Bárbara Corvelo Oliveira

Licenciada em Engenharia Informática

RAISE UP: Desenvolvimento de framework em plataforma Cloud para permitir a automatização de classificação

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática

Orientador: Fernando Gil Fallé Quartin d'Assunção, Licenciatura em Ensino da Matemática, WorldIT - Consulting Services

Coorientador: Artur Miguel de Andrade Vieira Dias, Professor Auxiliar, Universidade Nova de Lisboa

Júri

Presidente: Prof. Doutor Nuno Manuel Ribeiro Preguiça

Arguente: Prof. Doutor José Augusto Legatheaux Martins

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Março, 2021

RAISE UP: Desenvolvimento de framework em plataforma Cloud para permitir a automatização de classificação

Copyright © Bárbara Corvelo Oliveira, Faculdade de Ciências e Tecnologia, Universidade NOVA de Lisboa.
A Faculdade de Ciências e Tecnologia e a Universidade NOVA de Lisboa têm o direito, perpétuo e sem limites geográficos, de arquivar e publicar esta dissertação através de exemplares impressos reproduzidos em papel ou de forma digital, ou por qualquer outro meio conhecido ou que venha a ser inventado, e de a divulgar através de repositórios científicos e de admitir a sua cópia e distribuição com objetivos educacionais ou de investigação, não comerciais, desde que seja dado crédito ao autor e editor.

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "novathesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Dedico este trabalho aos meus pais, Roberto e Mª da Conceição, às minhas irmãs, Mafalda, Sara, Roberta e Mariana por todo o apoio e incentivo

Agradecimentos

Em primeiro, quero agradecer ao Fernando d'Assunção, o meu orientador na empresa, pela sua disponibilidade e acompanhamento exercidos durante o período de implementação desta dissertação.
De seguida, quero agradecer ao prof. Artur Miguel Dias, o meu coorientador, por toda a ajuda prestada e esclarecimentos feitos durante a realização desta etapa e da anterior.
Gostava de agradecer também à Marta Domingues Silva, colega na empresa, por todo o apoio e ensinamentos dados ao longo da realização da aplicação desta dissertação.
Agradeço, em geral, a toda a empresa pela maneira como me receberam e acompanharam, tornando a minha integração fácil e satisfatória.
Quero dar um agradecimento especial à minha família, realçando os meus pais, Roberto e Conceição, e as minhas irmãs, Mafalda, Sara, Roberta e Mariana, por todo o apoio, compreensão e incentivo dados ao longo de todo o meu percurso académico.
Por último, agradeço a todos os docentes e professores do Departamento de Informática e a todos os meus colegas que me acompanharam durante estes anos.
A todos, um muito obrigada!

Resumo

Uma das grandes dificuldades das empresas encontra-se em classificar os seus clientes, ou potenciais clientes (*leads*), de forma rápida e flexível, tendo como base, dados das suas experiências de negócio passadas ou novas informações recolhidas, de modo a oferecer propostas personalizadas.
A solução para este problema implicará a criação de uma nova aplicação dentro da plataforma *Salesforce* (plataforma de CRM que permite a gestão do contacto com os clientes). Essa aplicação efetuará uma análise aprimorada dos dados contidos no sistema e, posteriormente, classificará os clientes em diferentes categorias. Estas categorias serão determinadas a partir da adição de um conjunto de regras para cada critério, apoiadas em metadados, com o intuito de selecionar os clientes que se enquadram nestes.
Com a criação desta solução, as empresas ficarão mais informadas e preparadas para garantir melhores negócios, quer nos seus clientes atuais, quer na pesquisa de novos clientes, de modo a maximizar o retorno da sua atividade.

Palavras-chave: Salesforce, clientes, *leads*, critérios, regras, classificação

Abstract

One of the major difficulties of companies is classifying their customers, or potentials customers (*leads*), in a faster and flexible way, considering data for their previous business experiences or new informations, in a way that can offer personalized propositions.
The solution to this problem will imply the creation of the new app inside *Salesforce* platform (CRM platform that allows the management of customers contact).
This app will do a sofisticated analysis of the data contained in the system and, posteriorly, it will classify the customers in the different categories. This categories will be determined by the adition of a set of rules for each criteria, supported by metadata, with the purpose of selecting the customers that will fit in these.
With this solution, the companies will be more informed and prepared to ensure the best deals, either in your current customers, or in the search for new customers, in order to maximize the return of your activity.

Keywords: Salesforce, customers, leads, criteria, rules, classification

Índice

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Neste capítulo é exposto, de forma sumária, o trabalho proposto, definindo o contexto em que está integrado, a motivação e a solução mais adequada.

1.1 Contexto e Descrição

Antes de entrar na descrição desta dissertação, gostava de começar por descrever o que significa CRM, pois esta sigla relata o grande objetivo da plataforma usada e um dos objetivos da aplicação criada nesta dissertação. CRM é a sigla dada à gestão do relacionamento entre a empresa e os seus clientes. Este sistema permite às empresas criar e manter mais rápida e facilmente uma ligação com os clientes, de forma a acompanhar e tratar cada um de maneira personalizada [3]. (Definição mais detalhada na secção 2.1.1.1)
As empresas, hoje em dia, que têm grandes preocupações de crescimento, nomeadamente no aumento e gestão dos seus negócios e mantimento dos seus clientes, possuem, entre os seus instrumentos de trabalho, uma ferramenta de CRM.
O *Salesforce* é a plataforma de CRM mais conhecida e utilizada no mundo, pois permite tudo o que este sistema promete e encontra-se constantemente a inovar de forma a antecipar possíveis problemas dos clientes em relação à sua utilização [2]. Através das suas tecnologias, dá a possibilidade de ter as informações do cliente atualizadas, com acesso para todos os colaboradores, possibilitando o contacto destes com o cliente de maneira eficiente e rápida. Isto torna o serviço mais qualificado, consequentemente, leva a uma maior satisfação do cliente, garantindo, assim, a sua preservação e um aumento nos negócios efetuados.
Este *software* pode ser considerado um repositório de dados, ou seja, é possível guardar na nuvem, a sua base de dados única, as informações que são inseridas ou exportadas de um ficheiro como de uma base de dados externa.

Esta dissertação tem como propósito criar uma nova aplicação dentro do *Salesforce* que, a partir dos dados contidos, seja capaz de executar as regras definidas para cada critério, baseadas em metadados, e classificar os clientes selecionados.
O uso do *Salesforce* neste projeto foi imposto pela *WorldIT* sendo uma escolha natural, uma vez que é um dos CRM utilizados por esta empresa, que permite a inserção de aplicações criadas por empresas para o benefício de outras. Contudo, a razão mais importante é a possível complementação da aplicação elaborada nesta dissertação com uma já criada por eles.

1.2 Motivação

Com a classificação dos clientes e potenciais clientes, o objetivo é de obter uma métrica que pode ser utilizada para decisão de diversos fins, contribuindo para a determinação do sucesso do negócio. Ou seja, sem o resultado de um certo critério, a empresa não saberá se o cliente se integra nos seus padrões de negócio.
Esta prática permite realçar alguns aspetos positivos que melhorarão a interação com o cliente, sendo estes:

- **Personalização do negócio**
  A partir das classificações obtidas para cada critério, é possível perceber se o cliente adequa-se aos parâmetros considerados mais importantes para o negócio em questão. Se sim, a proposta elaborada será pensada e executada de acordo com os critérios e os dados do cliente.

- **Maior probabilidade de sucesso**
  Quando se decide quais os campos mais importantes para a definição de critérios, considera-se os parâmetros que serão mais adequados ao negócio a ser realizado. Se o resultado obtido for o ideal, a probabilidade do negócio ser bem sucedido, é elevada.

- **Assertividade na tomada de decisões**
  Tendo em consideração o que foi mencionado anteriormente, ou seja, ao escolher os campos mais relevantes, acredita-se que o cliente que obtém o maior valor na classificação, é o ideal para a realização do negócio.

1.3 Solução Proposta

O principal objetivo desta dissertação consiste na elaboração de uma aplicação que permita aplicar padrões e normas de negócio aos dados de cada cliente, para produzir resultados que sirvam de base a uma classificação do cliente. Esta será utilizada, tanto como parâmetro no registo do cliente, como base para executar propostas comerciais.
De forma a conseguir os parâmetros necessários para realização das regras, acede-se aos

dados contidos no repositório do cliente.
Esses dados contidos podem ser obtidos apenas dos registos das interações com os clientes ou ainda, dos dados obtidos da *Informa*, através da integração da aplicação *"INFORMA - Business by data"* pertencente à empresa (explicada mais adiante neste documento 2.1.2). Sendo do interesse de muitas empresas a finalidade da aplicação criada, esta será disponibilizada na loja do *Salesforce* onde poderá ser utilizada em inúmeros ambientes (tanto geográficos como de negócio).

1.4 Estrutura do Documento

Neste documento pode-se encontrar 6 capítulos.

1. **Introdução**

2. ***RAISE UP***
   É desenvolvida a descrição do conceito de CRM a partir da explicação do *software Salesforce*, ditando as suas vantagens e real integração com o problema. Além de tudo isso, é apresentada a empresa empregadora e como esta dissertação se encaixa nas suas áreas de desenvolvimento.

3. **Tecnologias do *Salesforce***
   É feita uma descrição mais profunda das tecnologias e efetuado um estudo de comparação dessas com outras nossas conhecidas (por exemplo: *Java*, *HTML*, *JavaScript e SQL*).

4. **Proposta de Solução**
   Demonstram-se as duas possíveis arquiteturas presentes por detrás da solução, as funcionalidades representantes e uma breve explicação de cada e o esclarecimento de como os testes e validações necessários para o bom desempenho da aplicação no final são conseguidos.

5. **Funcionalidades**
   Explicação e alguma discussão da forma como as diversas funcionalidades existentes na aplicação foram elaboradas, mencionando os problemas obtidos e as decisões tomadas.

6. **Avaliação**
   É feita uma análise à aplicação criada, tanto no tempo de arranque de cada página modificada como uma bateria de testes feita de modo a verificar a fiabilidade do código efetuado.

Capítulo 2

RAISE UP

Este capítulo serve para explicar o conteúdo da aplicação desta dissertação e integrá-lo no contexto de CRM. Neste sentido, explica-se em que consiste o *Salesforce*, a aplicação *"INFORMA - Business by data"* e o enquadramento com a empresa.

2.1 Definição

"RAISE UP" foi o nome dado à aplicação desenvolvida nesta dissertação. Esta foi implementada em ambiente *Salesforce*, um *software* de CRM (conceito abordado em profundidade na secção 2.1.1).
A aplicação pretende, através das informações contidas no sistema e com os dados fornecidos pelos relatórios da *Informa*, criar regras personalizadas para vários critérios distintos, apoiadas em metadados, de modo a obter uma classificação. Esta classificação será utilizada para várias finalidades, tal como propostas comerciais e sugestão de novos clientes a empresas parceiras.
Esta aplicação foi submetida na loja de aplicações do *Salesforce* (*AppExchange*), para possível utilização de outras empresas interessadas na sua finalidade.

2.1.1 *Salesforce*

Salesforce é o CRM (Gestão de Relacionamento com o Cliente) mais conhecido e usado no mundo. Este permite criar uma ligação entre a empresa e os seus clientes mais personalizada [14], contribuindo para um crescimento na economia da empresa e uma diminuição no risco de negócio.
Um serviço oferecido pelo *software* é a *AppExchange* que dá a possibilidade ao utilizador de estender as funcionalidades do seu ambiente com outras aplicações já criadas e disponíveis para uso.

Para um melhor entendimento da real função deste *software*, começa-se por definir do que realmente trata um CRM 2.1.1.1.

2.1.1.1 CRM

O CRM, Gestão de Relacionamento com o Cliente, que vem do inglês *Customer Relationship Management*, tem como principal objetivo aperfeiçoar o relacionamento entre a empresa e o cliente, percebendo e antecipando as necessidades deste. O CRM ajuda diretamente nas áreas de vendas, suporte, estratégias de *marketing*, entre muitas outras, de forma a executar o seu objetivo com sucesso. Resumindo, o sistema permite registar clientes e adicionar diversas informações, de modo a garantir uma relação entre a empresa e o cliente mais satisfatória.[3]
Os *softwares* de CRM agregam os dados e documentos referentes a cada cliente numa só base de dados, de modo a simplificar a sua obtenção e gestão para os vendedores. Estes permitem também registar todos os contactos com o cliente, sejam estes por *e-mail*, telefone, redes sociais ou outros meios, fazendo com que alguns processos de fluxo de trabalho (tarefas, calendários e alertas) sejam automatizados, possibilitando que os responsáveis sigam o desempenho e a produtividade a partir dos registos efetuados no sistema [10].

2.1.1.2 Definição

O *software Salesforce* funciona na nuvem, o que permite aos utilizadores aderirem às informações a qualquer hora e em qualquer dispositivo [10], não sendo necessários especialistas em TI (Tecnologias de Informação) para qualquer configuração ou gestão [5].
Este *software* possibilita registar as interações com os clientes, permitindo identificar as suas necessidades e estabelecer aplicações com foco nestas, tornando, assim, a relação com os clientes mais próxima e duradoura [5].
Podemos ainda verificar que o *Salesforce* oferece uma vasta gama de produtos, apresentados na seguinte secção 2.1.1.3, que permite que esse *software* seja tão bem sucedido no mercado [14]. Além destes produtos, esta plataforma também está bem equipada a nível de tecnologias (secção 2.1.1.4) e suporte à base de dados (secção 2.1.1.5).

2.1.1.3 Produtos Oferecidos

Os produtos disponibilizados estão contidos em áreas distintas, sendo que todos estão ligados à nuvem [14]. Na seguinte lista encontram-se alguns dos produtos que são mais relevantes para a solução desta dissertação.

- ***Sales Cloud***
  "O Sales Cloud é o CRM de vendas da Salesforce." [16] Este CRM permite automação das tarefas, agregação e sustentação da gestão das áreas de vendas e marketing e atendimento ao cliente. [16]

- ***Service Cloud***
  O *Service Cloud* permite às empresas a realização de um atendimento personalizado a qualquer momento, em qualquer meio, de modo a resolver problemas mais rapidamente. Isto leva à satisfação do cliente, e, consequentemente, a efetuar novamente negócios com a empresa. [17]

- ***Commerce Cloud***
  O *Salesforce* contém dois tipos de plataforma *e-commerce*: *B2B (Business to Business)* e *B2C (Business to Consumer)*. A solução para negócios virados para o consumidor contém o que é necessário para experienciar compras inovadoras e personalizadas. No entanto, a plataforma B2B apresenta uma solução mais vigorosa de modo a suportar os requisitos dos compradores. [8]

- ***Customer 360 Platform***
  O *Customer 360 Platform* permite aos utilizadores conceber aplicações e negócios mais depressa. [14]

- ***Einstein Analytics***
  Com a análise de Einstein é possível examinar todas as áreas e colaboradores de diversas maneiras: através da observação dos dados obtidos por outras fontes ou mesmo do *Salesforce*, *dashboards* incorporados ou modelos próprios ou por meio dos dados de IA (Inteligência Artificial) aumentada. [12]

Os produtos mencionados acima estão ligados de alguma maneira ao trabalho desenvolvido. *Sales Cloud*, porque o principal objetivo da aplicação é melhorar os negócios das empresas ao encontrar a classificação certa para cada cliente de acordo com os seus parâmetros. *Service Cloud*, pois, ao encontrar a classificação do cliente podemos fazer ofertas personalizadas, dependendo do resultado obtido. *Commerce Cloud*, visto que a aplicação se destina a qualquer tipo de cliente, organizações ou consumidores. Sendo que o grande propósito desta dissertação é a criação de uma aplicação, o *Customer 360 Platform* é um dos produtos que vai ajudar a que a aplicação seja concretizada mais facilmente. *Einstein Analytics*, uma vez que, de acordo com os resultados obtidos no processo de classificação, a empresa pode querer efetuar relatórios ou *dashboards* de maneira a analisar e obter estatísticas de diferentes informações.

2.1.1.4 Tecnologias

O *Salesforce*, sendo uma plataforma de personalização de negócio, tanto a nível *back-end* como *front-end*, usa tecnologias de programação e de *"clicar e arrastar"* (desenvolvimento declarativo).
Como desenvolvimento programático do *software* encontramos os seguintes itens:

- **Apex**
  É uma linguagem de programação com sintaxe semelhante ao *Java* e permite um

acesso direto à base de dados. Esta dá a possibilidade aos programadores de estabelecer lógica de negócios a ocorrências da aplicação (páginas de *Visualforce*, cliques em botões e progresso de registos referentes). Para melhor conhecimento, na secção 3.1.1 haverá uma descrição mais pormenorizada desta linguagem.

- ***Visualforce***
  É uma estrutura de elaboração da web, que dá a oportunidade aos programadores de desenvolver interfaces requintadas e distintas, que têm a possibilidade de ser integradas na plataforma *Lightning*.
  Esta concede aos programadores a possibilidade de alarguem os recursos internos do *Salesforce*, dando lugar a funcionalidades e elaboração de novas aplicações. [13]
  Nas figuras 2.1 e 2.2 conseguimos observar a implementação de código para páginas do *Visualforce* e a sua demonstração, respetivamente.

StationCheck.vfp

Preview API Version: 47

```
1  <apex:page sidebar="false">
2  <!--Flight Systems Checklist Visualforce Page-->
3     <h1>Station Status</h1>
4     <apex:form id="stationReadinessChecklist">
5        <apex:pageBlock title="Station Readiness Checklist">
6           <!--Second Section-->
7           <apex:pageBlockSection title="Fuel Tanks">
8              <apex:inputCheckbox immediate="true"/>Tank 1
9              <apex:inputCheckbox immediate="true"/>Tank 2
10             <apex:inputCheckbox immediate="true"/>Tank 3
11             <apex:inputCheckbox immediate="true"/>Tank 4
12             <apex:inputCheckbox immediate="true"/>Tank 5
13             <apex:inputCheckbox immediate="true"/>Tank 6
14          </apex:pageBlockSection>
15          <apex:pageBlockButtons >
16             <!--Adding Save Button-->
17             <apex:commandButton value="Save" action="{!save}"/>
18          </apex:pageBlockButtons>
19       </apex:pageBlock>
20    </apex:form>
21 </apex:page>
```

Figura 2.1: Exemplo de um código implementado para uma página do *Visualforce*

- ***Lightning***
  "(...) é a plataforma front-end do Salesforce" [5]. Através dessa plataforma é possível a execução, destinada a qualquer dispositivo, de aplicações responsivas.
  O *Lightning* contém as seguintes tecnologias: [22]

  – *Componentes do Lightning*
    Esses permitem uma maior execução na implementação e no comportamento da aplicação.

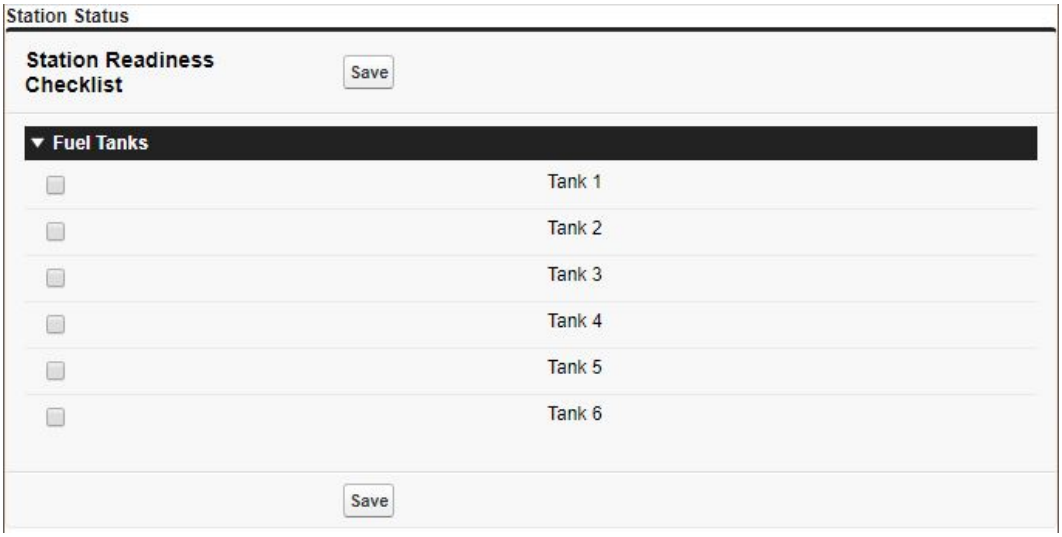


Figura 2.2: Exemplo da execução da página do *Visualforce*

- *Lightning App Builder*
  Possibilita aos responsáveis a implementação de páginas do *Lightning*, sem escrever código, recorrendo aos componentes referidos anteriormente.
- *Experience Builder*
  Concede aos desenvolvedores a geração de comunidades, sem escrever código, servindo-se dos componentes e modelos do *Lightning*.

A partir da figura 2.3 conseguimos obter um exemplo de componente Aura do *Lightning*, na figura 2.4 temos o controlador do *Lightning* e na figura 2.5 observamos a execução desta elaboração.

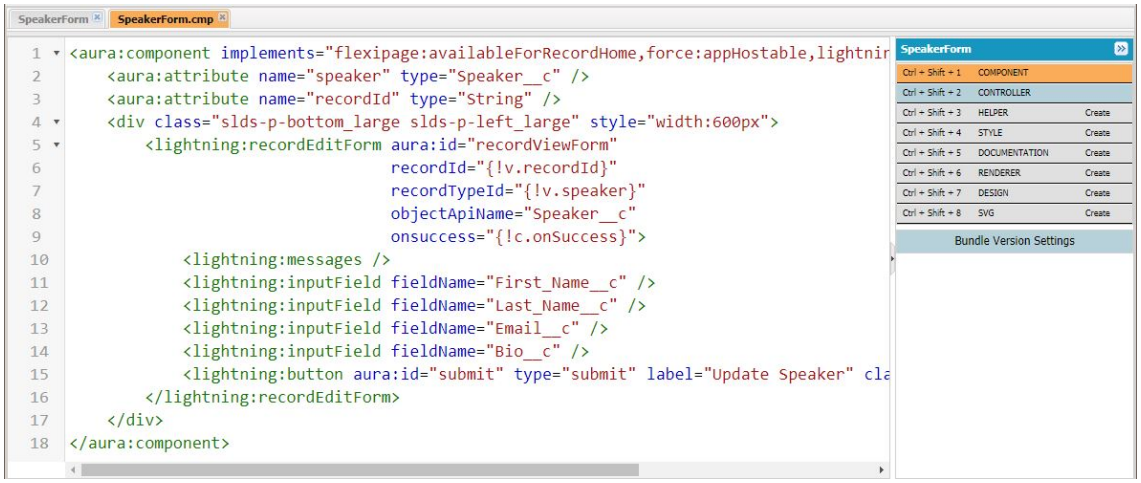


Figura 2.3: Exemplo de código de um componente do *Lightning*

Relativamente às tecnologias para desenvolvimento declarativo, encontramos o *Lightning*

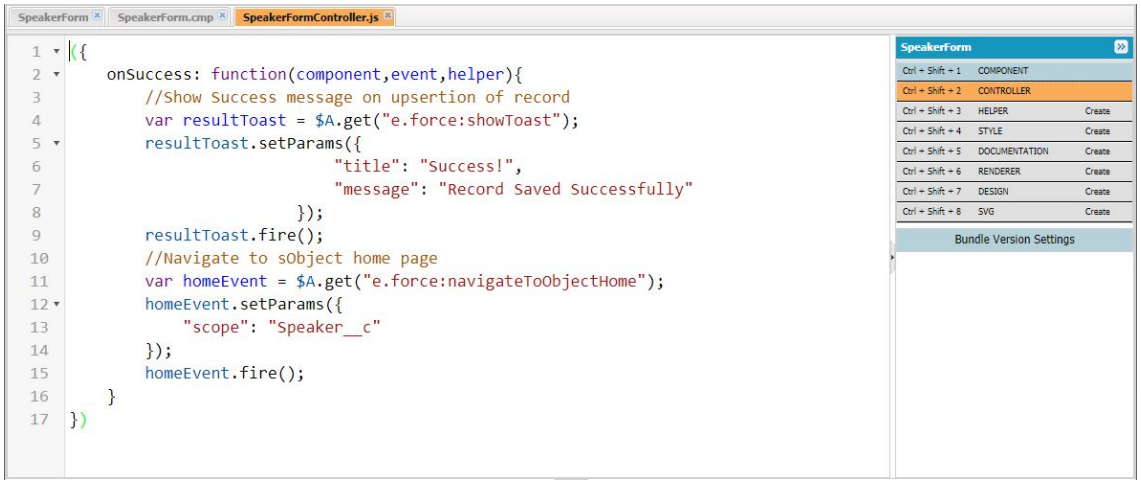


Figura 2.4: Exemplo de código de um *controller* do *Lightning*

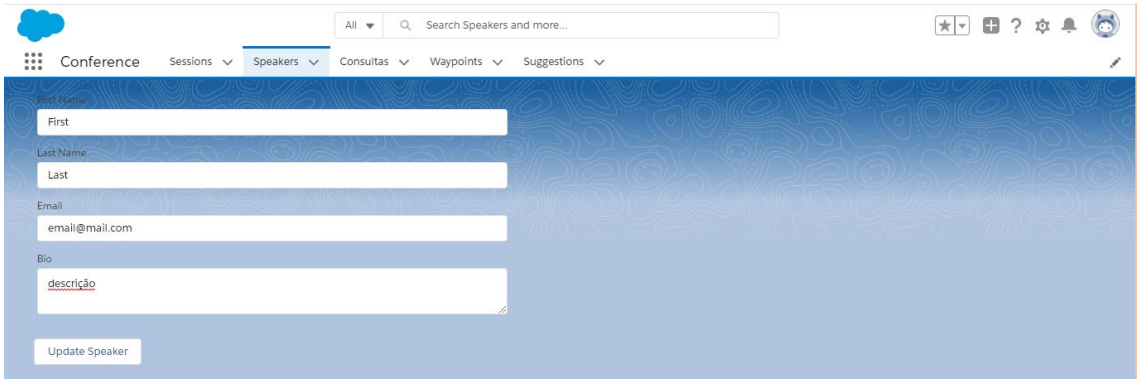


Figura 2.5: Exemplo da execução do *Lightning*

Flow. Esta tecnologia permite fazer automação declarativa de processos para cada aplicação, portal e experiência *Salesforce* [7]. Assim, para executar esta automação, este fornece duas ferramentas:

- *Process Builder*
  Como o nome indica, esta ferramenta é utilizada para fazer processos. Esta dá a possibilidade de automatizar processos comerciais com facilidade e, ao longo da sua criação, conseguimos observar uma representação gráfica. (Figura 2.6)

- *Flow Builder*
  Tal como anteriormente, nesta, o nome indica que é utilizada para fazer fluxos, permitindo automatizar um processo comercial obtendo dados. (Figura 2.7)

Nesta dissertação foi necessário utilizar algumas das tecnologias mencionadas anteriormente na realização das funcionalidades mencionadas na secção 4.3. Nesta secção ficamos com as ideias mais precisas das funcionalidades, alterações e adições feitas, contudo, é no capítulo 5 que é explorado esse assunto (tecnologias utilizadas e alterações efetuadas).

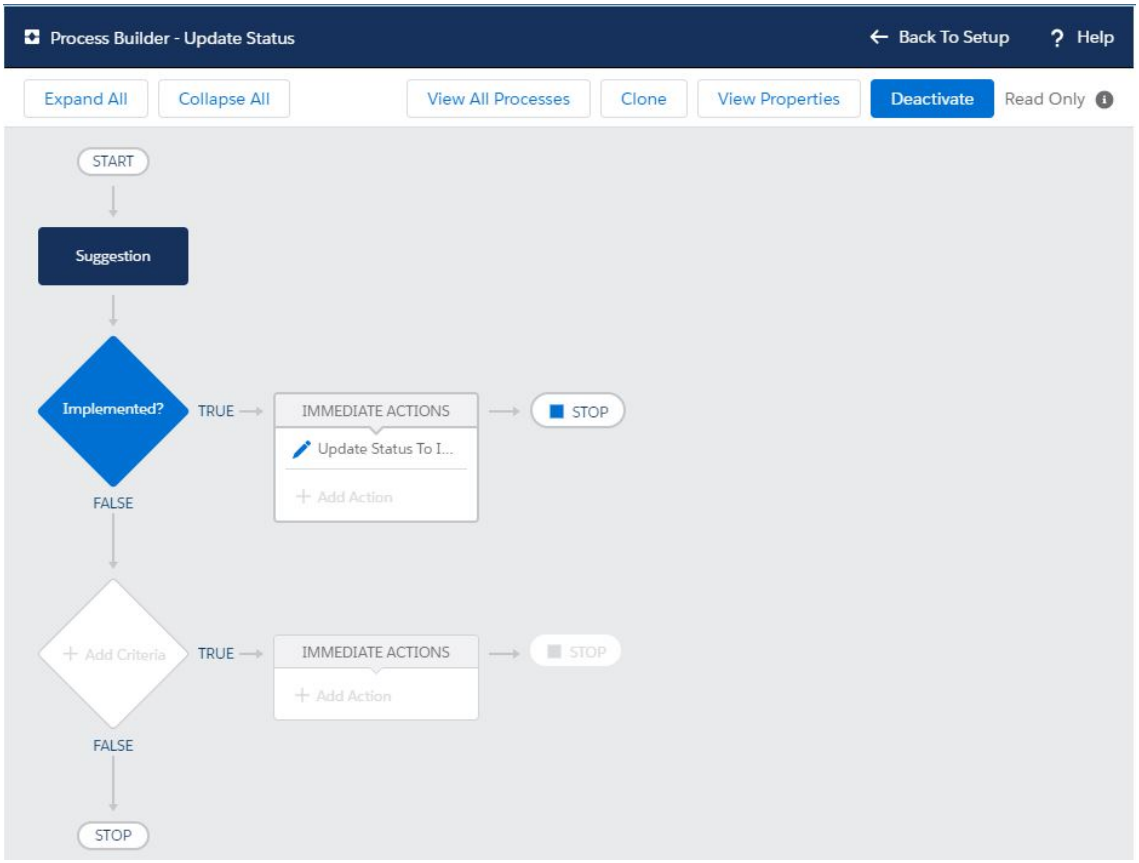


Figura 2.6: Exemplo da página de execução de um *Process Builder*

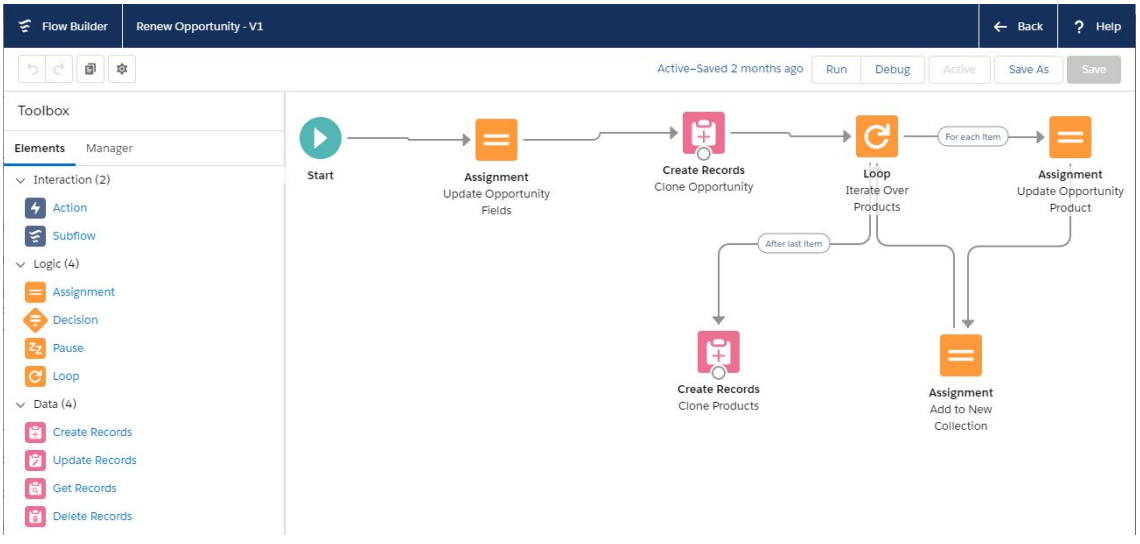


Figura 2.7: Exemplo da página de execução de um *Flow Builder*

2.1.1.5 Objetos

A plataforma *Salesforce* dá a possibilidade de se usar objetos padrão (existentes) e objetos personalizados (novos). A possibilidade de criar novos objetos serve para a plataforma ficar mais ajustada ao objetivo da empresa [18].

Os objetos já existentes são os considerados mais importantes para uma empresa enriquecer o seu negócio, tais como: Contas, Contactos, *Leads*, Oportunidades, etc. [18]. Os utilizados foram Conta e *Lead*, uma vez que, para uma empresa poder classificar tem de ser um cliente registado como Conta ou ser um potencial cliente registado como *Lead*, para, posteriormente, ser convertido para Conta ou Contacto [9].
Em relação aos objetos personalizados, estes são criados de maneira a completar a aplicação. Como existem vários tipos de sectores de empresas a usar o *Salesforce*, cada uma necessita de ajustar a aplicação às suas necessidades. De modo a resolver esta situação, o *Salesforce* permite a criação de objetos ajustados ao utilizador [18]. Para esta dissertação, foi necessário construir alguns desses objetos que serão posteriormente mencionados no capítulo 5.

2.1.2 *INFORMA - Business by data*

A empresa *WorldIT - Consulting Services* contém uma versão que pode ser integrada diretamente à aplicação implementada nesta dissertação chamada, atualmente, *"INFORMA - Business by data"*, antiga *"RAISE"*. Esta solução permite completar as informações dos clientes existentes no sistema através da obtenção dos dados fornecidos dos relatórios da *Informa* usando os seus identificadores fiscais. Para alcance destes dados, esta aplicação está ligada à *INFORMADB*, base de dados da *Informa* que contém os relatórios. Deste modo, para satisfazer estas características, a aplicação tem as seguintes funcionalidades: [28]

- Obter relatórios da *Informa*;
- Receber alertas quando há uma atualização nos relatórios;
- Gerar relatórios e *dashboards* a partir dos relatórios;
- Incorporar o documento no sistema;
- Configurar a lista de atributos a consultar;
- Registar todas as operações efetuadas.

Para melhor perceção da integração e funcionamento desta, pode-se encontrá-la na arquitetura do sistema, na secção 4.2, juntamente com a explicação destes aspetos.

2.2 Abordagem da empresa

Esta dissertação foi realizada em cooperação com a empresa *WorldIT - Consulting Services*, sendo que esta disponibilizou espaço e material de trabalho, troca de conhecimento e apoio financeiro, com a expetativa da disponibilização desta aplicação com o exterior, o que se tornou num facto.

2.2.1 *WorldIT - Consulting Services*

A *WorldIT* é uma empresa de consultoria que trabalha em diversas áreas, tais como: Portais, *Business Intelligence*, Soluções CX, CRM, *Mobile*, entre outras [26].
Esta empresa tem a preocupação de fornecer formações constantes aos seus colaboradores e criar parcerias com plataformas/empresas de acordo com as áreas de desenvolvimento [25].

2.2.2 Enquadramento

Como foi mencionado na secção acima, 2.2.1, uma das áreas de trabalho é o CRM. Esta área divide-se entre *Salesforce* e *Siebel*, sendo que o *Salesforce* está em forte crescimento, ao nível do desenvolvimento e angariação de novos projetos na empresa.
Como já referi anteriormente, há uma aplicação criada pela empresa que pode ser integrada com a que foi desenvolvida nesta dissertação, a *"INFORMA - Business by data"*. A *"RAISE UP"* é um complemento a esta aplicação, sendo que podem ou não ser utilizadas em conjunto. Se ambas estiverem integradas no *Salesforce*, haverá um maior conjunto de dados a ser trabalhado e terá funcionalidades extras para o utilizador.

Capítulo 3

Tecnologias do *Salesforce*

Neste capítulo é feita uma descrição mais pormenorizada e uma comparação das ferramentas que estão integradas no *Salesforce* com as já existentes e conhecidas no mercado. Como foi mencionado no capitulo 2, o *Salesforce* integra dois tipos de tecnologias: programáticas e declarativas.

3.1 Programática

Estas tecnologias são mencionadas assim porque é necessário haver código na sua implementação.

3.1.1 Apex

O *Apex* é uma linguagem de programação orientada a objetos, baseada no *Java*, e, como tal, tem uma sintaxe semelhante. Esta linguagem permite aos utilizadores a criação de acionadores em objetos, a execução de serviços da *web*, a realização de ações nos eventos do sistema (cliques em botões, atualizações de registos, etc.), o acesso direto à base de dados, entre outros [21].
Em *Java*, para se poder aceder à base de dados, é necessário criar um conector. Em *Salesforce*, de forma a ser possível ter acesso direto à base de dados no *Apex*, existem duas linguagens semelhantes ao *SQL (Structured Query Language)*:

- *SOQL (Salesforce Object Query Language)*: usada para consultas à base de dados obtendo todos os registos pertencentes às condições da pesquisa [23]
- *SOSL (Salesforce Object Search Language)*: usada para pesquisas à base de dados explorando texto nos registos, quando não se conhece quais os campos e objetos onde se deve procurar [24]

O *SQL (Structured Query Language)* é uma linguagem de programação de conexão à base de dados.
Na figura 3.1 é possível observar a programação feita numa classe *Apex* com acesso à base de dados usando a linguagem *SOQL*.

DailyLeadProcessor.apxc

Code Coverage: None API Version: 47 Go To

```
1  global class DailyLeadProcessor implements Schedulable{
2      global void execute(SchedulableContext sc){
3          List<Lead> lstOfLead = [SELECT Id FROM Lead WHERE LeadSource = null LIMIT 200];
4
5          List<Lead> lstOfUpdatedLead = new List<Lead>();
6          if(!lstOfLead.isEmpty()){
7              for(Lead ld : lstOfLead){
8                  ld.LeadSource = 'Dreamforce';
9                  lstOfUpdatedLead.add(ld);
10             }
11
12             UPDATE lstOfUpdatedLead;
13         }
14     }
15 }
```

Figura 3.1: Exemplo de código *Apex* com acesso à base de dados utilizando a linguagem *SOQL*

O *Apex* dá a possibilidade de criar e executar testes de unidade de forma a avaliar o código implementado. Estes testes, quando corridos, permitem visualizar a porção do código coberto com os testes efetuados [21]. Esta cobertura, para se considerar que os testes são suficientes para avaliar o código como eficaz, é necessário ser acima dos 75%, também para se introduzir em ambiente de produção [20].

3.1.1.1 Comparação do *Apex* com *Java*

Como foi mencionado, o *Apex* foi criado baseado na linguagem de programação *Java*. Desta maneira, iremos ver algumas das suas semelhanças e diferenças.

- **Semelhanças**:
  - Sintaxe
  - Bibliotecas de estruturas de dados, exceto, por exemplo, *Queues*
- **Diferenças:**
  - O *Apex* é uma plataforma como serviço e o *Java* é um *software* como serviço. O que diferencia essas duas definições é o facto de, no *Java* ser necessário uma instalação e esta precisar de *hardware*, banco de dados, *middleware*, servidores e

frameworks, enquanto que o *Apex*, como está alojado numa plataforma situada na *internet*, divide um servidor remoto e a base de dados, estando dependente da velocidade da *internet* [1].

- As variáveis e métodos sem modificador de acesso no *Apex* são privados por defeito, enquanto que no *Java* são públicos [11].
- Se a variável ou método no *Apex* estiver como público, pode ser acedido por qualquer outra classe, enquanto que no *java* só podem aceder classes do mesmo pacote [11].
- Em *Java*, para comparar texto é necessário usar o ".equals", no *Apex* pode-se fazer igual ou como se compara números "==".

3.1.1.2 Comparação do *SOQL* com *SQL*

As duas linguagens são de acesso à base de dados, contudo existem umas diferenças que devem ser mencionadas [1]:

- A *SOQL* não permite pesquisar todos os campos tal como o *SQL* (*Select * from object*). No *SOQL* é necessário escrever todos os campos que queremos obter do objeto.
- A *SOQL* não permite normas de inserção, atualização e remoção da base de dados. Essas são instruídas por métodos *DML (Data Manipulation Language)*.

3.1.2 Visualforce

O *Visualforce* é um documento com o formato semelhante ao *HTML*, na medida em que a sua estrutura é feita de marcas.
Neste documento definimos a estrutura da página, o seu conteúdo e o seu estilo. As operações executadas na página, tal como, obtenção e inserção de dados na base de dados, são tratadas por uma classe *Apex*, chamada controlador ou extensor, dependendo se tem controlador padrão definido por um objeto.
O *Visualforce* é utilizado para customizar interfaces que podem ser incluídas na plataforma ou como assistente a novos ou renovados mecanismos do sistema, por exemplo, botões. Este é enriquecido por uma enorme quantidade de componentes, correspondente a cada marca, que ajuda na eficácia e rapidez da sua criação [19].
Esta estrutura é centrada na página, pois a sua criação prevê a substituição total da página existente na plataforma [6].

3.1.3 Lightning

O *Lightning*, como foi referido na secção 2.1.1.4, é constituído por diversas tecnologias. Aqui, iremos mencionar apenas os componentes.
Existem dois componentes do *Lightning*: componentes do Aura e componentes da *Web* do

Lightning. Os componentes da *Web* do *Lightning* (LWC) foram criados a partir dos componentes do *Aura*, de forma a melhorar a sua execução [27]. Esses componentes podem ser usados juntos na mesma página.
Quando criados, o seu projeto contém uma classe *HTML* e uma *JavaScript*. Tal como no *Visualforce*, os componentes também têm um vasto conjunto de marcas. Esses componentes têm uma forma de adaptar o desenho da página a qualquer tipo de dispositivo (móvel ou *desktop*), além de dar apoio às mais recentes tecnologias dos navegadores [15]. Os componentes usam o *JavaScript* para o cliente e o *Apex* para o servidor, de modo a haver troca de dados. Estes componentes são centrados na aplicação, pois a sua aplicação não modifica totalmente a página já existente [6].

3.1.4 *Visualforce* ou Componentes do *Lightning*

Vamos discutir rápido as diferenças entre o *Visualforce* e os componentes do *Lightning* (CL) [6].

1. O *Visualforce* tem uma interação direta com o servidor enquanto que nos CL têm o *javaScript* que liga ao servidor;

2. O *Visualforce* é mais fácil e rápido de aprender a sua implementação e os CL levam um pouco mais de tempo;

3. Nos CL criam-se um complemento a uma página, no *Visualforce* cria-se a página;

4. O *Visualforce* possibilita a renderização de páginas como ficheiros PDF enquanto que os CL não suportam.

3.2 Declarativa

As tecnologias declarativas são todas aquelas que não necessitam de código, mas que a sua implementação altera algum mecanismo na organização.
No *Salesforce* existe o *Lightning Flow* que integra duas tecnologias: o *Flow Builder* e o *Process Builder*.

3.2.1 *Flow Builder*

O *Flow Builder* é uma ferramenta de automação de negócios que permite manobrar dados de diversas formas.
Esse criador possibilita adicionar novos caminhos às ações padrão do *Salesforce*, ou seja, há alguma ação que aciona o fluxo, por exemplo a criação de algum registo.
A formulação do fluxo é feita de forma idêntica à expressão *Case*: pensar em várias condições e ações caso sejam verdadeiras ou falsas. Visualmente, esta forma é mais percetível, mas a nível de execução é mais complicada.

3.2.2 *Process Builder*

O *Process Builder*, tal como o anterior, é uma ferramenta de negócios que permite manobrar dados.
Esta ferramenta é de fácil uso e aprendizagem, pois apenas é necessário pensar nas condições *If-Then-Else* que se quer executar. Contrariamente ao *Flow Builder*, este processo a nível de execução é mais acessível, mas a sua visualização é ligeiramente mais complexa.

3.2.3 *Flow Builder* ou *Process Builder*

As duas tecnologias têm o mesmo propósito, modificar algo no sistema sem fazer uma linha de código, contudo, são muito diferentes [4].

- O *Process Builder* foi construído de forma a ser uma tecnologia de fácil utilização (funciona como se fosse *IF-THEN-ELSE*), mas pode ter um processo complexo;
- O *Flow Builder* funciona como expressões CASE;
- O *Flow Builder* oferece todas as ações do *Process Builder*, adicionando a ação de remover registos;
- No *Process Builder* permite apenas criar/atualizar registos ligados ao registo que acionou o processo e todos são atualizados da mesma forma;
- No *Flow Builder* é possível criar/atualizar qualquer registos que pertençam a uma condição (não necessitam de estar relacionados com o registo acionador).

Capítulo 4

Solução

De forma a construir a solução desta dissertação, explica-se o seu funcionamento, seguido da sua integração com a arquitetura geral do *Salesforce*. Por fim, lista-se todas as funcionalidades existentes e como estas podem ser testadas e validadas.

4.1 Funcionamento do sistema

Na *"RAISE UP"*, o passo mais importante para que a aplicação funcione como pretendido é criar, pelo menos, um critério com, pelo menos, uma regra e indicar o respetivo resultado. Após esse passo, pode-se proceder à classificação dos cliente de três formas diferentes:

1. **isoladamente**: tem de se ir à página de detalhes do cliente que se pretende classificar e clicar no botão "Rank".

2. **massivamente**: encontra-se uma guia na aplicação com o nome "Classifications" onde somente observamos dois botões ("classificar contas"e "classificar *leads*"). Após selecionar o botão do objeto que se pretende classificar em massa, o sistema vai buscar todos os registos deste objeto e classifica-os de acordo com os critérios existentes.

3. **programada**: quando se instala a aplicação na organização, o utilizador pode escolher qual o período de tempo que quer para avaliar automaticamente todos os clientes.

A aplicação está preparada para funcionar com ou sem a *"INFORMA"*, o que significa que o cliente no seu ambiente de produção pode ter as duas aplicações em simultâneo ou apenas a *"RAISE UP"*. No caso de se usar apenas a *"RAISE UP"*, tem-se as seguintes particularidades de funcionamento:

- Os campos mencionados na criação das regras para os critérios não vão conter os pertencentes à *"INFORMA"*;

- Deixa de ter a classificação diária programada (classificação que estava ligada aos alertas provenientes da *INFORMADB* que, caso algum cliente tivesse um novo relatório, reclassificava-o automaticamente).

4.2 Arquitetura do Sistema

A arquitetura do sistema baseia-se na plataforma *Salesforce*, tendo na *AppExchange* (repositório de aplicações do *Salesforce*), duas aplicações (*"INFORMA - Business by data"* e *"RAISE UP"*). A primeira aplicação, foi implementada na empresa (secção 2.1.2) e pode complementar a segunda, aplicação desenvolvida nesta dissertação, de forma a obter o máximo de informações acerca do cliente.
Aprofundando um pouco a arquitetura da *"INFORMA"*, esta realiza pedidos através de *Web Service* à *INFORMADB*, para obtenção dos relatórios dos clientes. Se o utilizador subscrever a receção de alertas sempre que um relatório é atualizado, o utilizador recebe um alerta na aplicação e, consequentemente, um e-mail com a notícia da atualização, que, posteriormente, leva ao envio e armazenamento do relatório na aplicação.
Relativamente ao *"RAISE UP"*, este possibilita a classificação de todos os cliente através da obtenção dos dados contido no *Salesforce*.
Com tudo o que foi mencionado acima, é possível ter duas arquiteturas nesta dissertação, a que integra a *"INFORMA - Business by data"* e a que simplesmente contém a *"RAISE UP"*. Deste modo, as diferentes arquiteturas encontram-se nas imagens 4.1 e 4.2, sendo a primeira referente à integração e a segunda à utilização solitária da aplicação deste trabalho.

4.3 Funcionalidades

A partir da tarefa *desenho da solução* (Apêndice A) é possível definir a estrutura de dados e funcionalidades que a aplicação contém. Desta forma, as funcionalidades definidas são:

- Criação e edição de critérios;

- Classificação dos clientes de acordo com os critérios inseridos.

De forma a construir essas funcionalidades na aplicação, foi necessário definir o que tinha de ser feito no ambiente de desenvolvimento:

- Construção da interface para a definição de critérios;

- Armazenamento do critério;

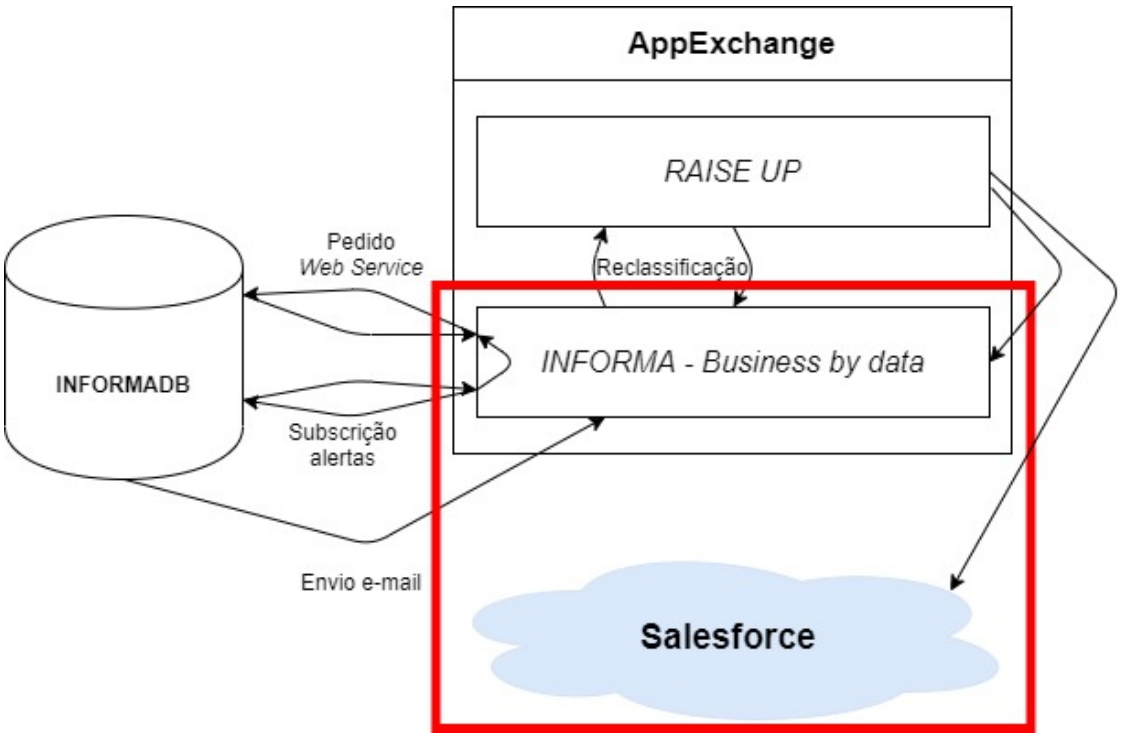


Figura 4.1: Arquitetura do sistema com a integração da *"INFORMA - Business by data"* com a *"RAISE UP"*

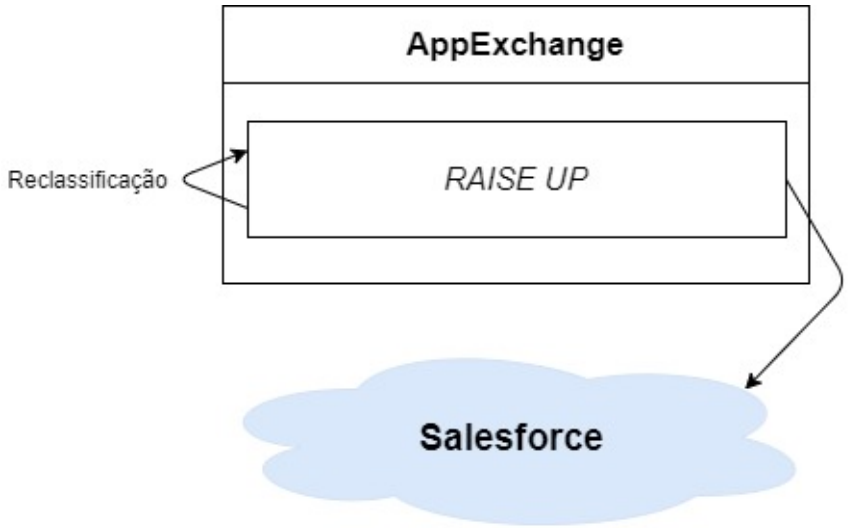


Figura 4.2: Arquitetura do sistema com a utilização apenas da aplicação desta dissertação, *"RAISE UP"*

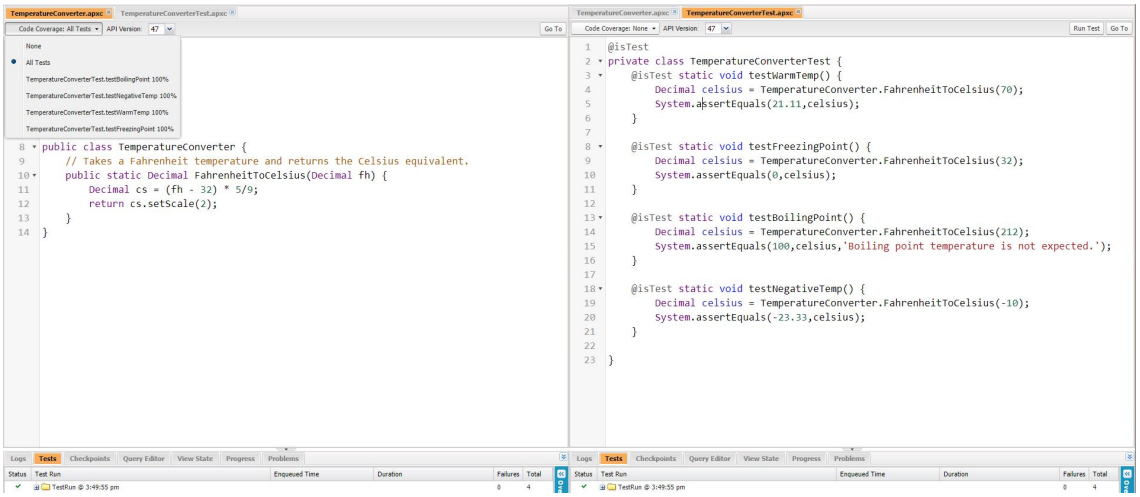
- Processamento do critério para edição;
- Inserção de um mecanismo na página de detalhes da conta/*lead* para efetuar a classificação;
- Criação de um componente para visualização das classificações mais recentes para cada critério;

- Obtenção das classificações mais recentes;
- Construção da interface de listagem do histórico das classificações do cliente;
- Obtenção de todas as classificações ordenadas por data;
- Classificação para um cliente;
- Classificação massiva;
- Reclassificação mensal dos clientes;
- Reclassificação diária de acordo com os relatórios da *Informa*.

No capítulo 5 iremos compreender melhor a construção e execução de cada ponto referido anteriormente, utilizando as tecnologias referidas na secção 2.1.1.4 do capítulo 2.

4.4 Teste e validação

Antes de iniciar o desenvolvimento do projeto e após a finalização do desenho da solução, inicia-se a tarefa "desenho de testes". Nesta tarefa indica-se os casos de uso e qual deve ser o comportamento da aplicação na sua execução (aprofundado no Apêndice C).
Contudo, durante a criação do código, são feitos testes para cada classe/componente criado.
O *Apex* tem uma estrutura própria para a elaboração de testes às suas classes de forma a verificar a quantidade de código que está realmente a ser testado e permite mostrar os resultados obtidos.
Relativamente à cobertura do código que os testes analisam, quando um teste é executado, é possível medir a porção do código da classe que está, realmente, a ser avaliado. Se a cobertura for 100%, a qualidade e quantidade de testes é a exata, se a cobertura for abaixo dos 75%, fica-se a saber que os testes são insuficientes para qualificar a classe.
No que se refere aos resultados esperados, o *Apex* analisa o código de testes e compara o seu resultado com o da classe elaborada. Se os resultados não forem os esperados, sabe-se que há um erro no código ou nos testes, caso os resultados forem o que se espera, considera-se que o código está a fazer o que se pretende.
O objetivo, neste caso, é ter todos os testes com os resultados esperados e com cobertura mínima de 75%. Dado isto, ao correrrmos todos os testes da aplicação e estes estarem com os parâmetros certos, a aplicação está qualificada nos aspetos testados.
De seguida pode-se observar as figuras 4.3a e 4.3b, que ilustram os conceitos mencionados de forma a torná-los mais percetíveis.
No capítulo 6, é possível perceber os testes que foram executados a cada classe existente na aplicação criada.



(a) Classe do *Apex* testada com a demonstração da cobertura em 100%

(b) Classe de teste

Capítulo 5

Funcionalidades

Como já foi mencionado anteriormente, as funcionalidades desta componente dividem-se em dois grandes grupos: declaração de critérios e classificação dos clientes. E, de modo a executar essas funcionalidades, foi cumprido um conjunto de tarefas:

1. Construção da interface para a definição de critérios;
2. Armazenamento dos critérios;
3. Processamento do critério para edição;
4. Inserção de um mecanismo na página de detalhes da conta/*lead* para efetuar a classificação;
5. Criação de um componente para visualização das classificações mais recentes para cada critério;
6. Obtenção das classificações mais recentes;
7. Construção da interface de listagem do histórico das classificações do cliente;
8. Obtenção de todas as classificações ordenadas por data;
9. Classificação para um cliente;
10. Classificação massiva;
11. Reclassificação mensal dos clientes;
12. Reclassificação diária de acordo com os relatórios da *Informa*.

De seguida, analisa-se a construção das tarefas e faz-se uma pequena discussão das diversas e possíveis formas de implementação e a justificação pela escolhida.

5.1 Construção da interface para a definição de critérios

A construção das interfaces no *Salesforce* podem ser feitas de diversas formas: *Visualforce*, *Lightning Web Component* e *Aura Component*.
A implementação usando o *Visualforce* é mais prática no sentido em que, o seu controlador é diretamente a classe *Apex*, e, como já foi referido anteriormente (secção 3.1.1), no *Apex* podemos fazer tudo, inclusive acesso direto à base de dados.
Usando o *Lightning Web Component* (LWC), o controlo da página é feito por uma classe *JavaScript* que tem ligação a uma classe *Apex* para acesso à base de dados. Apesar deste método de implementação ser mais trabalhoso, pois contém mais classes de acesso, o seu *layout* é mais idêntico ao restante *Salesforce*, visto que existem muitas semelhanças no seu funcionamento, criação de ações, páginas *web*, entre outras funcionalidades do *Salesforce*, e, como a sua estrutura é apoiada numa pilha da *web*, o seu funcionamento torna-se mais rápido.
Comparando diretamente o *LWC* e o *Aura Component*, o *Aura Component* tem o funcionamento mais lento devido à sua estrutura não conter pilha na *Web*. Assim, o *LWC* vem em sequência do *Aura* de forma a resolver os problemas e desafios que este causava, facilitando o trabalho dos desenvolvedores no salvamento dos dados [27].
Nesta tarefa, foi escolhida a implementação via *Visualforce* motivada pela necessidade de criar uma página completa.

5.1.1 Implementação

Como se encontra no apêndice A (desenho da solução), a estrutura da página pensada no final foi alterada devido ao possível desenho que se pode ou não ter na *Visualforce* e, sobretudo, no *Salesforce*. Contudo, a estrutura principal é comum: a indicação do nome do critério e a criação de diversas regras. O que se acrescenta é a secção do filtro lógico com a sua classificação.
Em primeiro lugar, antes de se iniciar a criação da página, foi necessário pensar como se poderia colocar as regras juntamente com a lógica. Em segundo lugar, como desenhar esta ideia em *Visualforce* de maneira a ser uma interface de fácil entendimento e rápido uso.
Começando pela primeira preocupação, a ideia da resolução foi retirada de outras funcionalidades que o *Salesforce* oferece, onde se encontra um conjunto de regras e se pode escrever a sua lógica, porque esta não é apenas de *OR/AND*, tal como nos critérios do *Process Builder*.
Passando para a segunda dificuldade, o que demorou mais tempo a concretizar foi a disposição das diversas secções na página e como se iria colocar a criação, tanto das regras como dos filtros lógicos. Ao resolver estes obstáculos com a inserção de uma tabela para ambas as criações, o que se tornou mais desafiador foi a obtenção dos campos disponíveis dos dois objetos utilizados (*Conta e Lead*), a diferenciação dos operadores relativamente ao campo escolhido e a verificação dos valores inseridos no final com os campos e operadores

selecionados.
A *Apex*, linguagem de programação do *Salesforce*, oferece uma vasta gama de bibliotecas que permite aceder a quase todas as funcionalidades da ferramenta. Deste modo, a obtenção dos campos de cada objeto é feita através de uma das bibliotecas disponibilizadas pela ferramenta (*"Schema.SObjectField"*). As restantes dificuldades tiveram de ser resolvidas manualmente, ou seja, implementação de código que verifica os operadores certos para os campos selecionados e a certeza que o valor inserido no último campo de texto estava de acordo com o tipo de campo selecionado.
Mais pormenores a nível de implementação *back-end*, irão ser descritos na tarefa seguinte.

5.2 Armazenamento do critério

Antes de iniciar a descrição da implementação *back-end*, explicação dos métodos necessários na página *Visualforce*, é oportuno relatar o pensamento que foi necessário ter para a construção do modelo de entidade-relação, de forma a albergar toda a informação inserida pelo utilizador e a ser de fácil acesso quando preciso.
Primeiramente, foi pensado quais os campos que eram realmente importantes na base de dados para posterior uso (por exemplo: na edição, na demonstração dos campos, na classificação, etc.). Esses campos são:

- ***Expression API***: filtro lógico com as regras, em modo API (com *API Name* do campo selecionado e os operadores em forma matemática), utilizado para fazer a pesquisa à base de dados na classificação, obtenção das regras na edição do critério. (por exemplo: Name = 'a');
- ***Expression Label***: filtro lógico com as regras, em modo visível, ou seja, com o nome que é mostrado ao utilizador e os operadores por extenso. Utilizado para mostrar ao utilizador nos detalhes do critério, colocação nos campos indicados na página de edição do critério. (por exemplo: Account Name equals 'a');
- **Nome do critério**;
- **Objeto selecionado**: define quais os critérios que vão correr para os objetos relacionados, ou seja, os critérios para o objeto *Account* só correm para registos Conta e os critérios com o objeto *Lead* só correm para registos *Lead*;
- ***Result***: guardado para cada filtro lógico. Quando o registo pertencer ao filtro lógico, obtém-se a referida avaliação, classificando-se assim o critério;
- ***Average***: para saber a média do utilizador para todos os critérios que ele se insere.

Após saber quais os campos essenciais no salvamento do critério e posterior processo de classificação, foi pensado que objetos eram necessários criar e como inserir os campos nestes. O objeto principal é o critério (*Criteria*), sendo este o centro da base de dados

(como se pode observar na imagem do apêndice B).
O objeto *Criteria* guarda, de certa forma, toda a informação necessária de um critério que, posteriormente, será utilizada para proceder à classificação.
O segundo objeto é o *Logic Filter*. Este, como já foi mencionado anteriormente neste documento, tem ligação direta ao objeto *Criteria*, uma relação de *master-details*, que permite ao último objeto referido ter vários filtros lógicos inseridos na sua composição. Por outras palavras, é permitido inserir mais do que um filtro lógico para cada critério, como é visível na composição da página de criação de critérios.
Por último, e não menos importante do que o restante, é o objeto que guarda os registos das classificações efetuadas para cada conta/*lead* avaliada.
As alterações aos objetos existentes (Conta e *Lead*) e a criação e configuração dos novos objetos está detalhada no apêndice A.
Após ultrapassar estes obstáculos, procedeu-se à implementação do armazenamento dos dados inseridos pelo utilizador na criação do critério (no controlador da página *Visualforce*).

5.2.1 Implementação

Iniciando pela forma como é feito o salvamento dos dados durante a criação do critério, foi necessário adicionar duas classes auxiliares de forma a armazenar os dados para cada coluna da tabela, uma para as regras e outra para os filtros lógicos. A classe que guarda os valores das colunas das regras contém as seguintes variáveis: índice, objeto, campo, operador e valor. A classe do filtro lógico contém: índice, filtro e resultado.
Quando se procede ao salvamento do critério, apenas se procede ao seu armazenamento, após ocorrer diversas verificações.

1. Verifica-se se o nome está preenchido (Figura 5.1);

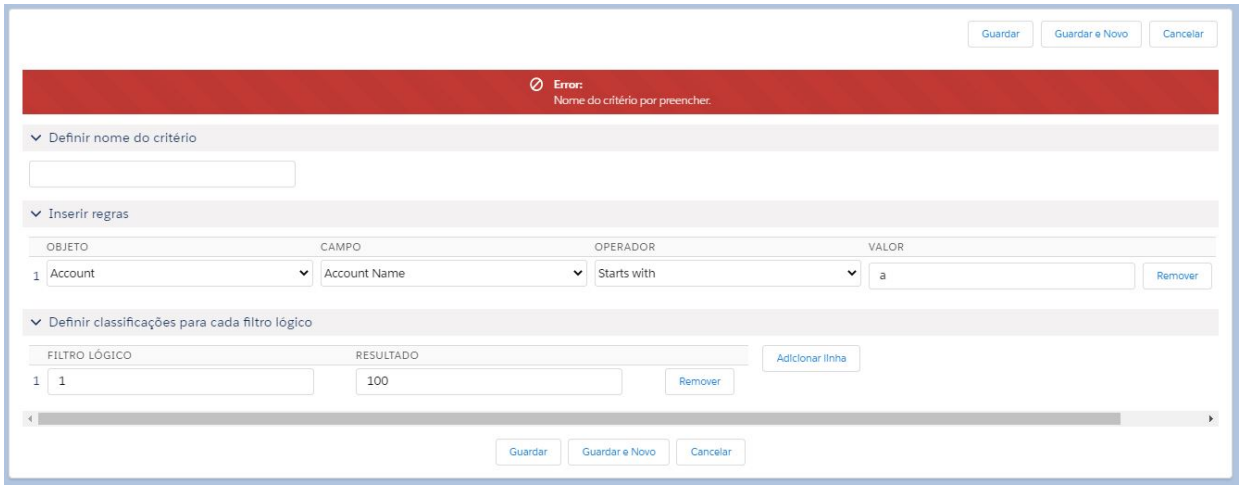


Figura 5.1: Ocorrência de erro quando o critério tem o campo do nome por preencher

2. Verifica-se se o nome inserido já existe no sistema (Figura 5.2);

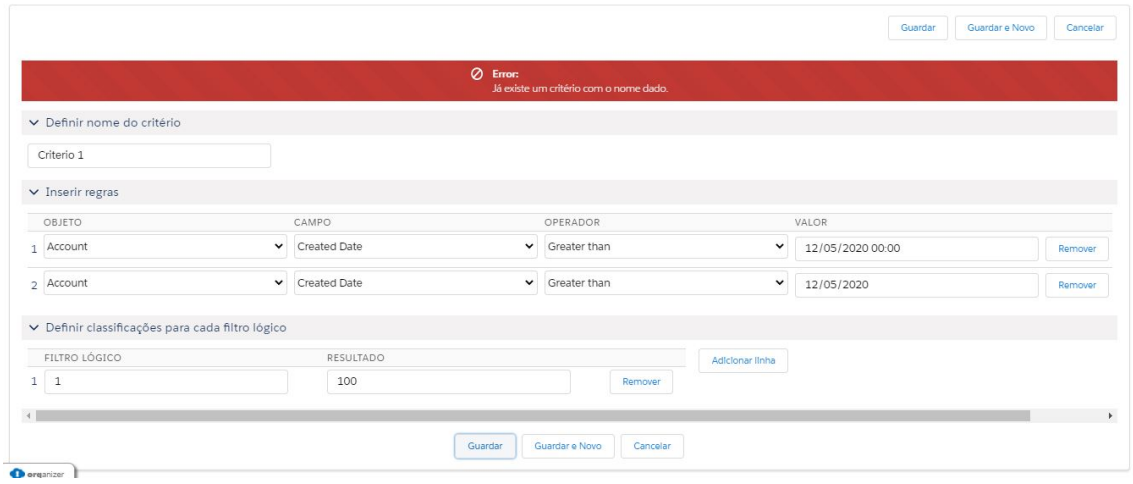


Figura 5.2: Ocorrência de erro quando o nome do critério já existe no sistema

3. Verifica-se todas as regras inseridas nos seguintes parâmetros:

    - se todos os campos estão preenchidos (Figura 5.3);

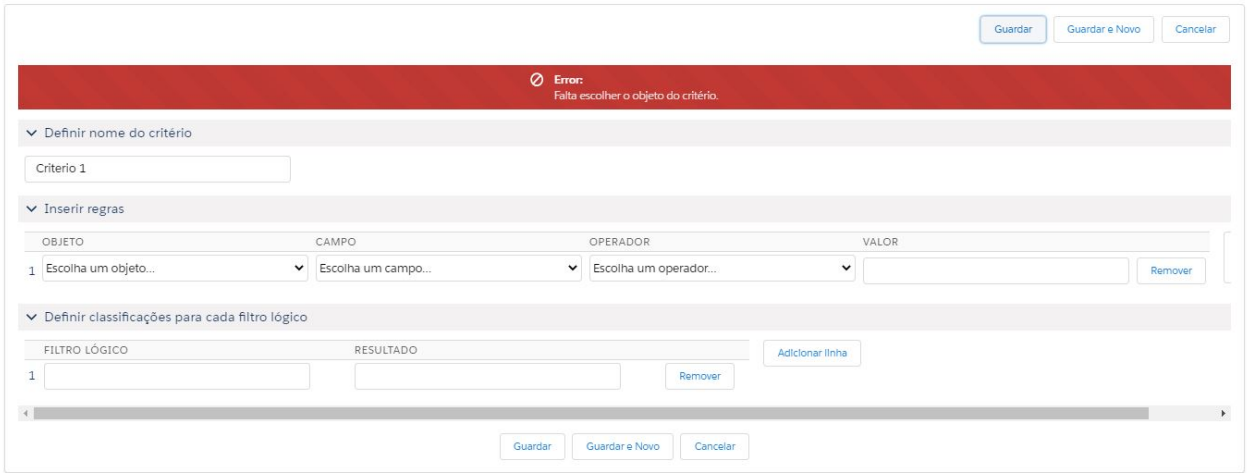


Figura 5.3: Ocorrência de erro quando falta preencher um campo em alguma regra

    - se o valor inserido no campo *value* está no formato pretendido (Figura 5.4);
    - se existem duas ou mais regras iguais (Figura 5.5);

4. verifica-se todos os filtros lógicos inseridos nos seguintes parâmetros:

    - se todos os campos estão preenchidos (Figura 5.6);
    - se existem dois ou mais filtros iguais (Figura 5.7);
    - se o resultado inserido é um número (Figura 5.8);
    - se o resultado inserido está dentro do intervalo de valores [0, 100] (Figura 5.9);

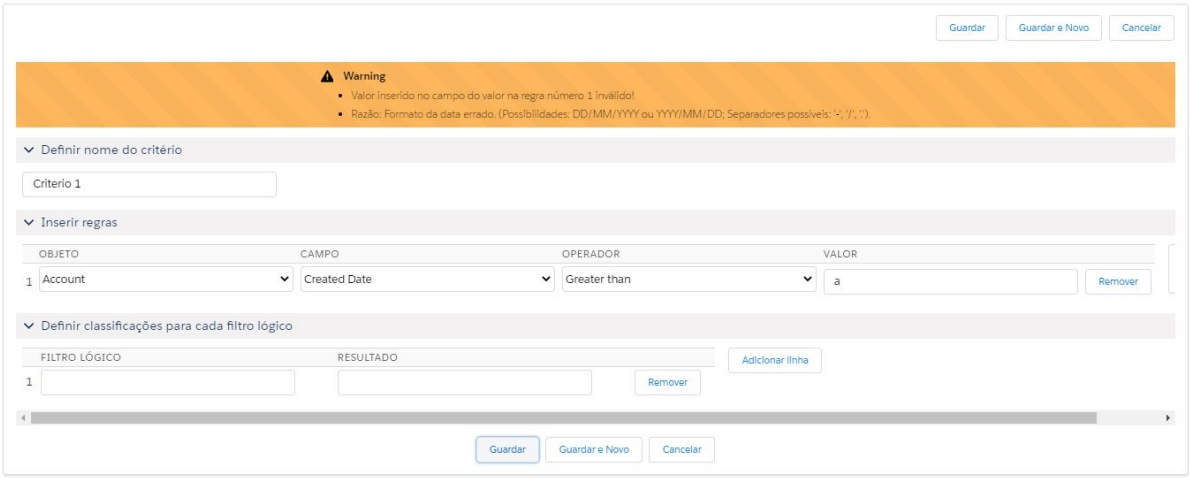


Figura 5.4: Ocorrência de erro quando o valor inserido na regra não é válido

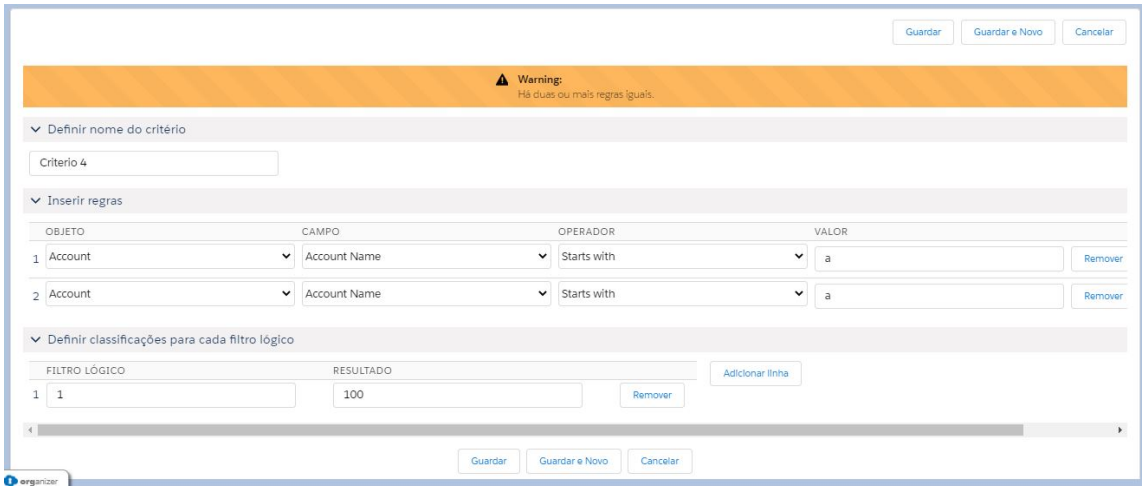


Figura 5.5: Ocorrência de erro quando a regra não é única no critério

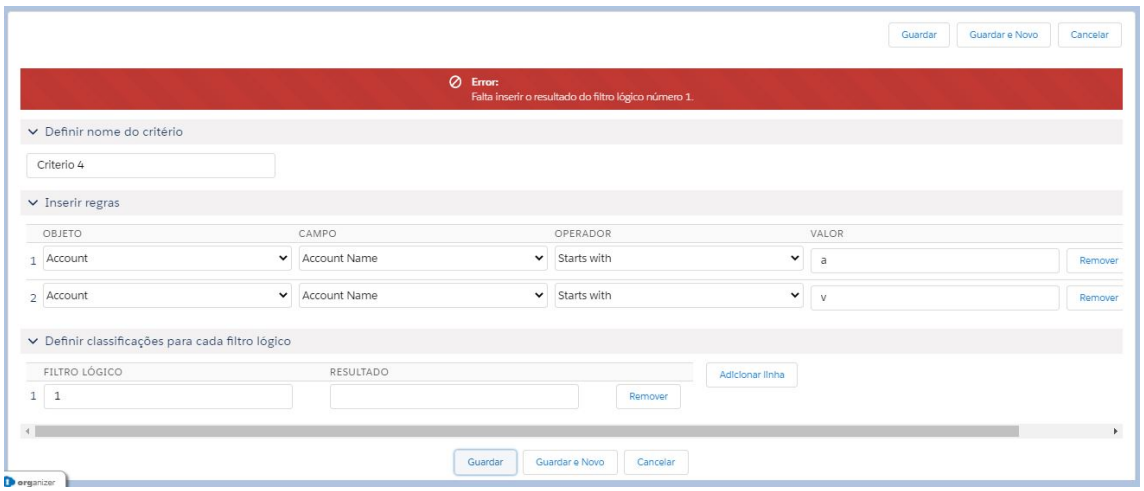


Figura 5.6: Ocorrência de erro quando falta preencher algum campo na tabela dos filtro lógicos

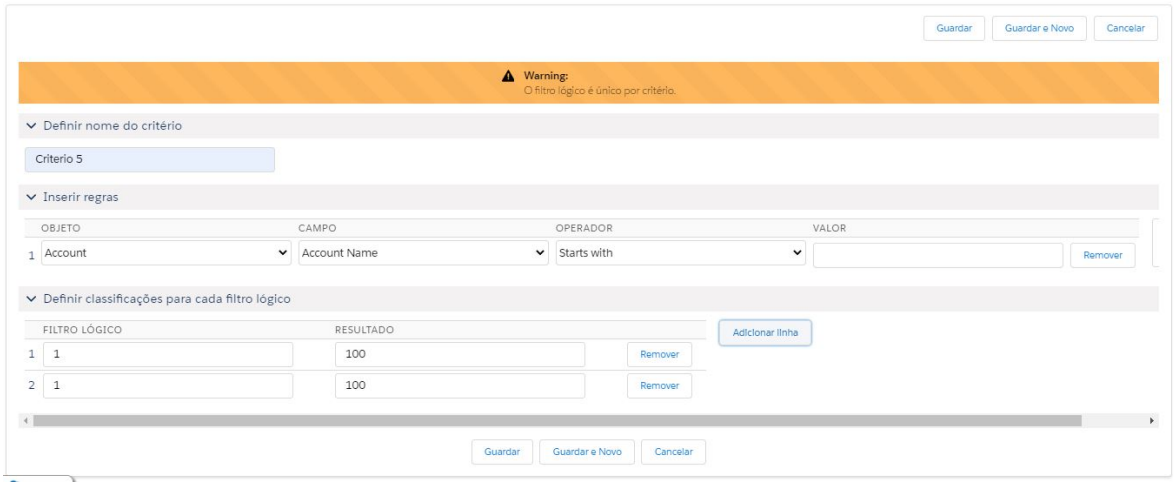


Figura 5.7: Ocorrência de erro quando o filtro não é único no critério

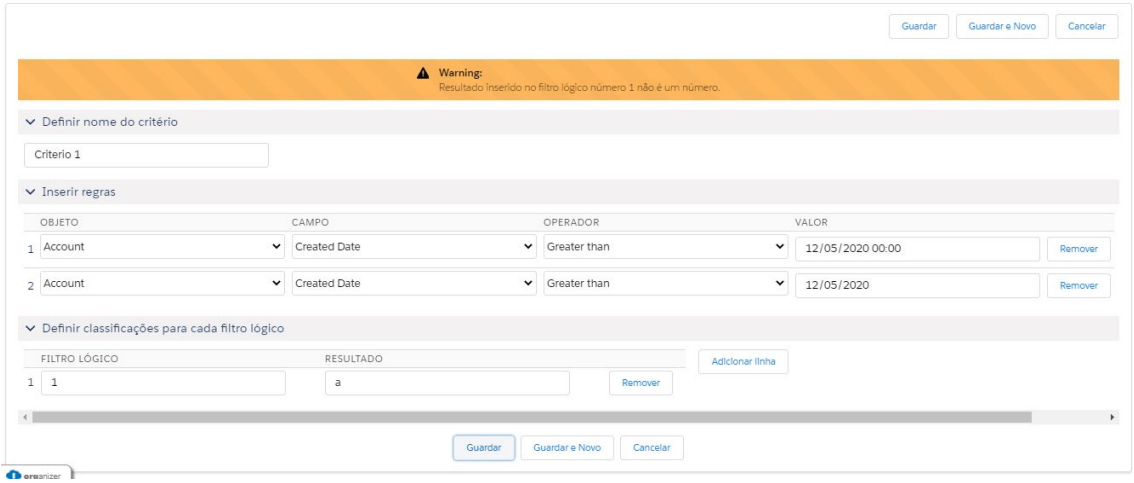


Figura 5.8: Ocorrência de erro quando o resultado inserido em algum filtro não é um número

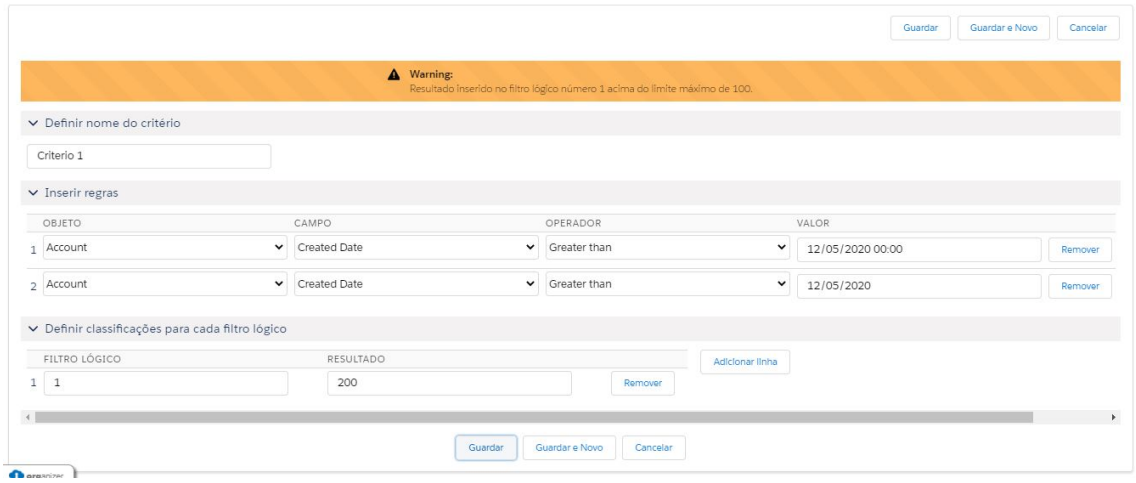


Figura 5.9: Ocorrência de erro quando o resultado inserido está fora do intervalo [0, 100]

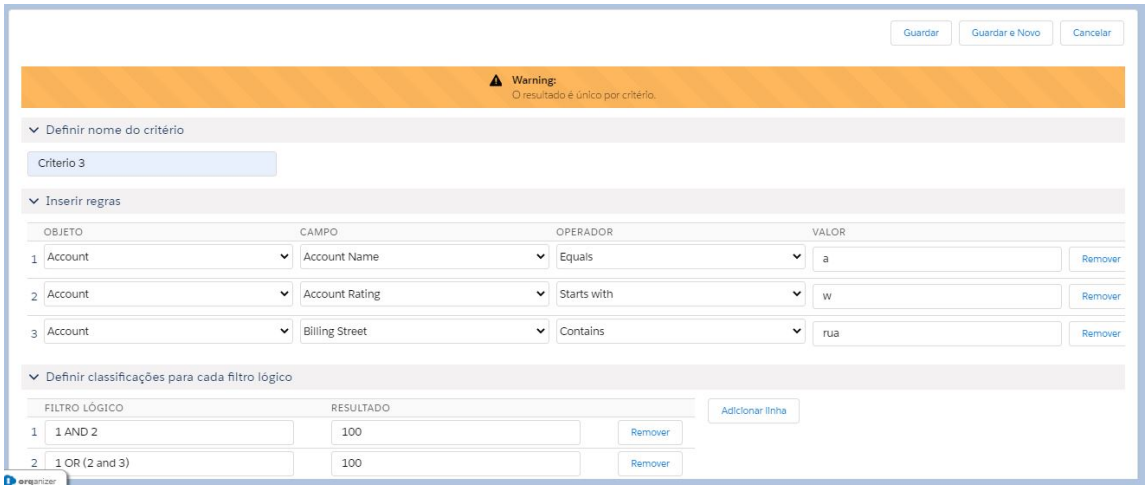


Figura 5.10: Ocorrência de erro quando o resultado não é único no critério

- se o resultado é único no critério (Figura 5.10);
- se a sintaxe do filtro está correta (Figura 5.11);

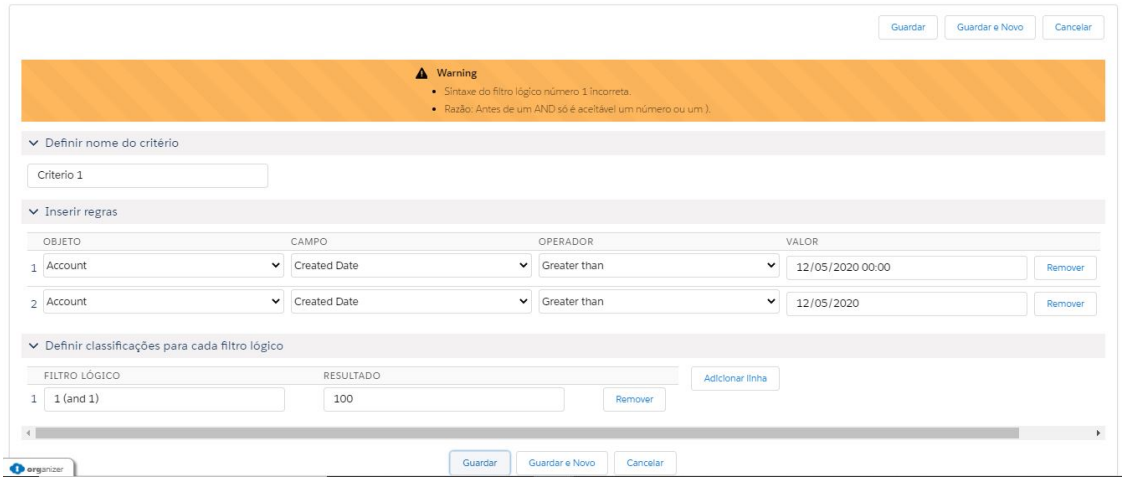


Figura 5.11: Ocorrência de erro quando a sintaxe de algum filtro inserido não está correta

Após todas as verificações mencionadas, dá-se inicio ao processamento, passando os filtros lógicos para dois tipos de expressões: *API* e *Label*. Essas formas foram obtidas utilizando em paralelo as classes geradas, onde se substitui os números mencionados nos filtros lógicos pelos índices das regras na tabela. Em primeiro lugar, foi necessário obter o *API Name* do campo, a forma matemática do operador e, de seguida, transformar a regra na forma *API*. Em segundo lugar, foram necessários criar dois valores para acolher os dois formatos dos filtros lógicos e transformá-los nas duas expressões pretendidas. Não houve necessidade de transformar as regras em formato *Label*, porque é o seu formato pré-definido.
Após ter os formatos dos filtros lógicos de acordo com o pretendido, começa-se por salvar as informações na base de dados. Primeiramente, guarda-se os vários filtros lógicos na

entidade *Logic Filter* indicando, para cada um, o nome do critério, o objeto selecionado, as duas expressões e o seu resultado. De seguida, associam-se ao registo do critério. Quando todos esses passos estiverem executados e com sucesso, o utilizador é redirecionado para a página de detalhes do critério, se tiver feito apenas salvar. Mas se escolheu "salvar e novo", a página sofre uma atualização e começa novamente o processo.

5.3 Processamento do critério para edição

Como foi acima referido, o armazenamento dos dados foi pensado de forma a ter fácil acesso e uma maneira rápida de os mostrar. Deste modo, passamos à explicação de como se efetua o processo para edição do critério.

5.3.1 Implementação

Em primeiro lugar, usa-se a página de criação do critério com a alteração de não se poder salvar e criar um novo critério.
Em segundo lugar, e quando a página é carregada, necessita-se de obter toda a informação do critério para colocar nos campos destinados. Assim, processa-se a informação guardada no registo do critério selecionado para edição:

1. guarda-se o nome;
2. pesquisa-se na entidade *Logic Filter* os registos que pertencem ao critério;
3. processa-se a expressão *Label* dos filtros de forma a obter as regras e substituí-las no filtro pelos índices que cada ocupa na tabela;
4. colocar os filtros resultantes do processo anterior na tabela dos filtros lógicos.

Depois de todo o processamento inicial, a página de edição fica preenchida com toda a informação inserida aquando da sua criação (Fig. C.5).
O utilizador pode editar o que pretender:

- O nome do critério;
- Algum elemento das regras existentes;
- Adicionar uma regra;
- Alterar algum filtro lógico existente;
- Alterar o resultado de algum filtro lógico existente;
- Adicionar um filtro lógico.

Após acabar a edição pretendida e clicar no botão "*save*", o processo de salvamento da informação alterada ocorre da mesma forma ao descrito na tarefa 2, isto é, em primeiro lugar é feita a verificação de cada valor inserido em cada campo, em segundo, os filtros passam para expressões *API* e *Label* e, por último, é tudo atualizado na base de dados.

5.4 Inserção de um mecanismo na página de detalhes da conta/*lead* para efetuar a classificação

Para efetuar a classificação dos clientes no sistema, foi necessário pensar num mecanismo que fosse acessível e que demonstrasse o seu real objetivo.
Para isso, pensou-se em duas possibilidades:

1. Botão na página de detalhes;
2. Botão na guia de classificação da página de detalhes.

A razão pela qual não se optou pela segunda hipótese, deve-se ao facto de o botão não estar visível na página de detalhes, sendo necessário abrir a guia das classificações do cliente para o encontrar. Assim, de modo a este estar sempre acessível e utilizável em todo o *layout* da página, faz mais sentido o botão ficar na página de detalhes juntamente com os padrão do *Salesforce*.

5.4.1 Implementação

O botão tem interação direta com uma página *Visualforce*. Esta página tem, somente na sua composição, uma ação que corre o método de classificar um cliente.

5.5 Classificação para um cliente

Esta tarefa vem na sequência da anterior e que explica o funcionamento do mecanismo escolhido para classificar apenas um cliente.

5.5.1 Implementação

Este processo realiza a classificação, e o processo é executado da seguinte forma:

1. Obter qual o objeto a que o cliente pertence (*Account* ou *Lead*);
2. Obter todos os critérios do objeto obtido;
3. Obter as últimas classificações de cada critério do cliente;
4. Correr todos os critérios para o cliente em causa:

- Se o cliente corresponder a algum filtro do critério, atualiza-se/insere-se essa classificação na base de dados mencionando a expressão *Label* e o resultado correspondente, o critério e a menção que é a classificação mais recente para o critério;
- se o cliente não corresponder, atualiza-se/insere-se mas com o resultado a 0 e a expressão como *NA*;

5. Calcula-se a nova média das classificações e atualiza-se o registo do cliente.

5.6 Criação de um componente para visualização das classificações mais recentes para cada critério

Na página de detalhes de um objeto, quando este está interligado com outro através de um relacionamento de pesquisa ou de mestre-detalhes, pode aparecer uma lista de registos do objeto relacionado automaticamente na lista de relacionados quando assim se pretende. Podia-se optar pela demonstração da lista de classificações nos relacionados, mas não teria apenas a lista das mais recentes para cada critério, teria todas as existentes. Por este motivo, foi necessário criar um componente que listasse apenas as classificações mais recentes de cada critério, e para isso havia várias hipóteses:

- Página *Visualforce*;
- *Lightning Web Component*;
- *Aura Component*.

Como foi mencionado na secção 3.1.3, o *Lightning Component* é mais adequado para implementar um complemento a uma página, enquanto que o *Visualforce* é recomendado para fazer uma página inteira. Assim sendo, optou-se pelo *Lightning Component*, uma vez que se pretende fazer um componente que contém uma lista para colocar na página de detalhes do registo.
Antes de passar à criação do componente na página de detalhes, foi necessário pensar como se iria disponibilizar:

1. se colocar na guia dos relacionados; ou
2. se adicionar uma guia na secção do centro para as classificações.

Uma vez que não é possível ter um componente criado para a lista na zona dos relacionados, foi necessário criar uma guia que inclui as classificações e permite ao utilizador ter um acesso mais imediato.

5.6.1 Implementação

Em primeiro lugar, na edição da página dos detalhes do objeto, adicionou-se a guia para as classificações na secção do centro com o nome *"Classifications"*.
Em segundo lugar, criou-se um *Lightning Web Component* para implementar a demonstração da lista e ter um botão que permite aceder ao histórico das classificações do cliente. O *LWC* tem duas classes que são automáticamente criadas com ele, uma "*.html*" e outra "*.js*". A *HTML* é para desenhar a página como a pretendemos, a *JavaScript* dá apoio técnico à *HTML* e consegue-se aceder a uma terceira classe *Apex*, se for necessário.
Na classe *HTML* foram criados apenas dois componentes: um que adiciona à cabeça o botão do histórico e outro que contém a tabela das classificações. Esta tabela tem três colunas que são definidas no *JavaScript*: nome da classificação (com link direto para a página de detalhes desta), o nome do critério e o resultado. Esta lista é obtida através de uma classe *Apex* que teve de ser criada para o efeito e tem apenas um método que obtém da base de dados as últimas classificações do cliente para cada critério.

5.7 Obtenção das classificações mais recentes

Como foi referido acima, foi necessário criar uma classe *Apex* para obtenção das classificações mais recentes de cada critério para um dado cliente.

5.7.1 Implementação

Para mandar a informação de uma classe *Apex* para a classe *JavaScript* de um *Lightning Web Component*, é necessário enviar a informação de maneira a ser fácil de introduzir no *HTML*. Por este motivo, foi preciso criar uma classe extra com as variáveis especificas para o *JavaScript*: url (indica qual o link de encaminhamento se clicar no nome da classificação), nome da classificação, critério e resultado. Além disso, e como já foi referido, tem apenas um método que faz a pesquisa à base de dados e obtém da entidade *"Classification"* os registos mais recentes para cada critério do cliente mencionado. Quando se tem toda a informação precisa, preenche cada variável da classe auxiliar e retorna a lista.

5.8 Construção da interface de listagem do histórico das classificações do cliente

Sempre que se refere "criação de interfaces", e, como já foi mencionado nas tarefas anteriores, há 3 hipóteses: *Visualforce*, *Lightning Web Component* e *Aura Component*.
Como neste caso é necessário fazer uma nova página com a listagem de todas as classificações para um cliente, e não interfere com um *layout* já existente, optou-se por escolher o *Visualforce*.

5.8.1 Implementação

A página *Visualforce* contém, no topo, um campo do tipo *link* com o nome do cliente onde, ao clicar, direciona-se para a página de detalhes. Abaixo desse campo, encontra-se a tabela com a listagem de todas as classificações de todos os critérios (para aquele objeto), onde as colunas são:

- nome do critério com o campo do tipo *link* que, tal como acontece no campo de topo, quando se clica, direciona-se para a página de detalhes do critério;
- filtro lógico no formato *Label*;
- resultado;
- data de modificação.

5.9 Obtenção de todas as classificações ordenadas por data

De modo a obter, na página do histórico, a lista das classificações do cliente, é necessário ter uma classe *Apex* como servidor, como já é do nosso conhecimento.

5.9.1 Implementação

Como foi dito para a classe de apoio à criação do critério, foi necessário criar uma classe auxiliar para sustentar a informação que era posta/obtida da tabela. Neste caso, como se vai colocar informação numa tabela, também foi necessário criar uma classe auxiliar para guardar a informação que vai ser necessária para cada linha da tabela: índice, id, nome do critério, expressão do filtro lógico, resultado e data de modificação.
Partindo para o objetivo principal, começa-se por obter da base de dados todas as classificações do cliente ordenadas pela data de modificação, de seguida, adiciona-se, para cada linha, os valores às variáveis da classe auxiliar de modo a preencher a tabela com todos os valores necessários.

5.10 Classificação massiva

Nesta tarefa é necessário mencionar dois aspetos: Visual e Processamento. Isto é necessário porque, a classificação massiva pode ser feita manualmente, mas também é feita automaticamente (como será explicado nas duas próximas tarefas). Para ser feita manualmente, é necessário haver algum mecanismo visual que permita ao utilizador efetuar esta tarefa.
Então, para isso, foi necessário pensar numa forma de disponibilizar ao utilizador esse mecanismo e a única ideia foi a criação de uma guia geral chamada *"Classifications"*, onde encontramos apenas dois botões para classificar todas as contas e/ou todas as *leads*.

O processamento por detrás dos dois botões e das duas tarefas seguintes é o mesmo e será explicado na seguinte subsecção.

5.10.1 Implementação

Em primeiro lugar, começa-se por explicar como foi implementada a guia para a classificação massiva e, de seguida, explica-se de como foi feita a classificação para todas as contas/*leads* registadas no sistema.
Começando pela guia, foi criada como sendo *"Visualforce tab"*, devido à informação da página ser uma *Visualforce* criada para o efeito. Esta página contém, como já foi indicado, apenas dois botões: *Classificar contas* e *Classificar Leads*. Os dois botões têm o mesmo processamento, apenas se distinguem no conjunto de critérios que vão ser analisados, ou seja, para as contas, apenas irão conter critérios cujo objeto é *Account*, e para as *Leads*, o objeto é *Lead*.
Relativamente ao processamento da classificação massiva, este é ligeiramente simples. Quando o utilizador clica num dos botões, é criado um *batch* que executa a classificação de 200 em 200 registos. O *batch* é um mecanismo do *Salesforce* que permite efetuar uma ação para quantos registos quisermos, sendo que, de cada vez, pode correr no máximo 200 registos. A classe deste mecanismo tem três métodos: *start*, *execute* e *finish*. O método *start* serve para obter da base de dados a informação que é necessária para o método seguinte *execute*. Este método *execute* faz toda a ação pretendida pelo desenvolvedor, neste caso, a classificação. A classificação neste método é feita de forma igual à referida na tarefa 5.5, classificação para um cliente, só que para todos os registo obtidos no primeiro método. No *finish* pode-se deixar vazio ou enviar uma notificação ao utilizador a dizer que a ação já terminou e qual o resultado dela, se foi bem sucedido ou se houve algum problema.

5.11 Reclassificação mensal dos clientes

Esta tarefa tem o processamento da tarefa referida acima, mas a sua implementação tem uma adição. Esta tarefa é feita automaticamente e para isso é necessário mecanizar.

5.11.1 Implementação

Quando se instala a aplicação no ambiente de desenvolvimento, uma das instruções é automatizar a classificação dos clientes.
Para isso, basta ir às configurações, escrever *Apex Classes*, abrir, clicar no botão *Schedule Apex*. Cria-se um novo registo, escolhe-se a classe da classificação massiva, coloca-se o período mensal e a hora que o utilizador achar mais oportuna para a sua realização.

5.12 Reclassificação diária de acordo com os relatórios da *Informa*

Esta tarefa é muito semelhante à anterior, havendo uma ligeira diferença, ou seja, esta tarefa apenas faz sentido ser executada, se a organização usufrui da aplicação *"INFORMA - Business by data"*.

5.12.1 Implementação

A execução desta tarefa é igual à anterior, mas no momento de escolha da classe, opta-se pela que contém a *"INFORMA"* no nome e na periodicidade, predefine-se como diariamente e o horário que o utilizador pretender.

Capítulo 6

Avaliação

Ao proceder à avaliação da aplicação podemos seguir dois caminhos:

1. Diferença na velocidade de arranque das páginas com novos elementos;
2. Bateria de testes.

6.1 Comparação dos tempos de arranque das páginas

Começando pela primeira, apresenta-se as seguintes tabelas que consistem em mostrar os tempos de arranque em cinco procedimentos de cada página e a sua média, sendo a primeira para as páginas sem novos componentes e a segunda com os componentes.

Página	Proc. 1	Proc. 2	Proc. 3	Proc. 4	Proc. 5	Média
Detalhes da Conta	7.6s	6.8s	5.6s	7.0s	7.8 s	6.96s
Detalhes da *Lead*	5.5s	6.5s	5.8s	5.6s	5.2s	5.72s
Criação do critério	5.4s	6.2s	4.4s	4.0s	5.3s	5.06s

Tabela 6.1: Análise da velocidade antes da inserção dos novos componentes

O tempo de arranque das páginas tem alguns fatores que influenciam o seu desenvolvimento, como por exemplo, a velocidade da rede e do navegador.
Analisando os dados das duas tabelas, concluí-se que a adição dos componentes não influenciam de forma significativa os seus tempos de arranque. Nem mesmo com todos os componentes presentes ocorre uma grande alteração no tempo. Assim, pode-se determinar que o processamento do novo código é feito quase à mesma velocidade que o nativo.
No meu ponto de vista e antes da realização dos testes, pensei que iria haver um ligeiro aumento no tempo devido à quantidade de dados que foram adicionados ao processamento. Contudo, como o *Salesforce* é processado na nuvem, e muita informação fica guardada em

Página	Proc. 1	Proc. 2	Proc. 3	Proc. 4	Proc. 5	Média
Detalhes da conta com guia das classificações	6.8s	6.7s	5.8s	5.8s	5.0s	6.02s
Detalhes da conta com botão de classificação	6.4s	7.9s	6.2s	5.8s	5.3s	6.32s
Detalhes da conta com ambos	7.3s	5.5s	5.4s	5.0s	7.7s	6.18s
Detalhes da *Lead* com guia das classificações	6.5s	5.8s	5.0s	5.2s	7.0s	5.9s
Detalhes da *Lead* com botão de classificação	6.0s	5.0s	5.5s	5.4s	6.5s	5.68s
Detalhes da *Lead* com ambos	7.5s	5.7s	6.2s	5.5s	5.8s	6.14s
Criação do critério (*Visualforce*)	6.4s	7.2s	8.3s	6.0s	5.5s	6.68s

Tabela 6.2: Análise da velocidade depois da inserção dos novos componentes

cache, faz sentido que, ao fazer refrescagem à página, os tempos não variem muito, não considerando o fator da velocidade da rede, que varia conforme o tráfego.

6.2 Bateria de testes

Passando ao ponto 2, foi necessário efetuar uma bateria de testes para confirmar a qualidade do código implementado. Deste modo, cada classe criada tem o seu próprio teste.

- Controlador da página de criação do critério;
- Controlador da página de edição do critério;
- Classe de apoio à listagem das classificações mais recentes;
- Controlador do histórico das classificações;
- Controlador do processo de classificação individual;
- Classe da classificação massiva;
- Classe da classificação massiva através dos relatórios da Informa.

Os testes às classes são feitos para cada método e pensando nas várias alternativas existentes. Assim, começamos por analisar os testes para cada classe.

6.2.1 Controlador da página de criação do critério

Antes de iniciar a execução do teste a esta classe, é necessário dizer que foi criada uma classe que contém os métodos em comum na criação e edição do critério. Com esta informação é óbvio que muito do que se testará aqui, será igualmente testado na seguinte classe (6.2.2).
Para realização dos testes, começamos por analisar os métodos que a classe contém:

1. Adição de uma linha na tabela das regras;
2. Remoção de uma linha na tabela das regras;
3. Adição de uma linha na tabela dos filtros lógicos;
4. Remoção de uma linha na tabela dos filtros lógicos;
5. Salvar;
6. Salvar e criar um novo.

Os métodos de salvamento têm mais umas quantas verificações necessárias:

- Se o nome do critério está preenchido;
- Se o nome do critério existe no sistema;
- Se todos os campos das regras estão preenchidos;
- Se o valor inserido no campo *value* da regra está de acordo com o tipo do campo escolhido;
- Se as regras são únicas no critério;
- Se todos os campos dos filtros lógicos estão preenchidos;
- Se os filtros inseridos são únicos no critério;
- Se o valor inserido no resultado é um número;
- Se o valor inserido no resultado está contido no intervalo [0, 100];
- Se o resultado inserido é único no critério;
- Se a sintaxe do filtro está bem construída.

Após a análise ao que deve ser testado, passa-se à execução dos vários testes a essa classe e procura-se a cobertura de 100%.
No final de cada teste, faz-se a verificação dos erros ou da inserção do critério na base de dados. Há a possibilidade de comparar o resultado esperado com o certo, e é isso que dá a certeza que os testes efetuados, apesar de dar 100% de cobertura à classe, dão os resultados que são os pretendidos.
A cobertura dos testes obtida na classe *"CriteriaMethods"* é de 81% (Figura 6.1) e na classe *"RegisterCriteriaController* é de 82% (Figura 6.2). Ao se observar as figuras, há dois métodos em comum que não foram testados, o *RemoveRowR* e o *RemoveRowF*. Estes métodos são usados na página para remover a linha selecionada, e, como tal, na execução de testes não se consegue remover uma linha certa pois não temos o seu índice. Na classe do registo, houve outro método que não se conseguiu testar, *SaveAndNew*, porque o *url* de

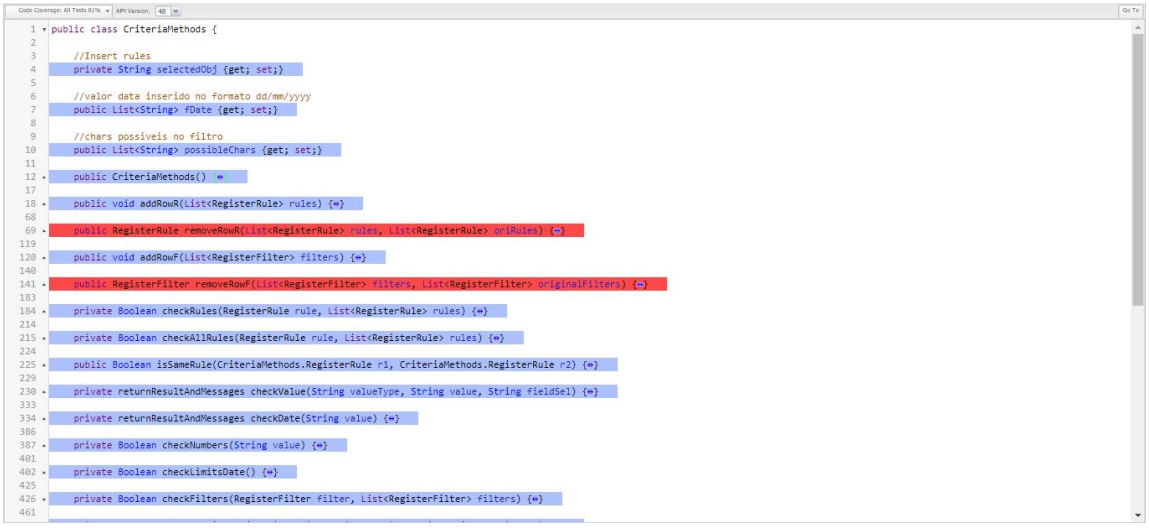


Figura 6.1: Cobertura de testes da classe *"CriteriaMethods"*

Figura 6.2: Cobertura de testes da classe *"RegisterCriteriaController"*

direcionamento dá *null* uma vez que em testes não se tem um *url* para fazer refrescagem. Por estes motivos, as coberturas não chegam aos 100%, contudo são acima de 75% e pode-se concluir que as duas classes estão qualificadas na sua execução.

6.2.2 Controlador da página de edição do critério

Como foi mencionada na explicação da classe acima, houve a necessidade de criar uma classe intermédia que agrega os métodos em comum nas duas classes. Assim, nestes testes, só há o extra de obter a informação de cada campo, além de que não há a possibilidade de guardar e voltar a criar outro.
A diferença da classe de testes anterior para esta é que, em vez de se criar tudo de novo, tem de se ter informação já adicionada previamente no ambiente (classe).

6.2.3 Classe de apoio à listagem das classificações mais recentes

Sendo uma classe de apoio a um *Lightning Web Component*, não é necessário uma classe de testes, contudo, para verificar que a informação que é enviada está correta, foi criada a classe de testes (Figura 6.3). Esta classe de testes tem de começar por ter dados previamente inseridos para que se tenha a certeza que está a retornar o que é esperado (Figura 6.4).

```
static void getClassifications() {
    Criteria__c criteria = new Criteria__c();
    criteria.Name = 'Test account 1';
    Criteria__c criteria1 = new Criteria__c();
    criteria1.Name = 'Test account 2';
    Insert new List<Criteria__c> {criteria, criteria1};

    Logic_Filter__c filter1 = new Logic_Filter__c();
    filter1.Name='filtro1';
    filter1.Object__c = 'Account';
    filter1.Expression_API__c = 'AnnualRevenue > 1000 AND Name like \'t%\' ';
    filter1.Expression_Label__c = 'Annual Revenue Greater than 1000 AND Account Name Starts with t';
    filter1.Result__c=100;
    filter1.Criteria__c=criteria.Id;

    Logic_Filter__c filter2 = new Logic_Filter__c();
    filter2.Name='filtro2';
    filter2.Object__c = 'Account';
    filter2.Expression_API__c = 'Name like \'t%\' ';
    filter2.Expression_Label__c = 'Account Name Starts with t';
    filter2.Result__c=100;
    filter2.Criteria__c=criteria1.Id;
    Insert new List<Logic_Filter__c>{filter1, filter2};

    Account a = new Account();
    a.Name = 'teste';
    a.AnnualRevenue = 500;
    insert a;

    List<Account_Classification__c> classifs = new List<Account_Classification__c>();
    Account_Classification__c ac1 = new Account_Classification__c();
    ac1.Account__c = a.Id;
    ac1.Criteria__c = criteria.Name;
    ac1.Expression_Label__c = filter1.Expression_Label__c;
    ac1.Last_Classification__c = 'yes';
    ac1.Result__c = filter1.Result__c;
    ac1.Update_Date__c = Datetime.now();
    classifs.add(ac1);

    Account_Classification__c ac2 = new Account_Classification__c();
    ac2.Account__c = a.Id;
    ac2.Criteria__c = criteria1.Name;
    ac2.Expression_Label__c = filter2.Expression_Label__c;
    ac2.Last_Classification__c = 'yes';
    ac2.Result__c = filter2.Result__c;
    ac2.Update_Date__c = Datetime.now();
    classifs.add(ac2);
    Insert classifs;

    system.debug(LastClassAccount.getLastClass(a.Id));
```

Figura 6.3: Resultado retornado pelo teste às classificações mais recentes

```
[54]|DEBUG|(IdAndClass:[criteria=Test account 1, name=a023h00000FoeWF, result=100, url=/a023h00000FoeWF], IdAndClass:[criteria=Test account 2, name=a023h00000FoeWG, result=100, url=/a023h00000FoeWG])
```

Figura 6.4: Resultado retornado pelo teste às classificações mais recentes

6.2.4 Controlador do histórico das classificações

Tal como já foi referido nas classes anteriores, o objetivo é comparar os resultados que são retornados. Para isso, é necessário inserir dados de teste para que possamos ver se estes

são os retornados no pedido. Assim, para testar essa classe é preciso apenas inserir esses dados e correr a classe do histórico, porque esta apenas tem um método de obtenção das classificações do registo por ordem decrescente da data de modificação. A classe de teste é muito semelhante à da subsecção anterior.

6.2.5 Controlador do processo de classificação individual

Para as próximas classes, o principal objetivo é classificar registos, seja individualmente ou massivamente. Para isso, há uma classe comum a todas que efetua a classificação de cada registo e calcula a média das classificações mais recentes de cada critério.
No método de classificação é necessário percorrer vários caminhos:

1. Registo ser Conta;
2. Registo ser *Lead*;
3. O registo não pertencer a nenhum filtro do critério;
4. O registo pertencer a pelo um filtro;
5. Ser a primeira classificação para o critério;
6. Ser a reclassificação para o critério (resultado e filtro diferentes);
7. O registo não alterar a classificação que já tinha para um critério.

O passo a seguir a essa análise é a execução dos testes, criando um para cada caminho. No final, optem-se a avaliação completa do processo de classificação para cada registo. Voltando à classe inicial, a que classifica apenas um registo, é apenas necessário criar uma conta ou *Lead*, criar, pelo menos, um critério para cada objeto e, pelo menos, uma classificação para a conta/*Lead* criada. No final, quando executado o teste, é verificado se a média retornada é a esperada e, assim, fica-se com a certeza que os testes estão bem executados. Na figura 6.5 observa-se o método de classificação de uma *Lead*. Na última linha tem-se a expressão*"system.assertEquals"* onde se verifica a média que se pretende com a que foi calculada.

6.2.6 Classe da classificação massiva

Os testes para essa classes já está a maior parte feita na classe anterior. A única diferença é a forma de testar um *batch* (Figura 6.6).

6.2.7 Classe da classificação massiva através dos relatórios da Informa

Esta classe tem a forma de teste igual à anterior, contudo os registos das contas/*leads* são obtidos através dos relatórios da *Informa* criados no dia anterior (Figura 6.7).

No final da execução de todos os testes, o esperado é que a cobertura total seja maior ou igual a 75%, sendo que o ideal seria os 100%.

```
Criteria__c criteria1 = new Criteria__c();
criteria1.Name = 'Test lead 2';
Insert new List<Criteria__c> {criteria, criteria1};

Logic_Filter__c filter1 = new Logic_Filter__c();
filter1.Name='filtro1';
filter1.Object__c = 'Lead';
filter1.Expression_API__c = 'AnnualRevenue > 1000 AND LastName like \'t%\' ';
filter1.Expression_Label__c = 'Annual Revenue Greater than 1000 AND Last Name Starts with t';
filter1.Result__c=100;
filter1.Criteria__c=criteria.Id;

Logic_Filter__c filter2 = new Logic_Filter__c();
filter2.Name='filtro2';
filter2.Object__c = 'Lead';
filter2.Expression_API__c = 'LastName like \'t%\' ';
filter2.Expression_Label__c = 'Last Name Starts with t';
filter2.Result__c=100;
filter2.Criteria__c=criteria1.Id;

Logic_Filter__c filter3 = new Logic_Filter__c();
filter3.Name='filtro3';
filter3.Object__c = 'Lead';
filter3.Expression_API__c = 'AnnualRevenue = 1000 AND LastName like \'t%\' ';
filter3.Expression_Label__c = 'Annual Revenue Equals 1000 AND Last Name Starts with t';
filter3.Result__c=80;
filter3.Criteria__c=criteria.Id;
Insert new List<Logic_Filter__c>{filter1, filter2, filter3};

Lead l = new Lead();
l.lastName = 'teste';
l.AnnualRevenue = 5000;
l.Company = 'teste';
insert l;

Lead_Classification__c lc1 = new Lead_Classification__c();
lc1.Lead__c = l.Id;
lc1.Criteria__c = criteria.Name;
lc1.Expression_Label__c = filter3.Expression_Label__c;
lc1.Last_Classification__c = 'yes';
lc1.Result__c = filter3.Result__c;
lc1.Update_Date__c = Datetime.now();
insert lc1;

ApexPages.StandardController sc = new ApexPages.StandardController(l);
ClassificationLead ca = new ClassificationLead(sc);

ca.getClassification();

List<Lead> lead = [Select id, average_classification__c from lead where id = :l.Id];
system.assertEquals(100, lead[0].average_classification__c);
```

Figura 6.5: Método de teste onde se classifica apenas uma *Lead*

6.3 Melhorias e possíveis adições

Nesta aplicação, muitas das funcionalidades podiam estar melhor executadas. Esta afirmação é feita pelo facto de ser o primeiro contacto com esta plataforma e, no decorrer do tempo, fui ganhando alguma experiência, o que faz com que perceba que alguns aspetos,

```
@IsTest
static void testBatchObjTypeAccount(){

    Account acc1 = new Account(Name = 'test1', AnnualRevenue=1080);
    Account acc2 = new Account(Name = 'test2');
    Insert new List<Account>{acc1, acc2};

    Test.startTest();

    Database.executeBatch(new ClassifyObjectBatch('Account'));

    Test.stopTest();

    List<Account> lstAccs = [SELECT Average_Classification__c FROM Account];

    System.assert(lstAccs[0].Average_Classification__c != null, 'account was not classified!');

}
```

Figura 6.6: Exemplo de teste a uma classe *batch*

```
@IsTest
static void testBatchObjTypeAccountWithInfo(){

    Account acc1 = new Account(Name = 'test1', AnnualRevenue=1080);
    Account acc2 = new Account(Name = 'test2');
    Insert new List<Account>{acc1, acc2};

    List<wit_raise__CreditInfo__c> credInfos = new List<wit_raise__CreditInfo__c>();
    wit_raise__CreditInfo__c info1= new wit_raise__CreditInfo__c(wit_raise__Account__c=acc1.Id);
    wit_raise__CreditInfo__c info2= new wit_raise__CreditInfo__c(wit_raise__Account__c=acc2.Id);
    Insert new List<wit_raise__CreditInfo__c>{info1, info2};

    Test.startTest();

    Database.executeBatch(new ClassifyObjectBatch('Account', true));

    Test.stopTest();

    List<Account> lstAccs = [SELECT Average_Classification__c FROM Account];

    System.assert(lstAccs[0].Average_Classification__c != null, 'account was not classified!');

}
```

Figura 6.7: Método de teste *batch* com integração da *"INFORMA"*

a nível de código, possam ser melhorados. Relativamente ao nível da interface, este não necessita de alteração, pois, a meu ver, está bem pensado e implementando.
No futuro, pode-se adicionar mais funcionalidades, tais como:

- Classificar os registos que tenham o mesmo NIF, assim, podia-se ter uma ideia generalizada de todos os registos pertencentes ao mesmo cliente, melhorando a ideia do negócio a propor;
- Na página de detalhes do critério, ter uma lista dos clientes com a sua respetiva classificação, de modo a ser mais fácil identificar o(s) cliente(s) ideal(ais) para a

realização do negócio.

Bibliografia

[1] K. Anastasov. *Java vs Apex 5 Key Points*. URL: https://www.linkedin.com/pulse/java-vs-apex-5-key-points-kiril-anastasov. (accessed: 20.10.2020).

[2] R. Dantas. *Conheça a história da Salesforce, líder no mercado de cloud computing*. URL: https://blog.vindi.com.br/conheca-a-historia-da-salesforce-lider-no-mercado-de-cloud-computing/. (accessed: 29.01.2020).

[3] L. Gabriel. *O que é CRM e como ele otimiza o seu relacionamento com os seus clientes*. URL: https://rockcontent.com/blog/o-que-e-crm/. (accessed: 14.01.2020).

[4] M. A. Grandel. *Process Builder Vs Flows - Become the Ultimate Admin*. URL: https://www.salesforceben.com/process-builder-vs-flows-become-the-ultimate-admin/. (accessed: 23.10.2020).

[5] IntelliPaat. *What is Salesforce*. URL: https://intellipaat.com/blog/what-is-salesforce/. (accessed: 27.01.2020).

[6] Jade. *Visualforce Vs Lightning - Which to choose and When*. URL: https://www.forcetalks.com/blog/visualforce-vs-lightning-which-to-choose-and-when/. (accessed: 22.10.2020).

[7] Salesforce.com. *Choose the Right Automation Tool*. URL: https://trailhead.salesforce.com/content/learn/modules/business_process_automation/process_whichtool. (accessed: 10.02.2020).

[8] Salesforce.com. *Commerce Cloud*. URL: https://www.salesforce.com/br/products/commerce-cloud/overview/#. (accessed: 14.01.2020).

[9] Salesforce.com. *Create and Convert Leads*. URL: https://trailhead.salesforce.com/en/content/learn/modules/leads_opportunities_lightning_experience/create-and-convert-leads-lightning. (accessed: 10.02.2020).

[10] Salesforce.com. *Customer Relationship Management (CRM)*. URL: https://www.salesforce.com/ap/definition/crm/. (accessed: 27.01.2020).

[11] Salesforce.com. *Differences Between Apex Classes and Java Classes*. URL: https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_classes_java_diffs.htm. (accessed: 20.10.2020).

[12] Salesforce.com. *Einstein Analytics*. URL: https://www.salesforce.com/br/products/einstein-analytics/overview/. (accessed: 14.01.2020).

[13] Salesforce.com. *Introdução ao Visualforce*. URL: https://trailhead.salesforce.com/en/content/learn/modules/visualforce_fundamentals/visualforce_intro. (accessed: 28.01.2020).

[14] Salesforce.com. *O que é a Salesforce?* URL: https://www.salesforce.com/br/products/what-is-salesforce/. (accessed: 27.01.2020).

[15] Salesforce.com. *Por que usar a estrutura de Componente do Lightning?* URL: https://help.salesforce.com/articleView?id=aura_features.htm&type=5. (accessed: 22.10.2020).

[16] Salesforce.com. *Sales Cloud*. URL: https://www.salesforce.com/br/products/sales-cloud/overview//?d=70130000000ryrh. (accessed: 14.01.2020).

[17] Salesforce.com. *Service Cloud*. URL: https://www.salesforce.com/br/products/service-cloud/overview//?d=70130000000rysG. (accessed: 14.01.2020).

[18] Salesforce.com. *Understand Custom Standard Objects*. URL: https://trailhead.salesforce.com/en/content/learn/modules/data_modeling/objects_intro. (accessed: 28.01.2020).

[19] Salesforce.com. *Visualforce*. URL: https://help.salesforce.com/articleView?id=pages_about.htm&type=5. (accessed: 22.10.2020).

[20] Salesforce.com. *Visão geral do código do Apex*. URL: https://help.salesforce.com/articleView?id=code_about.htm&type=5. (accessed: 20.10.2020).

[21] Salesforce.com. *What is Apex?* URL: https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_intro_what_is_apex.htm. (accessed: 20.10.2020).

[22] Salesforce.com. *What is Salesforce Lightning?* URL: https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/intro_lightning.htm. (accessed: 28.01.2020).

[23] Salesforce.com. *Write SOQL Queries*. URL: https://trailhead.salesforce.com/en/content/learn/modules/apex_database/apex_database_soql. (accessed: 11.02.2020).

[24] Salesforce.com. *Write SOSL Queries*. URL: https://trailhead.salesforce.com/en/content/learn/modules/apex_database/apex_database_sosl. (accessed: 11.02.2020).

[25] worldit consulting services. *Quem somos*. URL: https://www.worldit.pt/pt/sobre-nos/quem-somos/. (accessed: 11.02.2020).

[26] worldit consulting services. *WorldIT - About us*. URL: https://www.linkedin.com/company/worldit. (accessed: 11.02.2020).

[27] R. Singh. *Componentes da Web Aura x Lightning: o que você precisa saber*. URL: https://springml.com/blog/aura-vs-lightning-web-components/. (accessed: 16.11.2020).

[28] worldit. *INFORMA - Business by data*. URL: https://appexchange.salesforce.com/appxListingDetail?listingId=a0N3A00000FvKw8UAF. (accessed: 29.01.2020).

APÊNDICE A

Desenho da Solução

Para realização da solução proposta, foi necessário alterar e acrescentar algumas configurações aos objetos padrão e aos objetos personalizados. De modo a explicitar essas alterações, segue-se a enumeração destas para cada objeto utilizado.

- ***Account/Lead***
  As entidades de *Account* e *Lead* foram enriquecidas quer na estrutura de modelo de dados, quer no que concerne aos detalhes da própria entidade.
  - Customização
    1. Criação de novos campos;
    2. Disponibilização de um botão para classificação manual;
    3. Configuração do *compact layout* com a informação dos campos mais relevantes;
    4. Disponibilização no painel principal de uma *tab* para a lista do histórico das classificações.

Nome do campo	Tipo	Obrigatório?	Leitura apenas?	Compact Layout
Average Classification	Number	-	Sim	Sim

Tabela A.1: Tabela dos campos adicionais nos objetos *Account/Lead*

  - Automatismo
    * Classificação isolada;
    * Classificação massiva;
    * Classificação automática (programada pelo utilizador).
  - Os *mockups* são os seguintes:

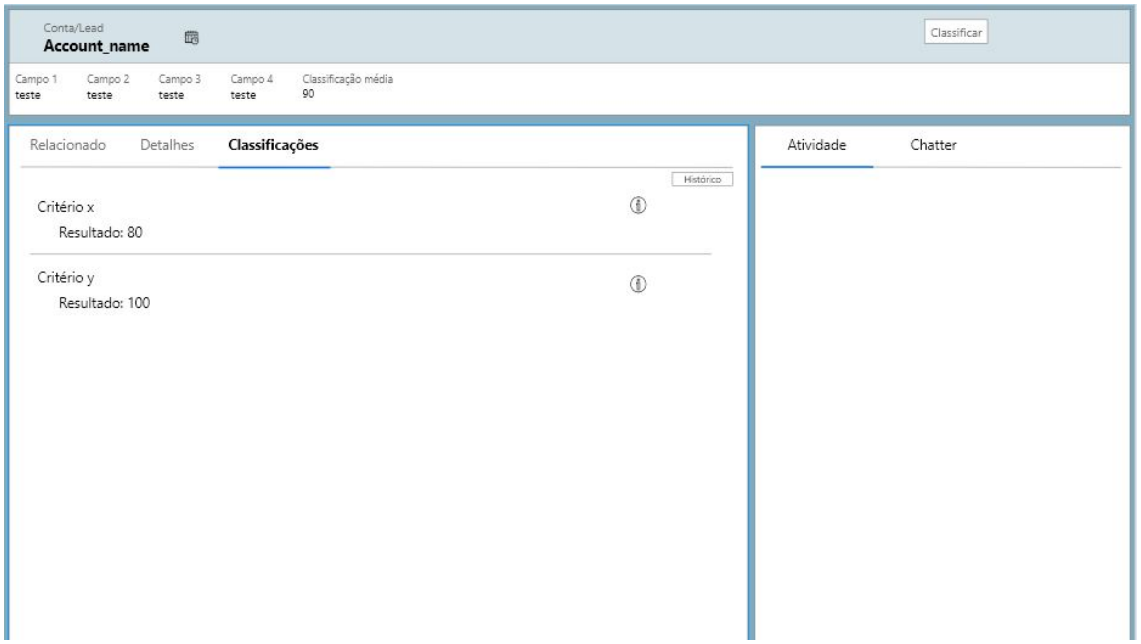


Figura A.1: Página de detalhes da *Account/Lead*

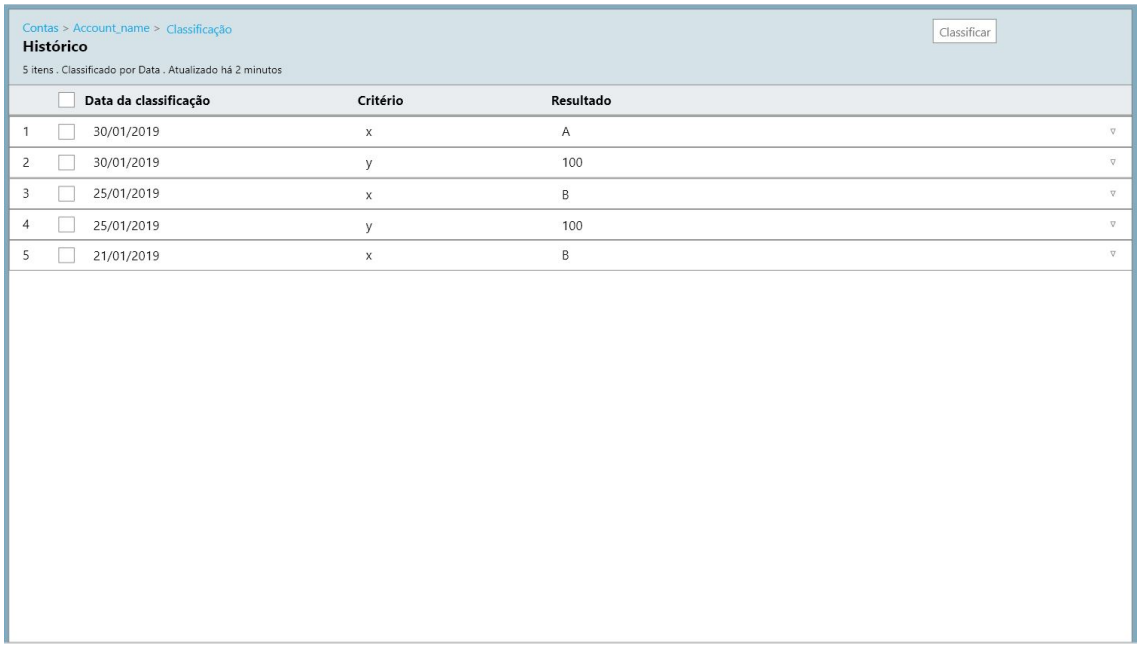


Figura A.2: Listagem do histórico das classificações do cliente

- **Account Classification**
  A entidade *Account Classification* foi criada com o objetivo de armazenar os dados de todas as classificações referentes a cada conta.
  Os registos são guardados automaticamente quando ocorre a classificação.

  – Customização

    1. Criação de novos campos.

Nome do campo	Tipo	Obrigatório?	Leitura apenas?	Compact Layout
Account	MasterDetails(Account)	-	Sim	-
Criteria	Text	-	Sim	-
Result	Number	-	Sim	-
Modification Date	Date/Time	-	Sim	-
Expression Label	Text	-	Sim	-

Tabela A.2: Tabela dos campos necessários no objeto *Account Classification*

- **Lead Classification**
  A entidade *Lead Classification* tem o mesmo funcionamento que a entidade mencionada anteriormente.
  - Customização
    1. Criação de novos campos.

Nome do campo	Tipo	Obrigatório?	Leitura apenas?	Compact Layout
Lead	Lookup(Lead)	-	Sim	-
Criteria	Text	-	Sim	-
Result	Number	-	Sim	-
Modification Date	Date/Time	-	Sim	-
Expression Label	Text	-	Sim	-

Tabela A.3: Tabela dos campos necessários no objeto *Lead Classification*

- **Criteria**
  A entidade *Criteria* é utilizada simplesmente para a criação dos critérios. Os restantes objetos que ligam com esta é que contêm toda a informação necessária.
  - Os mockups são os seguintes:
- **Logic Filter**
  A entidade *Logic Filter* guarda, tal como o nome indica, os filtros lógicos para cada critério e as classificações correspondentes.
  - Customização
    1. Criação de novos campos.

Nome do campo	Tipo	Obrigatório?	Leitura apenas?	Compact Layout
Criteria	Master/Details(Criteria)	-	Sim	-
Object	Text	-	Sim	-
Expression API	Text	-	Sim	-
Expression Label	Text	-	Sim	-
Result	Number	-	Sim	-

Tabela A.4: Tabela dos campos necessários no objeto *Logic Filter*

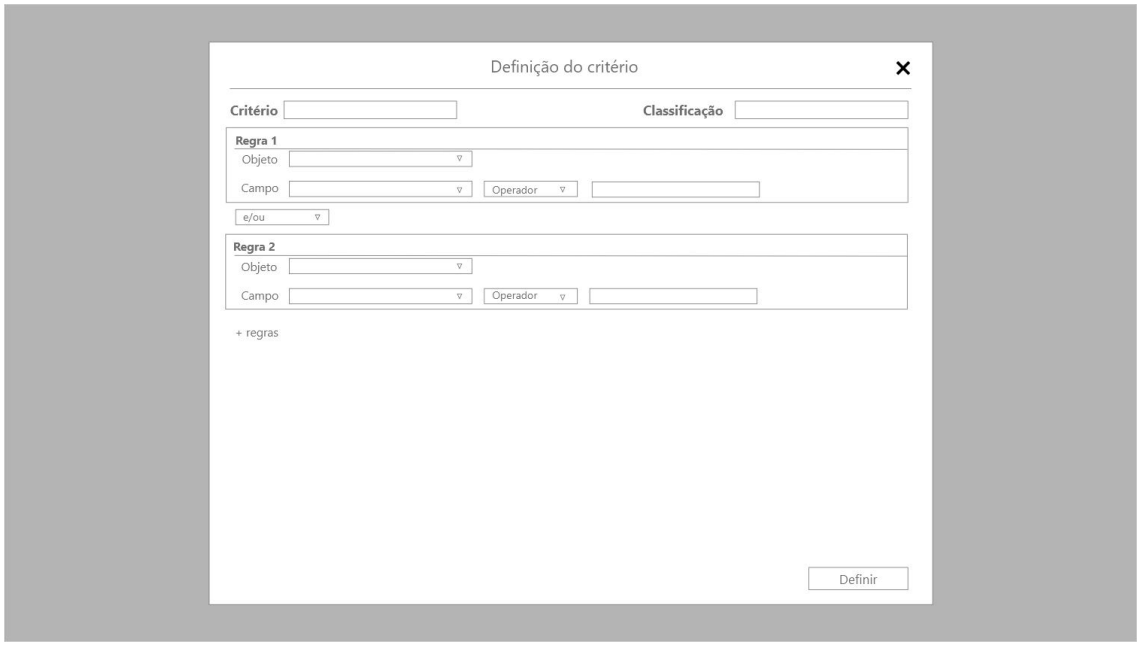
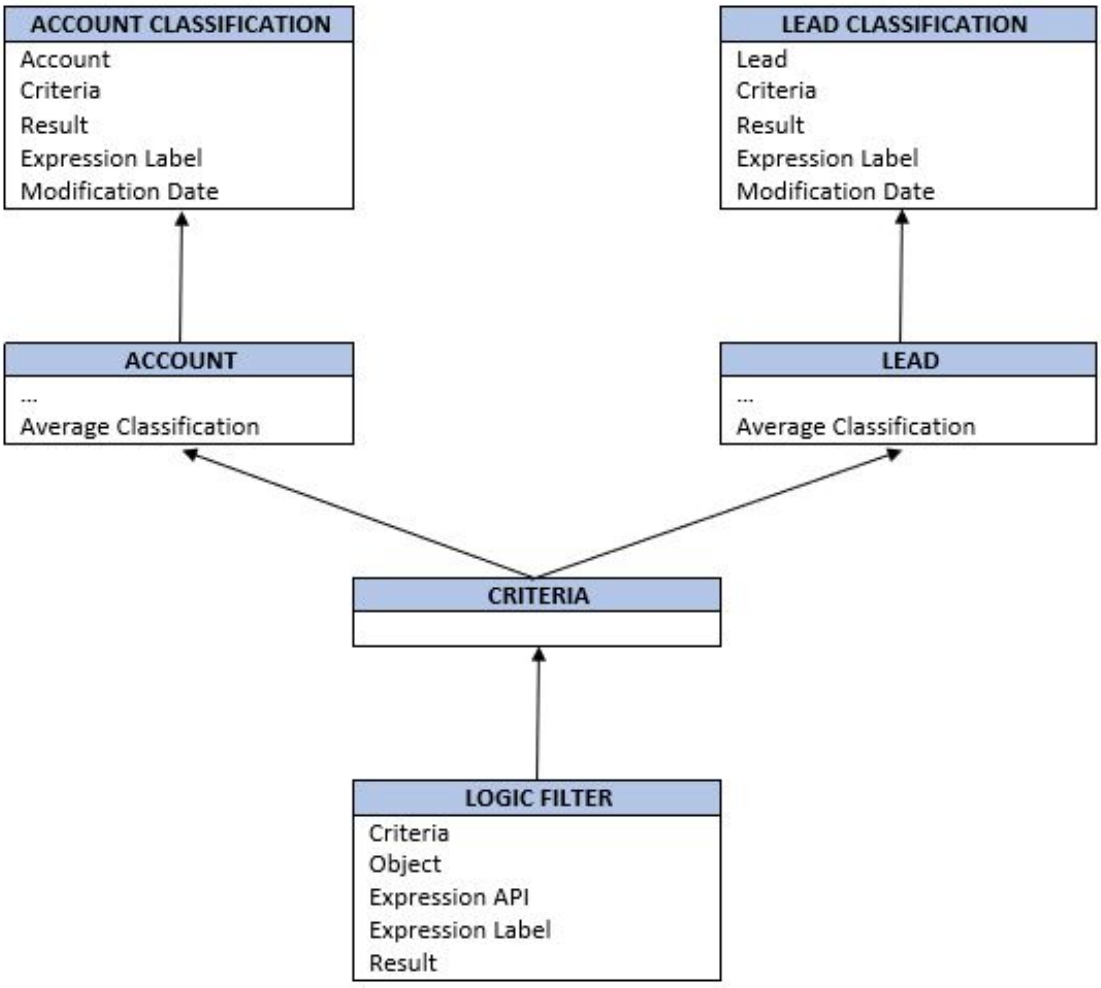


Figura A.3: Página de criação do critério

Figura A.4: Página de edição do critério

APÊNDICE B

DESENHO TÉCNICO



Apêndice C

Desenho de testes

1. Casos de uso

Caso de uso #	Teste
1	Criteria: Registar um critério para o objeto "Account"
2	Criteria: Registar um critério para o objeto "Lead"
3	Criteria: Editar um critério
4	Criteria: Adicionar mais um filtro lógico a um critério
5	Account: Verificar novo campo *Average Classification*
6	Account: Classificar manualmente uma conta
7	Account: Verificar as classificações recentes
8	Account: Ver o histórico de classificações da conta
9	Lead: Verificar novo campo *Average Classification*
10	Lead: Classificar manualmente uma *lead*
11	Lead: Verificar as classificações recentes
12	Lead: Ver o histórico de classificações da *lead*
13	Classificar todas as contas
14	Classificar todas as leads

Tabela C.1: Lista dos casos de uso

2. Detalhes

   a) Caso de uso 1 (Tabela C.2)

      - Desktop (Figura C.1)
      - Salesforce1 (Figura C.2)

   b) Caso de uso 2 (Tabela C.3)

      - Desktop (Figura C.3)
      - Salesforce1 (Figura C.4)

Objetivo:	Obter um critério para o objeto *Account*
Pressuposto & Condições:	
Passos:	i. Aceder à área de *Criterias* ii. Selecionar o botão "new" iii. Escolher o objeto "Account" iv. Preencher os outros campos e carregar "save"
Resposta(s) esperada(s):	i. Abrir uma nova página com o formulário de criação de um critério ii. Após salvar, reencaminho para a página de detalhes do critério criado
Razão para não executar:	
Notas e comentários:	É necessário ser preenchido pelo menos uma regra e pelo menos um filtro lógico

Tabela C.2: Detalhe caso de uso 1: Criteria: Registar um critério para o objeto "Account"

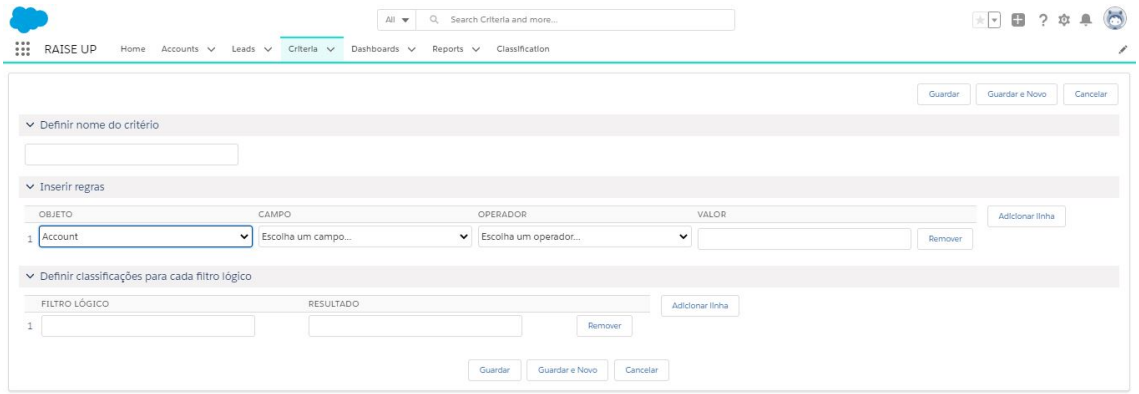


Figura C.1: Página de registo de um critério com o objeto *Account* selecionado

c) Caso de uso 3 (Tabela C.4)
   - Desktop (Figura C.5)

d) Caso de uso 4 (Tabela C.5)
   - Desktop (Figura C.6)

e) Caso de uso 5 (Tabela C.6
   - Desktop (Figura C.7)
   - Salesforce1 (Figura C.8)

f) Caso de uso 6 (Tabela C.7)
   - Desktop (Figura C.9)
   - Salesforce1 (Figura C.10)

Figura C.2: Página de registo de um critério com o objeto *Account* selecionado na aplicação móvel

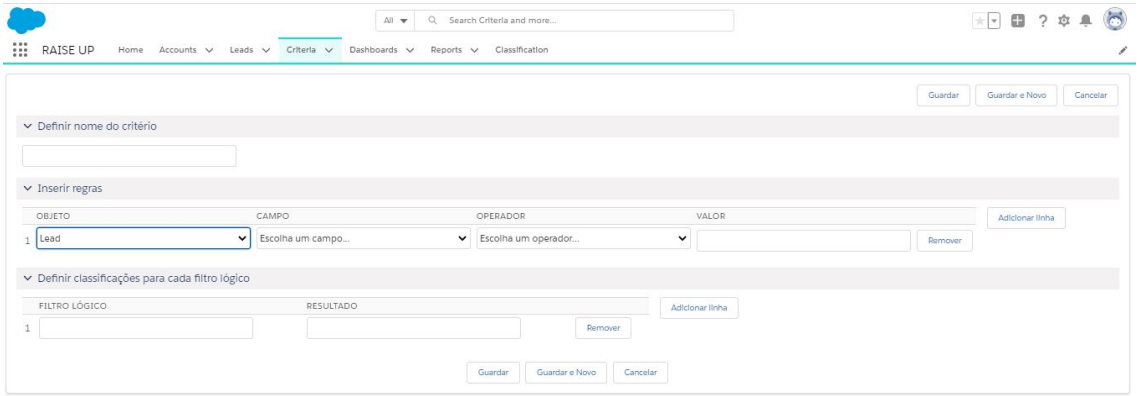


Figura C.3: Página de registo de um critério com o objeto *Lead* selecionado

g) Caso de uso 7 (Tabela C.8)
   - Desktop (Figura C.11)
   - Salesforce1 (Figura C.12)

h) Caso de uso 8 (Tabela C.9)
   - Desktop
     3) Aceder à área de classificações (Figura C.11)
     4) Ver histórico (Figura C.13)
   - Salesforce1 3) Aceder à área de classificações (Figura C.12) 4) Ver histórico (Figura C.14)

i) Caso de uso 9 (Tabela C.10)

Objetivo:	Obter um critério para o objeto *Lead*
Pressuposto & Condições:	
Passos:	i. Aceder à área de *Criterias* ii. Selecionar o botão "new" iii. Escolher o objeto "Lead" iv. Preencher os outros campos e carregar em "save"
Resposta(s) esperada(s):	i. Abrir uma nova página com o formulário de criação de um critério ii. Após salvar, reencaminho para a página de detalhes do critério criado
Razão para não executar:	
Notas e comentários:	É necessário ser preenchido pelo menos uma regra e pelo menos um filtro lógico

Tabela C.3: Detalhe caso de uso 2: Criteria: Registar um critério para o objeto "Lead"

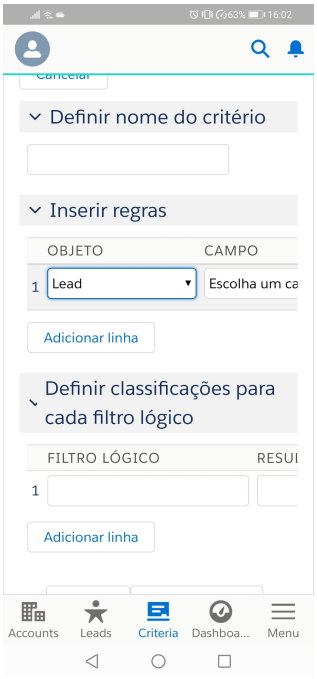


Figura C.4: Página de registo de um critério com o objeto *Lead* selecionado na aplicação móvel

- Desktop (Figura C.15)
- Salesforce1 (Figura C.16)

j) Caso de uso 10 (Tabela C.11)

- Desktop (Figura C.17)

Objetivo:	Editar um critério
Pressuposto & Condições:	Haver pelo menos um critério no sistema
Passos:	i. Aceder à área de *Criterias* ii. Selecionar um critério iii. Carregar no botão "Editar"
Resposta(s) esperada(s):	Abrir uma nova página com o formulário do critério com as informações do selecionado
Razão para não executar:	
Notas e comentários:	

Tabela C.4: Detalhe caso de uso 3: Criteria: Editar um critério

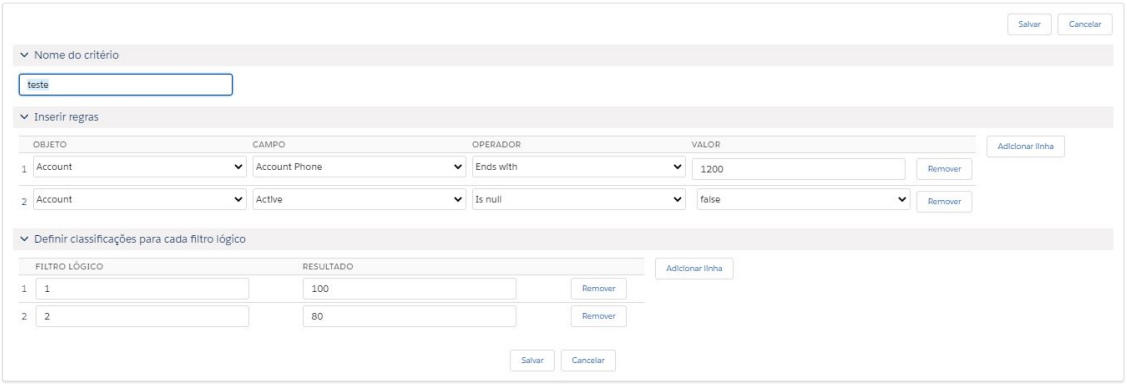


Figura C.5: Página de editar um critério

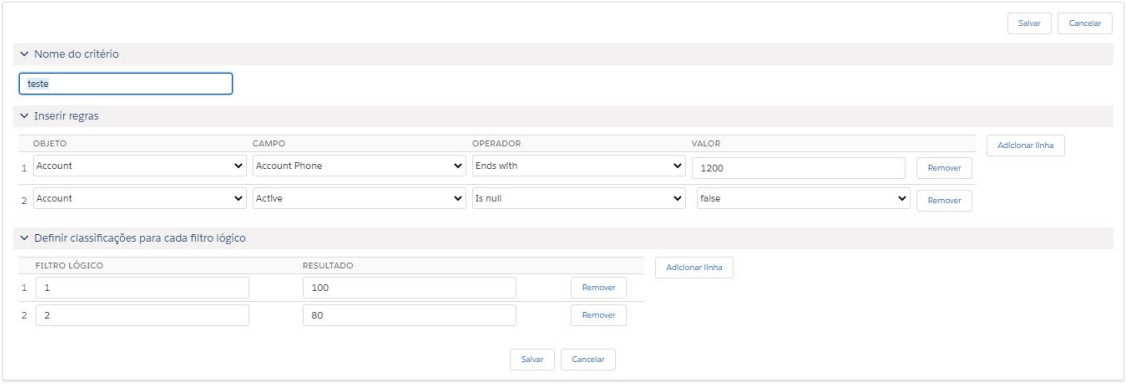


Figura C.6: Página de editar/adicionar filtro(s) lógico(s) num critério

- Salesforce1 (Figura C.18)

k) Caso de uso 11 (Tabela C.12)

- Desktop (Figura C.19)
- Salesforce1 (Figura C.20)

l) Caso de uso 12 (Tabela C.13)

Objetivo:	Adicionar mais um filtro lógico ao critério
Pressuposto & Condições:	Haver pelo menos um critério criado no sistema
Passos:	i. Aceder à área de *Criterias* ii. Selecionar um critério iii. Aceder à area de relacionados iv. Carregar no botão "Adicionar um filtro lógico" v. Na página selecionar "Adicionar linha"na secção do filtro lógico
Resposta(s) esperada(s):	Abrir uma nova página com o formulário do critério com as informações do selecionado
Razão para não executar:	
Notas e comentários:	

Tabela C.5: Detalhe caso de uso 4: Criteria: Adicionar mais um filtro lógico a um critério

Objetivo:	Verificar campo *Average Classification*
Pressuposto & Condições:	Haver pelo menos uma conta no sistema
Passos:	i. Aceder à área de *Accounts* ii. Selecionar uma conta iii. Verificar no compact layout a existência do campo
Resposta(s) esperada(s):	
Razão para não executar:	
Notas e comentários:	

Tabela C.6: Detalhe caso de uso 5: Account: Verificar novo campo *Average Classification*

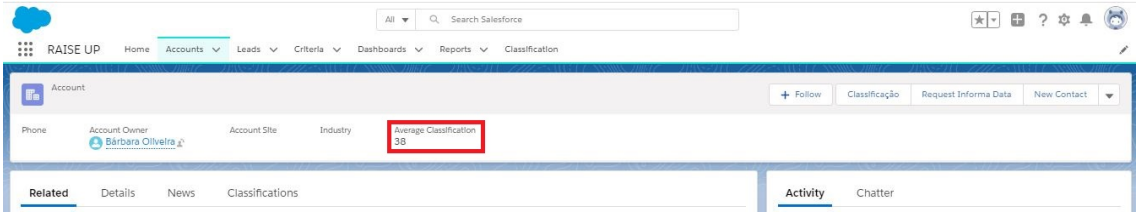


Figura C.7: Página de detalhes de uma conta

- Desktop
  3) Aceder à área de classificações (Figura C.19)
  4) Ver histórico (Figura C.21)
- Salesforce1 3) Aceder à área de classificações (FiguraC.20) 4) Ver histórico (Figura C.22)

m) Caso de uso 13 (Tabela C.14)

- Desktop (Figura C.23)

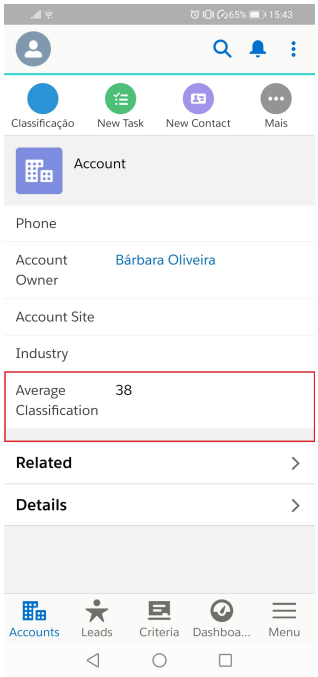


Figura C.8: Página de detalhes de uma conta na aplicação móvel

Objetivo:	Classificar uma conta
Pressuposto & Condições:	• Haver pelo menos uma conta no sistema • Haver pelo menos um critério no sistema
Passos:	i. Aceder à área de *Accounts* ii. Selecionar uma conta iii. Carregar no botão "Classificação"
Resposta(s) esperada(s):	Atualização do campo *Average Classification*
Razão para não executar:	
Notas e comentários:	O campo pode não ser atualizado se a conta não pertencer a nenhum dos critérios registados.

Tabela C.7: Detalhe caso de uso 6: Account: Classificar manualmente uma conta

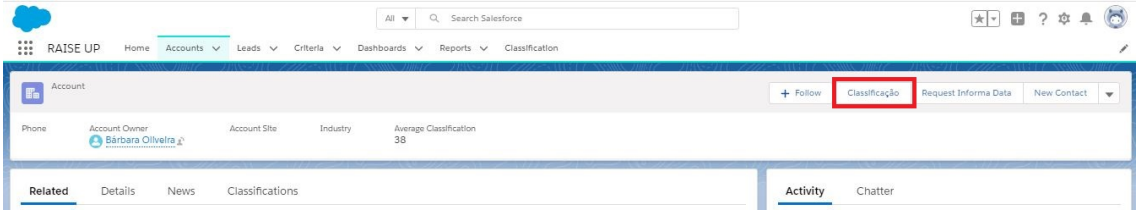


Figura C.9: Página de detalhes de uma conta

- Salesforce1 (Figura C.24)

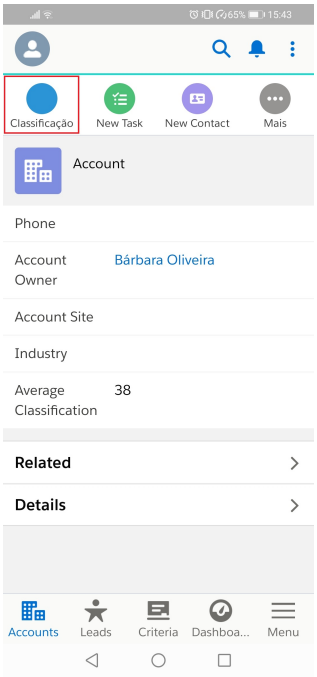


Figura C.10: Página de detalhes de uma conta na aplicação móvel

Objetivo:	Visualizar as classificação mais recentes
Pressuposto & Condições:	• Haver pelo menos uma conta no sistema • Haver pelo menos um critério no sistema
Passos:	i. Aceder à área de *Accounts* ii. Selecionar uma conta iii. Aceder à área de classificações
Resposta(s) esperada(s):	Lista com as classificações mais recentes para cada critério criado cujo objeto seja *Account*
Razão para não executar:	
Notas e comentários:	

Tabela C.8: Detalhe caso de uso 7: Account: Verificar as classificações recentes

n) Caso de uso 14 (Tabela C.15)

- Desktop (Figura C.25)
- Salesforce1 (Figura C.26)

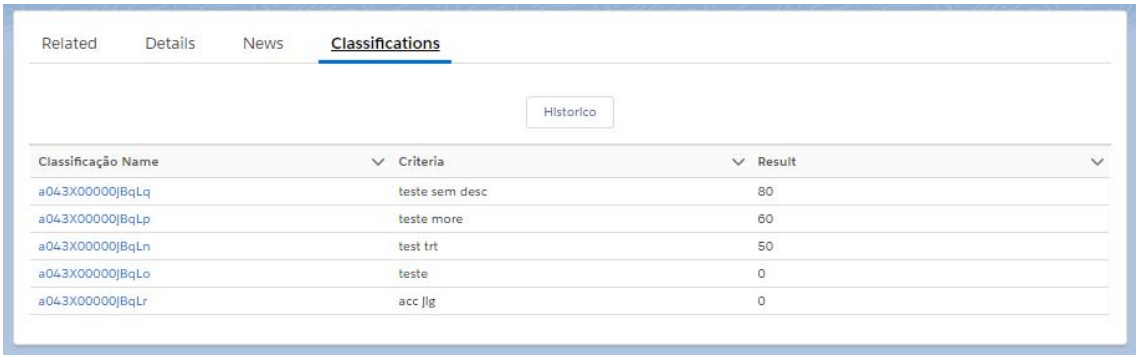


Figura C.11: Lista das classificações recentes de uma conta

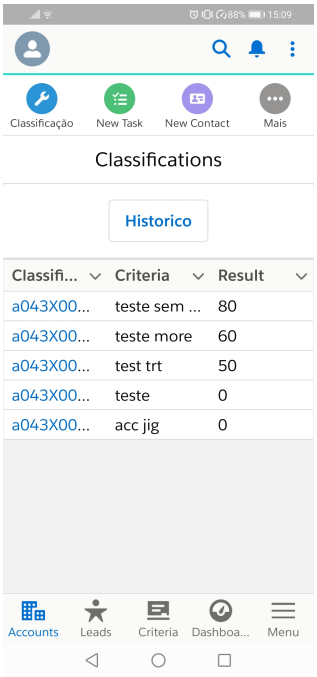


Figura C.12: Lista das classificações recentes de uma conta na aplicação móvel

Figura C.13: Lista com todas as classificações de uma conta

Objetivo:	Visualizar todas as classificações da conta
Pressuposto & Condições:	• Haver pelo menos uma conta no sistema • Haver pelo menos um critério no sistema
Passos:	i. Aceder à área de *Accounts* ii. Selecionar uma conta iii. Aceder à área de classificações iv. Carregar no botão "Histórico"
Resposta(s) esperada(s):	Lista com todas as classificações para cada critério criado cujo objeto seja *Account*
Razão para não executar:	
Notas e comentários:	

Tabela C.9: Detalhe caso de uso 8: Account: Ver o histórico de classificações de uma conta

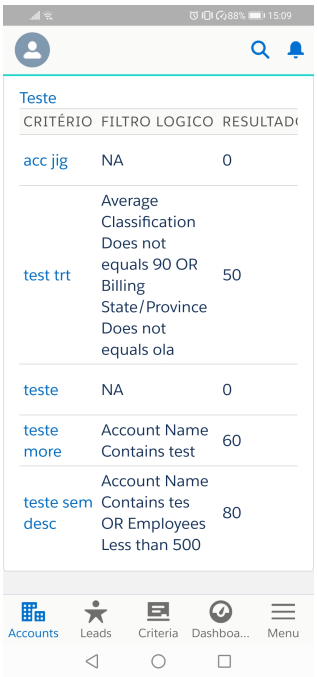


Figura C.14: Lista com todas as classificações de uma conta na aplicação móvel

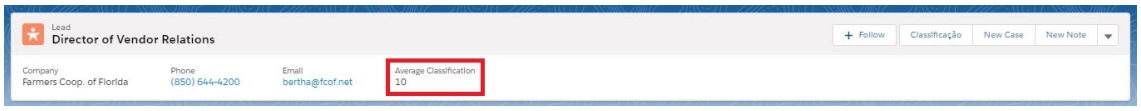


Figura C.15: Página de detalhes de uma lead

Objetivo:	Verificar campo *Average Classification*
Pressuposto & Condições:	Haver pelo menos uma *lead* no sistema
Passos:	i. Aceder à área de *Leads* ii. Selecionar uma *lead* iii. Verificar no compact layout a existência do campo
Resposta(s) esperada(s):	
Razão para não executar:	
Notas e comentários:	

Tabela C.10: Detalhe caso de uso 9: Lead: Verificar novo campo *Average Classification*

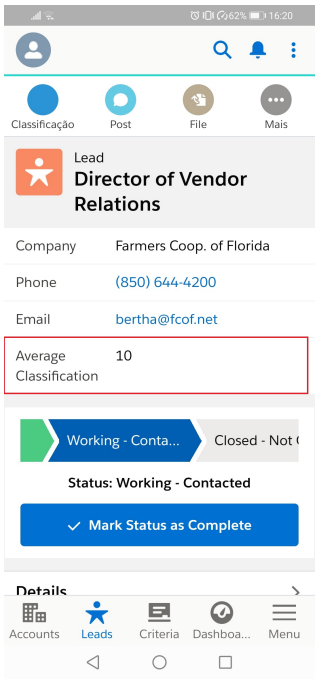


Figura C.16: Página de detalhes de uma lead na aplicação móvel

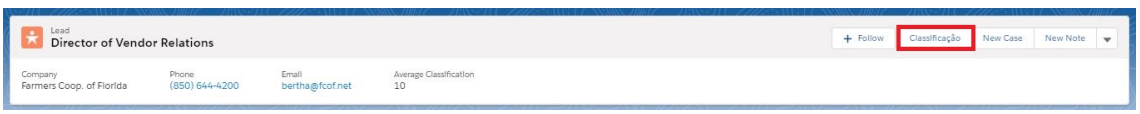


Figura C.17: Página de detalhes de uma lead

Objetivo:	Classificar uma lead
Pressuposto & Condições:	• Haver pelo menos uma *lead* no sistema • Haver pelo menos um critério no sistema
Passos:	i. Aceder à área de *Leads* ii. Selecionar uma lead iii. Carregar no botão "Classificação"
Resposta(s) esperada(s):	Atualização do campo *Average Classification*
Razão para não executar:	
Notas e comentários:	O campo pode não ser atualizado se a *lead* não pertencer a nenhum dos critérios registados.

Tabela C.11: Detalhe caso de uso 10: Lead: Classificar manualmente uma lead

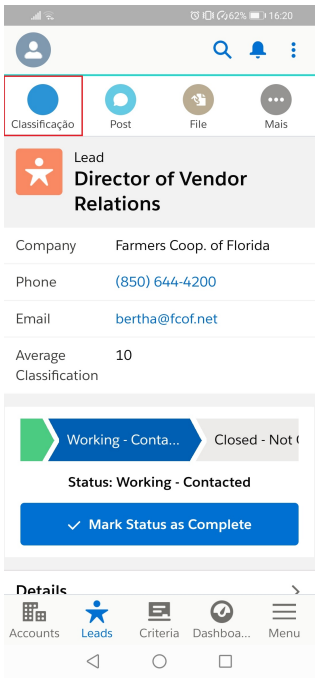


Figura C.18: Página de detalhes de uma lead na aplicação móvel

Objetivo:	Visualizar as classificação mais recente
Pressuposto & Condições:	• Haver pelo menos uma *lead* no sistema • Haver pelo menos um critério no sistema
Passos:	i. Aceder à área de *Leads* ii. Selecionar uma *lead* iii. Aceder à área de classificações
Resposta(s) esperada(s):	Lista com as classificações mais recentes para cada critério criado cujo objeto seja *Lead*
Razão para não executar:	
Notas e comentários:	

Tabela C.12: Detalhe caso de uso 11: Lead: Verificar as classificações recentes

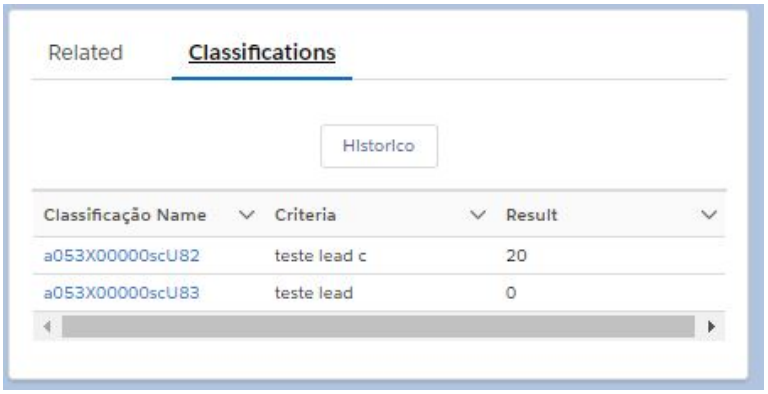


Figura C.19: Lista das classificações recentes de uma lead

Objetivo:	Visualizar todas as classificações da *lead*
Pressuposto & Condições:	• Haver pelo menos uma *lead* no sistema • Haver pelo menos um critério no sistema
Passos:	i. Aceder à área de *Leads* ii. Selecionar uma lead iii. Aceder à área de classificações iv. Carregar no botão "Histórico"
Resposta(s) esperada(s):	Lista com todas as classificações para cada critério criado cujo objeto seja *Lead*
Razão para não executar:	
Notas e comentários:	

Tabela C.13: Detalhe caso de uso 12: Lead: Ver o histórico de classificações de uma lead

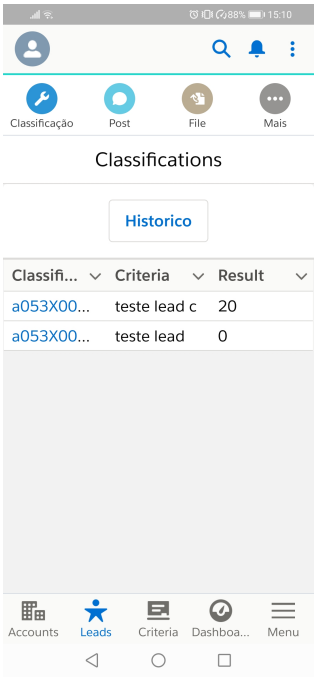


Figura C.20: Lista das classificações recentes de uma lead na aplicação móvel

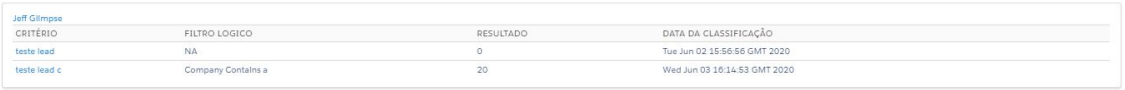


Figura C.21: Lista com todas as classificações de uma lead

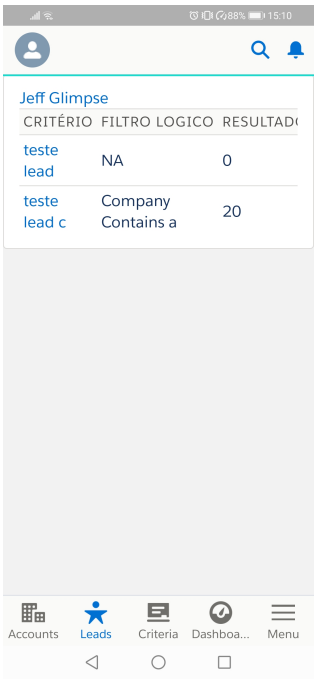


Figura C.22: Lista com todas as classificações de uma lead na aplicação móvel

Objetivo:	Classificação massiva das contas registadas no sistema
Pressuposto & Condições:	• Haver pelo menos uma conta no sistema • Haver pelo menos um critério no sistema
Passos:	i. Aceder à guia *Classification* ii. Carregar no botão "Classificar contas"
Resposta(s) esperada(s):	• Mensagem de sucesso • Receção de email quando o processo terminar
Razão para não executar:	
Notas e comentários:	

Tabela C.14: Detalhe caso de uso 13: Classificar todas as contas

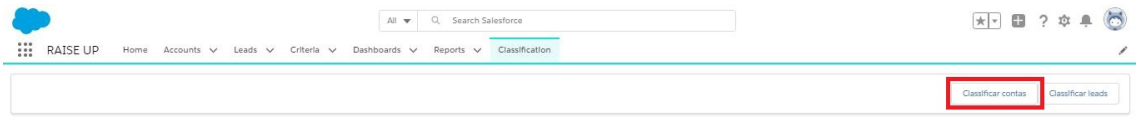


Figura C.23: Página de classificação massiva

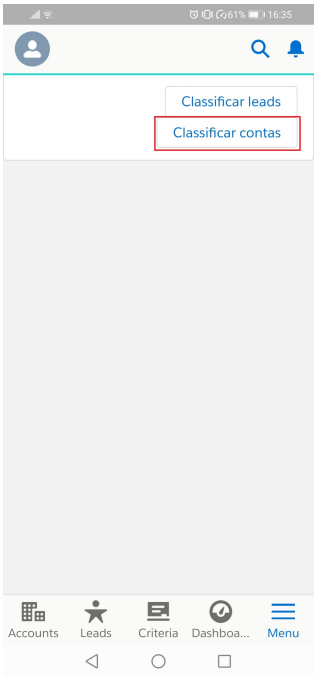


Figura C.24: Página de classificação massiva na aplicação móvel

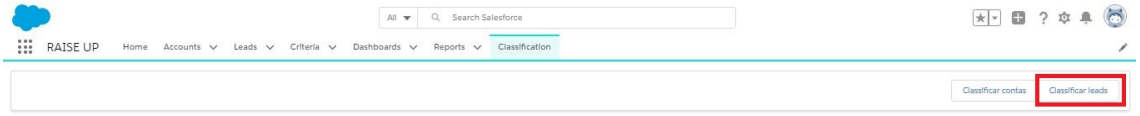


Figura C.25: Página de classificação massiva

Objetivo:	Classificação massiva das leads registadas no sistema
Pressuposto & Condições:	• Haver pelo menos uma *lead* no sistema • Haver pelo menos um critério no sistema
Passos:	i. Aceder à guia *Classification* ii. Carregar no botão "Classificar *leads*"
Resposta(s) esperada(s):	• Mensagem de sucesso • Receção de email quando o processo terminar
Razão para não executar:	
Notas e comentários:	

Tabela C.15: Detalhe caso de uso 14: Classificar todas as leads

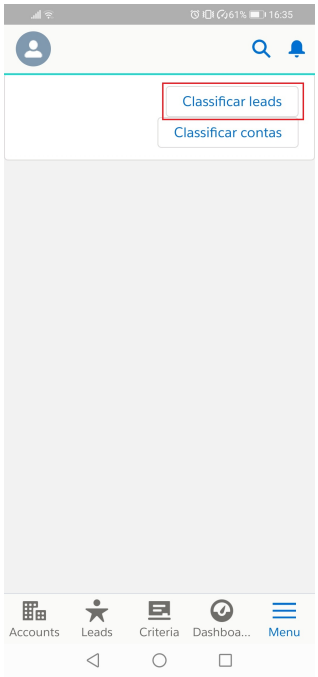


Figura C.26: Página de classificação massiva na aplicação móvel