem empresas duplicadas foram adicionadas as condições de que o nome e o NIF das empresas deveram ser únicos. O mesmo sucede com as sucursais onde o nome de cada sucursal deve ser único para cada empresa, podendo existir o mesmo nome de sucursal para diferentes empresas.

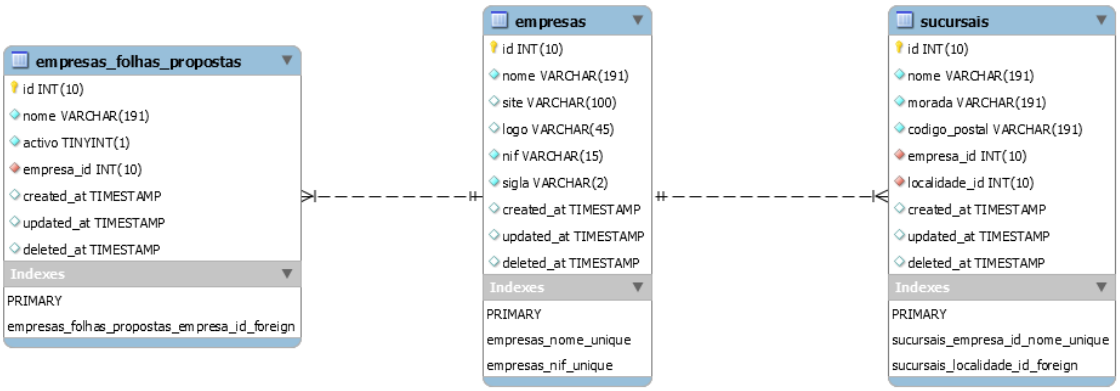


Figura 3.5: DER – Empresas e Sucursais

Face à necessidade da empresa manter o registo de todos os clientes e fornecedores, foram criadas as tabelas **clientes** e **fornecedores** e as respetivas relações com as tabelas de contactos. No caso dos clientes foi também criada a tabela **clientes_tipos**, como ilustrado na Figura 3.6. Esta tabela tem como objetivo distinguir os diferentes tipos de clientes, tais como particulares, revendedores, empresas e/ou parceiros, e posteriormente também diferenciar os níveis de desconto a que cada tipo de cliente deverá ter. A Figura 3.7 é

referente às tabelas dos **fornecedores**. Desde logo é possível observar diversas semelhanças entre as tabelas dos **clientes** e dos **fornecedores**, sendo a única diferença a existência da tabela "**clientes_tipos**" no lado dos clientes. Em ambas as tabelas, o seu relacionamento é de 1:N, pois cada registo de cliente e fornecedor pode conter múltiplos contactos, e cada contacto diz respeito a um cliente ou fornecedor. O mesmo se pode observar em relação aos tipos de cliente, pois cada cliente tem um tipo, mas cada tipo pode corresponder a vários clientes. Na situação de um cliente ser igualmente um fornecedor ou vice-versa, será necessário proceder à criação do registo na tabela correspondente.

De forma a garantir que cada cliente e fornecedor são únicos por empresa, e como as diferentes empresas não devem poder aceder/modificar os dados clientes/fornecedores umas das outras, foi criada a condição que o nome do cliente/fornecedor deverá ser único. Para evitar diversos contactos com o mesmo nome num determinado cliente ou fornecedor foi adicionada a condição de que o conjunto composto por nome e apelido é único.

Figura 3.6: DER – Clientes

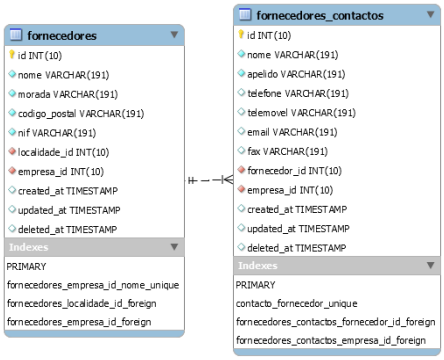


Figura 3.7: DER – Fornecedores

Para auxiliar os utilizadores na fase da orçamentação dos produtos foram seccionados em categorias, subcategorias e tipos de produto, ilustrado na Figura 3.8, estabelecendo com isto diversas formas de filtragem.

Nas **categorias** é guardada a informação mais generalizada do produto, como por exemplo, o "Sistema Automático de Deteção de Incêndio – SADI", "Sistema Automático de Deteção de Intrusão/Roubo – SADIR", ou "Sistema de Videovigilância – CCTV".

Nas **subcategorias** é guardada a informação mais detalhada do produto. No caso do "Sistema Automático de Deteção de Incêndio – SADI" pode-se dividir em duas

subcategorias, sendo elas o “Endereçável” e o “Convencional”, ou no caso do “Sistema de Videovigilância – CCTV” será em “Analógico”, “IP” e “Tríbrido”.

Sendo o relacionamento entre as tabelas de categorias e subcategorias uma relação de muitos para muitos (M:N), ou seja, uma categoria pode conter várias subcategorias e vice versa. Portanto, será essencial implementar uma tabela intermédia, onde irão ser relacionadas as categorias com as respetivas subcategorias, denominada de **subcats**.

Por último, foi criada a tabela relacionada com os tipos de produto e que foi denominada de **tipos_produto**. Nesta tabela é guardada a informação relacionada com o tipo de produto em si como, por exemplo, o “Sistema de Videovigilância – CCTV” podemos ter os tipos “servidor”, “discos rígidos”, “câmeras”, “acessórios” e “cabos”.

Mais uma vez, a relação entre as tabelas **tipos_produto** e **subcats** é do tipo M:N, uma vez que o tipo de produto poderá está associado a várias categorias/subcategorias, e uma categoria/subcategoria também pode conter vários tipos de produto, surgindo então a necessidade de implementar a tabela **subtipos**. Ao contrário das tabelas referentes às localidades, estas tabelas relacionadas com as categorias necessitam de um campo que relacione a qual empresa corresponde, para que apenas os utilizadores dessa empresa utilizem às categorias criadas.

Uma vez que existe a possibilidade das propostas serem em diferentes idiomas, a descrição referente às categorias, subcategorias e tipos de produto devem estar definidas para os idiomas comercializados. Como tal, foi necessário a criação das tabelas **categorias_idiomas**, **subcategorias_idiomas** e **tipos_produtos_idiomas** que relacionam as categorias, subcategorias e tipos de produto respetivamente com os diferentes idiomas.

Mais uma vez para garantir que não existe informação duplicada foram criadas diversas condições tais como: a sigla na tabela **categorias** deverá ser única para cada empresa; as descrições das categorias, subcategorias e tipos de produto são únicas para cada conjunto composto pelo id da categoria, subcategoria ou tipo_produto e do id do idioma; na tabela subcats, o conjunto composto pelos atributos categoria_id e subcategoria_id devem ser únicos; e na tabela subtipos, o conjunto composto pelo tipo_produto_id e subcat_id também devem ser únicos.

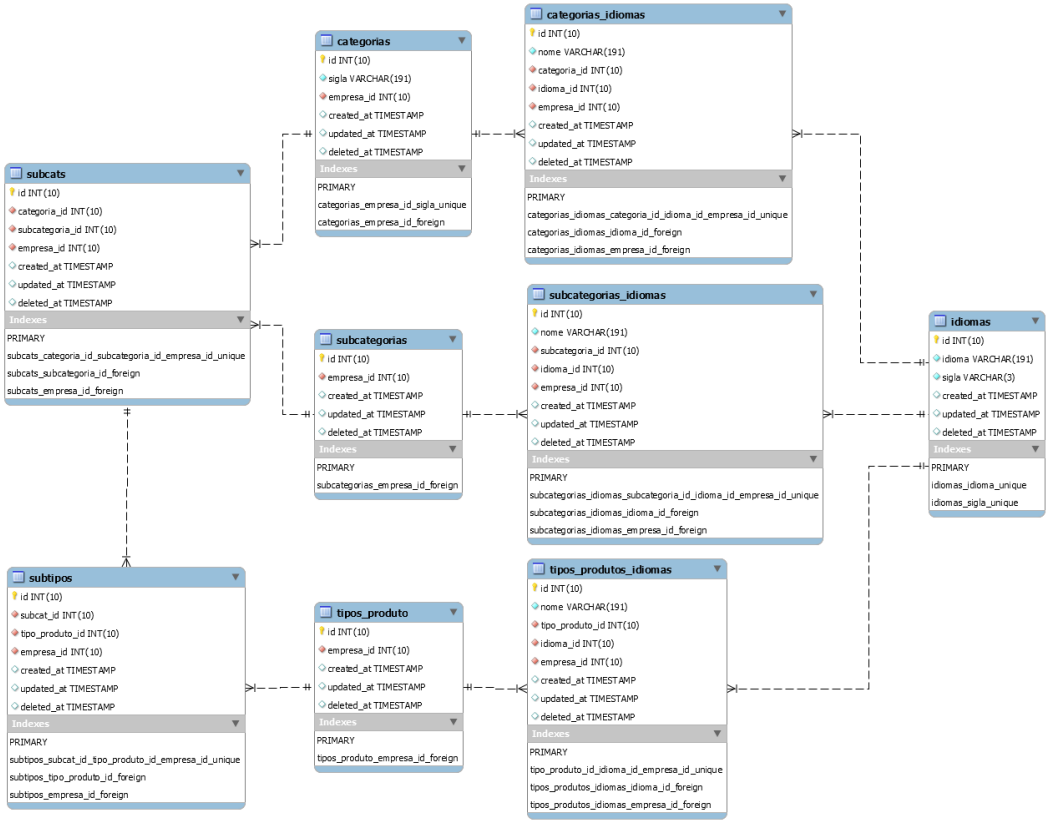


Figura 3.8: DER – Categorias

Após finalizar a criação das tabelas referentes às categorias, surgem então as tabelas relacionadas com os **produtos,** Figura 3.9. Existindo a possibilidade de vários fornecedores disporem dos mesmos produtos, foi necessário criar a tabela **fornecedores_produtos** que relaciona o produto com os seus respetivos fornecedores. Além da referida tabela, e de forma a registar a evolução dos preços de compra e venda, foi necessário criar as tabelas **fornecedores_produtos_precos** correspondendo ao registo dos preços de compra de determinado produto ao seu respetivo fornecedor ao longo do tempo, e **produtos_tabela_precos** que por sua vez regista a variação dos preços de venda e revenda para cada produto. Através da tabela **clientes_tipos** obteve-se a tabela **produtos_tabelas_desconto**, para definir os descontos a que cada tipo de cliente tem direito. Posteriormente, e face ao projeto/orçamento em questão, poderá ser reajustado esses mesmos descontos.

Nestas tabelas, a maioria dos relacionamentos existentes é de 1:N, com a exceção entre as tabelas **produtos** e **subtipos** que é de M:N, pelo facto de cada produto poder ter vários subtipos e esses subtipos corresponderem a diversos produtos, dando origem então à tabela **produtos_tipos**.

Tal como acontecia nas tabelas relacionadas com as categorias, as tabelas referentes aos produtos também contêm um campo que relaciona com a empresa que os vai comercializar. Isto evita, não só o acesso indevido aos produtos de outras empresas, como também à informação de preços que esta pratica/adquire.

A semelhança do que foi efetuado para as tabelas das descrições das categorias, subcategorias e tipos de produto, foi criada a condição de que cada descrição de produto corresponde a um conjunto composto por os atributos "produto_id" e "idioma_id". Cada tipo de cliente é único para cada empresa e o desconto na tabela **produtos_tabela_descontos** também deve ser único para cada conjunto composto pelos atributos "tipo_cliente_id", "subtipo_id" e "empresa_id".

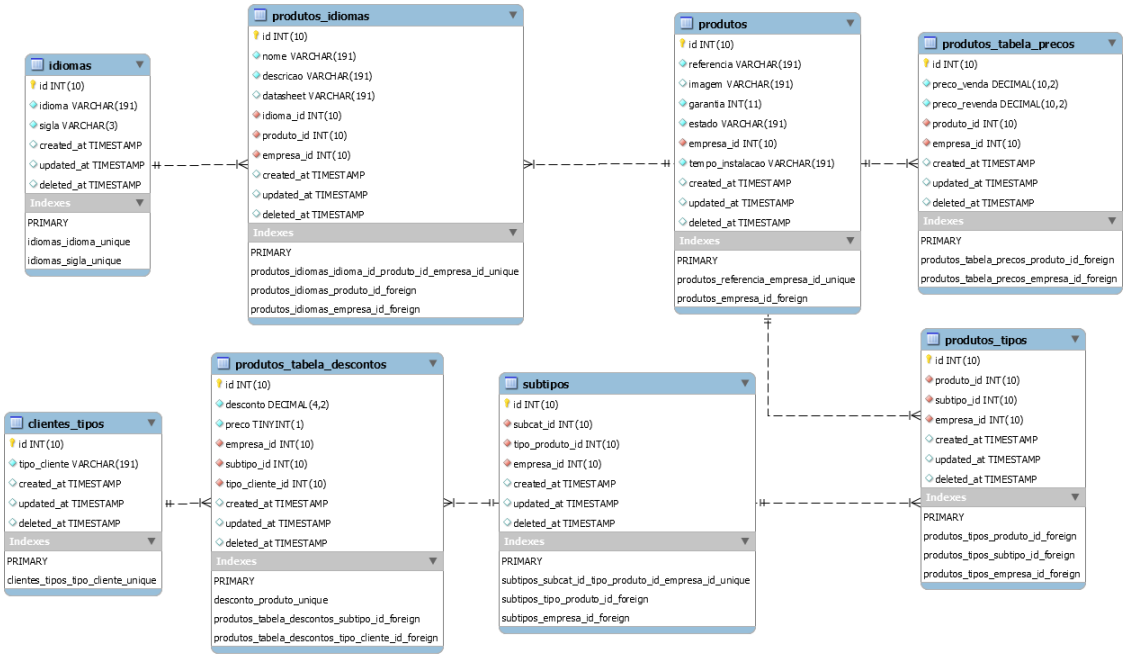


Figura 3.9: DER – Produtos

Após introduzido os dados relacionados com as empresas e as respetivas sucursais, é possível inserir os dados dos funcionários/utilizadores do sistema. Só assim será possível utilizar a aplicação e consequentemente criar as propostas para os clientes, Figura 3.10.

A tabela utilizada para guardar informação relacionada com os utilizadores será denominada de "**users**". Existindo funções atribuídas aos utilizadores dentro da empresa, essas mesmas devem estar armazenadas na tabela **funções.** As designações das funções podem ser por exemplo "comerciais", "técnicos" ou "administrativos". Apesar de um utilizador poder estar associado a diversas sucursais da mesma empresa, a sua função deverá ser sempre a mesma.

Além de cada utilizador estar associado a uma função, este estará igualmente associado a uma empresa/sucursal. Sendo possível um utilizador estar relacionado a diversas empresas, como por exemplo o gestor ou como pessoal administrativo, foi implementada a tabela **empresas_funcionarios.** Deste modo, será relacionado os utilizadores a cada sucursal, facilitando no momento de obter a listagem de funcionários, para estes serem os responsáveis por determinada proposta e também poderá facilitar na obtenção das estatísticas de propostas efetuadas pelas respetivas sucursais ou até de cada funcionário.

De forma a evitar o acesso a determinada informação dentro de cada empresa, qualquer utilizador estará limitado a aceder apenas ao conteúdo correspondente ao seu nível de acesso, nível esse que apenas o administrador da empresa terá acesso a alterar. As limitações impostas aos utilizadores poderão impedir o acesso a determinados menus, os seus conteúdos e funcionalidades. Os níveis de acesso e as funções genéricas estarão disponíveis para qualquer empresa, e não necessitam da relação com a empresa.

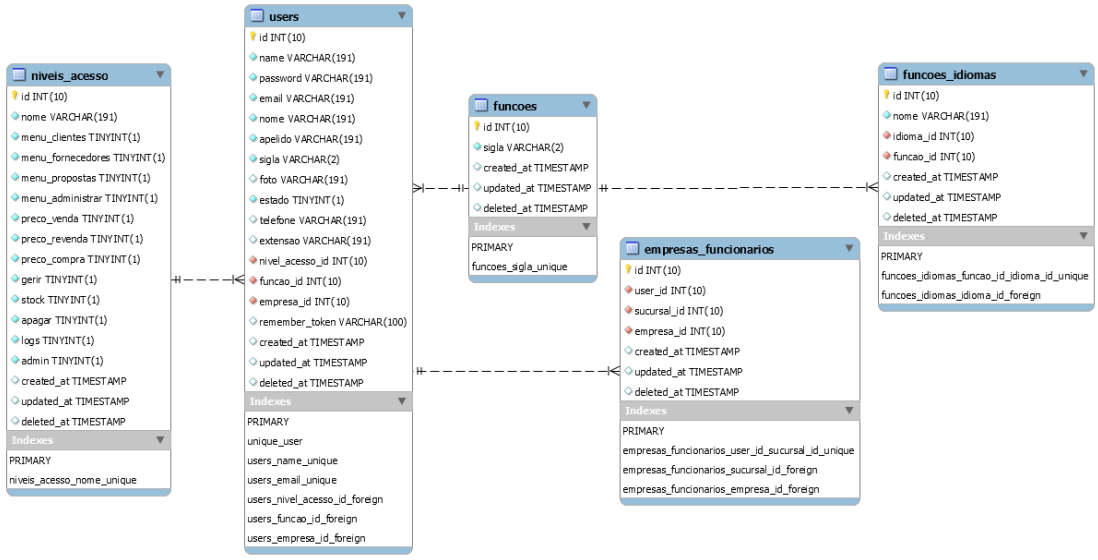


Figura 3.10: DER – Funcionários/Utilizadores

No seguimento do registo dos dados referentes às empresas e respetivas sucursais, é essencial guardar a informação que será, posteriormente, utilizada como modelo para as propostas, Figura 3.11. Essa informação consiste no modelo base das condições comerciais, condições de venda e o modelo de introdução da proposta. A informação deverá encontrar-se em diversos idiomas permitindo ao utilizador escolher o idioma mais indicado para cada proposta. Após selecionado o idioma, a aplicação apresenta de imediato a referida informação na proposta.

Apesar do atributo "id" garantir que todas as tabelas cumprem a 3ª FN, foram adicionadas mais algumas condições tais como na tabela **idiomas** para que o "nome" e a "sigla" sejam únicas, comos nas condições comerciais, condições de venda e o texto introdução deve ser único por idioma e empresa.

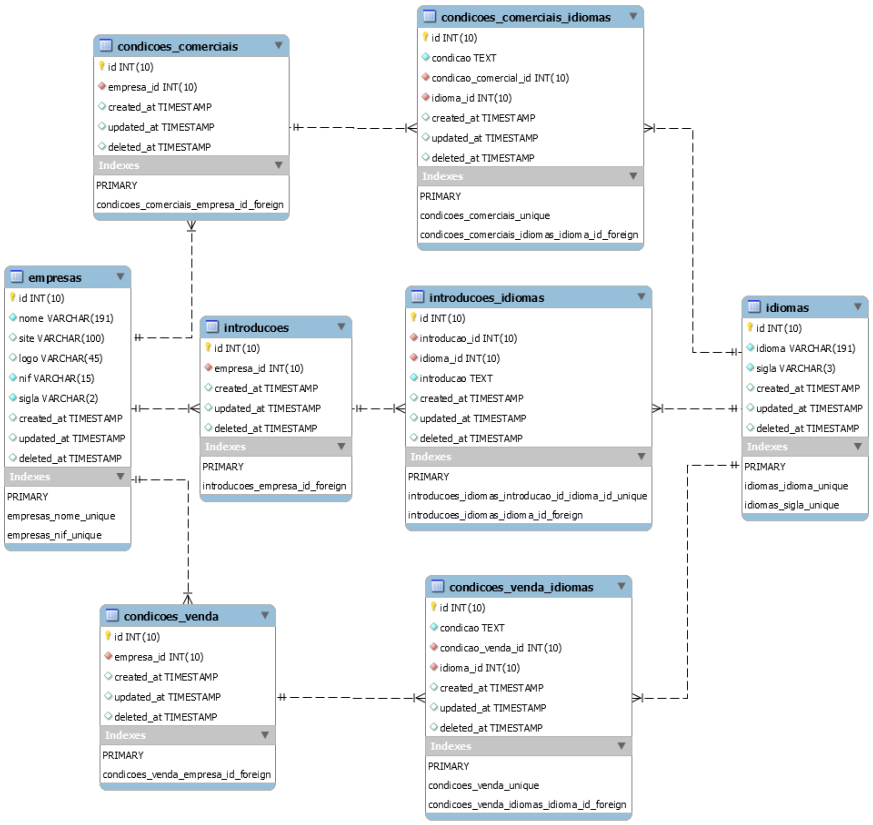


Figura 3.11: DER – Condições

Concluída a criação de todas as tabelas e respetivos registos, poder-se-á dar lugar à elaboração das propostas. Na Figura 3.12 encontram-se ilustradas as tabelas referentes à informação armazenada para as **propostas**. Pelo facto das propostas poderem ser constituídas por mais do que uma categoria/subcategoria, é necessário uma tabela que relacione a proposta à categoria/subcategoria em questão denominada de **propostas_sistemas**. A mesma proposta poderá conter vários categorias/subcategorias repetidas, assim sendo, é dada a opção de escolher o nível a que esta proposta corresponde, como por exemplo "sistema base", "sistema alternativo" ou "complementar". Atendendo à necessidade do cliente, o sistema pode ser um fornecimento de equipamento, uma instalação ou uma assistência técnica. Sempre que seja necessária uma intervenção técnica, e consequentemente envolvendo um custo extra para o cliente, essa informação deverá ser guardada na tabela "**propostas_instalações**". Caso a proposta inclua fornecimento de equipamento, e cada sistema pode conter um

número indeterminado de produtos, a informação relacionada com o produto do sistema deverá ser guardada na tabela "**propostas_produtos**".

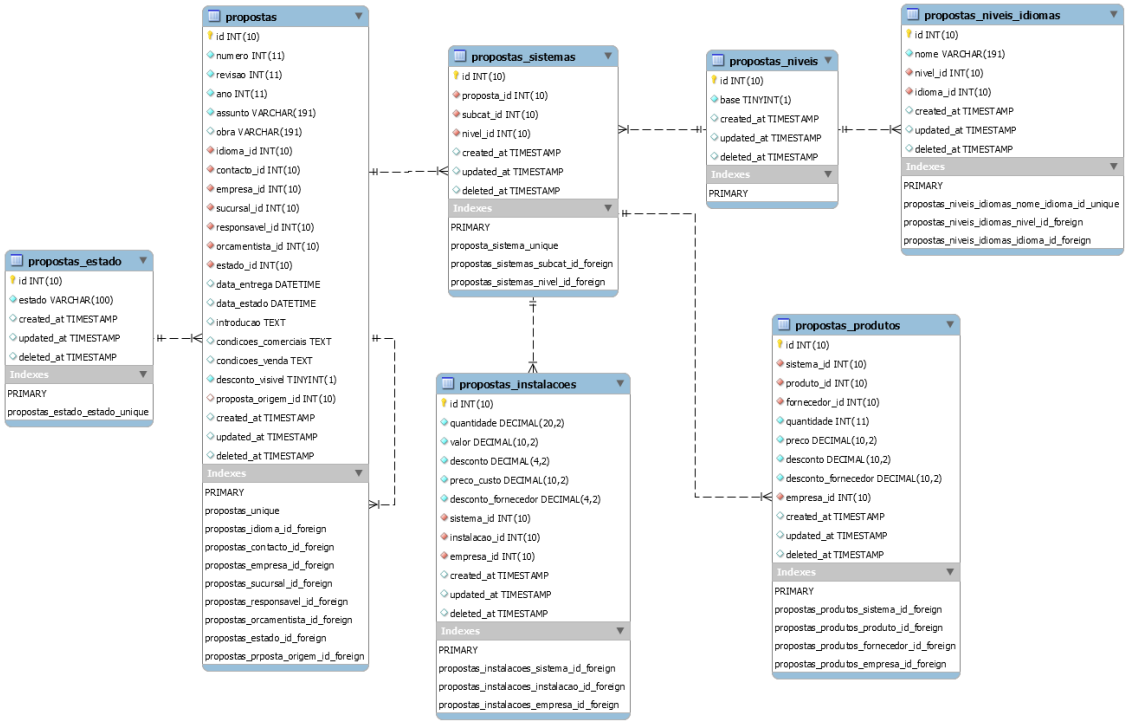


Figura 3.12: DER – Propostas

Terminado este capítulo ficou-se a entender como se deve proceder para a implementação de uma base dados e quais as regras que se deve seguir para evitar erros na sua criação. Ainda ao longo deste capítulo foi ilustrada a base dados da aplicação e os motivos pela qual foi implementada da forma em questão.

Concluída a implementação da base dados do sistema onde será guardada toda a informação, iniciar-se-á no próximo capítulo o estudo e implementação da aplicação, começando por detalhar as ferramentas utilizadas e quais as suas funcionalidades ao longo da aplicação de orçamentação.

4. IMPLEMENTAÇÃO

Após a análise do sistema efetuado no capítulo anterior, e antes de começar a fase de desenvolvimento, é necessário estudar as tecnologias a adotar na implementação do sistema. Assim, neste capítulo que serão referenciadas algumas das tecnologias utilizadas para a implementação do projeto.

Na implementação desta aplicação são utilizadas algumas tecnologias de forma indireta como por exemplo o *Git* [8] que é utilizado para controlo de versões da aplicação e o *Composer* [9], que tratando-se de um gestor de pacotes permite instalar diversas bibliotecas de funções de forma fácil e rápida. Além das tecnologias utilizadas de forma indireta, é também necessário as tecnologias de forma direta, sendo estas linguagens de programação como por exemplo o PHP, JavaScript e CSS.

4.1. TECNOLOGIAS UTILIZADAS

As escolhas das ferramentas/tecnologias implementadas nesta aplicação tiveram como base o estudo efetuado por Tyler Crawford e Tauqeer Hussain [10] com diferentes tecnologias tais como o PHP, Node.js, Django e Rails, Tabela 4.1.

	Node.js	PHP	Django	Rails
Criar aplicação	Muito bom	Excelente	Excelente	Excelente
Ajuda e suporte	Bom	Excelente	Normal	Muito bom
Popularidade	Muito bom	Excelente	Normal	Muito bom
Desenvolvimento Ferramentas e Pacote Gestão Sistemas	Excelente	Bom	Normal	Muito bom
Integrações com bases de dados	Excelente	Muito bom	Normal	Muito bom
Performance	Excelente	Normal	Muito bom	Normal

Tabela 4.1: Comparação entre diferentes tecnologias [1]

Verificando os dados da Tabela 4.1 verifica-se que a melhor tecnologia seria o Node.js e de seguida o PHP. Para este projeto foi considerando que o suporte disponível e a facilidade de implementação das funcionalidades necessárias era mais importante que o desempenho, pelo que o PHP foi a tecnologia selecionada.

Assim, além do PHP [11, 12, 13], as restantes ferramentas/tecnologias necessárias para implementar a aplicação são o *Composer*, *Git* e o *XAMPP* de forma indireta, ficando o

[1] Tabela adaptada do artigo "*A Comparison of Server Side Scripting Technologies*" [10]

HTML [11, 12], CSS [11, 12] e *JavaScript* [11, 12, 14] para o desenvolvimento de forma direta.

COMPOSER

O *Composer* é uma ferramenta de gestão de pacotes de PHP criada por Nils Adermann e Jordi Boggiano em 2012 [15]. Este tem como objetivo auxiliar a criação dos projetos, criando todos os ficheiros necessários para as bibliotecas PHP pretendidas. Como as bibliotecas podem conter inúmeras dependências, o *Composer* será o responsável por instalar todas as dependências necessárias para o funcionamento das bibliotecas evitando com isto falhas que poderiam acontecer numa instalação manual.

GIT

O *Git* é uma ferramenta de controlo de versões *Open Source* desenvolvida por Linus Torvalds em 2005 [16]. Esta ferramenta permite registar o histórico das alterações efetuadas ao logo do desenvolvimento do projeto, admitindo igualmente visualizar os autores das modificações e os comentários efetuados.

Para ser possível começar a utilizar o *Git,* no projeto, e caso este ainda não contenha um repositório*,* é necessário executar o comando "*git init*" no diretório local do projeto, de forma a criar um diretório de trabalho denominado por repositório, como referenciado na sua documentação [8, 17].

Só após a criação do repositório é que será possível utilizar os restantes comandos, tais como o "*git add"* seguido do nome do ficheiro pretendido para o caso de se desejar adicionar um ficheiro específico à lista de monitorização, o "*git add **" para adicionar todos os ficheiros alterados ou ainda o comando "*git add -A*" que adiciona todos os ficheiros alterados com a exceção dos ficheiros listados no documento "*.gitignore*".

Uma vez adicionados à lista de monitorização, os ficheiros pretendidos, é necessário proceder à confirmação dessas mesmas alterações antes destes seguirem para o repositório do *Git*. Deste modo é imprescindível a execução do comando "*git commit*" seguido do comentário referente às alterações efetuadas. Esta mensagem é importante para os outros utilizadores/desenvolvedores entendam as alterações que foram efetuadas ao longo do projeto.

No caso de o projeto se encontrar num repositório remoto, permitindo que vários programadores trabalharem em simultâneo no mesmo projeto, é necessário realizar o

comando "*git push*" após os ficheiros alterados tenham sido devidamente adicionados, permitindo assim que os ficheiros sejam enviados para o repositório remoto.

Para que os restantes elementos do projeto tenham acesso às alterações efetuadas, estes deverão utilizar o comando "*git pull*". Ao executar este último comando, o *Git* vai verificar a informação existente no repositório remoto e compara com a informação alojada no repositório local de destino. Após comparar a informação contida em ambos os repositórios, local e remoto, a informação será automaticamente atualizada se esta informação no repositório local não for mais recente. Na situação oposta, o Git, adiciona as partes da informação alterada nos ficheiros contidos no repositório remoto nos ficheiros do utilizador, dando a informação a este que deverá proceder à verificação/confirmação das alterações implementadas.

No caso de se tratar de um projeto existente basta utilizar o comando "git clone". Este comando cópia os ficheiros existentes no repositório remoto para o novo repositório local. Após a execução deste comando será então possível continuar o projeto do ponto em que este se encontrava.

Para auxiliar a perceção das funcionalidades dos comandos referidos, encontra-se ilustrado na Figura 4.1 o diagrama dos comandos *Git*.

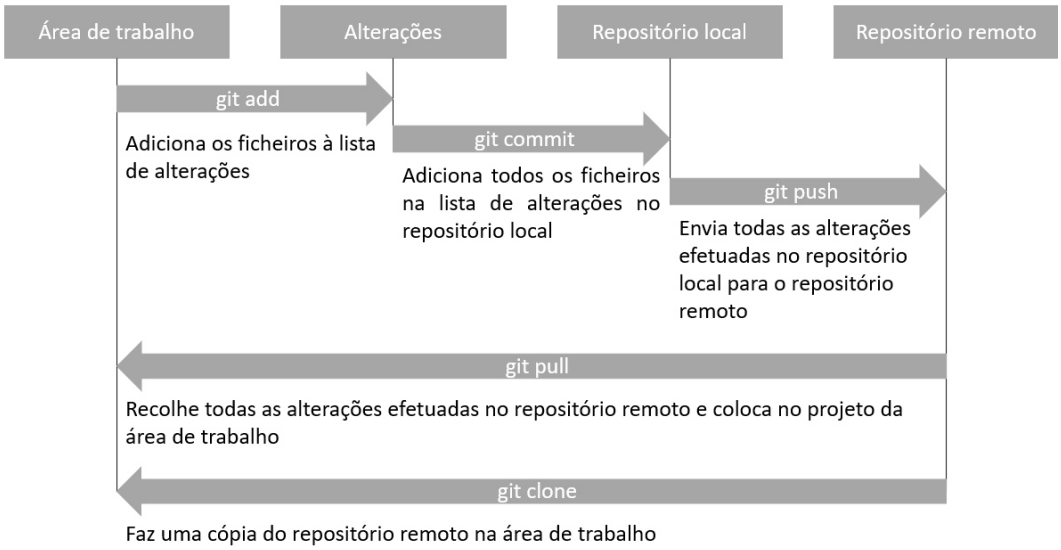


Figura 4.1: Diagrama de comandos Git

Na Figura 4.2 pode-se observar um exemplo de "*Commits*" efetuados para o repositório do *Git*, sendo visível o autor da modificação, a mensagem deixada e a data em que essa alteração foi colocada no repositório.

Autor	Commit	Mensagem	Data
Emanuel Loure...	5a2a2bc	Menu - Peril e actualizações em menus existentes	2018-03-09
Emanuel Loure...	b4e7250	Administrar Categorias - Actualização	2018-02-27
Emanuel Loure...	6bdfa2e	Administrar - Categorias	2018-02-27
Emanuel Loure...	2d52977	Correções Correções de "bug" existentes. Correções no modo responsivo para smartphone	2018-02-17
Emanuel Loure...	56bc561	Menu - Configurações Actualizações no menu de configurações. Alteração da forma de criar empresas.	2018-02-14
Emanuel Loure...	23c63d4	Menu de configuração Actualização	2018-02-13
Emanuel Loure...	e8a9d0f	Tela - Zonas Criação do menu de configuração onde ira ficar alojado as telas de zonas, tipos de cliente, func...	2018-02-13
Emanuel Loure...	bf2218c	Tela - Utilizadores Inicio da tela de utilizadores	2018-02-07
Emanuel Loure...	f211750	Tela - Empresas Alteração da base dados; Criação das telas de visualização e criação de empresas/sucursais.	2018-02-05
Emanuel Loure...	92684ea	Tela - Administração Inicio da criação das telas de administração	2018-02-04
Emanuel Loure...	cc24877	Telas Administração Criação das telas de Zonas e Categorias. Por finalizar: telas de produtos	2018-02-01
Emanuel Loure...	586c475	Tela Produtos Criação das telas de listagem e de novo produto	2018-01-30
Emanuel Loure...	0cdce7d	Tabelas - Produto Inserção de algumas tabelas relacionadas com os produtos	2018-01-29
Emanuel Loure...	c169b4a	Telas de Clientes e Fornecedores Telas praticamente concluidas ficando a faltar alguns promenores que só ...	2018-01-28
Emanuel Loure...	f7d6955	Tela Fornecedores Finalização temporária da tela Clientes e inicio da Tela Fornecedores	2018-01-28
Emanuel Loure...	70ff15e	Tela clientes Adicionado a seleção de localidades para um novo registo de cliente. Adicionar e apagar conta...	2018-01-27

Figura 4.2: Lista parcial de "Commits" do projeto

XAMPP

O XAMPP foi criado por um grupo intitulado de "Apache Friends" em 2002 [18, 19] e este consiste numa compilação de softwares livres, tais como um servidor Apache, um sistema de gestão de base de dados MySQL/MariaDB, interpretadores para linguagens de programação em PHP e Perl. O seu nome deriva do facto de este operar nos diversos sistemas operativos, denominado por *Cross Platform* onde representaram por **X**, de conter um servidor *Apache* representado por **A**, permite alojar bases de dados em *MySQL* [20]/MariaDB, representado por **M**, as últimas letras derivam do suporte deste *software* às linguagens de programação PHP e Perl, **PP**.

Na Figura 4.3 encontra-se representado a consola do XAMPP onde é possível iniciar/parar os seus diversos módulos, efetuar as suas configurações, verificar o seu historial através dos ficheiros de *log*, entre outras funcionalidades. Apesar de no painel de controlo existir os módulos de "FileZilla", "Mercury" e "Tomcat", os mesmos não serão utilizados no projeto.

Figura 4.3: Painel de Controlo do XAMPP

Com o módulo "MySQL" é possível efetuar a gestão das bases de dados, para isso basta aceder à aplicação *phpMyAdmin*. No exemplo da Figura 4.4 é possível observar que o sistema contém algumas bases de dados ilustradas do lado esquerdo, sendo a maioria neste caso, para a gestão das funcionalidades da ferramenta tais como a base de dados "*information_schema"* ou a "*performance_schema"*. Ao selecionar a base de dados observamos a listagem de todas as tabelas existentes. O *phpMyAdmin* dá-nos diversas opções como criar, modificar e apagar tabelas existentes e funcionalidades relacionadas com a gestão das tabelas, tais como pesquisar informação no seu conteúdo, inserir novos registos e verificar a sua estrutura.

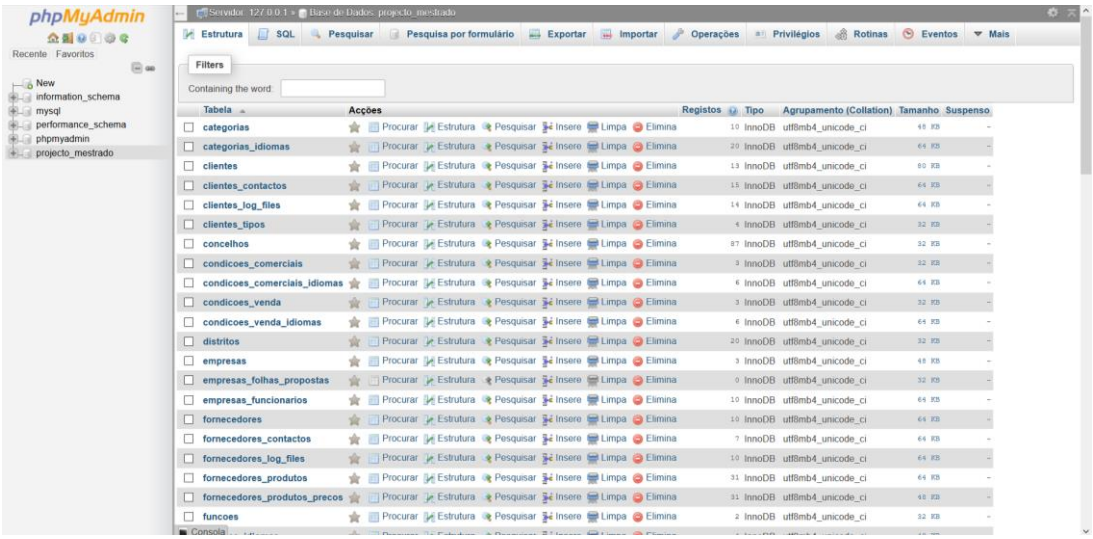


Figura 4.4: phpMyAdmin

PHP

Este projeto é maioritariamente desenvolvido recorrendo à linguagem de programação *open source* PHP [13]. Desenvolvido por Rasmus Lerdorf em 1994, o seu nome derivava de ***P**ersonal **H**ome **P**age*, onde mais tarde passou a sua propriedade de desenvolvimento para o *PHP Group*, alterando o seu significado para *Hypertext Preprocessor* [21].

Apesar da alteração da empresa responsável pelo seu desenvolvimento e do seu significado, as suas funcionalidades permaneceram. Normalmente o código PHP é interpretado no lado do servidor, denominado de *Server-Side* [22], onde as requisições são processadas do lado do servidor antes de enviar a informação para o navegador/cliente poupando recursos do equipamento do utilizador final.

O facto de uma aplicação construída inteiramente em PHP e sem recurso a uma *framework* iria requerer mais tempo de desenvolvimento, pois seria necessário criar

diversas funcionalidades que já se encontram implementadas numa *framework*. Assim sendo e após alguma pesquisa, é possível visualizar em diversos *sites* que o *Laravel* [7], ainda que recente, é considerada uma das melhores *frameworks* de PHP da atualidade como, por exemplo, no fórum "Coderseye" [23], Figura 4.5. Este deve-se à sua enorme comunidade e aos recursos disponíveis.

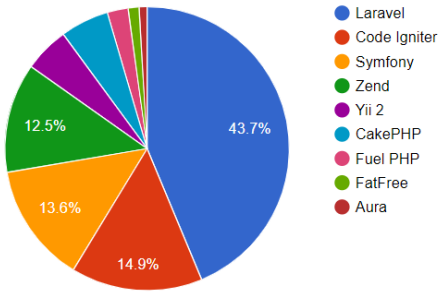


Figura 4.5: Gráfico de utilização de frameworks PHP [23]

LARAVEL

Neste projecto optou-se por utilizar a *framework – Laravel* [7, 24, 25]. A *framework* Laravel, desenvolvida por Taylor Otwell que a disponibilizou a partir de 2011, é baseada em outra *framework – Symfony* [26].

O *Laravel* segue o padrão de arquitetura *Model – View – Controller* (MVC) [27], que divide o projeto em três parte: *Model, View* e *Controller,* como ilustrado na Figura 4.6. Elegeu-se esta *framework* pelas funções já implementadas e pela agilidade que esta permite desenvolver aplicações *Web* seguindo o paradigma MVC.

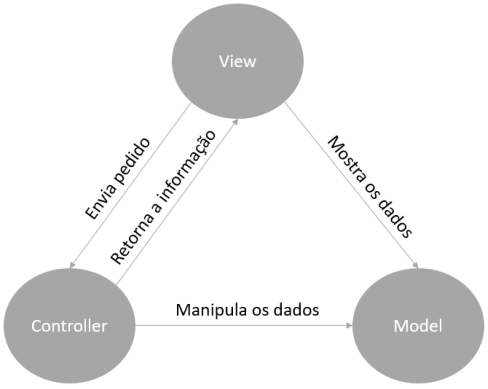


Figura 4.6: Arquitectura MVC

Além do padrão MVC utilizado pela *framework Laravel*, existem ainda outros padrões, tais como o *Model – View – Presenter* (MVP) [28] ou o *Model – View – ViewModel* (MVVM) [29], Figura 4.7. De forma a se perceber como funcionam e quais as suas

diferenças, será efetuado uma pequena análise dos padrões, começando então por explicar o padrão MVC passando posteriormente para o MVP e terminando no MVVM.

No caso do padrão MVC, este divide a aplicação em três componentes principais, sendo que cada um tem uma funcionalidade diferente.

No *Model* encontram-se representadas os dados e a sua inter-relação lógica, onde serão manipulados pelo sistema. O *Controller* é responsável por processar as solicitações recebidas. Este processa a informação, enviando a informação de forma a que esta possa ser manipulada na *View*. Por último, a *View*, exibe a informação enviada pelo *Controller,* podendo utilizar classes presentes no *Model*.

No caso do padrão MVP o funcionamento é semelhante ao do anterior, uma vez que este é uma derivação do padrão MVC. A principal diferença é que este utiliza o *Presenter* em vez do *Controller* do padrão anterior. Ao passo que o *Controller* recebia os dados diretamente do utilizador e os processava utilizando as classes do *Model*, o *Presenter* recebe os dados dos utilizadores por intermédio da *View*, processando os dados com a ajuda de *Model*, passando posteriormente os dados de volta para a *View*. Tanto na *View* como no *Model*, as funcionalidades são as mesmas do padrão MVC.

Por último, o padrão MVVM permite o envio de dados da *View* para a *ViewModel* bem como o oposto. O *ViewModel* é responsável por mostrar os métodos, comandos e outras funcionalidades que permitem manter o estado da *View* e manipular o *Model*. Enquanto que a *View* tem como referência o *ViewModel*, sendo que o oposto, *ViewModel*, não possui informações sobre a *View*, existindo a possibilidade de uma *ViewModel* controlar diversas *Views*.

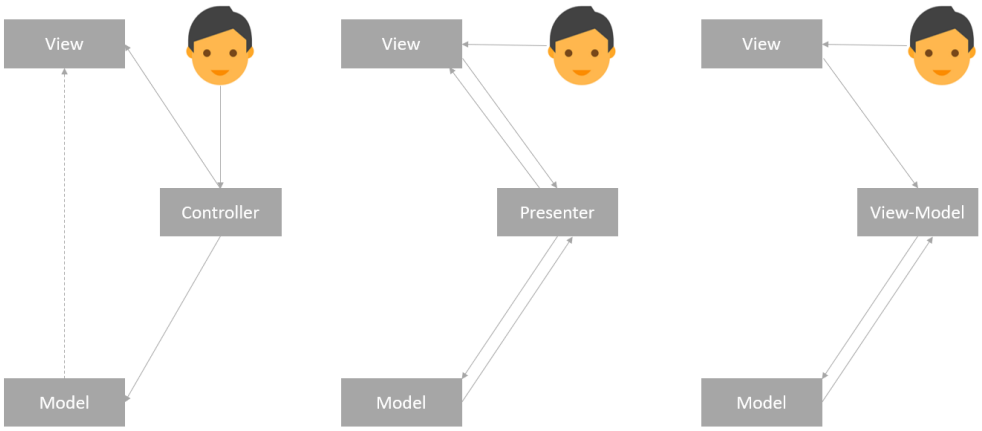


Figura 4.7: Padrões de Design de Software

JAVASCRIPT

O *JavaScript* [14, 30] foi desenvolvido por Brendan Eich em 2005 [31], e inicialmente chamado de *Mocha*, passando depois por se designar de *LiveScript*, até finalmente obter o nome pelo qual hoje é conhecido.

Nesta aplicação, e ao contrário do PHP, esta linguagem será interpretada do lado do cliente, *Client-Side* [32]. As funcionalidades utilizadas são principalmente para a criação de menus dinâmicos do tipo *dropdown*, obtenção de dados para formulários e para criação de gráficos.

CASCADING STYLE SHEETS – CSS

Por último, o *Cascading Style Sheets* (CSS) [33], foi desenvolvido por Håkon Wium e por Bert Bos em 1994 [34], face à necessidade de Håkon Wium produzir um *layout* semelhante ao de um jornal em papel.

O CSS permite uma maior flexibilidade e controlo nas especificações de apresentação, admitindo com isso ajustar mais facilmente o conteúdo e a aparência do *layout*, ou seja, permite modificar qualquer elemento, como por exemplo tipos, formas e tamanhos de letra, tal como alterar a aparência e/ou posicionamento de outros elementos de forma simples. De modo a compreender melhor a funcionalidade do CSS, encontra-se ilustrado na Figura 4.8 um código em HTML com CSS. Este código permite passar de uma simples página HTML com a mensagem "Olá mundo!", ilustrado na Figura 4.9, para uma página com o fundo na cor "*cyan*", aumentando o tamanho da fonte, *font-size*, para 80px, alinhando a mesma ao centro, *text-align*, do ecrã e alterando o tipo de letra, *font-family*, para "*Indie Flower*", como demostrado na Figura 4.10.

```
<html>
    <head>
        <style type="text/css">
            body{
                background-color: cyan;
            }
            h3{
                font-size: 80px;
                text-align: center;
                font-family: 'Indie Flower', cursive;
            }
        </style>
    </head>
    <body>
        <h3>Ola mundo!</h3>
    </body>
</html>
```

Figura 4.8: Exemplo de código HTML com CSS

Figura 4.9: Resultado do HTML sem CSS

Figura 4.10: Resultado do HTML com CSS

4.2. DESENVOLVIMENTO DA APLICAÇÃO

Após definidas e instaladas as tecnologias necessárias será possível, iniciar o desenvolvimento da aplicação. Com a instalação do *Composer* tornou-se possível a criação do projeto em *Laravel* de forma simples e com todas as suas dependências necessárias, bastando executar o comando "*composer create-project --prefer-dist laravel/laravel"* seguido do nome pretendido para o projeto. Deste modo, para além da criação de todas as dependências, foram adicionadas todas as pastas e ficheiros necessários para começar a desenvolver o projeto, como ilustrado na Figura 4.11. Um dos ficheiros criados através do processo anterior é o ficheiro de configuração "*.env*", responsável pela configuração dos acessos à aplicação. Na Figura 4.12 é possível visualizar a forma como é efetuada a configuração de acesso à base de dados, e neste caso também ao e-mail.

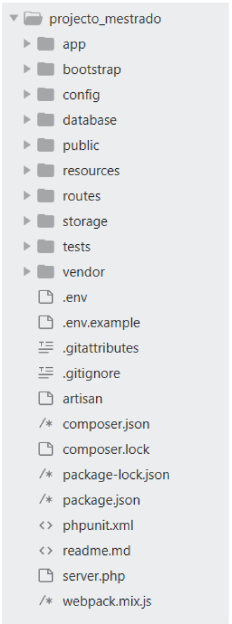


Figura 4.11: Estrutura do projeto em Laravel

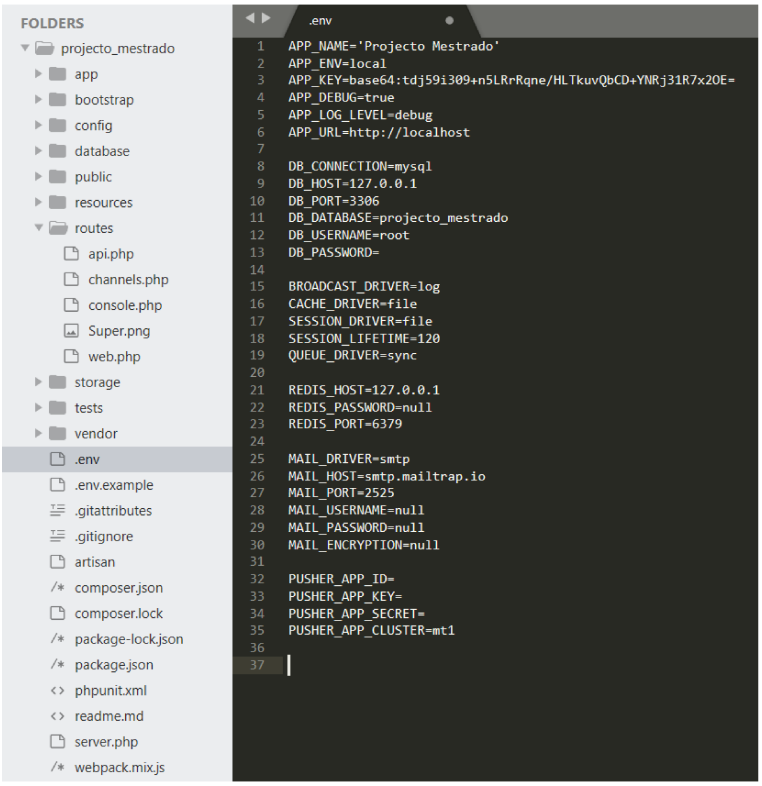


Figura 4.12: Ficheiro de configuração de acessos – Laravel

Após a configuração, é possível utilizar a funcionalidade *Migrations* presente no *Laravel*. Segundo a documentação do *Laravel* [7], esta funcionalidade *Migrations* assemelha-se à tecnologia *Git,* no entanto são exclusivamente para controlo de versões da base de dados, mostrando a data e hora da criação do ficheiro juntamente com o nome atribuído. Na Figura 4.13 é possível observar um exemplo de uma *Migration* efetuada para a tabela *"Users",* Figura 3.10. Para a criação da referida *Migration* é necessário executar o comando "*php artisan make:migration*" seguido do nome pretendido para o ficheiro, da ação pretendida, que neste caso se trata de uma criação de uma tabela, e por fim o nome da tabela, ficando o comando a executar neste caso especifico de "*php artisan make:migration create_users_table --create=users*". Com a execução do referido comando obtém-se um ficheiro cujo nome corresponde à data e hora da criação do ficheiro seguido do nome descrito no comando, ficando o nome de "2018_01_12_000000_create_users_table.php". Ainda na Figura 4.13, nas linhas 29 a 36 é possível verificar a existência de chaves estrangeiras. Neste caso específico as chaves fazem referência às tabelas "níveis_acesso", "funções" e "empresas".

2018_01_12_000000_create_users_table.php

```
3  use Illuminate\Support\Facades\Schema;
4  use Illuminate\Database\Schema\Blueprint;
5  use Illuminate\Database\Migrations\Migration;
6
7  class CreateUsersTable extends Migration
8  {
9      /**
10     * Run the migrations.
11     *
12     * @return void
13     */
14     public function up()
15     {
16         Schema::create('users', function (Blueprint $table) {
17             $table->increments('id');
18             $table->string('name')->unique();
19             $table->string('password');
20             $table->string('email')->unique();
21             $table->string('nome');
22             $table->string('apelido');
23             $table->string('sigla', 2);
24             $table->string('foto') ->nullable();
25             $table->boolean('estado')->default('1');
26             $table->string('telefone')->nullable();
27             $table->string('extensao')->nullable();
28             $table->integer('nivel_acesso_id')->unsigned();
29             $table->foreign('nivel_acesso_id')
30                 ->references('id')->on('niveis_acesso');
31             $table->integer('funcao_id')->unsigned();
32             $table->foreign('funcao_id')
33                 ->references('id')->on('funcoes');
34             $table->integer('empresa_id')->unsigned();
35             $table->foreign('empresa_id')
36                 ->references('id')->on('empresas');
37             $table->rememberToken();
38             $table->timestamps();
39             $table->softDeletes();
40
41             $table->unique(['nome', 'apelido', 'sigla', 'empresa_id',],'unique_user');
42         });
43
44     }
45
46     /**
47      * Reverse the migrations.
48      *
49      * @return void
50      */
51     public function down()
52     {
53         Schema::table('users', function (Blueprint $table) {
54             $table->dropSoftDeletes();
55         });
56     }
57 }
```

Figura 4.13: Exemplo de Migration - Tabela Users

Para evitar que o utilizador seja eliminado por engano é adicionado a função *soft delete* [7]. Esta função permite que o utilizador seja eliminado da aplicação, mantendo o seu registo na base de dados. Para remover definitivamente o utilizador é utilizado a função "*forceDelete*", ou para o restaurar é utilizada a função "*restore*".

Concluída a criação de todas as tabelas necessárias, recorrendo à funcionalidade *Migration*, é necessário executar o comando "*php artisan migrate*". Este comando permite criar as tabelas correspondentes à informação contida nos ficheiros das *Migrations* na base de dados previamente definida no ficheiro "*.env*".

Depois de criadas as Migrations deve-se criar os *Models,* como demostrado na Figura 4.14. Sendo as *Migrations* responsáveis pela criação das tabelas na base de dados, os *Models* são responsáveis pela sua gestão. Como tal, estes devem corresponder ao modelo de dados e suas relações de modo a permitir posteriormente na aplicação a consulta e manipulação da informação.

Recorrendo mais uma vez às ferramentas do *Laravel*, é executado o comando "*php artisan make:model*" seguido do nome pretendido, neste caso "*User*", criando um ficheiro com o nome correspondente seguido da terminação "*.php*".

User.php

```php
<?php

namespace App;

use Illuminate\Notifications\Notifiable;
use Illuminate\Database\Eloquent\SoftDeletes;
use Illuminate\Foundation\Auth\User as Authenticatable;

class User extends Authenticatable
{
    use SoftDeletes;
    use Notifiable;

    protected $fillable = [ 'nome','apelido', 'sigla', 'email', 'password','foto','estado','telefone','email', 'empresa_id'];
    protected $guarded = ['id', 'created_at', 'update_at'];
    protected $hidden = ['password', 'remember_token',];
    protected $table = 'users';
    protected $dates = ['deleted_at'];

    public function niveis_acesso(){
        return $this->belongsTo(Nivel_Acesso::class,'nivel_acesso_id','id')->orderBy('nome');
    }

    public function funcoes(){
        return $this->belongsTo(Funcao::class,'funcao_id','id')->orderBy('sigla');
    }

    public function empresas(){
        return $this->belongsTo(Empresa::class,'empresa_id','id')->orderBy('nome');
    }
}

```

Figura 4.14: Exemplo de Model – User

No exemplo da Figura 4.14 observamos que além de conter os campos referentes à tabela "*users*", contem igualmente funções para o relacionamento com as outras tabelas. Estas

funções permitem que no *Controller* e nas *Views* se possa ter acesso à informação do relacionamento das tabelas.

Nas *Views* é onde se encontra o código que dará origem às páginas *Web*. Na Figura 4.15, mais propriamente na linha 30, podemos observar a utilização da função "localidades" na variável "$cliente", permitindo mostrar o campo "localidade" da tabela localidades.

detalhe.blade.php

```
1  @extends('layouts.app')
2
3  @section ('content')
4      <br>
5      <div class="container">
6          <table class="page-title" width="100%" >
7              <tr>
8                  <td>
9                      Cliente
10                 </td>
11                 <td align="right" >
12                     <a class="btn btn-info" href="{{ url('clientes/') }}"><i class="fas fa-chevron-circle-left"></i> Voltar</a>
13                 </td>
14             </tr>
15         </table>
16         <br>
17         <div class="card-body">
18             <table class="table">
19                 <thead>
20                     <tr>
21                         <th class="col-md-6 col-sm-4 col-xs-6">Nome</th>
22                         <th class="col-md-6 col-sm-6 col-xs-6">Morada</th>
23                     </tr>
24                 </thead>
25                 <tbody>
26                     <tr>
27                         <td>{{ $cliente ->nome }}</td>
28                         <td>
29                             {{ $cliente ->morada }} <br>
30                             {{$cliente ->codigo_postal}}, {{$cliente->localidades->localidade}}
31                         </td>
32                     </tr>
33                 </tbody>
34             </table>
```

Figura 4.15: Exemplo de View - Detalhe de clientes

Para limitar o acesso à aplicação, e consequentemente da informação nela contida é essencial a utilização de um sistema de *login.* Como ilustrado na Figura 4.16, os utilizadores para utilizarem a aplicação devem inserir um *E-Mail* previamente registado e a *password*. Uma vez que o *Laravel* tem uma função com essas características, recorreu-se à função "*Auth*", seguindo a sua documentação basta executar o comando "*php artisan make:auth*". Este cria um sistema de *login* simples e previamente configurado. Apesar de o "*Auth*" conter um sistema de registo de utilizadores, "*Users*", este foi modificado de forma a que o formulário contenha toda a informação necessária, como ilustrado na Figura 4.17.

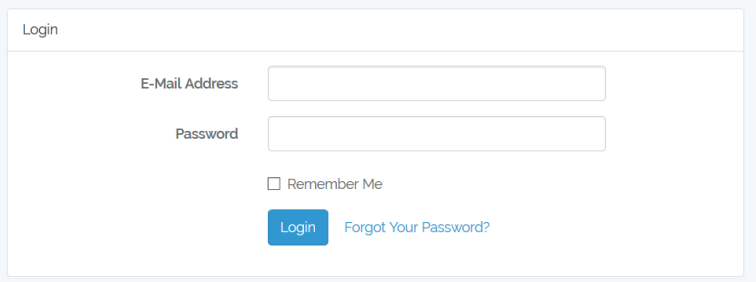


Figura 4.16: Sistema de Autenticação

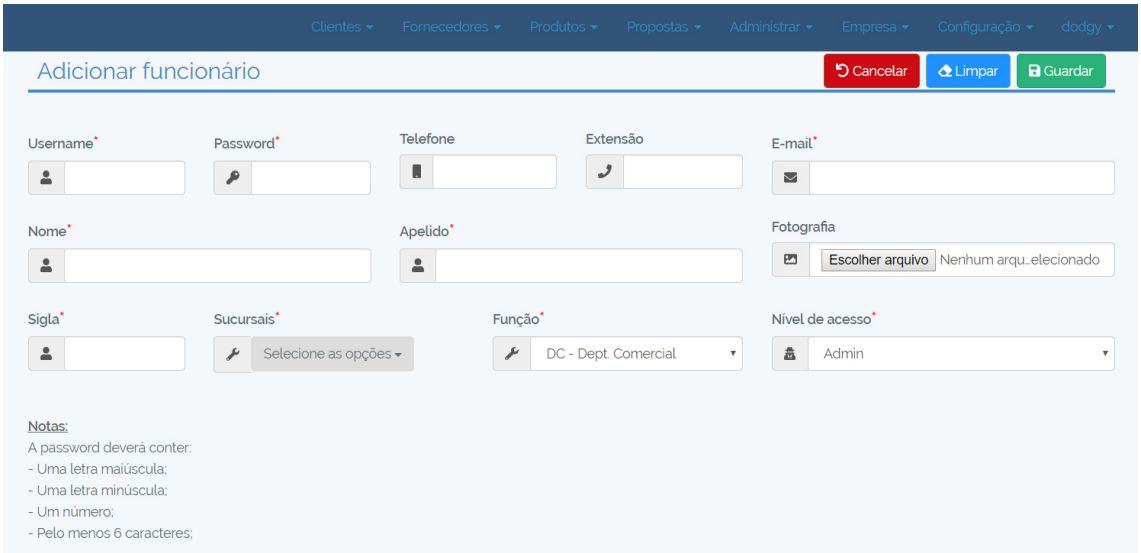


Figura 4.17: Formulário de registo

Como descrito anteriormente, o utilizador deverá efetuar o *login* e consequentemente introduzir um *E-Mail* previamente registado e uma *password*, estando esta encriptada no servidor. Para aumentar a segurança da informação contida na aplicação, foi imposto aos utilizadores que a *password* tivesse pelo menos seis caracteres, constituída por uma letra minúscula, outra maiúscula e um número. Esta imposição é efetuada através da utilização da expressão regular "*RegEx*" [35], como ilustrado na Figura 4.18.

```
protected function validator(array $data)
{
    return Validator::make($data, [
        'name' => 'required|string|max:255',
        'email' => 'required|string|email|max:255|unique:users',
        'password' => 'required|min:6|regex:/(?=.*[a-z])(?=.*[A-Z])(?=.*\d).+$/|confirmed',
    ]);
}
```

Figura 4.18: Validação de registo de utilizador utilizando a função RegEx

Na Figura 4.19 encontra-se alguns exemplos possíveis de password, no entanto, face à limitação imposta, apenas algumas das passwords representadas, e assinaladas a azul, serão aceites no sistema.

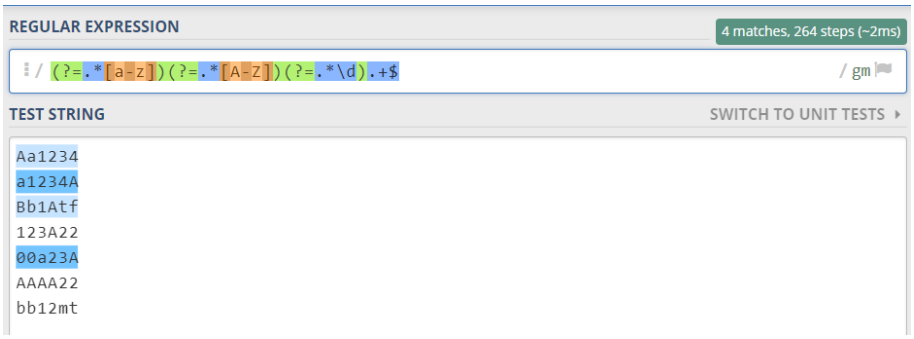


Figura 4.19: Verificação da função RegEx criada

Após a criação da funcionalidade de *login*, e antes de ser possível utilizá-la, é necessário definir a rota de acesso. Estas rotas vão definir o que vai ser executado consoante o *URL* e o método escolhido. Apesar de existirem diversos métodos, os mais utilizados neste projeto foram o GET, POST, PUT e DELETE, estando cada um deles associados a uma funcionalidade.

O método GET é dos mais utilizados tanto para navegação como para efetuar consultas. Sempre que uma consulta necessite de parâmetros, estes serão passados junto da URL.

Para a criação de registos é utilizado o método POST. Ao contrário do método GET, o POST envia os parâmetros no corpo da requisição, não sendo visíveis na URL.

Sempre que se necessita editar um registo recorre-se ao método PUT. Este método envia um parâmetro junto da URL, normalmente o ID do registo, e os restantes parâmetros serão enviados junto da requisição tal como no método POST.

Por fim o método DELETE que como o nome indica é utilizado quando se quer eliminar os registos. Tal como no método anterior, este também envia um parâmetro junto da URL que deverá ser o ID do registo que se pretende eliminar.

Na Figura 4.20 encontram-se ilustradas algumas das rotas. Para os casos em que se pretende a criação de grupos de rotas é utilizado o método "*resource*". Na Tabela 4.2 encontra-se ilustrado o exemplo das rotas definidas para a *URI* "/clientes" utilizando o método "*resource"* presente na Figura 4.20, linha 45.

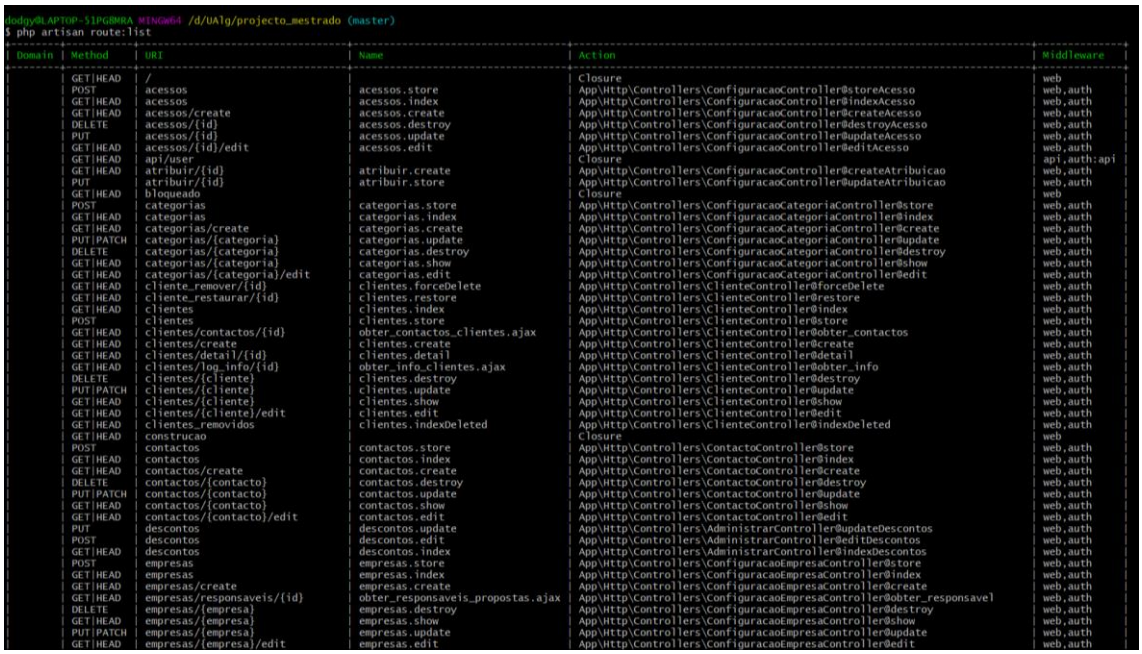
```
17
18  Route::get('/bloqueado', function () {
19      return view('bloqueado');
20  });
21
22  Auth::routes();
23
24  Route::get('/home', 'HomeController@index')->name('home');
25
26  Route::get('/construcao', function () {
27      return view('construcao');
28  });
29
30  //.....................................................................................
31  //....CCCCCCC....LLLL......IIIII.EEEEEEEEEE.NNNN...NNNN..TTTTTTTTTTEEEEEEEEEE..SSSSSSS.....
32  //...CCCCCCCCC...LLLL......IIIII.EEEEEEEEEE.NNNNN..NNNN..TTTTTTTTTTEEEEEEEEEE.SSSSSSSSS....
33  //..CCCCCCCCCCC..LLLL......IIIII.EEEEEEEEEE.NNNNN..NNNN..TTTTTTTTTTEEEEEEEEEE.SSSSSSSSS...
34  //..CCCC...CCCCC.LLLL......IIIII.EEEE........NNNNNN.NNNN.....TTTT....EEEE.......ESSSS..SSSS...
35  //.CCCC.....CCC..LLLL......IIIII.EEEE........NNNNNN.NNNN.....TTTT....EEEE.......ESSSS.........
36  //.CCCC..........LLLL......IIIII.EEEEEEEEE..NNNNNNNNNN.....TTTT....EEEEEEEEE..SSSSSSS.....
37  //.CCCC..........LLLL......IIIII.EEEEEEEEE..NNNNNNNNNN.....TTTT....EEEEEEEEE...SSSSSSSSS...
38  //.CCCC..........LLLL......IIIII.EEEEEEEEE..NNNNNNNNNN.....TTTT....EEEEEEEEE.....SSSSSSS...
39  //.CCCC.....CCC..LLLL......IIIII.EEEE........NNNNNNNNNN.....TTTT....EEEE.............SSSSS..
40  //..CCCC...CCCCC.LLLL......IIIII.EEEE........NNNN.NNNNN.....TTTT....EEEE.......ESSS....SSSS..
41  //..CCCCCCCCCCC..LLLLLLLLLLIIIII.EEEEEEEEEE.NNNN..NNNNN.....TTTT....EEEEEEEEEESSSSSSSSSS..
42  //...CCCCCCCCC..LLLLLLLLLLIIIII.EEEEEEEEEE.NNNN..NNNNN.....TTTT....EEEEEEEEEE.SSSSSSSSS...
43  //....CCCCCCC....LLLLLLLLLLIIIII.EEEEEEEEEE.NNNN...NNNN.....TTTT....EEEEEEEEEE..SSSSSSS....
44  //.....................................................................................
45  Route::resource('/clientes', 'ClienteController')->middleware('auth');
46  Route::get('/clientes/detail', 'ClienteController@detail')->name('clientes.detail')->middleware('auth');
47  Route::get('/clientes/contactos',array('as'=>'obter_contactos_clientes.ajax','uses'=>'ClienteController@obter_contactos'))->
        middleware('auth');
48  Route::get('/clientes_removidos', 'ClienteController@indexDeleted')->name('clientes.indexDeleted')->middleware('auth');
49  Route::put('/cliente_restaurar/{id}', 'ClienteController@restore')->name('clientes.restore')->middleware('auth');
50  Route::delete('/cliente_remover/{id}', 'ClienteController@forceDelete')->name('clientes.forceDelete')->middleware('auth');
51  Route::get('/clientes/log_info', array('as'=>'obter_info_clientes.ajax','uses'=>'ClienteController@obter_info'))->middleware('auth');
52
```

Figura 4.20: Ficheiro configuração das rotas – Laravel

Método	URI	Acção	Nome da rota
GET	/clientes	index	clientes.index
GET	/clientes/create	create	clientes.create
POST	/clientes	store	clientes.store
GET	/clientes/{cliente}	show	clientes.show
GET	/clientes/{cliente}/edit	edit	clientes.edit
PUT/PATCH	/clientes/{cliente}	update	clientes.update
DELETE	/clientes/{cliente}	destroy	clientes.destroy

Tabela 4.2: Exemplo rotas utilizando o método resource

Na Figura 4.21 é possível observar parte das rotas existentes no projeto, após executar o comando do *Laravel* "*php artisan route:list*". Podemos verificar logo no início, a existência de duas *URI* com o mesmo nome, "acessos", no entanto cada uma delas contem um método diferente. No primeiro caso é utilizado o método *POST* e é redirecionado para o *Controller* "*ConfiguracaoController*", onde vai ser executada a função "*storeAcesso*". Já no segundo caso o método a ser utilizado é o *GET*, chamando o mesmo *Controller*, mas desta vez será executado uma função diferente, "*indexAcesso*".



```
dodgy@LAPTOP-51PGBMRA MINGW64 /d/UAlg/projecto_mestrado (master)
$ php artisan route:list
```

Domain	Method	URI	Name	Action	Middleware
	GET|HEAD	/		Closure	web
	POST	acessos	acessos.store	App\Http\Controllers\ConfiguracaoController@storeAcesso	web,auth
	GET|HEAD	acessos	acessos.index	App\Http\Controllers\ConfiguracaoController@indexAcesso	web,auth
	GET|HEAD	acessos/create	acessos.create	App\Http\Controllers\ConfiguracaoController@createAcesso	web,auth
	DELETE	acessos/{id}	acessos.destroy	App\Http\Controllers\ConfiguracaoController@destroyAcesso	web,auth
	PUT	acessos/{id}	acessos.update	App\Http\Controllers\ConfiguracaoController@updateAcesso	web,auth
	GET|HEAD	acessos/{id}/edit	acessos.edit	App\Http\Controllers\ConfiguracaoController@editAcesso	web,auth
	GET|HEAD	api/user		Closure	api,auth:api
	GET|HEAD	atribuir/{id}	atribuir.create	App\Http\Controllers\ConfiguracaoController@createAtribuicao	web,auth
	PUT	atribuir/{id}	atribuir.store	App\Http\Controllers\ConfiguracaoController@updateAtribuicao	web,auth
	GET|HEAD	bloqueado		Closure	web
	POST	categorias	categorias.store	App\Http\Controllers\ConfiguracaoCategoriaController@store	web,auth
	GET|HEAD	categorias	categorias.index	App\Http\Controllers\ConfiguracaoCategoriaController@index	web,auth
	GET|HEAD	categorias/create	categorias.create	App\Http\Controllers\ConfiguracaoCategoriaController@create	web,auth
	PUT|PATCH	categorias/{categoria}	categorias.update	App\Http\Controllers\ConfiguracaoCategoriaController@update	web,auth
	DELETE	categorias/{categoria}	categorias.destroy	App\Http\Controllers\ConfiguracaoCategoriaController@destroy	web,auth
	GET|HEAD	categorias/{categoria}	categorias.show	App\Http\Controllers\ConfiguracaoCategoriaController@show	web,auth
	GET|HEAD	categorias/{categoria}/edit	categorias.edit	App\Http\Controllers\ConfiguracaoCategoriaController@edit	web,auth
	GET|HEAD	cliente_remover/{id}	clientes.forceDelete	App\Http\Controllers\ClienteController@forceDelete	web,auth
	GET|HEAD	cliente_restaurar/{id}	clientes.restore	App\Http\Controllers\ClienteController@restore	web,auth
	GET|HEAD	clientes	clientes.index	App\Http\Controllers\ClienteController@index	web,auth
	POST	clientes	clientes.store	App\Http\Controllers\ClienteController@store	web,auth
	GET|HEAD	clientes/contactos/{id}	obter_contactos_clientes.ajax	App\Http\Controllers\ClienteController@obter_contactos	web,auth
	GET|HEAD	clientes/create	clientes.create	App\Http\Controllers\ClienteController@create	web,auth
	GET|HEAD	clientes/detail/{id}	clientes.detail	App\Http\Controllers\ClienteController@detail	web,auth
	GET|HEAD	clientes/log_info/{id}	obter_info_clientes.ajax	App\Http\Controllers\ClienteController@obter_info	web,auth
	DELETE	clientes/{cliente}	clientes.destroy	App\Http\Controllers\ClienteController@destroy	web,auth
	PUT|PATCH	clientes/{cliente}	clientes.update	App\Http\Controllers\ClienteController@update	web,auth
	GET|HEAD	clientes/{cliente}	clientes.show	App\Http\Controllers\ClienteController@show	web,auth
	GET|HEAD	clientes/{cliente}/edit	clientes.edit	App\Http\Controllers\ClienteController@edit	web,auth
	GET|HEAD	clientes_removidos	clientes.indexDeleted	App\Http\Controllers\ClienteController@indexDeleted	web,auth
	GET|HEAD	construcao		Closure	web
	POST	contactos	contactos.store	App\Http\Controllers\ContactoController@store	web,auth
	GET|HEAD	contactos	contactos.index	App\Http\Controllers\ContactoController@index	web,auth
	GET|HEAD	contactos/create	contactos.create	App\Http\Controllers\ContactoController@create	web,auth
	DELETE	contactos/{contacto}	contactos.destroy	App\Http\Controllers\ContactoController@destroy	web,auth
	PUT|PATCH	contactos/{contacto}	contactos.update	App\Http\Controllers\ContactoController@update	web,auth
	GET|HEAD	contactos/{contacto}	contactos.show	App\Http\Controllers\ContactoController@show	web,auth
	GET|HEAD	contactos/{contacto}/edit	contactos.edit	App\Http\Controllers\ContactoController@edit	web,auth
	PUT	descontos	descontos.update	App\Http\Controllers\AdministrarController@updateDescontos	web,auth
	POST	descontos	descontos.edit	App\Http\Controllers\AdministrarController@editDescontos	web,auth
	GET|HEAD	descontos	descontos.index	App\Http\Controllers\AdministrarController@indexDescontos	web,auth
	POST	empresas	empresas.store	App\Http\Controllers\ConfiguracaoEmpresaController@store	web,auth
	GET|HEAD	empresas	empresas.index	App\Http\Controllers\ConfiguracaoEmpresaController@index	web,auth
	GET|HEAD	empresas/create	empresas.create	App\Http\Controllers\ConfiguracaoEmpresaController@create	web,auth
	GET|HEAD	empresas/responsaveis/{id}	obter_responsaveis_propostas.ajax	App\Http\Controllers\ConfiguracaoEmpresaController@obter_responsavel	web,auth
	DELETE	empresas/{empresa}	empresas.destroy	App\Http\Controllers\ConfiguracaoEmpresaController@destroy	web,auth
	GET|HEAD	empresas/{empresa}	empresas.show	App\Http\Controllers\ConfiguracaoEmpresaController@show	web,auth
	PUT|PATCH	empresas/{empresa}	empresas.update	App\Http\Controllers\ConfiguracaoEmpresaController@update	web,auth
	GET|HEAD	empresas/{empresa}/edit	empresas.edit	App\Http\Controllers\ConfiguracaoEmpresaController@edit	web,auth

Figura 4.21: Lista de rotas – Laravel

Para possibilitar aos utilizadores pesquisarem determinada informação por data, ver Figura 4.24, como por exemplo para obtenção de dados estatísticos, recorreu-se a um *plugin* existente, sendo neste caso instalado o *Bootstrap Datetime Picker* [36]. Este contém uma grande variedade de funcionalidades implementadas e devidamente documentadas, apesar de nesta aplicação ter como única função a anteriormente descrita.

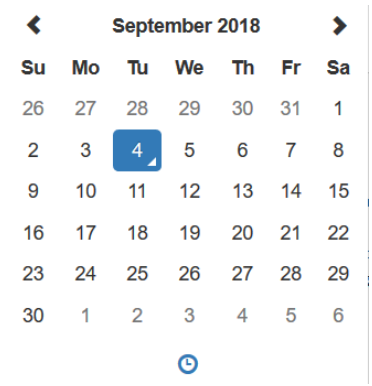


Figura 4.22: Exemplo Calendário – Bootstrap Datetime Picker

Após o utilizador efetuar uma determinada pesquisa, é necessário apresentar os resultados, para auxiliar/complementar essa informação consultada optou-se por apresentar gráficos. Deste modo, recorreu-se mais uma vez a *plugins* existentes, como é o caso do *Google Charts* [37]. Este *plugin* contém inúmeros tipos de gráficos, estando alguns deles ilustrados nas Figuras 4.23, 4.24 e 4.26.

Por exemplo, nas estatísticas das propostas foi utilizado um gráfico do tipo "*Pie Chart*", mostrando em percentagens os estados das propostas consultadas e a sua legenda correspondente, ilustrado na Figura 4.23.

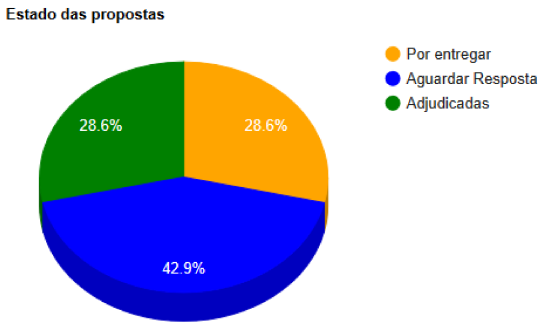


Figura 4.23: Exemplo de gráfico Pie Chart

Para acompanhar a evolução das propostas e a sua origem, optou-se por utilizar um gráfico do tipo "*Organization Chart*", como ilustrado na Figura 4.24. Ao selecionar a revisão pretendida, serão exibidas as alterações efetuadas face à versão de origem. No exemplo da Figura 4.25 observa-se que o Sistema de Videovigilância Analógico se mantém face à versão anterior, ao passo que o Sistema Automático de Detecção de Incêndio Convencional deixa de existir.

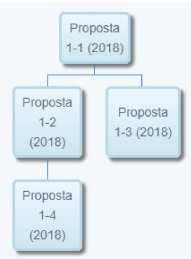


Figura 4.24: Exemplo de gráfico Organization Chart

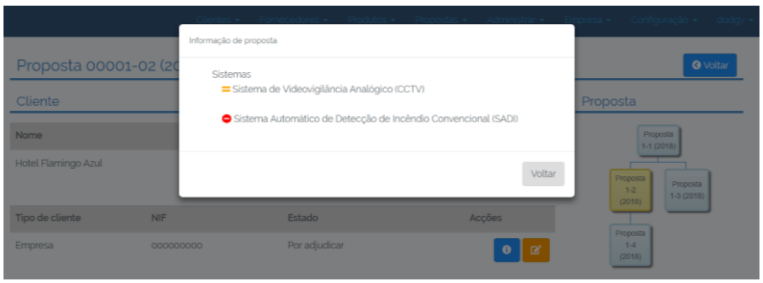


Figura 4.25: Exemplo de variação de revisões

Por último o gráfico do tipo "*Line Chart* que será utilizador para demostrar a oscilação dos preços dos produtos. Este mostra a variação do preço de venda e de revenda ao longo do tempo com a respetiva legenda, Figura 4.26.

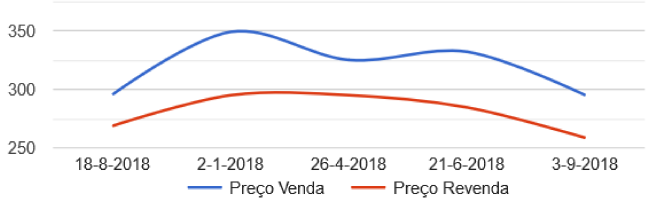


Figura 4.26: Exemplo Line Chart

Para inserir algumas das informações na aplicação, é necessário preencher determinados formulários, como por exemplo inserir um produto. No formulário da Figura 4.27 é possível observar diversos campos para preenchimento, no entanto, alguns destes dependem de outros como, por exemplo, o campo "Tipo de produto" depende da "Subcategoria", e este por sua vez das opções escolhidas no campo "Categoria".

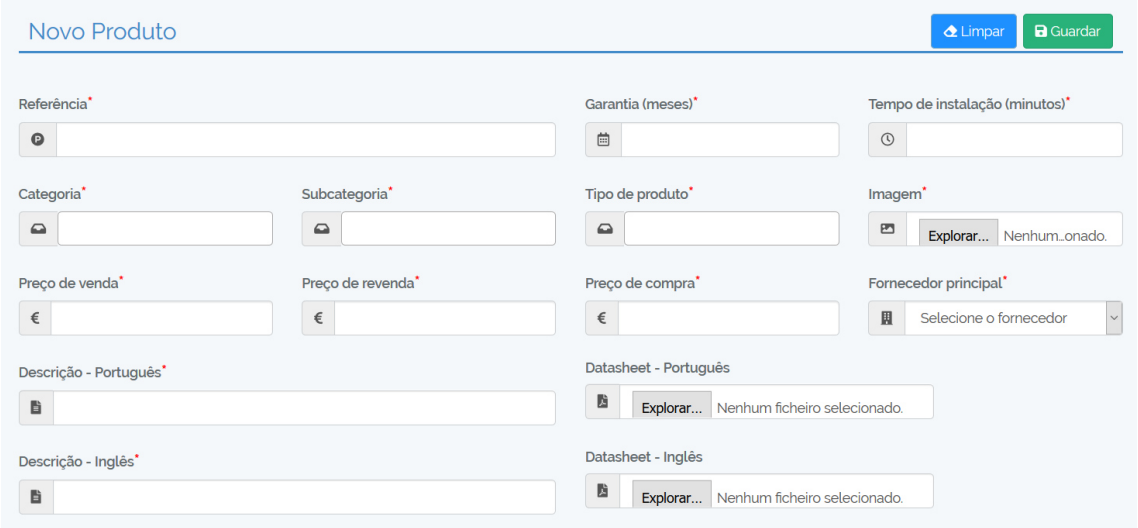


Figura 4.27: Formulário de inserção de produto

Assim, de forma a se colocar todas as opções relacionadas com as categorias previamente selecionadas, utilizou-se o *AJAX JS* [12, 38]. Este permite obter informação de forma dinâmica. Como é utilizada uma *framework* com o padrão MVC, como descrito anteriormente, a informação obtida na função *AJAX* é proveniente de um método do *Controller*, e este por sua vez faz o acesso à base de dados. No exemplo da Figura 4.28 é possível observar a função "subcategorias". Esta função obtém o valor da variável "categoria" do formulário e guarda numa outra variável para posteriormente enviar esse valor juntamente com a *URL*.

```
function subcategorias(){
    var categoria = $('#proposta-categorias').val();
    if(categoria) {
        $.ajax({
            url: '/produtos/categorias/' + categoria,
            type: "GET",
            dataType: "json",
            success: function (data) {
                $('select[name="subcategorias"]').empty();
                $('select[name="tipos"]').empty();
                $('select[name="tipos"]').append('<option value="0" selected>Todos</option>');
                if(data.length > 0){
                    $('select[name="subcategorias"]').append('<option value="0" selected>Todos</option>');
                    $.each(data, function (index, element) {
                        $('select[name="subcategorias"]').append('<option value="' + element.id + '">' + element.nome + '</option>');
                    });
                }
            }
        });
    }
}
```

Figura 4.28: Função AJAX para obtenção de subcategorias

Para não serem efetuados pedidos em vão ao servidor, foi criada a condição que só é executado o pedido se a variável "categoria" possuísse algum valor. Caso a condição se verifique, é então solicitado a requisição ao servidor, mas antes é necessário definir a rota para onde será efetuado essa requisição e qual o seu método, *type*. Depois de realizado o pedido, a resposta é enviada no formato "*JSON*" como ilustra a Figura 4.29.

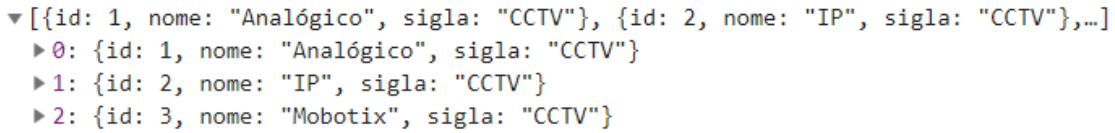


Figura 4.29: Exemplo da resposta em "JSON" resultante da chamada AJAX

Antes de se colocar as novas opções, e de forma a evitar a duplicação ou mesmo opções erradas, é necessário eliminar todas as opções anteriores. Uma vez que a informação, neste exemplo, é enviada em vetores será necessário iterar sobre todas as linhas, onde cada linha corresponde a uma opção de seleção e as colunas aos parâmetros necessários, como por exemplo o "id", "nome" e "sigla", Figura 4.29.

Para ser possível executar a função da Figura 4.28, é necessário inserir a rota bem como o seu método no ficheiro das rotas "web.php", como ilustrado na Figura 4.30. Caso contrário, o sistema não consegue obter a informação pretendida, apresentando o erro correspondente.

```
Route::get('/produtos/categorias/{categoria}',
    array('as'=>'obter_categoria.ajax','uses'=>'ProdutoController@obter_categoria')) ->middleware('auth');
```

Figura 4.30: Rota para execução do AJAX da função subcategoria

Ao consultar a rota é possível observar que esta requer a variável "categoria", e é enviada junto do *URL* no momento do pedido para o *ProdutoController*. Dentro deste *Controller* é executado a função "obter_categoria". Na função presente na Figura 4.31, verifica-se a existência de algumas variáveis tal como o "$user", "$idiomas" e "$subcategorias". Na variável "$user" é guardada a informação relativa ao utilizador do sistema. Já na variável "$idiomas" é guardado o valor do "id" do idioma pretendido, que neste caso na aplicação apenas está disponível a opção de Português. Após recolhida a informação sobre o utilizador e o idioma, é efetuada a pesquisa nas tabelas pelas subcategorias dos produtos correspondentes à categoria selecionada no formulário, cujo nome está no idioma pretendido e às mesmas correspondem às utilizadas pela empresa do utilizador. O resultado da pesquisa é guardado na variável "$subcategorias" e esta apresenta os valores

correspondentes para os números de identificação da tabela de relação de categorias com as subcategorias, o nome das subcategorias em Português e a sigla correspondente à categoria selecionada.

```php
public function obter_categoria($categoria)
{
    $user     = User::where('id', Auth::user()->id)->first();
    $idiomas = Idioma::where('idioma','Português')->pluck('id');
    foreach ($idiomas as $value){
        $idioma = $value;
    }
    $subcategorias = DB::table('subcats')
            ->join('subcategorias_idiomas','subcategorias_idiomas.subcategoria_id','=','subcats.subcategoria_id')
            ->join('categorias','categorias.id','=','subcats.categoria_id')
            ->where ('subcats.categoria_id', $categoria)
            ->where('subcategorias_idiomas.idioma_id',$idioma)
            ->where('subcategorias_idiomas.empresa_id',$user->empresa_id)
            ->select('subcats.id','subcategorias_idiomas.nome','categorias.sigla')
            ->get();
    return json_encode($subcategorias);
}
```

Figura 4.31: Função para obtenção das opções das subcategorias

Tratando-se de uma aplicação *web*, e de forma a não limitar o utilizador a um equipamento específico, mas torná-lo compatível com as diversas tecnologias atuais, tanto equipamentos fixos como móveis, tais como *smartphones* e *tablets*, existiu a necessidade de incorporar o modo responsivo [39] de forma a que a aplicação se ajustasse de forma autónoma ao equipamento do utilizador. Existindo atualmente algumas bibliotecas com tal funcionalidade, optou-se por incorporar na aplicação o *Bootstrap* [40].

O *Bootstrap* é uma biblioteca de desenvolvimento de componentes, onde incorpora atualmente diversas funcionalidades tal como o modo responsivo, poupando imenso tempo de implementação. Esta biblioteca foi criada por Mark Otto e Jacob Thornton em 2011 [41] contanto com inúmeras versões e funcionalidades. O modo responsivo consiste no seccionamento do ecrã em doze células que, posteriormente, dependendo da dimensão do ecrã, serão redimensionadas consoante a necessidade. Estas células podem ser utilizadas individualmente ou em grupos, como mostra a Figura 4.32.

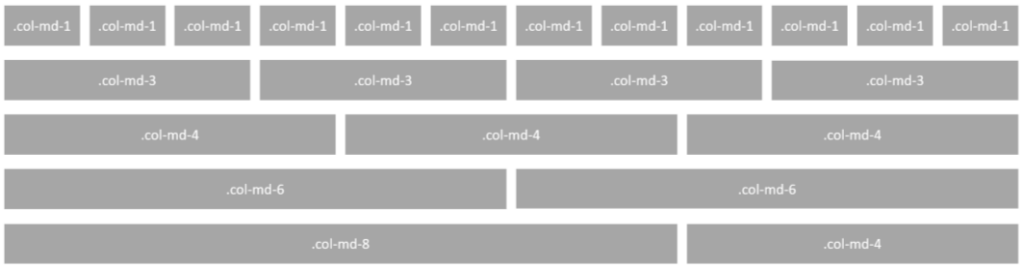


Figura 4.32: Esquema de colunas utilizado pelo Bootstrap [2]

[2] Figura adaptada do site do Bootstrap, disponível em https://getbootstrap.com/docs/3.3/css/ [Acedido em 15 08 2017]

Esta tecnologia define por defeito quatro resoluções distintas, correspondendo a quatro classes, sendo estas o "col-lg", "col-md", "col-sm" e "col-xs". A classe "col-lg" é destinada a equipamentos com elevada dimensão, como por exemplo TV's e monitores superiores a 1200 pixéis – "px". A classe "col-md" aplica-se a equipamentos onde a dimensão não é tão elevada, como por exemplo computadores portáteis ou pequenos monitores entre 992px e os 1200px. No entanto, ainda é superior à dimensão dos dispositivos móveis onde a sua dimensão é inferior a 992px. Para os dispositivos móveis são definidas duas classes: a classe "col-sm" para os equipamentos de dimensões superiores a 768px, tais como tablets, e outra classe "col-xs" para os smartphones com dimensão até 768px. No momento de desenvolvimento da aplicação é necessário ter em conta as respetivas dimensões dos equipamentos e das classes do *Bootstrap* correspondentes. Desta forma quer-se evitar que a informação se torne ilegível ao utilizador. Na Tabela 4.3 é pode-se observar os dados referentes às classes descritas acima.

	Equipamentos extra pequenos (<768px)	**Equipamentos pequenos (≥768px)**	**Equipamentos médios (≥992px)**	**Equipamentos grandes (≥1200px)**
Comportamento				
Largura	Automático	750px	970px	1170px
Número de colunas	12			
Comprimento das colunas	Automático	~62px	~81px	~97px
Largura da calha	30px (15px para cada lado da coluna)			

Tabela 4.3: Resoluções do Bootstrap para criação do Modo responsivo [3]

Nas Figuras 4.33, 4.34 e 4.35 é possível observar o modo responsivo em funcionamento para o mesmo menu e nos diversos tipos de dispositivos. Na Figura 4.33 examinamos o ajuste para ecrãs de grandes dimensões utilizando a classe "col-md" (*medium*). É utilizado para cada uma das opções de tipo de cliente a classe "col-md-1", tornando possível obter doze opções por linha. Para os campos "Nome" e "NIF" foi utilizada a classe "col-md-6", o que permitiu a visualização desses dois campos por linha. Foi utilizada a mesma lógica com os campos "País", "Distrito", "Concelho" e "Localidade", só que nestes casos a linha foi dividida em quatro elementos, coincidindo a cada um a classe "col-md-3". Por

[3] Tabela adaptada do site do Bootstrap, disponível em https://getbootstrap.com/docs/3.3/css/ [Acedido em 15 08 2017]

último, o campo "Morada" optou-se pela classe "col-md-10" por estar sujeita a muita informação, e "Código Postal" elegeu-se a classe "col-md-2" por conter significativamente menor quantidade de informação.

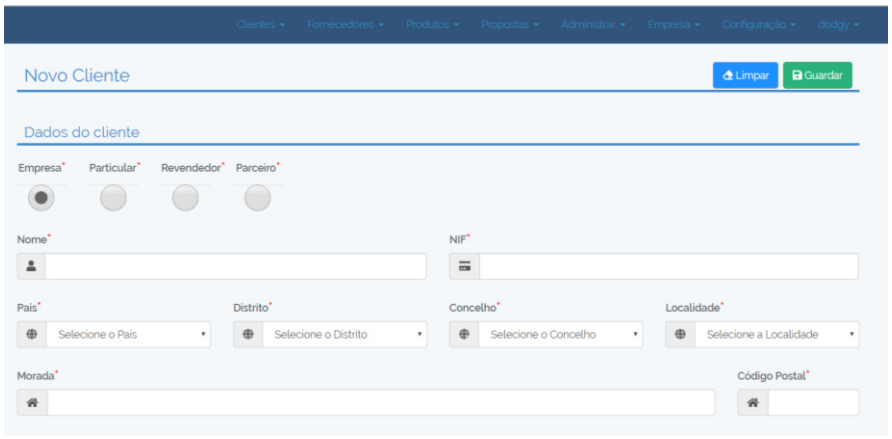


Figura 4.33: Formulário de novo cliente utilizando a classe "col-md"

Na Figura 4.34, a dimensão foi ajustada de modo a corresponder a uma tablet, utilizando a classe "col-sm" (*small*), onde é evidente algumas diferenças face à Figura 4.33, derivado do ajuste do conteúdo à dimensão do ecrã. Existindo neste caso a utilização da classe "col-sm-3" para cada uma das opções de tipo de empresa, deste apenas é possível quatro opções por linha, embora no exemplo anterior era possível obter até doze. Para os campos "Nome" e "NIF", de forma a manter visível os dois campos na mesma linha, foi utilizada a classe "col-sm-6", utilizando-se a mesma classe para os campos "País", "Distrito", "Concelho" e "Localidade", ficando dois campos por linha, o que também difere da figura anterior que mantinha os quatro campos na mesma linha. Mais uma vez, os últimos campos "Morada" e "Código Postal" mantiveram-se na mesma linha, optando-se por colocar as classes "col-sm-9" e "col-sm-3".

Por último, na figura 4.35 a dimensão foi reajustada correspondendo a um smartphone, utilizando neste caso a classe "col-xs" (*extra small*). É notável a alteração da forma apresentada do menu, em que nos exemplos anteriores este tinha os nomes visíveis, o que não se verifica nesta última figura, onde os nomes foram substituídos por um menu do tipo D*rop-Down*, estando apenas acessível após o utilizador clicar no botão. Em relação à apresentação dos tipos de cliente manteve-se a estrutura da visão de *tablet* ao apresentar apenas quatro opções por linha, "col-xs-3". Nos restantes campos, para a informação contida ser legível nos equipamentos de resoluções mais baixas, foi colocada a classe "col-xs-12", tornando cada campo visível em cada linha.

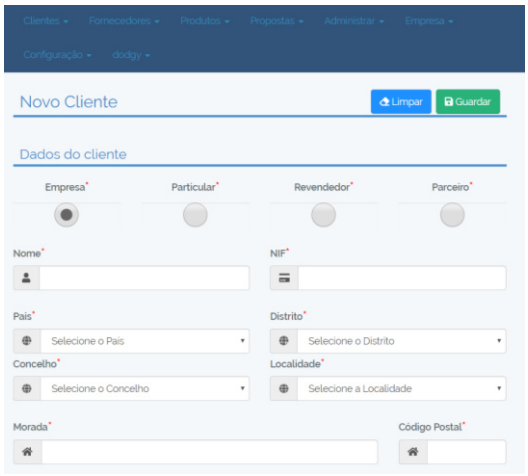


Figura 4.34: Formulário de novo cliente utilizando a classe "col-sm"

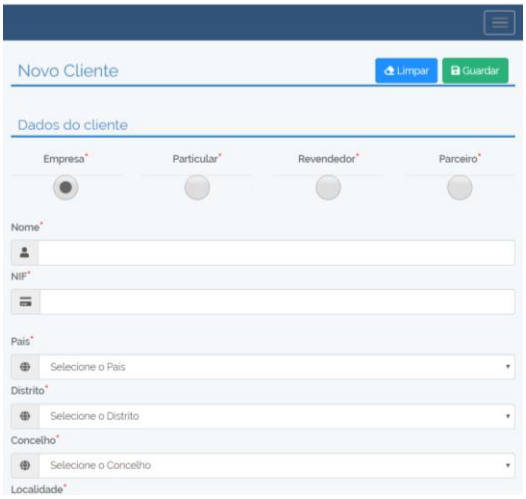


Figura 4.35: Formulário de novo cliente utilizando a classe "col-xs"

Tratando-se de uma plataforma de orçamentação multiempresa, é imprescindível permitir, na criação e na edição de uma empresa, a alteração dos dados e seus termos comerciais. Como tal, recorreu-se mais uma vez a um *plugin* existente, o *Summernote* [42], permitindo ao utilizador editar facilmente a informação referente à introdução da proposta, das condições de venda e comerciais nos diversos idiomas, tornando possível a sua utilização mais tarde na criação das propostas. Na figura 4.36 é ilustrada a ferramenta *Summernote* e as suas opções de edição de texto.

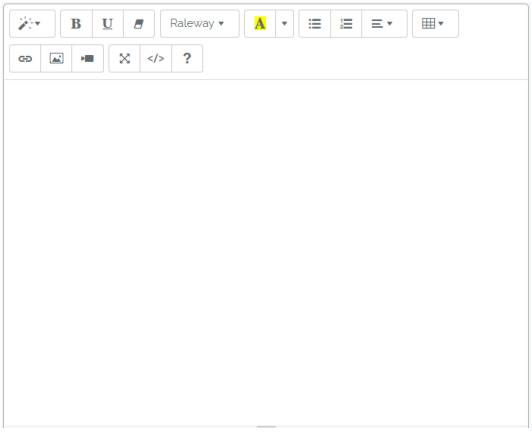


Figura 4.36: Editor de texto Summernote

Após utilização das tecnologias anteriormente descritas, bem como das suas funcionalidades, foi necessário utilizar uma ferramenta de exportação de informação. Esta funcionalidade deve-se ao facto dos clientes não terem acesso à plataforma, e consequentemente acesso às propostas efetuadas. Assim, recorreu-se a um *plugin*,

TCPDF [43], para que fosse possível exportar a informação relativa às propostas para o formato PDF e posteriormente o utilizador enviar para o cliente.

Uma proposta envolve inúmeras páginas como, por exemplo, a introdução, os conteúdos da proposta com os respetivos sistemas, bem como as condições comerciais e de venda. Assim na Figura 4.37 encontra-se apenas representado parte desta nomeadamente o conteúdo da proposta referente ao Sistema de Videovigilância CCTV. Através dessa figura é possível verificar a diferença na descrição, sendo esta mais detalhada quando comparada com a descrição na aplicação, Figura 4.38.

Refª:TC/DC/EL/00001-04/2018 Olhão, 28-05-2019

SISTEMA DE VIDEOVIGILâNCIA - ANALóGICO (CCTV) - BASE

REF	DESCRIÇÃO	QTD	P. UNIT.	P. TOTAL
AWN-72SMVR	Câmara Bullet exterior cor,1/3' 800TVL , ICR,42 leds IV, 40m. Lente : 2.8-12 mm. Alimentação eléctrica: 12VDC (±10%)/ 400mA. Classe de protecção: IP66	8.00	40,00 €	320,00 €
HDDS4TB	Disco Rigido SATA 4Tb.	2.00	167,00 €	334,00 €
HAVR-0872H1		1.00	190,00 €	190,00 €
AU-S12		4.00	13,65 €	54,60 €
	Instalação dos equipamentos, parametrização do sistema e formação. Sem rede de cabos e sem infra-estrutura	1.00	360,00 €	360,00 €
	TOTAL DA PROPOSTA			**1.258,60 €**

Figura 4.37: PDF da proposta de Sistema de Videovigilância CCTV

CCTV - Analógico (Base)

	PRODUTO					VENDA	
ACÇÃO	REFª	DESCRIÇÃO	QTD	P. UNIT	P. TOTAL	DESC.	P. UNIT
	AU-S12	Adaptador 230 VAC/ 12 VDC, 2A	4	13,65	54,60 €	0.00	13,65 €
	AWN-72SMVR	Câmara Bullet exterior cor,1/3 800TVL , ICR,42 leds IV, 40m, varifocal	8	40,00	320,00 €	0.00	40,00 €
	HAVR-0872H1	DVR 8 canais tribrido Analógico	1	190,00	190,00 €	0.00	190,00 €
	HDDS4TB	Disco Sata de 4TB	2	167,00	334,00 €	0.00	167,00 €
	-	Instalação dos equipamentos, parametrização do sistema e formação. Sem rede de cabos e sem infra-estrutura	1.00	360,00	360,00 €	0.00	360,00 €
		TOTAL DA PROPOSTA			1.258,60 €		

Figura 4.38: Proposta de Sistema de Videovigilância CCTV

Ainda na Figura 4.37 é possível notar no lado esquerdo do cabeçalho da proposta a referência da mesma. Esta referência consiste nas iniciais da empresa, seguido das iniciais do departamento, iniciais do orçamentista, número e revisão da proposta, e por fim o ano da proposta inicial. Já no lado direito do cabeçalho indica a localidade da sucursal que

efetuou a proposta bem como da data de entrega da mesma. Na Figura 4.39 é possível observar como foi efetuado a construção do cabeçalho.

```
if($proposta->data_entrega){
    $data_proposta = Carbon::parse($proposta->data_entrega)->format('d-m-Y');
}else{
    $data_proposta = Carbon::parse( NOW())->format('d-m-Y');
}

$html = '<br>
        <table width="100%">
            <tr>
                <td width="30%">Refª:'.$empresa->sigla.'/'.$responsavel->sigla_funcao.'/'.$responsavel->sigla.'/'.sprintf('%05d', $
                    proposta->numero).'-'.sprintf('%02d', $proposta->revisao).'/'.$proposta->ano.'</td>
                <td width="70%" style="text-align: right">'.$empresa->localidade.', '.$data_proposta.'</td>
            </tr>
        </table>';

PDF::writeHTML($html, true, false, true, false, '');
```

Figura 4.39: Criação de cabeçalho no PDF

4.3. ARMAZENAMENTO DE INFORMAÇÃO

Para além das ferramentas previamente referidas, como seria de esperar recorreu-se a uma base de dados para armazenar a informação da aplicação de orçamentação. A informação armazenada consiste, por exemplo, em informações de clientes, fornecedores, produtos, utilizadores, propostas.

Pelo facto de ser necessário efetuar diversas consultas em que algumas poderão ser com um elevado grau de complexidade [44], optou-se por implementar uma base de dados relacional. Este tipo de base dados tem diversas vantagens quando comparados com outras base dados não relacionais tal como a facilidade com que se manipula os dados e a sua consistência, acabando por tornar a sua estrutura menos flexível quando comparado com outras bases dados não relacionais. Outras das suas vantagens é o facto da grande maioria das bases de dados relacionais suportarem as propriedades *ACID* [44, 45, 46, 47, 48], Figura 4.40.

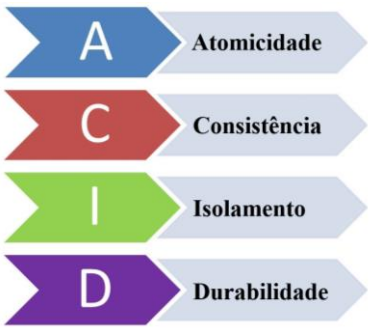


Figura 4.40: Propriedades do ACID

A primeira propriedade do *ACID* é a **Atomicidade.** Esta garante que apenas é executada a totalidade da operação. No caso não se verifique tal situação, a operação não será executada.

Outra propriedade do *ACID* é a **Consistência**. Esta propriedade previne que sejam criados ou editados registos que não cumpram as regras ou definições previamente estabelecidas. Em caso de tentativa de edição de registos que não cumpram as regras, este mantém os dados existentes em detrimento dos novos dados inseridos.

O **Isolamento** evita que várias operações em simultâneo tenham influências umas nas outras.

Por último, a propriedade da **Durabilidade.** Esta propriedade indica que se uma operação é efetuada, esta deverá permanecer mesmo em caso de falha ou reinício do sistema. Deste modo, toda a informação registada deverá ser guardada em memória não volátil.

4.4. GESTÃO DE CONTEÚDOS

Esta aplicação providencia a gestão de diversos conteúdos tais como contactos, orçamentos e informações de produtos. Essa gestão é feita com base em registos efetuados pelos diversos utilizadores.

Na gestão de contactos é possível obter informação sobre os clientes, fornecedores e utilizadores. Com base na informação disponível na base dados, será possível criar propostas, obter informações e características técnicas dos produtos. Estas informações apenas estão disponíveis aos utilizadores com acesso aos menus de "Clientes" e "Fornecedores". Sempre que existe uma alteração na informação é criado um registo na base de dados, de forma a ser possível identificar o autor dessas alterações.

Em relação à gestão de orçamentos, e como nem sempre é a primeira proposta enviada que satisfaz o cliente, é possível a criação de diversas revisões da proposta. Depois de criada a nova revisão, é possível alterar, apagar e/ou adicionar novos itens à mesma. Para visualização dessa informação basta o utilizador selecionar a revisão pretendida, Figura 4.24, e será apresentada a listagem das alterações efetuadas face à revisão anterior. A Figura 4.25 é um exemplo de como a informação está disponível para o utilizador, existindo nele duas alterações face à revisão anterior: foi adicionado o Sistema Automático de Deteção de Intrusão/Roubo e eliminado o Sistema Automático de Deteção de Incêndio, e o Sistema de Videovigilância manteve-se. A elaboração dos orçamentos é

suportada em dados de preços de mão-de-obra e matérias-primas existentes na base de dados.

Existem situações em que múltiplos clientes pedem cotações dos mesmos produtos, como por exemplo concursos, em que são várias empresas a concorrer à mesma obra. Nestes casos, e para que o utilizador não tenha a necessidade de inserir novamente todos os itens, foi implemento uma função que cria uma cópia integral dos dados da proposta selecionada, à exceção do cabeçalho onde se encontram os dados do cliente, sendo este cabeçalho substituído pela informação referente ao novo cliente. Esta função permite não só ganhar temo ao evitar criar diversas vezes a mesma proposta como previne eventuais erros ou omissões de produtos/equipamentos necessários.

A aplicação permite restringir os acessos aos menus "Clientes", "Fornecedores", "Propostas" e "Administração", conforme a Figura 4.41, para que nem todos os utilizadores tenham o acesso a toda a informação disponível na aplicação. Estes níveis apenas podem ser geridos pelo administrador do sistema, que consoante a necessidade pode criar, remover ou editar os diversos níveis.

O menu "Configuração" apenas está disponível para o nível de acesso "*Admin*". Além de restringir os menus, é possível ocultar igualmente informação relativamente aos preços dos produtos, tais como preço de venda ao público (PVP), preço de venda aos revendedores (PVR), bem como o preço de custo. No entanto, o acesso a esta informação deriva da anterior, ou seja, para o utilizador ver o preço de revenda é necessário ter acesso ao preço de venda, bem como para visualizar o preço de custo deverá ter o acesso aos anteriores.

Apesar de o utilizador ter acesso a determinados menus/informações, tal não significa que possa alterar ou apagar a informação. Para poderem modificar a informação, o utilizador deverá ter acesso à funcionalidade "Gerir" para poder criar registos, editar e apagar os existentes. No entanto, se o utilizador não tiver acesso à funcionalidade "Apagar", os registos continuam alojados no servidor, sendo que apenas os utilizadores com esse acesso poderão realmente remover os registos.

Na Figura 4.41 encontram-se ilustrados alguns dos possíveis níveis de controlo de acessos à aplicação, indicando o nome do nível, os menus e funcionalidades a que estes dão acesso. Estes níveis serão comuns para cada uma das empresas pelo que qualquer

alteração nos níveis de acesso irá se refletir em todos os utilizadores que contenham esse tipo de acesso.

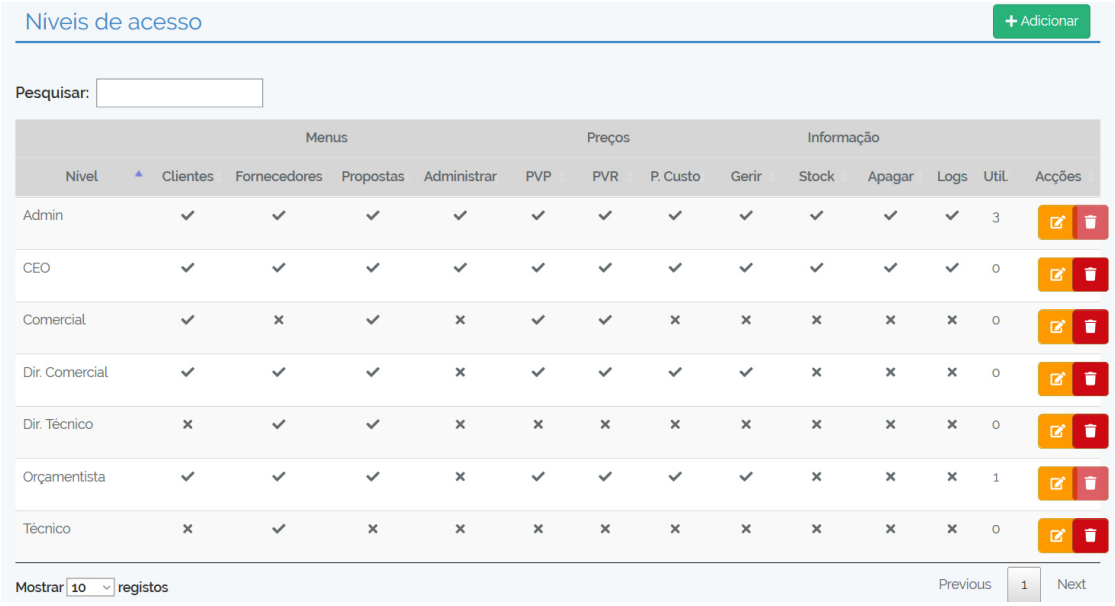


Figura 4.41: Exemplo de configuração de níveis de acesso

4.5. GESTÃO DE PROPOSTAS

De forma a acompanhar o desenvolvimento dos orçamentos, foi necessário implementar uma função que permitisse consultar as estatísticas das propostas efetuadas, apresentando como opções de seleção das propostas a data, empresa, sucursal, departamento de uma sucursal especifica e/ou por funcionário. Para a consulta efetuada é apresentado um gráfico com a respetiva percentagem de proposta por cada um dos seus estados, o tempo médio de execução e a listagem das mesmas para ser possível a sua consulta, como ilustrado na Figura 4.43.

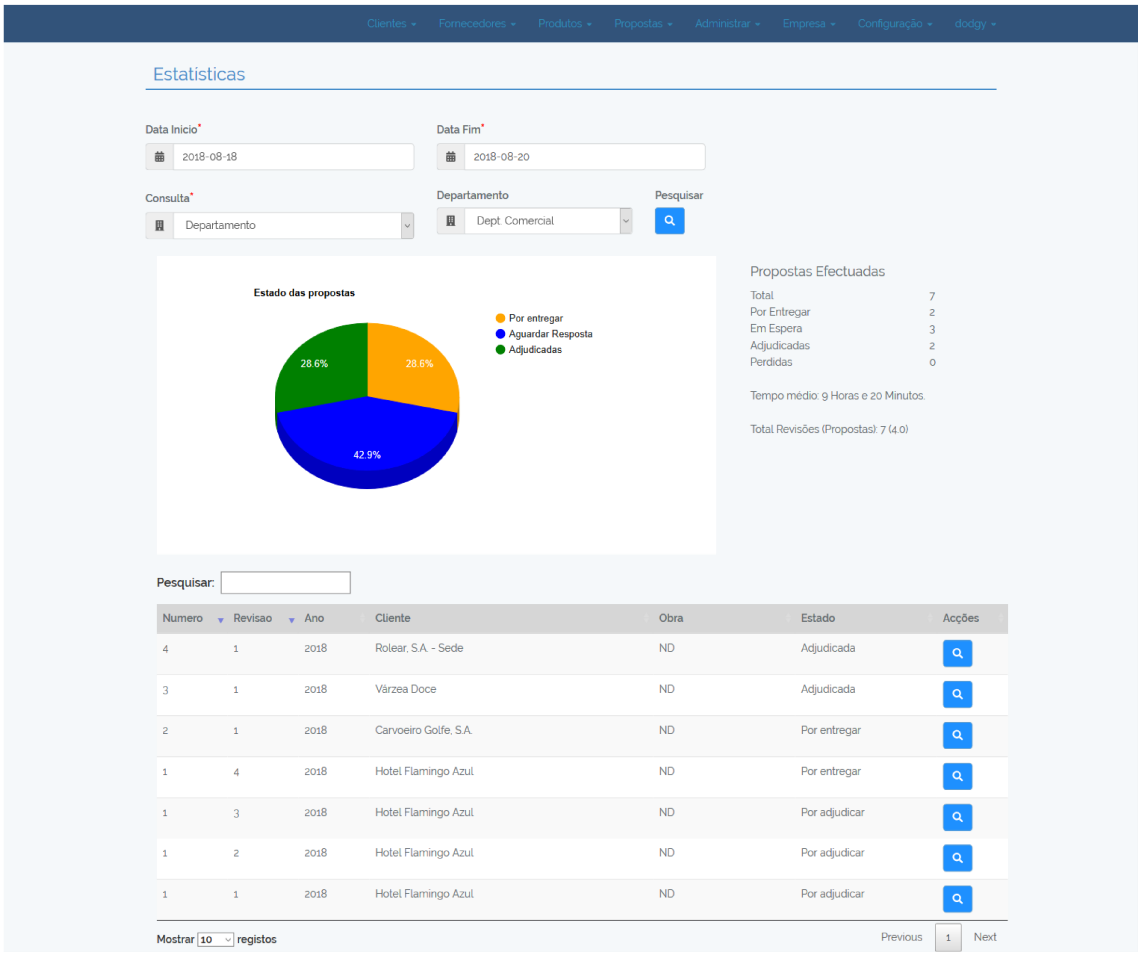


Figura 4.42: Exemplo de gestão de propostas

Concluída a implementação da aplicação de orçamentação com todas as funcionalidades necessárias, já é possível dar início ao exercício das funções de orçamentista. No entanto ainda é possível agilizar mais o processo de orçamentação com ajuda de um sistema de recomendação de forma a prever os produtos que possam vir a interessar ao cliente. Assim serão no próximo capítulo detalhados alguns dos sistemas de recomendação atuais e a forma como um destes sistemas foi implementado na aplicação.

5. SISTEMA DE RECOMENDAÇÃO

Um dos principais objetivos da aplicação é agilizar o processo de orçamentação, e para isso será necessário recorrer a um sistema de recomendação. Com este sistema, a aplicação de orçamentação irá permitir poupar tempo na criação de propostas, prever quais as necessidades do cliente, aumentar as margens de venda e, consequentemente aumentar a satisfação dos clientes. Pelo facto de existirem variadas técnicas para implementação de sistemas de recomendação, tais como as baseadas em conteúdo e as colaborativas, é possível combinar alguns métodos e criar sistemas híbridos, melhorando as recomendações dos produtos.

5.1. TIPOS DE SISTEMAS

5.1.1. SISTEMA DE RECOMENDAÇÃO BASEADO EM CONTEÚDOS

Os sistemas baseados em conteúdos [49, 50, 51] fazem a comparação de um conjunto de itens atualmente propostos com os adquiridos anteriormente pelos clientes. A cada item é associado pelo menos um descritor, como por exemplo, a categoria do produto e/ou o seu tipo.

Este tipo de sistemas analisam as características/propriedades dos diversos itens. Depois de efetuada a análise dos produtos bem como das características dos clientes, será criada uma lista de produtos de modo a que esta se enquadre nas necessidades de cada cliente, Figura 5.1.

A vantagem deste tipo de sistema é não necessitar de muita informação sobre o cliente para sugerir os produtos [49], necessitando apenas da categorização dos itens.

Uma das desvantagens deste tipo de sistema é a possibilidade de sugerir sempre itens que o cliente já adquiriu, ignorando outros que poderão interessar ao cliente, apesar deste nunca os ter adquirido. Outra das desvantagens deste tipo de sistema é a sua subjetividade, pois nem sempre é fácil categorizar os itens.

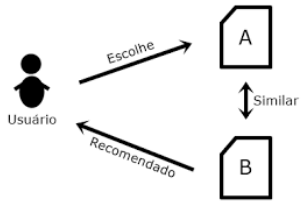


Figura 5.1: Sistema Baseado em Conteúdos

5.1.2. SISTEMA DE RECOMENDAÇÃO BASEADO EM FILTRAGEM COLABORATIVA

Os sistemas baseados em filtragem colaborativa [49, 50] consistem na recomendação de produtos que os clientes adquiriram no passado. Analisando um conjunto de clientes pressupõem-se que "Se um cliente A comprou os produtos A, B e C, e um outro cliente B que também comprou os produtos A e B, poderá vir a estar interessado em adquirir o produto C", Figura 5.2. Este tipo de recomendação apresenta bons resultados, evitando o problema existente nos sistemas baseados em conteúdos, onde poderiam existir recomendações repetitivas [49]. Ao contrário do sistema baseado em conteúdos, este sistema já requer um elevado número de registos de compras efetuadas pelos clientes.

Dentro dos sistemas baseados em filtragem colaborativa, é possível dividir em duas categorias: as baseadas em produtos e os baseados em clientes [49].

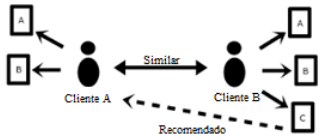


Figura 5.2: Sistemas Baseados em Filtragem Colaborativa

5.1.2.1. SISTEMAS BASEADOS EM CLIENTES

O sistema baseado em clientes [52, 53, 54, 55, 56] prevê o interesse do cliente num determinado produto com base no perfil de clientes semelhantes. Por exemplo, se o "Cliente A" adquiriu os produtos "A, B, C e D" e o "Cliente C" que também adquiriu os produtos "B e C" poderá estar interessado nos produtos "A e D", como ilustrado na Figura 5.3.

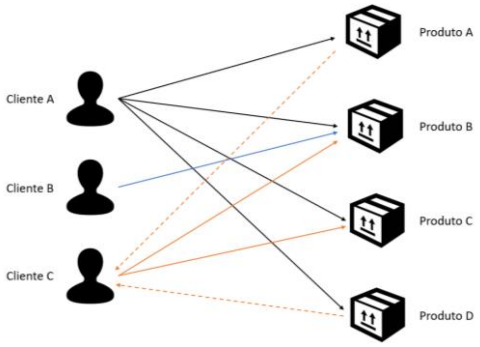


Figura 5.3: Esquema de recomendação Baseados em Clientes

Este sistema necessita de duas condições para o seu correto funcionamento: a listagem de clientes e a listagem de produtos. Após a obtenção destes dados, será possível a criação de uma matriz, onde serão relacionados os clientes com os produtos adquiridos.

Uma das vantagens deste sistema é a sua fácil implementação e obtenção de resultados mais precisos do que noutros sistemas abordados anteriormente [56]. No entanto, este sistema também tem algumas desvantagens, tal como a escalabilidade, ou seja, quanto mais clientes existirem no sistema maior o custo para encontrar clientes. Outra das desvantagens é quando é inserido um novo cliente ou produto, e por estes ainda não conterem nenhuma informação, será mais difícil obter recomendações utilizando este tipo de sistema.

5.1.2.2. SISTEMAS BASEADOS EM PRODUTOS

Os sistemas baseados em produtos [52, 55, 56, 57, 58, 59] funcionam de forma semelhante ao anterior, em que a diferença entre os sistemas é que neste a recomendação é efetuada com base nos produtos e o anterior é com base nos clientes. Observando a Figura 5.4 constata-se que se o "Produto C" foi comprado pelos "Clientes A, B e C" e o "Produto A" foi comprado pelos "Clientes A e B", então o "Cliente C" também poderá estar interessado neste "Produto A" [56].

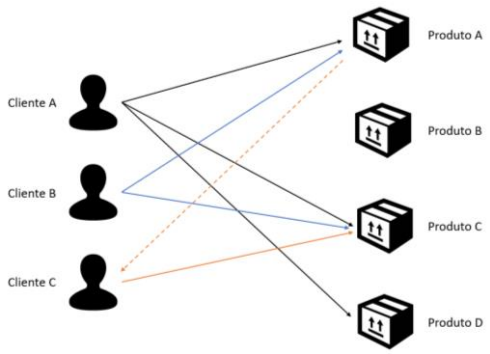


Figura 5.4: Esquema de recomendação Baseados em Produtos

O primeiro passo na filtragem colaborativa baseada em produtos consiste em calcular a similaridade entre produtos e posteriormente selecionar os mais semelhantes. Para se verificar a similaridade entre os dois produtos "i" e "j", deve-se isolar os clientes que adquiriram esses dois produtos, aplicando uma das formas de calcular a similaridade. Visto existirem diversas formas de calcular a similaridade [57] – $sim(i,j)$ – apenas serão abordadas duas delas, tratando-se das que serão baseadas em cossenos e as baseadas em correlações [57].

Como referido anteriormente, é necessário isolar os clientes que adquiriram os produtos, na Tabela 5.1, as linhas da matriz representam os clientes, as colunas representam os produtos adquiridos e "R" representa a quantidade de produto.

	1	2	…	i	…	j	….	n-1	n
1				R		R			
2				-		R			
⋮									
u				R		R			
⋮									
m-1				R		R			
m				R		-			

Tabela 5.1: Tabela de Isolamento dos produtos

No cálculo baseado em cossenos, os dois produtos são considerados como dois vetores, sendo a semelhança entre eles calculada pelo cosseno do ângulo formado por estes. O cosseno é calculado através do produto interno dos vetores $\vec{\imath}$ e $\vec{\jmath}$ dividindo o seu resultado pela raiz da norma de ambos os vetores, como representado na Equação 5.1 [57, 58].

$$sim(i,j) = \cos(\vec{\imath},\vec{\jmath}) = \frac{\vec{\imath}\,.\,\vec{\jmath}}{\|\vec{\imath}\|_2\,\|\vec{\jmath}\|_2}$$

Equação 5.1: Similaridade baseada em cossenos

Já nos casos baseados em correlações, o cálculo é efetuado com base na correlação de Pearson [60]. O conjunto de clientes que adquiriram os produtos "i" e "j" é denominado de "U", à classificação atribuída pelos clientes "U" aos produtos "i" e "j" é denominado por "$R_{u,i}$" e "$R_{u,j}$". Por fim, a classificação média atribuída aos produtos "i" e "j" é definida por "$\overline{R}_\imath$" e "$\overline{R}_\jmath$", Equação 5.2 [57].

$$sim(i,j) = \frac{\sum_{u\,\in\,U}\left(R_{u,i} - \overline{R}_\imath\right)\left(R_{u,j} - \overline{R}_\jmath\right)}{\sqrt{\sum_{u\,\in\,U}\left(R_{u,i} - \overline{R}_\imath\right)^2}\sqrt{\sum_{u\,\in\,U}\left(R_{u,j} - \overline{R}_\jmath\right)^2}}$$

Equação 5.2: Similaridade baseada em correlações

Para se entender melhor como funciona a similaridade, encontra-se representado na Tabela 5.2 um exemplo de classificações de quatro clientes para três produtos, verificando-se que alguns produtos ainda não foram classificados, representados por "?".

		Produtos		
		P1	P2	P3
Clientes	C1	?	2	3
	C2	5	?	2
	C3	3	3	?
	C4	1	2	2

Tabela 5.2: Matriz de classificações

Antes de se poder verificar a semelhança entre os produtos, é necessário calcular a sua similaridade, o que neste caso será utilizada a fórmula de cálculo baseado em cossenos, Equação 5.1. Para calcular a similaridade entre os produtos P1 – P2, é necessário obter dois vetores. Na Tabela 5.2 observa-se que os clientes C3 e C4 classificaram os referidos produtos P1 e P2, podendo-se utilizar a classificação dada por estes clientes para criar os vetores v_1 e v_2 que iriam corresponder aos produtos P1 e P2.

$$v_1 = 3C_3 + 1C_4 \qquad v_2 = 3C_3 + 2C_4$$

$$\cos(v_1, v_2) = \frac{(3,1).(3,2)}{\|(3,1)\|_2 \, \|(3,2)\|_2} = \frac{9+2}{\sqrt{(3^2 + 1^2)}\sqrt{(3^2 + 2^2)}}$$

$$\cos(v_1, v_2) = \frac{11}{\sqrt{(9+1)(9+4)}} = \frac{11}{\sqrt{130}} \cong 0.965$$

Equação 5.3: Cálculo da similaridade entre os vetores v_1 e v_2

Finalizados os cálculos da similaridade entre os produtos P1 – P2, Equação 5.3, é necessário determinar para P1 – P3, para estes utiliza-se a classificação atribuída pelos clientes C2 e C4, ficando os vetores a corresponder a v_1 e v_3.

$$v_1 = 5C_2 + 1C_4 \qquad v_3 = 2C_2 + 2C_4$$

$$\cos(v_1, v_3) = \frac{(5,1).(2,2)}{\|(5,1)\|_2 \, \|(2,2)\|_2} = \frac{10+2}{\sqrt{(25+1)}\sqrt{(4+4)}} = \frac{12}{\sqrt{208}} \cong 0.832$$

Equação 5.4: Cálculo da similaridade entre os vetores v_1 e v_3

Por último, para determinar a similaridade entre os produtos P2 – P3, v_2 e v_3, utiliza-se a classificação atribuída pelos clientes C1 e C4.

$$v_2 = 2C_1 + 2C_4 \qquad v_3 = 3C_1 + 2C_4$$

$$\cos(v_2, v_3) = \frac{(2,2).(3,2)}{\|(2,2)\|_2 \, \|(3,2)\|_2} = \frac{6+4}{\sqrt{(4+4)}\sqrt{(9+4)}} = \frac{10}{\sqrt{104}} \cong 0.981$$

Equação 5.5: Cálculo da similaridade entre os vetores v_2 e v_3

Após concluídos todos os cálculos da similaridade entre os produtos, é possível criar a matriz de similaridades, Tabela 5.3.

Produtos	P1	P2	P3
P1	1	0.965	0.832
P2	0.965	1	0.981
P3	0.832	0.981	1

Tabela 5.3: Matriz de similaridade

Clientes	P1	P2	P3
C1	1.075	2	3
C2	5	2.686	2
C3	3	3	?
C4	1	2	1.612

Tabela 5.4: Matriz de classificações com as respetivas previsões

Para prever a classificação atribuída pelo cliente C1 ao produto P1, utiliza-se as médias de similaridade calculadas entre P1 – P2 e P1 – P3, multiplicado pelas classificações atribuídas pelo cliente C1 aos produtos P2 e P3, C_1P_2 e C_1P_3 respetivamente, e dividindo tudo pelas médias de similaridade, ficando então a previsão definida pela Equação 5.7.

$$Previsão = \frac{(C_1P_2)(P_1 - P_2) + (C_1P_3)(P_1 - P_3)}{(P_1 - P_2) + (P_1 - P_3)}$$

Equação 5.6: Fórmula de cálculo da previsão

$$Previsão = \frac{2 * 0.965 + 3 * 0.832}{0.965 + 0.832} \cong 1.075$$

Equação 5.7: Cálculo da previsão da classificação do cliente C1 ao produto P1

5.2. ANÁLISE DO SISTEMA DE RECOMENDAÇÃO

Antes de implementar o sistema de recomendação, é necessário compreender como se processa uma proposta, seguindo um exemplo. Pretende-se implementar um sistema de videovigilância simples, como ilustrado na Figura 5.5. Este sistema é composto por um

servidor, quatro câmaras de exterior, com os respetivos suportes e alimentadores, cabo e calha técnica com 150 metros, como referenciado na Tabela 5.5.

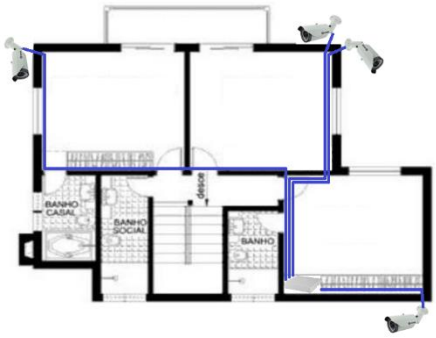


Figura 5.5: Exemplo de sistema de CCTV

Produto	Quantidade
Servidor 4 canais	1 un
Disco rígido 500gb	1 un
Câmara de Exterior	4 un
Alimentador	4 un
Suporte para câmara	4 un
Conector	8 un
Cabo	150 m
Calha técnica	150 m

Tabela 5.5: Listagem de equipamentos

Normalmente os sistemas de recomendação funcionam com um sistema de classificações, embora neste caso não existam classificações, a recomendação será efetuada com base nas quantidades. Com base nas quantidades do exemplo anterior, a lista de recomendação de produtos começava por recomendar o cabo e calha, como ilustrado na Tabela 5.6.

Posteriormente, ao se especificar os diferentes tipos de calha e cabos possíveis para cada sistema/instalação, tornar-se-ia mais difícil recomendar produtos vendidos em quantidades inferiores como, por exemplo, os servidores e discos rígidos. Para tornar a recomendação mais adequada, optou-se por considerar as quantidades dos produtos como unitárias por proposta, alterando com isto a ordem com que os produtos serão recomendados, Tabela 5.7.

Produto	Quantidade
Cabo	150 m
Calha técnica	150 m
Conector	8 un.
Alimentador	4 un
Câmara de Exterior	4 un
Suporte para câmara	4 un
Disco rígido 500gb	1 un
Servidor 4 canais	1 un

Tabela 5.6: Lista de recomendação por quantidade real proposta

Produto	Quantidade
Alimentador	1 un
Cabo	1 un
Calha técnica	1 un
Câmara de Exterior	1 un
Conector	1 un
Disco rígido 500gb	1 un
Servidor 4 canais	1 un
Suporte para câmara	1 un

Tabela 5.7: Lista de recomendação por quantidade unitária

Apesar da empresa fornecer diversos tipos de sistemas, para os exemplos seguintes apenas foi considerado o sistema de videovigilância, por ser o sistema mais simples de analisar.

Na Tabela 5.8 encontra-se ilustrado o equivalente a mais cinco propostas efetuadas para o Sistema de Videovigilância CCTV.

Produtos	Propostas					
	1	2	3	4	5	6
Servidor 4 canais	1 un	0 un	0 un	0 un	0 un	0 un
Servidor 8 canais	0 un	1 un	0 un	1 un	0 un	0 un
Servidor 16 canais	0 un	0 un	1 un	0 un	0 un	2 un
Disco rígido 500gb	1 un	0 un	0 un	0 un	0 un	0 un
Disco rígido 1tb	0 un	1 un	2 un	1 un	0 un	4 un
Câmara de Exterior	4 un	6 un	6 un	0 un	2 un	22 un
Câmara de Interior	0 un	2 un	6 un	8 un	1 un	8 un
Alimentador	4 un	8 un	12 un	8 un	3 un	30 un
Suporte para câmara	4 un	8 un	12 un	8 un	0 un	30 un
Conector	8 un	16 un	0 un	16 un	0 un	60 un
Cabo	150 m	220 m	0 m	0 m	0 m	1.280 m
Calha técnica	150 m	220 m	0 m	0 m	0 m	0 m

Tabela 5.8: Exemplos de propostas de CCTV

Analisando as propostas presentes na Tabela 5.8, confirma-se que os produtos propostos com as maiores quantidades voltam a ser o cabo e a calha técnica. Se não se considerassem as quantidades das propostas como unitárias, a próxima listagem de recomendação seria como ilustrado na Tabela 5.9 com o cabo a surgir em primeiro, seguido da calha técnica e posteriormente os conectores. Ignorando as quantidades reais propostas, e analisando as vezes que determinado produto foi proposto, a listagem torna-se diferente, começando por recomendar os alimentadores, passando posteriormente pelas câmaras de exterior e interior, Tabela 5.10.

Produto	Quantidade
Cabo	1.650 m
Calha técnica	370 m
Conector	100 un
Alimentador	65 un
Suporte para câmara	62 un
Câmara de Exterior	40 un
Câmara de Interior	25 un
Disco rígido 1tb	8 un
Servidor 16 canais	3 un
Servidor 8 canais	2 un
Disco rígido 500gb	1 un
Servidor 4 canais	1 un

Tabela 5.9: Listagem de produtos das propostas por quantidade real

Produto	Quantidade
Alimentador	6
Câmara de Exterior	5
Câmara de Interior	5
Suporte para câmara	5
Conector	4
Disco rígido 1tb	4
Cabo	3
Calha técnica	2
Servidor 8 canais	2
Servidor 16 canais	2
Disco rígido 500gb	1
Servidor 4 canais	1

Tabela 5.10: Listagem de produtos das propostas por quantidade unitária

5.3. IMPLEMENTAÇÃO DO SISTEMA

O sistema de recomendação num sistema de orçamentação pode sugerir novos produtos que possam a vir interessar aos clientes, bem como, escoar produtos em *stock* em armazém. Como tal, o sistema implementado nesta aplicação consiste nessas duas vertentes, aplicando o conceito da análise efetuada anteriormente.

O sistema de recomendação da aplicação é baseado no conceito dos sistemas de recomendação utilizados por algumas empresas tais como a *Amazon*, *Youtube* e *Netflix* [61] que aproveitam estes sistemas para preverem as necessidades dos seus clientes. Geralmente estas empresas utilizam os produtos que os clientes inserem no "carrinho" e fazem diversas iterações com a respetiva informação.

Na aplicação de orçamentação serão utilizados os produtos que o utilizador/orçamentista selecionar, Figura 5.6. Ao adicionar um produto à proposta, vai permitir ao utilizador/orçamentista consultar na aba "Recomendações" a lista de produtos recomendados pelo sistema.

Figura 5.6: Produtos propostos

O algoritmo presente na seguinte Figura 5.7 é utilizado como base para o sistema de recomendação implementado na aplicação de orçamentação. A aplicação da empresa *Amazon* [58] é exemplo de desenvolvimento deste algoritmo.

Por cada produto no catalogo de produtos, "P1";
For cada cliente "C" que comprou os produtos "P1";
Por cada produto "P2" comprado pelos clientes "C";
Listagem dos clientes que compraram "P1" e "P2";
Por cada item "P2";
Calcular a similaridade entre i1 e i2;

Figura 5.7: Algoritmo genérico de recomendação utilizado pela Amazon [4]

Em primeiro lugar é executado uma iteração de forma a obter todos os produtos inseridos, como indicado na secção "P1", linhas 1495 a 1500, da Figura 5.8.

Completada a primeira iteração, é efetuada uma segunda iteração para obter a listagem de todos os clientes que já adquiriram os mesmos produtos, secção "C" da Figura 5.8, linhas 1501 a 1508.

Posteriormente, e com base na listagem de clientes, é efetuada uma nova iteração para listar os produtos adquiridos por esses clientes listados anteriormente, secção "P2" da Figura 5.8, linhas 1509 a 1517.

[4] Imagem adaptada do artigo "*Amazon.com Recommendations Item-to-Item Collaborative Filtering*" [54]

```php
public function obterRecomendacoes(Request $request)
{
    $user               = User::where('id', Auth::user()->id)->first();
    $clientes           = Cliente_Contacto::where('empresa_id', $user->empresa_id)->get();
    $produtos           = Produto::where('empresa_id', $user->empresa_id)->get();
    $proposta_actual    = Proposta::findOrFail($request->id);
    $sistemas_propostos = Proposta_Sistema::where('proposta_id', $request->id)->select('subcat_id')->get();
    $sistemas_propostos->toArray();

    /* P1 - LISTA DE TODOS OS PRODUTOS PROPOSTOS */
    $produtos_propostos = DB::table('propostas_sistemas AS a')
                        ->join('propostas_produtos AS b', 'a.id', 'b.sistema_id')
                        ->where('proposta_id', $request->id)->distinct()
                        ->orderBy('produto_id')->pluck('produto_id');
    /* P1 - LISTA DE TODOS OS PRODUTOS PROPOSTOS */
    /* C  - LISTA DE TODOS OS CLIENTES QUE ADQUIRIRAM OS PRODUTOS PROPOSTOS */
    $clientes_aquiquiriram_produtos_propostos = DB::table('propostas_sistemas AS a')
                        ->join('propostas_produtos AS b', 'a.id', 'b.sistema_id')
                        ->join('propostas AS c', 'a.proposta_id', 'c.id')
                        ->join('clientes_contactos AS d', 'c.contacto_id', 'd.id')
                        ->whereIn('produto_id', $produtos_propostos)
                        ->distinct()->pluck('cliente_id');
    /* C  - LISTA DE TODOS OS CLIENTES QUE ADQUIRIRAM OS PRODUTOS PROPOSTOS */
    /* P2 - LISTA DE TODOS OS PRODUTOS AQUIRIDOS PELOS CLIENTES */
    $produtos_anteriormente_adquiridos_clientes = DB::table('propostas_sistemas AS a')
                        ->join('propostas_produtos AS b', 'a.id', 'b.sistema_id')
                        ->join('propostas AS c', 'a.proposta_id', 'c.id')
                        ->join('clientes_contactos AS d', 'c.contacto_id', 'd.id')
                        ->whereIn('cliente_id', $clientes_aquiquiriram_produtos_propostos)
                        ->distinct()
                        ->orderBy('produto_id')->pluck('produto_id');
    /* P2 - LISTA DE TODOS OS PRODUTOS AQUIRIDOS PELOS CLIENTES */
```

Figura 5.8: Algoritmo de recomendação de produtos

Após a obtenção da listagem de produtos inseridos e da listagem de produtos adquiridos pelos clientes, é efetuado o cálculo de similaridade, Figura 5.9, e entre eles utilizada a fórmula de cálculo baseada em cossenos, referida na Equação 5.1.

```
/* CRIAR MATRIZ DE SIMILARIDADE */
$matriz_similaridade = [];
foreach($produtos_anteriormente_adquiridos_clientes as $p1){
    $v1 = array();
    $v2 = array();
    $vetor_produtos_1 = DB::table('propostas_sistemas AS a')
                    ->join('propostas_produtos AS b', 'a.id', 'b.sistema_id')
                    ->join('propostas AS c', 'a.proposta_id', 'c.id')
                    ->join('clientes_contactos AS d', 'c.contacto_id', 'd.id')
                    ->where('produto_id', $p1)
                    ->select('cliente_id', 'produto_id', DB::raw("count('cliente_id') as quantidade"))
                    ->distinct()
                    ->groupBy('cliente_id', 'produto_id')
                    ->orderBy('produto_id')->orderBy('cliente_id')->get();

    if($vetor_produtos_1){
        foreach($vetor_produtos_1 as $vetor_1){
            $v1[$vetor_1->cliente_id] = $vetor_1->quantidade;
        }
    }
    foreach($produtos_anteriormente_adquiridos_clientes as $p2){
        if($p1 == $p2){
            $matriz_similaridade[$p1][$p2] = 1;
        }else{
            $vetor_produtos_2 = DB::table('propostas_sistemas AS a')
                    ->join('propostas_produtos AS b', 'a.id', 'b.sistema_id')
                    ->join('propostas AS c', 'a.proposta_id', 'c.id')
                        ->join('clientes_contactos AS d', 'c.contacto_id', 'd.id')
                        ->where('produto_id', $p2)
                        ->select('cliente_id', 'produto_id', DB::raw("count('cliente_id') as quantidade"))
                        ->distinct()
                        ->groupBy('cliente_id', 'produto_id')
                        ->orderBy('produto_id')->orderBy('cliente_id')->get();

            if($vetor_produtos_2){
                foreach($vetor_produtos_2 as $vetor_2){
                    $v2[$vetor_2->cliente_id] = $vetor_2->quantidade;
                }
            }

            $produto_similaridade  = 0;
            $divisor_similaridade1 = 0;
            $divisor_similaridade2 = 0;
            foreach($v1 as $iv1 => $v_1){
                foreach($v2 as $iv2 => $v_2){
                    if($iv1 == $iv2){
                        $produto_similaridade = $produto_similaridade + $v_1 * $v_2;
                        $divisor_similaridade1 = $divisor_similaridade1 + $v_1 * $v_1;
                        $divisor_similaridade2 = $divisor_similaridade2 + $v_2 * $v_2;
                    }
                }
            }

            if($divisor_similaridade1 * $divisor_similaridade2 > 0){
                $similaridade = $produto_similaridade / sqrt($divisor_similaridade1 * $divisor_similaridade2);
                $matriz_similaridade[$p1][$p2] = $similaridade;
            }else{
                $matriz_similaridade[$p1][$p2] = 0;
            }
        }
    }
}
/* CRIAR MATRIZ DE SIMILARIDADE */
```

Figura 5.9: Algoritmo de matriz de similaridade

Como ilustrado na Figura 5.6, foram adicionados um produto para a categoria “Sistema de Videovigilância – CCTV” e outro para a categoria “Sistema Automático de Deteção de Incêndio – SADI”. Com base nas propostas efetuadas anteriormente pela aplicação e onde foram igualmente adquiridos os mesmos equipamentos, o sistema recomendou três produtos que existem em armazém, e nove produtos que foram anteriormente adquiridos, Figura 5.10.

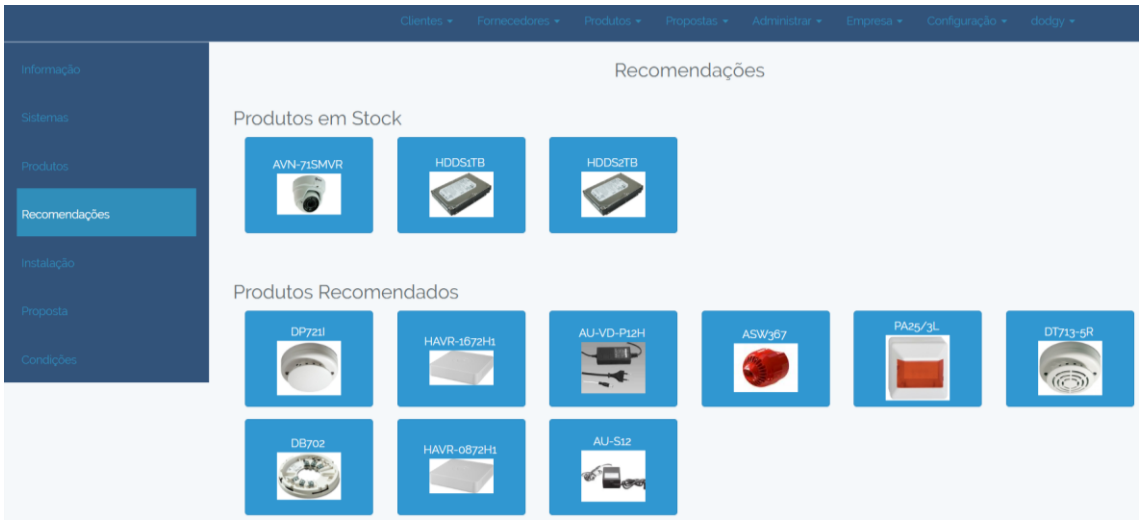


Figura 5.10: Produtos recomendados

De forma a evitar as sucessivas atualizações da página, o que tornaria a aplicação demasiado lenta e pouco funcional, recorreu-se à função *AJAX* de forma a atualizar a informação necessária e sem que o utilizador se aperceba. Esta função é executada cada vez que o utilizador seleciona a opção "Recomendações" e esta recebe o número que corresponde à proposta em questão. A função irá enviar uma requisição através da *URL* "/proposta.obterrecomendacoes", Figura 5.11, em conjunto com o método "PUT". Consultando a lista de rotas definidas no ficheiro "web.php" ou executando o comando "php artisan route:list", Figura 4.23, observa-se que este executa o método "obterRecomendacoes" presente no *Controller* "PropostaController", Figura 5.12.

```
PUT       | produtos_novo_valor/{id}          | produtos_novo_valor.update          | App\Http\Controllers\ProdutoController@updatePreco
PUT       | proposta.obterrecomendacoes       | proposta.obterrecomendacoes         | App\Http\Controllers\PropostaController@obterRecomendacoes
POST      | propostas                         | propostas.store                     | App\Http\Controllers\PropostaController@store
GET|HEAD  | propostas                         | propostas.index                     | App\Http\Controllers\PropostaController@index
```

Figura 5.11: Rota para função de recomendação

```
function obter_recomendacoes(proposta_id){

    $.ajax({
        url: '/proposta.obterrecomendacoes',
        type: "PUT",
        dataType: "json",
        data: $.param({
            "id": proposta_id,
        }),
        success: function (data) {
            $('#listagem_produtos_stock').empty();
            $.each(data.produtos_stock, function (index, element) {
                $('#listagem_produtos_stock').append(`
                    <div class="col-md-2 div_produtos_propostas">
                        <button id="btn-produto${element.id}" type="button"
                        class="btn btn-primary btn-sistema" data-toggle="modal"
                        data-target="#info-produto-modal"
                        onClick="EditarModal(${element.id})" data-id="${element.id}">
                            ${element.referencia}
                            <br>
                            <img src="/storage/produtos/imagens/${element.imagem}" width="80" />
                        </button>
                    </div>`);
            });
            $('#listagem_produtos_recomendados').empty();
            $.each(data.propor_produtos, function (index, element) {
                $('#listagem_produtos_recomendados').append(`
                    <div class="col-md-2 div_produtos_propostas">
                        <button id="btn-produto${element.id}" type="button"
                        class="btn btn-primary btn-sistema" data-toggle="modal"
                        data-target="#info-produto-modal"
                        onClick="EditarModal(${element.id})" data-id="${element.id}">
                            ${element.referencia}
                            <br>
                            <img src="/storage/produtos/imagens/${element.imagem}" width="80" />
                        </button>
                    </div>`);
            });
        },
        error: function () {
            alert('Sem recomendações');
        }
    });
}
```

Figura 5.12: Função AJAX para listagem de produtos recomendados

Depois de ser efetuado o pedido, e em caso de sucesso, a função "obterRecomendacoes" inclusa no "PropostaController" envia a informação pretendida à função "obter_recomendacoes". Assim que a função "obter_recomendacoes" obtém a informação solicitada, serão efetuados dois procedimentos para listar a informação fornecida pelo servidor, como ilustrado na Figura 5.12. No primeiro procedimento será listada toda a informação referente aos produtos em *stock*, colocando a informação na "div" cujo "id" corresponde à "listagem_produtos_stock", Figura 5.13. Já no segundo será listada a informação dos produtos recomendados pelo sistema baseados nas propostas anteriores, e tal como no ciclo anterior este guarda a informação numa "div", cujo "id" é "listagem_produtos_recomendados".

```
1  <div class="container">
2      <div class="row">
3          <div id="listagem_produtos_stock"></div>
4          <div id="listagem_produtos_recomendados"></div>
5      </div>
6  </div>
```

Figura 5.13: Conteúdo HTML referente à aba de recomendação

Na proposta da Figura 5.6, foi selecionado o produto "AWN-72SMVR". Com base na informação relacionada com esse produto, foram selecionadas algumas das propostas anteriores onde foi proposto esse mesmo produto, Tabela 5.11.

		Propostas				
		2	3	4	5	12
Produtos	HAVR-0472H1	0	0	0	0	0
	HAVR-0872H1	0	0	0	1	1
	HAVR-1672H1	1	1	1	0	0
	AWN-72SMVR	1	1	1	1	1
	AVN-71SMVR	1	1	0	0	0
	AU-VD-P12H	1	1	1	0	1
	AU-S12	0	0	0	1	1
	AU-S12A48	0	0	0	0	0
	HDDS1TB	0	0	0	0	0
	HDDS2TB	0	0	0	0	1
	HDDS4TB	1	1	1	1	0
	1X-F2-xx	1	0	0	0	1
	1X-F4-xx	0	0	0	0	0
	1X-F8-xx	0	0	0	0	0
	DB702	1	0	0	0	1
	DP721I	1	0	0	0	1
	DT713-5R	1	0	0	0	1
	DMN700L	1	0	0	0	1
	PA25/3L	1	0	0	0	1
	AS363	0	0	0	0	0
	ASW367	1	0	0	0	1
	JA-82K	0	0	1	0	1
	JA-81E	0	0	1	0	1
	JS-20 LARGO	0	0	1	0	1

Tabela 5.11:Lista de propostas

Após calcular a matriz de similaridade, é necessário prever a probabilidade de recomendação para cada um dos produtos, Equação 5.7, recorrendo a quantidade de produtos adquiridos anteriormente pelo cliente, Figura 5.14.

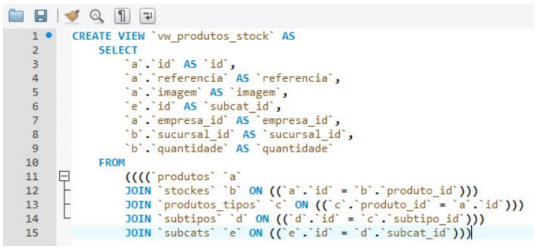
Concluídos todos os cálculos, será então selecionados uma listagem de até doze produtos ordenados com melhor qualificação. Uma vez que na matriz de similaridade apresenta a listagem de todos os produtos, de forma a não apresentar produtos que não sejam do sistema proposto, foi essencial incorporar um filtro na seleção dos produtos, impedindo o sistema recomendar produtos de outros sistemas.

```php
/* OBTER LISTA DE PRODUTOS ADQUIRIDOS ANTERIORMENTE PELO CLIENTE */
$cliente_proposta = DB::table('propostas_sistemas AS a')
                        ->join('propostas_produtos AS b', 'a.id', 'b.sistema_id')
                        ->join('propostas AS c', 'a.proposta_id', 'c.id')
                        ->join('clientes_contactos AS d', 'c.contacto_id', 'd.id')
                        ->where('proposta_id', $request->id)
                        ->distinct()
                        ->select('cliente_id')->first();

$produtos_adquiridos_anteriormente = DB::table('propostas_sistemas AS a')
                        ->join('propostas_produtos AS b', 'a.id', 'b.sistema_id')
                        ->join('propostas AS c', 'a.proposta_id', 'c.id')
                        ->join('clientes_contactos AS d', 'c.contacto_id', 'd.id')
                        ->where('cliente_id', $cliente_proposta->cliente_id)
                        ->whereNotIn('produto_id', $produtos_propostos)
                        ->distinct()
                        ->select('produto_id', DB::raw("count('produto_id') as quantidade"))
                        ->groupBy('produto_id')
                        ->orderBy('produto_id')->get();
/* OBTER LISTA DE PRODUTOS ADQUIRIDOS ANTERIORMENTE PELO CLIENTE */
/* CRIAR VECTOR DE QUANTIDADES ADQUIRIDAS POR PRODUTO */
foreach($produtos as $produto){
    $array_produtos[$produto->id] = 0;
}
foreach($produtos_adquiridos_anteriormente as $produto_adquirido){
    $array_produtos[$produto_adquirido->produto_id] = $produto_adquirido->quantidade;
}
/* CRIAR VECTOR DE QUANTIDADES ADQUIRIDAS POR PRODUTO */
/* CALCULAR VALOR DE RECOMENDAÇÃO POR PRODUTO */
foreach($array_produtos as $i1 => $p1){
    foreach($matriz_similaridade[$cliente_proposta->cliente_id] as $i2 => $p2){
        if($i1 == $i2){
            $recomendacao[$i1] = $p1 * $p2;
        }elseif($i2 > $i1){
            break;
        }
    }
}
/* CALCULAR VALOR DE RECOMENDAÇÃO POR PRODUTO */
/* LISTAR 12 PRODUTOS COM MELHOR CLASSIFICAÇÃO DE RECOMENDAÇÃO */
arsort($recomendacao);
$recomendacao = array_slice($recomendacao, 0, 12, true);
$lista_recomendacao = '';
foreach($recomendacao as $index => $value){
    if($value == 0){
        unset($recomendacao[$index]);
    }else{
        $recomendacao[$index] = $index;
    }
    if(!$lista_recomendacao){
        $lista_recomendacao = $index;
    }else{
        $lista_recomendacao = $lista_recomendacao.', '.$index;
    }
}
$propor_produtos = DB::table('produtos AS a')
                        ->join('produtos_tipos AS b', 'a.id', 'b.produto_id')
                        ->join('subtipos AS C', 'c.id', 'b.subtipo_id')
                        ->whereIn('subcat_id', $sistemas_propostos)
                        ->whereIn('a.id', $recomendacao)
                        ->select('a.*')
                        ->orderByRaw(DB::raw("FIELD(a.id,".$lista_recomendacao.")"))
                        ->distinct()->take(12)->get();
/* LISTAR 12 PRODUTOS COM MELHOR CLASSIFICAÇÃO DE RECOMENDAÇÃO */
```

Figura 5.14: Algoritmo de seleção de produtos recomendados e produtos em stock

Como a empresa também necessita de escoar os produtos existentes em armazém, será igualmente apresentada uma listagem até doze produtos. Para facilitar a consulta, foi criada uma vista denominada por "vw_produtos_stock", Figura 5.15, guardando os resultados obtidos como ilustrado na Figura 5.16.



```
CREATE VIEW `vw_produtos_stock` AS
    SELECT
        `a`.`id` AS `id`,
        `a`.`referencia` AS `referencia`,
        `a`.`imagem` AS `imagem`,
        `e`.`id` AS `subcat_id`,
        `a`.`empresa_id` AS `empresa_id`,
        `b`.`sucursal_id` AS `sucursal_id`,
        `b`.`quantidade` AS `quantidade`
    FROM
        ((((`produtos` `a`
        JOIN `stockes` `b` ON ((`a`.`id` = `b`.`produto_id`)))
        JOIN `produtos_tipos` `c` ON ((`c`.`produto_id` = `a`.`id`)))
        JOIN `subtipos` `d` ON ((`d`.`id` = `c`.`subtipo_id`)))
        JOIN `subcats` `e` ON ((`e`.`id` = `d`.`subcat_id`)))
```

Figura 5.15: Criação da vista vw_produtos_stock

id	referencia	imagem	subcat_id	empresa_id	sucursal_id	quantidade
1	HAVR-0472H1	NULL	1	1	1	0
1	HAVR-0472H1	NULL	1	1	2	0
1	HAVR-0472H1	NULL	2	1	1	0
1	HAVR-0472H1	NULL	2	1	2	0
2	HAVR-0872H1	NULL	1	1	1	0
2	HAVR-0872H1	NULL	1	1	2	0
2	HAVR-0872H1	NULL	2	1	1	0
2	HAVR-0872H1	NULL	2	1	2	0
3	HAVR-1672H1	NULL	1	1	1	0
3	HAVR-1672H1	NULL	1	1	2	0
3	HAVR-1672H1	NULL	2	1	1	0
3	HAVR-1672H1	NULL	2	1	2	0
4	AWN-71SMIR	NULL	1	1	1	0
4	AWN-71SMIR	NULL	1	1	2	0
5	AWN-72SMVR	NULL	1	1	1	0
5	AWN-72SMVR	NULL	1	1	2	0
6	AVN-71SMIR	NULL	1	1	1	0
6	AVN-71SMIR	NULL	1	1	2	0

Figura 5.16: Resultados da vista vw_produtos_stock

Ao longo deste capítulo foi possível ter conhecimento de alguns dos tipos de sistemas de recomendações existentes no mercado, bem como das suas vantagens e desvantagens. A escolha do sistema de recomendação recaiu num sistema baseado em produtos, tal como é utilizado pela *Amazon* e *Ebay*. Este sistema aliado a uma recomendação de produtos em stock ajuda nos casos em que não existe informação referente aos novos clientes/produtos.

6. CONSIDERAÇÕES FINAIS

6.1. CONCLUSÃO

Neste relatório foram apresentadas as várias etapas para a criação de uma aplicação de orçamentação, utilizando diversas tecnologias. Este trabalho iniciou-se com a análise da função de orçamentação, de modo a ser possível conhecer os passos, e consequentemente as funcionalidades necessárias para desempenhar a função.

Concluída a análise e o conhecimento das funcionalidades necessárias para o desempenho da função, foi imprescindível analisar as tecnologias existentes e verificar quais as que mais se enquadravam no projeto.

Depois da análise das funções e das tecnologias, projetou-se a fase de implementação da aplicação. O processo começou com um estudo minucioso para que a aplicação fosse a mais intuitiva possível, sem comprometer a sua utilização. No entanto, ainda era necessário agilizar ainda mais o processo de orçamentação. Desse modo nasceu a necessidade de incorporar um sistema de recomendação de produtos, permitindo recomendar produtos que pudessem interessar ao cliente.

Foram analisados dois tipos de sistemas de recomendação, destacando-se o sistema de recomendação por filtragem colaborativa, ao qual deu origem ao sistema presente para a recomendação dos produtos na aplicação. Este sistema baseia-se na informação dos produtos adquiridos anteriormente por outros clientes, para que consiga sugerir novos produtos no orçamento atual.

Com a finalização do processo referente ao orçamento, foi necessário incluir uma função que permitisse aos elementos responsáveis da empresa efetuar a avaliação do orçamento. Essa avaliação poderá ser efetuada mediante a consulta das margens da proposta e a consulta das estatísticas de duração de realização da proposta. A função das estatísticas permite de igual modo obter diversa informação, nomeadamente o número de propostas efetuadas num período de tempo por sucursal, departamento, função e funcionário.

De forma geral, julga-se que os principais objetivos deste projeto foram cumpridos. No entanto, ainda existe caminho a percorrer até à obtenção de um produto final comercializável, sendo possível criar mais algumas funcionalidades que serão descritas seguidamente na secção do Trabalho Futuro.

6.2. TRABALHO FUTURO

Numa fase futura deste projeto, a partir da diversa informação contida na base de dados seria interessante a implementação de novos módulos/funcionalidades que permitissem dar continuidade ao trabalho efetuado na área da orçamentação.

Um dos trabalhos previstos é a criação de um módulo para auxílio do departamento técnico. Uma vez que cada produto contém uma previsão do tempo de instalação, esse módulo poderia permitir à parte técnica gerir o trabalho consoante as propostas adjudicadas.

Uma outra sugestão de trabalho poderia ser a criação de uma Interface de Programação de Aplicações, designada por API. Esta serviria para a interligação da aplicação de orçamentação ao sistema de faturação existente na empresa. Toda a informação necessária já se encontra no sistema, tais como os dados dos clientes e dos produtos.

7. BIBLIOGRAFIA

[1] A. M. Rodrigues da Silva e C. A. Escaleira Videira, UML, Metodologias e Ferramentas CASE, V. N. Famalicão: Centro Atlântico, Lda., 2001.

[2] A. A. A. Jilani, M. Usman, A. Nadeem, Z. I. Malik e Z. Halim, "Comparative Study on DFD to UML Diagrams Transformations," *World of Computer Science and Information Technology Journal(WCSIT),* pp. 10-16, 2011.

[3] M. Nunes e H. O'Neill, Fundamental do UML, FCA - Editora de Informática, Lda., 2004.

[4] Visual Paradigm International Ltd., "What is Entity Relationship Diagram (ERD)?," [Online]. Available: https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/. [Acedido em 04 Dezembro 2018].

[5] I.-Y. Song e K. Froehlich, "A Practical Guide to Entity-Relationship Modeling," *College of Information Science and Technology Drexel University,* pp. 213-232.

[6] M. Demba, "An Algorithmic Approach to Database Normalization," em *Future Generation Computer Systems*, 2003, pp. 57-65.

[7] T. Otwell, "Laravel," [Online]. Available: https://laravel.com/. [Acedido em 13 Fevereiro 2018].

[8] "GIT," Software Freedom Conservancy, [Online]. Available: https://git-scm.com/. [Acedido em 08 Janeiro 2018].

[9] N. Adremann e J. Boggiano, "COMPOSER - Dependency Manager for PHP," 06 Julho 2017. [Online]. Available: https://getcomposer.org.

[10] T. Crawford e T. Hussain, "A Comparison of Server Side Scripting Technologies," em *Proceedings of the International Conference on Software Engineering Research and Practice (SERP)*, Atenas, 2017.

[11] A. Pereira e C. Poupa, Linguagens Web, Lisboa: Edições Sílabo, 2013.

[12] Refsnes Data, "W3SCHOOLS Online Web Tutorials," [Online]. Available: https://www.w3schools.com/. [Acedido em 06 Julho 2017].

[13] The Php Group, “PHP,” [Online]. Available: http://php.net/. [Acedido em 06 Julho 2017].

[14] Pluralsight, “JavaScript,” 2019. [Online]. Available: https://www.javascript.com/. [Acedido em 17 Dezembro 2017].

[15] Wikimedia Foundation, Inc., “Composer (software),” [Online]. Available: https://en.wikipedia.org/wiki/Composer_(software). [Acedido em 20 Novembro 2018].

[16] Wikimedia Foundation, Inc., “GIT,” [Online]. Available: https://pt.wikipedia.org/wiki/Git. [Acedido em 08 Janeiro 2018].

[17] “Basic Git commands,” Atlassian, [Online]. Available: https://confluence.atlassian.com/bitbucketserver/basic-git-commands-776639767.html. [Acedido em 08 Janeiro 2018].

[18] Wikimedia Foundation, Inc., “XAMPP,” [Online]. Available: https://pt.wikipedia.org/wiki/XAMPP. [Acedido em 20 Novembro 2017].

[19] “XAMPP,” Apache Friends, 2019. [Online]. Available: https://www.apachefriends.org/. [Acedido em 06 Julho 2017].

[20] Oracle Corporation, “MySQL,” [Online]. Available: https://www.mysql.com/. [Acedido em 06 Julho 2017].

[21] Wikimedia Foundation, Inc., “PHP,” [Online]. Available: https://pt.wikipedia.org/wiki/PHP. [Acedido em 20 Novembro 2018].

[22] Wikimedia Foundation, Inc., “Server-side,” [Online]. Available: https://en.wikipedia.org/wiki/Server-side. [Acedido em 20 Novembro 2018].

[23] coderseye.com, “FREE CODING GUIDES FOR BEGINNERS,” 2018. [Online]. Available: https://coderseye.com/best-php-frameworks-for-web-developers/. [Acedido em 13 Fevereiro 2018].

[24] M. Bean, Laravel 5 Essentials, Packt Publishing Ltd, 2015.

[25] M. Stauffer, Laravel: Up & Running: A Framework for Building Modern PHP Apps, O'Reilly Media, Inc., 2019.

[26] Wikimedia Foundation, Inc., “Laravel,” [Online]. Available: https://en.wikipedia.org/wiki/Laravel. [Acedido em 20 Novembro 2018].

[27] Wikimedia Foundation, Inc., “MVC,” [Online]. Available: https://pt.wikipedia.org/wiki/MVC. [Acedido em 26 Julho 2018].

[28] Wikimedia Foundation, Inc., “Model-View-Presenter,” [Online]. Available: https://pt.wikipedia.org/wiki/Model-view-presenter. [Acedido em 26 Julho 2018].

[29] Wikimedia Foundation, Inc., “Model–View–ViewModel,” [Online]. Available: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel. [Acedido em 26 Julho 2017].

[30] D. Flanagan, JavaScript: The Definitive Guide, O'Reilly Media, Inc., 2006.

[31] Wikimedia Foundation, Inc., “JavaScript,” [Online]. Available: https://pt.wikipedia.org/wiki/JavaScript. [Acedido em 20 Novembro 2018].

[32] Wikimedia Foundation, Inc., “Client-Side,” [Online]. Available: https://en.wikipedia.org/wiki/Client-side. [Acedido em 20 Novembro 2018].

[33] E. A. Meyer, Cascading Style Sheets: The Definitive Guide, O'Reilly Media, Inc, 2004.

[34] Wikimedia Foundation, Inc., “Cascading Style Sheets,” [Online]. Available: https://pt.wikipedia.org/wiki/Cascading_Style_Sheets. [Acedido em 21 Outubro 2017].

[35] Wikimedia Foundation, Inc., “Expressão regular,” [Online]. Available: https://pt.wikipedia.org/wiki/Express%C3%A3o_regular. [Acedido em 13 Abril 2018].

[36] “Bootstrap 3 Datetimepicker,” [Online]. Available: https://eonasdan.github.io/bootstrap-datetimepicker/. [Acedido em 10 Abril 2018].

[37] “Google Charts,” [Online]. Available: https://developers.google.com/chart/. [Acedido em 10 Abril 2018].

[38] Wikimedia Foundation, Inc., “Ajax (programming),” [Online]. Available: https://en.wikipedia.org/wiki/Ajax_(programming). [Acedido em 17 Dezembro 2017].

[39] Refsnes Data, “HTML Responsive Web Design,” [Online]. Available: https://www.w3schools.com/html/html_responsive.asp. [Acedido em 21 Outubro 2017].

[40] M. Otto e ". , “Bootstrap,” Core team, [Online]. Available: https://getbootstrap.com/. [Acedido em 15 Agosto 2017].

[41] Wikimedia Foundation, Inc., “Bootstrap (framework front-end),” [Online]. Available: https://pt.wikipedia.org/wiki/Bootstrap_(framework_front-end). [Acedido em 21 Outubro 2017].

[42] Summernote Team, “Summernote,” [Online]. Available: https://summernote.org/. [Acedido em 10 Abril 2018].

[43] N. Asuni , “TCPDF - Open Source PHP class for generating PDF documents,” 2004. [Online]. Available: https://tcpdf.org/. [Acedido em 18 Abril 2018].

[44] S. S. Pore e S. B. Pawar, “Comparative Study of SQL & NoSQL Databases,” *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET),* pp. 1747-1753, 5 Maio 2015.

[45] W. Vogels, “Eventually Consistent,” pp. 40-44, 2009.

[46] P. Bailis e A. Ghodsi, “Eventual Consistency Today: Limitations, Extensions, and Beyond,” pp. 1-13, 2013.

[47] IBM, “ACID properties of transactions,” [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SSGMCP_5.4.0/product-overview/acid.html. [Acedido em 08 Janeiro 2018].

[48] S. Yu, “ACID Properties in Distributed Databases,” *Advanced eBusiness Transactions for B2B-Collaborations,* 2009.

[49] IBM, “Sistemas de Recomendação,” [Online]. Available: https://www.ibm.com/developerworks/br/local/data/sistemas_recomendacao/index.html. [Acedido em 02 Maio 2018].

[50] Y. S. Marko Balabanovic, “Content-Based, Collaborative Recommendation,” *COMMUNICATIONS OF THE ACM,* pp. 66-72, Março 1997.

[51] C. Basu, H. Hirsh e W. Cohen, "Recommendation as Classification: Using Social and Content-Based Information in Recommendation.," *In Proceedings of the Fifteenth National Conference on Artificial Intelligence,* pp. 1-7, 1998.

[52] J. Wang, A. P. de Vries e M. J. T. Reinders, "Unifying User-based and Item-based Collaborative Filtering Approaches by Similarity Fusion," *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval,* 06 Agosto 2006.

[53] P. J. S. Cardoso, P. Guerreiro, J. A. Pereira e R. J. Veiga, "A Route Planner Supported on Recommender Systems Suggestions.," *Enhancing Visits to Cultural Heritage Places,* pp. 1-8, 2018.

[54] P. J. S. Cardoso, J. M. F. Rodrigues, J. Pereira e J. D. P. Sardo, "An Object Visit Recommender Supported in Multiple Visitors and Museums," *International Conference on Universal Access in Human-Computer Interaction,* pp. 1-11, 2017.

[55] P. J. S. Cardoso, P. Guerreiro, J. Monteiro e J. M. Rodrigues, "Applying an Implicit Recommender System in the Preparation of Visits to Cultural Heritage Places," *International Conference on Universal Access in Human-Computer Interaction,* pp. 1-16, 2018.

[56] C. Pinela, "Recommender Systems– User-Based and Item-Based Collaborative Filtering"," 05 Novembro 2017. [Online]. Available: https://medium.com/@cfpinela/recommender-systems-user-based-and-item-based-collaborative-filtering-5d5f375a127f. [Acedido em 02 Maio 2018].

[57] B. Sarwar, G. Karypis, J. Konstan e J. Riedl, "Item-Based Collaborative Filtering Recommendation Algorithms," *Proceedings of the 10th international conference on World Wide Web,* pp. 285-295, 2001.

[58] G. Linden , B. Smith e J. York, "Amazon.com Recommendations: Item-to-Item Collaborative Filtering," em *Internet Computing*, IEEE Xplore, 2003, pp. 76-80.

[59] D. Li, C. Chen, Q. Lv, L. Shang, Y. Zhao, T. Lu e N. Gu, "An algorithm for efficient privacy-preserving item-based collaborative filtering," pp. 311-320, Fevereiro 2016.

[60] J. Benesty, J. Chen, Y. Huang e I. Cohen, “Noise Reduction in Speech Processing,” em *Noise Reduction in Speech Processing*, Springer Science & Business Media, 2009, pp. 37-38.

[61] B. Smith e G. Linden, “Two Decades of Recommender Systems at Amazon.com,” em *The Test of Time*, IEEE Internet Computing, 2017, pp. 12-18.

ANEXOS

A. ESTRUTURA DO MODELO DE DADOS

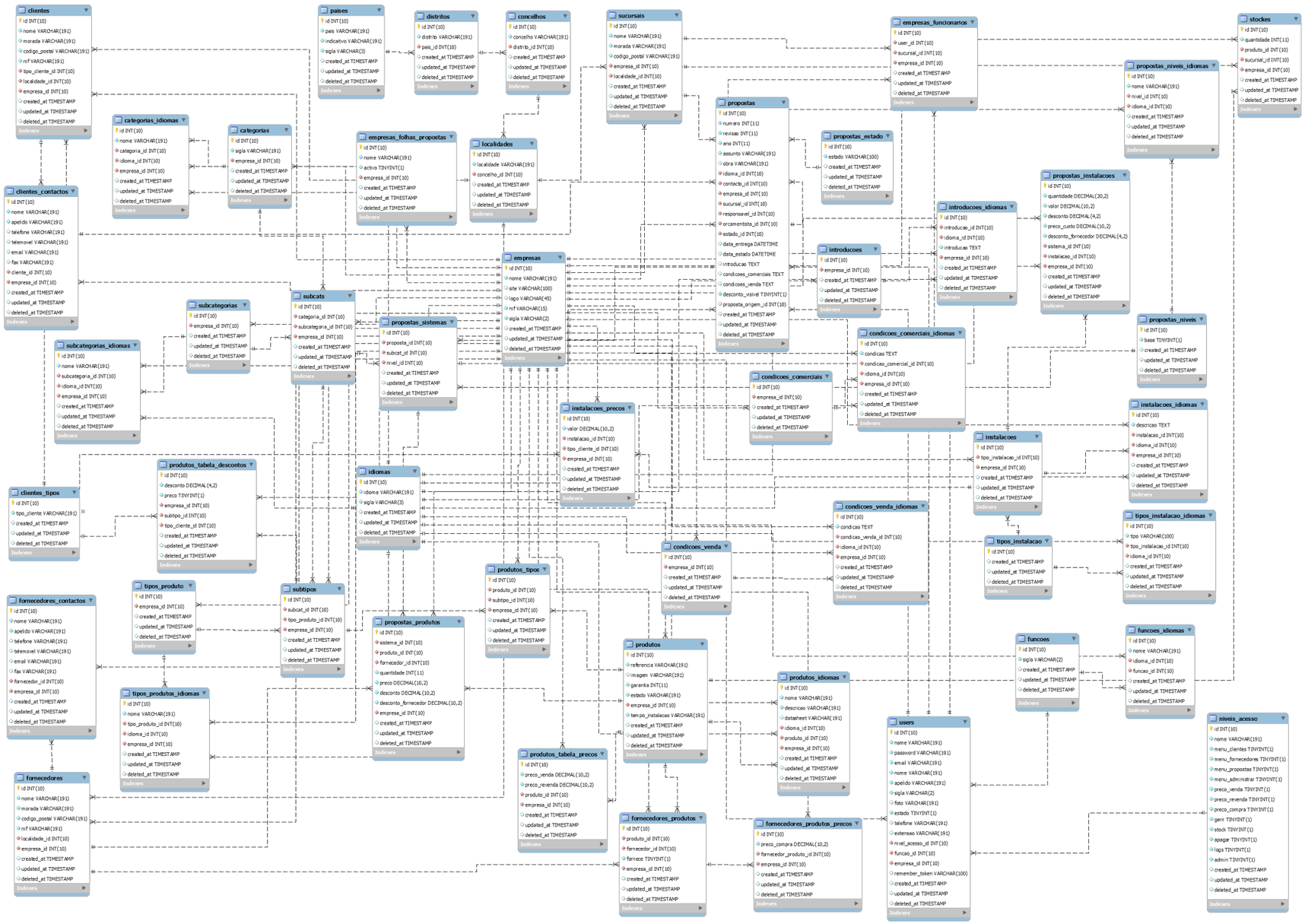


Figura A.1: Modelo de dados