

**CENTRO PAULA SOUZA  
FACULDADE DE TECNOLOGIA DE FRANCA  
“Dr. THOMAZ NOVELINO”**

**TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS**

**GETÚLIO VINICIUS TEIXEIRA DA SILVA**

**DESENVOLVIMENTO DE SOFTWARE**  
Aspectos Gerais e Documentação

**FRANCA/SP  
2018**

**GETÚLIO VINÍCIUS TEIXEIRA DA SILVA**

**DESENVOLVIMENTO DE SOFTWARE**

Aspectos Gerais e Documentação

Trabalho de Graduação apresentado à Faculdade de Tecnologia de Franca - "Dr. Thomaz Novelino", como parte dos requisitos obrigatórios para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador: Dra. Jaqueline Brigladori Pugliesi

**FRANCA/SP**

**2018**

**GETÚLIO VINÍCIUS TEIXEIRA DA SILVA**

**DESENVOLVIMENTO DE SOFTWARE**

Trabalho de Graduação apresentado à Faculdade de Tecnologia de Franca - “Dr. Thomaz Novelino”, como parte dos requisitos obrigatórios para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Trabalho avaliado e aprovado pela seguinte Banca Examinadora:

Orientador(a) ..... : \_\_\_\_\_.

Nome..... : Dr. Jaqueline Brigladori Pugliesi

Instituição ..... : Faculdade de Tecnologia de Franca – “Dr. Thomaz Novelino”.

Examinador(a) 1.. : \_\_\_\_\_.

Nome..... : Examinador\_1.

Instituição ..... : Instituição\_1.

Examinador(a) 2.. : \_\_\_\_\_.

Nome..... : Examinador\_2.

Instituição ..... : Instituição\_2.

**Franca, \_\_ de junho de 2018.**

## AGRADECIMENTO

Xxxxxxxxxxxxxx

Dedico o presente Trabalho de Graduação a Deus e aos meus familiares, em especial à minha esposa (Nome) e filhos (Nomes).

*Aqui está uma pergunta: quando foi a última vez que você ouviu um argumento, e com base nesse argumento, mudou de ideia? Não apenas sobre algo que você realmente não pensou muito, mas algo que, antes de considerar o argumento em questão, sentiu-se bastante certo em relação à sua posição original.*

*Em outras palavras, quando foi a última vez que percebeu que estava completamente errado em uma questão de opinião?*

*Se a sua resposta é "nunca", ou mesmo "há muito tempo", é porque você está sempre certo?*

John Gruber

## RESUMO

O vídeo fornece uma maneira poderosa de ajudá-lo a provar seu argumento. Ao clicar em Vídeo Online, você pode colar o código de inserção do vídeo que deseja adicionar. Você também pode digitar uma palavra-chave para pesquisar online o vídeo mais adequado ao seu documento. Para dar ao documento uma aparência profissional, o Word fornece designs de cabeçalho, rodapé, folha de rosto e caixa de texto que se complementam entre si. Por exemplo, você pode adicionar uma folha de rosto, um cabeçalho e uma barra lateral correspondentes. Clique em Inserir e escolha os elementos desejados nas diferentes galerias. Temas e estilos também ajudam a manter seu documento coordenado.

Palavras-chave: Markdown. Documentação. Projeto de Software. Trabalho de Graduação.

### ABSTRACT

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas porttitor congue massa. Fusce posuere, magna sed pulvinar ultricies, purus lectus malesuada libero, sit amet commodo magna eros quis urna. Nunc viverra imperdiet enim. Fusce est. Vivamus a tellus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Proin pharetra nonummy pede. Mauris et orci. Aenean nec lorem. In porttitor. Donec laoreet nonummy augue. Suspendisse dui purus, scelerisque at, vulputate vitae, pretium mattis, nunc

**Keywords:** GPI. NBR 6.028. Pre-text. Summary. GPI Homework



## LISTA DE FIGURAS

Figura 1 - Um modelo geral do processo de projeto. ....	20
Figura 2 - Camadas da ES. ....	21
Figura 3 - Modelo Cascata.....	23
Figura 4 - O processo da <i>Extreme Programming (XP)</i> . ....	26
Figura 5 - O processo <i>Scrum</i> .....	26
Figura 6 - Criação de um framework.....	27
Figura 7 - Exemplo de <i>Kanban</i> com tarefas. ....	28
Figura 8 - Exemplo do sistema bancário com caso de uso e atores. ....	31
Figura 9 - Fluxo de mensagens conectadas a objetos de fluxo em duas piscinas. ....	32
Figura 10 - Diagrama Entidade Relacionamento (DER) para sistema de farmácias. ....	33
Figura 11 - Protótipo de tela de <i>login</i> com <i>mockup</i> .....	36
Figura 12 - Chamada ao <i>Man</i> para exibição da documentação do <i>git</i> . ....	41
Figura 13 - Documentação do <i>Git</i> exibida através do <i>Man</i> . ....	41
Figura 14 - Trecho da documentação do <i>framework Vue.js</i> . ....	42
Figura 15 - Links para documentação colaborativa do software Inkscape. ....	43
Figura 16 - Chamada ao navegador <i>Lynx</i> .....	46
Figura 17 - Navegador <i>web Lynx</i> exibindo a página inicial do site <i>daringfireball.net</i> .....	47
Figura 18 - Formato de <i>e-mail</i> de texto simples. ....	49
Figura 19 - <i>Markdown</i> : Saída HTML renderizada - Parágrafos e linhas. ....	51
Figura 20 - <i>Markdown</i> : Saída HTML renderizada - HTML em Bloco. ....	52
Figura 21 - <i>Markdown</i> : Saída HTML renderizada - HTML em linha. ....	52
Figura 22 - <i>Markdown</i> : Saída HTML renderizada - Ênfase.....	53
Figura 23 - <i>Markdown</i> : Saída HTML renderizada - Cabeçalho modelo <i>setext</i> . ....	55
Figura 24 - <i>Markdown</i> : Saída HTML renderizada - Cabeçalho modelo <i>atx</i> .....	56
Figura 25 - <i>Markdown</i> : Saída HTML renderizada - Listas ordenadas.....	57
Figura 26 - <i>Markdown</i> : Saída HTML renderizada - Lista não ordenada.....	58
Figura 27 - <i>Markdown</i> : Saída HTML renderizada - Listas ordenadas e não ordenadas.....	59
Figura 28 - <i>Markdown</i> : Saída HTML renderizada - Citação em bloco.....	60
Figura 29 - <i>Markdown</i> : Saída HTML renderizada - Citação em bloco composta por mais elementos. ....	61
Figura 30 - <i>Markdown</i> : Saída HTML renderizada – <i>Links inline</i> . ....	63
Figura 31 - <i>Markdown</i> : Saída HTML renderizada – <i>Links</i> por referência.....	64
Figura 32 - <i>Markdown</i> : Saída HTML renderizada - Imagens. ....	65
Figura 33 - <i>Markdown</i> : Saída HTML renderizada - Código. ....	67
Figura 34 - Conversão de texto em <i>Markdown</i> para HTML utilizando <i>Markdown.pl</i> 1.0.1.....	68
Figura 35 - Conversão de texto no formato <i>Markdown</i> para <i>DOCX</i> com o software <i>Pandoc</i> . ....	69
Figura 36 - Texto convertido pelo <i>Pandoc</i> de <i>Markdown</i> para <i>DOCX</i> . ....	70
Figura 37 - <i>Markdown</i> : Saída HTML renderizada - Tabela.....	73
Figura 38 - Sintaxe <i>Markdown</i> na formatação do texto de descrição da tarefa no <i>Trello</i> . ....	76
Figura 39 - Descrição da tarefa feita com <i>Markdown</i> renderizada em HTML no cartão do <i>Trello</i> . ....	77
Figura 40 - Abertura de <i>issue</i> no <i>GitHub</i> utilizando <i>Markdown</i> para formatar o texto. ....	78
Figura 41 - Visualização da <i>issue</i> formatada com <i>Markdown</i> após a publicação. ....	79
Figura 42 - Recorte de tela da página inicial do repositório <i>Laravel</i> armazenado no <i>GitHub</i> .....	80
Figura 43 - Página inicial da <i>wiki</i> do projeto <i>HHVM</i> armazenado em um repositório do <i>GitHub</i> .....	81
Figura 44 - Visualização inicial de um site estático com a <i>template</i> padrão do <i>Jekyll</i> .....	82
Figura 45 - Visualização inicial de um site estático com a <i>template</i> padrão do <i>MkDocs</i> . ....	83

<b>Figura 46</b> - Comparação entre a estrutura de um projeto <i>Jekyll</i> e um projeto <i>MkDocs</i> . .....	84
<b>Figura 47</b> - Exemplo da estrutura de diretórios e arquivos de um projeto com <i>MkDocs</i> . .....	87
<b>Figura 48</b> - Exemplo de site estático criado com <i>MkDocs</i> . .....	88
<b>Figura 49</b> - Resposta a uma requisição feita com verbo HTTP GET para um recurso da API. ....	91
<b>Figura 50</b> - <i>Trello</i> : tarefas planejadas inicialmente em 23/02/2018. ....	94
<b>Figura 51</b> - <i>Trello</i> : evolução das tarefas em 30/03/2018. ....	95
<b>Figura 52</b> - <i>Trello</i> : evolução das tarefas em 11/05/2018. ....	95

## LISTA DE QUADROS

<b>Quadro 1</b> - Exemplo de requisitos para o sistema de <i>software</i> de bomba de insulina. ....	29
<b>Quadro 2</b> - Uma especificação estruturada de um requisito para uma bomba de insulina.....	30
<b>Quadro 3</b> - Modelo Conceitual de um cadastro de produtos.....	34
<b>Quadro 4</b> - <i>Markdown</i> : Sintaxe de formatação - Parágrafos e linhas.....	50
<b>Quadro 5</b> - <i>Markdown</i> : Sintaxe de formatação - HTML em bloco.....	51
<b>Quadro 6</b> - <i>Markdown</i> : Sintaxe de formatação - HTML em linha.....	52
<b>Quadro 7</b> - <i>Markdown</i> : Sintaxe de formatação - Ênfase. ....	53
<b>Quadro 8</b> - <i>Markdown</i> : Sintaxe de formatação - Cabeçalhos do modelo <i>setext</i> .....	54
<b>Quadro 9</b> - <i>Markdown</i> : Sintaxe de formatação - Cabeçalho no modelo <i>atx</i> . ....	55
<b>Quadro 10</b> - <i>Markdown</i> : Sintaxe de formatação - Listas ordenadas. ....	56
<b>Quadro 11</b> - <i>Markdown</i> : Sintaxe de formatação - Listas não ordenadas. ....	57
<b>Quadro 12</b> - <i>Markdown</i> : Sintaxe de formatação - Listas ordenadas e não ordenadas. ....	58
<b>Quadro 13</b> - <i>Markdown</i> : Sintaxe de formatação - Citação em Bloco. ....	60
<b>Quadro 14</b> - <i>Markdown</i> : Sintaxe de formatação - Citação em Bloco composta por mais elementos. ....	61
<b>Quadro 15</b> - <i>Markdown</i> : Sintaxe de formatação – <i>Links inline</i> .....	62
<b>Quadro 16</b> - <i>Markdown</i> : Sintaxe de formatação - <i>Links</i> por referência. ....	63
<b>Quadro 17</b> - <i>Markdown</i> : Sintaxe de formatação - Imagens.....	64
<b>Quadro 18</b> - <i>Markdown</i> : Sintaxe de formatação - Código. ....	66
<b>Quadro 19</b> - <i>Markdown</i> : Sintaxe de formatação - Caractere de escape. ....	67
<b>Quadro 20</b> - <i>Markdown</i> : Sintaxe de formatação - Tabelas.....	72
<b>Quadro 21</b> - Processo de instalação do <i>MkDocs</i> no SO <i>Debian GNU/Linux</i> . ....	84
<b>Quadro 22</b> - Arquivo de parâmetros para geração de sites estáticos com <i>MkDocs</i> .....	86
<b>Quadro 23</b> - Tecnologias utilizadas para o desenvolvimento e implantação da aplicação. ....	92
<b>Quadro 24</b> - Tecnologias utilizadas durante o desenvolvimento da aplicação. ....	93
<b>Quadro 25</b> - Tecnologias utilizadas para documentar a aplicação. ....	93

**LISTA DE TABELAS**

LISTA DE SIGLAS

**XXX** – XXXXXXXXXXXXX  
**YYY** – YYYYYYYYYYYYYYYY

## SUMÁRIO

<b>1 INTRODUÇÃO .....</b>	<b>16</b>
<b>2 PONTO DE PARTIDA - FUNDAMENTAÇÃO .....</b>	<b>18</b>
2.1 PROJETO DE SOFTWARE .....	19
2.2 ENGENHARIA DE SOFTWARE .....	20
2.2.1 Metodologias .....	22
2.2.1.1 Métodos prescritivos.....	22
2.2.1.2 Métodos ágeis .....	24
2.2.1.3 Extreme Programming, Scrum e Kanban .....	25
2.2.2 Ferramentas .....	28
2.2.2.1 Linguagem natural e linguagem natural estruturada .....	29
2.2.2.2 UML e BPMN .....	30
2.2.2.3 Banco de dados .....	33
2.2.2.4 Prototipação .....	35
2.2.3 Qualidade de software.....	36
<b>3 DOCUMENTAÇÃO DE SOFTWARE.....</b>	<b>38</b>
3.1 DOCUMENTAÇÃO DO PROJETO DE SOFTWARE .....	38
3.2 DOCUMENTAÇÃO DE SOFTWARE PARA USUÁRIOS .....	39
3.2.1 Apresentação de funcionalidades e documentação operacional.....	40
3.2.2 Documentação colaborativa .....	42
3.2.3 Segurança da informação .....	43
3.3 GERENCIAMENTO DA DOCUMENTAÇÃO .....	44
<b>4 LINGUAGEM MARKDOWN .....</b>	<b>45</b>
4.1 LINGUAGEM DE MARCAÇÃO LEVE .....	47
4.2 ARQUIVOS MARKDOWN.....	48
4.3 SINTAXE .....	48
4.3.1 Parágrafos e linhas .....	50
4.3.2 HTML incorporado.....	51
4.3.3 Ênfase .....	53
4.3.4 Cabeçalhos .....	54
4.3.5 Listas .....	56
4.3.6 Citações em bloco.....	59
4.3.7 Links .....	62
4.3.8 Imagens.....	64
4.3.9 Código .....	65
4.3.10 Caractere de escape .....	67
4.4 PARSERS .....	68
4.4.1 Parser criado por John Gruber .....	68
4.4.2 Novos parsers .....	69
4.4.3 Recursos implementados pelos novos parsers .....	71
4.5 PADRONIZAÇÃO DAS ESPECIFICAÇÕES .....	73
4.6 APLICAÇÃO DA LINGUAGEM MARKDOWN.....	75
4.6.1 Aplicações de uso geral .....	75
4.6.2 Geradores de sites estáticos .....	81
4.6.2.1 MkDocs .....	84
<b>5 APLICAÇÃO PRÁTICA - PROJETO .....</b>	<b>89</b>

5.1 CONCEPÇÃO .....	89
5.1.1 Arquitetura da aplicação.....	89
5.2 PLANEJAMENTO .....	92
5.2.1 Tecnologias .....	92
5.2.2 Metodologia.....	94
5.3 EXECUÇÃO .....	96
5.4 FECHAMENTO .....	96
<b>CONCLUSÃO .....</b>	<b>97</b>
<b>REFERÊNCIAS.....</b>	<b>98</b>
<b>APENDICE A .....</b>	<b>101</b>
<b>APENDICE B .....</b>	<b>104</b>

## 1 INTRODUÇÃO

Seres humanos utilizam programas de computadores para realizarem tarefas que muitas vezes não são capazes de concluir, ou não conseguiriam concluir em espaços de tempo satisfatórios com outras ferramentas.

O uso de papel e caneta, por exemplo, pode não ser suficiente para realizar de modo eficiente cálculos complexos de planejamento da produção de uma fábrica, ou ainda, de modo eficaz a identificação de padrões de som, de temperatura, entre outras atividades. Estes são problemas que necessitam, para melhor solução, do auxílio de ferramentas computadorizadas para o seu processamento.

Em seu curso de programação em linguagem *Python*, (GUANABARA, 2017) observa que em todas as áreas de atividade humana há um crescimento na demanda de programas para computadores e outros tipos de eletrônicos, bem como na procura por profissionais que atuam na área de Tecnologia da Informação (TI), especialmente programadores.

Exemplos...

Em resposta ao crescimento do setor, nota-se o surgimento de iniciativas como a organização sem fins lucrativos *Code.org*, que visa aumentar o número de programadores disponíveis no futuro, além da inclusão de mulheres e minorias sub-representadas no mercado de desenvolvimento de software (CODE.ORG, 2017).

A ação da *Code.org* é concentrada na educação básica, investindo em programas pedagógicos de iniciação a ciência da computação para crianças. O programa de iniciação tem sido adotado por escolas regulares, principalmente nos Estados Unidos e a iniciativa tem recebido investimento de empresas como *Google*, *Facebook* e *Microsoft* (CODE.ORG, 2017).

Tendo sido exposto alguns fatos acerca do cenário atual de desenvolvimento e de perspectivas de expansão deste mercado, em face as ações de organizações como a *Code.org*, pode-se concluir que existe uma dependência de *software* impactando diretamente a vida das pessoas na sociedade e diante dessa demanda surge a necessidade de empenhar esforços em estudos nessa área.

Neste cenário de TI, existe o ramo da Engenharia de *Software* (ES), uma área ampla de atuação que segundo (PRESSMAN, 2011, p. 29), é um processo que utiliza um conjunto de métodos e ferramentas que possibilitam o desenvolvimento de *software* com qualidade, e, segundo (SOMMERVILLE, 2011, p. 5), um dos motivos

**Comentado [GVTDS1]:** Apesar de saber que está deve ser a última etapa, teve um momento que eu não conseguia escrever sobre modelagem e decidi falar algumas coisas que vinham na minha mente como Introdução.

**Comentado [GVTDS2]:** Citar exemplos de uso de software que estão em alta.

**Comentado [GVTDS3]:** Corrigir as ligações e eliminar as redundâncias. Talvez seja necessário apresentar mais fatos. Citar os números do github e de canais no youtube.



pelos quais a ES é importante refere-se ao fato de que “cada vez mais, indivíduos e sociedades dependem dos sistemas de *software* avançados”. Portanto, ainda segundo Sommerville “temos de ser capazes de produzir sistemas confiáveis econômica e rapidamente”.

Adentrando um pouco mais nessa área de ES, nota-se a existência de uma tarefa em específico que demanda especial atenção, a Documentação de *Software*. Portanto, a partir de noções fundamentais de ES que serão abordadas no capítulo 2 (Referencial Teórico), este Trabalho de Graduação (TG) tem como objetivo ser um caso de uso para o desenvolvimento de um tipo específico de *software* cuja documentação será construída com a linguagem *Markdown*.

Assim, o capítulo 2 trará, também, uma visão sobre os aspectos gerais do processo de desenvolvimento e os tipos de documentação dos quais um *software* demanda; o capítulo 3 terá uma introdução a linguagem *Markdown*; o capítulo 4 apresentará uma aplicação prática, ou seja, um projeto de *software* elaborado para que o seu processo de desenvolvimento seja documentado com o uso da linguagem *Markdown*; e por fim o capítulo final deste trabalho apresentará uma conclusão sobre o estudo em torno do processo de documentação de *software* e a aplicação da linguagem *Markdown*.

**Comentado [GVTDS4]:** Adicionar a relação dos Capítulos.  
Não está fechado.

## 2 PONTO DE PARTIDA - FUNDAMENTAÇÃO

A escolha do tema para elaboração do TG foi motivada pela dificuldade em lidar com as tarefas impostas na disciplina de ES do curso de Análise e Desenvolvimento de Sistemas (ADS), especialmente a tarefa de documentação, na qual era necessária a apresentação de um projeto de *software* para acompanhamento dos estudos.

O marco inaugural foi a busca de uma alternativa ao modelo de Documento de Requisitos (DR) apresentado pela instituição, que consistia em um arquivo de texto, no formato “.docx”, contendo uma estrutura determinada para inclusão de conteúdo.

Ao término das atividades, foi obtida uma vasta documentação e não apenas um apanhado de requisitos, conforme o título levava a crer. Assim, o modelo finalizado continha informações oriundas de diversas ferramentas, forjadas através de inúmeras técnicas e, por vezes, abordagens distintas sobre um mesmo ponto do projeto.

Mais adiante no curso, conforme foram sendo apresentados novos conceitos da ES, somou-se ao DR a solicitação de tarefas que consistiam na elaboração de diagramas complementares e outros documentos textuais, em muitos aspectos redundantes.

Havia, portanto, uma grande confusão de técnicas, ferramentas e abordagens, bem como um processo pouco claro e generalista, que desencadeou em um grande desafio, qual seja, compreender e codificar tudo o que havia sido construído, em relação ao projeto apresentado, sob o prisma dos conceitos de ES.

Certamente havia a compreensão de quão necessário era, para o desenvolvimento de um projeto, a aplicação de um processo fundamentado em premissas da ES. Tal necessidade era fomentada pela ideia de construir, ainda na academia, um produto com finalidades comerciais, ou seja, usar todo o processo de ES, as ferramentas, os conceitos, enfim, qualquer boa prática existente para atribuir qualidade ao *software* que seria construído a partir do referido projeto.

Com um grande desafio já estipulado, e, considerando o conceito de “engenharia de *software* com o auxílio de computador” (PRESSMAN, 2011, p. 40), na qual ocorre a integração de ferramentas de modo a possibilitar que uma utilize informações geradas por outra, fez-se um novo desafio, o de unificar toda a documentação gerada no processo em um formato aceitável para publicação, de fácil manutenção e acessível, que por fim pudesse explicitamente indicar ao leitor que sua

**Comentado [GVTDS5]:** Verificar a possibilidade de incluir o modelo citado como Anexo do trabalho.

**Comentado [GVTDS6]:** Analisar se o projeto se enquadra nesse conceito.

produção seguiu por caminhos almejados a partir do emprego da ES e suas metodologias.

Portanto, o que será visto a seguir é um estudo que a partir de conceitos e opiniões visa apresentar uma forma viável de promover a gestão da documentação produzida em um projeto de *software*.

## 2.1 PROJETO DE SOFTWARE

(PRESSMAN, 2011, p. 47) afirma que “todo projeto de *software* é motivado por alguma necessidade de negócio”, ou seja, uma demanda financeira, industrial ou comercial.

Não apenas na área de TI, mas em diversas áreas de atividade humana é possível observar o termo projeto, que é um jargão conhecido e comumente empregado no intuito de expressar a ideia de construir algo novo de forma organizada. Geralmente em um intervalo de tempo determinado.

Seguindo este princípio, em se tratando de programas de computador, geralmente o desenvolvimento se dá através de um projeto, que segundo (FILHO, 2003) conta com “uma data de início, uma data de fim, uma equipe de desenvolvimento (da qual faz parte um responsável, denominado ‘gerente de projeto’) e outros recursos”.

Ainda segundo (FILHO, 2003), um projeto representa a execução de um processo e este, quando bem definido, possui etapas elaboradas de modo a possibilitar uma avaliação de progresso no desenvolvimento do projeto.

Assim, tem-se uma visão processual de projeto, ao passo que para (SOMMERVILLE, 2011, p. 25):

Um projeto de *software* é uma descrição da estrutura do *software* a ser implementado, dos modelos e estruturas de dados usados pelo sistema, das interfaces entre os componentes do sistema e, às vezes, dos algoritmos usados.

Tem-se então, uma visão documental de projeto e pode-se concluir que ambas as visões, processual e documental, contemplam essencialmente a finalidade organizacional expressa por meio do jargão.

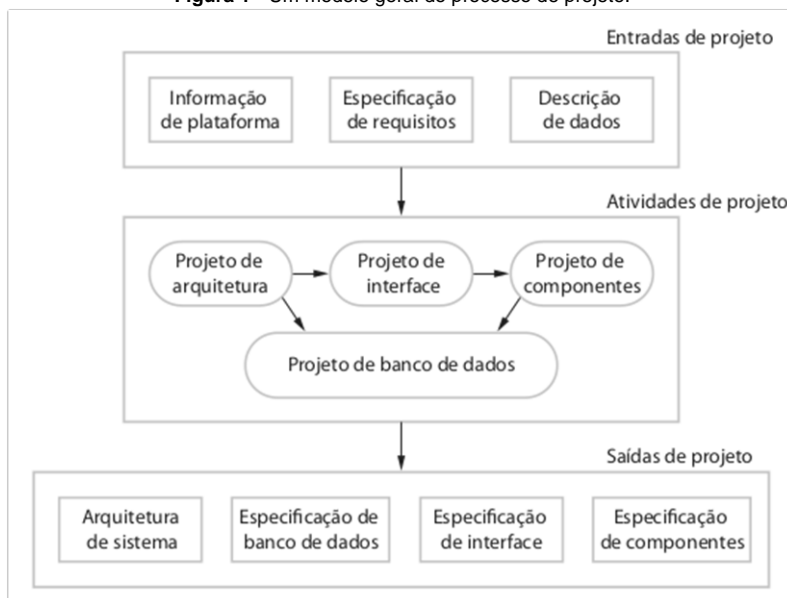
Deste modo, um processo generalista para um projeto de desenvolvimento de *software*, pode apresentar basicamente as etapas de:

1. Coleta de informações de entrada;

2. Execução de atividades que processam as informações da etapa 1; e
3. Construção do *software* com base nas especificações geradas na etapa 2.

Um modelo geral de processo de projeto pode ser visto de forma abstrata na Figura 1, representando atividades comuns no desenvolvimento de um sistema de informação.

**Figura 1** - Um modelo geral do processo de projeto.



**Fonte:** (SOMMERVILLE, 2011, p. 26).

A conclusão de cada etapa envolve a execução de atividades distintas, que podem ser interligadas direta ou indiretamente. Além disso, as atividades a serem realizadas no processo variam de projeto para projeto, produzindo saídas em forma de documentos de especificações, quando se utiliza métodos prescritivos de desenvolvimento, ou, se a abordagem proposta for um método ágil de desenvolvimento, as saídas são representadas diretamente no código da aplicação (SOMMERVILLE, 2011, p. 26).

## 2.2 ENGENHARIA DE SOFTWARE

Em se tratando da origem e da serventia da ES, segundo o relato de (SOMMERVILLE, 2011, p. xi) “o nome ‘engenharia de *software*’ foi proposto em 1969,

na conferência da Organização do Tratado do Atlântico Norte (OTAN), para a discussão de problemas relacionados com desenvolvimento de *software*” e para (PRESSMAN, 2011, p. 49), “a prática da engenharia de *software* é uma atividade de resolução de problemas que segue um conjunto de princípios básicos”, ou seja, programas de computador, que devem existir para solução de problemas, possuem problemas.

Comentado [GVTD57]: Precisa explicar a sigla?

Como consequência desta conclusão, pode-se constatar que, conforme citado na introdução deste TG, o objetivo da ES é a obtenção de *software* confiável dotado de qualidade. Em outras palavras, o tanto quanto possível livre de problemas.

A qualidade, na visão de (FILHO, 2003, p. 17), está estritamente relacionada com o grau de conformidade que o produto finalizado mantém com os respectivos requisitos, que “são as características que definem os critérios de aceitação de um produto” (FILHO, 2003, p. 13).

Por conseguinte, naturalmente chega-se ao princípio dos trabalhos em um projeto de *software*, que entre outras tarefas, primordialmente envolve o levantamento dos requisitos, talvez o primeiro artefato de documentação do projeto.

A documentação segue nas demais etapas do processo, que por sua vez é definido em conformidade com a visão de ES analogamente representada por camadas na Figura 2.



Fonte: (PRESSMAN, 2011, p. 39).

Pode-se dizer que não existem variações no esquema proposto na ilustração. Trata-se de:

- um objetivo o qual se pretende alcançar - o *software* de qualidade -, e
- uma forma de se obter o que é pretendido - a sequência de um processo através dos seus componentes, que são os métodos e as ferramentas.

Partindo dessa visão, sendo o projeto de desenvolvimento um empreendimento com data de início e data de conclusão previstas, chega-se ao o prazo para entrega, que é o primeiro problema do *software*, afinal, enquanto ele não é concluído, o problema real que necessita de auxílio no processo de solução, segue sendo abordado de outras maneiras, através de outras ferramentas teoricamente menos eficientes.

O prazo, metaforicamente entendido como um problema a ser equacionado pela ES, é útil para ilustrar a aplicação do esquema de camadas, ou seja, tendo como foco a qualidade, define-se um processo para obtenção do produto baseado em fatores como o prazo, por exemplo. Entretanto o prazo não é o único fator a ser debatido ao se estabelecer um processo para o desenvolvimento do projeto. Outros fatores como o escopo do *software*, a quantidade de requisitos, a quantidade de usuários atendidos e a mão de obra disponível devem ser considerados para a adoção de métodos e ferramentas.

### 2.2.1 Metodologias

As metodologias de desenvolvimento possuem uma diferenciação, de certo modo rasa, porém compreensível, entre os chamados métodos prescritivos e os métodos ágeis.

Sem adentrar no funcionamento e nas peculiaridades de cada um dos métodos existentes, o leitor deste TG pode considerar vago o propósito pelo qual este tópico foi abordado, contudo, é preciso salientar que a quantidade de documentos produzidos em um projeto de *software* depende do método empregado.

Assim, para melhor compreensão da proposta de documentação e suas aplicações, será feita uma breve introdução as metodologias de desenvolvimento mais frequentemente utilizadas.

**Comentado [GVTDS8]:** Será que essa introdução é necessária? Pode ser melhorada?

#### 2.2.1.1 Métodos prescritivos

Os métodos prescritivos surgiram entre as décadas de 1960 e 1970, na época em que foi cunhado o termo Engenharia de *Software*. Tais métodos, segundo (ANTONIO, 2011), “previam alto nível de regulação no ciclo de vida do *software*, prescrevendo um arcabouço rígido de desenvolvimento e grande quantidade de documentação”.

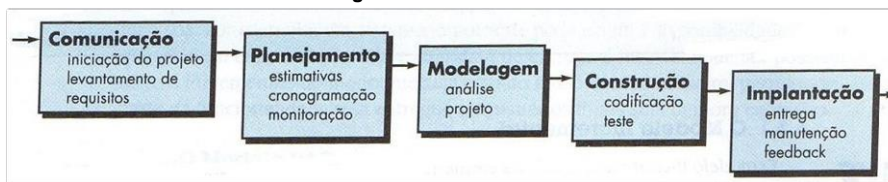
Ainda segundo (ANTONIO, 2011), o surgimento dos métodos prescritivos se deu em função de um período anterior crítico denominado “Crise do Software”, o que coaduna com o relato de (SOMMERVILLE, 2011, p. xi) sobre os problemas no desenvolvimento de *software* abordados na OTAN.

Também nessa época, surge o Modelo Cascata, que segundo (PRESSMAN, 2011, p. 60) é um dos métodos prescritivos mais conhecidos, sendo o paradigma mais antigo da ES. Este método se aproxima em vários aspectos da atividade de projeto de *software* proposta pela instituição de ensino que foi citada no início deste capítulo.

Trata-se de uma abordagem que segue um processo estritamente linear, começando pelo levantamento de requisitos junto ao cliente, seguindo pelo planejamento, pela modelagem, pela construção e, por fim, a implantação e o suporte ou manutenção do *software* (PRESSMAN, 2011, p. 59).

A Figura 3 ilustra as etapas do método denominado modelo cascata.

Figura 3 - Modelo Cascata.



Fonte: (PRESSMAN, 2011, p. 60)

Os métodos prescritivos são capazes de melhorar a qualidade do *software*, pois propiciam certas condições de gerenciamento que implicam em previsões realistas dos prazos, tendo em vista que se conhece as ferramentas e sabe-se exatamente quais documentos serão elaborados antes da codificação, quais etapas a codificação percorrerá e quais testes serão realizados, além de um procedimento para implantação já definido.

Contudo, o processo que utiliza um método tal qual o modelo cascata enfrenta um problema crítico em relação ao seu caráter linear, pois uma etapa só é iniciada após a conclusão da anterior, de modo que ao detectar um erro na etapa de comunicação, estando o projeto na etapa de construção, por exemplo, será preciso refazer as etapas intermediárias de planejamento e modelagem, bem como a própria etapa de construção.

Por essa razão o modelo cascata não é o único método prescritivo que existe, pode-se elencar os modelos de Processo Incremental, Evolucionário e Espiral, que

Comentado [GVTDS9]: Procurar uma boa referência.

Comentado [GVTDS10]: Considerar a fonte original, ou a fonte da internet? Parece ser de uma edição anterior ao livro da biblioteca da faculdade.

surgiram com a missão de amenizar este problema, além de outros modelos que não são ditos prescritivos totalmente, mas que possuem fortes ligações com os métodos prescritivos tradicionais, sendo eles os modelos de Processo Especializado, Processo Unificado, Processo Pessoal e de Equipe e o Processo de Produto.

**Comentado [GVTDS11]:** Será que precisa citar os modelos já que eu não pretendo explicá-los?

### 2.2.1.2 Métodos ágeis

O Desenvolvimento Ágil, amplamente adotado a partir do lançamento do Manifesto Ágil (BECK, BEEDLE, *et al.*, 2001), permite empreender projetos de *software* dentro dos objetivos da ES para segmentos que possuem questões críticas, como o caso das mudanças de requisitos em função das necessidades de negócio que por ventura venham a ocorrer após o início do projeto.

Os autores do Manifesto Ágil fundamentaram sua publicação em dose princípios, onde: constata-se a intensão de valorizar a entrega do *software* no processo de desenvolvimento, fazendo-a de forma parcial e continua de modo a agregar valor (funcionalidades) em intervalos curtos; tem-se as entregas como parâmetro para aferir o progresso rumo a conclusão do projeto; preconiza-se as interações entre os envolvidos no processo favorecendo conversas face a face; e promove-se a sustentabilidade por meio de processos simplificados, tendo em vista a redução de trabalho, além de constantes avaliações sobre a eficácia, modificando a abordagem quando preciso (BECK, BEEDLE, *et al.*, 2001).

Isto de fato cria um contraponto ao modelo tradicional de desenvolvimento encabeçado pelos métodos prescritivos, onde: a implantação do *software* era realizada apenas no final do processo; o progresso era medido por conclusões de etapas, sendo que a saída de algumas etapas constituíam documentos de especificações; a documentação produzida também possuía o viés comunicativo entre os envolvidos no projeto; e cumpria-se fielmente o processo sem variações na abordagem em face a eficácia ou a não eficácia do seu emprego.

Este contraponto é perceptível no Manifesto Ágil, que diz:

- Indivíduos e interações mais que processos e ferramentas
- Software* em funcionamento mais que documentação abrangente
- Colaboração com o cliente mais que negociação de contratos
- Responder a mudanças mais que seguir um plano



Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda (BECK, BEEDLE, *et al.*, 2001).

Contudo, segundo (PRESSMAN, 2011, p. 81), “o desenvolvimento ágil poderia ser mais bem denominado ‘engenharia de *software* flexível””. Para ele, atividades básicas como comunicação, planejamento, modelagem, construção e emprego, vistas no modelo cascata, podem ser notadas em métodos ágeis, onde ao invés de serem tidas como paradigmas constituem um conjunto mínimo de tarefas que impulsionam o desenvolvimento.

As atividades citadas invariavelmente fariam parte de um planejamento, porém, as origens dos projetos de desenvolvimento de *software*, que são as necessidades de negócio, passam constantemente por mudanças, e, segundo (SOMMERVILLE, 2011, p. 38), inicialmente os clientes (donos do negócio) consideram “impossível obter um conjunto completo de requisitos de *software* estável”.

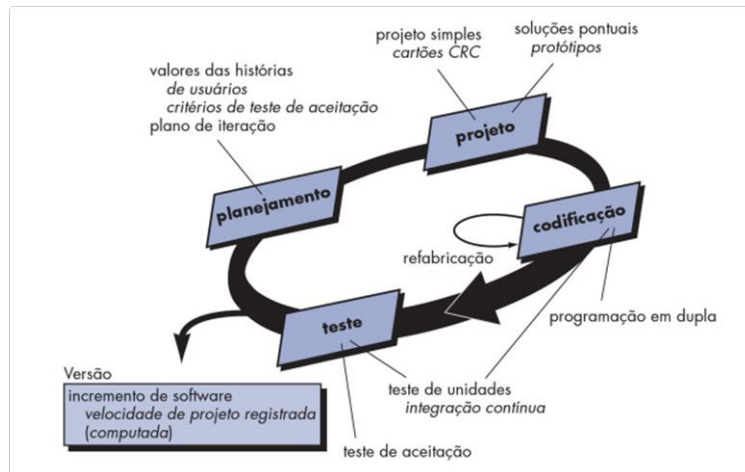
Portanto, a flexibilidade está na capacidade de adaptar o projeto as novas necessidades na medida em que forem surgindo, ou seja, responder as mudanças em detrimento ao planejamento anterior, conforme previsto no manifesto.

#### 2.2.1.3 *Extreme Programming, Scrum e Kanban*

O *Extreme Programming* - **XP** (em português, Programação Extrema), e o *Scrum* são duas metodologias consideradas expoentes do desenvolvimento ágil. No caso da XP, entre outras características, (PRESSMAN, 2011, p. 87) cita como valores da metodologia a intensão de evitar documentação volumosa como forma de comunicação e a restrição do desenvolvimento as necessidades imediatas, a fim de liberar as funcionalidades para uso rapidamente.

A Figura 4 exibe o processo adotado na metodologia *XP*, onde constata-se que as entregas de funcionalidades representam o progresso do projeto, conforme preconizado pelo manifesto ágil.

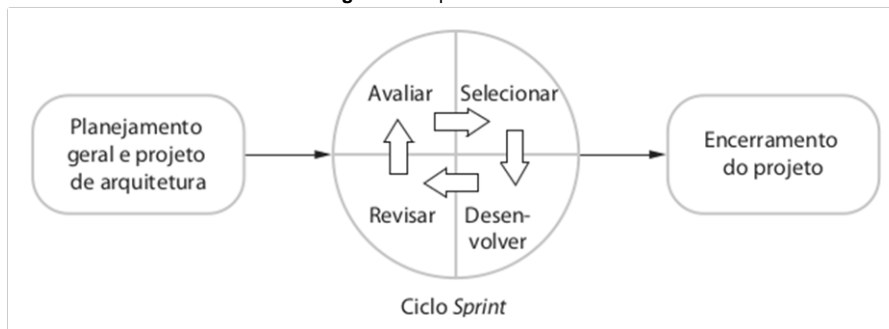
**Figura 4** - O processo da *Extreme Programming (XP)*.



Fonte: (PRESSMAN, 2011, p. 88)

Já no *Scrum*, a outra metodologia expoente do desenvolvimento ágil, segundo (SOMMERVILLE, 2011, p. 50), o processo é constituído de três fases: o planejamento, o *sprint* e o encerramento do projeto, conforme demonstrado na Figura 5.

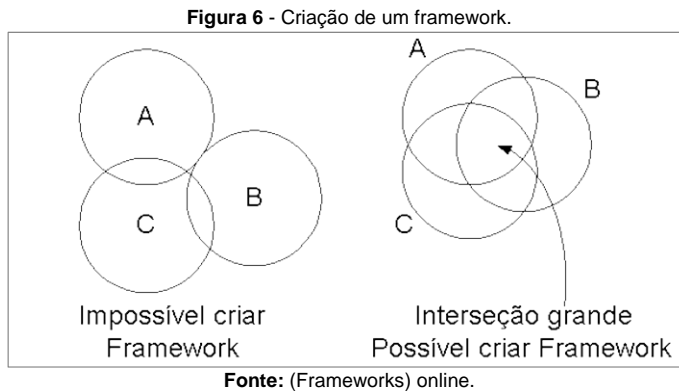
**Figura 5** - O processo *Scrum*.



Fonte: (SOMMERVILLE, 2011, p. 50).

O *sprint*, que é a segunda fase do processo, ocorre de forma recorrente a fim de obter as entregas incrementais de funcionalidades do *software*. Assim, o *Scrum*, que muitas vezes é tido como metodologia, e pode-se dizer que é um método, possui outra faceta pela qual é conhecido, a de ser um *framework* de desenvolvimento. Os próprios criadores do *Scrum* o definem com *framework* (SCHWBER e SUTHERLAND, 2017, p. 3).

A Figura 6 demonstra quando é possível obter um *framework*, a partir da comparação entre três atividades distintas A, B e C.



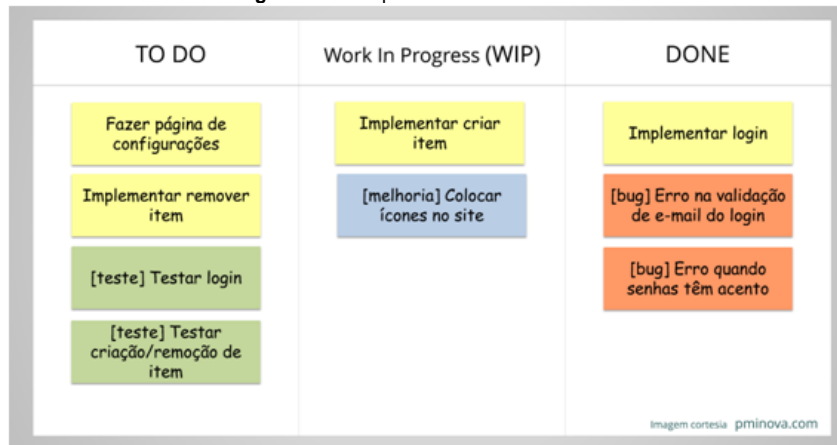
Um conjunto de soluções constitui um *framework*, que pode ser utilizado em várias aplicações, ou seja, soluciona problemas que são inerentes ao desenvolvimento de determinadas atividades, ainda que essas atividades não sejam relacionadas diretamente ou que pertençam a áreas distintas do conhecimento.

Voltando ao *Scrum*, durante a fase do *sprint* surge a oportunidade de implementar outras metodologias no processo, como por exemplo o *Kanban*.

É comum a adoção do *Scrum* em conjunto com o *Kanban* que possui uma dinâmica de fácil compreensão, consistindo basicamente em um quadro colunado onde: cada coluna representa um estado para as tarefas; as tarefas são representadas por cartões contendo as informações básicas daquilo que deve ser executado; os cartões se movem da esquerda para a direita no quadro representando o progresso da execução de uma tarefa.

Em um quadro *Kanban* é comum a existência de três colunas intituladas: tarefas para fazer, tarefas em andamento e tarefas concluídas, conforme demonstrado na Figura 7, onde lê-se, em inglês: *TO DO*, *WORK IN PROGRESS* e *DONE*, respectivamente.

**Figura 7** - Exemplo de *Kanban* com tarefas.



Fonte: (MARTINS) online.

Eventualmente a quantidade de listas no quadro pode variar de acordo com as necessidades de um projeto, incluindo novos estados para as tarefas ou até mesmo substituindo os que foram apresentados.

Uma das características do *Kanban* remete ao princípio de priorização das tarefas e, no quadro apresentado na Figura 7, pode-se presumir que foi adotado um método de priorização baseado nas cores dos cartões, por exemplo, um cartão da cor vermelha talvez deva ser iniciado antes de um cartão amarelo ou vice versa. Este é um tipo de convenção a ser decidido na primeira fase do *Scrum*, caso o quadro *Kanban* faça parte do rol de ferramentas do projeto.

Em métodos ágeis, de modo geral, os requisitos levantados no início do projeto são desenvolvidos de forma incremental, obedecendo as prioridades estabelecidas pelos usuários (clientes).

### 2.2.2 Ferramentas

Algumas ferramentas podem ser mais bem utilizadas por uma ou outra metodologia. O emprego de determinadas ferramentas implica na análise em particular do projeto que se deseja empreender, bem como a metodologia a ser adotada.

No início de um projeto, geralmente tem-se a tarefa de levantamento dos requisitos, os quais o produto deverá estar em conformidade quando o projeto estiver

concluído. Para essa tarefa em específico, existe um amplo estudo denominado Engenharia de Requisitos, que dispõe de alguns mecanismos que auxiliam a coleta de tais dados.

Sobre o levantamento de requisitos, algumas das ferramentas utilizadas, também chamadas de técnicas, são: entrevistas, histórias de usuários, etnografia que consiste na observação das tarefas durante a execução, casos de usos e outras, sendo que, após coletados os requisitos devem ser documentados.

A documentação pode se dar de três formas:

- Texto com linguagem natural ou linguagem natural estruturada;
- Modelo conceitual; ou
- Híbrido, contemplando as duas formas anteriores.

#### 2.2.2.1 Linguagem natural e linguagem natural estruturada

O documento elaborado com linguagem natural possibilita o entendimento dos requisitos por todos os envolvidos no projeto, mesmo aqueles que não são especializados no desenvolvimento de *software*, sendo necessário apenas a compreensão do conjunto de regras do negócio.

O Quadro 1 exibe um trecho de um **DR** no qual é possível observar o uso de linguagem natural para descrever os requisitos de um *software*.

**Quadro 1** - Exemplo de requisitos para o sistema de *software* de bomba de insulina.

Requisitos em linguagem natural
...
3.2 - O sistema deve medir o açúcar no sangue e fornecer insulina, se necessário, a cada dez minutos. (Mudanças de açúcar no sangue são relativamente lentas, portanto, medições mais frequentes são desnecessárias; medições menos frequentes podem levar a níveis de açúcar desnecessariamente elevados.)
...
3.6 - O sistema deve, a cada minuto, executar uma rotina de autoteste com as condições a serem testadas e as ações associadas definidas no Quadro 4.3 (A rotina de autoteste pode descobrir problemas de hardware e software e pode alertar o usuário para a impossibilidade de operar normalmente.)
...

**Fonte:** (SOMMERVILLE, 2011, p. 67).

No Quadro 1 é possível notar, no requisito identificado como item 3.6, que há uma referência a um trecho do documento onde os requisitos são documentados com linguagem natural estruturada. Este trecho é visto no Quadro 2.

**Quadro 2** - Uma especificação estruturada de um requisito para uma bomba de insulina.

Requisitos em linguagem natural estruturada	
Bomba de insulina/Software de controle/SRS/3.3.2	
Função	Calcula doses de insulina: nível seguro de açúcar.
Descrição	Calcula a dose de insulina a ser fornecida quando o nível de açúcar está na zona de segurança entre três e sete unidades.
Entradas	Leitura atual de açúcar (r2), duas leituras anteriores (r0 e r1).
Fonte	Leitura atual da taxa de açúcar pelo sensor. Outras leituras da memória.
Saídas	CompDose - a dose de insulina a ser fornecida.
Destino	Loop principal de controle.
Ação	CompDose é zero se o nível de açúcar está estável ou em queda ou se o nível está aumentando, mas a taxa de aumento está diminuindo. Se o nível está aumentando e a taxa de aumento está aumentando, então CompDose é calculado dividindo-se a diferença entre o nível atual de açúcar e o nível anterior por quatro e arredondando-se o resultado. Se o resultado é arredondado para zero, então CompDose é definida como a dose mínima que pode ser fornecida.
Requisitos	Duas leituras anteriores, de modo que a taxa de variação do nível de açúcar pode ser calculada.
Pré-condição	O reservatório de insulina contém, no mínimo, o máximo de dose única permitida de insulina.
Pós-condições	r0 é substituída por r1 e r1 é substituída por r2.
Efeitos colaterais	Nenhum.

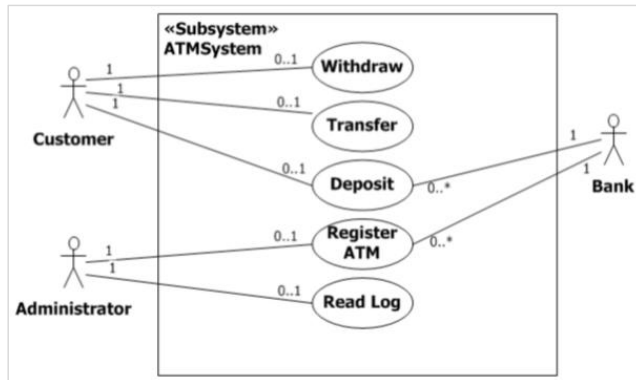
**Fonte:** (SOMMERVILLE, 2011, p. 68).

#### 2.2.2.2 UML e BPMN

A documentação elaborada a partir de modelos conceituais implica em prévio conhecimento da notação utilizada. Por exemplo, em projetos desenvolvidos com base no paradigma da Orientação a Objeto, é comum o uso de diagramas da **UML** - *Unified Modeling Language* (em português, Linguagem de Modelagem Unificada). A

Figura 8 exibe um modelo conceitual como um diagrama UML de caso de uso para um sistema bancário.

**Figura 8** - Exemplo do sistema bancário com caso de uso e atores.



Fonte: (OBJECT MANAGEMENT GROUP, 2017, p. 643)

No diagrama, as figuras de bonecos de palito representam os atores, que são os responsáveis por executar as ações; as ações são representadas pelas figuras elípticas e o nome de cada ação, verbos, são inscritos no interior das elipses (OBJECT MANAGEMENT GROUP, 2017, p. 644 e 645).

A UML possui diversos diagramas capazes de exemplificar, de acordo com o seu propósito, os requisitos levantados para o desenvolvimento de um *software*, sendo, portanto, confirmada a necessidade de o indivíduo envolvido com o projeto conhecer a notação empregada para compreender os requisitos documentados com um modelo conceitual.

Neste ponto, observa-se a ligação entre a tarefa de levantamento de requisitos e a consequente tarefa de modelagem do *software*, que além de contar com a linguagem UML, possui outras ferramentas, sendo uma delas a modelagem de processos denominada **BPMN** - *Business Process Model and Notation* (em português, Notação de Modelagem de Processos de Negócio).

A modelagem BPMN consiste na elaboração de diagramas e foi desenvolvida para representar graficamente os processos de um negócio, não se atendo apenas ao desenvolvimento de *software*.

O principal objetivo do BPMN é fornecer uma notação que seja facilmente compreensível por todos os usuários empresariais, do negócio: os analistas que criam os rascunhos iniciais dos processos, os desenvolvedores técnicos responsáveis pela implementação da tecnologia que irá realizar esses

processos e, finalmente, os empresários que gerenciarão e monitorarão aqueles processos (OBJECT MANAGEMENT GROUP, INC. (OMG), 2011, p. 1).

Deste modo, os diagramas BPMN são constantemente utilizados durante o processo de desenvolvimento de *software*, justamente por sua contribuição considerável para o entendimento e a elaboração dos processos os quais um *software* deverá servir como ferramenta de apoio.

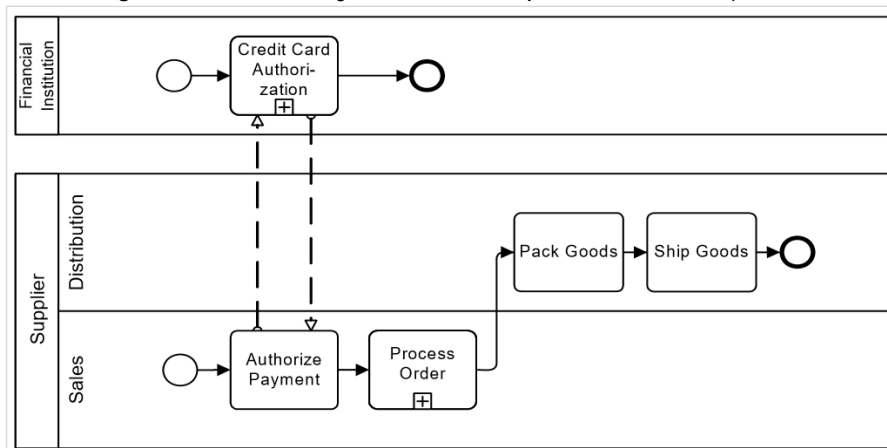
Um diagrama BPMN possui elementos gráficos, figuras geométricas, que são dispostas de modo a estabelecer um fluxo do princípio ao fim de um processo, inclusive abrangendo subprocessos.

Nos diagramas é possível observar, entre outras características do processo, a interação os setores de uma empresa. Neste caso, os setores são tidos como atores, cada um responsável pela realização de tarefas que se complementam mutuamente para no fim constituírem o processo.

Assim, no diagrama BPMN as tarefas pertinentes a cada ator são dispostas em raias específicas, pois, analogamente, vê-se o processo como uma piscina olímpica e as raias dessa piscina representam os limites de atuação dos atores.

A Figura 9, traz um exemplo de diagrama BPMN simulando a comunicação entre um cliente e um fornecedor no processo de expedição de um cartão de crédito.

**Figura 9** - Fluxo de mensagens conectadas a objetos de fluxo em duas piscinas.



Fonte: (OBJECT MANAGEMENT GROUP, INC. (OMG), 2011, p. 113)

No exemplo, é possível observar que a piscina inferior, que representa a parcela do processo executada pelo Fornecedor (*Supplier*), possui duas raias/atores,



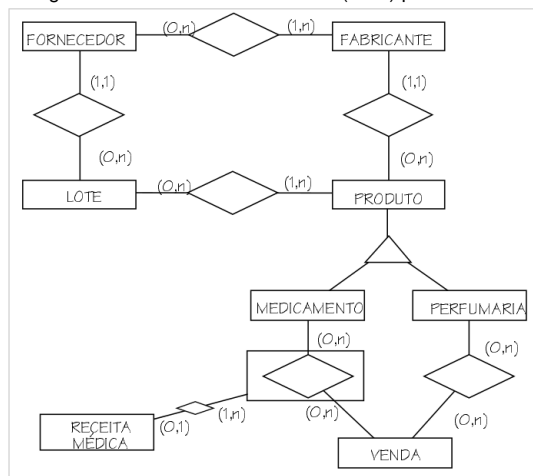
representando os setores de Vendas (*Sales*) e Distribuição (*Distribution*) e a piscina superior, que representa a parcela do processo executada pela Instituição Financeira (*Financial Institution*) possui apenas uma raia, indicando que toda da instituição representa um ator no processo.

### 2.2.2.3 Banco de dados

Outro tipo de modelagem, este específico para projetos de *software* que utilizam Banco de Dados (BD), é o Modelo Entidade Relacionamento (MER). A contrução, parte de uma concepção denominada Modelo Conceitual, onde o BD “é descrito de forma independente de implementação” (HEUSER, 2009, p. 25).

A abordagem Entidade Relacionamento (ER), de onde deriva o MER, consiste em uma técnica de representação do BD através de um diagrama denominado Diagrama Entidade Relacionamento (DER), ilustrado pela Figura 10.

**Figura 10** - Diagrama Entidade Relacionamento (DER) para sistema de farmácias.



**Fonte:** (HEUSER, 2009, p. 67).

O diagrama apresentado, elaborado para um sistema de farmácias, permite observar:

- as entidades que são representadas por retângulos com o nome de cada entidade inscrito no interior do retângulo;
- os relacionamentos entre as entidades representados por losangos;

- as entidades associativas, que são representadas por losangos inscritos em retângulos;
- a indicação de especialização de uma entidade representada pelo triângulo;
- as respectivas conexões entre os elementos dadas pelas linhas; e
- os números entre parênteses, que representam a cardinalidade, ou seja, tipo de relacionamento entre duas entidades.

Quanto a cardinalidade, utilizando como exemplo o relacionamento entre LOTE e PRODUTO, demonstrado no diagrama da Figura 10, diz-se que, um LOTE possui no mínimo um e no máximo  $n$  PRODUTOS, ao passo que um PRODUTO pode pertencer no mínimo a nenhum LOTE e no máximo a  $n$  LOTES.

Outra concepção relacionada com a elaboração de BD é o Modelo Lógico, o qual é constituído por uma descrição textual e estruturada das tabelas de um BD.

O Modelo Lógico representa uma abstração do BD e é tido como uma transformação do Modelo Conceitual a fim de implantá-lo em um determinado Sistema de Gerenciamento de Banco de Dados (SGDB).

Neste ponto da modelagem tem-se uma estrutura representativa com todas as tabelas e os respectivos atributos (nomes das colunas em uma tabela do BD) conforme mostrado no Quadro 3.

**Quadro 3** - Modelo Conceitual de um cadastro de produtos.

Modelo Conceitual - Cadastro de Produtos
TipoDeProduto (CodTipoProd, DescrTipoProd) Produto (CodProd, DescrProd, PrecoProd, CodTipoProd) CodTipoProd referencia TipoProduto

**Fonte:** (HEUSER, 2009, p. 27).

O trabalho de modelagem do BD implica, além do levantamento de requisitos, no entendimento sobre as regras de negócio as quais o *software* estará sujeito, pois são elas que determinam os tipos de dados e as especificações as quais os dados deverão estar em conformidade antes de serem armazenados.

As especificações, ou seja, a tipificação dos dados aparece em outros tipos de representação do BD e, segundo (HEUSER, 2009, p. 24), “cada representação de um modelo de dados através de uma linguagem de modelagem recebe a denominação de Esquema de Banco de dados”.

**Comentado [GVTDS12]:** Rever

#### 2.2.2.4 Prototipação

O termo protótipo remete a ideia de primeiro exemplar de um produto, uma versão preliminar que é utilizada para fins de teste e aperfeiçoamento. Em relação ao desenvolvimento de *software*, a tarefa de prototipação “tem como objetivo principal validar os requisitos, abordar questões de interface, e avaliar tanto a viabilidade quanto a complexidade do sistema”, segundo (SOUZA, VALE e ARAÚJO, 2008, p. 45).

Assim, o que se espera com um protótipo é expor os conceitos do projeto, sendo, portanto, uma prática comum que pode tanto ser desenvolvida com o auxílio de programas de computador quando através de desenhos feitos a mão.

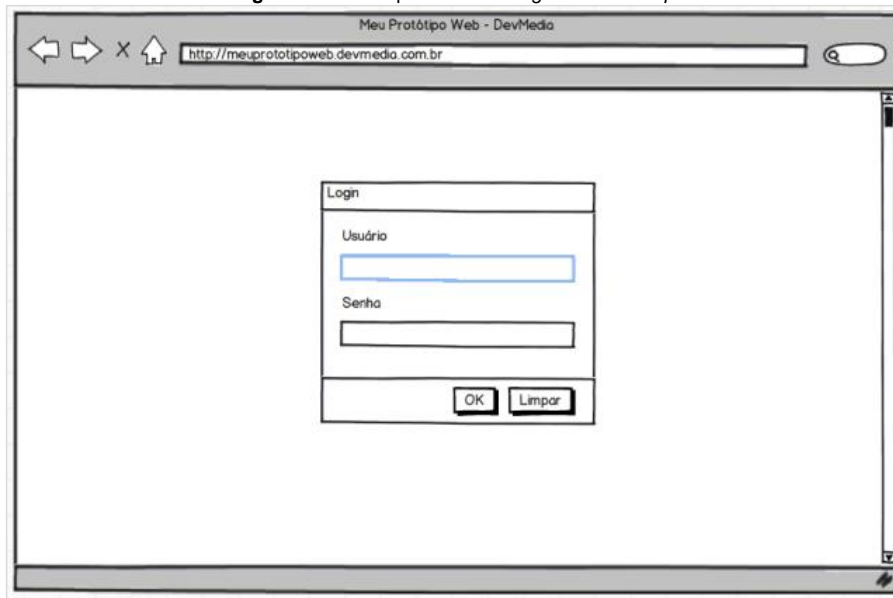
Pode-se classificar os protótipos em três categorias: baixa fidelidade, média fidelidade e alta fidelidade. Deste modo, os protótipos de baixa fidelidade são destinados a crítica do produto em relação ao levantamento de requisitos; os protótipos de média fidelidade são destinados a representação visual do produto de maneira fidedigna; e por fim os protótipos de alta fidelidade destinam-se ao teste componentes e solução de problemas técnicos.

Por conseguinte, percebe-se que a medida que a fidelidade de um protótipo avança da baixa para a alta, o esforço, o tempo gasto e consequentemente o custo de produção tende a aumentar. Deste modo, o uso dessa ferramenta é regado por metodologias ágeis que preconizam a avaliação do custo benefício que uma tarefa desta natureza possui, ou seja, se o valor agregado pela adoção de protótipos de todas as categorias compensa o esforço, o tempo e o custo de desenvolvimento da tarefa.

No entanto, considerando a escolha do nível de fidelidade dos protótipos de um projeto, algo estritamente ligado ao seu tamanho e relevância, um ponto de partida interessante para projetos de aplicações *web*, é o desenvolvimento de *wireframes*, pois esses permitem vislumbrar o produto e criticar questões de interface e experiência de usuário em um nível de fidelidade aceitável para grande parte dos projetos que são empreendidos por desenvolvedores, individualmente ou em pequenas empresas com recursos limitados.

A Figura 11 apresenta um exemplo de *wireframe* denominado *mockup*, representando a tela de autenticação em um sistema *web*.

**Figura 11** - Protótipo de tela de *login* com *mockup*.



The image shows a web browser window with the title "Meu Protótipo Web - DevMedia". The address bar contains the URL "http://meuprototipoweb.devmedia.com.br". The main content area displays a login form with the following elements:

- A title "Login" at the top of the form.
- A label "Usuário" above a text input field.
- A label "Senha" above a text input field.
- Two buttons at the bottom: "OK" and "Limpar".

Fonte: (MALHERBI, 2013) online.

### 2.2.3 Qualidade de *software*

Enfim, a almejada qualidade de *software* vem à tona trazendo consigo uma série de questões subjetivas que dividem opiniões e motiva o estudo acerca do processo de desenvolvimento. O tema qualidade já foi abordado no início da sessão 2.2, de modo este trecho pretende apenas completar o raciocínio iniciado anteriormente.

Inicialmente, retomando a definição de qualidade, (SENE, 2012, p. 46) recorre a NBR (norma brasileira) 13.596 de agosto de 1996, que descreve qualidade como "a totalidade de características de um produto de *software* que lhe confere a capacidade de satisfazer necessidades explícitas e implícitas". Neste ponto, ainda segundo (SENE, 2012, p. 47) a "qualidade é um processo sistemático que focaliza todas as etapas e artefatos produzidos com o objetivo de garantir a conformidade de processos e produtos, prevenindo e eliminando defeitos".

Deste modo, nota-se que a qualidade depende de inúmeros fatores e que não se trata de algo trivial, afinal, satisfazer necessidade explícitas e implícitas utilizando

um processo sistemático que previne e elimina defeitos é uma atividade deveras complexa.

Assim, limitando esta pesquisa ao foco deste **TG**, que é a documentação de *software*, e considerando a documentação como um dos fatores que implicam em qualidade, é possível retomar um ponto já discutido que trata da importância dada a tarefa de documentação por diferentes metodologias de desenvolvimento, observando que, em relação a métodos:

Algumas pessoas pensam que a qualidade de *software* pode ser alcançada por meio de processos prescritivos, baseados em padrões organizacionais e procedimentos de qualidade associados que verificam que esses padrões serão seguidos pela equipe de desenvolvimento de *software*. Seu argumento é que os padrões incorporam as boas práticas de engenharia de *software* e que segui-las levará a produtos de alta qualidade. Na prática, contudo, existe muito mais no gerenciamento de qualidade do que apenas padrões e a burocracia associada para garantir que sejam seguidos (SOMMERVILLE, 2011, p. 456).

Talvez essa visão colabore para a adoção de metodologias ágeis em detrimento aos métodos prescritivos, e dessa forma tem-se a diminuição das tarefas relacionadas a documentação.

Ainda assim, existem aqueles que defendem e priorizam o processo de documentação e análise detalhados, enquanto outros, em razão do que foi exposto, consideram que manter o foco no código e entregar funcionalidades o mais rapidamente possível seja o ponto mais importante a se observar no desenvolvimento.

Além disso, nota-se que ao adotar métodos ágeis de desenvolvimento, às vezes ao concluir a elaboração de um DR, este já está ultrapassado e consequentemente o trabalho empenhado foi perdido.

Nessa peleja, não há que se dizer que um lado tem razão em detrimento ao outro, cabe apenas observar que um repositório de informações, ou seja, de documentação do projeto, é algo conveniente a todos aqueles que em dado momento necessitam se orientar em relação ao trabalho que deve ser desenvolvido.

Por fim, a qualidade do *software* não se define apenas durante o desenvolvimento. Há que se pensar também, na qualidade do *software* considerando todas as suas partes, inclusive a documentação, sendo que, após a entrega do *software*, o usuário deve dispor de manuais de utilização que possam conduzi-lo a correta operação.

**Comentado [GVTDS13]:** Citar os slides do Professor Ely que falam sobre a mudança dos requisitos durante o processo de documentação em métodos ágeis

### 3 DOCUMENTAÇÃO DE SOFTWARE

Considerando o *software* como produto, ou seja, resultado de um processo, pode-se definir documentação de um *software* como sendo todo material elaborado com a finalidade de tornar o produto compreensível para desenvolvedores, engenheiros, usuários, representantes comerciais, enfim, qualquer indivíduo que em determinado momento necessite de informações do projeto, o que pode ocorrer durante os processos de planejamento e desenvolvimento ou quando o *software* já se encontra em fase de produção.

Muitas pessoas pensam que *software* é simplesmente outra palavra para programas de computador. No entanto, quando falamos de engenharia de *software*, não se trata apenas do programa em si, mas de toda a documentação associada e dados de configurações necessários para fazer esse programa operar corretamente. Um sistema de *software* desenvolvido profissionalmente é, com frequência, mais do que apenas um programa; ele normalmente consiste em uma série de programas separados e arquivos de configuração que são usados para configurar esses programas. Isso pode incluir documentação do sistema, que descreve a sua estrutura; documentação do usuário, que explica como usar o sistema; e sites, para usuários baixarem a informação recente do produto (SOMMERVILLE, 2011, p. 3).

O propósito de se documentar um *software* atende a duas necessidades, quais sejam: facilitar a comunicação entre eventuais membros da equipe ou entre a equipe e o cliente; e facilitar o entendimento das funções em eventuais atividades de manutenção ou atualização do *software* (FURTADO, 2016).

A abordagem de (SOMMERVILLE, 2011, p. 3), extraída de uma obra voltada para área de ES, faz menção tanto a documentação estrutural de um *software* quanto a documentação para o usuário final. Nota-se ali, a importância que é dada a documentação, sob todos os aspectos, na medida que foi considerada como parte do *software* e não algo que o acompanha em sua distribuição. Em outras palavras, a documentação faz parte daquilo que define o *software*.

A mesma visão pode ser observada na definição de (PRESSMAN, 2011, p. 32):

*Software* consiste em: (1) instruções (programas de computador) que, quando executadas, fornecem características, funções e desempenho desejados; (2) estrutura de dados que possibilitam aos programadores manipular informações adequadamente; e (3) informação descritiva, tanto na forma impressa como na virtual, descrevendo a operação e o uso de programas.

#### 3.1 DOCUMENTAÇÃO DO PROJETO DE SOFTWARE

Durante o processo de desenvolvimento do *software*, caso a equipe ou mesmo alguém que trabalha individualmente não anote, não documente aquilo que está fazendo como por exemplo os objetivos, as ações tomadas, mas principalmente para que serve e como deve ser utilizado cada um dos componentes do *software*, seguramente em um dado momento haverá, entre outros problemas alguns questionamentos como:

- Para que serve e por que tal componente foi desenvolvido?
- Como usar?
- Quais as dependências?
- Quem fez?

Para que o projeto prossiga e possa ser entregue, esses questionamentos devem obter respostas. Portanto, aceitando-se as definições de *software* apresentadas anteriormente e considerando o que já foi abordado acerca de projeto e ES, constata-se que, durante o processo devem ser elaborados muitos diagramas, protótipos, especificações, instruções e outras tentativas de esclarecer as ideias, diretrizes, procedimentos, normas e afins.

Assim, todo este conteúdo produzido é denominado documentação de projeto de *software* e esse trabalho é explicado por (FURTADO, 2016) conforme segue:

Das definições de engenharia de software, a mais adequada para a documentação de projetos é que ela é uma área da computação voltada para a especificação, desenvolvimento e manutenção de sistemas de *software*, com aplicação de tecnologias e práticas de gerência de projetos e outras disciplinas visando organização, produtividade e qualidade.

Deste modo, tem-se novamente o foco na qualidade do produto como objetivo. Nesse sentido, algo que deve ser obtido ao produzir a documentação do projeto, seja através de uma plataforma de modelagem com notação específica ou através de textos em linguagem natural ou natural estruturada, é a definição precisa do requisito, ou seja, uma especificação de características de funcionamento que seja suficiente para codificar, testar e implementar o *software*.

### 3.2 DOCUMENTAÇÃO DE SOFTWARE PARA USUÁRIOS

Esta parte da documentação do *software*, voltada para o usuário, descreve os modos de instalação, parametrização e operação.

### 3.2.1 Apresentação de funcionalidades e documentação operacional

Algumas empresas desenvolvedoras, ao distribuírem os *softwares* optam por oferecer algum tipo de manual impresso, contendo a descrição das funcionalidades e modos de operação. Porém, a prática mais comum neste sentido é a distribuição do manual operacional do *software*, e até mesmo material de apresentação das funcionalidades, em formato digital.

Outra prática comum é a distribuição dos manuais operacionais em formato de páginas de hipertexto estáticas, as páginas **HTML**<sup>1</sup> - *Hiper Text Markup Language* (em português, Linguagem de Marcação para Hipertexto), que são executadas localmente no computador do usuário, ou então em arquivos no formato **PDF** - *Portable Document File* (em português pode-se considerar Formato de Documento Portátil).

Entretanto, nota-se que as opções de ajuda, que são encontradas a partir de algum atalho na interface do *software* (neste caso considera-se interface gráfica), atualmente tem encaminhado o usuário diretamente para a página de ajuda *on-line* no site da empresa desenvolvedora.

Evidentemente este modo de distribuição torna mais simples a manutenção da documentação para usuários finais do *software*, considerando principalmente os processos de atualizações, onde é comum o *software* receber novas funcionalidades que necessitam serem reportadas nos documentos.

Também existem softwares criados especialmente para a leitura de documentação, como o caso do *Man*, bastante utilizado em terminais de **CLI** - *Comand Line Interface* (em português, Interface de Linha de Comando) dos Sistemas Operacionais **(SO)s GNU/Linux**.

Neste caso em especial, a documentação de um dado *software* é formatada em arquivo estruturado de forma específica e empacotada juntamente com os binários de instalação do programa. Durante a instalação ocorre a indexação da documentação na base de dados do programa *Man*, tornando-a acessível para o usuário do computador.

Um exemplo de utilização do *Man* pode ser observado na Figura 12, onde é solicitada a documentação do software de controle de versão *Git* através da CLI em um terminal do SO *Debian GNU/Linux*.

---

<sup>1</sup> **HTML**: Linguagem de marcação de texto estruturado para publicação de conteúdo (texto, imagem, vídeo e áudio) na *web* (FERREIRA e EIS, 2011).



**Figura 12** - Chamada ao *Man* para exibição da documentação do *git*.

```
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
~ man git
```

Fonte: autoria própria.

A Figura 13 exibe no terminal do SO *Debian GNU/Linux* um trecho da documentação do *Git* apresentada pelo *Man*.

**Figura 13** - Documentação do *Git* exibida através do *Man*.

```
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
GIT(1)                                     Git Manual                                GIT(1)

NAME
  git - the stupid content tracker

SYNOPSIS
  git [--version] [--help] [-C <path>] [-c <name>=<value>]
    [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
    [-p|--paginate|--no-pager] [--no-replace-objects] [--bare]
    [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
    [--super-prefix=<path>]
    <command> [<args>]

DESCRIPTION
  Git is a fast, scalable, distributed revision control system with an unusually rich command set that
  provides both high-level operations and full access to internals.

  See gittutorial(7) to get started, then see giteveryday(7) for a useful minimum set of commands. The
  Git User's Manual[1] has a more in-depth introduction.

  After you mastered the basic concepts, you can come back to this page to learn what commands Git
  offers. You can learn more about individual Git commands with "git help command". gitcli(7) manual
  page gives you an overview of the command-line command syntax.

  A formatted and hyperlinked copy of the latest Git documentation can be viewed at
  https://git.github.io/htmldocs/git.html.

OPTIONS
  --version
    Prints the Git suite version that the git program came from.

Manual page git(1) line 1 (press h for help or q to quit)
```

Fonte: autoria própria.

Ademais, quanto a interação do usuário com o programa de computador, sabe-se que existem *softwares* de vários tipos. Tomando como exemplo um tipo especial de *software*, os chamados *frameworks*, que são voltados para desenvolvedores de *software*, e sua definição é a mesma adotada na sessão 2.2.1.3, especificamente demonstrada na Figura 6, constata-se o quão importante é a tarefa de documentação.

A Figura 14 exibe um trecho da documentação de um *framework* em linguagem *Javascript* para desenvolvimento de aplicações *web* chamado *Vue.js*.

Figura 14 - Trecho da documentação do *framework* Vue.js.

## Renderização Declarativa

No núcleo do Vue.js está um sistema que nos permite declarativamente renderizar dados no DOM (Document Object Model) usando uma sintaxe de *template* simples:

```
<div id="app">
  {{ message }}
</div>
```

HTML

```
var app = new Vue({
  el: '#app',
  data: {
    message: 'Olá Vue!'
  }
})
```

JS

Olá Vue!

Acabamos de criar nosso primeiro aplicativo Vue! Isso parece muito similar a renderizar uma *template string*, mas Vue fez bastante trabalho interno. Os dados e o DOM estão agora interligados e tudo se tornou **reativo**. Como podemos ter certeza? Apenas abra o *console* JavaScript de seu navegador (agora mesmo, nesta página) e atribua um valor diferente em `app.message`. Você verá o exemplo renderizado acima se atualizando de acordo.

Fonte: (VUE.JS, 2018) online.

A documentação voltada para o usuário desse tipo de *software* é imprescindível para sua utilização e a qualidade da mesma pode refletir no sucesso do *software* em relação a sua adoção pelo mercado, afinal, partindo do exemplo exposto, sem a demonstração da maneira como se utiliza o *software*, não haveria nele serventia alguma.

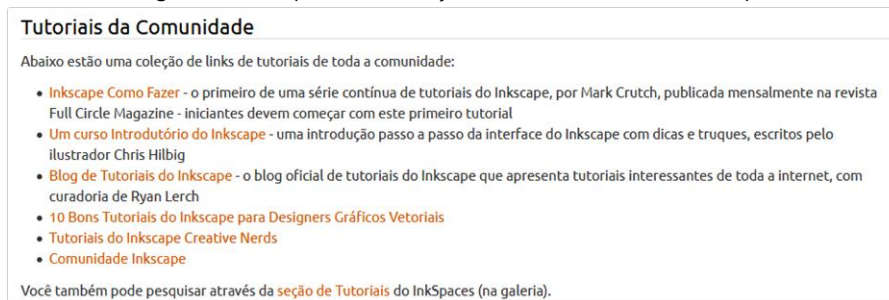
### 3.2.2 Documentação colaborativa

Outro ponto no qual a documentação do *software* para o usuário possui destaque, é no tocante a grandes projetos de *software* livre (*free software*) e *software* de código aberto (*open source*).

Nesses casos, forma-se uma comunidade de desenvolvedores que compartilham além de conhecimento e código, tempo e trabalho na produção de guias de usuário, em especial os manuais introdutórios e tutoriais para quem deseja iniciar no uso da tecnologia.

Um exemplo desse trabalho comunitário é o *software* livre para desenho vetorial *Inkscape*. A Figura 15 exibe um trecho de uma página no site do projeto que mantém o desenvolvimento do *software*, onde é possível observar links para conteúdo produzido pela comunidade.

**Figura 15** - Links para documentação colaborativa do software Inkscape.



**Fonte:** (INKSCAPE, 2018) online.

### 3.2.3 Segurança da informação

Considerando o exemplo de um Sistema de Informação (SI), existe ainda uma documentação segmentada por níveis de acesso ao sistema, ou seja, a documentação é distribuída para os usuários de acordo com o papel que eles irão desempenhar operando o *software*.

Um dos tipos de documentação é a de caráter técnico, específica para administradores de sistemas, onde normalmente aconselha-se o usuário a proceder com a leitura de arquivos contendo instruções de instalação e configurações iniciais, bem como as estruturas do BD e rotinas de manutenção.

Sendo assim, esperasse que haja determinado sigilo sobre a existência de alguns documentos, além de um rigoroso esquema de restrição ao acesso, permitindo

apenas aos usuários com a devida autorização, tenham contato com conteúdo de caráter técnico estrutural, bem como os demais conteúdos de caráter gerencial.

Há também a preocupação com a documentação de projetos de *software* de código fechado, que devem ser preservadas:

- nos limites da empresa desenvolvedora;
- em alguns casos nos limites de uma equipe ou departamento da empresa; e
- em outros casos, onde existe cooperação entre empresas, disponibilizada para os membros do projeto cooperativo, e mantida em sigilo para todo o restante.

### 3.3 GERENCIAMENTO DA DOCUMENTAÇÃO

Não menos importante que a produção dos documentos do *software*, é a tarefa de gerenciar o conteúdo. Gerir o conteúdo de documentação que foi produzido, as vezes de modo descentralizado, consiste em agrupá-lo em uma plataforma e apresentá-lo para o público interessado de forma simples, familiar, atraente, acessível e pesquisável, entre outras características.

Tudo isso deve ser feito considerando que pode haver a necessidade de editar a documentação a qualquer tempo e que as alterações devem ser disponibilizadas o mais breve possível, para que qualquer indivíduo que recorra a essa base de dados de especificações encontre informações atualizadas.

Sendo assim, a linguagem *Markdown* e as respectivas ferramentas para tratamento e produção de documentação que serão apresentadas no próximo capítulo, servirão como base para cumprir a proposta deste TG.

**Comentado [GVTDS14]:** Feito para valer um ponto na Disciplina do Fausto, estava pensando em tirar, mas...

**Comentado [GVTDS15]:** Precisa concluir o capítulo já introduzindo a Linguagem Markdown como ferramenta que pode solucionar o problema de descentralização da produção da documentação.

#### 4 LINGUAGEM MARKDOWN

Graduado em Ciência da Computação, o norte americano John Gruber mantém na internet um *weblog* de tecnologia chamado *Daring Fireball* (BLANC, 2008). O foco principal deste *weblog*, ou simplesmente *blog*, como costuma-se dizer, é a escrita de artigos sobre: tecnologias empregadas no sistema operacional *MacOS*, da empresa *Apple*; aplicativos desenvolvidos para o sistema *MacOS*; e outros produtos da empresa *Apple*.

Além disso, ou talvez em função disso (escrever artigos para seu *blog*), John Gruber inventou o que se conhece como linguagem de marcação leve *Markdown*, que foi lançada em março de 2004 e visa simplificar o trabalho de escrita de textos para *web*, que são publicados no formato HTML - *Hipertext Markup Language* (em português, Linguagem para Marcação de Hipertexto).

Na página dedicada ao projeto, que faz parte do *blog Daring Fireball*, (GRUBER, 2017) define *Markdown* como “uma ferramenta de conversão de texto simples em texto no formato HTML para escritores *web*”.

Para melhor compreender aquilo que de fato vem a ser *Markdown*, é preciso contextualizar o problema que John Gruber enfrentava na época, que deu vazio a iniciativa de criação da linguagem.

Conforme o relato de (GRUBER, 2004), manipular as *tags*<sup>2</sup> que compõe a sintaxe da linguagem HTML era uma tarefa fácil e, por esse motivo, ao escrever os artigos de seu *blog* ele costumava formatá-los diretamente com as *tags* HTML.

Apesar de ter facilidade para trabalhar com HTML, em determinado momento o criador da linguagem *Markdown* pôs-se a refletir sobre a morosidade do processo de escrita que antecedia suas postagens, que segundo ele consistia em: “(1) Escrever no *BBEdit*<sup>3</sup>; (2) Visualizar em um navegador; (3) Voltar para *BBEdit* e revisar o que foi escrito; (4) Repetir o processo até terminar; e (5) Fazer *login*<sup>4</sup> no *MT*<sup>5</sup>, colar o artigo e publicar” (GRUBER, 2004).

<sup>2</sup> **Tag:** Refere-se a definição atribuída aos elementos sintáticos do texto, utilizados na linguagem HTML que são responsáveis por delimitar o início e o fim da construção de um elemento da estrutura de um arquivo HTML (MDN WEB DOC - MOZILLA, 2015).

<sup>3</sup> **BBEdit:** Editor de textos HTML feito para o sistema operacional *Macintosh* da empresa *Apple* (BARE BONES, 2017).

<sup>4</sup> **Login:** Autenticação em um sistema.

<sup>5</sup> **MT:** Abreviação para *Movable Type*, que é um *software* para criação e gerenciamento de conteúdo para *web*. Em inglês CMS (*Content Management System*) (MOVABLETYPE.ORG, 2017).

Este processo contrastava com a opinião John Gruber de em relação ao uso de computadores. Segundo ele “a principal vantagem de usar um computador para escrever é o imediatismo da edição. Escreva, leia, revise, tudo na mesma janela, tudo no mesmo modo” (GRUBER, 2004).

Assim, Gruber percebeu que não havia vantagem em utilizar diretamente a formatação HTML para a criação dos artigos de seu *blog*, chegando à seguinte conclusão:

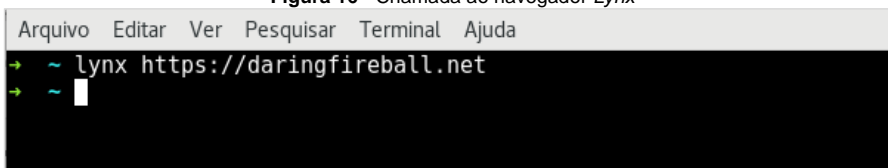
Há uma razão pela qual navegadores de texto simples como o *Lynx* não mostram apenas o código-fonte HTML bruto. Simplesmente não é para ser um formato legível. Não lhe parece estranho escrever em um formato que não é legível? De repente, me pareceu absurdo (GRUBER, 2004).

A conclusão de John Gruber, em especial a referência ao navegador web *Lynx*, que é um navegador *web* em modo texto (LYNX, 2017), não parece adequada, considerando que o propósito para o qual existem navegadores *web* é para que o conteúdo marcado com *tags* HTML seja renderizado e apresentado como texto legível.

Certamente, na maioria dos casos, o objetivo almejado ao se fazer uso um navegador *web*, não se trata de observar o código fonte das páginas, ainda que o acesso as páginas se de através de navegadores *web* simples como o *Lynx*.

Na CLI do SO *Debian GNU/Linux*, exibida na Figura 16, vê-se a sintaxe do comando que faz uma chamada ao navegador web *Lynx* para exibição da página inicial do site *daringfireball.net*.

Figura 16 - Chamada ao navegador *Lynx*



Fonte: Autoria própria.

Já na Figura 17, pode-se observar o navegador *web Lynx* exibindo, em um terminal do SO *Debian GNU/Linux*, uma parte da página inicial do site *daringfireball.net*.

**Figura 17** - Navegador web Lynx exibindo a página inicial do site *daringfireball.net*

```

#alternate alternate
Daring Fireball
By John Gruber
* Archive
* The Talk Show
* Projects
* Contact
* Colophon
* RSS Feed
* Twitter
* Sponsorship
KubeCon
Join leading technologists for Kubernetes, Docker and Cloud Native architectures.
Squarespace Domains
My thanks to Squarespace for sponsoring this week's DF RSS feed. Buying a domain name from
Squarespace is quick, simple, and fun. Search for the domain you want, or type any word or
phrase into the search field, and Squarespace will suggest some great options. Every domain
comes with a beautiful, ad-free parking page, WHOIS Privacy, and a 2048-bit SSL certificate to
secure your website - all at no additional cost. Once you lock down your domain, create a
beautiful website with one of Squarespace's award-winning templates.
Try Squarespace for free. When you're ready to subscribe, get 10 percent off at
-- pressione a barra de espaço para ir para a próxima página --
Setas para cima/baixo move. A direita segue um link; A esquerda para voltar.
H)Ajuda O)Opções P)Imprimir G)Segue M)Principal Q)Sair /=procura [delete]=Histórico

```

**Fonte:** Autoria própria.

A partir do conteúdo exibido na Figura 17, constata-se que o navegador *Lynx* exibe a página inicial do site *daringfireball.net* renderizada de forma simples e não o código fonte (marcação HTML) da página, conforme John Gruber havia concluído. Ou seja, apesar de óbvio fica comprovado que a marcação HTML não tem como objetivo ser apresentada à humanos.

Em face a este contexto, John Gruber se sentiu motivado a criar a linguagem *Markdown*, que em uma segunda definição, um pouco mais elaborada que a primeira, pois dessa vez considera-se também a sintaxe da linguagem, ele afirma: “*Markdown* são duas coisas: (1) uma sintaxe de formatação de texto simples; e (2) uma ferramenta de software, escrita em linguagem de programação *Perl* que converte a formatação de texto simples para HTML e XHTML<sup>6</sup> válido<sup>7</sup>”.

Por fim, (GRUBER, 2017) conclui: “Um documento com formatado com *Markdown* deve ser publicado como é, como texto sem formatação, sem parecer que tenha sido marcado com *tags* ou instruções de formatação”.

#### 4.1 LINGUAGEM DE MARCAÇÃO LEVE

<sup>6</sup> **XHTML:** (*Extensible Hiper Text Markup Language*) é um formato que estende a marcação HTML permitindo o tratamento dos documentos como *XML* (*Extensible Markup Language*) (W3C, 2000).

<sup>7</sup> **Válido:** Refere-se ao emprego correto das marcações (*tags*) HTML e XHTML no documento.

Em sua dissertação de Mestrado (ALEXANDRE, 2017) cita que “o princípio das linguagens de marcação leve (em inglês, *lightweight markup languages*) é que os textos sejam fáceis de serem digitados e lidos por humanos”. Assim, embora John Gruber não use exatamente essas palavras para definir seu invento, *Markdown* é considerada uma linguagem de marcação leve. O que é passível de constatação ao observar a sintaxe da linguagem que será demonstrada no tópico 4.3.

Em linhas gerais, o propósito de se utilizar *Markdown* é a conveniência de ter um texto estruturado de fácil compreensão para leitura, ao passo que a redação também seja facilitada pelo uso de uma sintaxe minimalista empregada na marcação de elementos tais como: parágrafos, títulos, citações, imagens, *links*, entre outros. Soma-se a isto, o fato de que no final do processo, através do uso de uma ferramenta de software para conversão, obtém-se o mesmo texto em formato HTML, pronto para a publicação.

Existem inúmeras linguagens de marcação leve disponíveis, duas delas que possuem certa relevância são a *reStructuredText*, muito utilizada para documentação de projetos escritos em na linguagem *Python* e a *wikitexto*, utilizada principalmente para edições de artigos na *Wikipédia*.

Comentado [GVTDS16]: Fazer citação.

## 4.2 ARQUIVOS MARKDOWN

Geralmente, os textos escritos com a sintaxe de formatação *Markdown* são salvos em arquivos com a extensão “.md”, porém, é possível usar a extensão “.markdown”. Arquivos com ambas as extensões podem ser consumidos por ferramentas de conversão para o formato HTML.

Em inglês, as ferramentas de software que executam a conversão do formato *Markdown* para o formato HTML são denominadas pela palavra *parser*.

## 4.3 SINTAXE

Para a definição da sintaxe de formatação da linguagem *Markdown*, (GRUBER, 2017) cita que contou com a ajuda de pares como: Aron Swartz, Nathaniel Irons, Dan Benjamin, Daniel Bogan e Jason Perkins. Além dessa ajuda, ele também relatou que obteve inspiração para elaboração da sintaxe no formato de *e-mail* de texto simples.

Comentado [GVTDS17]: Não sei se é relevante citar isso.



Tal formato consiste em um texto onde os pontos de destaque - títulos, seções do texto, palavras importantes e *links* -, são sinalizados pelo uso de caracteres especiais ao seu redor, conforme pode-se observar na Figura 18.

**Figura 18** - Formato de *e-mail* de texto simples.

```

Avangate webinar - Top Myths & Misconceptions About Affiliate Marketing
<http://www.avangate.com/lp/webinar-myths-and-misconceptions-about-affiliate-marketing.html>
=====

Hello Justine ,

Every day, news articles talk about affiliate marketing, all the money being made there, and how it's the hot new way to grow your business. But does
all this hype make you take pause? Do you wonder if affiliate marketing is worth the effort? Maybe the reason you've hesitated is because of
common misconceptions. There is an array of erroneous information out there about what affiliate marketing really is and how it works – it's time to
learn the facts.

Industry veteran and affiliate marketing thought leader, Geno Prussakov, will take you through some of the most popular myths and misconceptions
surrounding affiliate marketing, analyze and debunk them, and help you get the most out of your affiliate marketing endeavors.

Be sure to save your seat!
<http://www.avangate.com/lp/webinar-myths-and-misconceptions-about-affiliate-marketing.html>

The Avangate Team

=====

When:
Tue, October 14, 2014
10:00 AM - 11:00 AM PDT

Speaker:
Geno Prussakov
Founder and CEO, AM Navigator
<http://www.amnavigator.com>

=====
Register for the webinar:
<http://www.avangate.com/lp/webinar-myths-and-misconceptions-about-affiliate-marketing.html>
=====

<http://www.avangate.com>
<http://blog.avangate.com>
<http://www.facebook.com/avangate>

```

Fonte: (SMITH, 2014) *online*.

O referido formato de *e-mail* de texto simples, faz parte da estratégia de envio de *e-mail marketing*, na qual faz-se uso de uma funcionalidade presente em servidores de *e-mail*, o **Multi-part MIME (Multipurpose Internet Mail Extension)**.

Assim, envia-se as mensagens em dois formatos de mídia: *text/plain* e *text/html*. Deste modo, caso o cliente de *e-mail* do destinatário não seja capaz de exibir a mensagem no formato *text/html*, possivelmente contendo imagens e outros gráficos, exibirá a mensagem no formato *text/plain*, semelhante ao que foi apresentado na Figura 18.

A seguir, serão expostos exemplos de utilização da sintaxe de formatação da linguagem *Markdown*. Os exemplos irão abranger alguns dos elementos de um **texto estruturado**, expondo em quadros de duas colunas, respectivamente, o texto escrito

**Comentado [GVTDS18]:** Refletir sobre a relevância disso para o trabalho.

**Comentado [GVTDS19]:** Falar sobre texto estruturado.

em *Markdown* e os resultados em HTML obtidos após a conversão. Na sequência, será apresentada para cada exemplo uma figura contendo a visão do texto HTML renderizado pelo navegador.

O objetivo de tal demonstração será introduzir o leitor deste TG à um conjunto mínimo de elementos sintáticos utilizado com maior frequência em textos escritos com a linguagem *Markdown*.

Trata-se de um experimento para o qual foram utilizadas a especificação da linguagem *Markdown* definida por (GRUBER, 2017) e a ferramenta de conversão *Dingus*, disponível online no blog *Daring Fireball*.

Comentado [GVTDS20]: Encontrar uma forma melhor de citar.

4.3.1 Parágrafos e linhas

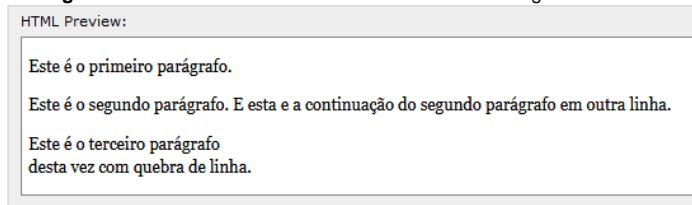
Em *Markdown* um parágrafo é constituído de uma ou várias linhas escritas consecutivamente podendo ou não conter caracteres de quebra de linha. Assim, para obter mais de um parágrafo no texto, deve haver uma linha em branco, sem qualquer caractere imprimível, separando duas linhas com caracteres imprimíveis. Uma quebra de linha no texto HTML pode ser obtida separando as palavras por dois espaços.

O Quadro 4 exibe um exemplo de emprego dos parágrafos.

Quadro 4 - <i>Markdown</i> : Sintaxe de formatação - Parágrafos e linhas.	
Texto em <i>Markdown</i>	Texto convertido para HTML
Este é o primeiro parágrafo.	<p>Este é o primeiro parágrafo.</p> <p>Este é o segundo parágrafo. E esta e a continuação do segundo parágrafo em outra linha.</p> <p>Este é o terceiro parágrafo desta vez com quebra de linha.</p>
Este é o segundo parágrafo.	
E esta e a continuação do segundo parágrafo em outra linha.	
Este é o terceiro parágrafo, desta vez com quebra de linha.	

Fonte: Autoria própria.

A Figura 19 apresenta três os parágrafos renderizados.

**Figura 19 - Markdown:** Saída HTML renderizada - Parágrafos e linhas.

Fonte: Autoria própria.

#### 4.3.2 HTML incorporado

A especificação da sintaxe de formatação *Markdown* contempla um conjunto limitado de *tags* HTML, por esse motivo, quando se pretende usar um elemento textual presente na sintaxe HTML, que não foi implementado em *Markdown*, pode-se inserir diretamente as *tags* HTML.

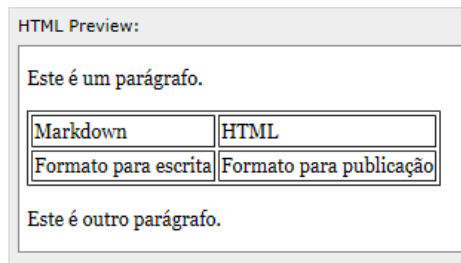
Para utilização de elementos de blocos tais como as *tags*: `<div>`, `<table>`, `<ul>` e `<pre>`, deve-se separar o bloco formatado com *tags* HTML do restante do conteúdo formatado com *Markdown*, inserindo uma linha em branco conforme demonstrado no Quadro 5.

**Quadro 5 - Markdown:** Sintaxe de formatação - HTML em bloco.

Texto em <i>Markdown</i>	Texto convertido para HTML
<p>Este é um parágrafo.</p> <pre>&lt;table border="1"&gt;   &lt;tr&gt;     &lt;td&gt;Markdown&lt;/td&gt;     &lt;td&gt;HTML&lt;/td&gt;   &lt;/tr&gt;   &lt;tr&gt;     &lt;td&gt;Formato para escrita&lt;/td&gt;     &lt;td&gt;Formato para publicação&lt;/td&gt;   &lt;/tr&gt; &lt;/table&gt;</pre> <p>Este é outro parágrafo.</p>	<pre>&lt;p&gt;Este é um parágrafo.&lt;/p&gt; &lt;table border="1"&gt;   &lt;tr&gt;     &lt;td&gt;Markdown&lt;/td&gt;     &lt;td&gt;HTML&lt;/td&gt;   &lt;/tr&gt;   &lt;tr&gt;     &lt;td&gt;Formato para escrita&lt;/td&gt;     &lt;td&gt;Formato para publicação&lt;/td&gt;   &lt;/tr&gt; &lt;/table&gt; &lt;p&gt;Este é outro parágrafo.&lt;/p&gt;</pre>

Fonte: Autoria própria.

O resultado renderizado, onde são apresentados um parágrafo, uma tabela e outro parágrafo, pode ser observado na Figura 20.

**Figura 20** - *Markdown*: Saída HTML renderizada - HTML em Bloco.

Fonte: Autoria própria.

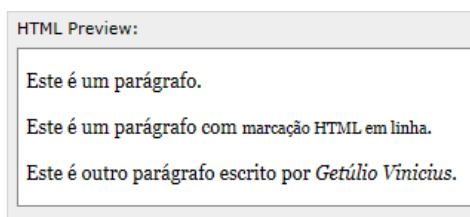
Para utilização de elementos em linha tais como as *tags*: `<span>`, `<small>`, `<cite>` e `<del>` não existe restrição de posicionamento, podendo ser inseridos juntamente com linhas formatadas com sintaxe *Markdown*, conforme demonstrado no Quadro 6.

**Quadro 6** - *Markdown*: Sintaxe de formatação - HTML em linha.

Texto em <i>Markdown</i>	Texto convertido para HTML
Este é um parágrafo com <code>&lt;small&gt;</code> marcação HTML em linha <code>&lt;/small&gt;</code> .	<code>&lt;p&gt;</code> Este é um parágrafo com <code>&lt;small&gt;</code> marcação HTML em linha <code>&lt;/small&gt;</code> . <code>&lt;/p&gt;</code>
Este é outro parágrafo escrito por <code>&lt;cite&gt;</code> Getúlio Vinicius <code>&lt;/cite&gt;</code> .	<code>&lt;p&gt;</code> Este é outro parágrafo escrito por <code>&lt;cite&gt;</code> Getúlio Vinicius <code>&lt;/cite&gt;</code> . <code>&lt;/p&gt;</code>

Fonte: Autoria própria.

O resultado renderizado, onde são apresentados dois parágrafos contendo elementos HTML em linha pode ser observado na Figura 21.

**Figura 21** - *Markdown*: Saída HTML renderizada - HTML em linha.

Fonte: Autoria própria.

Uma observação importante em relação a sintaxe *Markdown* original e o *parser* escrito em *Perl* que foi disponibilizado por John Gruber, é que estes permitem a utilização de *tags* HTML mesmo para aqueles elementos que foram especificados. Assim, por exemplo, caso se opte por utilizar a *tag* `<img>` da sintaxe de formatação

HTML, ao invés de usar a sintaxe definida para a linguagem *Markdown* em relação a inserção de imagens, que será vista no tópico 4.3.8, o *parser* irá processar normalmente a conversão.

#### 4.3.3 Ênfase

A sintaxe *Markdown* permite o emprego de ênfase de dois modos distintos, correspondendo respectivamente as *tags* `<em>` e `<strong>` da sintaxe HTML. Os caracteres utilizados para obter a formatação são “\_” (*underscore*) e “\*” (asterisco).

Para obter o efeito de ênfase indicando um contraste implícito ou explícito, o texto a ser destacado deve ser envolvido com 1 (um) caractere em cada extremidade - qualquer que seja entre os dois aceitos -, e, do mesmo modo, para obter o efeito de ênfase indicando que a palavra ou o trecho possui grande importância, o texto a ser destacado deve ser envolvido com 2 (dois) caracteres em cada extremidade.

O Quadro 7 demonstra o emprego dos dois tipos de ênfase com a linguagem *Markdown*.

**Quadro 7 - Markdown:** Sintaxe de formatação - Ênfase.

Texto em <i>Markdown</i>	Texto convertido para HTML
No meu tempo existiam os profissionais chamados _Webmaster_. Hoje em dia são chamados de desenvolvedores *front-end*.	<p>No meu tempo existiam os profissionais chamados <em>Webmaster</em>. Hoje em dia são chamados de desenvolvedores <em>front-end</em>.</p>
_Documentar_ o projeto de software é fundamental para o **correto desenvolvimento**.	<p><strong>Documentar</strong> o projeto de software é fundamental para o <strong>correto desenvolvimento</strong>.</p>

**Fonte:** Autoria própria.

O resultado renderizado, onde é possível observar o emprego dos dois tipos de ênfase é apresentado na Figura 22.

**Figura 22 - Markdown:** Saída HTML renderizada - Ênfase.

HTML Preview:
No meu tempo existiam os profissionais chamados <i>Webmaster</i> . Hoje em dia são chamados de desenvolvedores <i>front-end</i> .
<b>Documentar</b> o projeto de software é fundamental para o <b>correto desenvolvimento</b> .

**Fonte:** Autoria própria.

O emprego de ênfase na estruturação de texto para a *web* geralmente provoca confusão em profissionais, sejam eles programadores – as vezes iniciantes -, ou ainda

escritores jornalistas, blogueiros (pessoas que escrevem artigos para *blogs*) entre outros.

A confusão se dá principalmente pelo resultado obtido quando o texto é renderizado. A ênfase empregada pelo uso da *tag* HTML `<em>`, geralmente é apresentada em formato de fonte itálico, enquanto que a ênfase empregada pelo uso da *tag* HTML `<strong>`, geralmente é apresentada em formato de fonte negrito.

A obtenção de negrito e itálico com *tags* HTML se dá respectivamente pelo uso de outras duas *tags* distintas: `<i>` e `<b>`. Estas não carregam consigo nenhum tipo de apelo semântico em relação ao texto, como a indicação de contraste e grau de importância, segundo a documentação da linguagem HTML oferecida pela (MDN WEB DOC - MOZILLA, 2017) e (MDN WEB DOC - MOZILLA, 2017)

#### 4.3.4 Cabeçalhos

Para obtenção de cabeçalhos com a linguagem *Markdown*, pode-se recorrer a dois modelos distintos derivados de outras linguagens de marcação leve denominadas *setext* e *atx*.

O primeiro modelo, derivado da *setext*, permite apenas cabeçalhos de nível 1 e nível 2, e consiste em redigir o texto do cabeçalho, sem qualquer marcação especial, em uma linha, e, na linha seguinte sublinhar o texto com caracteres “=” (igual) para cabeçalho de nível 1 e “-” (hífen) para cabeçalho de nível 2.

É necessário ao menos um caractere sublinhando o texto do cabeçalho e a linha do sublinhado pode conter apenas caracteres referente ao nível de cabeçalho desejado, conforme pode ser visto no Quadro 8.

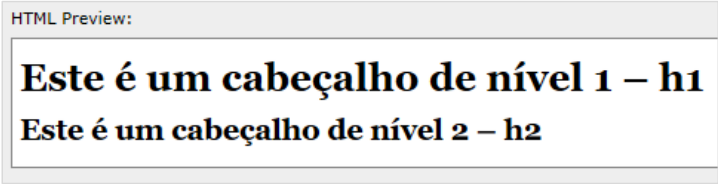
**Quadro 8** - *Markdown*: Sintaxe de formatação - Cabeçalhos do modelo *setext*.

Texto em Markdown	Texto convertido para HTML
Este é um cabeçalho de nível 1 – h1 =====	<code>&lt;h1&gt;Este é um cabeçalho de nível 1 – h1&lt;/h1&gt;</code>
Este é um cabeçalho de nível 2 – h2 -----	<code>&lt;h2&gt;Este é um cabeçalho de nível 2 – h2&lt;/h2&gt;</code>

Fonte: Autoria própria.

O resultado renderizado, onde são apresentados dois cabeçalhos, pode ser visto na Figura 23.

Figura 23 - Markdown: Saída HTML renderizada - Cabeçalho modelo *setext*.



Fonte: Autoria própria.

O segundo modelo, derivado da *atx*, permite a utilização dos seis níveis de cabeçalho implementados nas especificações do HTML. Consiste e marcar o início da linha que contém o texto do cabeçalho com a quantidade de caracteres “#” (tralha) referente ao nível de cabeçalho que se deseja obter, sendo 1 (um) caractere para cabeçalho de nível 1, 2 (dois) caracteres para cabeçalho de nível 2 e assim sucessivamente até o cabeçalho de nível 6, conforme pode-se observar no Quadro 9.

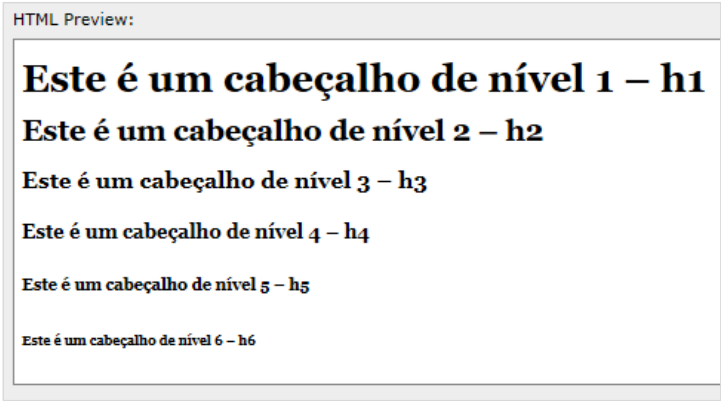
Quadro 9 - Markdown: Sintaxe de formatação - Cabeçalho no modelo *atx*.

Texto em Markdown	Texto convertido para HTML
# Este é um cabeçalho de nível 1 – h1	
## Este é um cabeçalho de nível 2 – h2	<h1>Este é um cabeçalho de nível 1 – h1</h1>
### Este é um cabeçalho de nível 3 – h3	<h2>Este é um cabeçalho de nível 2 – h2</h2>
#### Este é um cabeçalho de nível 4 – h4	<h3>Este é um cabeçalho de nível 3 – h3</h3>
##### Este é um cabeçalho de nível 5 – h5	<h4>Este é um cabeçalho de nível 4 – h4</h4>
##### Este é um cabeçalho de nível 6 – h6	<h5>Este é um cabeçalho de nível 5 – h5</h5>
	<h6>Este é um cabeçalho de nível 6 – h6</h6>

Fonte: Autoria própria.

O resultado renderizado, onde são apresentados os seis níveis de cabeçalho, pode ser visto na Figura 24.

Figura 24 - *Markdown*: Saída HTML renderizada - Cabeçalho modelo atx.



Fonte: Autoria própria.

4.3.5 Listas

Obter listas em um texto formatado com a sintaxe *Markdown* é extremamente simples. Blocos de listas, quando precedidos por texto corrido, devem ser iniciados após uma linha em branco, delimitando a separação da lista do parágrafo que a antecede. O mesmo critério vale para parágrafos que sucedem uma lista.

Para listas ordenadas basta utilizar os caracteres numéricos sucedidos por um ponto no início da linha, conforme demonstrado no Quadro 10.

Quadro 10 - *Markdown*: Sintaxe de formatação - Listas ordenadas.

Texto em Markdown	Texto convertido para HTML
1. Primeiro item da lista. 2. Segundo item da lista. 3. Terceiro item da lista. 4. Quarto...	<ol> <li>Primeiro item da lista.</li> <li>Segundo item da lista.</li> <li>Terceiro item da lista.</li> <li>Quarto...</li> </ol>
Este parágrafo sucede uma lista ordenada e por isso existe uma linha em branco separando-o da lista.	<p>Este parágrafo sucede uma lista ordenada e por isso existe uma linha em branco separando-o da lista.</p>
Este parágrafo antecede uma lista ordenada e por isso existe uma linha em branco abaixo separando-o da lista.	<p>Este parágrafo antecede uma lista ordenada e por isso existe uma linha em branco abaixo separando-o da lista.</p>
1. Primeiro item da lista. 2. Segundo item da lista.	<ol> <li>Primeiro item da lista.</li> <li>Segundo item da lista.</li> </ol>

Fonte: Autoria própria.



O resultado renderizado pode ser observado na Figura 25, onde são apresentadas duas listas e dois parágrafos.

**Figura 25 - Markdown:** Saída HTML renderizada - Listas ordenadas.

HTML Preview:
<pre> 1. Primeiro item da lista. 2. Segundo item da lista. 3. Terceiro item da lista. 4. Quarto...  Este parágrafo sucede uma lista ordenada e por isso existe uma linha em branco separando-o da lista.  Este parágrafo antecede uma lista ordenada e por isso existe uma linha em branco abaixo separando-o da lista.  1. Primeiro item da lista. 2. Segundo item da lista. </pre>

**Fonte:** Autoria própria.

Para listas não ordenadas os critérios de separação de blocos e parágrafos seguem inalterados, mas os caracteres de início da linha que constitui um item da lista são diferentes. Pode-se utilizar, inclusive de forma mesclada, os caracteres “+” (sinal de adição), “-” (hífen) e “\*” (asterisco), conforme demonstrado no Quadro 11.

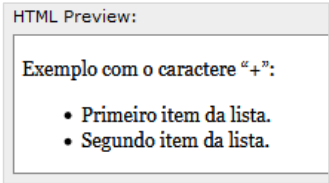
**Quadro 11 - Markdown:** Sintaxe de formatação - Listas não ordenadas.

Texto em Markdown	Texto convertido para HTML
Exemplo com o caractere “+”:	<pre>&lt;p&gt;Exemplo com o caractere “+”:&lt;/p&gt; &lt;ul&gt; &lt;li&gt;Primeiro item da lista.&lt;/li&gt; &lt;li&gt;Segundo item da lista.&lt;/li&gt; &lt;/ul&gt;</pre>
Exemplo com o caractere “-”:	<pre>&lt;p&gt;Exemplo com o caractere “-”:&lt;/p&gt; &lt;ul&gt; &lt;li&gt;Primeiro item da lista.&lt;/li&gt; &lt;li&gt;Segundo item da lista.&lt;/li&gt; &lt;/ul&gt;</pre>
Exemplo com o caractere “*”:	<pre>&lt;p&gt;Exemplo com o caractere “*”:&lt;/p&gt; &lt;ul&gt; &lt;li&gt;Primeiro item da lista.&lt;/li&gt; &lt;li&gt;Segundo item da lista.&lt;/li&gt; &lt;/ul&gt;</pre>
Exemplo com os caracteres mesclados:	<pre>&lt;p&gt;Exemplo com os caracteres mesclados:&lt;/p&gt; &lt;ul&gt; &lt;li&gt;Primeiro item da lista.&lt;/li&gt; &lt;li&gt;Segundo item da lista.&lt;/li&gt; &lt;li&gt;Terceiro item da lista.&lt;/li&gt; &lt;/ul&gt;</pre>

**Fonte:** Autoria própria.

A Figura 26 mostra o resultado renderizado do primeiro exemplo do Quadro 11. Uma lista não ordenada com dois itens construída com o caractere “+”.

Figura 26 - Markdown: Saída HTML renderizada - Lista não ordenada.



Fonte: Autoria própria.

É possível obter listas encadeadas, níveis de lista, tanto para listas ordenadas quanto para listas não ordenadas ou, de forma conjunta mesclando os dois tipos de listas.

Assim, para obter uma lista de nível 2, imediatamente após um item da lista de nível 1, deve-se aplicar um recuo com o caractere não imprimível de tabulação, utilizando a tecla “tab” e então as próximas linhas que constituírem um item da lista de nível 2 também deverão ser precedidas do mesmo caractere de tabulação.

Um elemento de uma lista de nível 3 deve ser precedido por dois caracteres de tabulação, e assim sucessivamente para os demais níveis, conforme demonstrado no Quadro 12.

Quadro 12 - Markdown: Sintaxe de formatação - Listas ordenadas e não ordenadas.

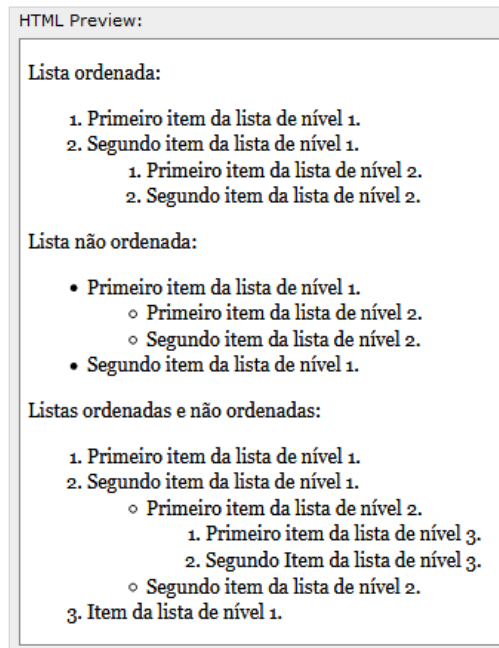
Texto em Markdown	Texto convertido para HTML
Lista ordenada:  1. Primeiro item da lista de nível 1. 2. Segundo item da lista de nível 1. 1. Primeiro item da lista de nível 2. 2. Segundo item da lista de nível 2.	<p>Lista ordenada:</p> <ol> <li>Primeiro item da lista de nível 1.</li> <li>Segundo item da lista de nível 1. <ol><li>Primeiro item da lista de nível 2.</li> <li>Segundo item da lista de nível 2.</li></ol></li> </ol>
Lista não ordenada:  * Primeiro item da lista de nível 1. * Primeiro item da lista de nível 2. * Segundo item da lista de nível 2. * Segundo item da lista de nível 1.	<p>Lista não ordenada:</p> <ul> <li>Primeiro item da lista de nível 1. <ul><li>Primeiro item da lista de nível 2.</li> <li>Segundo item da lista de nível 2.</li></ul></li> <li>Segundo item da lista de nível 1.</li> </ul>
Listas ordenadas e não ordenadas:  1. Primeiro item da lista de nível 1. 2. Segundo item da lista de nível 1. * Primeiro item da lista de nível 2. 1. Primeiro item da lista de nível 3.	<p>Listas ordenadas e não ordenadas:</p> <ol> <li>Primeiro item da lista de nível 1.</li> <li>Segundo item da lista de nível 1. <ul> <li>Primeiro item da lista de nível 2. <ol> <li>Primeiro item da lista de nível 3.</li> </ol> </li> </ul> </li> </ol>

2. Segundo Item da lista de nível 3. * Segundo item da lista de nível 2. 3. Item da lista de nível 1.	<ul><li>Primeiro item da lista de nível 2. <ol><li>Primeiro item da lista de nível 3.</li> <li>Segundo Item da lista de nível 3.</li></ol></li> <li>Segundo item da lista de nível 2.</li></ul></li> <li>Item da lista de nível 1.</li> </ol>
---	--

Fonte: Autoria própria.

A Figura 27 exibe o resultado daquilo que foi demonstrado no Quadro 12. Uma lista ordenada com dois níveis, uma lista não ordenada com dois níveis e uma lista que mescla listas ordenadas e não ordenadas com três níveis.

**Figura 27** - *Markdown*: Saída HTML renderizada - Listas ordenadas e não ordenadas.



Fonte: Autoria própria.

#### 4.3.6 Citações em bloco

A existência de um elemento para formatação de bloco de citação na sintaxe da linguagem *Markdown*, denota claramente a intenção de John Gruber em criar uma ferramenta para simplificar o trabalho daqueles que escrevem texto para a internet,

sendo que citações são elementos que se observa mais frequentemente em artigos de *blogs* e sites de notícias.

A tarefa de destacar um texto como citação em bloco usando *Markdown* consiste em iniciar a linha com um caractere ">" (maior que) e, para interromper a sequência do bloco de citação basta inserir um parágrafo normal, ou seja, uma linha em branco conforme a demonstração constante no Quadro 13.

**Quadro 13 - Markdown:** Sintaxe de formatação - Citação em Bloco.

Texto em Markdown	Texto convertido para HTML
<p>Este texto constitui um parágrafo normal. Abaixo será mostrado um exemplo de citação em bloco:</p> <p>&gt; Este texto é um exemplo de como funciona a citação em bloco. Um bloco de citação pode conter várias linhas.</p> <p>Este é um outro parágrafo. Abaixo virá uma citação com dois parágrafos:</p> <p>&gt; Este é o primeiro parágrafo da citação.</p> <p>&gt; Citação em bloco pode conter vários parágrafos.</p>	<pre>&lt;p&gt;Este texto constitui um parágrafo normal. Abaixo será mostrado um exemplo de citação em bloco:&lt;/p&gt; &lt;blockquote&gt;   &lt;p&gt;Este texto é um exemplo de como funciona a citação em bloco. Um bloco de citação pode conter várias linhas.&lt;/p&gt; &lt;/blockquote&gt; &lt;p&gt;Este é um outro parágrafo. Abaixo virá uma citação com dois parágrafos:&lt;/p&gt; &lt;blockquote&gt;   &lt;p&gt;Este é o primeiro parágrafo da citação.&lt;/p&gt;   &lt;p&gt;Citação em bloco pode conter vários parágrafos.&lt;/p&gt; &lt;/blockquote&gt;</pre>

**Fonte:** Autoria própria.

O resultado renderizado, onde são apresentados dois blocos de citação, pode ser visto na Figura 28.

**Figura 28 - Markdown:** Saída HTML renderizada - Citação em bloco.

HTML Preview:
<p>Este texto constitui um parágrafo normal. Abaixo será mostrado um exemplo de citação em bloco:</p> <p>Este texto é um exemplo de como funciona a citação em bloco. Um bloco de citação pode conter várias linhas.</p> <p>Este é um outro parágrafo. Abaixo virá uma citação com dois parágrafos:</p> <p>Este é o primeiro parágrafo da citação.</p> <p>Citação em bloco pode conter vários parágrafos.</p>

**Fonte:** Autoria Própria.

Outra possibilidade de uso contemplada pela sintaxe de formatação da linguagem *Markdown*, para blocos de citação, é a composição da estrutura em

conjunto com outros elementos já apresentados, tais como: listas, cabeçalhos e destaques de texto.

O Quadro 14 traz uma demonstração de bloco de citação composto por mais elementos da sintaxe *Markdown*.

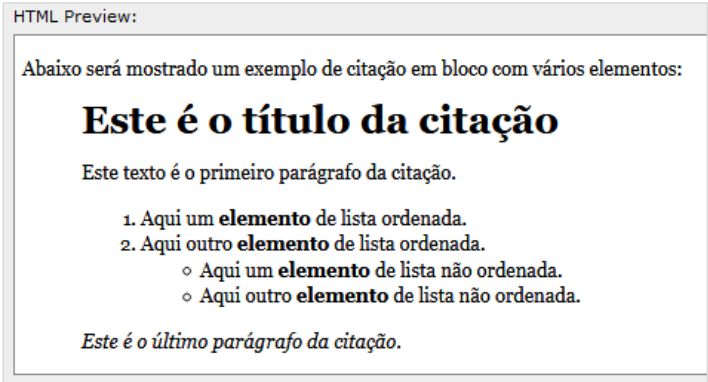
Quadro 14 - *Markdown*: Sintaxe de formatação - Citação em Bloco composta por mais elementos.

Texto em Markdown	Texto convertido para HTML
<p>Abaixo será mostrado um exemplo de citação em bloco com vários elementos:</p> <p>&gt; # Este é o título da citação</p> <p>&gt; Este texto é o primeiro parágrafo da citação.</p> <p>&gt; 1. Aqui um <b>elemento</b> de lista ordenada.</p> <p>&gt; 2. Aqui outro <b>elemento</b> de lista ordenada.</p> <p>&gt; + Aqui um <b>elemento</b> de lista não ordenada.</p> <p>&gt; + Aqui outro <b>elemento</b> de lista não ordenada.</p> <p>&gt; _Este é o último parágrafo da citação._</p>	<pre>&lt;p&gt;Abaixo será mostrado um exemplo de citação em bloco com vários elementos:&lt;/p&gt; &lt;blockquote&gt;   &lt;h1&gt;Este é o título da citação&lt;/h1&gt;   &lt;p&gt;Este texto é o primeiro parágrafo da citação.&lt;/p&gt;   &lt;ol&gt;     &lt;li&gt;Aqui um &lt;strong&gt;elemento&lt;/strong&gt; de lista ordenada.&lt;/li&gt;     &lt;li&gt;Aqui outro &lt;strong&gt;elemento&lt;/strong&gt; de lista ordenada.     &lt;ul&gt;&lt;li&gt;Aqui um &lt;strong&gt;elemento&lt;/strong&gt; de lista não ordenada.&lt;/li&gt;     &lt;li&gt;Aqui outro &lt;strong&gt;elemento&lt;/strong&gt; de lista não ordenada.&lt;/li&gt;&lt;/ul&gt;&lt;/li&gt;   &lt;/ol&gt;   &lt;p&gt;&lt;em&gt;Este é o último parágrafo da citação.&lt;/em&gt;&lt;/p&gt; &lt;/blockquote&gt;</pre>

Fonte: Autoria própria.

A Figura 29 exibe o resultado renderizado do código demonstrado no Quadro 14.

Figura 29 - *Markdown*: Saída HTML renderizada - Citação em bloco composta por mais elementos.



Fonte: Autoria própria.

#### 4.3.7 Links

Quanto aos *links*, a linguagem *Markdown* implementa o recurso utilizando duas sintaxes de formatação distintas: *inline* e por referência. A sintaxe *inline* consiste em envolver entre colchetes “[ ]” o texto chamado de âncora e, imediatamente após envolver o endereço (caminho para o arquivo) por parênteses “( )”, sendo que:

- O texto âncora é o texto que será exibido renderizado na página e o endereço corresponde ao atributo *href* da *tag* HTML `<a>`;
- Opcionalmente pode-se informar um título para o *link* que corresponderá ao atributo *title* da *tag* `<a>`;
- O título de um link deve ser posto entre aspas, separado do endereço por um caractere de espaço e envolvido pelo mesmo conjunto de parênteses do endereço;
- O endereço pode ser local ou remoto (da rede ou *internet*);
- Sendo um arquivo remoto, deve-se indicar a *URL* - *Uniform Resource Location* (em português, Localizador Uniforme de Recursos), ou seja, o endereço de rede pelo qual se acessa um recurso, neste caso o arquivo ou página que se deseja referenciar através do link; e
- Sendo um endereço local, (arquivo armazenado no mesmo servidor), pode-se usar um caminho relativo a partir do diretório corrente, onde será armazenado o arquivo de extensão “.md” que contém o *link*.

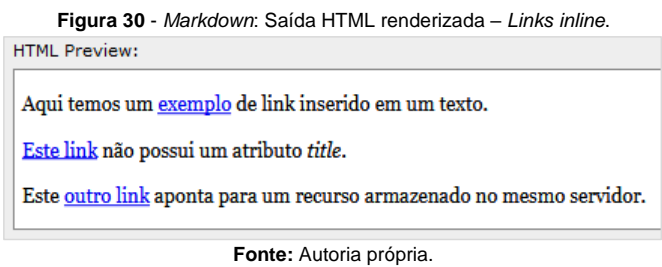
O Quadro 15 traz uma demonstração acerca do uso de *links inline* com a linguagem *Markdown*.

**Quadro 15** - *Markdown*: Sintaxe de formatação – *Links inline*.

Texto em Markdown	Texto convertido para HTML
Aqui temos um [exemplo] (http://exemplo.com/ "Link de exemplo") de link inserido em um texto.	<p>Aqui temos um <a href="http://exemplo.com/" title="Link de exemplo">exemplo</a> de link inserido em um texto.</p>
[Este link](http://example.net/) não possui um atributo _title_.	<p><a href="http://example.net/">Este link</a> não possui um atributo <em>title</em>.</p>
Este [outro link](/outra-pagina/) aponta para um recurso armazenado no mesmo servidor.	<p>Este <a href="/outra-pagina/">outro link</a> aponta para um recurso armazenado no mesmo servidor.</p>

**Fonte:** Autoria própria.

A Figura 30 exibe o exemplo renderizado, do demonstrado no Quadro 15 renderizado.



A sintaxe de *links* por referência pode ser usada para simplificar a leitura do texto escrito em *Markdown*, isto porque diminui a quantidade de elementos sintáticos e endereços longos no corpo do texto, substituindo-os por chaves curtas que os identificam em um bloco de texto separado na redação principal.

Para inserir *links* através de referências, deve-se envolver o texto de ancora em colchetes “[ ]” e logo na sequência, envolvido por um novo par de colchetes deve-se informar a chave de referência do *link*.

Em um bloco separado do parágrafo, ou qualquer outro elemento que contenha o *link*, deve-se construir a referência, que consiste em:

- Uma linha iniciada pela chave de identificação envolta por colchetes e sucedida pelo caractere “.” (dois pontos);
- Um caractere de espaço sucedido pelo endereço (local ou remoto) referente ao atributo *href* da tag HTML <a>; e
- Opcionalmente, o título do link, o atributo *title* da tag <a>, entre aspas.

O Quadro 16 demonstra o uso de links por referência na linguagem *Markdown*.

Texto em Markdown	Texto convertido para HTML
Aqui temos um [exemplo][ex1] de link inserido por referência.	<p>Aqui temos um <a href="http://exemplo.com" title="Título de exemplo">exemplo</a> de link inserido por referência.</p>
[Este link][ex2], também inserido por referência, não possui um atributo _title_.	<p><a href="/pagina-de-exemplo/">Este link</a>, também inserido por referência, não possui um atributo <em>title</em>.</p>

Aqui temos apenas um parágrafo. A indicação dos endereços vem logo abaixo e não aparece no texto renderizado.	<p>Aqui temos apenas um parágrafo. A indicação dos endereços vem logo abaixo e não aparece no texto renderizado.</p>
[ex1]: <a href="http://exemplo.com">http://exemplo.com</a> "Título de exemplo"	
[ex2]: <a href="/pagina-de-exemplo/">/pagina-de-exemplo/</a>	

Fonte: Autoria própria.

A Figura 31 exibe o resultado renderizado da marcação *Markdown* elaborada no Quadro 16.

**Figura 31 - Markdown:** Saída HTML renderizada – *Links* por referência.

HTML Preview:
Aqui temos um <a href="#">exemplo</a> de link inserido por referência.
<a href="#">Este link</a> , também inserido por referência, não possui um atributo <i>title</i> .
Aqui temos apenas um parágrafo. A indicação dos endereços vem logo abaixo e não aparece no texto renderizado.

Fonte: Autoria própria.

#### 4.3.8 Imagens

O recurso de inserção de imagens através da linguagem *Markdown* possui sintaxe semelhante a inserção de *links*. A diferença consiste na existência de um ponto de exclamação “!” precedendo todo o restante da marcação.

Na inserção de imagens, o texto envolto por colchetes representa o atributo *alt* da tag HTML *<img>*; o texto entre parênteses corresponde respectivamente a *URL* do arquivo de imagem, representando o atributo *src* da tag *<img>* e o título da imagem vem entre aspas, após o espaço, correspondendo ao atributo *title* da tag *<img>*.

O Quadro 17 exibe a formatação necessária para inserção de imagens a partir sintaxe da linguagem *Markdown*.

**Quadro 17 - Markdown:** Sintaxe de formatação - Imagens.

Texto em Markdown	Texto convertido para HTML
Inserção de uma imagem externa ao site a partir de uma url:	<p>Inserção de uma imagem externa ao site a partir de uma url:</p>
![Daring Fireball - Logo](https://daringfireball.net/graphics/logos/"Logo do site Daring Fireball")	<p></p>
Inserção de uma imagem local a partir do caminho relativo:	<p>Inserção de uma imagem local a partir do caminho relativo:</p>
	<p></p>

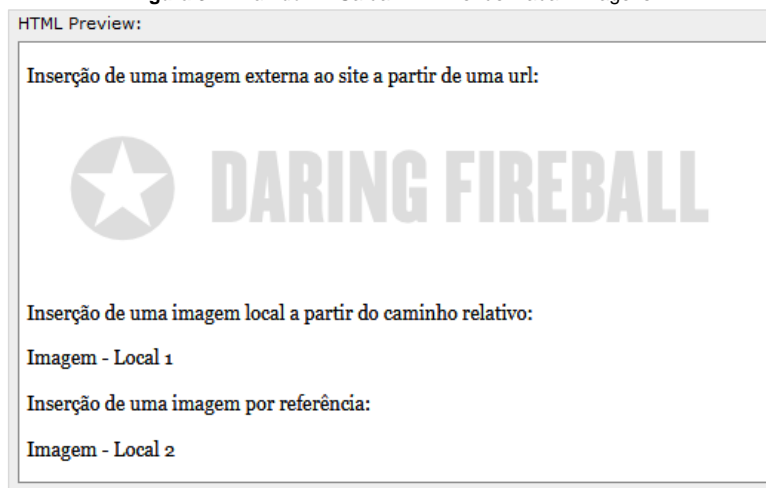


<p>![[Imagem - Local 1]](/imagens/logo1 "Título 1")</p> <p>Inserção de uma imagem por referência:</p> <p>![[Imagem - Local 2]][2]</p> <p>[2]: /imagens/logo2 "Título 2"</p>	<p>&lt;p&gt;Inserção de uma imagem por referência:&lt;/p&gt;</p> <p>&lt;p&gt;&lt;img src="/imagens/logo2" alt="Imagem - Local 2" title="Título 2" /&gt;&lt;/p&gt;</p>
---	---

**Fonte:** Autoria própria.

A Figura 32 exibe o resultado renderizado a partir da formatação demonstrada no Quadro 17.

**Figura 32 - Markdown:** Saida HTML renderizada - Imagens.



**Fonte:** Autoria própria.

É possível observar na Figura 32 que ao invés das imagens *logo1* e *logo2*, a renderização apresentou o texto alternativo, que foi definido entre colchetes na formatação. Isto porque os respectivos arquivos não foram localizados no servidor onde a ferramenta utilizada para promover a conversão do formato *Markdown* para o formato HTML, *Dingus*, está instalada.

#### 4.3.9 Código

*Markdown* possui sintaxe de formatação para exibição de código, que representa a tag HTML `<code>` e pode ser inserida *inline*, ou seja, destacando um trecho do parágrafo como fragmento de código, ou ainda, como elemento de bloco de

texto pré-formatado, onde o conteúdo da *tag* `<code>` é encapsulado pela *tag* `<pre>` da marcação HTML.

Segundo a documentação de marcação HTML oferecida pela (MDN WEB DOC - MOZILLA, 2017) “a *tag* HTML `<code>` é utilizada para representar um fragmento de código de computador. Por padrão, o conteúdo da *tag* `<code>` é exibido pelos navegadores de internet usando fonte mono espaçada”.

Para obtenção de um fragmento de código em um parágrafo qualquer é preciso envolver o trecho com o caractere `'` (apostrofe) e para a obtenção de um bloco de código, deve-se separar o trecho do texto normal com uma linha em branco, dessa forma, cada linha pertencente ao código deve ser precedida por um caractere de tabulação.

O Quadro 18 demonstra a sintaxe de formatação *Markdown* para exibição de código.

Quadro 18 - *Markdown*: Sintaxe de formatação - Código.

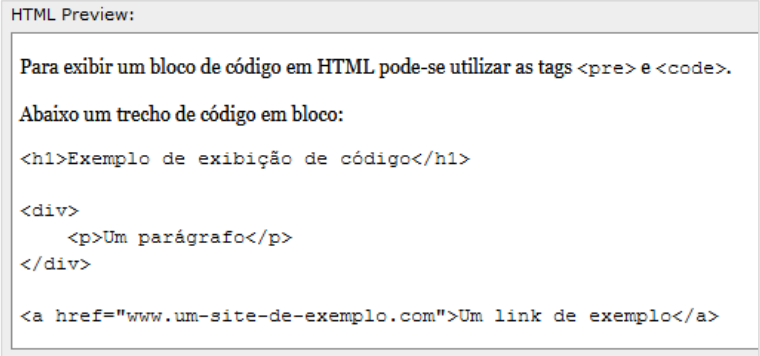
Texto em Markdown	Texto convertido para HTML
Para exibir um bloco de código em HTML pode-se utilizar as tags <code>&lt;pre&gt;</code> e <code>&lt;code&gt;</code> .	<code>&lt;p&gt;Para exibir um bloco de código em HTML pode-se utilizar as tags</code> <code>&lt;code&gt;&amp;lt;pre&amp;gt;&lt;/code&gt; e</code> <code>&lt;code&gt;&amp;lt;code&amp;gt;&lt;/code&gt;.&lt;/p&gt;</code>
Abaixo um trecho de código em bloco:	<code>&lt;p&gt;Abaixo um trecho de código em bloco:&lt;/p&gt;</code> <code>&lt;pre&gt;&lt;code&gt;&amp;lt;h1&amp;gt;Exemplo de exibição de código&amp;lt;/h1&amp;gt;</code>
<code>&lt;div&gt;</code> <code>&lt;p&gt;Um parágrafo&lt;/p&gt;</code> <code>&lt;/div&gt;</code>	<code>&amp;lt;div&amp;gt;</code> <code>&amp;lt;p&amp;gt;Um parágrafo&amp;lt;/p&amp;gt;</code> <code>&amp;lt;/div&amp;gt;</code>
<code>&lt;a href="www.um-site-de-exemplo.com"&gt;Um link de exemplo&lt;/a&gt;</code>	<code>&amp;lt;a href="www.um-site-de-exemplo.com"&amp;gt;Um link de exemplo&amp;lt;/a&amp;gt;</code> <code>&lt;/code&gt;&lt;/pre&gt;</code>

Fonte: Autoria própria.

O resultado renderizado, onde pode ser observada a exibição de código *inline* (em linha) e de um bloco e código é apresentado na Figura 33.

Comentado [GVTDS21]: Ou crase???

Figura 33 - Markdown: Saída HTML renderizada - Código.



Fonte: Autoria própria.

4.3.10 Caractere de escape

Por fim, o último exemplo em relação a definição *Markdown* de John Gruber será o caractere de escape. Em algumas situações o que se deseja é a representação fidedigna de um caractere e não que o mesmo seja interpretado pelo *parser* como sendo parte de uma marcação sintática. Nestes casos é preciso informar o *parser* que o caractere seguinte faz parte do contexto da redação e na linguagem *Markdown* isso é feito inserido uma “\” (contra barra) antes do caractere que se deseja escapar.

O Quadro 19 demonstra a inserção de caracteres utilizando o caractere de escape e uma listagem de todos os caracteres que para serem representados literalmente devem ser escapados.

Quadro 19 - Markdown: Sintaxe de formatação - Caractere de escape.

Texto em Markdown	Texto convertido para HTML
+ \ "Contra barra	<ul>
+ \' "Apóstrofe"	<li>\ "Contra barra</li>
+ \* "Asterisco"	<li>\' "Apóstrofe"</li>
+ \_ "Sublinhado"	<li>* "Asterisco"</li>
+ \{ "Chaves"	<li>_ "Sublinhado"</li>
+ \[ "Colchetes"	<li>{ "Chaves"</li>
+ \() "Parênteses"	<li>[ "Colchetes"</li>
+ \# "Tralha"	<li>() "Parênteses"</li>
+ \+ "Sinal de adição"	<li># "Tralha"</li>
+ \- "Hífen"	<li>+ "Sinal de adição"</li>
+ \. "ponto"	<li>- "Hífen"</li>
+ \! "Ponto de exclamação"	<li>. "ponto"</li>
	<li>! "Ponto de exclamação"</li>
	</ul>

Fonte: Autoria própria.

## 4.4 PARSERS

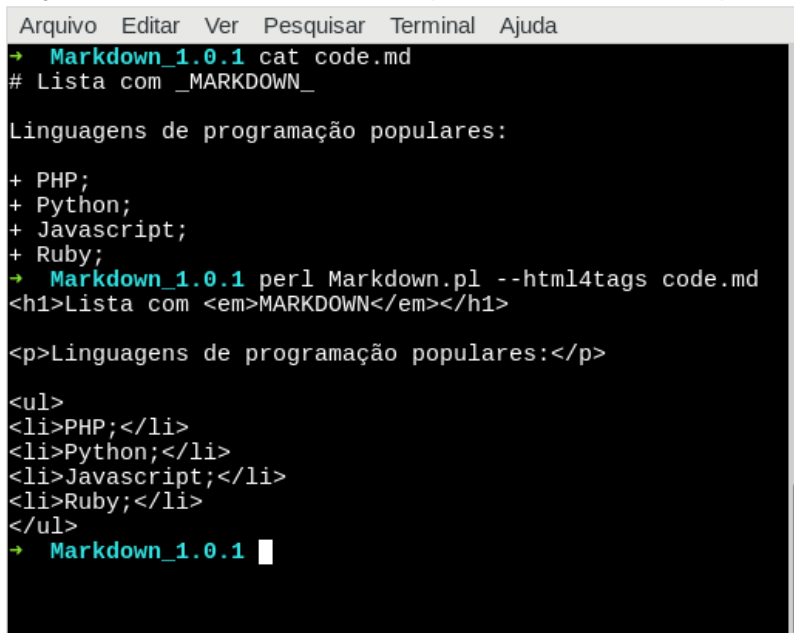
### 4.4.1 Parser criado por John Gruber

A implementação em feita por John Gruber, escrita com a linguagem de programação *Perl*, foi o primeiro *parser* para *Markdown* (MACFARLANE, GREENSPAN, *et al.*, 2017) e pode ser utilizado em conjunto com aplicativos para publicação de conteúdo web.

Tal *software* está disponível para *Download* na página do projeto e licenciado como *software* livre. Sua versão atual é a 1.0.1 e a última atualização no código ocorreu em 17 de dezembro de 2004 (GRUBER, 2017).

Este *parser* também pode ser utilizado para conversão de textos através de comandos na CLI de sistemas operacionais *GNU/Linux*, como no exemplo da Figura 34.

Figura 34 - Conversão de texto em *Markdown* para HTML utilizando *Markdown.pl* 1.0.1.



```

Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
→ Markdown_1.0.1 cat code.md
# Lista com _MARKDOWN_

Linguagens de programação populares:

+ PHP;
+ Python;
+ Javascript;
+ Ruby;
→ Markdown_1.0.1 perl Markdown.pl --html4tags code.md
<h1>Lista com <em>MARKDOWN</em></h1>

<p>Linguagens de programação populares:</p>

<ul>
<li>PHP;</li>
<li>Python;</li>
<li>Javascript;</li>
<li>Ruby;</li>
</ul>
→ Markdown_1.0.1

```

Fonte: Autoria própria.

Na imagem, antes da conversão, foi exibido o conteúdo do arquivo *code.md* utilizando o comando *cat* do SO *Debian GNU/Linux*, e em seguida, através de uma

chamada ao interpretador da linguagem *Perl*, foi executado o *script Markdown.pl*, passando como parâmetro de conversão *-html4tags*, que indica a versão da linguagem HTML usada na conversão e, por fim, o nome do arquivo que continha o texto a ser convertido, no caso *code.md*.

#### 4.4.2 Novos *parsers*

A sintaxe de formatação *Markdown* foi implementada em várias linguagens de programação, dando origem a outros *softwares* de conversão. Tais *softwares* são capazes de converterem textos escritos em *Markdown* para o formato HTML e, em alguns casos, convertem textos de *Markdown* para outros formatos além do HTML.

Assim como ocorre com o *parser* de John Gruber, os novos *parsers* são capazes de trabalharem de maneira solo, com suas interfaces próprias de entrada de texto e também podem ser utilizados em conjunto com outras aplicações, ou seja, como *plugins* (complementos que estendem as funcionalidades de um aplicativo).

O *software Pandoc*, por exemplo, desenvolvido com a linguagem de programação *Haskell*, permite realizar a conversão de texto escrito em diversos formatos de arquivo em outros diversos formatos arquivo. Este software possui uma implementação para conversão de textos *Markdown*, não apenas para HTML, mas também para outros formatos, tais como: *PDF*, *DOCX* e *TXT* (MACFARLANE, 2017).

A Figura 35 demonstra o procedimento para conversão de um arquivo em formato *Markdown* para o formato *DOCX* (formato de arquivo padrão do editor de textos *Microsoft Word*) utilizando o *Pandoc*.

**Figura 35** - Conversão de texto no formato *Markdown* para *DOCX* com o *software Pandoc*.



```

Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
→ conversoes-markdown git:(develop) x ls
teste-1.md
→ conversoes-markdown git:(develop) x cat teste-1.md
# Teste - 1

Testando a conversão 1 para 2:
1. Texto em Markdown.
2. Texto em Docx.
→ conversoes-markdown git:(develop) x pandoc teste-1.md -s -o teste-1.docx
→ conversoes-markdown git:(develop) x ls
teste-1.docx  teste-1.md
→ conversoes-markdown git:(develop) x █

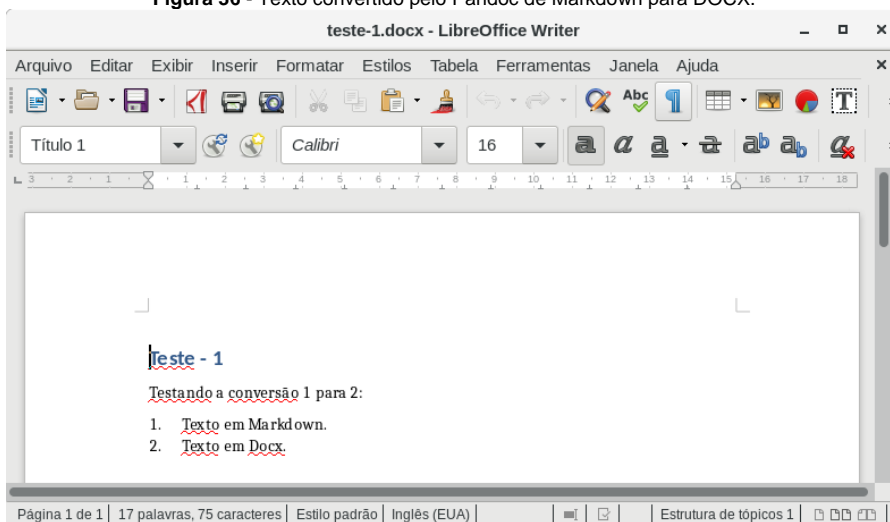
```

Fonte: Autoria própria.

Na demonstração, feita através do terminal do SO *Debian GNU/Linux*, inicialmente é listado o conteúdo do diretório corrente, inserido na CLI o comando *ls*; na sequência, o conteúdo do arquivo a ser convertido (*teste-1.md*) é impresso na tela com o uso do comando *cat*; então o comando para conversão, *pandoc*, é executado recebendo como parâmetros: o nome do arquivo original, a opção *-s* que indica a criação de um único arquivo e a opção *-o* que especifica o nome do arquivo de saída inserido no final da linha (*teste-1.docx*); por fim o conteúdo do diretório corrente é listado novamente, a fim de verificar a criação do arquivo com o texto convertido.

A Figura 36 exibe o conteúdo do arquivo convertido (*teste-1.docx*) no editor de textos *LibreOffice Writer*, que permite a leitura de arquivos de texto no formato DOCX.

Figura 36 - Texto convertido pelo Pandoc de Markdown para DOCX.



Fonte: Autoria própria.

Existe uma vasta lista de *parsers* e extensões para *Markdown*, sendo que, algumas das implementações notáveis são:

- **PHP Markdown e PHP Markdown Extra:** Uma biblioteca e uma extensão para a biblioteca criada por Michel Fortin utilizando a linguagem de programação *PHP*, que possibilitam a utilização de *Markdown* para postagem de conteúdo em sistemas *web* com a conversão sendo realizada em tempo de execução das aplicações (FORTIN, 2018). A biblioteca *PHP*

*Markdown Extra* amplia a quantidade de recursos sintáticos da especificação original da linguagem *Markdown*.

- **Python Markdown:** Uma biblioteca criada utilizando a linguagem de programação *Python*, que além da implementação original da sintaxe *Markdown*, possibilita, através de uma API - *Application Programing Interface* (em português, Interface de Programação de Aplicativos), a integração de diversas extensões que ampliam os recursos sintáticos especificados por John Gruber, (THE PYTHON-MARKDOWN PROJECT, 2017). Os primeiros autores do projeto são: Waylan Limberg, Yuri Takteyev, Manfred Stienstra, Artem Yunusov e David Wolever.
- **Kramdown:** Uma biblioteca escrita por Thomas Leitner com a linguagem de programação *Ruby*. Ela implementa a sintaxe original de *Markdown* e possibilita o uso de extensões que ampliam a quantidade de recursos (LEITNER, 2017).

#### 4.4.3 Recursos implementados pelos novos *parsers*

Retomando a questão da sintaxe, após o lançamento da ferramenta criada por John Gruber, os *parsers* que foram lançados em diversas linguagens de programação agregaram novas funcionalidades na linguagem *Markdown*, tais como: tabelas, notas de rodapé, listas de definição, diagramas UML e outros.

Não cabe neste TG, demonstrar cada uma dessas novidades, uma vez que já foi realizada uma introdução a sintaxe básica que abrange a maior parte das funcionalidades da linguagem *Markdown*. Todavia, será feita uma exceção para o recurso de tabelas, pois este recurso, que será demonstrado utilizando o *parser PHP Markdown Extra*, é amplamente utilizado nos documentos *Markdown*.

Assim, para que se obtenha uma tabela formatada com *Markdown*, é necessário envolver o conteúdo (dados) de cada célula da tabela por caracteres “|” (pipe). Além disso, é possível indicar as células que compõe o cabeçalho da tabela (título de cada coluna) e as células que compõe o corpo da tabela. Para isso, é preciso inserir uma linha especial na estrutura, especificamente a segunda linha da marcação, de modo que a primeira linha corresponderá ao título das colunas, e a partir da terceira linha serão inseridos os dados, corpo da tabela.

Na segunda linha, o conteúdo disposto entre os caracteres “|” (pipe), deve ser composto por:

- Um caractere “ ” (espaço);
- Na sequência, uma cadeia de caracteres “-” (hífen), não existindo limite determinado de caracteres que podem ser inseridos; e
- Por fim, após essa cadeia e antes do novo caractere “|” (pipe) é preciso inserir um caractere “ ” (espaço).

Deste modo, o texto de cada coluna seguirá o alinhamento padrão das tabelas, ou seja, alinhado à esquerda. É possível determinar o alinhamento que o texto de cada coluna seguirá inserindo um caractere “:” (dois pontos) em lugar dos caracteres “ ” (espaço) que antecedem e sucedem a cadeia de caracteres “-” (hífen). Assim, para obter alinhamento

- A esquerda, deve-se inserir o caractere “:” (dois pontos) antes da cadeia de caracteres “-” (hífen);
- A direita, o caractere “:” (dois pontos) deve ser inserido após a cadeia de caracteres “-” (hífen); e
- Centralizado, as duas extremidades da cadeia de caracteres “-” (hífen) devem conter, ao invés do caractere “ ” (espaço) o caractere “:” (dois pontos).

O Quadro 20 traz um exemplo da marcação feita no texto para obtenção de uma tabela, de modo que, ao visualizar o texto em *Markdown* também é possível prever que a estrutura se trata de uma tabela.

**Quadro 20 - Markdown:** Sintaxe de formatação - Tabelas.

Texto em Markdown	Texto convertido para HTML
<pre># Linguagens de programação  ID* *Linguagem   Site*   -- ----- ----    1   PHP   http://www.php.net     2   Python   https://www.python.org     3   Ruby   https://www.ruby-lang.org  </pre>	<pre>&lt;h1&gt;Linguagens de programação&lt;/h1&gt; &lt;table&gt; &lt;thead&gt; &lt;tr&gt;   &lt;th align="center"&gt;ID&lt;/th&gt;   &lt;th align="left"&gt;Linguagem&lt;/th&gt;   &lt;th align="right"&gt;Site&lt;/th&gt; &lt;/tr&gt; &lt;/thead&gt; &lt;tbody&gt; &lt;tr&gt;</pre>



	<pre> &lt;td align="center"&gt;1&lt;/td&gt; &lt;td align="left"&gt;PHP&lt;/td&gt; &lt;td align="right"&gt;http://www.php.net&lt;/td&gt; &lt;/tr&gt; &lt;tr&gt; &lt;td align="center"&gt;2&lt;/td&gt; &lt;td align="left"&gt;Python&lt;/td&gt; &lt;td align="right"&gt;https://www.python.org&lt;/td&gt; &lt;/tr&gt; &lt;tr&gt; &lt;td align="center"&gt;4&lt;/td&gt; &lt;td align="left"&gt;Ruby&lt;/td&gt; &lt;td align="right"&gt;https://www.ruby-lang.org&lt;/td&gt; &lt;/tr&gt; &lt;/tbody&gt; &lt;/table&gt; </pre>
--	---

Fonte: Autoria própria.

A Figura 37 - *Markdown*: Saída HTML renderizada - Tabela. exibe o resultado renderizado a partir da formatação demonstrada no Quadro 20, sendo que, a conversão para HTML e renderização da marcação HTML, foram realizadas utilizando a aplicação *online Babelmark 2*.

Figura 37 - *Markdown*: Saída HTML renderizada - Tabela.

Linguagens de programação	
ID Linguagem	Site
1 PHP	http://www.php.net
2 Python	https://www.python.org
3 Ruby	https://www.ruby-lang.org

Fonte: Autoria própria.

#### 4.5 PADRONIZAÇÃO DAS ESPECIFICAÇÕES

O tipo de mídia *text/markdown* é registrado pela *RFC-7763*, documento cuja sigla possui o seguinte significado em inglês: *Request For Comments number 7763* (Em português uma tradução condizente é: Requisição de Mudanças número 7763).

Este tipo de documento define um padrão para uso de determinada tecnologia e é emitido pela *Internet Engineering Task Force (IETF)*, um órgão constituído por uma comunidade internacional que trabalha em função da elaboração de padrões abertos para a internet, que são obtidos através de procedimentos que também são abertos, (INTERNET ENGINEERING TASK FORCE (IETF), 2018).

A *RFC-7763* define o referido tipo de mídia para uso com uma família de sintaxe (a linguagem *Markdown* em sua sintaxe original e suas derivações) para “formatação de texto simples, que opcionalmente pode ser convertido em linguagens de marcação formal, como HTML” e é complementada pela *RFC-7764* que traz orientações acerca da utilização do tipo de mídia *text/Markdown* (LEONARD, 2016, p. 1).

A *RFC-7764* diz que:

“*Markdown* é, especificamente, uma família de sintaxes que se baseiam no trabalho original de John Gruber com contribuições substanciais de Aaron Swartz, lançado em 2004. Desde a sua liberação, uma série de aplicações *web* ou voltadas para a *web* incorporaram o *Markdown* em seus sistemas de entrada de texto, frequentemente com extensões personalizadas” (LEONARD, 2016, p. 4).

A constatação exposta na *RFC-7764* é de fácil verificação, no entanto a sintaxe original da linguagem não abrange todos os elementos disponíveis na linguagem HTML e com o passar do tempo e a constante adoção de *Markdown*, surgiram diversos *softwares* para conversão implementados em várias linguagens de programação, conforme citado na sessão anterior.

Dessa forma, a implementação dada por um desenvolvedor, ou por um grupo de desenvolvedores dentro de um projeto, por vezes destoa das demais implementações, mantendo a compatibilidade apenas com a sintaxe original. Assim, em uma comparação rasa, o usuário que submeter seu texto formatado em *Markdown* ao *parser Python Markdown*, poderá receber um resultado que não seria igual ao produzido se viesse a submeter o mesmo texto ao *parser Kramdown*.

Por essa razão o professor e entusiasta da linguagem *Markdown*, John MacFarlane, publicou a aplicação *web* chamada *Babelmark 2*, utilizada na sessão 4.4.3 no processo de demonstração do uso de *Markdown* para criação de tabelas. A aplicação funciona *online* e permite ao usuário verificar como ficaria a conversão de um texto formatado com *Markdown* após a ação de vários *parsers*.

A aplicação de MacFarlane devolve como resultado de sua execução um quadro, contendo o código HTML gerado por cada um dos conversores, ou a renderização obtida no navegador para o código retornado (MACFARLANE, 2012).

Além dessa aplicação, MacFarlane encabeça um grupo de pessoas usuárias da linguagem que propõem “uma especificação de sintaxe padrão e inequívoca para o *Markdown*, juntamente com um conjunto de testes abrangentes para validar as implementações do *Markdown* contra esta especificação”. A especificação proposta

**Comentado [GVTD522]:** Talvez de pra tirar esse parágrafo.

foi intitulada *CommonMark Spec*, (MACFARLANE, GREENSPAN, *et al.*, 2017) e atualmente está na versão 0.28, que foi lançada no dia primeiro de agosto de 2017.

Tal especificação foi adotada pelo *GitHub*, (site para armazenamento e compartilhamento de código que faz uso intenso da sintaxe *Markdown*), porém foi chamada de GFM - *GitHub Flavored Markdown* (em português, Markdown com sabor GitHub).

#### 4.6 APLICAÇÃO DA LINGUAGEM MARKDOWN

Segundo (GRUBER, 2017) “A sintaxe do *Markdown* destina-se a um propósito: ser usada como um formato para escrever para a web”. Nesse sentido, *Markdown* foi bem recebido por usuários técnicos, desenvolvedores, e por usuários comuns de sistemas diversos tais como: *blogs*, fóruns, *wiki*, gerenciadores de tarefa e outros.

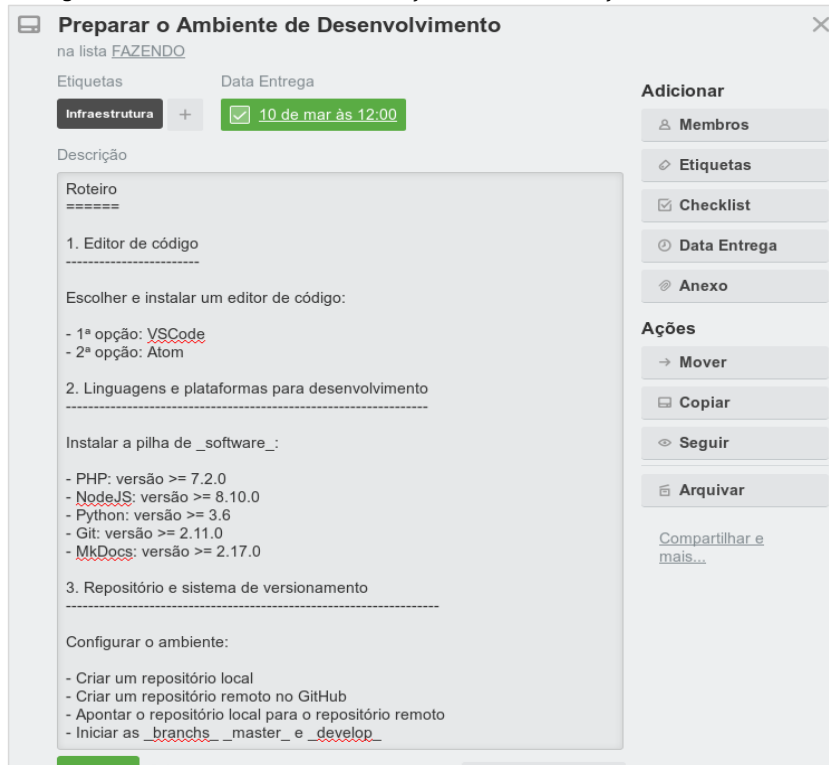
##### 4.6.1 Aplicações de uso geral

Um exemplo da aplicação de *Markdown* são os cartões do aplicativo de gestão de projetos e tarefas *Trello*. Nesse aplicativo, que adota a dinâmica da metodologia *Kanban*, tem-se os elementos chamados cartões que representam as tarefas e são organizados em listas dentro de um quadro, sendo que as listas representam o estado de cada tarefa e o quadro representa o projeto.

Os cartões do *Trello* dispõem de diversos elementos para inserção de dados sobre a tarefa e entre eles há o campo de descrição textual. Este campo em específico permite a inserção de texto formatado com *Markdown*, proporcionando a elaboração de descrições ricas em detalhes.

A Figura 38 traz um exemplo do uso de *Markdown* em cartões do *Trello*.

**Figura 38** - Sintaxe *Markdown* na formatação do texto de descrição da tarefa no *Trello*.



Fonte: Autoria própria.

O resultado da formatação com *Markdown*, mostrada no modo de edição na Figura 38, pode ser visualizado na Figura 39, que exibe o cartão no modo de visualização.

**Figura 39** - Descrição da tarefa feita com *Markdown* renderizada em HTML no cartão do *Trello*.

**Preparar o Ambiente de Desenvolvimento**

na lista [FAZENDO](#)

Etiquetas: **Infraestrutura** + ☒ 10 de mar às 12:00

Data Entrega: ☒ 10 de mar às 12:00

Descrição

[Editar](#)

### Roteiro

#### 1. Editor de código

Escolher e instalar um editor de código:

- 1ª opção: VSCode
- 2ª opção: Atom

#### 2. Linguagens e plataformas para desenvolvimento

Instalar a pilha de *software*:

- PHP: versão >= 7.2.0
- NodeJS: versão >= 8.10.0
- Python: versão >= 3.6
- Git: versão >= 2.11.0
- MkDocs: versão >= 2.17.0

#### 3. Repositório e sistema de versionamento

Configurar o ambiente:

- Criar um repositório local
- Criar um repositório remoto no GitHub
- Apontar o repositório local para o repositório remoto
- Iniciar as *branches master* e *develop*

**Adicionar**

- ☐ Membros
- ☐ Etiquetas
- ☒ Checklist
- ☐ Data Entrega
- ☐ Anexo

**Ações**

- Mover
- 📄 Copiar
- 👁 Seguir
- 📁 Arquivar

[Compartilhar e mais...](#)

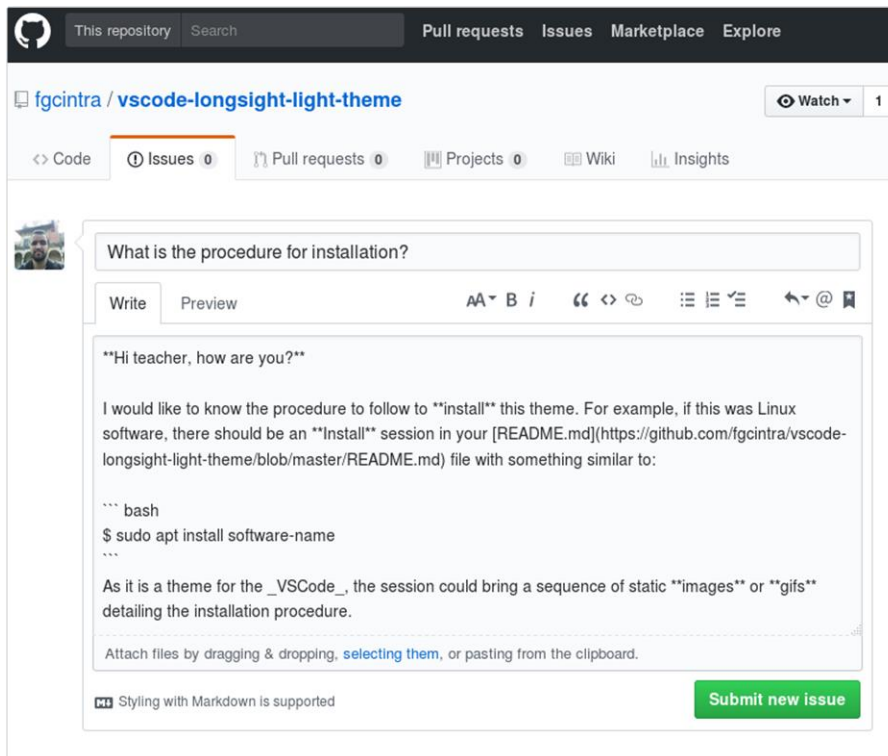
**Fonte:** Autoria própria.

Outro exemplo no qual pode-se aplicar *Markdown* são as interações no *GitHub*, que são feitas com base nas especificações *Markdown* GFM. O uso da linguagem *Markdown* no *GitHub* é uma prática comum e encorajada, pois propicia a publicação de documentos formatados com detalhes que facilitam o entendimento dos projetos, de eventuais dúvidas e de procedimentos que devem ser adotados, por exemplo, para instalação e utilização do *software*.

No processo de abertura de *issues* (diálogos para reportagem de problemas no código ou esclarecimento de dúvidas em um projeto), a sintaxe *Markdown* colabora substancialmente para formulação do questionamento, pois, conforme é mostrado na

na Figura 40, é possível destacar palavras no texto, criar um *link* para um arquivo do projeto, citar trechos de código, entre outras facilidades.

**Figura 40** - Abertura de *issue* no GitHub utilizando *Markdown* para formatar o texto.

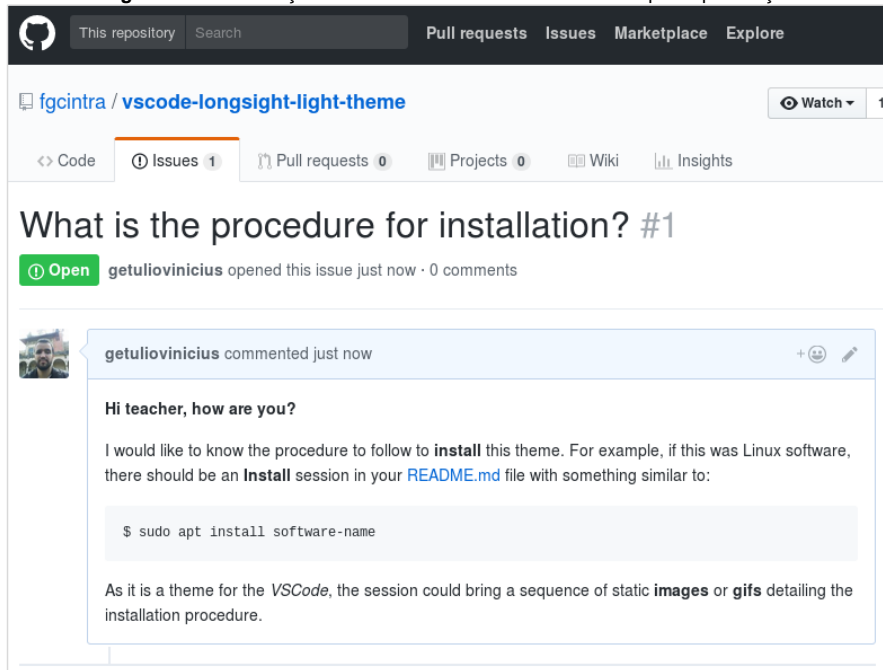


**Fonte:** Autoria própria.

A visualização da *issue*, cuja demonstração da abertura foi realizada na Figura 40, é mostrada, após a sua publicação, na Figura 41.

Nota-se na imagem os destaques feitos no texto (ênfase em negrito e ênfase em itálico); o link ancorado sobre a palavra “*README.md*”; e o destaque do comando utilizado como exemplo, apresentado com fonte mono - espaçada e o fundo do texto na cor cinza.

Figura 41 - Visualização da *issue* formatada com *Markdown* após a publicação.



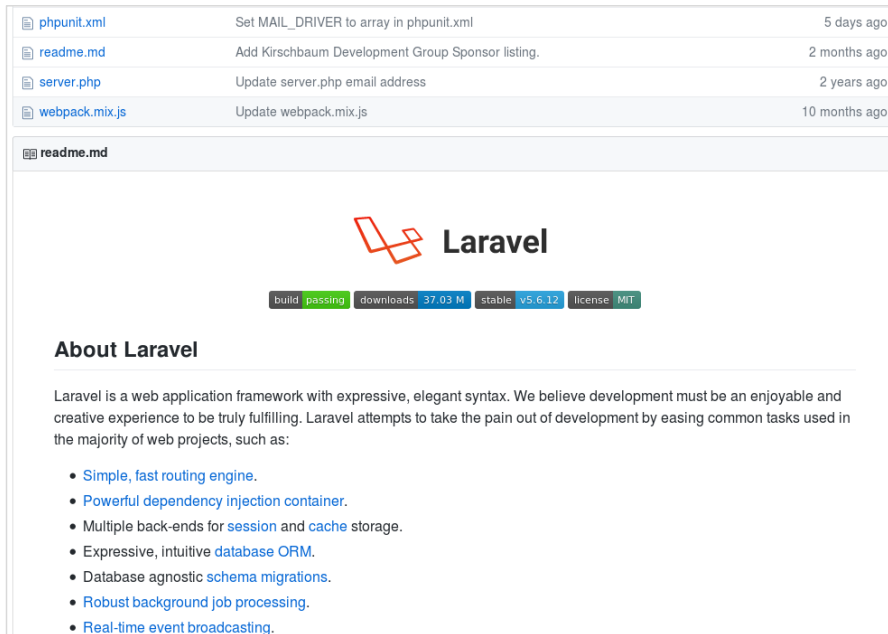
Fonte: Autoria própria.

Além das *issues*, o *GitHub* permite a criação de um arquivo de texto formatado com *Markdown*, que em inglês é chamado de *README.md* (em português, “Leia-me”). Esse arquivo é colocado no diretório raiz de um projeto de *software* e contém uma descrição geral do repositório e as instruções de uso, instalação, colaboração e outras informações relevantes do projeto.

Tal arquivo, é automaticamente processado pelo *GitHub*, que exibe o conteúdo de forma renderizada após a listagem de arquivos e diretórios do projeto na página inicial de um repositório.

A Figura 42 traz um recorte de tela, onde observa-se uma demonstração dessa funcionalidade do *GitHub*. A imagem exibe um trecho da página inicial de um repositório armazenado no *GitHub*, especificamente o repositório do *framework* para construção de aplicações *web* na linguagem de programação *PHP* chamado *Laravel*.

**Figura 42** - Recorte de tela da página inicial do repositório Laravel armazenado no Github.



**Fonte:** Autoria própria.

A aplicação da linguagem *Markdown* para construir os arquivos de instruções dos *softwares* disponibilizados em repositórios do *GitHub* e em outros sites de armazenamento e compartilhamento de código, como o *Gitlab* e o *Bitbucket*, talvez seja o primeiro contato de muitos desenvolvedores com a tecnologia, e o grande caso de sucesso da linguagem *Markdown*.

A partir da simples ideia de se promover a criação de arquivos “*README.md*”, para instruções gerais acerca de um repositório de software, criou-se uma cultura de utilização da linguagem *Markdown* expandida para outras áreas dos sites, como por exemplo a área de *issues*, já citada, e a área de *wiki* (um local para publicação de arquivos como mais detalhes do *software*).

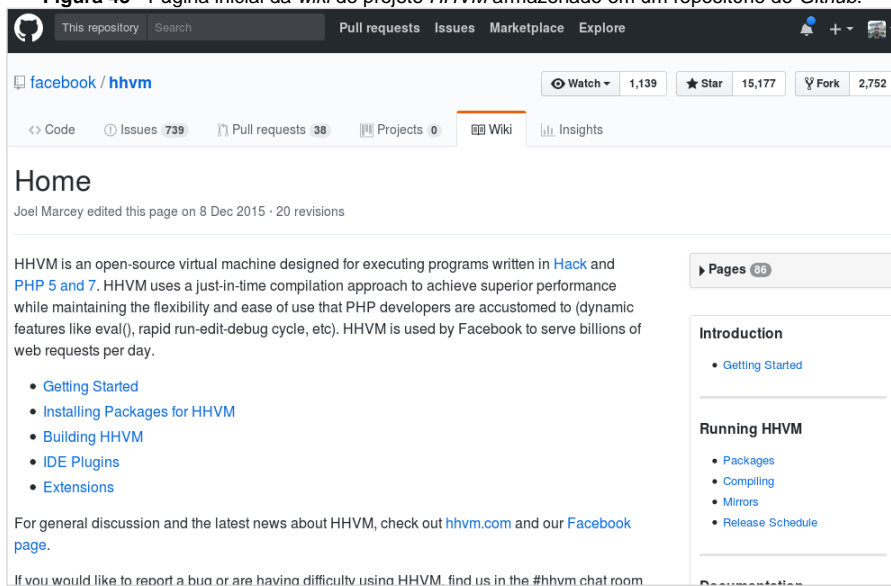
As páginas das áreas de *wiki* dos três sites de armazenamento de repositórios citados, *GitHub*, *Gitlab* e *Bitbucket* são escritas utilizando sintaxe de formatação *Markdown*, de modo que se tornaram em alguns casos a página de documentação oficial do *software*.

A Figura 43 exibe um trecho da página inicial da *wiki*, escrita com formatação de texto *Markdown*, para o *software* de código aberto (*open source*) *Hip Hop Virtual*



*Machine (HHVM)*. A *wiki*, disponível no *Github*, traz uma vasta documentação para o *HHVM*, que foi desenvolvido pela empresa *Facebook* e consiste em uma aplicação que transforma o código escrito na linguagem *PHP* para as linguagens *C* e *C++*.

**Figura 43** - Página inicial da *wiki* do projeto *HHVM* armazenado em um repositório do *Github*.



**Fonte:** Autoria própria.

#### 4.6.2 Geradores de sites estáticos

Por fim, após toda a explicação e demonstrações do uso de *Markdown* já realizada, resta apenas falar sobre a aplicação em sistemas geradores de sites estáticos.

Pois bem, um site estático se caracteriza pelo fato de que o usuário não insere dados ou interage com a interface de modo a provocar reações maiores do que uma simples troca de páginas, e, no limite, uma pesquisa por conteúdo.

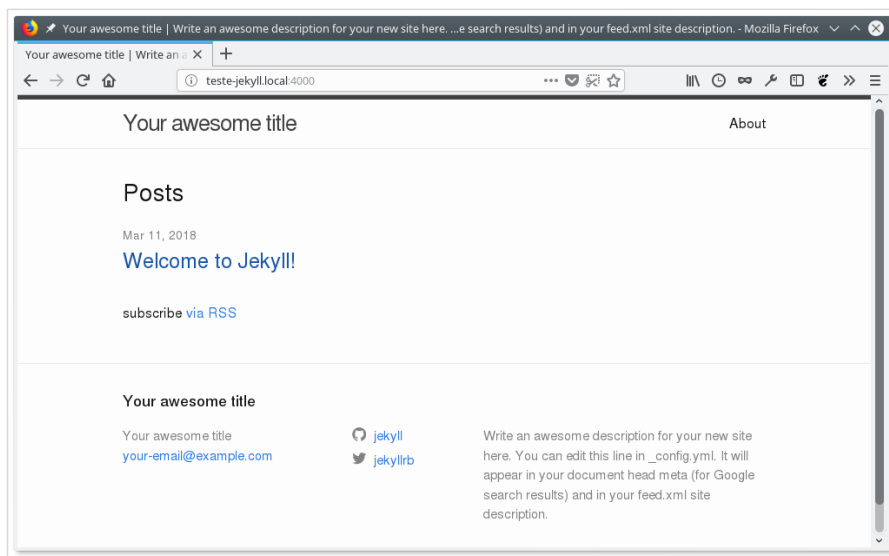
Assim, existem aplicações capazes de converter arquivos escritos em linguagem *Markdown* para sites estáticos compostos por arquivos HTML e folhas de estilo CSS - *Cascading Style Sheet* (em português, Folha de Estilo em Cascata), com a possibilidade de utilização de *templates*, que neste caso, representam modelos feitos com HTML e CSS prontos para serem servidos na *web*.

O site *StaticGen* ([www.staticgen.com](http://www.staticgen.com)) traz uma listagem com dezenas de *softwares* para este propósito, que estão armazenados em repositórios do *GitHub*, sendo que, segundo as estatísticas apuradas pelo próprio *GitHub*, o software *Jekyll* é o projeto de gerador de site estáticos que recebeu maior quantidade de estrelas, que é um método de avaliação da popularidade dos repositórios (STATICGEN, 2018).

O famoso *Jekyll*, é desenvolvido com a linguagem de programação *Ruby* e se utiliza do *parser Kramdown*, que também é escrito em *Ruby*, para converter os arquivos *Markdown* em HTML. O *Jekyll* é o motor por trás da renderização das páginas no *Github Pages*, um serviço oferecido pelo *GitHub* para servir sites estáticos que apresentam os softwares armazenados em repositórios do próprio *GitHub* (JEKYLL, 2018).

A Figura 44 exibe uma amostra da visualização inicial de um site estático pré-configurado por padrão na instalação do *Jekyll*.

**Figura 44** - Visualização inicial de um site estático com a *template* padrão do *Jekyll*.



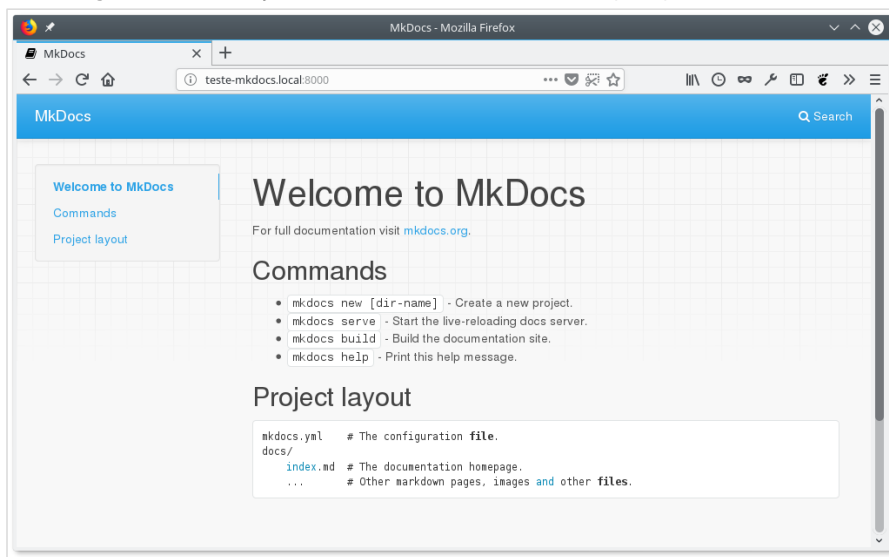
**Fonte:** Autoria própria.

Outro software gerador de sites estáticos muito utilizado é o *MkDocs*, escrito com a linguagem de programação *Python*. O *MkDocs* utiliza o *parser Python Markdown* para converter os arquivos de *Markdown* para HTML e, entre suas facetas, está a funcionalidades de envio do site estático, criado em um repositório local (no

computador do usuário), para o serviço *GitHub Pages* (CHRISTIE, 2014). Com o funcionamento similar ao do *Jekyll*.

A Figura 45 exibe uma amostra da visualização inicial de um site estático pré-configurado por padrão na instalação do *MkDocs*.

**Figura 45** - Visualização inicial de um site estático com a *template* padrão do *MkDocs*.

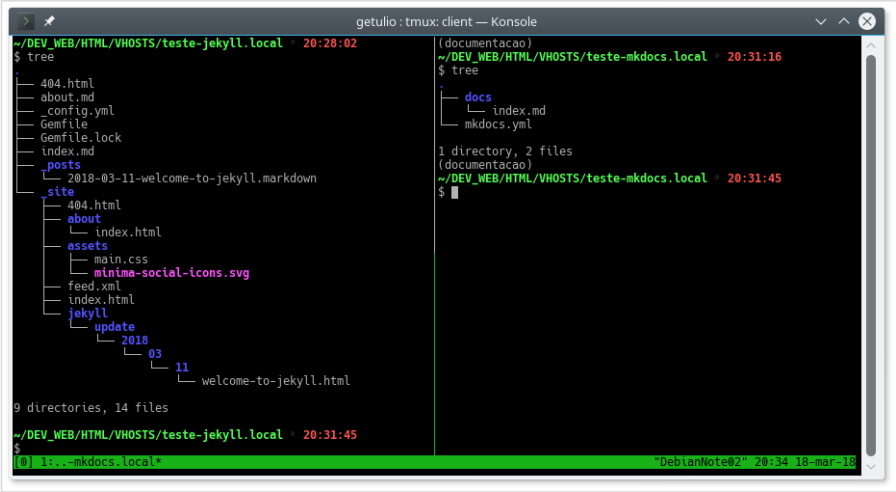


Fonte: Autoria própria.

Comparando o *MkDocs* com o *Jekyll*, nota-se que o *Jekyll* possui mais recursos, permitindo até a criação de estruturas complexas de postagens tais como os *blogs*, entretanto, o processo de configuração e o tempo de aprendizado da ferramenta podem ser considerados desvantagens do *Jekyll* frente ao *MkDocs*.

Apesar da documentação do *MkDocs* citar a possibilidade de ajustes e personalizações no estilo visual e na disposição dos elementos do site estático, ou seja, no *template*, tal qual ocorre com a documentação do *Jekyll*, nota-se que existe maior complexidade no uso do *Jekyll*. A própria instalação das duas aplicações, após concluídas, levam o usuário a pressupor que o *MkDocs* possui um ambiente de trabalho menos complexo do que o *Jekyll*, o que pode ser observado na Figura 46, onde, do lado direito da imagem tem-se a estrutura de diretórios de uma instalação nova do *Jekyll*, e do lado esquerdo a estrutura do *MkDocs*.

Figura 46 - Comparação entre a estrutura de um projeto *Jekyll* e um projeto *MkDocs*.



Fonte: Autoria própria.

4.6.2.1 MkDocs

Por considerar o *MkDocs* um gerador de sites estáticos de mais fácil utilização, além do fato dele atender as necessidades de documentação do projeto que será apresentado no capítulo 5, esta foi a escolha de gerenciador de sites estáticos para demonstração e uso na sequência deste TG.

O processo de instalação do *MkDocs* em um SO *Debian GNU/Linux* será descrito a seguir no Quadro 21.

Quadro 21 - Processo de instalação do *MkDocs* no SO *Debian GNU/Linux*.

Instalação do <i>MkDocs</i>
<div>1. Fazer o <i>download</i> do instalador do gerenciador de pacotes "<i>pip</i>". \$ <i>wget https://bootstrap.pypa.io/get-pip.py</i></div> <div>2. Instalar o gerenciador de pacotes "<i>pip</i>". \$ <i>sudo python get-pip.py</i></div> <div>3. Instalar o pacote "<i>virtualenv</i>". \$ <i>sudo pip install virtualenv</i></div> <div>4. Instalar o pacote "<i>virtualenvwrapper</i>". \$ <i>sudo pip install virtualenvwrapper</i></div> <div>5. Editar o arquivo "<i>.bashrc</i>" na localizado no diretório "<i>home</i>" do usuário atual e adicionando as linhas abaixo no final do arquivo.</div>

```
# Virtualenvwrapper
export WORKON_HOME=~/.virtualenvs
source /usr/local/bin/virtualenvwrapper.sh
export PIP_REQUIRE_VIRTUALENV=true
```

6. Recarregar as configurações do "bash".  
\$ source ~/.bashrc
7. Criar um diretório para armazenar os arquivos do projeto com "MkDocs".  
\$ mkdir -p ~/teste-mkdocs
8. Criar um ambiente virtual Python para execução do "MkDocs".  
\$ mkvirtualenv docs
9. Ativar o uso do ambiente virtual Python.  
\$ workon docs\_mkdocs
10. Instalar o "MkDocs".  
\$ pip install mkdocs
11. Iniciar um projeto com "MkDocs".  
\$ mkdocs new .

Os itens numerados representam a descrição das ações que devem ser realizadas (comandos), e, as linhas iniciadas com o caractere "\$" (cifrão) representam os comandos que devem ser inseridos, sem o "\$", que apenas indica um ponto de inserção de comandos na CLI em um terminal do SO *Debian GNU/Linux*.

**Fonte:** Autoria própria.

Com o *MkDocs* instalado, toda a configuração para geração do site estático é realizada inserindo parâmetros no arquivo "*mkdocs.yml*", que a princípio traz somente o parâmetro "*site\_name*" configurado. O arquivo "*mkdocs.yml*" é salvo no diretório raiz da instalação do projeto de site estático, conforme pôde ser verificado na Figura 46, e, trata-se de um arquivo do tipo YAML uma acrônimo que em inglês possui o seguinte significado: *YAML Ain't Markup Language* (em português, YAML não é Linguagem de Marcação).

Um exemplo do arquivo "*mkdocs.yml*", com diversos parâmetros configurados, pode ser visualizado no Quadro 22, onde os parâmetros estão separados por categorias tais como: Dados do Projeto, Dados do Repositório, Conta do *Google Analytics* (para análise de visitas ao site), Extensões do *parser Python Markdown* (que serão ativadas), Tema, Mapa do site e Recursos Extras. Sendo que, tal categorização dos parâmetros não é obrigatória, tendo sido feita apenas para facilitar o entendimento.

**Quadro 22** - Arquivo de parâmetros para geração de sites estáticos com *MkDocs*.

Arquivo <i>mkdocs.yml</i>	
1.	# Dados do Projeto
2.	site_name: tgGV
3.	site_description: Documentação do Projeto de Software tgGV
4.	site_author: Getúlio Vinicius <getuliovinits@gmail.com>
5.	site_url: https://getuliovinicius.github.io/trabalho.graduacao/
6.	
7.	# Dados do Repositório
8.	repo_url: https://github.com/getuliovinicius/trabalho.graduacao
9.	repo_name: GitHub
10.	edit_uri: blob/master/docs/mds/
11.	remote_branch: gh-pages
12.	remote_name: origin
13.	
14.	# Conta do Google Analytics
15.	google_analytics: ['UA-YXYXYXYX-YX', 'auto']
16.	
17.	# Extensões do Python Markdown
18.	markdown_extensions:
19.	- smarty
20.	- nl2br
21.	- admonition
22.	- codehilite
23.	- footnotes
24.	- toc:
25.	permalink: True
26.	separator: " _ "
27.	
28.	# Tema
29.	theme:
30.	name: readthedocs
31.	
32.	# Mapa do Site
33.	docs_dir: 'mds'
34.	pages:
35.	- Início: 'index.md'
36.	- Desenvolvimento:
37.	- Descrição: 'doc-desenvolvimento/0-descricao.md'
38.	- Requisitos: 'doc-desenvolvimento/1-requisitos.md'
39.	- Modelagem: 'doc-desenvolvimento/2-modelagem.md'
40.	- Prototipação: 'doc-desenvolvimento/3-prototipacao.md'
41.	- Instalação:
42.	- Introdução: 'doc-instalacao/00-introducao.md'
43.	- Guia do usuário:
44.	- Introdução: 'doc-usuario-final/00-introducao.md'
45.	- Licença: 'licence.md'
46.	
47.	# Recursos Extra
48.	extra:
49.	version: 0.3.0

Os números a esquerda representam a contagem de linhas.

**Fonte:** Autoria própria.

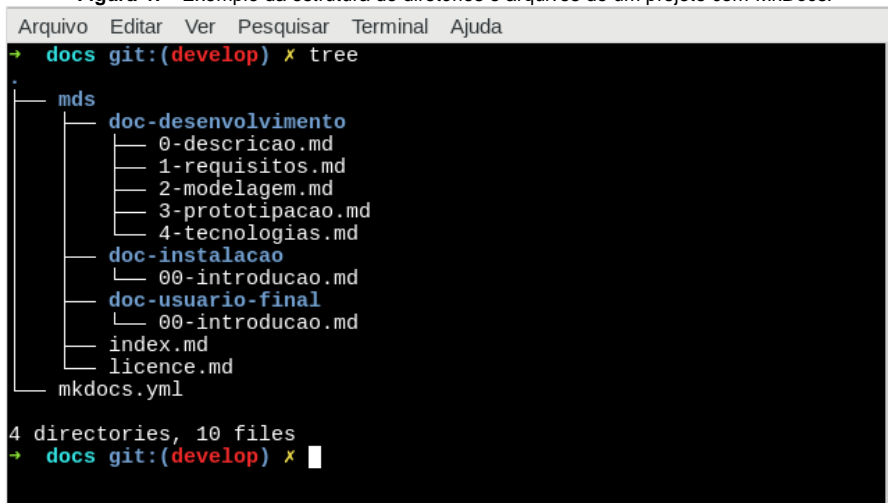
No arquivo de configurações exposto no Quadro 22, entre outras informações de destaque, estão os parâmetros para indicação do uso de extensões do *parser Python Markdown*, como por exemplo as extensões *codehilite*, que exibe trechos de códigos com sintaxe colorida, e a extensão *toc - table of contents* (em português, tabela de conteúdo), que cria índices e links para sessões de um texto.

Nota-se também, a definição de um tema visual para o site estático, e por fim, outra informação que merece destaque no arquivo “*mkdocs.yml*”, está contida na sessão Mapa do Site, que descreve a estrutura de navegação do site, onde cada item da navegação recebe como parâmetro o caminho para o arquivo *Markdown* relacionado.

O primeiro parâmetro da sessão Mapa do Site, define o subdiretório do projeto onde serão armazenados os arquivos *Markdown*, que possuem a extensão “.md”.

A estrutura de diretórios e arquivos definida no exemplo constante no Quadro 22 pode ser observada na Figura 47.

**Figura 47** - Exemplo da estrutura de diretórios e arquivos de um projeto com *MkDocs*.



```

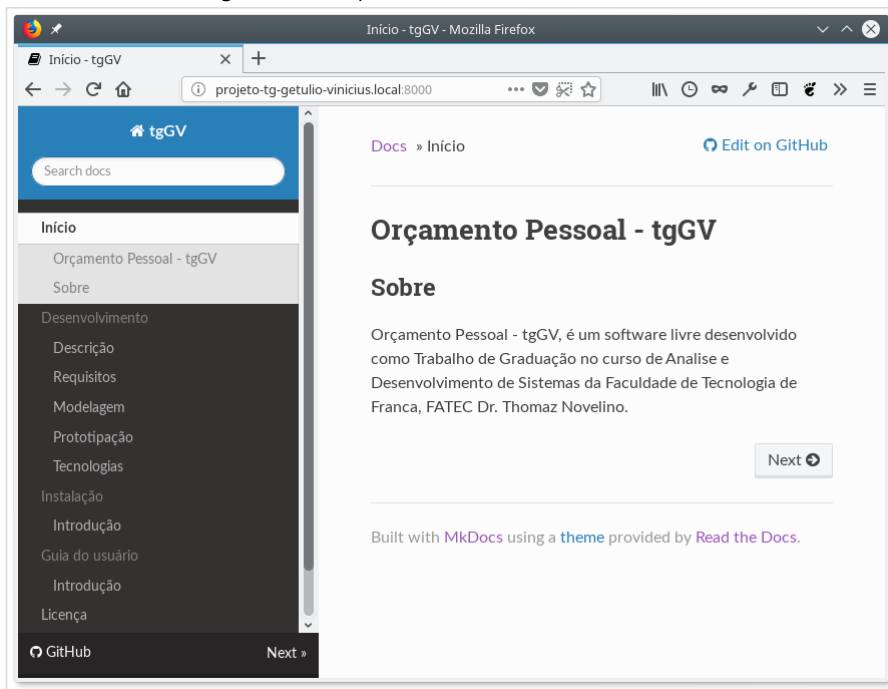
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
→ docs git:(develop) x tree
.
├── mds
│   ├── doc-desenvolvimento
│   │   ├── 0-descricao.md
│   │   ├── 1-requisitos.md
│   │   ├── 2-modelagem.md
│   │   ├── 3-prototipacao.md
│   │   └── 4-tecnologias.md
│   ├── doc-instalacao
│   │   └── 00-introducao.md
│   ├── doc-usuario-final
│   │   └── 00-introducao.md
│   ├── index.md
│   └── licence.md
└── mkdocs.yml
4 directories, 10 files
→ docs git:(develop) x

```

Fonte: Autoria própria.

Assim, concluindo as demonstrações acerca do *software MkDocs*, a **Figura 48** traz a visualização do site estático cuja configuração foi demonstrada no Quadro 22 e a estrutura de diretórios exibida na Figura 47.

**Figura 48** - Exemplo de site estático criado com *MkDocs*.



**Fonte:** Autoria própria.

Portanto, a partir da análise da Figura 48, pode-se considerar que o *MkDocs* é capaz de produzir sites estáticos com interface útil, navegável e visualmente atraente. Em outras palavras, vem de encontro a necessidade de agrupar o conteúdo documental produzido durante a execução de um projeto de *software* em uma plataforma centralizada, que possibilita a apresentação para o público interessado de forma simples, familiar, atraente, acessível e pesquisável, entre outras características.

Portanto, no próximo capítulo deste TG será apresentado um relatório acerca do desenvolvimento de uma aplicação para orçamento pessoal, no qual foi utilizada a linguagem *Markdown* e o gerador de sites estáticos *MkDocs* para apresentar o conteúdo documental.



## 5 APLICAÇÃO PRÁTICA - PROJETO

Como caso de uso para o estudo elaborado sobre documentação de *software*, foi desenvolvido um projeto de aplicação para finanças e orçamento pessoal.

A seguir serão relatados os processos de Concepção, Planejamento, Execução e **Fechamento** do projeto.

### 5.1 CONCEPÇÃO

O projeto foi concebido a partir da constatação de uma necessidade comum a muitas pessoas, que é a gestão das finanças pessoais.

Existem aplicativos que oferecem esse tipo de serviço e estão disponíveis para uso na *web*, em computadores, e para uso em dispositivos móveis dos tipos *smartphone* e *tablet*. Alguns deles são: Guia Bolso, Minhas Economias, *Mobilis* e *Tolsh* Finanças. Há muitos outros, que não necessitam serem mencionados.

Dos aplicativos que foram citados, alguns são disponibilizados para uso de forma gratuita e outros possuem recursos gratuitos e recursos pagos. Também existem aplicativos que possuem apenas recursos pagos, mas que não foram mencionados.

Além do uso de aplicativos, o controle de finanças pessoais pode ser realizado por meio de planilhas eletrônicas, que são aplicativos relativamente acessíveis as pessoas.

Apesar de existir uma gama de possibilidades para solução do problema apresentado, o controle de finanças pessoais, existem características do universo contábil como as partidas dobradas, por exemplo, que não são adotadas nos aplicativos citados, ou, ao menos não estão implementadas de uma forma perceptível.

A princípio esse ponto pareceu um fator de inovação, ou seja, tratar de finanças pessoais com enfoque em princípios de contabilidade pouco adotado nos aplicativos disponíveis e até certo ponto complexo para se manter em planilhas. Assim, partindo de uma análise dos aplicativos que já existiam foi decidido que seria desenvolvido a referida aplicação, e que ela receberia o nome de **"tgGV"**.

**Comentado [GVTDS23]:** Dar um nome para a aplicação.

#### 5.1.1 Arquitetura da aplicação

Trata-se de uma aplicação baseada em dois componentes principais:

- Uma API REST, e

- Uma aplicação cliente *Web*.

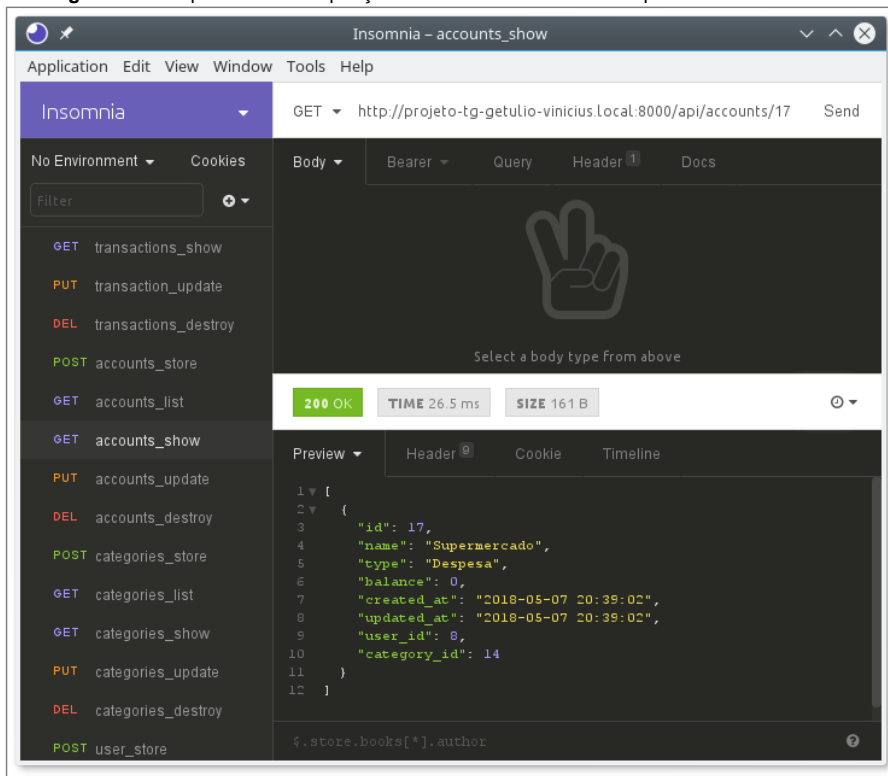
Essa abordagem foi escolhida pois propicia a escala da aplicação de modo sustentável, ou seja, tendo uma API REST como base, pode-se desenvolver aplicações para inúmeras plataformas capazes de consumir os recursos da API, tais como: aplicação *Web*, aplicação *Desktop* – para diversos sistemas operacionais – e aplicativo *mobile* – para *smartphones*.

Deste modo também é possível manter as partes da aplicação desacopladas e possivelmente integrar o código com outros sistemas mais facilmente, pois em se tratando de REST, que é um acrônimo para *Representational State Transfer* (em português, Transferência de Estado Representativo), todos os dados são considerados recursos, que estão disponíveis a partir de uma URI – *Universal Resource Identifier* (em português, Identificador de Recurso Universal).

Segundo (SAUDADE, 2014, p. 13), “REST é idealmente concebido para uso com o protocolo HTTP” – *Hypertext Transfer Protocol* (em português, Protocolo de Transferência de Hipertexto), e está é justamente a utilização que será feita da API REST ao ser consumida pela aplicação.

Assim, através do protocolo HTTP a aplicação web requisita os dados a uma URI da API, por exemplo: <http://dominio-da-api.com.br/api/accounts/1> e a API retorna os dados em formato de arquivo JSON – *Javascript Object Notation* (em português, Notação de Objetos Javascript). Este cenário pode ser observado na Figura 49 que traz a requisição realizada em uma ferramenta de testes para APIs chamada *Insomnia*, onde os dados de uma conta são retornados com sucesso.

**Figura 49** - Resposta a uma requisição feita com verbo HTTP GET para um recurso da API.



Fonte: Autoria própria.

Na Figura 49 é possível observar que a resposta da API veio acompanhada do código de *status* HTTP 200, que indica para aplicação web que houve sucesso na requisição. Caso algum erro tivesse ocorrido durante o processamento da requisição HTTP, a API retornaria um código de *status* diferente, indicando que não houve sucesso e que os dados não puderam ser retornados. Dessa forma, cabe a aplicação tratar o código de status HTTP devolvido pela API e exibir a informação adequada para o usuário.

Um exemplo de código de *status* HTTP que representa um erro é o 404, retornado quando o recurso não é encontrado.

Além dos códigos de *status*, o protocolo HTTP possui uma outra característica quanto as requisições que são os verbos. Assim, um verbo é utilizado com o objetivo de executar uma ação específica, por exemplo: usa-se o verbo *GET* quando necessita-se recuperar um determinado dado ou uma listagem de dados; usa-se o

verbo *POST* quando o objetivo é armazenar um ou mais dados, o que ocorre geralmente a partir do preenchimento de um formulário; usa-se o verbo *PUT* quando se deseja atualizar uma determinada informação; e completando a lista dos verbos mais utilizados existe o *DELETE*, cujo objetivo é requisitar a remoção de dados. Outros dois verbos HTTP são o *PATCH* e o *HEAD*.

Essa, portanto, foi a concepção do projeto de desenvolvimento da aplicação acompanhada por uma explanação em linhas gerais acerca da abordagem tecnológica.

## 5.2 PLANEJAMENTO

O planejamento abordou dois aspectos, a seleção de ferramentas e a escolha de alguns princípios de metodologia para o desenvolvimento.

### 5.2.1 Tecnologias

Diversas tecnologias foram utilizadas para o desenvolvimento e implantação da aplicação em modo de produção. Estas tecnologias estão elencadas no Quadro 23.

**Quadro 23** - Tecnologias utilizadas para o desenvolvimento e implantação da aplicação.

Software	Finalidade
PHP	Linguagem de <i>script open source</i> de uso comum, especialmente adequada para o desenvolvimento web, que foi utilizada como linguagem <i>server side</i> para a API.
Laravel	<i>Framework</i> para desenvolvimento em PHP, utilizado para desenvolvimento da API REST.
Nginx	Servidor <i>Web</i> utilizado para servir os recursos da API e as páginas da aplicação <i>Web</i> .
Maria DB	Servidor SGDB utilizado para persistir os dados da aplicação.
Javascript	Linguagem de scripts utilizada para criação das páginas da aplicação <i>Web</i> .
HTML	Linguagem de marcação utilizada para criação das páginas da aplicação <i>Web</i> .
CSS	Folhas de estilos para páginas HTML.
Vue.js	<i>Framework</i> para desenvolvimento <i>Javascript</i> utilizado para criação da aplicação <i>Web cliente side</i> ou <i>front-end</i> .

<i>Bootstrap</i>	<i>Framework</i> para criação de componentes visualmente estilizados com CSS para as páginas HTML.
------------------	--

**Fonte:** autoria própria.

Os *frameworks* utilizados ainda contam com outros softwares como dependência que não necessitam serem citados em sua totalidade.

Também foram utilizadas diversas tecnologias apenas para o desenvolvimento e versionamento da aplicação, que estão elencadas no **Quadro 24**.

**Quadro 24** - Tecnologias utilizadas durante o desenvolvimento da aplicação.

Software	Finalidade
<i>Git</i>	Sistema de versionamento utilizado para manter organizadas as versões e incrementos da aplicação.
<i>Git Flow</i>	Complemento para o sistema de versionamento <i>Git</i> utilizado para simplificar o trabalho com ramificações das versões.
<i>VSCode</i>	Editor de código.
<i>Node.js</i>	Plataforma para desenvolvimento de aplicações utilizando <i>Javascript</i> .
<i>Insomnia</i>	Cliente para testes de APIs REST.

**Fonte:** Autoria própria.

E por fim as tecnologias utilizadas para produzir a documentação da aplicação estão listadas no **Quadro 25**.

**Quadro 25** - Tecnologias utilizadas para documentar a aplicação.

Software	Finalidade
<i>Python</i>	Linguagem de programação sob a qual as ferramentas de conversão de <i>Markdown</i> para HTML foi escrita.
<i>MkDocs</i>	Ferramenta geradora de sites estáticos a partir de texto estruturado em <i>Markdown</i> escrito em linguagem <i>Python</i> .
<i>PyMarkdown</i>	<i>Parser</i> para conversão de <i>Markdown</i> para HTML escrito em <i>Python</i> .
<i>Draw.io</i>	Ferramenta para criação de diagramas.

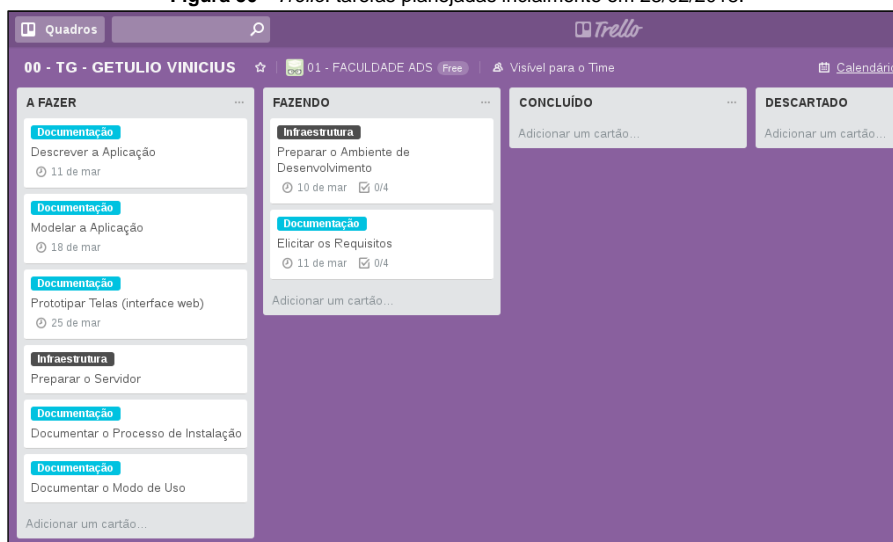
**Fonte:** Autoria própria.

### 5.2.2 Metodologia

Como metodologia para desenvolvimento do projeto foi escolhido o *Kanban*. A escolha se deu por ser este um método factível ao desenvolvimento solo da aplicação para o TG, conforme foi proposto na época em que o tema foi escolhido - 1º semestre do ano de 2017, durante a disciplina de Metodologia de Pesquisa Científica e Tecnológica.

A aplicação online *Trello* (<https://trello.com>) foi utilizada para elaboração do quadro *Kanban* com as atividades do projeto que foram planejadas, de modo que a Figura 50 traz uma visão das tarefas planejadas inicialmente em 23 de fevereiro de 2018.

**Figura 50 - Trello: tarefas planejadas inicialmente em 23/02/2018.**



**Fonte:** Autoria própria.

Já a Figura 51 traz uma visão das tarefas em 30 de março de 2018, demonstrando a evolução do processo de desenvolvimento.

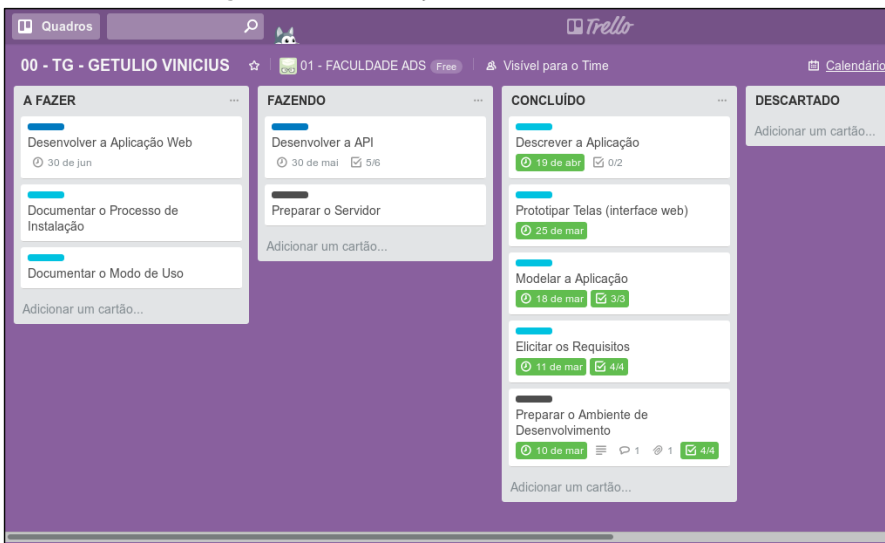
**Figura 51 - Trello: evolução das tarefas em 30/03/2018.**



Fonte: Autoria própria.

E por fim a Figura 52 mostra o estado das atividades em 11 de maio de 2018 em um cenário que restavam ainda algumas tarefas a serem entregues.

**Figura 52 - Trello: evolução das tarefas em 11/05/2018.**



Fonte: Autoria própria.

Também inerente a metodologia, foi decidido que o desenvolvimento da aplicação teria principalmente influência das características de métodos ágeis,

priorizando as entregas de funcionalidades em detrimento a documentação abrangente, em razão do tempo escasso para a entrega do TG.

Nota-se com as três figuras anteriores, que as atividades foram inseridas ou modificadas no decorrer da execução do projeto conforme é característico em abordagens de desenvolvimento que utilizam métodos ágeis.

No entanto, a documentação não foi posta em segundo plano, pois essa era desde o princípio a meta do TG, sendo, portanto, produzidos documentos à medida que a relevância dos mesmos para desenvolvimento do projeto era superior ao custo/esforço dispensado para obtenção.

A documentação escrita em *Markdown* pode ser visualizada no APENDICE A deste TG, onde foi inserida uma imagem com a visualização para cada uma das páginas do site estático gerada pelo software *MkDocs*.

A documentação em forma de site estático está disponível no endereço <https://getuliovinicius.github.io/trabalho.graduacao/>, e foi publicada por meio do recurso denominado *Github Pages* citado na sessão 4.6.2 deste TG.

### 5.3 EXECUÇÃO

O desenvolvimento do projeto, após a concepção e o planejamento d

### 5.4 FECHAMENTO

(O SISTEMA FOI TESTADO E INSTALADO NO SERVIDOR...)



## CONCLUSÃO

Após todo o trabalho...

## REFERÊNCIAS

- ALEXANDRE, E. D. S. M. Utilização de Markdown para elaboração de TCCs: concepção e experimento da ferramenta Limarka, João Pessoa, 04 mar. 2017. Disponível em: <<http://tede.biblioteca.ufpb.br:8080/handle/tede/9034>>. Acesso em: 23 nov. 2017.
- ANTONIO, D. Documentação de software – Vilã ou mocinha? **Devmedia**, 2011. Disponível em: <<https://www.devmedia.com.br/documentacao-de-software-vila-ou-mocinha/21795>>. Acesso em: 08 jan. 2018.
- BARE BONES. BBEEdit 12. **Bare Bones**, 2017. Disponível em: <<http://www.barebones.com/products/bbedit/>>. Acesso em: 23 nov. 2017.
- BECK, K. et al. Manifesto para Desenvolvimento Ágil de Software. **Manifesto for Agile Software Development**, 2001. Disponível em: <<http://agilemanifesto.org/iso/ptbr/manifesto.html>>. Acesso em: 22 jan. 2018.
- BECK, K. et al. Princípios por trás do Manifesto Ágil. **Manifesto for Agile Software Development**, 2001. Disponível em: <<http://agilemanifesto.org/iso/ptbr/principles.html>>. Acesso em: 22 jan. 2018.
- BELMIRO, N. J. **Sistemas de Informação**. São Paulo: Pearson Education do Brasil, 2012.
- BLANC, S. John Gruber: A Mix of the Technical, the Artful, the Thoughtful, and the Absurd. **Shawn Blanc**, 18 fev. 2008. Disponível em: <<http://shawnblanc.net/2008/02/interview-john-gruber/>>. Acesso em: 05 nov. 2017.
- CHRISTIE, T. MkDocs - Project documentation with Markdown. **MkDocs**, 2014. Disponível em: <<http://www.mkdocs.org>>. Acesso em: 19 mar. 2018.
- CODE.ORG. About US | Code.org. **Code.org**, 2017. Disponível em: <<https://code.org/about>>. Acesso em: 03 nov. 2017.
- FERREIRA, E.; EIS, D. HTML5 Curso W3C Escritório Brasil. **W3C Brasil**, 2011. Disponível em: <<http://www.w3c.br/Cursos/CursoHTML5>>. Acesso em: 23 nov. 2017.
- FILHO, W. D. P. P. **Engenharia de Software - Fundamentos, Métodos e Padrões**. 2. ed. ed. Rio de Janeiro: LTC, 2003.
- FORTIN, M. PHP Markdown. **Michel Fortin**, 14 jan. 2018. Disponível em: <<https://michelf.ca/projects/php-markdown/>>. Acesso em: 16 mar. 2018.
- FRAMEWORKS. **Projeto de Software Orientado a Objeto - Programa**. Disponível em: <<http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/frame/oque.htm>>. Acesso em: 13 fev. 2018.
- FURTADO, K. Metodologia ágil: documentação para projetos ágeis. **Devmedia**, 2016. Disponível em: <<https://www.devmedia.com.br/metodologia-agil-documentacao-para-projetos-ageis/37577>>. Acesso em: 03 nov. 2017.
- GRUBER, J. Dive Into Markdown. **Daring Fireball**, 19 mar. 2004. Disponível em: <[https://daringfireball.net/2004/03/dive\\_into\\_markdown](https://daringfireball.net/2004/03/dive_into_markdown)>. Acesso em: 09 nov. 2017.
- GRUBER, J. Introducing Markdown. **Daring Fireball**, 15 mar. 2004. Disponível em: <[https://daringfireball.net/2004/03/introducing\\_markdown](https://daringfireball.net/2004/03/introducing_markdown)>. Acesso em: 9 nov. 2017.
- GRUBER, J. Markdown. **Daring Fireball**, 2017. Disponível em: <<https://daringfireball.net/projects/markdown/>>. Acesso em: 04 nov. 2017.
- GRUBER, J. Markdown: Basics. **Daring Fireball**, 2017. Disponível em: <<https://daringfireball.net/projects/markdown/basics>>. Acesso em: 12 nov. 2017.

GRUBER, J. Markdown: Dingus. **Daring Fireball**, 2017. Disponível em: <<https://daringfireball.net/projects/markdown/dingus>>. Acesso em: 12 nov. 2017.

GRUBER, J. Markdown: Syntax. **Daring Fireball**, 2017. Disponível em: <<https://daringfireball.net/projects/markdown/syntax>>. Acesso em: 12 nov. 2017.

GUANABARA, G. Curso Python #01 - Seja um Programador. **YouTube**, Rio de Janeiro, 3 abr. 2017. Disponível em: <[https://www.youtube.com/watch?v=S9uPNppGsGo&list=PLHz\\_AreHm4dIKP6QQCekulPky1Ciwmdl6](https://www.youtube.com/watch?v=S9uPNppGsGo&list=PLHz_AreHm4dIKP6QQCekulPky1Ciwmdl6)>. Acesso em: 2 nov. 2017.

HEUSER, C. A. **Projeto de banco de dados**. 6 Ed. ed. Porto Alegre: Bookman, 2009.

INKSCAPE. Tutoriais em Texto. **Inkscape**, 2018. Disponível em: <<https://inkscape.org/pt-br/aprender/tutoriais/>>. Acesso em: 11 mar. 2018.

INTERNET ENGINEERING TASK FORCE (IETF). About. **Internet Engineering Task Force (IETF)**, 2018. Disponível em: <<https://www.ietf.org/about/>>. Acesso em: 16 mar. 2018.

JEKYLL. Jekyll - Transform your plain text into static websites and blogs. **Jekyll**, 2018. Disponível em: <<https://jekyllrb.com>>. Acesso em: 19 mar. 2018.

LEITNER, T. kramdown. **kramdown**, 2017. Disponível em: <<https://kramdown.gettalong.org/>>. Acesso em: 16 mar. 2018.

LEONARD, S. Guidance on Markdown: Design Philosophies, Stability Strategies, and Select Registration. **Internet Engineering Task Force (IETF)**, mar. 2016. Disponível em: <<https://tools.ietf.org/html/rfc7764>>. Acesso em: 16 mar. 2018.

LEONARD, S. RFC-7763 The text/markdown Media Type. **Internet Engineering Task Force (IETF)**, mar. 2016. Disponível em: <<https://tools.ietf.org/html/rfc7763>>. Acesso em: 16 mar. 2018.

LYNX. Lynx. **Lynx**, 2017. Disponível em: <<https://lynx.browser.org/>>. Acesso em: 23 nov. 2017.

MACFARLANE, J. Babelmark 2. **John MacFarlane**, 2012. Disponível em: <<http://johnmacfarlane.net/babelmark2/>>. Acesso em: 16 mar. 2018.

MACFARLANE, J. About Pandoc. **Pandoc - a universal document converter**, 2017. Disponível em: <<http://pandoc.org>>. Acesso em: 16 mar. 2018.

MACFARLANE, J. et al. CommonMark. **CommonMark**, 2017. Disponível em: <<http://commonmark.org>>. Acesso em: 05 nov. 2017.

MALHERBI, E. Prototipação de Sistemas utilizando a Ferramenta Balsamiq Mockup. **Devmedia**, 2013. Disponível em: <<https://www.devmedia.com.br/prototipacao-de-sistemas-utilizando-a-ferramenta-balsamiq-mockup/27232>>. Acesso em: 06 nov. 2017.

MARTINS, R. Kanban: 4 passos para implementar em uma equipe. **Devmedia**. Disponível em: <<https://www.devmedia.com.br/kanban-4-passos-para-implementar-em-uma-equipe/30218>>. Acesso em: 13 fev. 2018.

MDN WEB DOC - MOZILLA. Tag. **MDN Web Doc - Mozilla**, 03 ago. 2015. Disponível em: <<https://developer.mozilla.org/en-US/docs/Glossary/Tag>>. Acesso em: 22 nov. 2017.

MDN WEB DOC - MOZILLA. code. **MDN Web Doc - Mozilla**, 25 out. 2017. Disponível em: <<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/code>>. Acesso em: 23 nov. 2017.

MDN WEB DOC - MOZILLA. em. **MDN Web Doc - Mozilla**, 19 out. 2017. Disponível em: <<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/em>>. Acesso em: 20 nov. 2017.

MDN WEB DOC - MOZILLA. strong. **MDN Web Doc - Mozilla**, 04 ago. 2017. Disponível em: <<https://developer.mozilla.org/pt-BR/docs/Web/HTML/Element/strong>>. Acesso em: 20 nov. 2017.

MOVABLETYPE.ORG. What is Movable Type? **MovableType.Org**, 2017. Disponível em: <<https://www.movabletype.org/>>. Acesso em: 23 nov. 2017.

OBJECT MANAGEMENT GROUP. **OMG Unified Modeling Language (OMG UML)**. Version 2.5.1. ed. Needham: [s.n.], 2017. Disponível em: <<https://www.omg.org/spec/UML/2.5.1>>.

OBJECT MANAGEMENT GROUP, INC. (OMG). **Business Process Model and Notation (BPMN) - Version 2.0**. [S.I.]: [s.n.], 2011. Disponível em: <<http://www.omg.org/spec/BPMN/2.0>>.

PRESSMAN, R. S. **Sistemas de Informação - Uma Abordagem Profissional**. 7. ed. ed. Porto Alegre: AMGH, 2011.

ROBERT, K. S. perlintro. **perldoc.perl.org Pear Programming Documentation**, 2017. Disponível em: <<http://perldoc.perl.org/perlintro.html>>. Acesso em: 23 nov. 2017.

SAUDADE, A. **REST - Construa APIs inteligentes de maneira simples**. [S.I.]: Editora Casa do Código, 2014.

SCHWBER, K.; SUTHERLAND, J. **Guia do Scrum**. [S.I.]: [s.n.], 2017. Disponível em: <<http://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-Portuguese-Brazilian.pdf>>. Acesso em: 13 fev. 2018.

SENE, R. P. D. Processo, qualidade e métricas de desenvolvimento de software. **Engenharia de Software Magazine**, Rio de Janeiro, n. 55, p. 54, 2012. Disponível em: <<https://www.devmedia.com.br/revista-engenharia-de-software-magazine-55/26939>>.

SIGNIFICADOS. Significado de Blog. **Significados**, 2017. Disponível em: <<https://www.significados.com.br/blog/>>. Acesso em: 23 nov. 2017.

SMITH, L. Best Practices for Plain-Text Emails + A Look at Why They're Important. **litmus**, 15 out. 2014. Disponível em: <<https://litmus.com/blog/best-practices-for-plain-text-emails-a-look-at-why-theyre-important>>. Acesso em: 09 nov. 2017.

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. ed. São Paulo: Pearson, 2011.

SOUZA, V. R. D.; VALE, R. C.; ARAÚJO, M. A. P. Prototipação no Desenvolvimento de Software. **Engenharia de Software**, Rio de Janeiro, n. 17, p. 66, 2008. Disponível em: <<https://www.devmedia.com.br/revista-engenharia-de-software-17/14495>>.

STATICGEN. StaticGen - Top Open-Source Static Site Generators. **StaticGen**, 19 mar. 2018. Disponível em: <<https://www.staticgen.com>>. Acesso em: 19 mar. 2018.

THE PYTHON-MARKDOWN PROJECT. Python-Markdown. **Python-Markdown**, 2017. Disponível em: <<https://python-markdown.github.io/>>. Acesso em: 16 mar. 2018.

VUE.JS. Vue.js - Guia 2.x. **Vue.js**, 2018. Disponível em: <<https://br.vuejs.org/v2/guide/>>. Acesso em: 11 mar. 2018.

W3C. XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition). **W3C**, 26 jan. 2000. Disponível em: <<https://www.w3.org/TR/xhtml1/>>. Acesso em: 23 nov. 2017.

## APENDICE A

Arquivo: /mds/index.md
<p>Orçamento Pessoal - tgGV</p> <p>=====</p> <p><b>## Sobre</b></p> <p>Orçamento Pessoal - tgGV, é um software livre desenvolvido como Trabalho de Graduação no curso de Análise e Desenvolvimento de Sistemas da Faculdade de Tecnologia de Franca, FATEC Dr. Thomaz Novelino.</p>
Arquivo: /mds/licence.md
<p>Licença</p> <p>=====</p> <p>O projeto que estou desenvolvendo pode ser usado, copiado, modificado e redistribuído pois está licenciado sob [GNU GENERAL PUBLIC LICENSE - Version 3, 29 June 2007](https://github.com/getuliovinicius/trabalho.graduacao/blob/master/LICENCE).</p> <p>Códigos de terceiros utilizados na construção desta aplicação, tais como frameworks e bibliotecas, estão sujeitos as suas respectivas licenças:</p> <p>+ Laravel - The Laravel framework is open-sourced software licensed under the [MIT license](https://opensource.org/licenses/MIT)</p>
Arquivo: /mds/doc-desenvolvimento/0-descricao.md
<p>Descrição</p> <p>=====</p> <p><b>## Aplicação</b></p> <p>Trata-se de uma aplicação web com a finalidade de prover um serviço de ...</p>
Arquivo: /mds/doc-desenvolvimento/1-requisitos.md
<p>Documento de Requisitos</p> <p>=====</p> <p><b>**Atualizado em:**</b> 27/02/2018</p> <p><b>## Requisitos do usuário (cliente)</b></p>

- [ ] O usuário pode criar uma conta para gerenciar seu orçamento pessoal.
- [ ] O usuário pode alterar sua senha de acesso a aplicação e outros dados pessoais.
- [ ] O usuário deve criar um plano de contas observando os grupos Ativo, Passivo, Patrimônio Líquido, Receita e Despesa.
- [ ] O usuário deve alimentar o saldo inicial das contas.
- [ ] O usuário faz lançamentos nas contas.
- [ ] O usuário pode realizar correções nos lançamentos.
- [ ] O usuário lista os lançamentos de uma contas.
- [ ] O usuário visualiza os saldos das contas.
- [ ] O usuário visualiza os saldos das contas.

### ## Requisitos do usuário (empresa)

- [ ] O usuário do tipo gerente pode acessar realatórios gerenciais.
- [ ] O usuário do tipo administrador pode acessar realatórios gerenciais e gerenciar as contas de usuários.

### ## Requisitos da aplicação

- [ ] Autenticação de usuários.
- [ ] Ao criar uma conta o usuário deve ser direcionado para um processo de alimentação das contas iniciais.
- [ ] O usuário `root` não pode ser excluído.
- [ ] A aplicação deve registrar em log as ações do usuário (criação, alteração e exclusão de lançamentos e acessos)

### ## Requisitos de tecnologia

- [ ] Aplicação orientada a serviço.
- [ ] Web service Rest.
- [ ] Cliente web da aplicação.

Arquivo: /mds/doc-desenvolvimento/2-modelagem.md

Modelagem

=====

### ## Processo

![BPMN](img/bpmn-1.png)

### ## Caso de uso

### ## Banco de dados

Arquivo: /mds/doc-desenvolvimento/3-prototipacao.md

Prototipação =====
Arquivo: /mds/doc-desenvolvimento/4-tecnologias.md
Tecnologias =====
Pilha de software utilizada para desenvolvimento da aplicação.
Tecnologia   Descrição   Versão
:-----  :-----  :-----
Laravel   <i>_Framework_</i> PHP   5.6

APENDICE B

Página: Início

tgGV

Índice

Licença

Orçamento Pessoal - tgGV

Índice

Sobre

Sobre

Orçamento Pessoal - tgGV, é um software livre desenvolvido como Trabalho de Graduação no curso de Análise e Desenvolvimento de Sistemas da Faculdade de Tecnologia de Franca, FATEC. Dr. Thomaz Novelino.

Próximo

Licença

→

Copyright: 2018 Getúlio Vinicius

powered by MkDocs and Material for MkDocs

Página: Licença

tgGV

Índice

Licença

Licença

Índice

Sobre

O projeto que estou desenvolvendo pode ser usado, copiado, modificado e redistribuído pois está licenciado sob [GNU GENERAL PUBLIC LICENSE - Version 3, 29 June 2007](#).

Códigos de terceiros utilizados na construção desta aplicação, tais como frameworks e bibliotecas, estão sujeitos as suas respectivas licenças:

- Laravel - The Laravel framework is open-sourced software licensed under the [MIT license](#)

Anterior

Índice

←

Próximo

Descrição

→

Copyright: 2018 Getúlio Vinicius

powered by MkDocs and Material for MkDocs



Página: Desenvolvimento - Descrição

tgGV

Buscar

GitHub

0 Stars - 0 Forks

Início

Desenvolvimento

Instalação

Guia do usuário

Desenvolvimento

Descrição

Requisitos

Modelagem

Prototipação

Tecnologias

Descrição

Aplicação

Trata-se de uma aplicação web com a finalidade de prover um serviço de ...

Índice

Aplicação

Anterior

Próximo

←

→

Licença

Requisitos

Copyright 2018 Getulio Vinicius

powered by MDDocs and Material for MDDocs

Página: Desenvolvimento - Requisitos

tgGV

Buscar

GitHub

0 Stars - 0 Forks

Início

Desenvolvimento

Instalação

Guia do usuário

Desenvolvimento

Descrição

Requisitos

Modelagem

Prototipação

Tecnologias

Documento de Requisitos

Atualizado em: 27/02/2018

Requisitos do usuário (cliente)

☐ O usuário pode criar uma conta para gerenciar seu orçamento pessoal.

☐ O usuário pode alterar sua senha de acesso a aplicação e outros dados pessoais.

☐ O usuário deve criar um plano de contas observando os grupos Ativo, Passivo, Patrimônio Líquido, Receita e Despesa.

☐ O usuário deve alimentar o saldo inicial das contas.

☐ O usuário faz lançamentos nas contas.

☐ O usuário pode realizar correções nos lançamentos.

☐ O usuário lista os lançamentos de uma contas.

☐ O usuário visualiza os saldos das contas.

☐ O usuário visualiza os saldos das contas.

Requisitos do usuário (empresa)

☐ O usuário deve acessar cada semana o relatório de despesas.

Índice

Requisitos do usuário (cliente)

Requisitos do usuário (empresa)

Requisitos da aplicação

Requisitos de tecnologia

tgGV

Buscar

GitHub

0 Stars · 0 Forks

Início

Desenvolvimento

Instalação

Guia do usuário

Desenvolvimento

Descrição

Requisitos

Modelagem

Prototipação

Tecnologias

Modelagem

Processo

Índice

Processo

Caso de uso

Banco de dados

tgGV

Buscar

GitHub

0 Stars · 0 Forks

Início

Desenvolvimento

Instalação

Guia do usuário

Desenvolvimento

Descrição

Requisitos

Modelagem

Prototipação

Tecnologias

Prototipação

Anterior

Modelagem

Próximo

Tecnologias

Copyright 2018 Getúlio Vicius  
powered by MkDocs and Material for MkDocs

tgGV

Buscar

GitHub

0 Issues · 0 Forks

Início

Desenvolvimento

Instalação

Guia do usuário

Desenvolvimento

Descrição

Requisitos

Modelagem

Prototipação

Tecnologias

Tecnologias

Pilha de software utilizada para desenvolvimento da aplicação.

Tecnologia	Descrição	Versão
Laravel	Framework PHP	5.6

Anterior

Prototipação

Próximo

Introdução

Copyright 2018 Getulio Vinctus

powered by [MkDocs](#) and [Material for MkDocs](#)