

**CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA**  
**ESCOLA TÉCNICA ESTADUAL DA ZONA LESTE**  
**TÉCNICO EM INFORMÁTICA**

**SCRATCH OUT:**  
**Gerenciador de Tarefas**

**São Paulo**

**2018**

**Luis Augusto Lopes Silva**

**Luis Eduardo Lima Gonzaga**

**Paloma Rangel Rocha**

**Sarah de Souza Ribeiro Lucas**

## **SCRATCH OUT:**

### **Gerenciador de Tarefas**

Trabalho de Conclusão de Curso apresentado ao Técnico de Informática da Escola Técnica Estadual da Zona Leste, para a disciplina de Planejamento de Trabalho de Conclusão de Curso, administrada pelo Professor Rogério Bezerra Costa, como requisito final para obtenção do título de Técnico em Informática.

**São Paulo**

**2018**

## **AGRADECIMENTOS**

A

Prof. Rogério Bezerra Costa

Pela excelência no acompanhamento e orientação do projeto.

Prof. Wagner França

Pelo suporte e apoio.

Prof. Karen dos Reis Fernandes Teixeira

Pelo incentivo, apoio e divulgação do projeto.

Prof. Jeferson Roberto de Lima

Por expandir as oportunidades de divulgação e concretização do projeto.

## EPÍGRAFE

“Medir é importante: o que não é medido, não é gerenciado.” (KAPLAN; NORTON, 1997)

## RESUMO

Neste projeto é proposto demonstrar a influência da tecnologia mobile através de uma aplicação dedicada ao gerenciamento de tarefas e registro de influências, a fim de gerar gráficos indicadores estatísticos. Com estudo em *Unified Modeling Language* (UML), Banco de Dados, Engenharia de Software e React Native o aplicativo desenvolvido receberá os dados inseridos pelo usuário, e gerará gráficos intuitivos em prol de proporcionar o aumento da organização e da produtividade. O aplicativo permite a criação de tarefas, o registro de influências e formula gráficos estatísticos a partir dos dados inseridos pelo usuário. Se utilizado com frequência, o aplicativo pode fornecer ampla visão de comportamentos através das comparações entre diferentes resultados estatísticos, correlacionando influências e produtividade.

**PALAVRAS-CHAVE:** Aplicativo. Estatística. Gerenciamento. Tarefas. Produtividade.

## **ABSTRACT**

*In this project we propose to demonstrate the influence of mobile technology on the user, through an application dedicated to task management and recording of influences in order to generate statistical indicators. With Unified Modeling Language (UML), Database, Software Engineering and React Native study the developed application will receive the data entered by the user, and generate intuitive graphics to provide increased organization and productivity. The application allows the creation of tasks, record influences and formulate statistical graphs from the data entered by the user. If used frequently, the application can provide broad insight into behavior through comparisons between different statistical results correlating influences and productivity.*

**Keywords:** *Application. Statistic. Management. Tasks. Productivity.*

## LISTA DE ILUSTRAÇÕES

Figura 1 – Pocket.....	13
Figura 2 – OptimizeMe .....	14
Figura 3 – <i>Doctype</i> no HTML 4.....	16
Figura 4 – Estrutura Básica HTML .....	16
Figura 5 – Cadastro do Usuário, Código HTML .....	19
Figura 6 – Cadastro do Usuário, Código HTML (2).....	19
Figura 7 – Cadastro do Usuário, Página <i>Web</i> .....	20
Figura 8 – Opções da <i>Tag Select</i> Estado Civil.....	20
Figura 9 – Regra CSS.....	21
Figura 10 – Página HTML Estilizada .....	22
Figura 11 – Código CSS.....	22
Figura 12 – Caixa de Alerta.....	26
Figura 13 – Declaração de Variáveis no JavaScript .....	26
Figura 14 – Código da Função de Validação .....	27
Figura 15 – Caixa de Alerta da Função.....	27
Figura 16 – Sistemas Isolados .....	28
Figura 17 – Sistemas Integrados .....	28
Figura 18 – Tabela Alunos .....	29
Figura 19 – DER – Sistema de Pedido .....	30
Figura 20 – Cardinalidade Departamento Empregado .....	30
Figura 21 – Tabela Departamento e Empregado .....	31
Figura 22 – Tabela Cliente ÑN .....	32
Figura 23 – Tabela Cliente ÑN (2).....	32
Figura 24 – Tabela Clientes Normalizada na 1FN e Tabela Telefone .....	33
Figura 25 – Tabela Pedido ÑN na 2ºFN.....	33
Figura 26 – Tabela Pedido e Produto Normalizadas .....	34
Figura 27 - Tabela Pedidos Normalizada na 3FN.....	34
Figura 28 - Símbolos do Dicionário de Dados .....	35
Figura 29 - Exemplo de Dicionário de Dados.....	35
Figura 30 - Diagrama de Caso de Uso .....	41
Figura 31 - Associação de Generalização .....	42
Figura 32 - Diagrama de Atividades.....	43

Figura 33 - Classe.....	45
Figura 34 - Multiplicidade em Atributos .....	47
Figura 35 – Associação de Composição .....	48
Figura 36 - Generalização ou Especialização .....	49
Figura 37 - Dependência .....	49
Figura 38 - Diagrama de Sequência.....	51
Figura 39 - Entidade no Diagrama de Sequência .....	52
Figura 40 - Classe Carro .....	53
Figura 41 - Polimorfismo .....	54
Figura 42 - Herança .....	55
Figura 43 - Protótipo de Aplicativo Para Defensoria Pública .....	56
Figura 44 - Diagrama de Casos de Uso .....	58
Figura 45 - Diagrama de Atividade: Efetuar Login .....	59
Figura 46 - Diagrama de Atividades: Manter Influências .....	59
Figura 47 - Diagrama de Atividades: Manter Perfil.....	60
Figura 48 - Diagrama de Atividades: Manter Tarefa .....	60
Figura 49 - Diagrama de Classes .....	61
Figura 50 – Diagrama de Sequência: Cadastro .....	62
Figura 51 – Diagrama de Sequência: Indicador de Performance.....	62
Figura 52 – Diagrama de Sequência: Influências .....	63
Figura 53 - Diagrama de Sequência: Login.....	63
Figura 54 - Diagrama de Sequência: Tarefa.....	64
Figura 55 - DER.....	64
Figura 56 - Tela de <i>Splash</i> .....	65
Figura 57 - Tela Hall .....	65
Figura 58 - Tela de <i>Login</i> .....	66
Figura 59 - Home .....	66
Figura 60 - Lista de Tarefas.....	67
Figura 61 - Criação de Tarefas .....	67
Figura 62 - Tela de Inclusão de Detalhes e Edição de Tarefas.....	68
Figura 63 – Inserção de influências.....	68
Figura 64 – Inserção de influências (2) .....	69
Figura 65 - Humor.....	69



<b>Figura 66 – Gráfico de setores para frequência de humor .....</b>	<b>70</b>
<b>Figura 67 – Indicador de Performance .....</b>	<b>70</b>
<b>Figura 68 – Indicador de Performance (2).....</b>	<b>71</b>
<b>Figura 69 – Tela de Configurações .....</b>	<b>71</b>

## LISTA DE ABREVIATURAS E SIGLAS

Banco de Dados (BD)

*Cascading Style Sheets* (CSS)

Congresso Internacional de Tecnologia e Gestão (CITEG)

Dicionário de Dados (DD)

*General Public License* (GPL)

*Hypertext Markup Language* (HTML)

Instituto Brasileiro de Geografia e Estatística (IBGE)

Não Normalizada (ÑN)

Orientação a Objetos (OO)

*Pixels* (px)

Primeira Forma Normal (1FN)

*Python Software Foundation* (PSF)

*Red, Green, Blue* (RGB)

Segunda Forma Normal (2FN)

Terceira Forma Normal (3FN)

*Unified Modeling Language* (UML)

## SUMÁRIO

<b>1 INTRODUÇÃO .....</b>	<b>11</b>
<b>2 REFERENCIAL TEÓRICO .....</b>	<b>12</b>
2.1 Aplicativos Gerenciadores de Tarefas .....	12
2.2 Estatística .....	15
2.3 HTML .....	15
2.4 CSS.....	20
2.5 JavaScript.....	24
2.8 Banco de Dados .....	27
2.8.1 Abordagem Relacional .....	29
2.8.2 Normalização .....	31
2.8.3 Dicionário de Dados .....	34
2.9 UML .....	35
2.9.1 Levantamento de Análise e Requisitos .....	37
2.9.2 Diagrama de Casos de Uso.....	39
2.9.3 Diagrama de Atividades .....	42
2.9.4 Diagrama de Classes.....	44
2.9.5 Diagrama de Sequência .....	49
2.10 Paradigma de Orientação a Objetos.....	52
2.11 Engenharia de <i>Software</i> .....	55
<b>3 DESENVOLVIMENTO.....</b>	<b>58</b>
3.1 Diagrama de Casos de Uso .....	58
3.2 Diagrama de Atividades .....	58
3.3 Diagrama de Classes .....	60
3.4 Diagrama de Sequência.....	61
3.5 DER .....	64
3.6 Aplicação .....	65
<b>4 CONCLUSÃO .....</b>	<b>72</b>
<b>REFERÊNCIAS .....</b>	<b>73</b>

## 1 INTRODUÇÃO

No mundo contemporâneo, organizar as próprias pendências em detrimento do tempo tornou-se fundamental para a maior parte dos indivíduos. Sendo o aumento da produtividade e do aproveitamento do tempo desafios a serem encarados, quais seriam as possibilidades de aumentar a capacidade de gerenciamento e produção de tarefas nos âmbitos acadêmico e profissional, avaliando progressos obtidos por análises estatísticas?

No cenário tecnológico, é possível implementar ferramentas que auxiliem o autoconhecimento. Com esse propósito, a pesquisa tende a estimular novas pesquisas sobre estatística de uso, testando a efetividade de métodos e técnicas de produção, por meio de ferramentas tecnológicas, como um aplicativo.

A criação de um aplicativo que forneça ferramentas para possibilitar o controle de realização de tarefas e o aumento da capacidade de gerenciamento do usuário manifesta-se como um auxílio aplicável na gestão de tempo e tarefas.

Realizando pesquisas bibliográficas, pesquisas de campo, estudo de *Unified Modeling Language* (UML), Banco de Dados, Engenharia de *Software* e *React Native*, foi desenvolvido um aplicativo que receba dados informados pelo usuário, e que serão processados e retornados de forma gráfica e intuitiva, a fim de verificar seus dias de maior produtividade e suas influências seguindo os critérios de estatística.

A pesquisa ainda pretende abranger os tópicos de estatística e probabilidade, e verificar a aplicação de estatística em análise comportamental, possibilitando testes registrados de técnicas e métodos de estudo e produção, além de verificar aplicações que exercem funções semelhantes.

A tecnologia *React Native* se apresenta suficientemente capaz de suprir os objetivos da pesquisa, auxiliando a produção do aplicativo que se propõe ser um gerenciador de tarefas com recursos que se mostram eficientes para estimular a organização de seus usuários.

## 2 REFERENCIAL TEÓRICO

Este capítulo contém o embasamento teórico das tecnologias que serão utilizadas para a elaboração do projeto de pesquisa do aplicativo gerenciador de tarefas Scratch Out.

### 2.1 Aplicativos Gerenciadores de Tarefas

A globalização e a evolução dos meios de comunicação e transporte determinaram maior demanda de responsabilidade individual para se organizar em detrimento do tempo e da quantidade de afazeres que não podem ser esquecidos, ou que precisem de maior dedicação. A correria existente na vida de milhares de pessoas é um dos motivos que as fazem esquecer seus compromissos, tarefas, pensamentos e ideias. Segundo Lietti (2016), ficar desorganizado, em detrimento da quantidade de coisas para administrar e memorizar é uma situação comum.

Além de meios de pesquisa e comunicação, *smartphones* são um meio de obter organização pessoal. Para Munhoz (2015), essas são poderosas ferramentas tecnológicas que podem ser usadas para alcançar a produtividade, e como dito por Lietti (2016), os aplicativos podem aliviar um pouco o trabalho da memória, auxiliando a organização da rotina.

Dentre os aplicativos que disponibilizam recursos para a organização pessoal, o Wunderlist é um aplicativo que possibilita a criação de listas para diversos tipos de afazeres, desde eventos pessoais a profissionais. É um aplicativo gratuito e multiplataforma, disponível para iOS, Windows, Android e Kindle.

O aplicativo Evernote é ideal para o usuário que precisa ou se identifica com a organização através de notas. Para Estrella (2014), o Evernote é um banco de anotações. A plataforma possui recursos como criação de notas, *tags* para catálogo, inclusão de capturas de tela, fotos e documentos anexos. Também é possível sincronizar as notas criadas com outros dispositivos eletrônicos que possuem o aplicativo.

Uma plataforma próxima ao Evernote é o OneNote Microsoft, que também permite o acesso do usuário a suas anotações em diferentes dispositivos, tendo a opção de compartilhar o bloco de anotações com outros usuários. É multiplataforma, e para utilizá-lo é necessário possuir uma conta da Microsoft.

O Google Keep é uma plataforma para o registro de ideias e afazeres. É possível criar listas e registrar anotações, fotografias e até gravações.

O Trello é uma plataforma que oferece um serviço de gerenciamento de tarefas focado na gestão de projetos e equipes. Como dito por Munhoz (2015), o Trello é utilizado principalmente em ambientes corporativos e comerciais, mas também é útil para uso pessoal.

Concorrente direto do Trello, o Asana é um gerenciador de tarefas que também fornece o serviço focado na gestão de projetos e equipes. Ambas as plataformas oferecem recursos como definição de prazos, anexo de arquivos e comentários em cartões.

O Neotriad é uma plataforma brasileira desenvolvida pelo consultor em gestão de tempo Christian Barbosa. A plataforma fornece listagem de tarefas diárias e determinação de tempo dedicado a essas tarefas. Segundo Abrantes, a ideia do programa é permitir a visualização do uso do tempo do usuário em tarefas profissionais e pessoais.

Voltado para o incentivo e o auxílio a leitura, o Pocket é um aplicativo que permite salvar textos, notícias e artigos *offline* para a realização de leitura posteriormente. Como dito por Sulz (2017), é um aplicativo simples de usar.

**Figura 1 – Pocket**



Fonte: Autoria própria, 2018.

De acordo com Lietti (2016), o OptimizeMe é um aplicativo que propõe melhorar o bem-estar, como anuncia seu nome. A plataforma realiza um acompanhamento das atividades no cotidiano dos usuários. Segundo Brito (2014), esse acompanhamento é auxiliado por um rastreador de atividades. O aplicativo permite que o usuário informe como funciona sua rotina em quatro setores que são: saúde, prazer, rotina e criatividade. Sobre a saúde do usuário, é possível informar dados como peso, número de copos da água que o usuário bebeu ao longo do dia e tempo de atividade física. O aplicativo correlaciona esses dados elaborando gráficos e tabelas a respeito dos hábitos e bem-estar do usuário. As estatísticas começam a ser exibidas após 12 dias de uso e registros.

**Figura 2 – OptimizeMe**



Fonte: Brito (2014).

Segundo Lietti (2016), o Timeful é um aplicativo que possui algoritmos elaborados com base na ciência comportamental. Com o propósito de otimizar o tempo do usuário, o aplicativo busca compreender como funciona as necessidades do usuário e estabelecer níveis de prioridade para as tarefas do usuário.

Além dos aplicativos citados, existem diversos outros aplicativos com o objetivo de auxiliar a organização pessoal de seus usuários. Alguns são mais específicos, como o Google Agenda, que como dito por Sulz (2017) ajuda a organizar os compromissos marcando-os na agenda. Outros, como o Guia Bolso e o Organizze são conhecidos por destinarem-se à organização financeira.

Como dito por Munhoz (2015), a tecnologia existe no cotidiano como uma ajuda para a realização de diversas tarefas. Os aplicativos gerenciadores de tarefas buscam

aprimorar os métodos de administração e controle de quesitos pessoais e profissionais.

## **2.2 Estatística**

Estatística é a área da matemática, que tem por função estudar a coleta, análise e a interpretação de dados amostrais, pois é a partir destes que é possível definir os elementos sobre os dados que serão estudados e assim abstrair as variáveis do grupo ou do todo para assim obter as informações necessárias, que respondam os questionamentos propostos, facilitando conclusões e mensuração de ações ou valores estudados pela sociedade, diz Crespo (2012).

Afirma Fávero e Belfiore (2017) que na estatística, a amostragem se refere a seleção do grupo de pessoas ou horizonte de eventos, para representar o todo, dentro do universo estatístico, como é chamado, e desta forma identificar as similaridades e as diferenças. Por exemplo, é através da estatística que é possível identificar a classe social das pessoas, a etnia, o comportamento, distribuição de renda per capita, longevidade, entre outros tipos de dados, assim como acontece nos censos realizados pelo Instituto Brasileiro de Geografia e Estatística (IBGE). O mesmo princípio também é utilizado pela ferramenta *google forms*, que separa as reações dos usuários de acordo com as ações e escolhas deles e as classificam em grupos pré-estabelecidos.

Algumas funções das quais a estatística dispõe, como afirma Bussab e Morettin (2017), são: o cálculo das frequências, percentuais, moda, média, mediana, desvio padrão, probabilidade e outros afins; as mesmas permitem inúmeras possibilidades de aplicações da estatística em estudos ou ferramentas.

## **2.3 HTML**

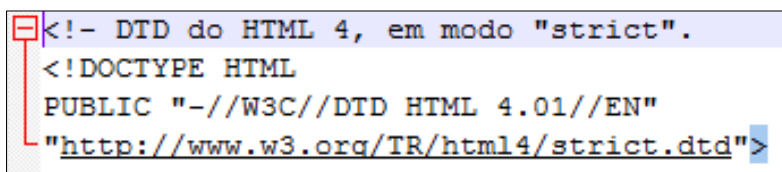
Segundo Ferreira e Eis (2011), *Hypertext Markup Language* (HTML), Linguagem de Marcação de Hipertexto, é uma das principais tecnologias para construir páginas da *web*, cuidando da estrutura das páginas. É utilizado para publicar conteúdos como texto, imagens, áudios e vídeos, formando um conjunto de elementos conectados, que é o conceito de hipertexto.



Essa conexão de elementos dá possibilidade aos assuntos de se envolverem de uma maneira organizada, gerando uma distribuição global de informações de fácil entendimento (FERREIRA; EIS, 2011).

Mazza (2012) defende que com a evolução do HTML, estando na versão 5, modificações significativas ocorreram. Existem novos elementos e funcionalidades que permitem melhores experiências. Para ele, uma das mudanças que fez mais diferença foi na sintaxe do código. Por exemplo, ao declarar *Doctype*, instrução de como o navegador deve processar o código, no HTML 4 ficaria:

**Figura 3 – Doctype no HTML 4**



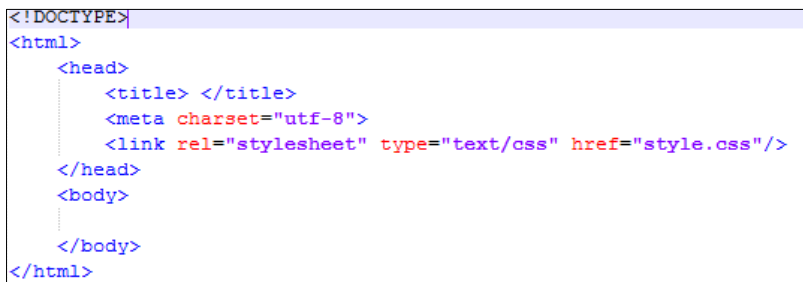
```
<!-- DTD do HTML 4, em modo "strict".
<!DOCTYPE HTML
PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

Fonte: Autoria própria, 2018.

Já no HTML 5 essas quatro linhas se resumem a `<!DOCTYPE html>`. Mazza (2012) ainda diz que outros comandos também encurtaram, como a *tag* de *link* e a de *script*. Outra mudança foram *tags* para substituir a `<div>`, que é utilizada para definir as estruturas da página e a divisão do conteúdo, assim o vício do programador em usá-la poderia diminuir. Como a *tag header* sendo específica para criações de cabeçalhos, títulos e introduções; a *footer* para o rodapé e a *article*, que identifica notícias recomendadas no final da que se está lendo ou para separar os comentários dos usuários.

A estrutura básica do HTML 5 pode ser montada de acordo com a figura 4:

**Figura 4 – Estrutura Básica HTML**



```
<!DOCTYPE>
<html>
  <head>
    <title> </title>
    <meta charset="utf-8">
    <link rel="stylesheet" type="text/css" href="style.css"/>
  </head>
  <body>
    ...
  </body>
</html>
```

Fonte: Autoria própria, 2018.

Conforme Ferreira e Eis (2011) essas *tags* têm o seguinte funcionamento e importância dentro do código:

- *Doctype*: deve estar na primeira linha do código. Indica ao navegador qual especificação de código que deve ser utilizada. Não é uma *tag*, mas sim uma instrução para o *browser*, o seu navegador, ter informações de qual versão de código a marcação foi escrita.
- *Html*: o código HTML é composto por vários elementos em árvore, onde uns são filhos dos outros. Nesta *tag* pode ser implementado a linguagem principal do documento a partir do atributo *lang*. Por exemplo: `<html lang="pt-br">`.
- *Head*: é onde se encontra a parte inteligente da página. Nesta ficam os metadados, que são informações sobre o conteúdo publicado e a página em si.
- *Title*: tag onde é escrito o título da sua página, o qual aparece na guia do seu navegador.
- *Meta charset*: é uma metatag encarregada por atribuir qual tabela de caracteres que a página estará utilizando.
- *Link*: existem dois tipos no HTML, um em que leva o usuário para outro documento e um para fontes externas usadas no documento. Neste exemplo a *tag link* importa o *Cascading Style Sheets* (CSS) para a página, o que indica esta importação é o atributo *rel="stylesheet"*.
- *Body*: esta última indica o corpo da sua página. Todo o conteúdo da sua página deverá ficar dentro desta *tag*.

Uma das principais finalidades do HTML 5 é facilitar a manipulação dos elementos, de maneira que o desenvolvedor modifique características dos objetos de uma forma clara e objetiva para o usuário final. Para que isso aconteça, ele fornece ferramentas para o CSS e JavaScript fazerem alterações, deixando a página mais interativa (FERREIRA; EIS, 2011).

O HTML faz uma comunicação com o *browser* para este fazer outra com o servidor. Ao acessar alguma página na internet e clicar em um *link*, o *browser* faz uma solicitação desta nova página ao servidor, que lhe envia a página e a exibe em sua janela. O HTML é responsável para dizer ao *browser* como vai ser o conteúdo e a estrutura da página. No momento em que o *browser* lê o seu código HTML ele

interpreta todas as *tags*, que informam a estrutura e o significado do seu texto (FREEMAN; FREEMAN, 2008).

Um par de *tag*, com abertura e fechamento, também pode ser denominado elemento, diz Silva (2015) que explica mais algumas *tags* do HTML:

- *Br*: elemento vazio, ou seja, somente com tag de abertura, destinado a quebra de linha.
- *P*: do tipo nível de bloco, é um elemento destinado a marcar parágrafos.
- *B* e *strong*: elementos do tipo *inline*, causam efeito de renderização em negrito. A diferença é que o elemento *b* dá o aspecto visual negrito e o elemento *strong* dá uma forte ênfase.
- *Div*: é um elemento destinado a criar um container geral para os outros elementos, também do tipo nível de bloco.
- *Form*: elemento que informa a criação do formulário.
- *Fieldset*: cria uma borda ao redor do formulário
- *Legend*: legenda do *fieldset*, um título entre a borda.
- *Input*: controla o que será adicionado no formulário, cria campos para entrada de dados, juntamente com o atributo "*type*".
- *Label*: legenda para um item do formulário, identificação dos campos.
- *Select*: *tag* para criação de uma lista com opções predefinidas
- *Option*: *tag* interna da *select*, cria os itens da lista de opções.

Para exemplificar o uso dessas *tags* e a criação de uma página da internet com HTML, seguem as figuras 5, 6 e 7. A figura 8 ilustra como ficam as opções do *select* de estado civil.

**Figura 5 – Cadastro do Usuário, Código HTML**

```
<!DOCTYPE>
<html>
<head>
<title> </title>
<meta charset="utf-8">
<link rel="stylesheet" type="text/css" href="style.css"/>
</head>
<body>

<div class="corpo">
<p> <b> Cadastro de Usuário </b> <p>
<form id="cad" action="">
<fieldset>
<legend> <b> Dados Pessoais </b> </legend>

<label for="nome" title="Digite o seu nome completo">
Nome:
</label>
<input type="text" maxlength="80" id="nome" size="50" /> <br /> <br />

<label for="datanasc">
Data de Nascimento:
</label>
<input type="date" id="datanasc">

<label for="sexo">
Sexo:
</label>
<input type="radio" name="sexo" id="feminino" value="f" /> Feminino
<input type="radio" name="sexo" id="masculino" value="m" /> Masculino <br /> <br />

<label for="estocivil">
Estado Civil:
</label>
```

Fonte: Autoria própria, 2018.

**Figura 6 – Cadastro do Usuário, Código HTML (2)**

```
<select id="estocivil" size="1">
<option value="0"> Selecione </option>
<option value="1"> Solteiro </option>
<option value="2"> Casado </option>
<option value="3"> Viúvo </option>
<option value="4"> Separado </option>
<option value="5"> Divorciado </option>
</select>

<label for="numcpf" title="Digite o seu CPF">
CPF:
</label>
<input type="text" maxlength="20" id="numcpf" name="cpf" size="20" /> <br /> <br />

<label for="email">
E-mail:
</label>
<input type="text" maxlength="80" id="email" size="50"/> <br /> <br />

<label for="cel">
Telefone Celular:
</label>
<input type="text" id="cel" size="2" />
<input type="text" id="cel2" size="10" /> <br /> <br />
<input type="submit" value="Enviar">
<input type="reset" value="Limpar">

</fieldset> <br />
</form>
</div>
</body>
</html>
```

Fonte: Autoria própria, 2018.

**Figura 7 – Cadastro do Usuário, Página Web**

**Cadastro de Usuário**

**Dados Pessoais**

Nome:

Data de Nascimento:  Sexo: ☐ Feminino ☐ Masculino

Estado Civil:  CPF:

E-mail:

Telefone Celular:

Fonte: Autoria própria, 2018.

**Figura 8 – Opções da Tag Select Estado Civil**

**Cadastro de Usuário**

**Dados Pessoais**

Nome:

Data de Nascimento:  Sexo: ☐ Feminino ☐ Masculino

Estado Civil:  CPF:

E-mail:

Telefone Celu:

Selecione

- Selecione
- Solteiro
- Casado
- Viúvo
- Separado
- Divorciado

Fonte: Autoria própria, 2018.

## 2.4 CSS

*Cascading Style Sheet (CSS)*, termo em inglês para Folha de Estilo em Cascata, é um mecanismo para adicionar estilo à página HTML de uma forma otimizada, dando acesso ao controle sobre o *layout* e o *design*. Com ele é possível estilizar títulos, tipos de fontes, bordas, espaçamentos, organizar espaços, criar animações, destacar links, entre outras coisas (MCFARLAND, 2015).

O CSS está em sua terceira versão, em que o CSS 1 estabeleceu a estrutura básica e o conceito de seletor, já o CSS 2 adicionou novas funcionalidades e seletores e a possibilidade de posicionar os elementos precisamente. O CSS 3 não tem nenhum padrão, nenhuma diferença marcante para a sua versão, ele é um conjunto de

diferentes módulos, em fases de finalizações diferentes, onde no futuro não haverá um CSS 4 e sim diferentes versões dos módulos, explica McFarland (2015).

Silva (2015) conclui que HTML é uma linguagem de estruturação e marcação de conteúdos e que não é responsável por fornecer informações sobre a apresentação de elementos, como tamanho de texto, cores das fontes e todo aspecto visual, o CSS que fica encarregado por estas formatações.

A formatação envolve um estilo, uma regra para descrever a aparência de uma parte específica, que pode ser aplicado a textos, imagens, títulos ou qualquer outro elemento da página, complementa McFarland (2015).

A regra CSS é a unidade básica de uma folha de estilo. Onde o termo unidade básica significa “a menor porção de código capaz de produzir efeito de estilização”, conforme Silva (2015, p. 70). A regra é composta por duas partes: seletor e declaração, e a declaração é composta por propriedade e valor. Silva (2015) ainda explica a definição de cada componente:

- Seletor: elemento da marcação HTML onde será aplicada a regra CSS.
- Declaração: contém a propriedade e o valor, define os parâmetros de estilização.
- Propriedade: define a característica do seletor que será estilizada.
- Valor: qualificação ou quantificação da estilização da propriedade.

Quando a regra conter mais de uma declaração, elas devem ser separadas por ponto e vírgula.

A figura 9 mostra como fica a sintaxe de uma regra CSS.

**Figura 9 – Regra CSS**

```
seletor {  
    propriedade_1: valor_1;  
    propriedade_2: valor_2;  
}
```

Fonte: Autoria própria, 2018.

Por meio da regra CSS é possível controlar os elementos HTML, adicionando estilo a eles. A regra pode ser usada externamente, colocando a *tag* `<link>` que informará ao navegador onde localizar o arquivo CSS que foi utilizado, ou internamente, colocando-a dentro do código HTML por meio do elemento `<style>`, que geralmente

fica dentro do elemento `<head>`. Usar a regra em uma folha de estilo externa traz mais vantagens, como a diminuição do código. Se várias páginas da web precisarem do mesmo estilo, não haverá repetição de código ao compartilhar a mesma folha de estilo (DUCKETT, 2016).

Utilizando as regras CSS, a figura 10 mostra como ficou a página HTML, figura 7, ao ser estilizada. E a figura 11 mostra o código CSS, com algumas propriedades e como os seus valores são aplicados.

**Figura 10 – Página HTML Estilizada**

Fonte: Autoria própria, 2018.

**Figura 11 – Código CSS**

```
body {
    color: white;
    font-family: Lao UI;
    font-size: 12pt;
    background: url('imgback2.jpg');
    background-repeat: no-repeat;
    background-size: cover;
}

.corpo {
    height: 100%;
    width: 54%;
    margin-left: 23%;
    margin-right: 23%;
    margin-top: 5%;
}

p {
    font-size: 14pt;
    text-align: center;
    text-transform: uppercase;
}

label, input[type=submit] {
    margin-left: 5%;
    margin-bottom: 4%;
}

form {
    padding: 5% 5%;
}

input[type=submit], input[type=reset] {
    margin-top: 10px;
    padding: 10px 22px;
    margin-left: 30px;
    background: White;
    color: Black;
    border: none;
    cursor: pointer;
    font-weight: 600;
}
```

Fonte: Autoria própria, 2018.

Duckett (2016) explica o conceito das propriedades utilizadas:

- *Color*: propriedade que permite a especificação da cor do texto do elemento. Podendo ser definida em valores *Red*, *Green*, *Blue* (RGB), códigos hexadecimais ou nome das cores.
- *Font-family*: especifica a fonte que será usada para qualquer texto dentro do elemento. Ao ser definida dentro do elemento *body*, o texto de toda a página ficará com esta fonte.
- *Font-size*: define o tamanho da fonte do texto, podendo ser definida por pontos, *pixels* (px) ou porcentagens, que são as maneiras mais comuns.
- *Font-weight*: deixa o texto do elemento em negrito.
- *Text-transform*: utilizado para alterar o texto em letras minúsculas ou maiúsculas. Podendo ser *uppercase*, que transforma em letras maiúsculas, *lowercase*, que transforma em minúsculas ou *capitalize*, que faz com que a primeira letra de cada palavra fique maiúscula.
- *Text-align*: define o alinhamento do texto. Podendo ser *left*, *right*, *center* ou *justify*, que deixa, respectivamente, alinhado à esquerda, direita, centralizado e justificado, onde todas as linhas do parágrafo, com exceção da última, ocupam toda a largura da caixa.
- *Width* e *height*: define a largura e altura da caixa. Podendo ser definida por porcentagem ou *pixels*.
- *Border*: permite especificar a largura, estilo e cor de uma borda, tudo na mesma propriedade. Os valores devem estar nesta ordem.
- *Padding*: especifica o tamanho do espaço que aparecerá entre a borda e o conteúdo do elemento da caixa. Pode-se especificar o lado da caixa na propriedade, como *padding-top*, *padding-right*, *padding-bottom* e *padding-left* para, respectivamente, o espaçamento na parte de cima, à direita, na parte de baixo e à esquerda. Também é possível colocá-los de forma abreviada, por exemplo: *padding*: 10px 15px 12px 14px. Representado pelos valores em sentido horário, sendo a mesma ordem em que as posições foram citadas. Outra forma de abreviar é quando os valores da esquerda e direita forem iguais e os de cima e de baixo também, resultando em: *padding*: 10px 5px.
- *Margin*: define o espaçamento entre as caixas. Se uma sobrepor a outra, a margem maior será usada e a menor desconsiderada. Do mesmo jeito que o *padding*, pode-se especificar o lado do *margin* e usá-lo de maneira abreviada.



- *Cursor*: esta propriedade define o tipo de cursor do mouse que será exibido aos usuários.
- *Background*: abreviação para todos os tipos de *background*, configura o fundo do seu elemento. Deve ser definido de acordo com a seguinte ordem: *background-color*, que determina a cor do fundo; *background-image*, que posiciona uma imagem atrás de qualquer elemento ao se colocar o caminho da imagem entre parênteses e aspas; *background-repeat*, que repete a imagem colocada no fundo, vertical e horizontalmente; *background-attachment*, que determina se a imagem do fundo ficará fixa ou irá se mover quando o usuário rolar a página; e *background-position*, para quando a imagem não se repetir, é utilizado para indicar ao navegador onde a imagem será posicionada. *Background-size* é utilizado para definir a dimensão da imagem do fundo.

## 2.5 JavaScript

Criada pela Netscape e em parceria com a Sun Microsystems, JavaScript tem a finalidade de fornecer interatividade a uma página da web, diz Silva (2010). A primeira versão, JavaScript 1.0, foi lançada em 1995. Em março de 1996, foi implementada no navegador Netscape Navigator 2.0, quando a Netscape dominava o mercado. Em seguida, com a guerra dos *browsers*, a Microsoft criou uma linguagem chamada JScript, em que sua primeira versão foi lançada no navegador Internet Explorer 3.0.

Silva (2010) ainda complementa que na criação de um formulário em HTML, há uma limitação de somente criar rótulos e campos para serem preenchidos, não tem a possibilidade de processar os dados ou enviá-los ao servidor. Para que isso possa acontecer são utilizadas linguagens como Java, PHP, Python, entre outras. Porém foram desenvolvidas para rodar no lado do servidor e JavaScript roda no lado do cliente, onde depende das funcionalidades hospedadas no navegador do usuário para que haja o funcionamento e interpretação da linguagem. Isso é possível por existir um interpretador de JavaScript no navegador.

Oliviero (2001) define JavaScript como uma linguagem de programação usada junto com HTML, onde são escritos pequenos programas, também conhecidos como

*scripts*, e inseridos no código HTML, permitindo que as páginas se tornem dinâmicas, gerando uma interatividade com o usuário.

Alguns exemplos de interatividade que Oliviero (2001) traz são:

- Mensagens de alerta por meio de uma nova janela ou de uma caixa de alerta;
- Validar os campos de um formulário;
- Mudar uma imagem assim que o mouse passar por ela;
- Manipular datas;
- E detectar se um plug-in está instalado.

Ele também é capaz de manipular conteúdo e apresentações, definindo, controlando e alterando a apresentação de um HTML, como aspectos de cor do fundo, cor de textos e de *links*, e até modificar a posição dos elementos. Além disso consegue criar regras CSS ou anular alguma regra existente (SILVA, 2010).

Outras funcionalidades que o JavaScript possui é de poder controlar alguns aspectos e funções do navegador, como apresentar mensagens para o usuário, criar janelas *pop-up*, retirar menus, abrir e fechar janelas, interferir na barra de *status*; interação com formulários, acessando campos e valores digitados pelos usuários, procedendo com uma validação dos dados, realização de cálculos ou aviso de campos não preenchidos; e interação com outras linguagens, acrescenta Silva (2010).

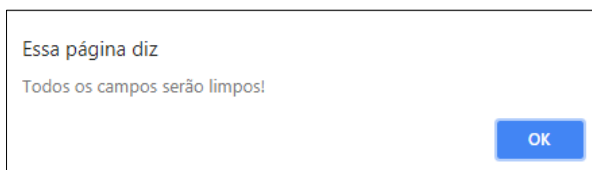
É preciso informar ao navegador que será usado o JavaScript por meio da *tag* `<script>`, colocando como atributo o tipo de linguagem que será utilizada, e então a *tag* de abertura ficaria `<script type="text/javascript">`. Ela pode ser colocada em qualquer lugar do código HTML, mas geralmente fica dentro do elemento `<head>` (MORRISON, 2008).

Silva (2010) explica que existem três tipos de caixas de diálogo, que são janelas *pop-up* destinadas a fornecer ou coletar informações do usuário. Há a caixa de alerta, que exibe na tela do usuário uma mensagem ou um aviso; a caixa de diálogo de confirmação, que exibe dois botões para confirmar ou cancelar uma ação do usuário; e a caixa de diálogo para entrada de *string*, onde aparece um campo para receber uma *string* digitada pelo usuário.

Outra funcionalidade do JavaScript são os eventos, ações que impulsionam uma reação, por exemplo, clicar em um *link* desencadearia a reação de abrir uma nova página. Os eventos viabilizam a interatividade na página da *web*, sem eles não seria possível fazer os *scripts* funcionarem (SILVA, 2010).

A figura 12 mostra o funcionamento de uma caixa de diálogo e de um evento. Ao se clicar no botão limpar, figura 10, aparece uma caixa de alerta.

**Figura 12 – Caixa de Alerta**



Fonte: Autoria própria, 2018.

No JavaScript há uma pequena mudança na sintaxe de declaração das variáveis. Onde não é necessário dizer o tipo dela, pois a linguagem reconhece o tipo de dado no momento em que é declarada, diz Morrison (2008). É importante destacar o escopo da variável, ou seja, o trecho do *script* em que ela irá assumir o valor atribuído.

Silva (2010) complementa dizendo que existe o escopo global e o local. No escopo global, a variável é reconhecida em qualquer trecho do *script*. Já no local, a variável é reconhecida somente onde ela foi declarada.

Para declarar a variável local é usada a palavra-chave *var*. A figura 13 apresenta a sintaxe da declaração de uma variável local e de uma variável global.

**Figura 13 – Declaração de Variáveis no JavaScript**

```
var a = 100;    //a variável pertence ao escopo local
b = 50;        //a variável pertence ao escopo global
```

Fonte: Autoria própria, 2018.

Outro ponto essencial das funcionalidades do JavaScript são as funções. Quando temos trechos de código repetidos em diferentes pontos, esses trechos podem se tornar funções. Função é um bloco de código que realiza ações, como efetuar cálculos ou manipular dados e retornar um valor. Também são criadas com o intuito de facilitar a manutenção e dar maior legibilidade ao programa (SILVA, 2010).

A figura 14 representa o código de uma função que faz a validação dos campos, identifica se o usuário preencheu todos os campos do formulário, se isso não ocorrer a função exibirá uma caixa de alerta na página, figura 15, avisando que há campos em branco e que devem ser preenchidos. A função é chamada na *tag* de abertura do formulário, ao apertar o botão enviar, figura 10, por meio do atributo *onsubmit="return Verificar()"*.

**Figura 14 – Código da Função de Validação**

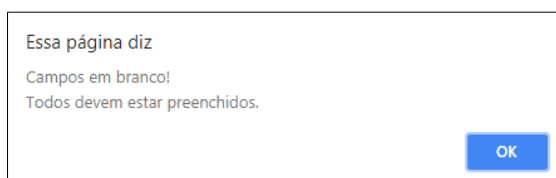
```
<!DOCTYPE>
<html>
  <head>
    <title> </title>
    <meta charset="utf-8">
    <link rel="stylesheet" type="text/css" href="EstiloTCC.css"/>
    <script type="text/javascript">

      function Verificar() {
        var NOME    = document.getElementById('cad').nome.value;
        var CPF      = document.getElementById('cad').numcpf.value;
        var EMAIL    = document.getElementById('cad').email.value;
        var CEL      = document.getElementById('cad').cel.value;
        var CELULAR  = document.getElementById('cad').celular.value;

        if (NOME == '' || CPF == '' || EMAIL == '' || CEL == '' || CELULAR == '') {
          alert ('Campos em branco!\nTodos devem estar preenchidos.');
```

Fonte: Autoria própria, 2018.

**Figura 15 – Caixa de Alerta da Função**



Fonte: Autoria própria, 2018.

## 2.8 Banco de Dados

"A expressão Banco de Dados surgiu do inglês *Databanks*, este foi trocado pela palavra *Databases*, Base de Dados, devido possuir significação mais apropriada" (SETZER; SILVA, 2005, p. 1).

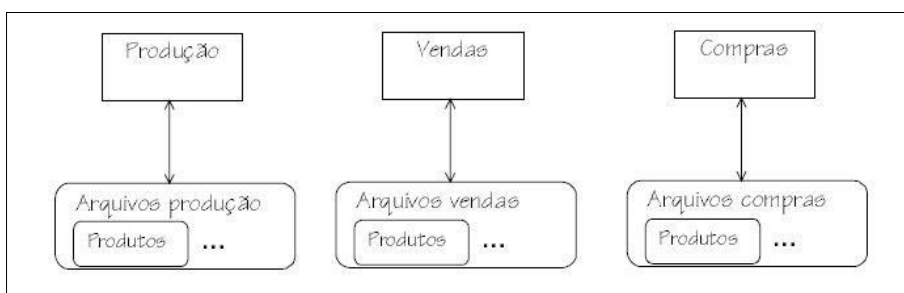
Como define Date (2004), um Banco de Dados (BD) é um conjunto de dados fixos, usado pelos sistemas em uma determinada aplicação para uma determinada função.

De uma forma mais simples de entendimento, o Banco de Dados é um local para armazenar dados necessários para a atividade de determinada organização, sendo ele a fonte de dados para aplicações que vierem a surgir para favorecer a produção desta organização.

Para Elmasri e Navathe (2011) um Banco de Dados detém as seguintes propriedades: devem representar alguma característica do mundo real, chamadas de minimundo ou universo de discurso onde as mudanças feitas serão também refletidas no banco de dados. O banco deve ser uma coleção lógica, coerente e de algum significado inerente de dados sendo o oposto disso descaracterizado como tal, e deve ser projetado e preenchido com dados para uma finalidade específica possuindo um grupo de usuários e aplicações previamente idealizadas. Assim, um BD é um conjunto de dados relacionados, criado para determinado objetivo que atenda aos usuários.

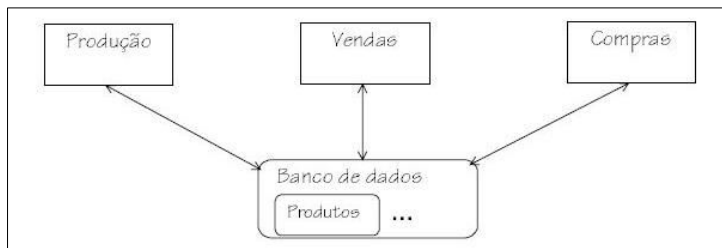
Segundo Heuser (1998), ao optar por não utilizar um Banco de Dados surge o problema de redundância de dados, que ocorre quando uma informação se repete em um computador por diversas vezes. A redundância traz consigo problemas como a reescrita, que é a repetição de dados no sistema, e Inconsistências de Dados que acontece quando uma informação é alterada sem que as demais fossem. O autor exemplifica, de forma hipotética, uma operação comercial que utiliza três sistemas que devem compartilhar informações entre si, porém de forma isolada há a ocorrência dos problemas mencionados, como mostra a figura 16.

**Figura 16 – Sistemas Isolados**



Fonte: Heuser (1998).

Aplicando os conceitos apresentados, acontece a centralização dos dados, exemplificado pela figura 17.

**Figura 17 – Sistemas Integrados**

Fonte: Heuser (1998).

De acordo com Date (2004), os modelos de dados servem para descrever de forma visível como os dados se relacionam dentro do banco de dados omitindo os detalhes de como eles estão sendo armazenados. Facilitando o entendimento e afetando de forma crítica o projeto de manipulação de dados correspondentes, que precisa ter a sua operação definida. Portanto, em questão sobre qual modelo deve ser utilizado, leva-se em conta qual o suporte de dados e operadores ele oferece.

### 2.8.1 Abordagem Relacional

A abordagem Relacional de um Banco de Dados é um modelo representativo ou de implementação criado por Ted Codd, em 1970. Um Banco de Dados que use a abordagem relacional é composto por tabelas e suas entidades e as relações que existem entre elas, como afirma Heuser (1998).

Em cada relação há um conjunto de colunas, que caracteriza a relação como um fato, objeto abstrato ou físico do mundo real, chamadas comumente de atributos, e linhas que armazenam a coleção de dados que compõe a tabela (ELMASRI E NAVATHE, 2011).

A figura 18 favorecerá maior entendimento do conceito.

**Figura 18 – Tabela Alunos**

Alunos			
CdAlunos	Matricula	Nome	Curso
A1	12254,3	João	Informática
A2	12254,4	Pedro	Administração
A3	12254,5	Augusto	Marketing
A4	12254,6	Gustavo	Filosofia

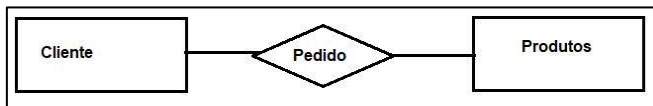
Fonte: Autoria própria, 2018.

A abordagem relacional se utiliza da técnica do modelo Entidade-Relacionamento (E-R) para ser estruturada. Com o Diagrama de Entidade e Relacionamento (DER)

podemos obter um modelo totalmente conceitual do sistema do Banco de Dados a ser implantado sem levar em conta de como esses dados são armazenados, porém evidenciando as relações entre as entidades e o funcionamento do BD, facilitando o projeto do Banco de Dados, como explicam Machado e Abreu (2004).

Silberschatz, Korth e Sudarchan (2006) definem entidade como um objeto ou uma coisa do mundo real caracterizada pelas suas propriedades e a interação que acontece entre as entidades recebem o nome de relacionamento/relação, sendo uma relação um objeto conceitual. Como apresenta a figura 19, a entidade cliente se relaciona com a entidade produtos, em uma leitura lógica ficaria: cliente faz pedido de produtos.

**Figura 19 – DER – Sistema de Pedido**



Fonte: Autoria própria, 2018.

Para Heuser (1998), um projeto de Banco de Dados deve dar grande importância em saber quantas vezes uma ocorrência de uma entidade pode estar associada a uma relação, essa propriedade é chamada de cardinalidade de uma entidade em um relacionamento podendo existir dois tipos a se considerar: a cardinalidade mínima e máxima. O autor ainda define que a cardinalidade mínima e máxima é quantas vezes é possível que a ocorrência de uma entidade pode acontecer em um relacionamento. Aplicando a um conjunto de entidade departamento, empregado, em que a relação seja lotação, a ideia de cardinalidade fica mais clara na figura 20. Fazendo a leitura lógica observamos que em um departamento estão lotados inúmeros empregados e o inverso desta leitura é que vários empregados estão lotados em apenas um departamento. Em outros termos, um departamento pode ter mais de um empregado, mas cada empregado tem apenas um departamento.

**Figura 20 – Cardinalidade Departamento Empregado**



Fonte: Heuser (1998).

Toda a relação entre as tabelas ocorre por suas linhas, por definição uma linha não é igual a outra, e para firmar esta distinção há um atributo que deve seguir um padrão não nulo que distingue cada linha da próxima e da anterior, como explica Navathe (2005). Esse atributo é chamado de Chave Primária, responsável pela identificação das ocorrências das entidades na relação e qual linha está se relacionando. A figura 18 deixa o conceito de Chave Primária mais claro, o atributo que não deve se repetir e ser utilizado como identificador da linha da tabela Alunos é o cdAluno. A Chave-Primária tem mais duas variações, que são a Chave Estrangeira e a Super Chave.

Heuser (1998) explica que Chave Estrangeira é um conjunto de colunas, atributos, que necessariamente aparecem como Chave Primária em uma tabela, ela é a ferramenta que permite a implementação de uma relação com outra tabela. O autor ainda explica que a Super Chave nada mais é que um conjunto de atributos únicos que podem ser utilizados para identificar uma linha. É possível visualizar esta determinação se idealizar uma Tabela pessoa onde existam os atributos Documento de identificação, Nº de cadastro de pessoa física e Nome. Esses três atributos juntos funcionam como uma Super Chave.

Para elucidar o conceito de chaves os autores Elmasri e Navathe (2005) explicam que Chave Estrangeira é toda Chave Primária que se encontra em uma tabela diferente da sua origem. Como exemplifica a figura 21, onde o código de departamento, que é uma Chave Primária, encontra-se também na tabela de empregados como Chave Estrangeira, criando a relação entre as tabelas e indicando o departamento de cada empregado.

**Figura 21 – Tabela Departamento e Empregado**

Dept	
CodigoDepto	NomeDepto
D1	Compras
D2	Engenharia
D3	Vendas

Emp				
CodigoEmp	Nome	CodigoDepto	CategFuncional	CIC
E1	Souza	D1	-	132.121.331-20
E2	Santos	D2	C5	891.221.111-11
E3	Silva	D2	C5	341.511.775-45
E5	Soares	D1	C2	631.692.754-88

Fonte: Heuser (1998).



## 2.8.2 Normalização

Depois de se construir um modelo de Banco de Dados, é aplicado sobre o projeto o conceito de normalização, que foi introduzido por E. F. Codd em 1970 com fundamentos na teoria matemática de conjuntos. Através deste processo é possível excluir do projeto possíveis anomalias que são prejudiciais para o funcionamento do sistema, como a repetição de dados, Chave Primária concatenadas, perda de informação e dificuldade de representar um fato e/ou objeto real. A normalização consiste em cinco fases, porém as três primeiras são de extrema importância para a eficiência do BD, sendo as seguintes inerentes à idealização do projeto (MACHADO; ABREU, 2004).

Como explica Heuser (1998), depois de obtido a primeira abstração da entidade, é passível de observação que ela não estará normalizada, ou seja, não se encontrará na Primeira Forma Normal (1FN), logo esta tabela Não Normalizada (ÑN) precisa passar pelos procedimentos de normalização para que evite a repetição de dados causando redundância. A 1FN identifica a Chave Primária da entidade, se contém informações independentes e se há grupos repetitivos, e se houver, cria uma entidade relacionada com a Chave Primária da sua origem. Como exemplo a Tabela Clientes ÑN, figura 22, contém diversas ocorrências de telefones em uma mesma linha. Aplicando os procedimentos da 1FN, geramos a entidade Telefones como na figura 23, e dividimos o endereço em novos atributos.

**Figura 22 – Tabela Cliente ÑN**

Código do cliente	nome	Telefone	Endereço
C001	João	(11)97053-0140 (11)98353-0130	Conjunto Habitacional Jovial Rural Rua Ary da Rocha Miranda, 36 CEP: 02281-190
C002	José	(11)97053-3280 (11)99053-2030	Pacaembu Rua Itápolis, 389 CEP: 01245-000
C003	Maria	(11)96553-1540	São Bernardo do Campo Av. Senador Vergueiro, 2432 CEP: 096000-000
C004	Eduardo	(11)97053-1040	Pinheiros Rua João Moura, 307 CEP: 05401-000

Fonte: Autoria própria, 2018.

Usando os conceitos da 1FN, é possível observar que o atributo Endereço tem informações independentes.

**Figura 23 – Tabela Cliente ÑN (2)**

Codigo Cliente	Nome	Telefone	Endereço	Bairro	CEP
C001	Jão	(11)9 7053-0140 (11)98353-0130	Rua Ary da Rocha Miranda, 36	Conj. Hab. Jova Rural	02281-190
C002	José	(11)9 7053-3280 (11)99053-2030	Rua Itápolis, 389	Pacaembu	01245-000
C003	Maria	(11)96553-1540	Av. Senador Vergueiro, 2432	São Bernardo do Campo	09600-000
C004	Eduarda	(11)9 7053-0140	Rua João Moura, 307	Pinheiros	05401-000

Fonte: Autoria própria, 2018.

Para a tabela clientes estar normalizada na 1FN ainda é preciso eliminar as duplas ocorrências de dados no atributo telefone, criando uma segunda entidade para o atributo e identificando pela Chave Primária da tabela de origem.

**Figura 24 – Tabela Clientes Normalizada na 1FN e Tabela Telefone**

Codigo Cliente	Nome	Endereço	Bairro	CEP
C001	Jão	Rua Ary da Rocha Miranda, 36	Conj. Hab. Jova Rural	02281-190
C002	José	Rua Itápolis, 389	Pacaembu	01245-000
C003	Maria	Av. Senador Vergueiro, 2432	São Bernardo do Campo	09600-000
C004	Eduarda	Rua João Moura, 307	Pinheiros	05401-000

Codigo Cliente	Telefones
C001	(11)97053-0140
C001	(11)98353-0130
C002	(11)97053-3280
C002	(11)99053-2030
C003	(11)96553-1540
C004	(11)97053-0140

Fonte: Autoria própria, 2018.

Antes de apresentar as próximas formas normais, é preciso inserir os conceitos de dependências funcional parcial e completa.

Segundo Date (1984), se em uma tabela com Chave Primária concatenada um ou mais atributos apresenta alguma relação direta com uma das chaves existentes, este atributo se torna dependente total desta chave. Se o atributo se repete em ocorrência desta chave ele se torna um dependente parcial.

Machado e Abreu (2004) explicam que para estar na Segunda Forma Normal (2FN), estas dependências precisam se tornar novas entidades onde a Chave Primária desta nova entidade é a relação de dependência que existirá na tabela de onde o atributo foi extraído. Por exemplo, em uma entidade Vendas que tem como Chave Primária Nº pedido, onde o atributo Nome do Produto depende da chave Cd

Produto. A figura 25 exemplifica de forma visual a ocorrência de uma entidade ainda Não Normalizada na Segunda Forma Normal.

**Figura 25 – Tabela Pedido ÑN na 2ºFN**

Nº Pedido	Cd Produto	Produto	Qtd.	Valor unit	Subtotal
5	935	SmartPhone	5	R\$ 1.300,00	R\$ 6.500,00
6	936	smartwatch	3	R\$ 1.450,00	R\$ 4.350,00
7	938	Fone wi-fi	6	R\$ 45,00	R\$ 270,00

Fonte: Autoria própria, 2018.

Aplicando os processos de normalização da 2FN apresentado nos parágrafos anteriores o atributo Produto que é dependente da chave Cd Produto deve pertencer a outra entidade. Sendo a chave agora Chave Estrangeira na tabela Pedido. A figura 26 mostra o resultado da tabela normalizada na Segunda Forma Normal.

**Figura 26 – Tabela Pedido e Produto Normalizadas**

Nº Pedido	Cd Produto	Qtd.	Valor unit	Subtotal
5	935	5	R\$ 1.300,00	R\$ 6.500,00
6	936	3	R\$ 1.450,00	R\$ 4.350,00
7	938	6	R\$ 45,00	R\$ 270,00

Cd Produto	Produto
935	SmartPhone
936	smartwatch
938	Fone wi-fi

Fonte: Autoria própria, 2018.

Uma entidade só não estará na Terceira Forma Normal (3FN) se ainda não estiver passado pelos processos de normalização anteriores e se existir dependência entre um de seus atributos que não são chaves. Como exemplo, o atributo Subtotal da entidade Pedidos, que pode ser gerada pela multiplicação dos atributos Qtd. e Valor unit, mostrado na figura 26. Para normalizar esta entidade na Terceira Forma Normal, basta excluir o atributo subtotal, gerando a tabela da figura 27.

**Figura 27 - Tabela Pedidos Normalizada na 3FN**

Nº Pedido	Cd Produto	Qtd.	Valor unit
5	935	5	R\$ 1.300,00
6	936	3	R\$ 1.450,00
7	938	6	R\$ 45,00

Fonte: Autoria própria, 2018.

### 2.8.3 Dicionário de Dados

Conhecido também como modelo lógico ou catálogo de dados, Heuser (1998) define o Dicionário de Dados (DD) como um nível de abstração ao nível do usuário, em que nele é descrito todos os componentes de uma entidade e seus atributos, como também sua descrição e função no Banco de Dados, desconsiderando a forma de como os dados são armazenados fisicamente. O DD usa uma notação própria de símbolos para escrita usando os elementos dispostos na figura 28.

**Figura 28 - Símbolos do Dicionário de Dados**

Símbolo	Significado
=	é constituído por ou é definido por
+	e (conjunção ou concatenação)
()	enquadram componentes opcionais
[]	enquadram componentes que são utilizadas alternativamente
	separam componentes alternativas enquadradas por [ ]
{ }	enquadram componentes que se repetem 0 ou mais vezes
**	enquadram comentários
@	identifica a chave primária de um depósito

Fonte: Autoria própria, 2018.

Para exemplificar, pode-se construir o DD da entidade da figura 27:

**Figura 29 - Exemplo de Dicionário de Dados**

<b>Pedido</b> = (@Nº_Pedido; Cd_Produto. Qtd; Valor unit)
<b>Nº_pedido</b> = {Digitos} *Chave Primária*
<b>Cd_produto</b> = {Digitos} *Chave estrangeira da entidade
<b>Qtd</b> = {Digitos} *Quantidade de produto*
<b>Valor_Unit</b> = {Digitos} *Valor de cada unidade*
<b>Digitos</b> = [0 1 2 3 4 5 6 7 8 9]

Fonte: Autoria própria, 2018.

## 2.9 UML

*Unified Modeling Language* ou Linguagem de Modelagem Unificada é definida por Guedes (2011) como uma linguagem gráfica utilizada principalmente para a

modelagem de projetos de *softwares*, e que possibilita a previsibilidade dos resultados destes.

Para Booch, Rumbaugh e Jacobson (2012), o objetivo da UML é oferecer a qualquer pessoa envolvida em produção, instalação e manutenção de um *software* uma notação padronizada para expressar o projeto de um sistema.

Na perspectiva de Guedes (2011), a UML auxilia engenheiros de *software* a definir as características do sistema proposto, os quais são requisitos, comportamento, estrutura lógica, dinâmica dos processos e as necessidades físicas em relação ao equipamento sobre o qual o sistema será implementado.

O ideal é que as características citadas sejam definidas por meio da UML, formando um modelo, antes do *software* começar a ser desenvolvido. Assim, há o planejamento de como cada elemento deverá funcionar dentro do *software*, especificado e analisado pelo projeto. Tais processos são necessários como prelúdio do desenvolvimento do *software*.

Bezerra (2015) afirma que uma característica intrínseca de sistemas de *software* é a complexidade de seu desenvolvimento, que aumenta à medida que cresce o tamanho do sistema. Desta forma, entende-se que a modelagem do *software* é o planejamento de cada etapa de seu desenvolvimento e das expectativas de seus resultados para que se torne possível, então, alcançá-los.

Com tamanha complexidade, a construção de um sistema de *software* pode ser comparada com a construção de sistemas habitacionais. Em ambos os casos, é necessário que sejam feitas análises e definições antes do início da construção. Essas análises e definições geram o modelo. Consequentemente, a realização do desenvolvimento do projeto deve corresponder aos parâmetros pré-definidos no modelo. “De uma perspectiva mais ampla, um modelo pode ser visto como uma representação idealizada de um sistema a ser construído”, (BEZERRA, 2007, p.2).

Booch, Rumbaugh e Jacobson (2012) definem a modelagem como a parte central de todas as atividades que levam à implantação de um *software*. Modelos são construídos para comunicar a estrutura e os comportamentos desejados do sistema, tanto quanto para visualizar e controlar a arquitetura do sistema, e assim, oferecer melhor compreensão do sistema que está sendo elaborado e gerenciar os riscos.

Em detrimento do mercado, legislações e até mesmo necessidades requeridas, sistemas de informação são considerados dinâmicos, pois mudam e atualizam-se constantemente. Esta é uma das razões que exemplifica a necessidade de atribuir a um projeto uma documentação precisa, detalhada e atualizada. “Grandes projetos não podem ser modelados de cabeça, nem mesmo a maioria dos pequenos projetos pode sê-lo, exceto, talvez, aqueles extremamente simples”, (GUEDES, 2011, p.20).

Como define Bezerra (2015), através da modelagem de um sistema, os indivíduos envolvidos no seu desenvolvimento podem fazer estudos e prever comportamentos do sistema em desenvolvimento. Vê-se que a modelagem é fundamental diante da limitação humana em lidar com a complexidade, sendo assim, necessário desenvolver modelos que representam perspectivas do sistema. Em cada modelo, devem ser ignorados os detalhes irrelevantes, pois podem dificultar o entendimento do sistema. Baseando-se na abstração, segundo a qual somente características relevantes à resolução de um problema devem ser consideradas, modelos cumprem a função de serem primariamente objetivos, revelando as características essenciais do sistema. “Construímos modelos de sistemas complexos porque não é possível compreendê-los em sua totalidade”, (BOOCH; RUMBAUGH; JACOBSON, 2012, p.7).

### **2.9.1 Levantamento de Análise e Requisitos**

De acordo com Guedes (2011), existem métodos para o processo de modelagem e desenvolvimento de um *software*. Apesar das diferentes nomenclaturas, o conceito por trás das etapas é próximo, quando não semelhante.

Uma das primeiras etapas desse processo denomina-se levantamento e análise de requisitos. Nesta etapa, o foco é esclarecer e dominar o problema do projeto em questão para que subsequentemente seja formulada a solução que será aplicada a esse problema através do *software*. Posteriormente, é discutido e determinado se tal formulação é aplicável e realmente pode ser desenvolvida. De forma sucinta, como é dito por Bezerra (2015), a etapa de levantamento de requisitos é dedicada a compreensão do problema, e seu principal objetivo é que a compreensão seja unânime entre os envolvidos no projeto.

Guedes (2011) ainda explica a existência de dois tipos de requisitos que devem ser levantados e analisados. São estes os requisitos funcionais e não-funcionais. Os requisitos funcionais remetem diretamente às funcionalidades do *software*, isto é, à sua performance. Requisitos não-funcionais determinam como serão aplicados os requisitos funcionais, caracterizando escopos como desempenho, interface e segurança, que enfim são condições que devem ser aplicadas na execução das funcionalidades do *software*. Há, ainda, a consideração de um terceiro tipo de requisito em alguns casos. É o chamado requisito normativo, que se refere a restrições como prazos, custos, legalidade, limitações e regras no *software* e seu desenvolvimento. Os requisitos por fim são um estudo e uma exploração das necessidades dos usuários do *software*. “Uma das formas de se medir a qualidade de um sistema de software é pela sua utilidade. E um sistema será útil para seus usuários se atender aos requisitos definidos e se esses requisitos refletirem as necessidades dos usuários” (BEZERRA, 2015, p.24).

A fase de análise é destinada ao estudo dos componentes e sua interação no funcionamento do sistema. Conforme Bezerra (2015), essa atividade permite que os requisitos levantados sejam estudados detalhadamente. Guedes (2011), conclui que essa é a fase responsável por determinar as necessidades do sistema, através da verificação dos requisitos convenientes e corretos e se estes são compreendidos unanimemente.

Como complemento às fases de análise e requisitos, a prototipagem é o processo de construção dos esboços do sistema. Esse processo, segundo Bezerra (2015), conta com o auxílio de programação visual. Guedes (2011) afirma que a técnica da prototipação possibilita a visualização hipotética do sistema finalizado, servindo de rascunho desse sistema. O protótipo, então, apresenta a interface do *software*, ou seja, suas telas, juntamente aos tipos de informações que serão inseridas no *software*.

Ainda para Guedes (2011), o protótipo é o produto das fases de análise e requisitos, onde demonstra-se não apenas o comportamento do *software*, mas os tipos de informações e quais informações serão inseridas e fornecidas nesse sistema. O protótipo possibilita que o *software* atenda às necessidades determinadas nas fases de análise e requisitos.

As fases anteriores trabalham diretamente com o domínio do problema. A etapa que se direciona ao domínio da solução é a etapa de projeto. Essa fase propõe a aplicação de como será solucionado o problema determinado antes, através do *software*. Como dito por Bezerra (2015), a fase de projeto possui dois processos, os quais são a construção do projeto da arquitetura e do projeto detalhado.

O projeto de arquitetura é considerado de alto nível, e é nele que ocorre a distribuição das classes de objetos do sistema. Já no projeto detalhado, são realizados os projetos de interface, banco de dados e algoritmos. Conforme Guedes (2011), essa etapa leva em conta aspectos como concorrência e distribuição para modelar suas funcionalidades. Alguns dos diagramas da UML utilizados aqui são os diagramas de classes, diagramas de casos de uso e diagramas de atividades. O projeto, por fim, leva em conta as restrições de tecnologia e os recursos existentes para a solução do problema através do desenvolvimento do *software*. Essas considerações definem quais serão os elementos utilizados para a produção desse *software*, como linguagens de programação e banco de dados, e ainda propõem a interface e *hardware* final do *software*.

Fundamental à modelagem, os diagramas da UML possuem o objetivo de oferecer múltiplas visões do sistema. Diagramas permitem análises e modelos de diversos aspectos, sendo alguns mais generalizados e outros mais específicos. Para Guedes (2011), é isso o que torna um projeto completo, pois são estudados todos os segmentos da produção do *software*.

Como afirma Góes (2014), a UML possui treze tipos de diagramas na versão 2.0. Esses diagramas são considerados dinâmicos e estáticos, distribuídos em três categorias classificadas, as quais são diagramas de estrutura, diagramas de comportamento e diagramas de interação.

### **2.9.2 Diagrama de Casos de Uso**

O diagrama de casos de uso é um diagrama estático e é classificado por Booch, Rumbaugh e Jacobson (2012) como modelagem básica de comportamento. Góes (2014), descreve o diagrama de casos de uso como o diagrama que representa um sistema na perspectiva do usuário. No diagrama é retratada toda a composição do sistema, desde seus módulos e usuários a quais os papéis de desempenho no



funcionamento geral. De acordo com Guedes (2011), desenvolvido nas fases de levantamento e análise de requisitos, é um diagrama que serve de base para outros diagramas e é comumente consultado durante todo o processo de modelagem.

“Casos de uso bem estruturados denotam somente o comportamento essencial do sistema ou subsistema e não são amplamente gerais, nem muito específicos”, (BOOCH; RUMBAUGH; JACOBSON, 2012, p.246).

Para Góes (2014), os principais elementos presentes em diagramas de casos de uso são:

- Casos de uso;
- Atores;
- Relacionamentos ou associações;
- Inclusão ou *include*;
- Extensão ou *extends*;
- E especialização ou generalização.

Como ilustrado na figura 39, as elipses representam os casos de uso. Essas elipses devem conter um nome, que anunciará o componente do sistema que será manipulado por seus usuários. Góes (2014) explica casos de uso como uma divisão de problemas, ou seja, a distribuição das tarefas do sistema, o que facilitará a compreensão e a solução desses problemas. Como descrevem Booch, Rumbaugh e Jacobson (2012), o cenário, que é o espaço retangular onde estarão as elipses, seus relacionamentos e dependências, é separado dos atores, que ficam do lado de fora deste espaço. Os atores são representados por figuras de que remetem ao estereótipo humano. Relacionamentos localizam-se dentro do retângulo, formando ligações entre casos de uso.

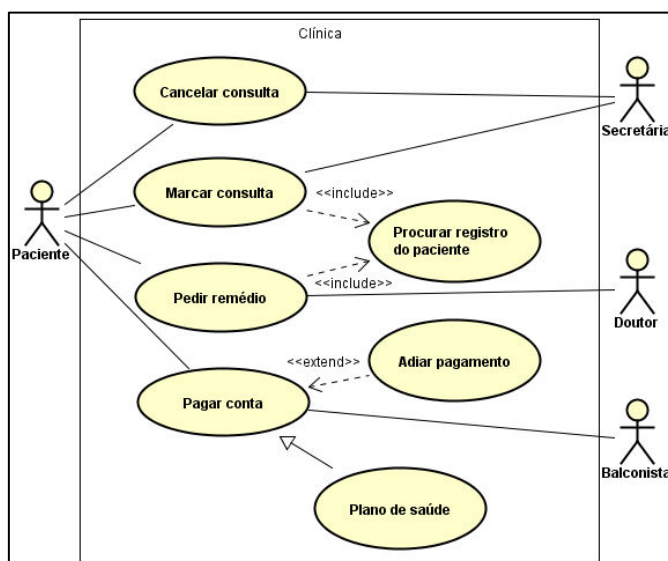
A figura 39 é o diagrama de caso de uso do sistema de uma clínica. Os atores que irão interagir com esse sistema são paciente, secretária, doutor e balconista. Atores são elementos externos que interagem com o sistema, e podem ser pessoas, empresas ou fragmentos dessa empresa como um setor, ao passo que também podem ser sistemas, órgãos e até equipamentos. Por esse mesmo motivo atores situam-se fora do espaço retangular em diagramas de casos de uso. Como referenciado por Góes (2014), atores interagem com um sistema buscando um objetivo. No exemplo da figura 39, pode-se observar os objetivos dos atores através

dos casos de uso com os quais estão relacionados, sob a ligação de um segmento de reta entre um e outro.

O ator paciente relaciona-se com o sistema da clínica através dos casos de uso marcar consulta, cancelar consulta, pedir remédio e pagar conta. Casos de uso são sempre escritos com verbos no infinitivo, e são capturas de requisitos. Como dito por Guedes (2011), referem-se aos serviços necessários para a performance do *software* e que serão úteis para seus usuários. Um paciente, ao requisitar tais ações, está interagindo com a secretária, a qual é responsável por marcar e cancelar consulta; com o doutor, o qual somente pode receitar remédios; e com o balconista, que é o ator responsável pela efetuação do pagamento da conta.

Ainda na figura 30, pode-se observar os relacionamentos de inclusão e exclusão. Inclusão ou *include* é um relacionamento onde ocorre uma obrigatoriedade entre casos de uso. É representado por uma reta tracejada com uma seta que aponta para o caso de uso incluído. O caso de uso procurar registro do paciente é obrigatoriamente incluído em marcar consulta. Para emissão de remédios, também é necessário do registro do paciente, logo esses mesmos casos de uso também possuem relação de inclusão. Conforme Guedes (2011), um relacionamento de inclusão ocorre para situações que são comuns a mais de um caso de uso. O *include*, portanto, ocorre quando uma situação é dependente da outra, isto é, para ser realizada precisa obrigatoriamente de outra.

**Figura 30 - Diagrama de Caso de Uso**



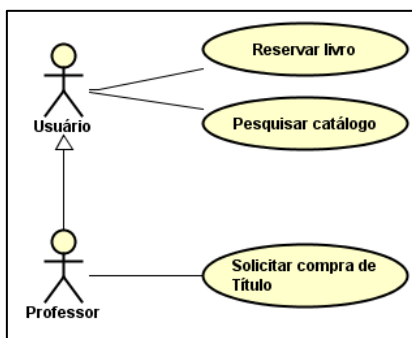
Fonte: Autoria própria, 2018.

A associação de extensão ou *extend*, ao contrário da associação de inclusão, ocorre quando um caso de uso pode ser agregado a outro caso de uso. Ou seja, a utilização do caso de uso com *extend* é opcional. Representado de forma semelhante ao *include*, com reta tracejada, porém sua seta aponta para o caso de uso estendido. Na figura 30, o caso de uso pagar conta possui a associação de extensão com adiar pagamento. Isso significa que caso paciente deseje pagar a conta ele pode fazê-lo, mas também possui a opção de adiar o pagamento se assim desejar. Como pontuado por Bezerra (2015), casos de uso estendidos são eventualmente executados.

O relacionamento de generalização ou especialização pode ocorrer tanto entre casos de usos como entre atores. Na figura 30, o caso de uso plano de saúde é uma especialização do caso de uso pagar conta. O caso de uso especializado herda as características ou ações do caso de uso generalizado. No exemplo da figura 30, o paciente pode utilizar seu plano de saúde para pagar a conta, sendo assim, uma forma específica de pagar a conta.

A generalização entre atores é exemplificada por Bezerra (2015) na figura 31, a qual mostra dois atores, usuário e professor. Professor é uma especialização de usuário, portanto herda todas as possibilidades da generalização desse ator.

**Figura 31 - Associação de Generalização**



Fonte: Bezerra (2007).

A partir do diagrama de casos de uso é possível documentar requisitos, utilizando-se da abstração, flexibilidade e da informalidade para apresentar um sistema de informação e seus módulos de composição na perspectiva do usuário ou usuários com seus papéis. Como afirma Góes (2014), o diagrama de casos de uso não é orientado a objeto, pois sua função é oferecer uma visão procedural do sistema.

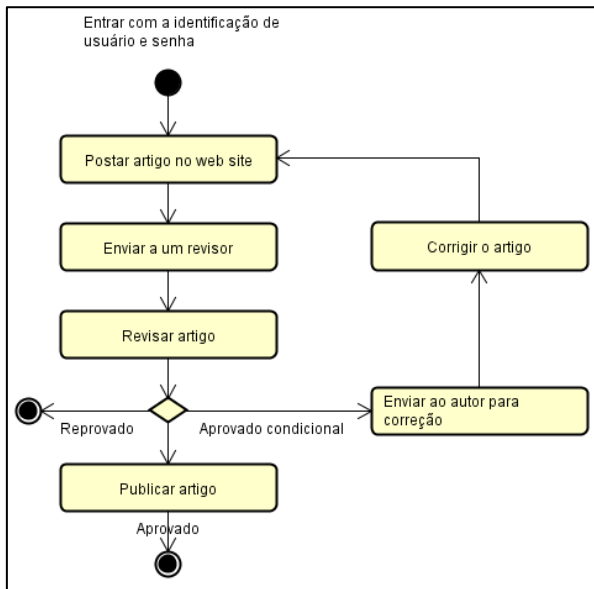
### 2.9.3 Diagrama de Atividades

Também classificado como modelagem de comportamento, o diagrama de atividades é um diagrama estático. O diagrama de atividades possui semelhança com fluxogramas, pois é destinado a modelagem de processos, algoritmos e módulos de sistemas de informação. Sendo assim, o diagrama de atividades modela aspectos dinâmicos de um sistema, ou seja, as atividades de um sistema. Booch, Rumbaugh e Jacobson (2012) definiram esse diagrama como um gráfico de fluxos.

Como evidencia Guedes (2011), o diagrama de atividades é usado para a representação de dois fluxos, os quais são fluxos de controle e de objetos. Seus principais elementos de composição são:

- Ação;
- Nó inicial;
- Nó de decisão;
- Fluxo de controle;
- E nó final.

A figura 32 é um diagrama de atividades que ilustra o fluxo de processos que irão autorizar ou reprovar a publicação de um artigo. Goés (2014) explica que o diagrama tem início a partir do círculo preenchido, que é o identificador do início do fluxo, sendo este o nó inicial. Em seguida, a figura retangular é uma ação contendo seu nome, que neste caso é postar artigo no *web site*. Ações devem começar com verbos no infinitivo, e são as ações que formam uma atividade. Essas ações são representações dos atos dos usuários e das reações do sistema. Ligando essas ações, existe um segmento de reta com uma seta que aponta para a próxima ação. Esses segmentos são os fluxos de controle, que servem para conectar as ações.

**Figura 32 - Diagrama de Atividades**

Fonte: Góes (2014).

O fluxo prossegue com ações que definem todo o percurso necessário para se publicar um artigo segundo o diagrama da figura 32. Após as ações enviar a um revisor e revisar artigo, há um nó de decisão. O nó de decisão é um losango, e é utilizado em situações em que há diferentes escolhas para o usuário ou reações do sistema diante uma condição. Como mencionado por Góes (2014), o nó de decisão é uma representação dos comandos condicionais *If* e *Else* utilizado nas linguagens de programação. Observando a situação ilustrada na figura 32 identifica-se que o nó de decisão pode levar a três situações diferentes. Se após a revisão o artigo for reprovado, o fluxo chega a figura circular preenchida com um envolto vazado. Se o artigo for aprovado, ele é publicado e então o fluxo chega ao final. E se o artigo for parcialmente aprovado, precisando de correções para sua publicação, ele passa por todas as etapas novamente até que a ação de revisar defina seu destino.

As atividades, como dito por Guedes (2011), podem ser métodos, algoritmos e processos completos. Assim, os diagramas de atividades podem ilustrar computação procedural, ao passo que modelam sistemas da informação.

“Uma atividade é composta por um conjunto de ações, ou seja, os passos necessários para que uma atividade seja concluída” (GUEDES, 2011, p. 277).

### 2.9.4 Diagrama de Classes

O diagrama de classes é um diagrama de estrutura. Conforme Booch, Rumbaugh e Jacobson (2012), o diagrama de classes possibilita que as visões estáticas de projetos de sistema sejam modeladas. Em clara definição, Góes (2014) define que o diagrama de classes é dedicado a dar ênfase nos dados que serão utilizados no sistema de informação e sua construção.

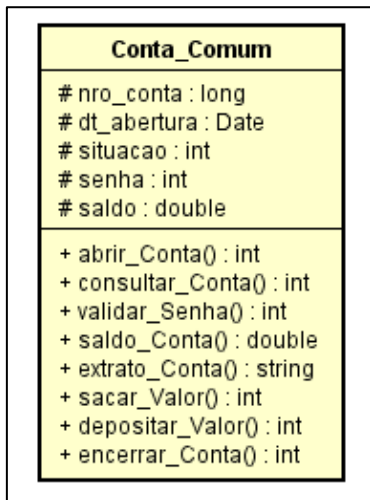
De acordo com Guedes (2011), esse diagrama compõe-se de classes e seus relacionamentos. Seus principais componentes são:

- Classes;
- Atributos;
- Associações;
- E relacionamentos.

O diagrama de classes viabiliza a visualização dos componentes da base de dados do sistema. Como referenciado por Góes (2014), o diagrama de classes foi baseado no modelo de Entidade e Relacionamento utilizado em bancos de dados. Por isso situa-se como um modelo estrutural, pois expõe parte da composição de um sistema, sem tratar aspectos dinâmicos, como nos diagramas de caso de uso e atividade, ou interativos.

No diagrama de classes, as classes representam classificações de objetos com atributos, que como explica Guedes (2011), armazenam os dados dos objetivos da classe e métodos em comum. A figura 33 é um exemplo de classe, que é uma figura retangular subdividida em três compartimentos na seguinte ordem: nome da classe, atributos e métodos. O nome de uma classe, como constata Góes (2014), é escrito no singular, em negrito e centralizado. Os *softwares* que oferecem as ferramentas para a criação de diagramas da UML realizam automaticamente essa formatação.

**Figura 33 - Classe**



Fonte: Guedes (2011).

No segundo compartimento é onde situam-se os atributos da classe `Conta_Comum`. Os atributos são sempre escritos com letras minúsculas, e podem ter nomes simples e compostos. Na figura 33 é possível observar exemplos de atributos simples e compostos. Os atributos `nro_conta` e `dt_abertura` são compostos por possuírem mais de uma palavra. Os atributos `situacao`, `senha` e `saldo` são atributos simples. Ao lado dos atributos estão símbolos de sustentado, e esses símbolos representam a visibilidade do atributo. Góes (2014) expõe as quatro classificações de visibilidade, que são:

- Pública ou *public*;
- Privada ou *private*;
- Protegida ou *protected*;
- E pacote ou *package*.

A visibilidade de um atributo é o indicador de utilização deste. No exemplo da figura 33, todos os atributos possuem a sua visibilidade protegida. A visibilidade protegida possui o símbolo de sustentado ou jogo da velha. Um atributo protegido só pode ser acessado dentro da mesma classe em que está declarado, ou pelas classes descendentes da classe na qual esse atributo pertence. Atributos do tipo público possuem o sinal de mais ou positivo e indicam que a sua acessibilidade é pública e permitida para qualquer classe. Ao contrário, atributos do tipo privado indicam que sua utilização só pode ser feita dentro da mesma classe na qual foi declarado. Atributos do tipo privado possuem o sinal de menos ou negativo. Por fim, atributos

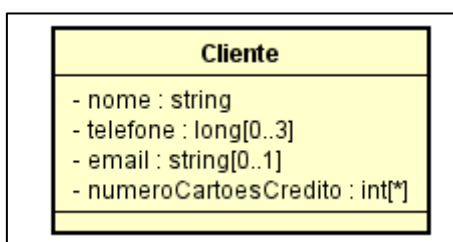
do tipo pacote possuem o sinal de til, e podem ser utilizados por outras classes desde que estejam no mesmo pacote.

Ao lado dos atributos na figura 33 é possível enxergar o tipo desses atributos. O tipo do atributo é declarado junto a este para que possa ocorrer a identificação dos dados que serão armazenados. O atributo `nro_conta` é um atributo do tipo *long*, o que significa que os dados que serão armazenados neste atributo são sequências numéricas longas. Utilizando mais um exemplo na figura 33, o atributo `dt_abertura` possui o tipo *Date*, o que indica que os dados armazenados neste atributo serão datas (GÓES, 2014).

A figura 33 é a classe de uma conta bancária. Seus métodos, listados no terceiro compartimento, possuem visibilidade pública. De acordo com Guedes (2011), os métodos são funções que um objeto de uma classe pode executar, e por isso, também são chamados de operações. Através dos métodos é possível definir quais os parâmetros de entrada dentro do diagrama de classes e definir quais valores são retornados. Os parâmetros vão dentro dos parênteses ao lado do nome do método, enquanto o valor de retorno vai após os dois pontos conseguinte a esses parênteses.

Como exemplifica Góes (2014), atributos podem possuir multiplicidade. A multiplicidade de um atributo define quantidade máxima e mínima de valores que este poderá ter. Na figura 34 observa-se o atributo `telefone`, e o número zero seguido de dois pontos até o número três, dentro de colchetes, indica que esse atributo pode conter até três números de telefone, ou nenhum. O mesmo ocorre com `email`, que indica que pode ou não haver um *e-mail*. Já em `numeroCartoesCredito`, a multiplicidade indica que vários cartões de crédito podem ser cadastrados.

**Figura 34 - Multiplicidade em Atributos**



Fonte: Autoria própria, 2018.



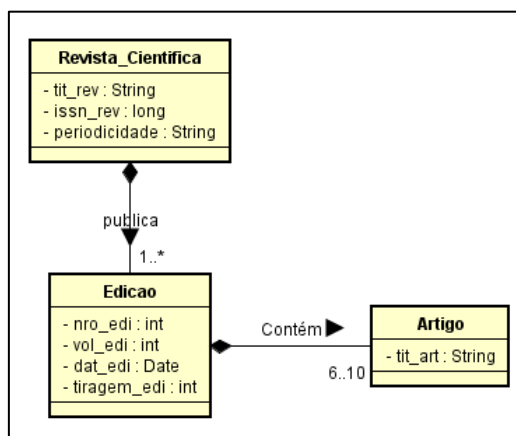
As associações no diagrama de classes permitem, nas palavras de Guedes (2011), que classes compartilhem informações umas com as outras, o que facilita a execução dos processos no sistema. Algumas das principais associações no diagrama de classes são:

- Associação de agregação;
- Associação de composição;
- Relacionamento de generalização ou especialização;
- E associação de dependência.

Como pontuado por Bezerra (2015), existem dois tipos de relacionamentos que são considerados relações de todo-parte. Essas relações indicam que entre dois objetos, um está contido no outro, ou então, contém o outro.

A associação de composição é, segundo Guedes (2011), uma variedade da agregação que estabelece um vínculo robusto entre objetos-todo e objetos-parte. A representação de uma composição é semelhante à de agregação com a diferença de que o losango na ponta é preenchido. Na figura 35, a classe *Revista\_Cientifica* possui uma associação de composição com a classe *Edicao*, que possui uma associação de composição com a classe *Artigo*. Lendo o diagrama, também é possível notar as multiplicidades em ambas associações. A primeira associação demonstra que um objeto da classe *Revista\_Cientifica* é composto por um ou mais objetos da classe *Edicao*. Na segunda associação, a multiplicidade de 6 a 10 indica que cada objeto da classe *Edicao* deve conter no mínimo 6 objetos da classe *Artigo* e no máximo 10 objetos dessa classe.

**Figura 35 – Associação de Composição**

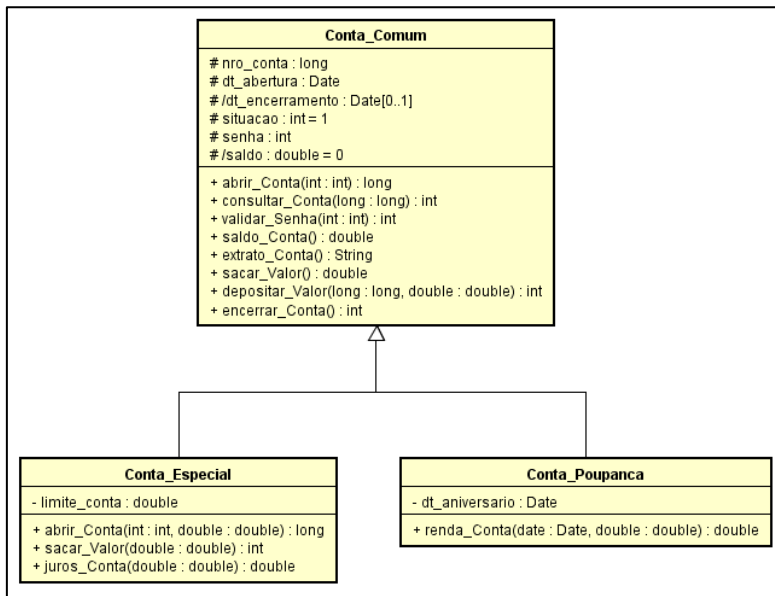


Fonte: Guedes (2011).

Especializações ou generalizações são classificadas por Bezerra (2015) como relacionamentos entre classes que indicam uma generalização ou especificação entre as classes. Essa relação também é conhecida como herança, pois uma classe específica herda características de uma classe generalizada, obtendo assim o mesmo sentido.

Como demarcam Booch, Rumbaugh e Jacobson (2012), classes em um relacionamento de generalização recebem nomes para serem identificadas. A classe generalizada é comumente conhecida como superclasse ou classe-mãe, entre outras denominações. As classes mais específicas são chamadas de subclasses ou classes filhas. Os objetos dessas subclasses herdam atributos e operações das superclasses.

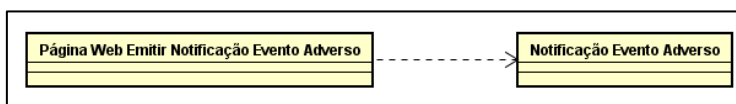
Na figura 38, um diagrama de classes com generalização por Guedes (2011), observa-se a classe `Conta_Comum` generalizada localizada acima das classes especializadas `Conta_Especial` e `Conta_Poupanca`. Como no diagrama de casos de uso, a generalização é um segmento de reta com uma seta na ponta que indica a classe generalizada, ligando-a com as demais. No exemplo dessa figura, as subclasses herdam todos os atributos da superclasse. A conta especial, tanto quanto a conta poupança, herdam o número da conta, data de abertura e encerramento, situação da conta, senha e saldo que são atributos de uma conta comum. O que especializa um objeto da conta especial é seu limite e suas operações, como na conta poupança, onde a data de aniversário é um atributo e há uma operação adicional.

**Figura 36 - Generalização ou Especialização**

Fonte: Guedes (2011).

Como coloca Góes (2014), uma associação de dependência define que há duas classes mantendo uma relação, uma independente e outra dependente. Essa associação define que modificações feitas na classe independente afetam os objetos das classes dependentes. Essa associação é representada por um segmento de reta tracejado com uma seta aberta em sua ponta, indicando a classe independente.

No exemplo da figura 37, criado por Góes (2014), a classe Página Web Emitir Notificação Evento Adverso depende diretamente da classe Notificação Evento Adverso. Qualquer modificação feita nos atributos da classe Notificação Evento Adverso consequentemente afetará o formulário eletrônico Página Web Emitir Notificação Evento Adverso.

**Figura 37 - Dependência**

Fonte: Góes (2014).

## 2.9.5 Diagrama de Sequência

Considerado um diagrama de comportamento, o diagrama de sequência é um diagrama de interação, que propõe descrever a interação e colaboração de objetos nos comportamentos do cenário (Fowler, 2005).

Conforme Guedes (2011), o diagrama de sequência é baseado no diagrama de casos de uso, sendo o diagrama de sequência utilizado para a descrição gráfica de um processo disparado por um ator. Sendo assim, cada diagrama de sequência é feito a partir de um caso de uso. Existe, ainda, uma dependência com o diagrama de classes, já que as classes dos objetos utilizados no diagrama de sequência estão descritas no diagrama de classes. Nas palavras de Guedes (2011), o diagrama de sequência permite validar e complementar o diagrama de classes, visualizando os métodos das classes.

Como descrito por Góes (2014), um diagrama de sequência contém:

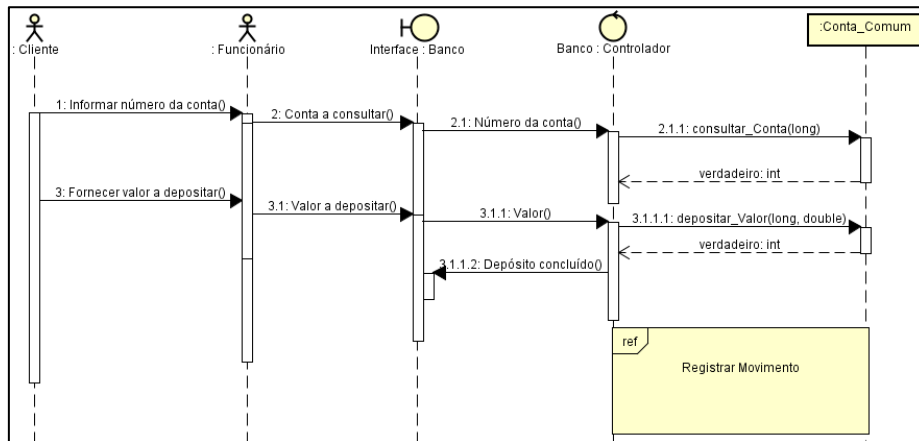
- Atores;
- Objetos;
- Fronteiras;
- Controles;
- Entidades;
- Mensagens ou estímulos;
- E linhas de vida.

Guedes (2011), descreve que os atores no diagrama de sequência são instâncias dos atores que são declarados nos diagramas de caso de uso. Apesar de não serem obrigatórios no diagrama de sequência, são usados com frequência.

Como explica Góes (2014), os atores agem como entidades interagindo com o sistema, enviando ou recebendo dados. Atores e classes de fronteira podem trocar mensagens entre si. Na figura 38, pode-se observar os atores Cliente e Funcionário.

Abaixo dos atores, como nos outros componentes, há uma linha vertical tracejada. Segundo Bezerra (2015), os componentes juntos a esses segmentos verticais tracejados formam uma linha de vida. A ordem comumente utilizada para facilitar a leitura do diagrama é posicionando, da esquerda para a direita, atores primariamente, posteriormente objetos de fronteira, objetos de controle e objetos de entidade.

**Figura 38 - Diagrama de Sequência**



Fonte: Guedes (2011).

O diagrama da figura 38 refere-se ao processo de realizar um depósito. Intuitivamente, é possível observar no diagrama através da sequência que o cliente informa ao funcionário o número da conta que receberá o valor a ser depositado. O funcionário realizará a verificação da conta, consultando a interface do sistema do banco. A interface do banco é um objeto de fronteira, ou seja, como descreve Goés (2014), um objeto responsável pela interface entre os usuários e o sistema, ou então pela comunicação entre sistemas.

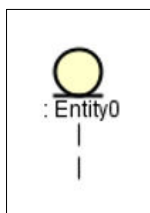
O sistema repassa o número da conta para o controlador, que por sua vez dispara o método `consultar_Conta`. O controlador representa controle, que para Guedes (2011), pode ser uma classe ou um objeto, como neste cenário, que executa as regras de negócio, separadamente da classe ou objeto de fronteira, que apenas exibe e recebe dados.

Essa sequência é seguida de mensagens síncronas, que são os segmentos de reta com setas preenchidas em sua ponta. Bezerra (2015), evidencia que mensagens ligam linhas de vida umas às outras. Se após o método `consultar_Conta`, ligado ao objeto `Conta_Comum`, a conta informada é encontrada, o ator funcionário solicita o valor a ser depositado pelo cliente. É utilizado um segmento tracejado de reta com seta aberta na ponta, indicando o sentido contrário para representar uma mensagem de retorno. Após fornecido o valor de depósito, o funcionário repassa para o sistema, que repassará para o controlador, que em sequência dispara o método `depositar_Valor`. Se o método for executado, o controlador solicita uma apresentação de mensagem a interface, informando que o depósito foi concluído.

Por último, o sistema realiza o registro do movimento, como um processo interno do sistema. Góes (2014), denomina esse recurso como ocorrência de interação. É representado por uma moldura, e é um elemento útil para simplificar diagramas.

Como explica Fowler (2005), no diagrama de sequência também existem classes ou objetos que armazenam e gerenciam dados do sistema. Essas classes ou objetos são conhecidos como entidades. O formato de uma entidade é um círculo com um segmento de reta acoplado abaixo, como ilustra a figura 39.

**Figura 39 - Entidade no Diagrama de Sequência**



Fonte: Autoria própria, 2018.

## 2.10 Paradigma de Orientação a Objetos

Como um modo de abordar problemas, o paradigma de orientação a objetos é descrito por Bezerra (2015) como indispensável para o desenvolvimento de *softwares* e sistemas contemporaneamente.

Um *software* de alta complexidade possui um desenvolvimento gradual e bem estruturado quando orientado a objetos. Conforme Silva (2009), a orientação a objetos (OO) modela o problema para solucioná-lo em código. A orientação a objetos é, dessa forma, um tratamento da complexidade para que se torne possível estruturar um *software* e então desenvolvê-lo.

A orientação a objetos remete a organização do *software* em um âmbito de diferentes objetos, que formam uma estrutura de dados e comportamentos. Para Blaha e Rumbaugh (2006), essa técnica de programação difere das outras, mais comumente a estruturada ou essencial, em que estruturas de dados e comportamentos são pouco ligadas.

Góes (2014), declara que um sistema de informação organizado como uma composição de objetos fornece melhor compreensão do mundo real. A orientação a objetos também é um método de modelagem, que tende a diminuir as dificuldades dos processos de análise, projeto e programação.

O conceito de abstração também é aplicado ao paradigma de orientação a objetos. Como dito por Bezerra (2015), a abstração em OO consiste em filtrar os aspectos mais relevantes para a construção de um sistema. Assim, se estabelece apenas o que é importante para o desenvolvimento do *software*.

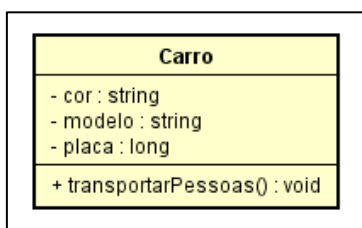
As principais características de um enfoque orientado a objetos incluem, segundo Blaha e Rumbaugh (2006), os seguintes aspectos:

- Identidade;
- Classificações;
- Herança;
- E polimorfismo.

Identidade refere-se aos dados, que são reconhecidos como entidades diferentes e chamados de objetos. Objetos, como descrevem Blaha e Rumbaugh (2006), podem ser concretos, como livro, veículo, pessoa, ou computador, ou então conceituais. Assim sendo, objetos possuem suas próprias identidades, o que significa que dois objetos diferentes podem ter atributos como nome e tamanho semelhantes, e serem objetos distintos.

A classificação é um dos aspectos que mais dependem da abstração. Guedes (2011), explica que classes são formadas para agrupar objetos de mesmas características e comportamentos. Usando os exemplos de carros, pessoas e casas, ilustra como o ser humano atribui uma visão orientada a objetos a respeito de objetos do mundo real. No exemplo de carros, entende-se que existem carros com diversos formatos, cores, tamanhos e modelos diferentes, como está exposto na classe Carro, na figura 40. Mas todos estes são carros, com atributos em comum, que variam em suas definições. Dessa maneira, carro é a denominação, ou a generalização de um grupo de objetos.

**Figura 40 - Classe Carro**

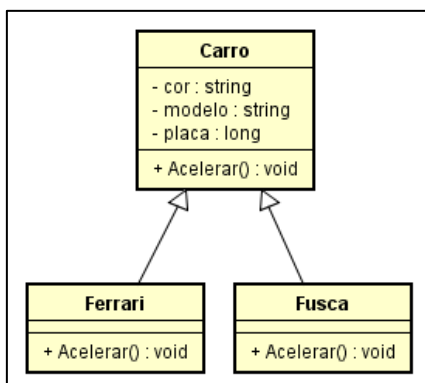


Fonte: Autoria própria, 2018.

Conforme Bezerra (2015), operações são ações que objetos ou classes podem manifestar. As operações podem alterar o valor dos atributos de uma classe, ou então, retornar informações sobre.

Segundo Blaha e Rumbaugh (2006), polimorfismo se trata de operações semelhantes se comportando de maneiras diferentes em diferentes classes. Na figura 41, pode-se observar na classe Carro a capacidade da mesma operação, que no caso é acelerar, se comportar de maneira diferente nas classes Ferrari e Fusca. Operações implementadas em classes específicas são chamadas de métodos. No exemplo da figura 41, a operação acelerar pode ser chamada por diferentes métodos, pois a aceleração de uma Ferrari certamente será diferente da aceleração de um fusca, sendo assim necessário que sejam implementados métodos diferentes de acelerar em ambas as classes.

**Figura 41 - Polimorfismo**

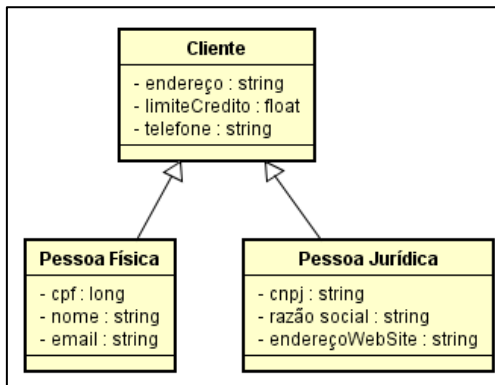


Fonte: Autoria própria, 2018.

Nas palavras de Bezerra (2015), a herança ou generalização é uma abordagem de abstração na OO. Isso significa que uma classe abstraída que contém os atributos e comportamentos de um grupo de objetos, pode ser a generalização maior entre classes, formando uma hierarquia. Os níveis da hierarquia indicam níveis de abstração. Como ilustrado na figura 53, as classes herdam os atributos e operações das classes com que estão associadas nos níveis acima. No exemplo da figura 53, as subclasses Pessoa Física e Pessoa Jurídica estão herdando os atributos de uma generalização de Cliente. Pessoas físicas e jurídicas nesse caso, são clientes, e tipos de clientes diferentes, mas que por serem clientes herdam os atributos e operações da classe Cliente.



**Figura 42 - Herança**



Fonte: Goés (2014).

Existe, ainda, o conceito de encapsulamento. Este é descrito por Goés (2014) como um ocultamento de informações, no qual o objetivo é fornecer segurança aos objetos. O encapsulamento permite que objetos sejam protegidos pelas classes, restringindo o acesso a operações e métodos.

## 2.11 Engenharia de Software

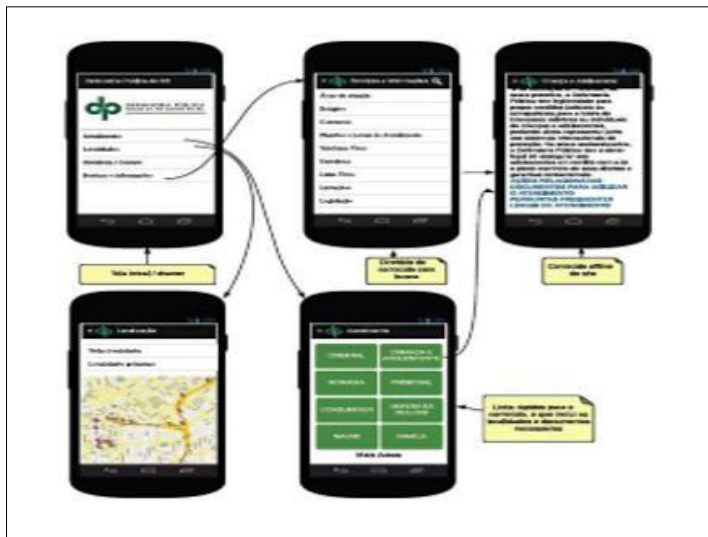
Os *softwares* desempenham papel fundamental em uma organização, sendo eles para controle de processos um auxílio para tomada de decisões, segurança e até entretenimento. Como *softwares* estão cada vez mais complexos e maiores, houve a necessidade de garantir os prazos, custos e a qualidade destes como produtos. Assim surge a Engenharia de *Software*. Ela consiste em um conjunto de ferramentas e métodos de apoio para orientar a realização das atividades pertinentes a produção de *softwares*, definir as funções e produtos a serem entregues como também a sua qualidade e custo (HIRAMA, 2012).

Engholm Junior (2010) diz que a qualidade de um *software* contempla diversos objetivos de sua construção, em sua maioria são requisitos não-funcionais como extensibilidade, capacidade de manutenção, reutilização de código, desempenho, usabilidade e integridade de dados. Optar por não utilizar alguns dos conceitos de engenharia de *software* é estar submisso a possíveis falhas de execução de projeto tais como dificuldade de manutenção, *softwares* inalteráveis, baixo desempenho, baixa qualidade de código. Na maior parte dos projetos a prototipação e métodos de testes são o suficiente para garantir a qualidade do produto a ser entregue.

Por diversas vezes o cliente define um conjunto de requisitos e objetivos gerais que um software deveria ter, mas não especificou os dados de entrada, processamento e saída de dados, não entregando a certeza das suas necessidades perante a análise do desenvolvedor para o produto. Há casos que o desenvolvedor não tem a certeza da eficácia do software ou a forma como o usuário vai interagir com o sistema, nesses casos e em diversos outros relacionados a requisitos e necessidades do cliente a prototipação é a melhor abordagem (SOMMERVILLE, 2011).

Pressman (1995) define que a prototipação é um processo essencial para demonstrar conceitos, ajudar no levantamento de informações sobre os desejos do cliente, validação de requisitos, estudar soluções e apoiar o projeto de interface do usuário. A prototipagem pode acontecer de várias formas, sendo um desenho feito em papel ou softwares específicos e interativos que dão uma melhor ideia da usabilidade do usuário, respeitando as características do produto, mas sendo apenas uma concepção da abstração feita na análise. Exemplo das telas na figura 43.

**Figura 43 - Protótipo de Aplicativo Para Defensoria Pública**



Fonte: Defensoria Pública do RS (2015).

A estratégia de teste é essencial para certificar a qualidade e o atendimento dos requisitos levantado na análise do projeto. Para testar um software ou aplicação é preciso criar um projeto de caso de testes, que são trajetos que devem ser repetidos dentro do sistema, muitas vezes chamado Verificação e Validação, uma estratégia de testes tem que responder as perguntas: Será que o produto está sendo criando

corretamente? Será que o produto criado é o correto? Os testes devem ser executados pelo próprio desenvolvedor ou por um grupo que tenha em mãos um escopo de todos os casos de uso ou funções que esse software precisa atender (PRESSMAN, 2011).

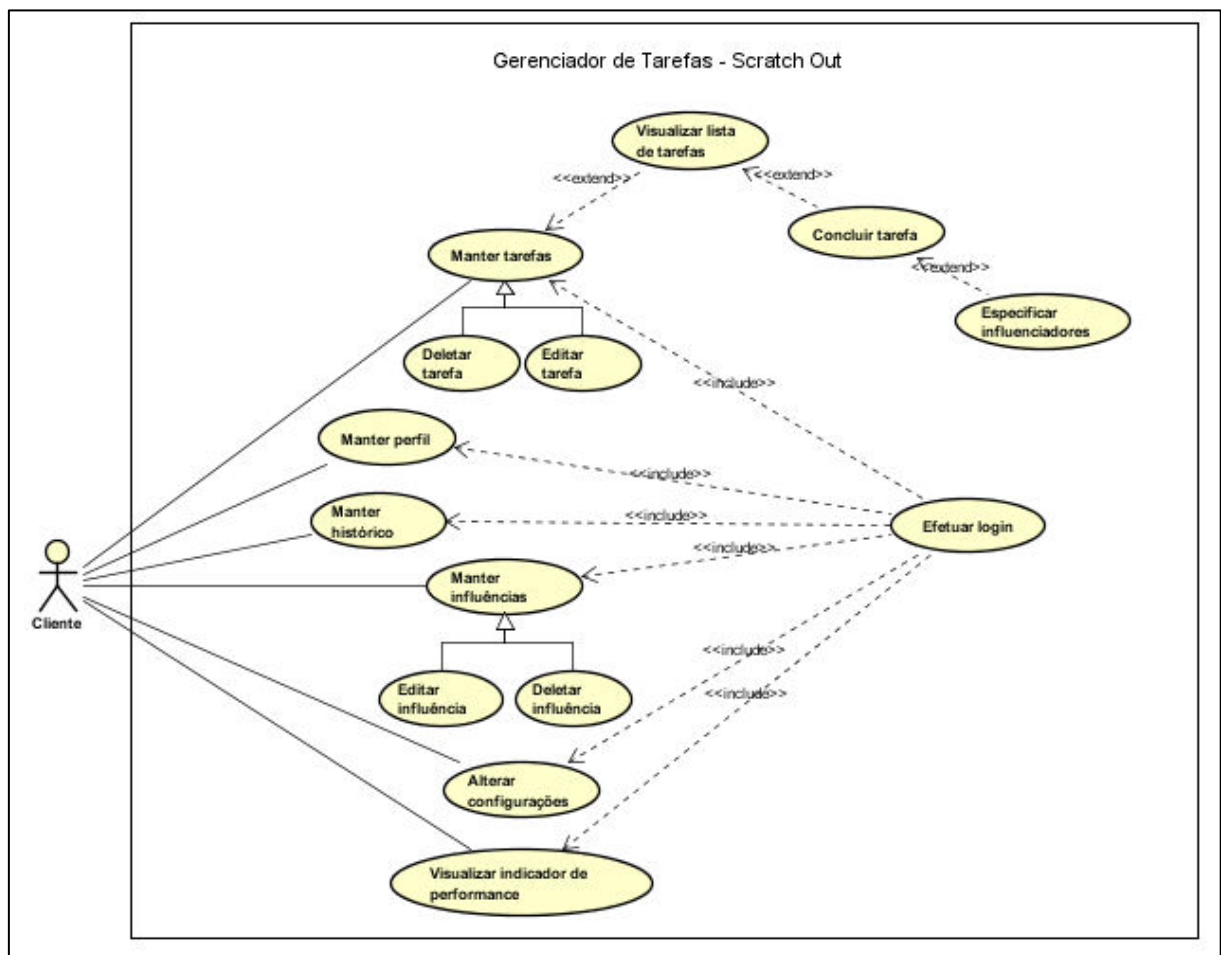
### 3 DESENVOLVIMENTO

Nesse capítulo será abordado o desenvolvimento da aplicação Scratch Out, através dos diagramas desenvolvidos no estudo de UML, a formação do banco de dados e as tecnologias aplicadas para a construção do aplicativo.

#### 3.1 Diagrama de Casos de Uso

O diagrama de casos de uso abaixo, na figura 44, expõe a interação do cliente com a aplicação, ilustrando a acessibilidade e as ações que o usuário poderá realizar através dos casos de uso e suas relações.

Figura 44 - Diagrama de Casos de Uso



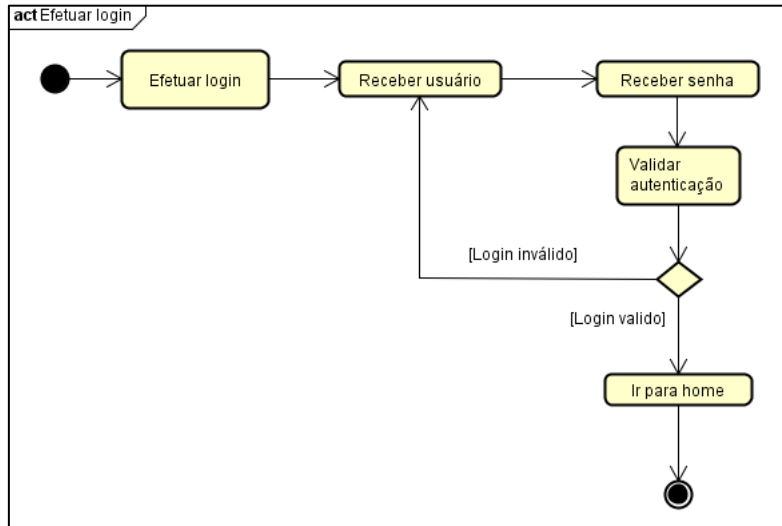
Fonte: Autoria própria, 2018.

#### 3.2 Diagrama de Atividades

Os diagramas de atividades foram divididos em quatro, os quais ilustram as atividades efetuar *login*, manter influências, manter perfil e manter tarefas.

A figura 45 descreve, através das atividades, o fluxo para efetuar o *login*, com a inserção de dados e a validação para autenticar o usuário.

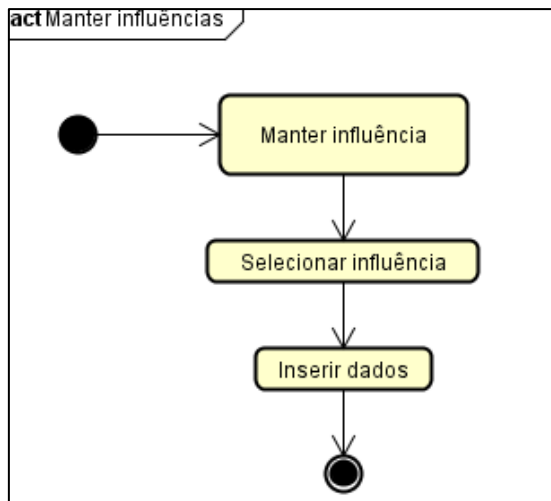
**Figura 45 - Diagrama de Atividade: Efetuar Login**



Fonte: Autoria própria, 2018.

A figura 46 é o diagrama de atividades para manter influências, ilustrando o fluxo de atividades para a inserção de uma influência.

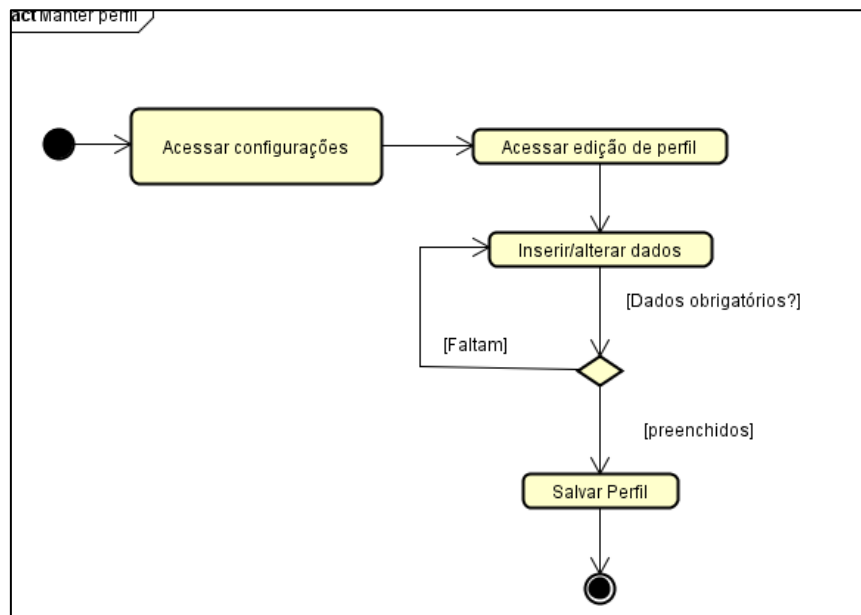
**Figura 46 - Diagrama de Atividades: Manter Influências**



Fonte: Autoria própria, 2018.

A figura 47 apresenta o fluxo de atividades para realizar alterações nos dados de um perfil, através do acesso às configurações.

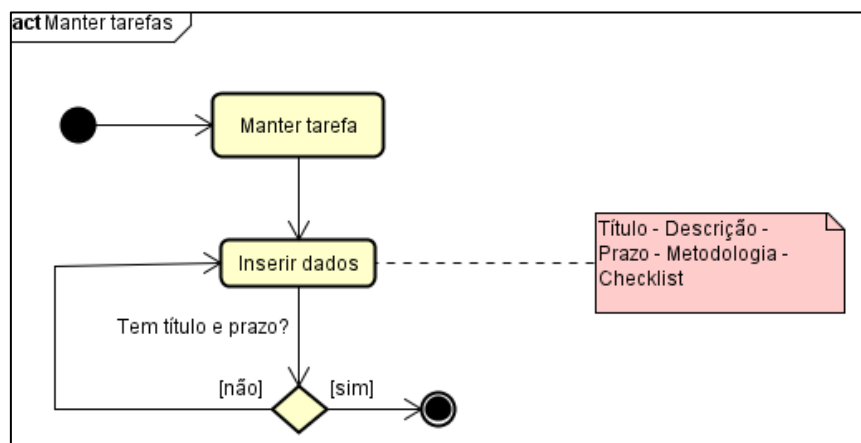
**Figura 47 - Diagrama de Atividades: Manter Perfil**



Fonte: Autoria própria, 2018.

Na figura 48, o diagrama de atividades manter tarefas ilustra o fluxo da criação de uma tarefa.

**Figura 48 - Diagrama de Atividades: Manter Tarefa**

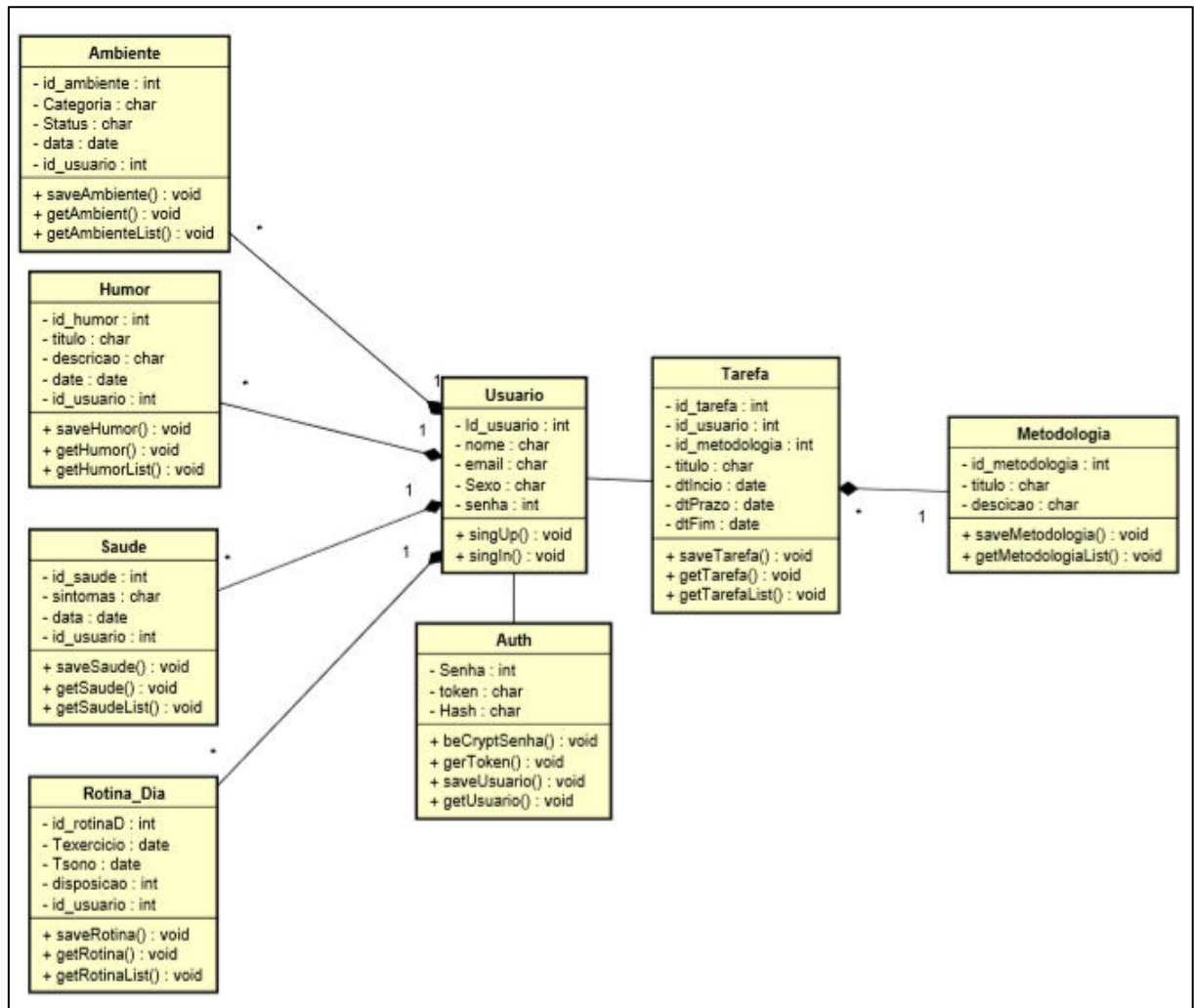


Fonte: Autoria própria, 2018.

### 3.3 Diagrama de Classes

O diagrama de classes, na figura 49, contém no total oito classes, que são as utilizadas na aplicação e no banco, exceto *auth*, que é utilizada apenas na aplicação por ser classe de autenticação.

Figura 49 - Diagrama de Classes



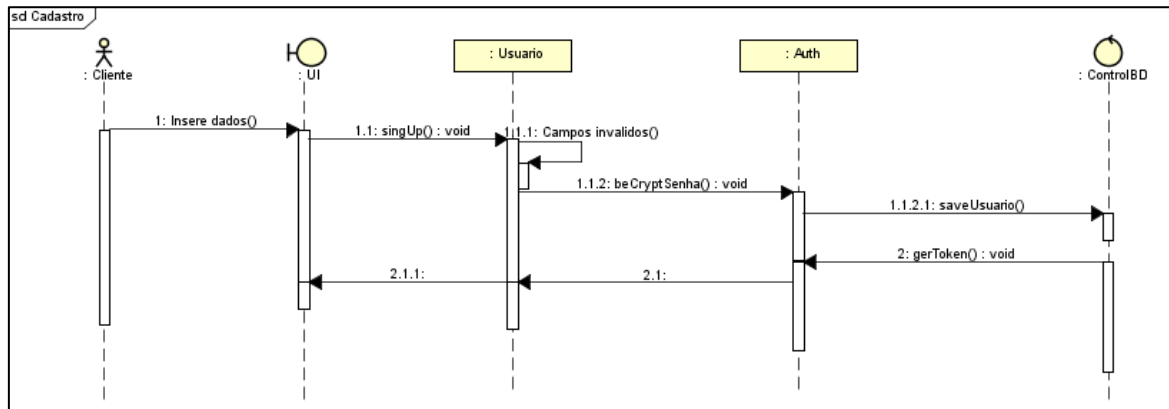
Fonte: Autoria própria, 2018.

### 3.4 Diagrama de Sequência

Os diagramas de sequência compõem no total cinco diagramas, os quais são cadastro, indicador de performance, influências, *login* e tarefa.

Na figura 50, o diagrama descreve a sequência da realização de cadastro na aplicação, ilustrando como esse processo ocorre da interface, passando pelas classes ao banco.

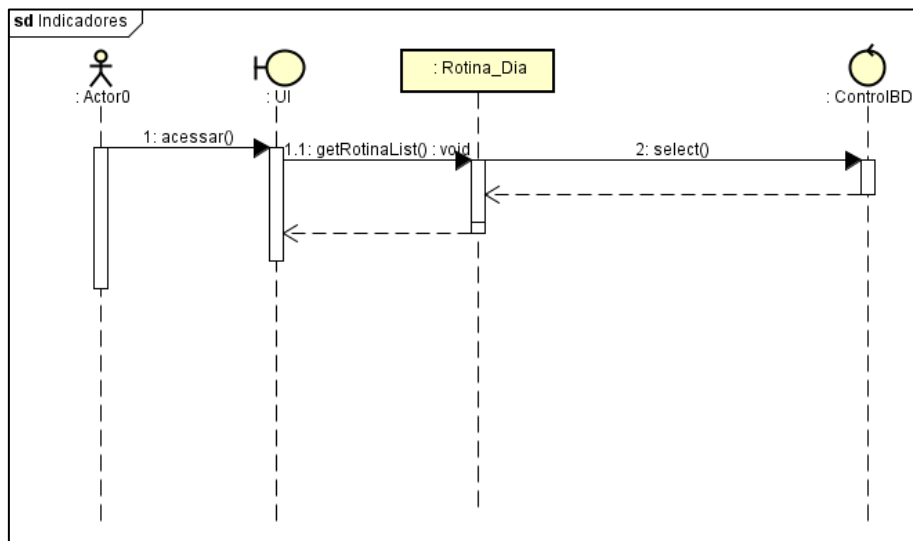
**Figura 50 – Diagrama de Sequência: Cadastro**



Fonte: Autoria própria, 2018.

Na figura 51, o diagrama ilustra o processo de formação do indicador de performance, que é formado a partir das inserções de dados do usuário e levado ao banco.

**Figura 51 – Diagrama de Sequência: Indicador de Performance**

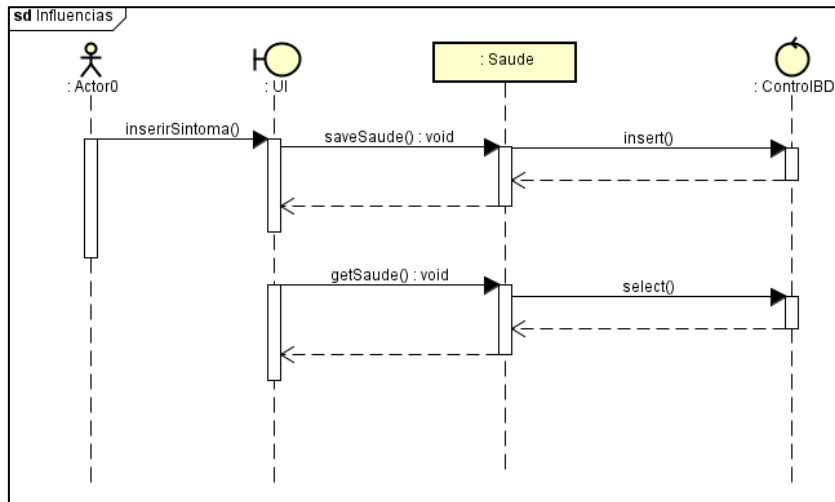


Fonte: Autoria própria, 2018.

Na figura 52, a sequência de inserção de influências do usuário. Em seu processo, os dados inseridos na interface passam pela classe antes de chegar ao banco.



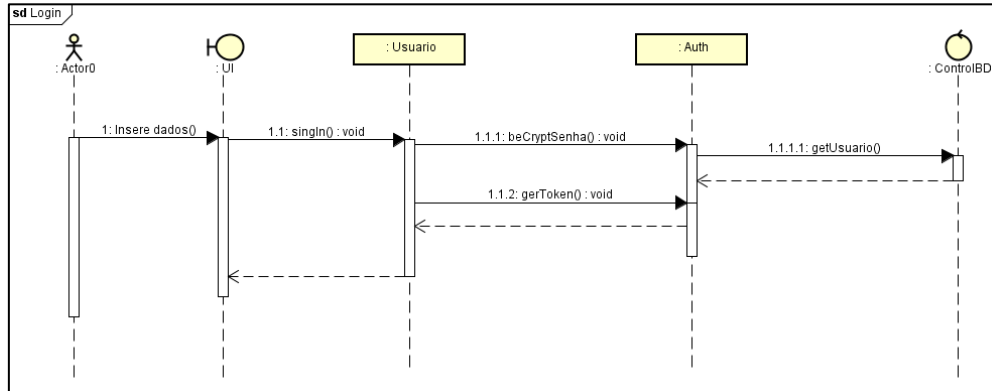
**Figura 52 – Diagrama de Sequência: Influências**



Fonte: Autoria própria, 2018.

A sequência de autenticação do *login* é descrita na figura 53. O processo é ilustrado com os métodos que passam da interface para as classes, até chegar ao banco e retornar os resultados.

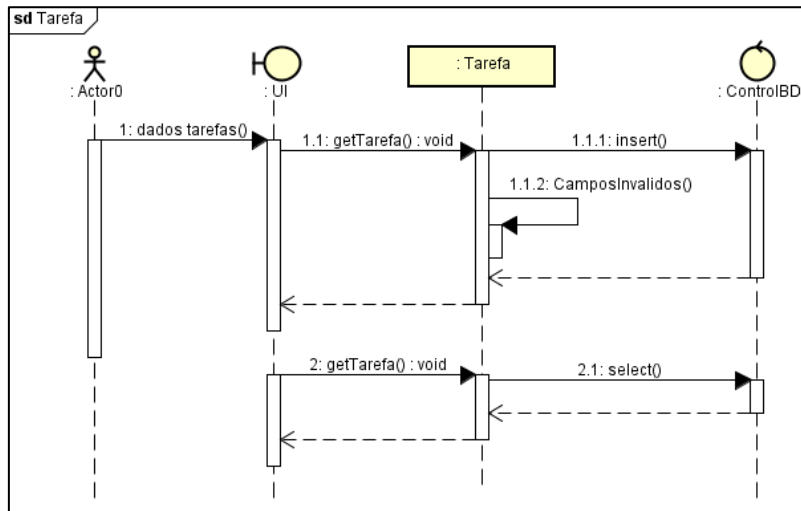
**Figura 53 - Diagrama de Sequência: Login**



Fonte: Autoria própria, 2018.

A criação de uma tarefa pelo usuário é sequenciada na figura 54, que não apenas ilustra o processo de criação, mas o que ocorre quando ocorre invalidação dos campos que devem ser preenchidos.

**Figura 54 - Diagrama de Sequência: Tarefa**

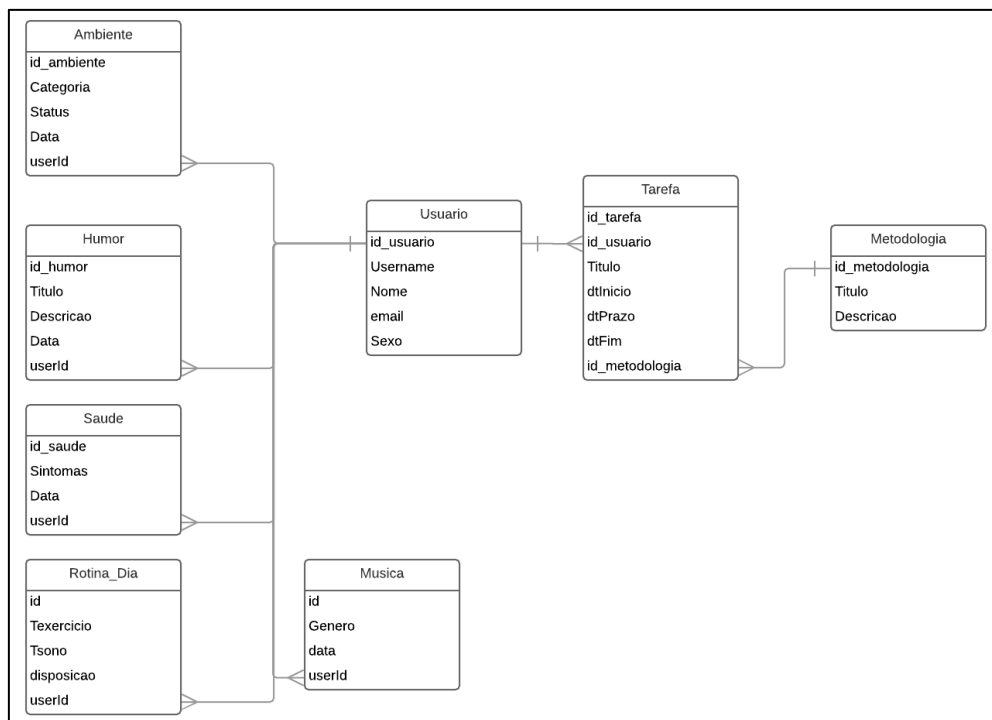


Fonte: Autoria própria, 2018.

### 3.5 DER

O diagrama entidade relacionamento da aplicação é composto por oito entidades, as quais estão diretamente ou indiretamente relacionadas com a entidade usuário.

**Figura 55 - DER**



Fonte: Autoria própria, 2018.

### 3.6 Aplicação

O aplicativo é composto em um total de 12 telas, que interagem entre si de acordo com as ações do usuário.

Após a tela de *splash*, figura 56, que inicia a aplicação, aparecerá a tela Hall, figura 57, que é a tela onde as opções de cadastro e *login* são expostas ao usuário.

**Figura 56 - Tela de *Splash***



Autoria própria, 2018.

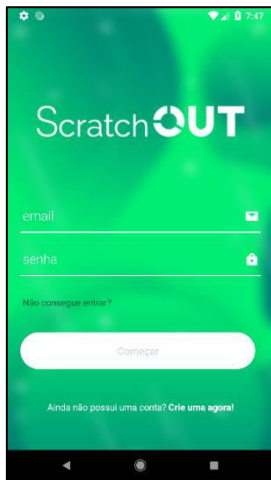
**Figura 57 - Tela Hall**



Autoria própria, 2018.

Quando cadastrado, o usuário entra utilizando o e-mail e senhas inseridos no cadastro.

**Figura 58 - Tela de *Login***



Autoria própria, 2018.

Efetuada o *login*, a aplicação entrará na tela Home, que irá expor um calendário interativo, e um gráfico de progresso que indicará quantas tarefas foram concluídas, junto a uma mensagem de quantas estão pendentes.

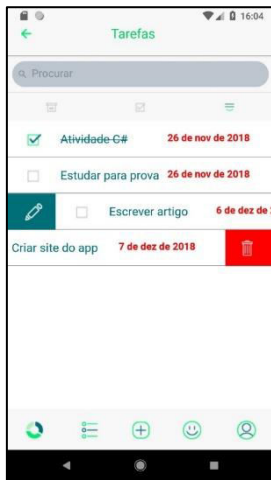
**Figura 59 - Home**



Autoria própria, 2018.

A tela de lista de tarefas, na figura 60, exibirá as tarefas criadas pelo usuário, bem como as opções de filtro e exclusão dessas tarefas. Ao arrastar a tarefa para os lados, serão exibidas as opções de exclusão e edição.

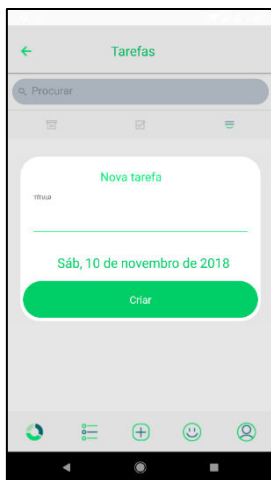
**Figura 60 - Lista de Tarefas**



Autoria própria, 2018.

A criação das tarefas é realizada por meio de um *modal*, o qual recebe o título e prazo da tarefa.

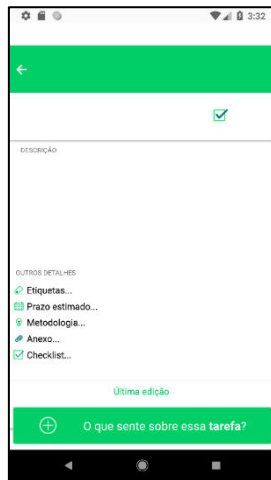
**Figura 61 - Criação de Tarefas**



Autoria própria, 2018.

A tela de inclusão de detalhes e edição de tarefas permite que o usuário insira descrição, metodologia, altere o prazo estimado, anexe links ou documentos, além de possibilitar a criação de uma *checklist* específica. Também é possível que o usuário insira o seu humor em relação a tarefa específica.

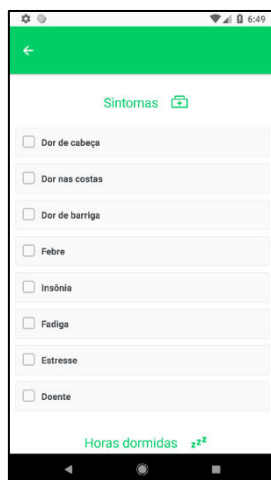
**Figura 62 - Tela de Inclusão de Detalhes e Edição de Tarefas**



Autoria própria, 2018.

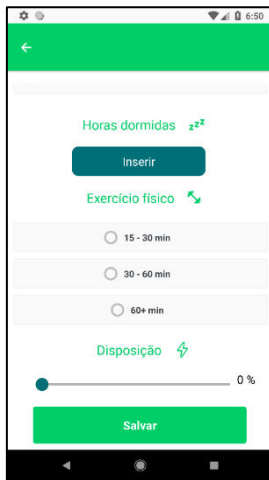
As figuras 63 e 64, respectivamente, referem-se as telas de inserção de influências do usuário, tais como os sintomas, as horas dormidas, o período de exercício praticado se houver, e a disposição mensurada pelo usuário.

**Figura 63 – Inserção de influências**



Autoria própria, 2018

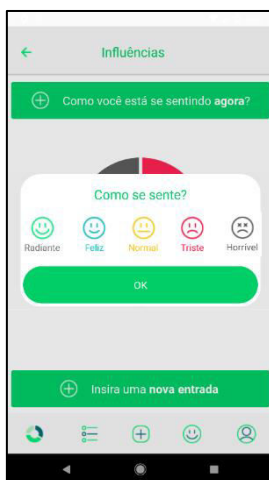
**Figura 64 – Inserção de influências (2)**



Autoria própria, 2018.

A figura 65 refere-se à inserção de humor pelo usuário. Tanto o humor, quanto as demais influências citadas anteriormente, são dados utilizados para a formulação dos gráficos indicadores.

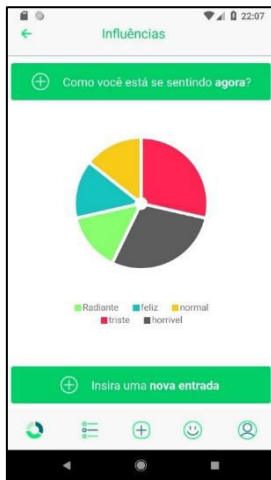
**Figura 65 - Humor**



Autoria própria, 2018.

A figura 66 exibe o gráfico de setores localizado na tela de influências, que exibirá a frequência mensal de humor do usuário.

**Figura 66 – Gráfico de setores para frequência de humor**



Autoria própria, 2018.

As figuras 67 e 68, respectivamente, são a tela de indicador de performance. Nessa tela estão os gráficos formulados com os dados do usuário. Cada gráfico é um indicador específico.

Na figura 67, o gráfico de linhas para frequência de humor exibe as inserções do usuário de acordo com a semana, enquanto o gráfico de barras abaixo compara a quantidade de tarefas concluídas por semana do mês corrente.

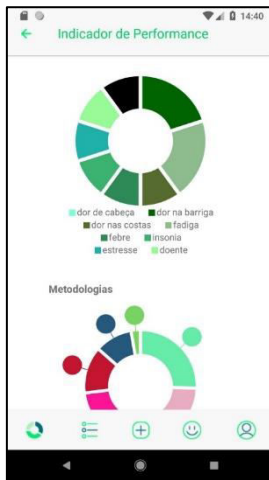
**Figura 67 – Indicador de Performance**



Autoria própria, 2018.

Na figura 68, o gráfico de setores refere-se aos sintomas inseridos pelo usuário, e exibe a frequência dos sintomas mais inseridos. Abaixo, o gráfico de metodologias exibe os métodos mais inseridos pelo usuário.



**Figura 68 – Indicador de Performance (2)**

Autoria própria, 2018.

A tela de configurações exibe opções para ajustar notificações do aplicativo, perfil e privacidade. Em conta, o usuário poderá alterar senha ou e-mail, e em ajuda, obter orientações sobre o uso da aplicação.

**Figura 69 – Tela de Configurações**

Autoria própria, 2018.

## 4 CONCLUSÃO

O estudo realizado em UML tornou possível um planejamento mais robusto para o desenvolvimento da aplicação. Os diagramas de casos de uso, atividades, classes e sequência orientaram e construíram uma visão do aplicativo em questões como *design*, banco de dados, interação do usuário, lógica e algoritmo.

O *framework* React Native foi fundamental para que o projeto fosse desenvolvido. Foram utilizadas bibliotecas para componentes visuais, rota entre telas, autenticação de *login* e gráficos.

Cumprindo o propósito, o desenvolvimento alcançou a construção de gráficos estatísticos indicadores, que se formam a partir dos dados inseridos pelo usuário, como as influências e as tarefas, exibindo correlações e frequências, além da criação de tarefas em si, em uma interface intuitiva e simples para facilitar e estimular a gestão pessoal do usuário.

A aplicação foi submetida a testes com alunos do curso de Recursos Humanos, que instalaram e utilizaram a aplicação. Após conhecerem os recursos disponíveis, sugeriram novas implementações e melhorias nas funcionalidades já existentes. A experiência indica adaptações para que a aplicação seja utilizável por funcionários em ambiente de trabalho em prol da gestão de recursos humanos, além de novas pesquisas e aprimorar.

O projeto serviu de base para o desenvolvimento de um artigo para o Congresso Internacional de Tecnologia e Gestão (CITEG), no qual foram sintetizados o desenvolvimento e a pesquisa realizados no trabalho. Além da aprovação do artigo, houve a participação presencial no congresso que ocorreu na FATEC Rubens Lara, para a apresentação do artigo inscrito.

O interesse de terceiros na aplicação desenvolvida estimulou a continuidade do projeto. A marca criada para a aplicação pretende expandir com a criação de novos produtos, como um sistema *desktop* sofisticado para ambientes corporativos, bem como a criação de uma empresa para distribuição e monetização dos produtos desenvolvidos.

## REFERÊNCIAS

- ABRANTES, Talita. **10 apps para organizar o tempo e a vida no trabalho**. Disponível em: <<https://exame.abril.com.br/carreira/10-apps-para-organizar-o-tempo-e-a-vida-no-trabalho/>>. Acesso em: 13 set. 2016.
- BEZERRA, Eduardo. **Princípios de análise e projeto de sistemas com UML**. 3. ed. Rio de Janeiro: Elsevier, 2015.
- BLAHA, Michael; RUMBAUGH, James. **Modelagem e projetos baseados em objetos com UML 2**. 2. ed. Rio de Janeiro: Elsevier, 2006.
- BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML: Guia do Usuário**. 2. ed. Rio de Janeiro: Elsevier, 2012.
- BRITO, Edivaldo. **Com OptimizeMe, monitore o dia a dia e aumente a qualidade de vida**. 2014. Disponível em: <<http://www.techtudo.com.br/tudo-sobre/optimizeme.html>>. Acesso em: 16 abr. 2014.
- BUSSAB, Wilton O.; MORETTIN, Pedro A.. **Estatística Básica**. 9. ed. São Paulo: Saraiva, 2017. 576 p.
- CRESPO, Antônio Arnot. **Estatística Fácil**. 19. ed. São Paulo: Saraiva, 2012. 232 p.
- DATE, C.j. **Introdução a Sistemas de banco de dados**. 8. ed. Rio de Janeiro: Campus Ltda., 1984.
- DEFENSORIA PÚBLICA DO RS (Rio Grande do Sul). Assessoria de Comunicação. **Defensoria Pública contará com aplicativo mobile**. 2015. Disponível em: <<http://www.defensoria.rs.def.br/conteudo/23805>>. Acesso em: 26 nov. 2018.
- DUCKETT, Jon. **HTML & CSS: projete e construa websites**. Rio de Janeiro: Alta Books, 2016. 512 p.
- ELMASRI, Ramez; NAVATHE, Shamkant B.. **Sistemas de Banco de Dados**. 4. ed. São Paulo: Pearson Education do Brasil Ltda, 2005.
- \_\_\_\_\_. **Sistemas de Banco de Dados**. 6. ed. São Paulo: Pearson Education do Brasil Ltda, 2011.
- ENGHOLM JUNIOR, Helio. **Engenharia de Software na Prática**. São Paulo: Novatec, 2010.
- ESTRELLA, Sérgio. **Evernote: o que é e para que serve?** Disponível em: <<https://canaltech.com.br/equipe/sergio-estrella/p2/>>. Acesso em: 02 jun. 2014.
- FÁVERO, Luiz Paulo; BELFIORE, Patrícia. **Manual de Análise de Dados**. São Paulo: Elsevier Editora Ltda, 2017. 1196 p.

FERREIRA, Elcio; EIS, Diego. **HTML5: Curso W3C Escritório Brasil**. São Paulo: W3C Escritório Brasil, 2011. 106 p.

FOWLER, Martin. **UML Essencial: Um breve guia para a linguagem-padrão de modelagem de objetos**. 3. ed. Porto Alegre: Bookman, 2005.

FREEMAN, Elisabeth; FREEMAN, Eric. **Use a Cabeça!: HTML com CSS e XHTML**. 2. ed. Rio de Janeiro: Alta Books, 2008. 616 p. (Use a Cabeça!).

GÓES, Wilson Moraes. **Aprenda UML por meio de estudos de caso**. São Paulo: Novatec, 2014.

GUEDES, Gilleanes T.a. **UML 2: Uma abordagem prática**. 2. ed. São Paulo: Novatec, 2011.

KAPLAN, R.S.; NORTON, D.P. **A estratégia em ação: balanced scorecard**. 4. ed. Rio de Janeiro: Campus, 1997.

HEUSER, Carlos Alberto. **Projeto de Banco de Dados**. 4. ed. Porto Alegre: Sagra Luzzatto, 1998.

HIRAMA, Kechi. **Engenharia de software: Qualidade e produtividade com tecnologia**. Rio de Janeiro: Elsevier Editora Ltda, 2012.

LIETTI, Tamires. **10 aplicativos para manter o seu dia a dia organizado**. Disponível em: <<https://revistapegn.globo.com/Administracao-de-empresas/noticia/2016/07/10-aplicativos-para-manter-seu-dia-dia-organizado.html>>. Acesso em: 13 jul. 2016.

MACHADO, Felipe; ABREU, Mauricio. **Projeto de Banco de Dados: Uma visão prática**. 11. ed. São Paulo: Editora Erica Ltda, 2004.

MAZZA, Lucas. **HTML5 e CSS3: Domine a Web do Futuro**. São Paulo: Casa do Código, 2012. 205 p.

MCFARLAND, David Sawyer. **CSS3: O Manual que Faltava**. Rio de Janeiro: Alta Books, 2015. 640 p.

MUNHOZ, Vinicius. **Tchau, desorganização: confira 10 apps para aumentar sua produtividade**. Disponível em: <<https://www.tecmundo.com.br/apps/79255-tchau-desorganizacao-confira-10-apps-aumentar-productividade.htm>>. Acesso em: 30 abr. 2015.

OLIVIERO, Carlos Antonio José. **Faça um Site: JavaScript - Orientado por Projeto**. São Paulo: Érica, 2001. 264 p. (Faça um Site).

PRESSMAN, Roger S.. **Engenharia de software**. 3. ed. São Paulo: Makron Books do Brasil Ltda., 1995.

\_\_\_\_\_. **Engenharia de software: Uma abordagem profissional**. 7. ed. Porto Alegre: Amgh Editora Ltda., 2011.

SETZER, Valdemar W.; SILVA, Flávio Soares Corrêa da. **Banco de Dados: Aprenda o Que São, Melhore Seu Conhecimento, Construa Os Seus**. São Paulo: Edgar Blucher, 2005.

SILBERSCHATZ, Abraham; KORTH, Henry F.; SUDARCHAN, S.. **Sistemas de Banco de Dados**. 5. ed. Rio de Janeiro: Elsevier Editora Ltda, 2006.

SILVA, Maurício Samy. **Fundamentos de HTML5 e CSS3**. São Paulo: Novatec, 2015. 302 p.

\_\_\_\_\_. **JavaScript: Guia do Programador**. São Paulo: Novatec, 2010.

SILVA, Ricardo Pereira e. **Como modelar com UML 2**. Florianópolis: Visual Book, 2009.

SOMMERVILLE, Ian. **Engenharia de software**. 9. ed. São Paulo: Pearson Education do Brasil Ltda, 2011.