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

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

GETÚLIO VINÍCIUS TEIXEIRA DA SILVA

DESENVOLVIMENTO DE SOFTWARE E DOCUMENTAÇÃO

Aspectos Gerais e o Uso da Linguagem *Markdown*

FRANCA/SP
2018

GETÚLIO VINÍCIUS TEIXEIRA DA SILVA

DESENVOLVIMENTO DE SOFTWARE E DOCUMENTAÇÃO

Aspectos Gerais e o Uso da Linguagem *Markdown*

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.

Orientadora: Dra. Jaqueline Brigladori Pugliesi

FRANCA/SP

2018

GETÚLIO VINÍCIUS TEIXEIRA DA SILVA

DESENVOLVIMENTO DE SOFTWARE 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.

Trabalho avaliado e aprovado pela seguinte Banca Examinadora:

Orientadora ..... : _____.
Nome..... : Profa. Dra. Jaqueline Brigladori Pugliesi
Instituição ..... : Faculdade de Tecnologia de Franca – "Dr. Thomaz Novelino".

Examinador(a) 1.. : _____.
Nome..... : Prof. Esp. Cláudio Eduardo Paiva
Instituição ..... : Faculdade de Tecnologia de Franca – "Dr. Thomaz Novelino".

Examinador(a) 2.. : _____.
Nome..... : Profa. Dra. Silvia Regina Viel
Instituição ..... : Faculdade de Tecnologia de Franca – "Dr. Thomaz Novelino".

Franca, 14 de junho de 2018.

AGRADECIMENTO

Agradeço aos meus pais e irmãos que são quem me sustentam moralmente e compreendem o esforço que tenho feito.

Oportunamente, agradeço aos professores da Fatec Franca e aos colegas de turma.

Agradeço aos amigos de toda minha vida.

Especialmente, agradeço minha amiga Andréia, que deveria receber um prêmio por ser a pessoa mais legal do mundo.

Dedico o presente Trabalho de Graduação aos meus familiares, em especial à minha esposa e meus filhos, que ainda não conheço, mas acho importante que um dia saibam que há muito tempo tenho pensado neles.

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

Este trabalho de graduação aborda o processo de documentação para projetos de *softwares*, que é respaldado por princípios da *Engenharia* de *Software* tais como a constante busca por atributos de qualidade e os métodos ágeis de desenvolvimento. Nesse contexto, foi realizado um estudo acerca da linguagem *Markdown*, que foi criada por John Gruber no ano de 2004 e tem sido adotada em muitas aplicações *Web* como padrão para formatação de textos. Ademais, *Markdown* é considerada uma linguagem de marcação de texto leve, sendo assim, é apropriada para o uso na elaboração da documentação completa de projetos. Esse processo é realizado com o auxílio de ferramentas, chamadas de *parsers,* que convertem textos escritos em *Markdown* para o formato HTML. Tais ferramentas são utilizadas por softwares como o gerador de sites estáticos *MkDocs*. Assim, para demonstrar essa possibilidade, foi iniciado o desenvolvimento de um *software* livre para a *Web* denominado “Orçamento Pessoal – tgGV”, que é composto por uma API REST desenvolvida com o *framework* PHP *Laravel* e uma aplicação *Web* cliente desenvolvida com o *framework Javascript Vue.js*. A aplicação foi documentada com *Markdown* e a documentação foi publicada no serviço *GitHub Pages* como um site estático gerado pelo *MkDocs*.

Palavras-chave: Documentação de projeto de software. Markdown. MkDocs. Orçamento Pessoal.

ABSTRACT

This paper talks about the documentation process for software projects, which is backed by principles of Software Engineering such as: the constant search for quality attributes and agile methods of development. In this context, a study about the *Markdown* language was developed, which was created by John Gruber in the year of 2004 and has been adopted in many Web applications as standard for text formatting. In addition, *Markdown* is considered a light text markup language and therefore is suitable for use in the development of complete project documentation. This process is performed with the help of tools, called *parsers*, that convert texts written in *Markdown* to HTML format. Such tools are used by software such as the static site generator *MkDocs*. To demonstrate this possibility, the development of free software for the Web called "Orçamento Pessoal - tgGV", which is composed of a REST API developed with the PHP Laravel framework and a client Web application developed with the framework Javascript Vue.js. The application was documented with *Markdown* and the documentation was published in the GitHub Pages service as a static site generated by *MkDocs*.

Keywords: Software Project Documentation. *Markdown*. *MkDocs*. Personal Budget.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

ADS – Análise e Desenvolvimento de Sistemas
API – *Application Programing Interface*
BD – Banco de Dados
BPMN – *Business Process Model and Notation*
CLI – *Command Line Interface*
CSS – *Cascading Style Sheet*
DER – Diagrama Entidade Relacionamento
DR – Documento de Requisitos
ER – Entidade Relacionamento
ES – Engenharia de *Software*
GFM – *GitHub Flavored Markdown*
GNU – *GNU is not Unix*
HHVM – *Hip Hop Virtual Machine*
HTML – *Hypertext Markup Language*
HTTP – *Hypertext Transfer Protocol*
IETF – *Internet Engineering Task Force*
JSON – *Javascript Object Notation*
MER – Modelo Entidade Relacionamento
MINE – *Multiporpouse Internet Mail Extension*
MPCT – Metodologia de Pesquisa Cientifica e Tecnológica
MT – *Movable Type*
OTAN – Organização do Tratado do Atlântico Norte
PDF – *Portable Document File*
PHP – PHP *hypertext processor*
REST – *Representational State Transfer*
RFC – *Request For Comments*
SGBD – Sistema de Gerenciamento de Banco de Dados
SI – Sistema de Informação
SO – Sistema Operacional
TG – Trabalho de Graduação
TI – Tecnologia da Informação
tgGV – trabalho de graduação de Getúlio Vinícius
UML – *Unified Modeling Language*
URI – *Universal Resource Identifier*
URL – *Uniform Resource Location*
XHTML – *Extensible Hypertext Markup Language*
XML – *Extensible Markup Language*
XP – *Extreme Programming*
YAML – *YAML Ain't Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Seres humanos utilizam programas de computadores para realizarem tarefas que muitas vezes não são capazes de concluírem, 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, *online*) 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.

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, *online*).

A ação da *Code.org* é concentrada na educação básica, investindo em programas pedagógicos de iniciação à 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, *online*).

Assim, expostos alguns fatos acerca do cenário atual de desenvolvimento e de perspectivas de expansão deste mercado, em face às ações de organizações como a *Code.org*, pode-se observar 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), 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 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 (2011, p. 5) “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 (Ponto de Partida - Fundamentação), este Trabalho de Graduação (TG) tem como objetivo ser um estudo de caso 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; o capítulo 3 (Documentação de *Software*) trará uma visão em linhas gerais sobre os tipos de documentação os quais um *software* demanda; o capítulo 4 (Linguagem *Markdown*) terá uma introdução a linguagem *Markdown,* que é muito utilizada em sites de repositório de código como o *GitHub*[1] e uma explicação sobre os *softwares* para geração de sites estáticos, especialmente o *MkDocs*; o capítulo 5 (Aplicação Prática - Projeto) 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*.

Além disso, o capítulo final deste trabalho apresentará considerações sobre o estudo em torno do processo de documentação de *software* aplicando a linguagem *Markdown* e o software gerador de sites estáticos *MkDocs.*

Por fim, os Apêndices A e B apresentarão, respectivamente, toda a documentação escrita com a linguagem *Markdown* e as figuras que representam a modelagem e a prototipação do projeto de software que será apresentado no capítulo 5.

[1] https://github.com

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, para a qual era necessária a apresentação de um projeto de *software* para acompanhamento dos estudos.

Assim, 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, utilizando o modelo proposto pela instituição, foi obtida uma vasta documentação e não apenas um conjunto de requisitos, conforme o título levava a crer. Assim, observou-se que o documento final continha informações oriundas de diversas ferramentas, obtidas 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, um conjunto de técnicas, ferramentas e abordagens, bem como um processo pouco claro e generalista que desencadeou um grande desafio, que era, compreender e codificar sob o prisma dos conceitos de ES tudo o que havia sido construído em relação ao projeto apresentado.

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 então 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, que sugere a integração de ferramentas de modo a possibilitar que uma utilize informações geradas por outra, chegou-se a um novo desafio Pressman (2011, p. 40). Tal desafio consistia em 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 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 visa apresentar uma forma viável de promover a gestão da documentação produzida em um projeto de *software*, de modo que esta forma será vista como uma alternativa, que, dependendo do projeto que se queira iniciar, pode vir a ser empregada.

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 Paula Filho (2013, p. 7) conta com “uma data de início, uma data de fim, uma equipe (da qual faz parte um responsável, a que chamaremos ‘gerente do projeto’) e outros recursos”.

Ainda segundo Paula Filho (2013), um projeto representa a execução de um processo e este, quando bem definido, possui etapas elaboradas de modo a possibilitar uma avalição 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.

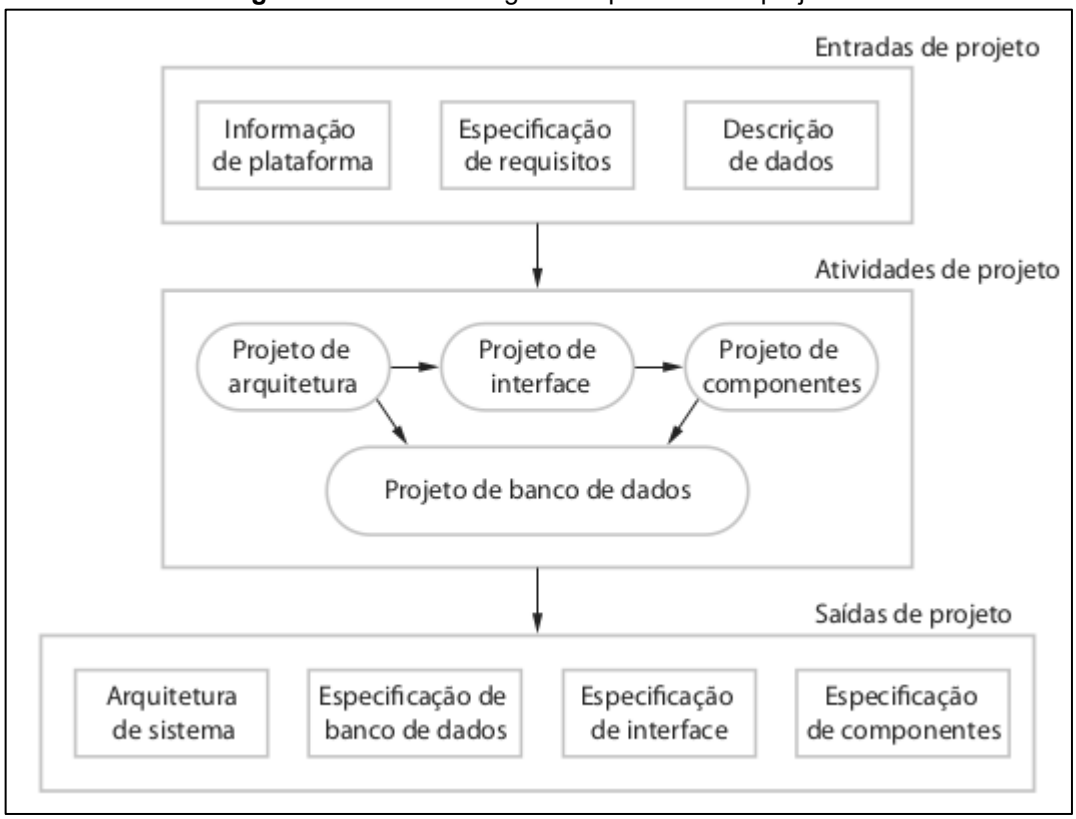
Tem-se então, uma visão documental de projeto e pode-se concluir que ambas as visões, processual e documental, contemplam essencialmente a finalidade organizacional expressa por meio do jargão projeto.

Desse 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;
3. Geração de documentos de especificações; e
4. Construção (codificação) do *software* com base nas especificações geradas na etapa 3.

Um modelo geral de processo de projeto, antecedendo a etapa de construção, 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 utilidade 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.

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 Paula Filho (2013), 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” (PAULA FILHO, 2013, p. 8).

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 da ES analogamente representada por camadas na Figura 2.

Figura 2 – Camadas da ES.

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 prazo para entrega, que é um dos problemas do *software* e também uma meta para a gestão de projetos Sommerville (2011, p. 414), 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

A respeito do processo de software, o estudo dos capítulos 2 e 3 da obra de Pressman (2011), pode conduzir ao entendimento de que 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.

2.2.1.1 Métodos prescritivos

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

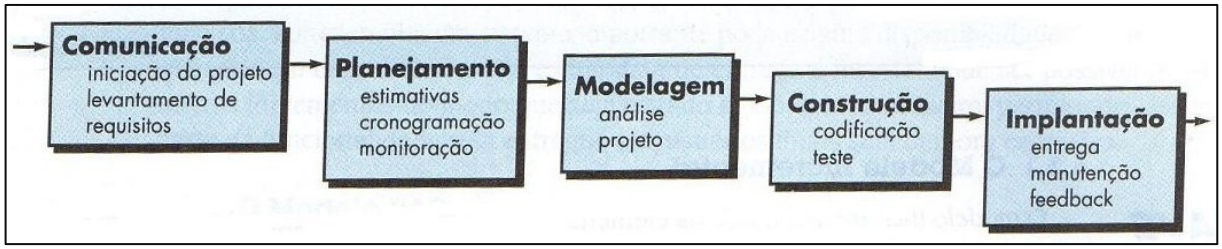
Ainda segundo Lusa (2011, *online*), o surgimento dos métodos prescritivos se deu em função de um período anterior crítico denominado “Crise do *Software*”, o que coaduna com o relato de Sommerville (2011, p. xi) sobre os problemas no desenvolvimento de *software* abordados na OTAN.

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 surgiram como alternativa, 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.

2.2.1.2 Métodos ágeis

O Desenvolvimento Ágil, amplamente adotado a partir do lançamento do Manifesto Ágil por Beck *et al.* (2001 a, *online*), 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 contínua 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 *et al.*, 2001 a, *online*).

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 *et al.*, 2001 b, *online*).

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”.

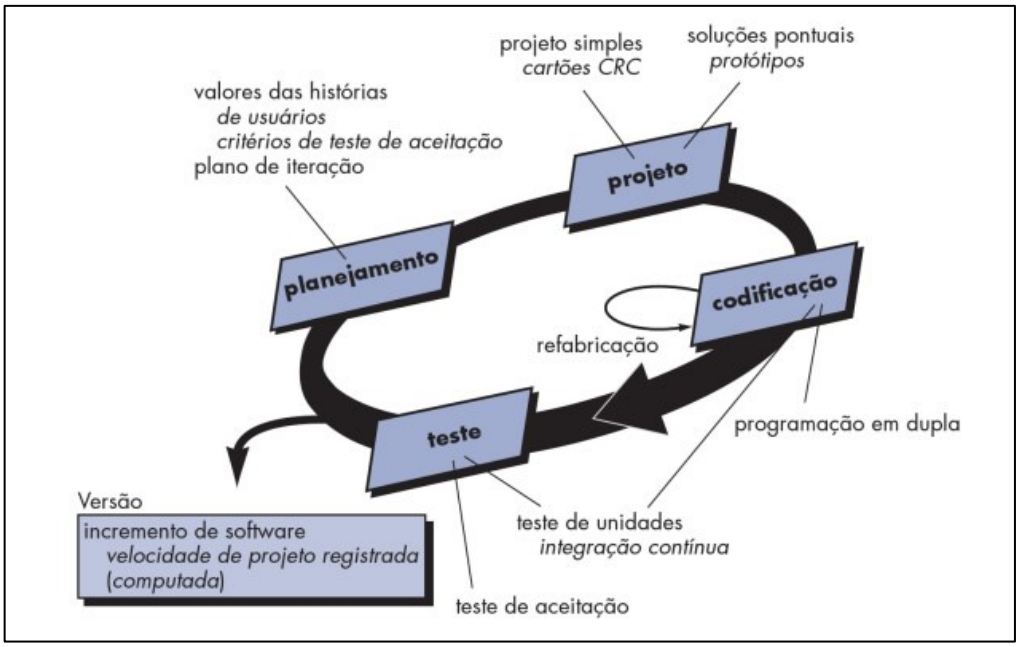
Portanto, a flexibilidade está na capacidade de adaptar o projeto as novas necessidades na medida em que forem surgindo, ou seja, responder as mudanças em detrimento ao planejamento anterior, conforme previsto no manifesto.

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

O *Extreme Programming – XP* (Programação Extrema), e o *Scrum* são considerados expoentes do desenvolvimento ágil. No caso do 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.

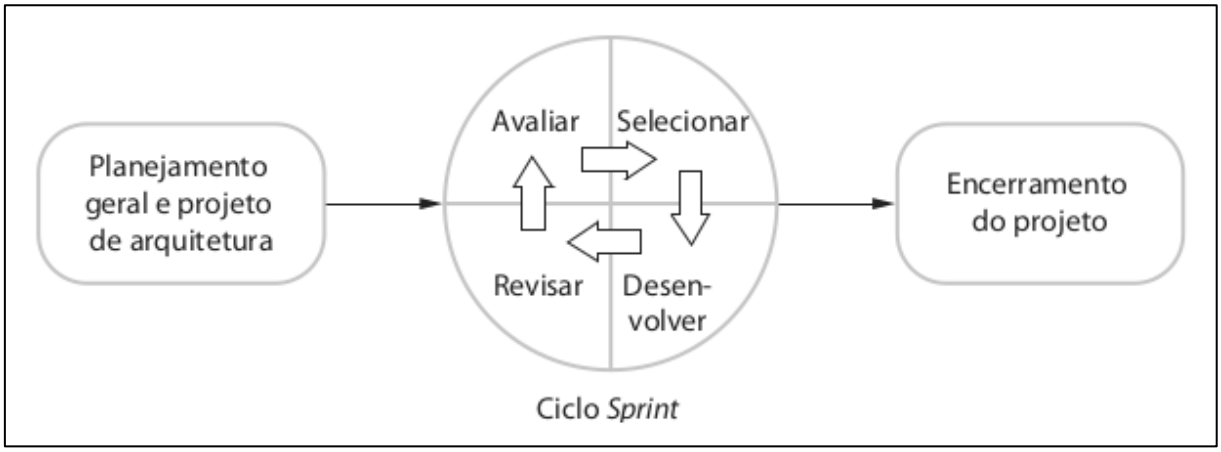
Figura 4 – O processo da *Extreme Programming (XP)*.



Fonte: Pressman, 2011, p. 88.

Já no *Scrum*, 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*.

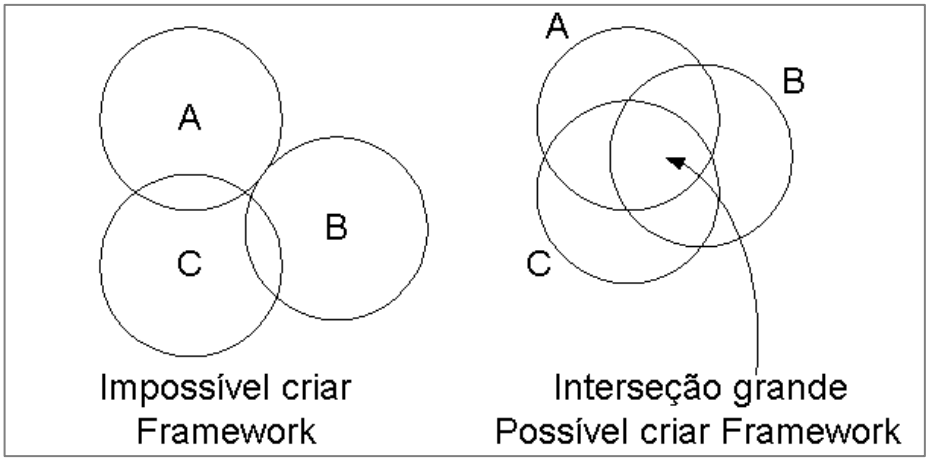


Fonte: Sommerville, 2011, p. 50.

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, caracteriza-se como um *framework* de desenvolvimento. Os próprios criadores do *Scrum* o definem com um *framework* (SCHWBER; SUTHERLAND, 2017, p. 3).

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

Figura 6 – Criação de um framework.



Fonte: Frameworks, 2018, *online*.

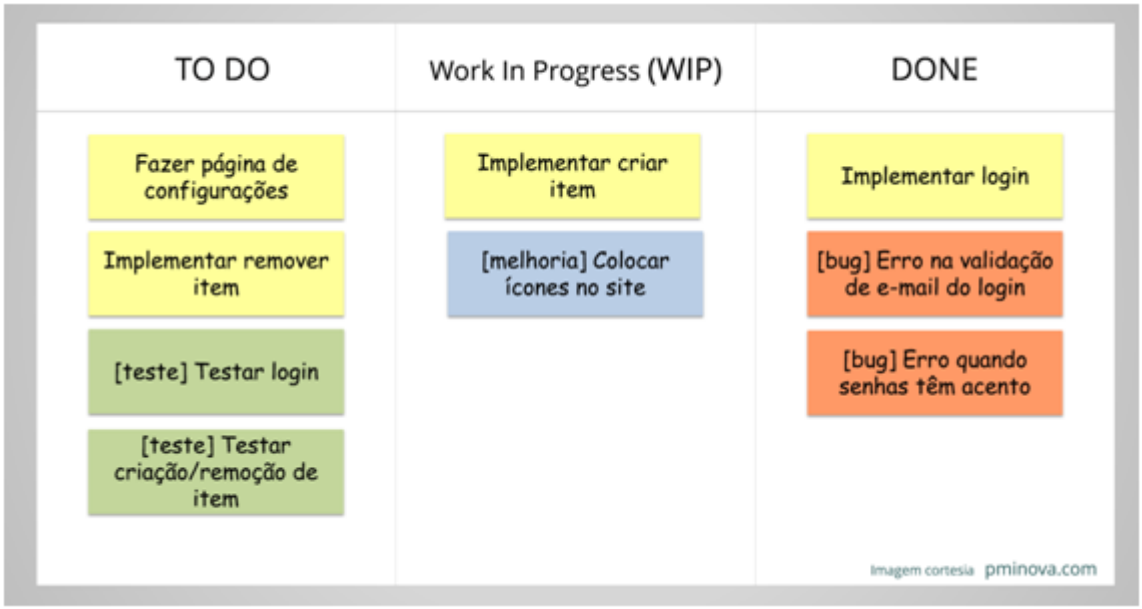
Assim, é possível chegar a uma breve explicação sobre o termo, que consiste em um conjunto de soluções. Tal conjunto pode ser utilizado em várias aplicações, ou seja, soluciona problemas que são inerentes ao desenvolvimento de determinadas atividades desde que haja algo em comum, ainda que essas atividades 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, 2018, *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.

Nota-se, ao ler a obra de Paula Filho (2013, p. 8), que no início de um projeto, geralmente tem-se a tarefa de levantamento dos requisitos, aos 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, estórias 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.

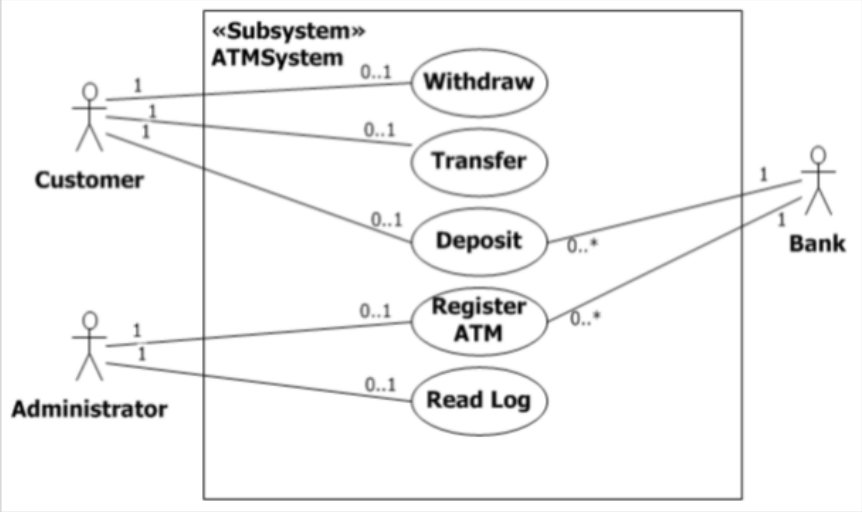
Fonte: Sommerville, 2011, p. 68.

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* (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* (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.

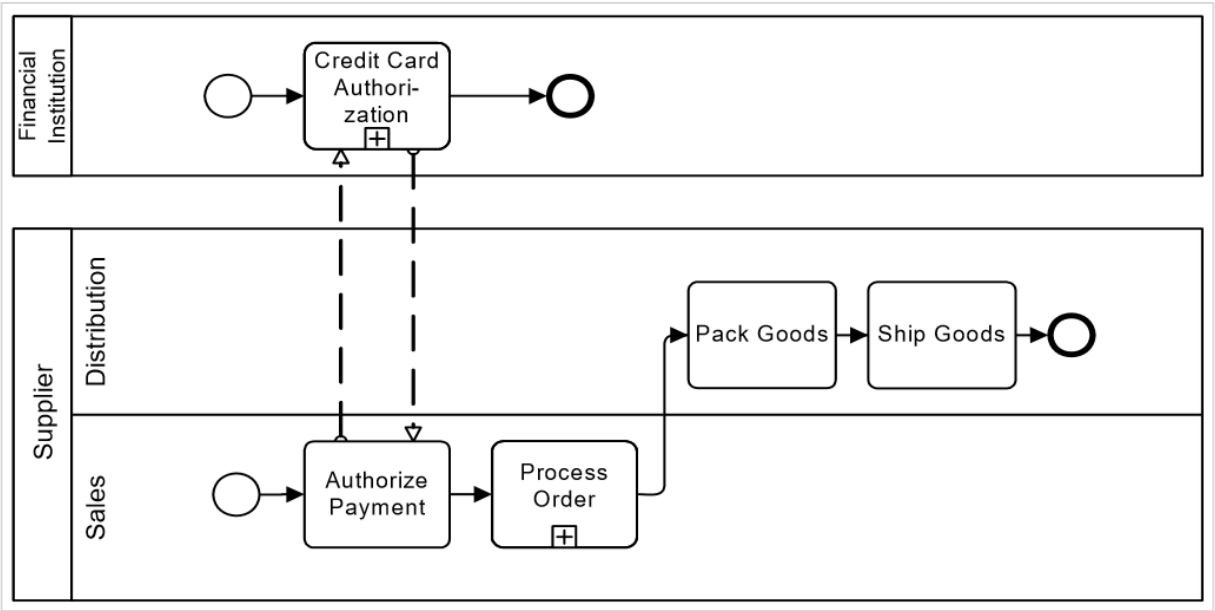
Um diagrama BPMN possui elementos gráficos, figuras geométricas, que são dispostas de modo a estabelecer um fluxo do princípio ao fim de um processo, inclusive abrangendo subprocessos.

Nos diagramas é possível observar, entre outras características do processo, a interação entre 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.

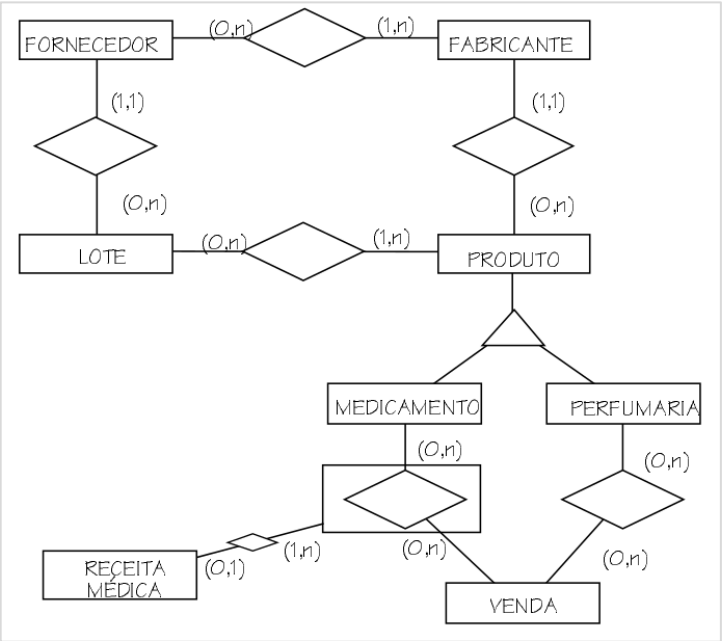
No exemplo, é possível observar que a piscina inferior, que representa a parcela do processo executada pelo Fornecedor (*Supplier*), possui duas raias/atores, representando os setores de Vendas (*Sales*) e Distribuição (*Distribuition*) e a piscina superior, que representa a parcela do processo executada pela Instituição Financeira (*Financial Institution*) possui apenas uma raia, indicando que toda a 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 construção, 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 a no mínimo nenhum LOTE e no máximo a *n* LOTES.

Outra concepção relacionada com a elaboração de BD é o Modelo Lógico, que 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 Lógico de um cadastro de produtos.

Modelo Lógico – Cadastro de Produtos
TipoDeProduto (CodTipoProd, DescrTipoProd) Produto (CodProd, DescrProd, PrecoProd, CodTipoProd) CodTipoProd referência 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 modelos 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".

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", (SOUZA; VALE; 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.

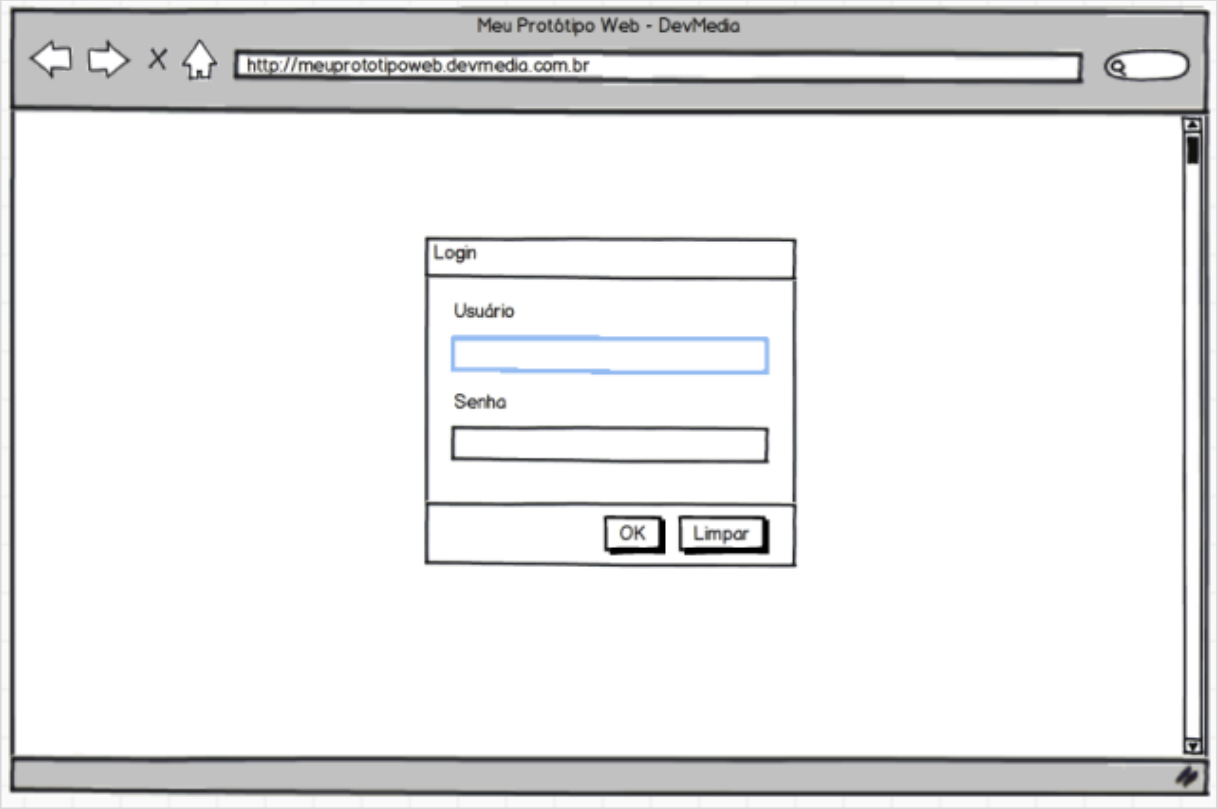
Pode-se classificar os protótipos em três categorias: baixa fidelidade, média fidelidade e alta fidelidade. Deste modo, os protótipos de baixa fidelidade são destinados a crítica do produto em relação ao levantamento de requisitos; os protótipos de média fidelidade são destinados a representação visual do produto de maneira fidedigna; e por fim os protótipos de alta fidelidade destinam-se ao teste de 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 tendem a aumentar. Deste modo, o uso dessa ferramenta é regrado 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* (desenhos de tela), 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*.



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 que, 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 (SENE, 2012).

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 (SOMMERVILLE, 2011, p. 63).

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.

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 comercias, 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).

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.

Além disso, segundo Furtado (2016, *online*), 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*.

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 ou não documente aquilo que está fazendo (como por exemplo, não somente os objetivos e 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, *online*) 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.

Desse 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 (guia de uso) 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[2] – *Hypertext Markup Language* (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 – *Command Line Interface* (Interface de Linha de Comando) dos Sistemas Operacionais (SO)s *GNU/Linux,* sendo GNU um acrônimo para *GNU is not Unix* (GNU não é Unix).

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

---

[2] **HTML:** Linguagem de marcação de texto estruturado para publicação de conteúdo (texto, imagem, vídeo e áudio) na *web* (FERREIRA; EIS, 2011, *online*).

documentação na base de dados do programa *Man*, tornando-a acessível para o usuário do computador.

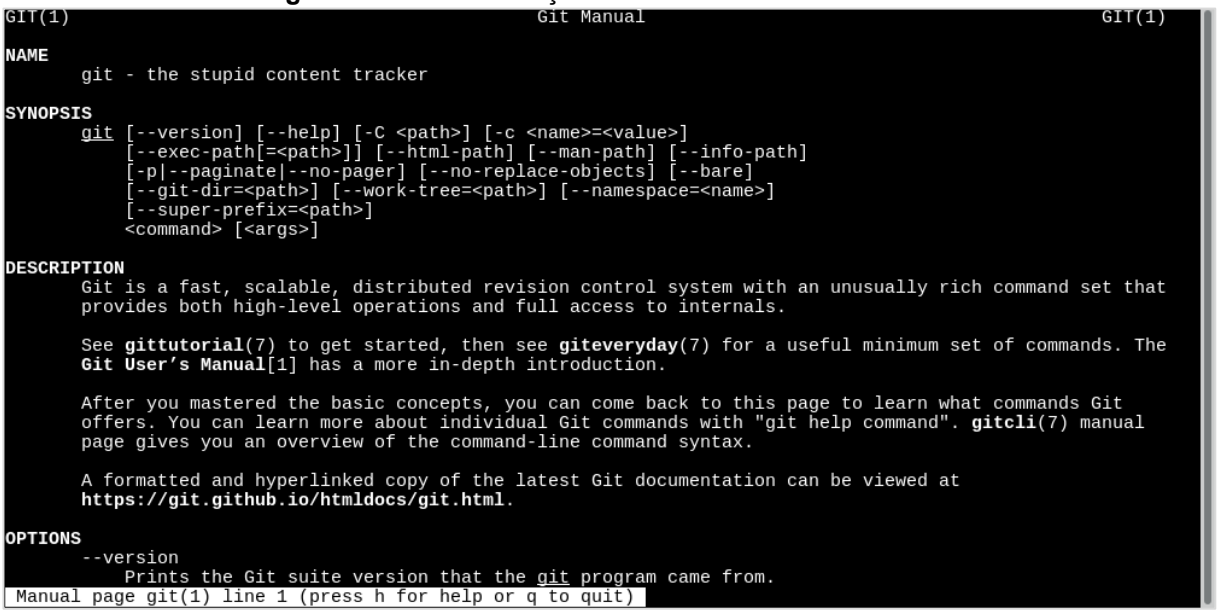
Um exemplo de utilização do *Man* pode ser observado na Figura 12, onde é solicitada a documentação do software de controle de versão *Git* através da CLI em um terminal do SO *Debian GNU/Linux*.

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



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:

HTML

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

JS

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

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.

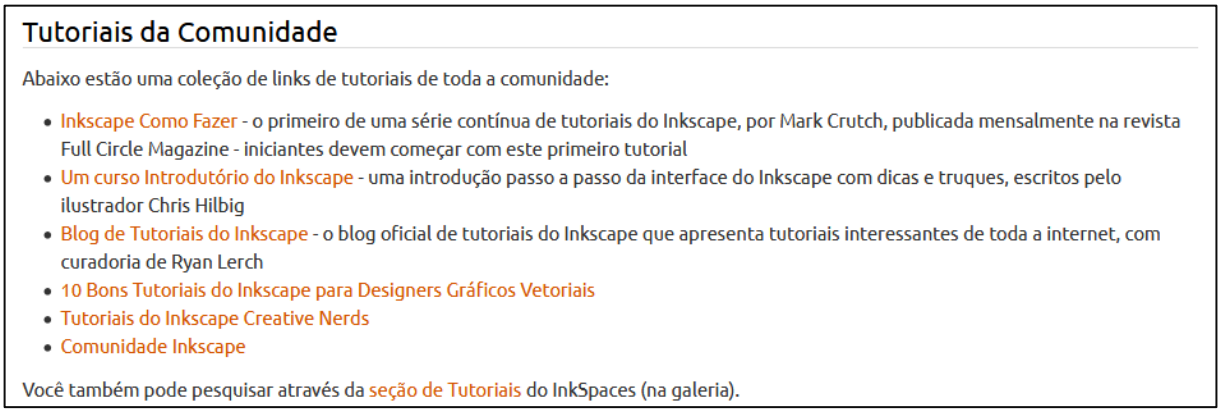
3.2.2 Documentação colaborativa

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 se 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 mantem o desenvolvimento do *software*, onde é possível observar links para conteúdo produzido pela comunidade.

Figura 15 – Links para documentação colaborativa do *sofware Inkscape*.



Fonte: Inkscape, 2018, *online*.

3.2.3 Segurança da informação

Considerando o exemplo de um Sistema de Informação (SI), que nas palavras de Belmiro (2012, p. 6) é definido como:

> Um conjunto de componentes relacionados entre si que coletam (ou recuperam), processam, armazenam e distribuem informações que servem para apoiar a tomada de decisões, a coordenação e o controle de uma organização.

Pode-se supor, que deve existir regras para acesso a documentação segmentada pelos níveis hierárquicos 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, especifica 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, espera-se 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, que 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.

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, *online*). 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.

Na página dedicada ao projeto, que faz parte do *blog Daring Fireball*, Gruber (2017, *online*) 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 vasão a iniciativa de criação da linguagem.

Conforme o relato de Gruber (2004 a, *online*), manipular as *tags*[3] 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*[4]; (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*[5] no *MT*[6], colar o artigo e publicar" (GRUBER, 2004 a, *online*).

Esse processo contrastava com a opinião de John Gruber em relação ao uso de computadores. Segundo ele "a principal vantagem de usar um computador para

---

[3] ***Tag*****:** Refere-se à 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 pela (MOZILLA, 2017, *online*).
[4] ***BBEdit*****:** Editor de textos HTML feito para o sistema operacional *Macintosh* da empresa *Apple* (BARE BONES, 2017, *online*).
[5] ***Login*****:** Autenticação em um sistema.
[6] ***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, *online*).

escrever é o imediatismo da edição. Escreva, leia, revise, tudo na mesma janela, tudo no mesmo modo” (GRUBER, 2004 a, *online*).

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:

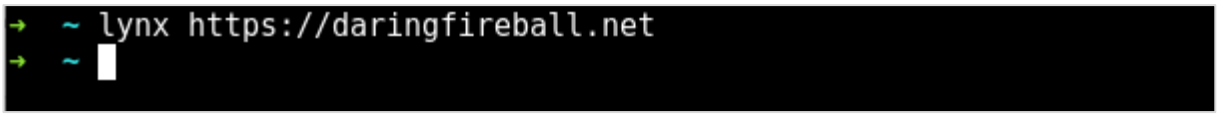
> Há uma razão pela qual navegadores de texto simples como o *Lynx* não mostram apenas o código-fonte HTML bruto. Simplesmente não é para ser um formato legível. Não lhe parece estranho escrever em um formato que não é legível? De repente, me pareceu absurdo (GRUBER, 2004 a, *online*).

A conclusão de John Gruber, em especial a referência ao navegador web *Lynx,* que é um navegador *web* em modo texto (LYNX, 2017, *online*), 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 de 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.*

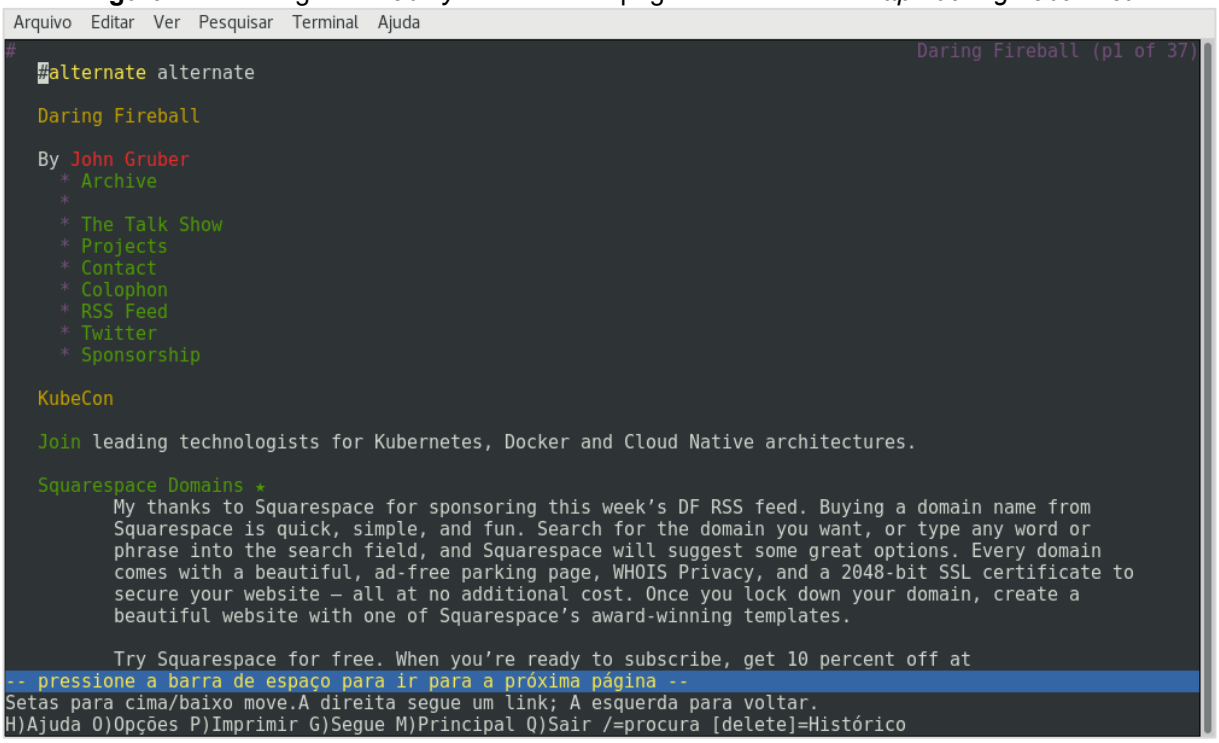
Figura 16 – Chamada ao navegador *Lynx*



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 *http://daringfireball.net*



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 Gruber (2004 a, *online*) havia concluído. Ou seja, apesar de óbvio fica claro que a marcação HTML não tem como objetivo ser apresentada à leitores.

Em face a este contexto, Gruber (2004 b, *online*) 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[7] válido[8]".

Por fim, Gruber (2017, *online*) concluí: "Um documento formatado com *Markdown* deve ser publicado como é, como texto sem formatação, sem parecer que tenha sido marcado com *tags* ou instruções de formatação".

[7] **XHTML:** *Extensible Hypertext Markup Language* (Linguagem de Marcação de Texto Extensível) é um formato que estende a marcação HTML permitindo o tratamento dos documentos como XML - *Extensible Markup Language* (Linguagem de Marcação Extensível) (W3C, 2000, *online*).
[8] **Válido:** Refere-se ao emprego correto das marcações *(tags)* HTML *e* XHTML no documento.

4.1 LINGUAGEM DE MARCAÇÃO LEVE

Em sua dissertação de Mestrado Alexandre (2017, *online*) 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 Gruber (GRUBER, 2004 b, *online*) 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 mostrada no tópico 4.2.

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 muitas linguagens de marcação leve disponíveis, algumas delas são:

- *atx:* Concebida por Aron Swartz com o intuito de simplificar a escrita de textos, que segundo ele era trazida “para o nível do computador” (SWARTZ, 2002, *online*).
- *setext:* Concebida por Feldman (1992, *online*) era destinada principalmente ao uso on-line na distribuição de publicações periódicas.
- *reStructuredText*: Muito utilizada para documentação de projetos escritos na linguagem *Python.* Um dos principais objetivos do *reStructuredText,* segundo a própria documentação, “é definir e implementar uma sintaxe de marcação para uso em *docstrings* do *Python*” (reStructuredText - Markup Syntax and Parser Component of Docutils, 2016, *online*), que se diferem dos comentários no código por sua característica de texto estruturado localizado especificamente na primeira linha da definição de classes, métodos e funções (STUMM JR, 2011, *online*).
- *Wikitexto:* Utilizada principalmente para edições de artigos na *Wikipédia*[9] (WIKIPEDIA, 2018, online).

[9] https://www.wikipedia.org

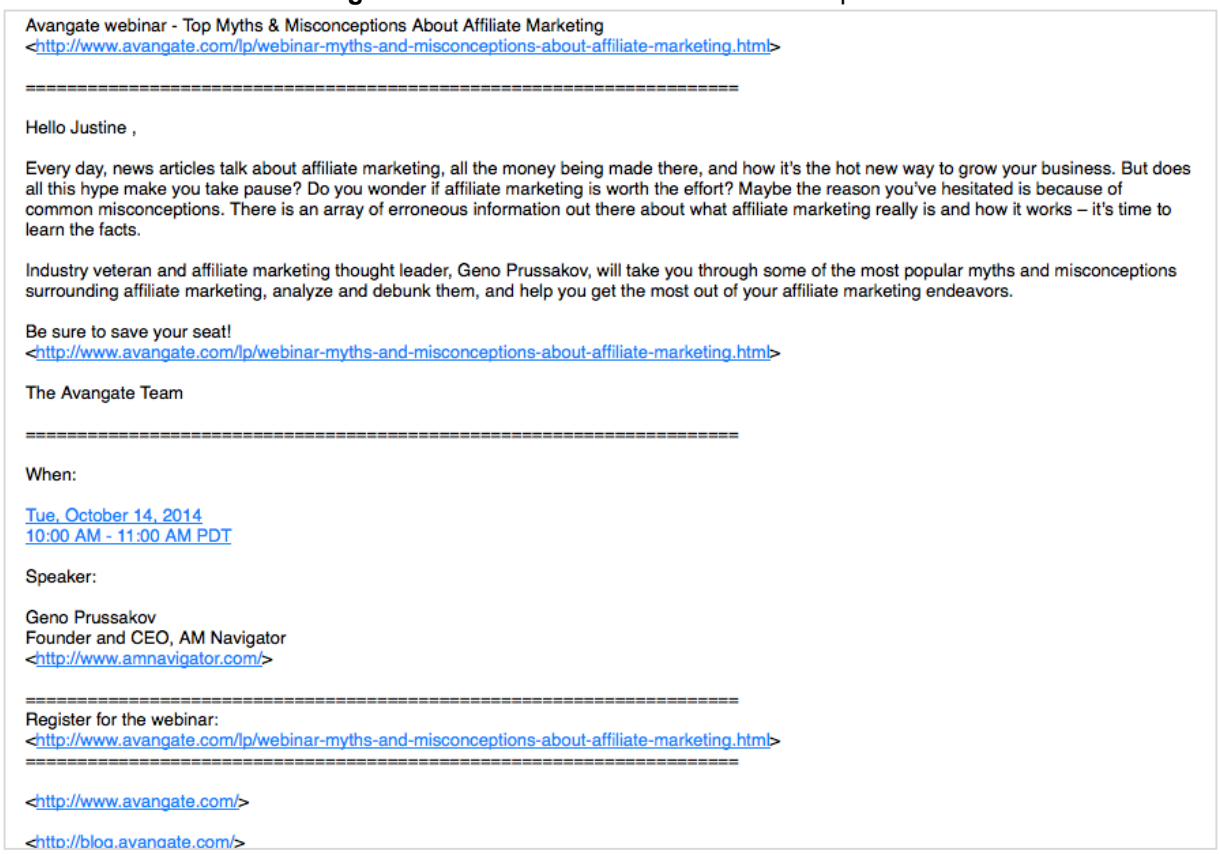
4.2 ARQUIVOS E A SINTAXE *MARKDOWN*

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.*

Para a definição da sintaxe de formatação da linguagem *Markdown*, Gruber (2017, *online*) 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.

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/>

Fonte: Smith, 2014, *online*.

O referido formato de *e-mail* de texto simples, faz parte da estratégia de envio de *e-mail marketing,* que faz uso de uma funcionalidade presente em servidores de *e-mail*, o *Multi-part MIME – Multiporpouse Internet Mail Extension* (Extensão de correio da Internet para fins múltiplos).

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 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 explanação 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, *online*) e a ferramenta de conversão *Dingus*[10], disponível *online* no *blog Daring Fireball*.

4.2.1 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.

Para obter mais de um parágrafo no texto, deve haver uma linha em branco, sem qualquer caractere imprimível, separando duas linhas com caracteres imprimíveis.

Além disso, pode-se inserir uma quebra de linha no parágrafo, separando as palavras que estão em uma linha por dois caracteres de espaços.

O Quadro 4 exibe um exemplo de emprego dos parágrafos.

[10] https://daringfireball.net/projects/markdown/dingus

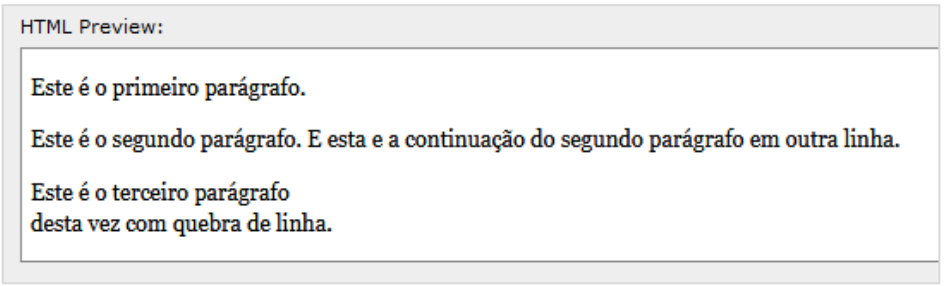
Quadro 4 – *Markdown*: Sintaxe de formatação – Parágrafos e linhas.

Texto em ***Markdown***	**Texto convertido para HTML**
`Este é o primeiro parágrafo.` `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.`	`<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>`

Fonte: Autoria própria.

A Figura 19 apresenta os três parágrafos mostrados no Quadro 4, já renderizados.

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



Fonte: Autoria própria.

4.2.2 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.

Assim, deve-se observar algumas regras, sendo que, para utilização de elementos de blocos tais como as *tags*: * , <table>, <ul> e <pre>,* é preciso separar o bloco formatado com *tags* HTML do restante do conteúdo formatado com *Markdown* inserindo uma linha em branco entre um conteúdo e outro, conforme é mostrado no Quadro 5.

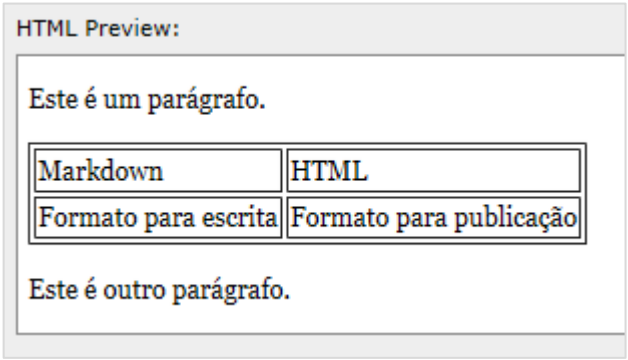
Quadro 5 – *Markdown*: Sintaxe de formatação – HTML em bloco.

Texto em ***Markdown***	Texto convertido para HTML
Este é um parágrafo. <table border="1"> <tr> <td>Markdown</td> <td>HTML</td> </tr> <tr> <td>Formato para escrita</td> <td>Formato para publicação</td> </tr> </table> Este é outro parágrafo.	Este é um parágrafo. <table border="1"> <tr> <td>Markdown</td> <td>HTML</td> </tr> <tr> <td>Formato para escrita</td> <td>Formato para publicação</td> </tr> </table> Este é outro parágrafo.

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 (*inline*) tais como as *tags*: *, <small>, <cite>* e *<del>* não existe restrição de posicionamento. Essas *tags* podem ser inseridas juntamente com linhas formatadas com sintaxe *Markdown*, conforme é mostrado no Quadro 6.

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

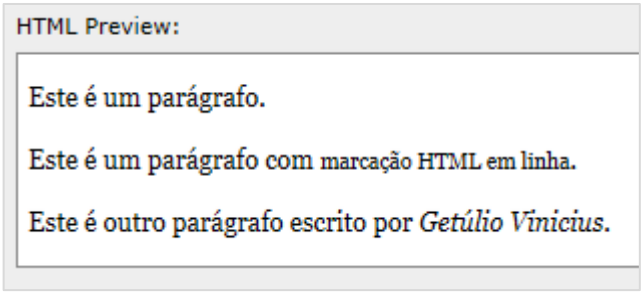
Texto em ***Markdown***	**Texto convertido para HTML**

`Este é um parágrafo com <small>marcação HTML em linha</small>.`
 	
`Este é outro parágrafo escrito por <cite>Getúlio Vinicius</cite>.`
	`<p>Este é um parágrafo com <small>marcação HTML em linha</small>.</p>`
	
	`<p>Este é outro parágrafo escrito por<cite>Getúlio Vinicius</cite>.</p>`

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 4.2.8, o *parser* irá processar normalmente a conversão.

4.2.3 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 explicito, 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 mostra o emprego dos dois tipos de ênfase com a linguagem *Markdown.*

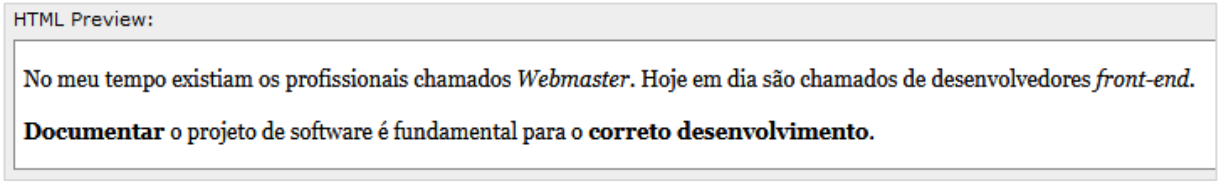
Quadro 7 – *Markdown*: Sintaxe de formatação – Ênfase.

Texto em ***Markdown***	Texto convertido para HTML
No meu tempo existiam os profissionais chamados _*Webmaster*_. Hoje em dia são chamados de desenvolvedores *front-end*. __Documentar__ o projeto de software é fundamental para o **correto desenvolvimento**.	No meu tempo existiam os profissionais chamados Webmaster. Hoje em dia são chamados de desenvolvedores front-end. Documentar o projeto de software é fundamental para o correto desenvolvimento.

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.



Fonte: Autoria própria.

O emprego de ênfase na estruturação de texto para a *web* pode causar confusões que levam ao uso incorreto do recurso, seja com 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 *,* geralmente é apresentada em formato de fonte itálico, enquanto a ênfase empregada pelo uso da *tag* HTML *,* 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 (MOZILLA, 2017, *online*).

4.2.4 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*.

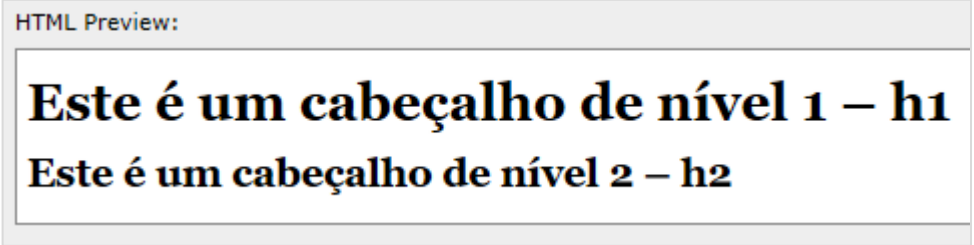
Texto em Markdown	Texto convertido para HTML

`Este é um cabeçalho de nível 1 – h1`
`==========`
 	
`Este é um cabeçalho de nível 2 – h2`
`----------`
	`<h1>Este é um cabeçalho de nível 1 – h1</h1>`
	
	`<h2>Este é um cabeçalho de nível 2 – h2</h2>`

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 em 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*.

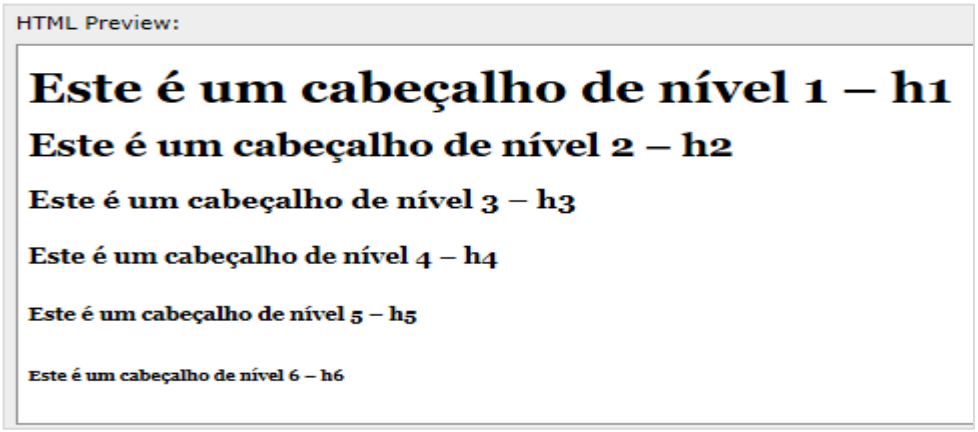
Texto em Markdown	Texto convertido para HTML

`# Este é um cabeçalho de nível 1 – h1`
 	
`## Este é um cabeçalho de nível 2 – h2`
 	
`### Este é um cabeçalho de nível 3 – h3`
 	
`#### Este é um cabeçalho de nível 4 – h4`
 	
`##### Este é um cabeçalho de nível 5 – h5`
 	
`###### Este é um cabeçalho de nível 6 – h6`
	`<h1>Este é um cabeçalho de nível 1 – h1</h1>`
	
	`<h2>Este é um cabeçalho de nível 2 – h2</h2>`
	
	`<h3>Este é um cabeçalho de nível 3 – h3</h3>`
	
	`<h4>Este é um cabeçalho de nível 4 – h4</h4>`
	
	`<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.

4.2.5 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.

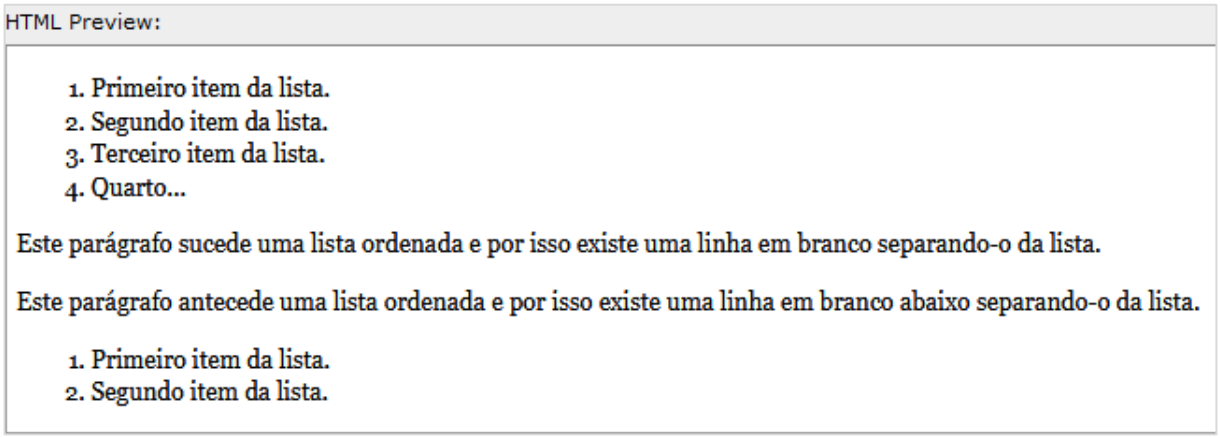
Quadro 10 – *Markdown*: Sintaxe de formatação – Listas ordenadas.

Texto em Markdown	Texto convertido para HTML
1. Primeiro item da lista. 2. Segundo item da lista. 3. Terceiro item da lista. 4. Quarto... 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.	 - Primeiro item da lista. - Segundo item da lista. - Terceiro item da lista. - 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. - Primeiro item da lista. - Segundo item da lista.

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.



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 mostrado no Quadro 11.

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

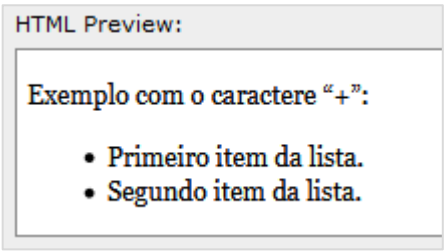
Texto em Markdown	Texto convertido para HTML
Exemplo com o caractere “+”: + Primeiro item da lista. + Segundo item da lista. Exemplo com o caractere “-”: - Primeiro item da lista. - Segundo item da lista. Exemplo com o caractere “*”: * Primeiro item da lista. * Segundo item da lista. Exemplo com os caracteres mesclados: + Primeiro item da lista. - Segundo item da lista. * Terceiro item da lista.	Exemplo com o caractere “+”: - Primeiro item da lista. - Segundo item da lista. Exemplo com o caractere “-”: - Primeiro item da lista. - Segundo item da lista. Exemplo com o caractere “*”: - Primeiro item da lista. - Segundo item da lista. Exemplo com os caracteres mesclados: - Primeiro item da lista. - Segundo item da lista.

	Terceiro item da lista.

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 mostrado 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.
 	
Lista não ordenada:
 	
* Primeiro item da lista de nível 1.
  * Primeiro item da lista de nível 2.
	Lista ordenada:
	
	   - Primeiro item da lista de nível 1.
	   - Segundo 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 2.
* Segundo item da lista de nível 1.

Listas ordenadas e não ordenadas:

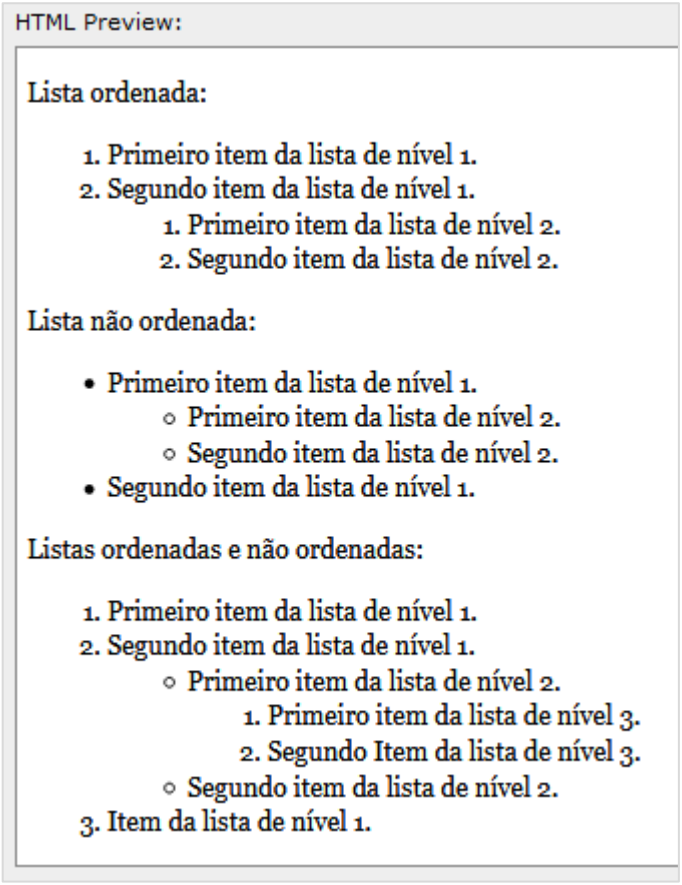
1. Primeiro item da lista de nível 1.
2. Segundo item da lista de nível 1.
  * Primeiro item da lista de nível 2.
    1. Primeiro item da lista de nível
3.
    2. Segundo Item da lista de nível 3.
  * Segundo item da lista de nível 2.
3. Item da lista de nível 1.
```

```
        </ol>
    </li>
</ol>
<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>
<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>
                    <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 mostrado 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.

4.2.6 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 utilizados 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 é mostrado no Quadro 13.

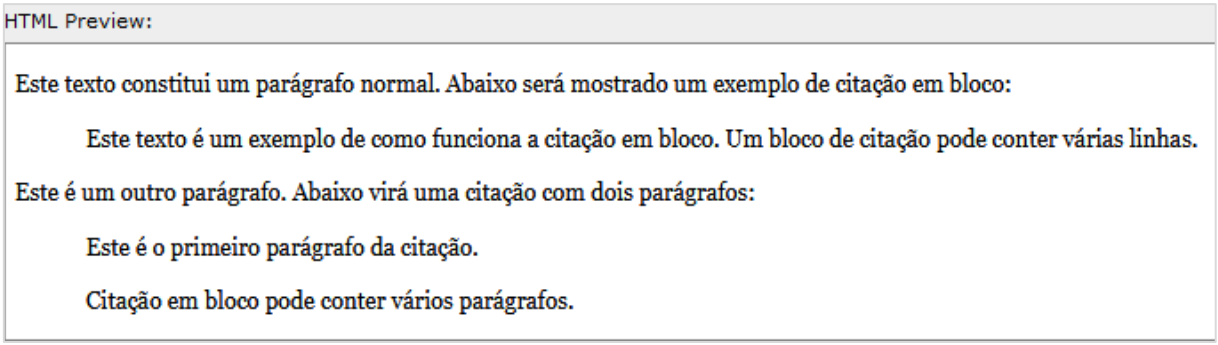
Quadro 13 – *Markdown*: Sintaxe de formatação – Citação em Bloco.

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

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.



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 mostra um bloco de citação composto por mais elementos da sintaxe *Markdown*.

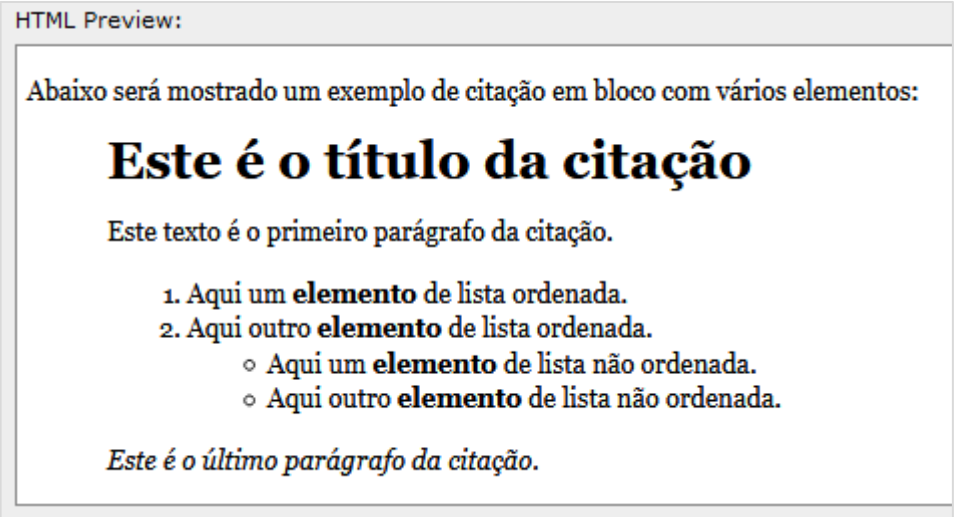
Quadro 14 – *Markdown*: Sintaxe de formatação – Citação em Bloco composta por mais elementos.

Texto em Markdown	Texto convertido para HTML
Abaixo será mostrado um exemplo de citação em bloco com vários elementos: > # Este é o título da citação > Este texto é o primeiro parágrafo da citação. > 1. Aqui um **elemento** de lista ordenada. > 2. Aqui outro **elemento** de lista ordenada. > + Aqui um **elemento** de lista não ordenada. > + Aqui outro **elemento** de lista não ordenada. > _Este é o último parágrafo da citação._	Abaixo será mostrado um exemplo de citação em bloco com vários elementos: <blockquote> Este é o título da citação Este texto é o primeiro parágrafo da citação. - Aqui um elemento de lista ordenada. - Aqui outro elemento de lista ordenada. - Aqui um elemento de lista não ordenada. - Aqui outro elemento de lista não ordenada. Este é o último parágrafo da citação. </blockquote>

Fonte: Autoria própria.

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

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



Fonte: Autoria própria.

4.2.7 *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 ancora e, imediatamente após envolver o endereço (caminho para o arquivo) por parênteses "()", sendo que:

- O texto ancora 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* (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 exposição acerca do uso de *links inline* com a linguagem Markdown.

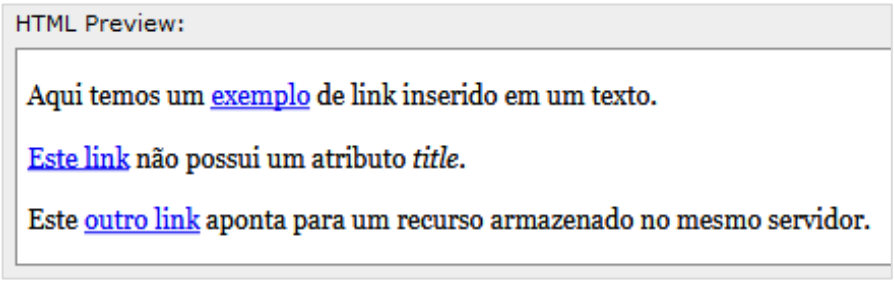
Quadro 15 – *Markdown*: Sintaxe de formatação – *Links inline*.

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

Fonte: Autoria própria.

A Figura 30 exibe o exemplo mostrado no Quadro 15 devidamente renderizado.

Figura 30 – *Markdown*: Saída HTML renderizada – *Links inline*.



Fonte: Autoria própria.

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 mostra o modo de uso de links por meio de referência na linguagem *Markdown.*

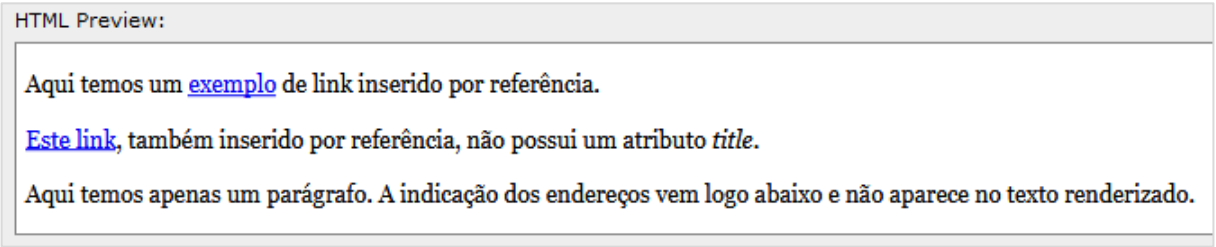
Quadro 16 – *Markdown*: Sintaxe de formatação – *Links* por referência.

Texto em Markdown	Texto convertido para HTML
Aqui temos um [exemplo][ex1] de link inserido por referência. [Este link][ex2], também inserido por referência, não possui um atributo _title_. Aqui temos apenas um parágrafo. A indicação dos endereços vem logo abaixo e não aparece no texto renderizado. [ex1]: http://exemplo.com "Título de exemplo" [ex2]: /pagina-de-exemplo/	Aqui temos um exemplo de link inserido por referência. Este link, também inserido por referência, não possui um atributo title. Aqui temos apenas um parágrafo. A indicação dos endereços vem logo abaixo e não aparece no texto renderizado.

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.



Fonte: Autoria própria.

4.2.8 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.*

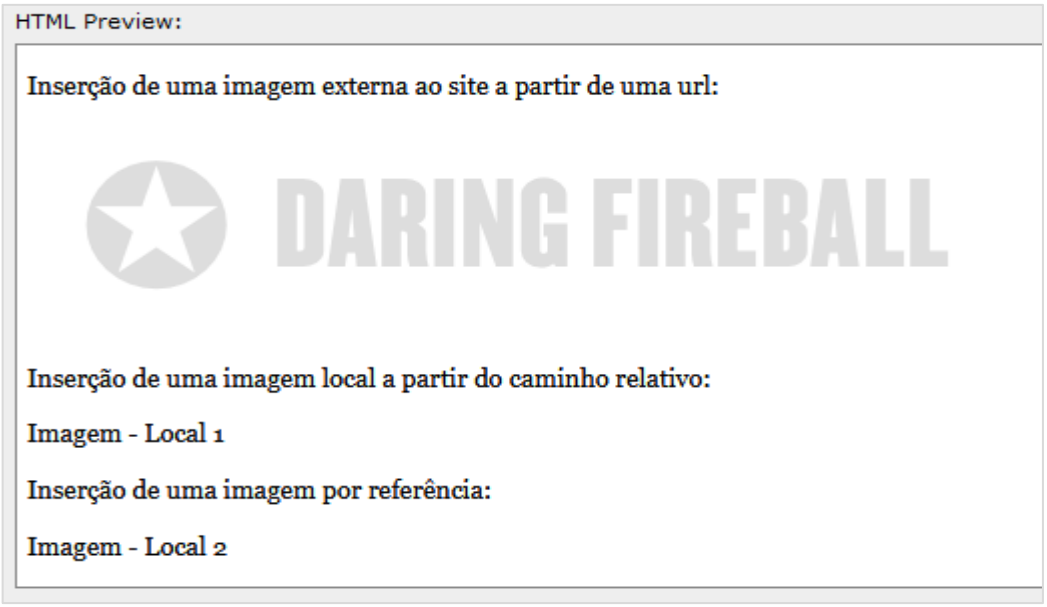
Quadro 17 – *Markdown*: Sintaxe de formatação – Imagens.

Texto em Markdown	Texto convertido para HTML
Inserção de uma imagem externa ao site a partir de uma url: ![Daring Fireball – Logo](https://daringfireball.net/graphics/logos/ "Logo do site Daring Fireball") Inserção de uma imagem local a partir do caminho relativo: ![Imagem – Local 1](/imagens/logo1 "Título 1") Inserção de uma imagem por referência: ![Imagem – Local 2][2] [2]: /imagens/logo2 "Título 2"	Inserção de uma imagem externa ao site a partir de uma url: Inserção de uma imagem local a partir do caminho relativo: Inserção de uma imagem por referência:

Fonte: Autoria própria.

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

Figura 32 – *Markdown*: Saída 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.

4.2.9 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 Mozilla (2017, *online*), “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.

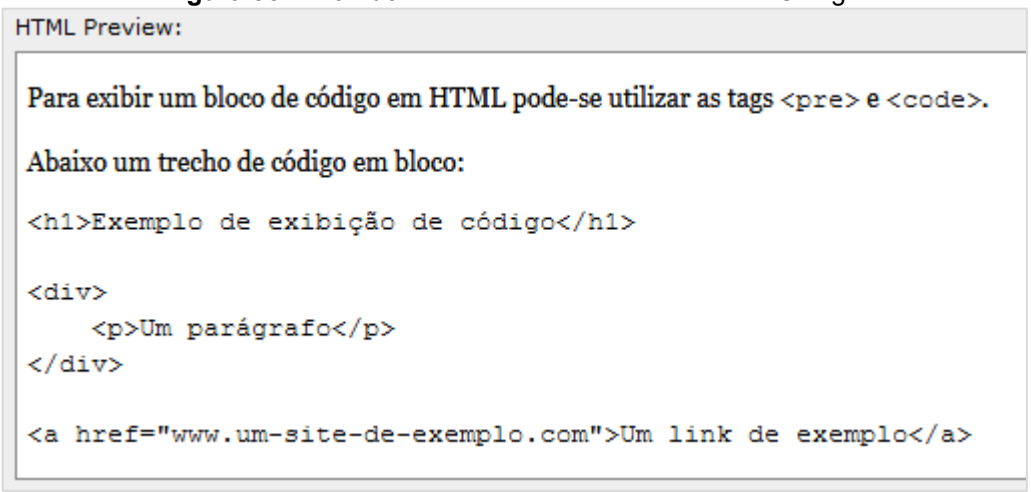
Texto em Markdown	Texto convertido para HTML

Para exibir um bloco de código em HTML pode-se utilizar as tags `<pre>` e `<code>`.
 	
Abaixo um trecho de código em bloco:
 	
 Exemplo de exibição de código
 	
 Um parágrafo
 	
Um link de exemplo
	Para exibir um bloco de código em HTML pode-se utilizar as tags <code><pre></code> e <code><code></code>.
	
	 Abaixo um trecho de código em bloco:
	
	<pre><code> Exemplo de exibição de código
	
	 Um parágrafo
	
	Um link de exemplo
	</code></pre>

Fonte: Autoria própria.

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

Figura 33 – *Markdown*: Saída HTML renderizada – Código.



Fonte: Autoria própria.

4.2.10 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 + \` "Apóstrofe" + * "Asterisco" + _ "Sublinhado" + \{\} "Chaves" + \[\] "Colchetes" + \(\) "Parênteses" + \# "Tralha" + \+ "Sinal de adição" + \- "Hífen" + \. "ponto" + \! "Ponto de exclamação"	 \ "Contra barra ` "Apóstrofe" * "Asterisco" _ "Sublinhado" {} "Chaves" [] "Colchetes" () "Parênteses" # "Tralha" + "Sinal de adição" - "Hífen" . "ponto" ! "Ponto de exclamação"

Fonte: Autoria própria.

4.3 *PARSERS*

Há uma variedade de *pasers*, *software* para conversão do texto escrito em *Markdown* para HTML, disponíveis para uso. A seguir, será feita uma análise acerca dos aspectos históricos e da evolução dos mesmos.

4.3.1 *Paser* 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 *et al.*, 2017,

online), podendo ser utilizado em conjunto com aplicativos para publicação de conteúdo web ou 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.

```
➜  Markdown_1.0.1 cat code.md
# MARKDOWN

Para exibir um bloco de código em HTML pode-se utilizar as tags `<pre>` e
 `<code>`.

Abaixo um trecho de código em bloco:

    <h1>Exemplo de exibição de código</h1>

    <div>
        <p>Um parágrafo</p>
    </div>

    <a href="www.um-site-de-exemplo.com">Um link de exemplo</a>
➜  Markdown_1.0.1 perl Markdown.pl --html4tags code.md
<h1>MARKDOWN</h1>

<p>Para exibir um bloco de código em HTML pode-se utilizar as tags <code>
&lt;pre&gt;</code> e <code>&lt;code&gt;</code>.</p>

<p>Abaixo um trecho de código em bloco:</p>

<pre><code>&lt;h1&gt;Exemplo de exibição de código&lt;/h1&gt;

&lt;div&gt;
    &lt;p&gt;Um parágrafo&lt;/p&gt;
&lt;/div&gt;

&lt;a href="www.um-site-de-exemplo.com"&gt;Um link de exemplo&lt;/a&gt;
</code></pre>
➜  Markdown_1.0.1
```

Fonte: Autoria própria.

Na imagem, nota-se que 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.*

Este *parser* 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, *online*).

4.3.2 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 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"* (formato de arquivo padrão do editor de textos *Microsoft Word*) *e ".txt"* (MACFARLANE, 2017, *online*).

A Figura 35 mostra o procedimento para conversão de um arquivo em formato *Markdown* para o formato *"docx"* utilizando o *Pandoc*.

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

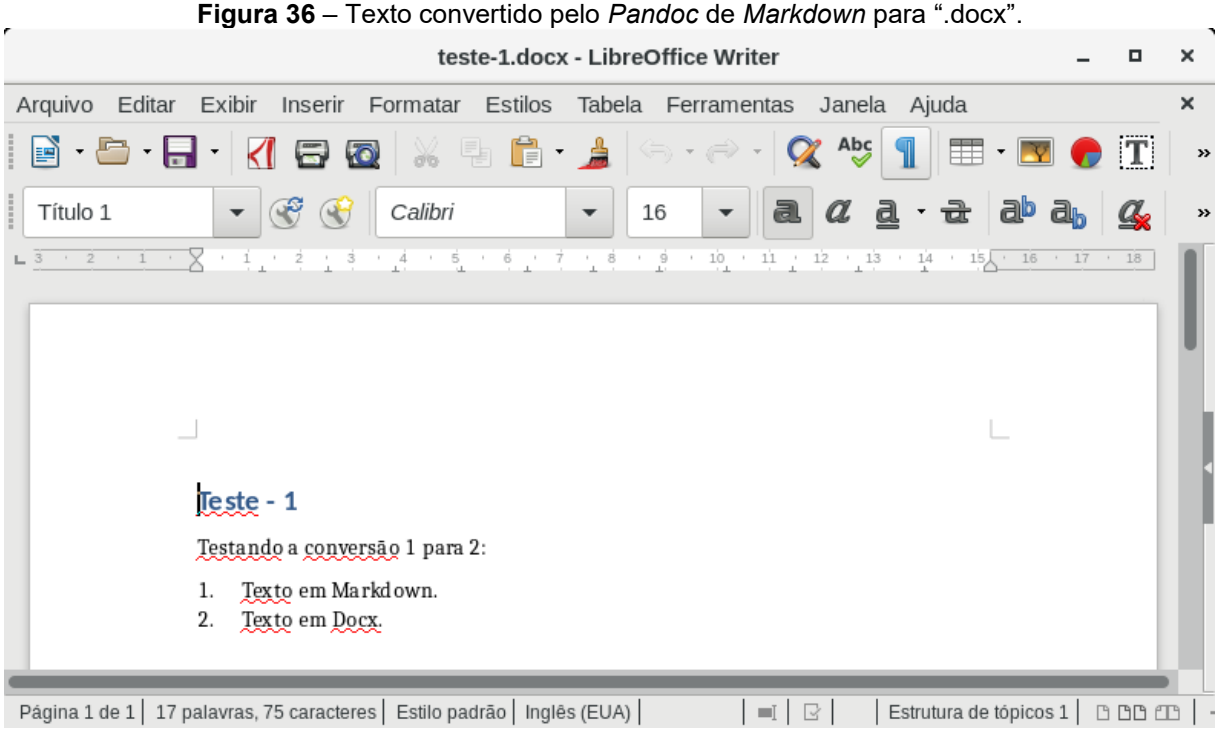
Fonte: Autoria própria.

No procedimento, realizado através do terminal do SO *Debian GNU/Linux*, inicialmente é listado o conteúdo do diretório corrente, inserindo 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 *parses* 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 – PHP hypertext processor* (PHP processador de hipertexto), 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, *online*). 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 (Interface de Programação de Aplicativos), a integração de diversas extensões que ampliam os recursos sintáticos especificados por John Gruber, (THE PYTHON-MARKDOWN PROJECT, 2017, *online*). Os primeiros autores do projeto são: Waylan Limberg, Yuri Takteyev, Manfed 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, *online*).

4.3.3 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

`# 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 |`
	`<h1>Linguagens de programação</h1>`
	`<table>`
	`<thead>`
	`<tr>`
	`<th align="center">ID</th>`
	`<th align="left">Linguagem</th>`
	`<th align="right">Site</th>`
	`</tr>`
	`</thead>`
	`<tbody>`
	`<tr>`
	`<td align="center">1</td>`
	`<td align="left">PHP</td>`
	`<td align="right">http://www.php.net</td>`
	`</tr>`

```
        <tr>
            <td align="center">2</td>
            <td align="left">Python</td>
            <td
align="right">https://www.python.org</td>
        </tr>
        <tr>
            <td align="center">4</td>
            <td align="left">Ruby</td>
            <td
align="right">https://www.ruby-
lang.org</td>
        </tr>
    </tbody>
</table>
```

Fonte: Autoria própria.

A Figura 37 exibe o resultado renderizado a partir da formatação feita no Quadro 20, sendo que, a conversão para HTML e renderização da marcação HTML, foram realizadas utilizando a aplicação *online Babelmark 2*[11] em conjunto com o *parser PHP Markdown Extra*.

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.

4.4 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 livre é: Requisição de Mudanças número 7763).

Este tipo de documento define um padrão para uso de determinada tecnologia e é emitido pela IETF – *Internet Engineering Task Force*, um órgão constituído por uma comunidade internacional que trabalha em função da elaboração de padrões

[11] http://johnmacfarlane.net/babelmark2/

abertos para a internet, que são obtidos através de procedimentos que também são abertos, (INTERNET ENGINEERING TASK FORCE (IETF), 2018, *online*).

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 a, 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 b, 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 4.3.3 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 retorna 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, *online*).

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

Tal especificação foi adotada pelo *GitHub*, que faz uso intenso da sintaxe *Markdown* porém foi chamada de GFM – *GitHub Flavored Markdown* (*Markdown* com sabor GitHub).

4.5 APLICAÇÃO DA LINGUAGEM *MARKDOWN*

Segundo Gruber (2017, *online*) “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.

4.5.1 Aplicações de uso geral

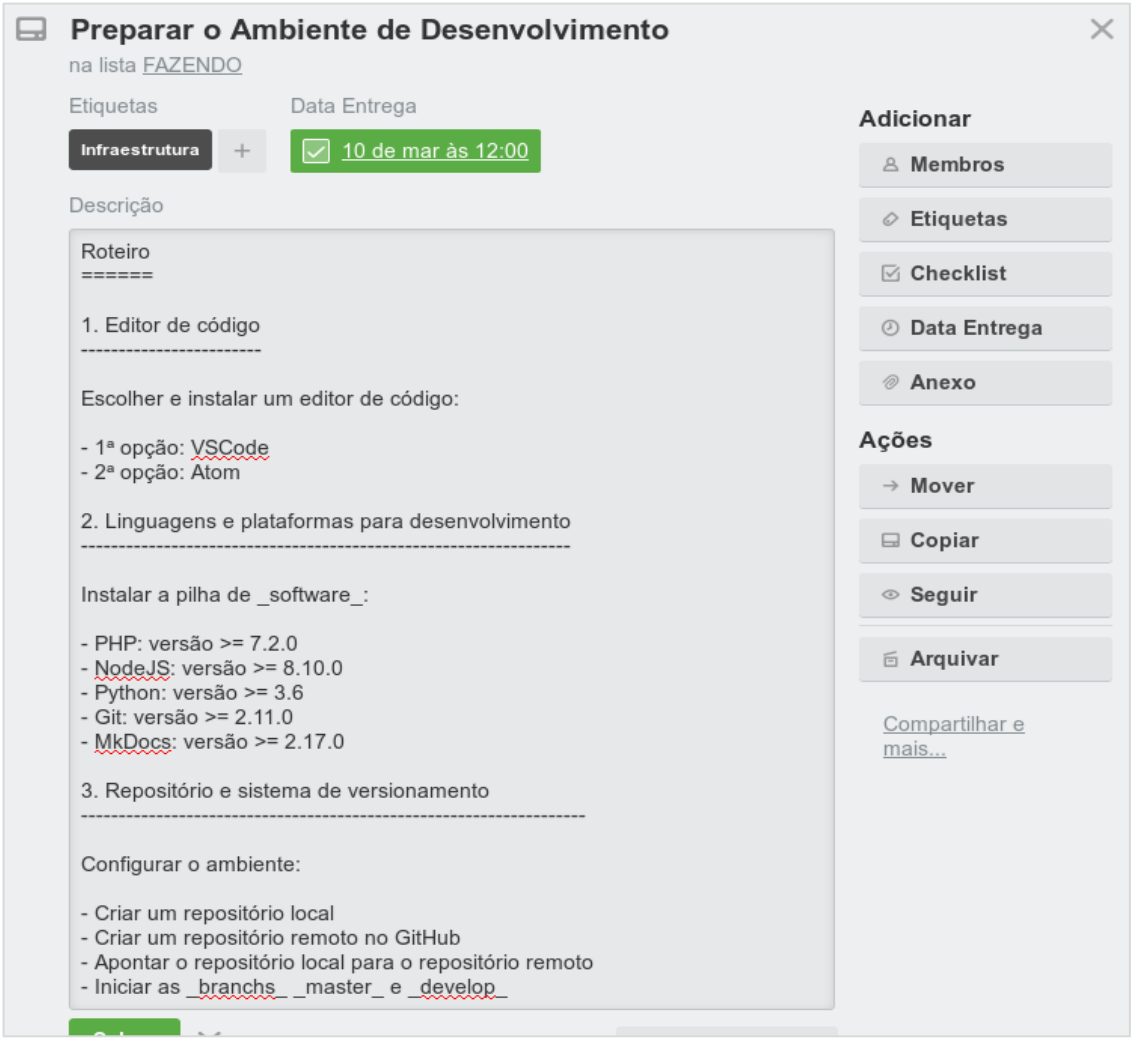
Um exemplo da aplicação de *Markdown* são os cartões do aplicativo de gestão de projetos e tarefas *Trello*[12]. Nesse aplicativo, que adota a dinâmica da metodologia *Kanban*, tem-se os elementos chamados cartões que representam as tarefas e são organizados em listas dentro de um quadro, sendo que as 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*.

[12] https://trello.com

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

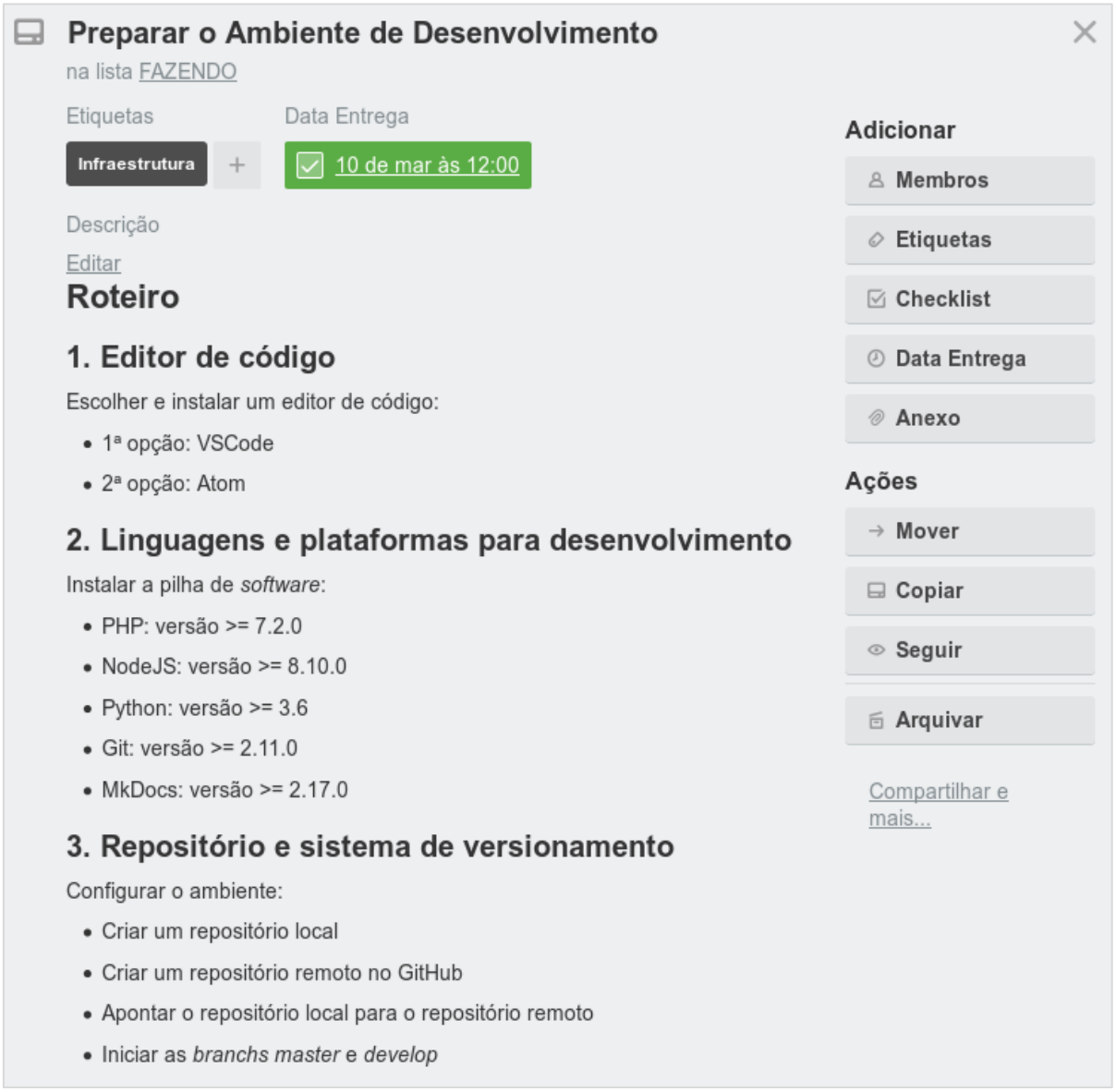


Fonte: Autoria própria.

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.

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*.

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



Fonte: Autoria própria.

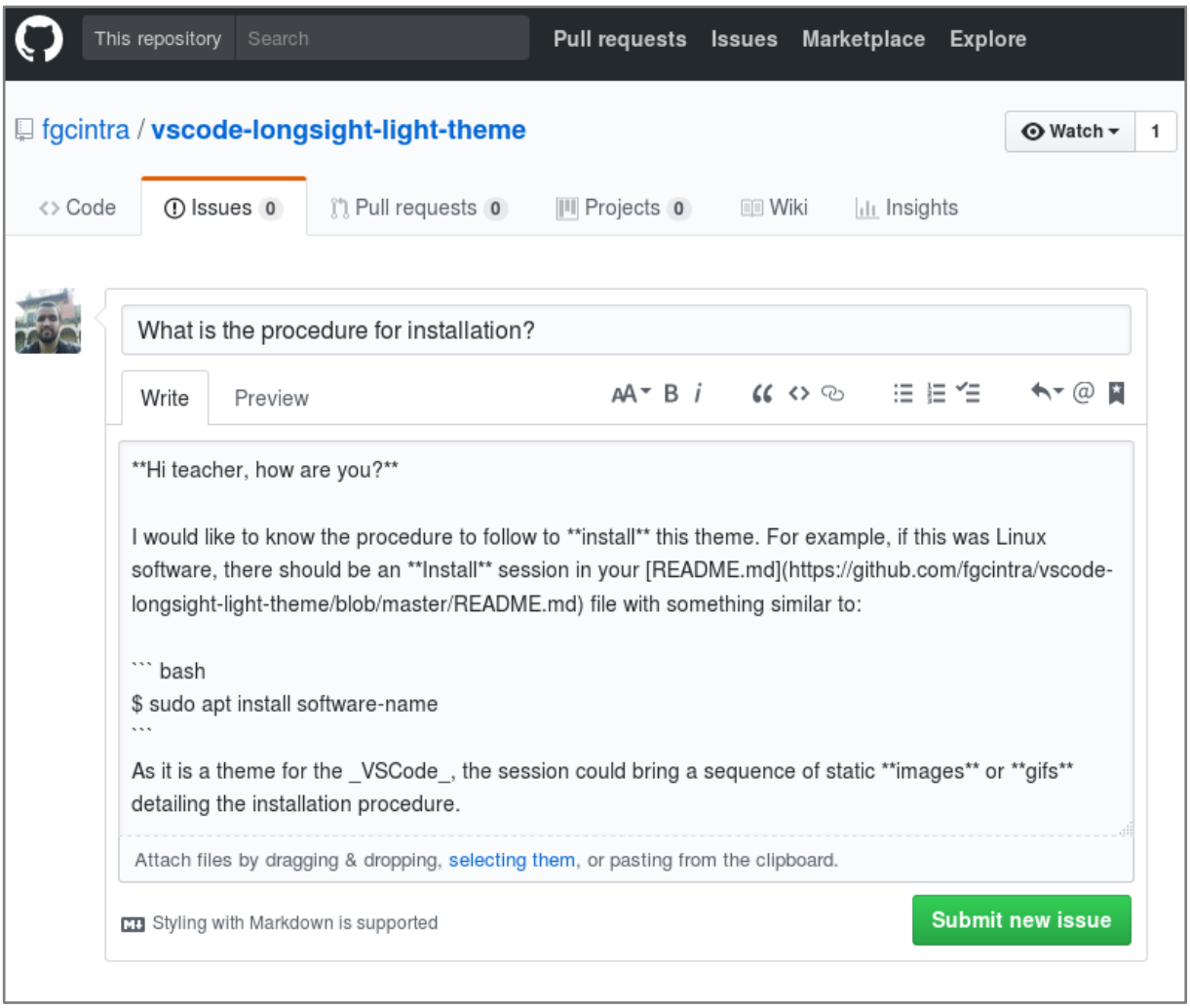
No processo de abertura de *issues* (diálogos para reportagem de problemas no código ou esclarecimento de dúvidas em um projeto), a sintaxe *Markdown* colabora substancialmente para formulação do questionamento, pois, conforme é mostrado na Figura 40, é possível destacar palavras no texto, criar um *link* para um arquivo do projeto, citar trechos de código, entre outras facilidades.

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 com o fundo do texto na cor cinza.

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

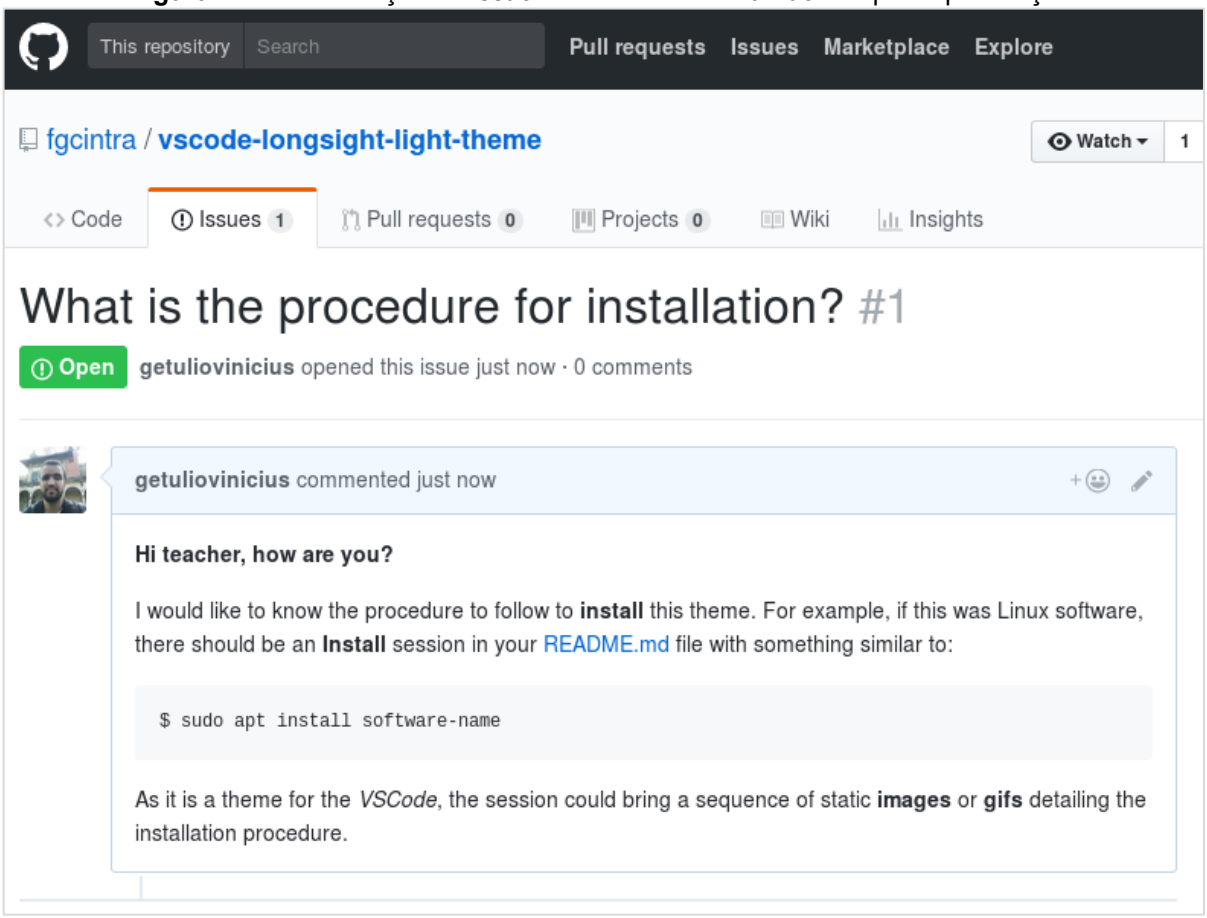


Fonte: Autoria própria.

Além das *issues*, o *Github* permite a criação de um arquivo de texto formatado com *Markdown*, que em inglês é chamado de *README.md* (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.

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



Fonte: Autoria própria.

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*.

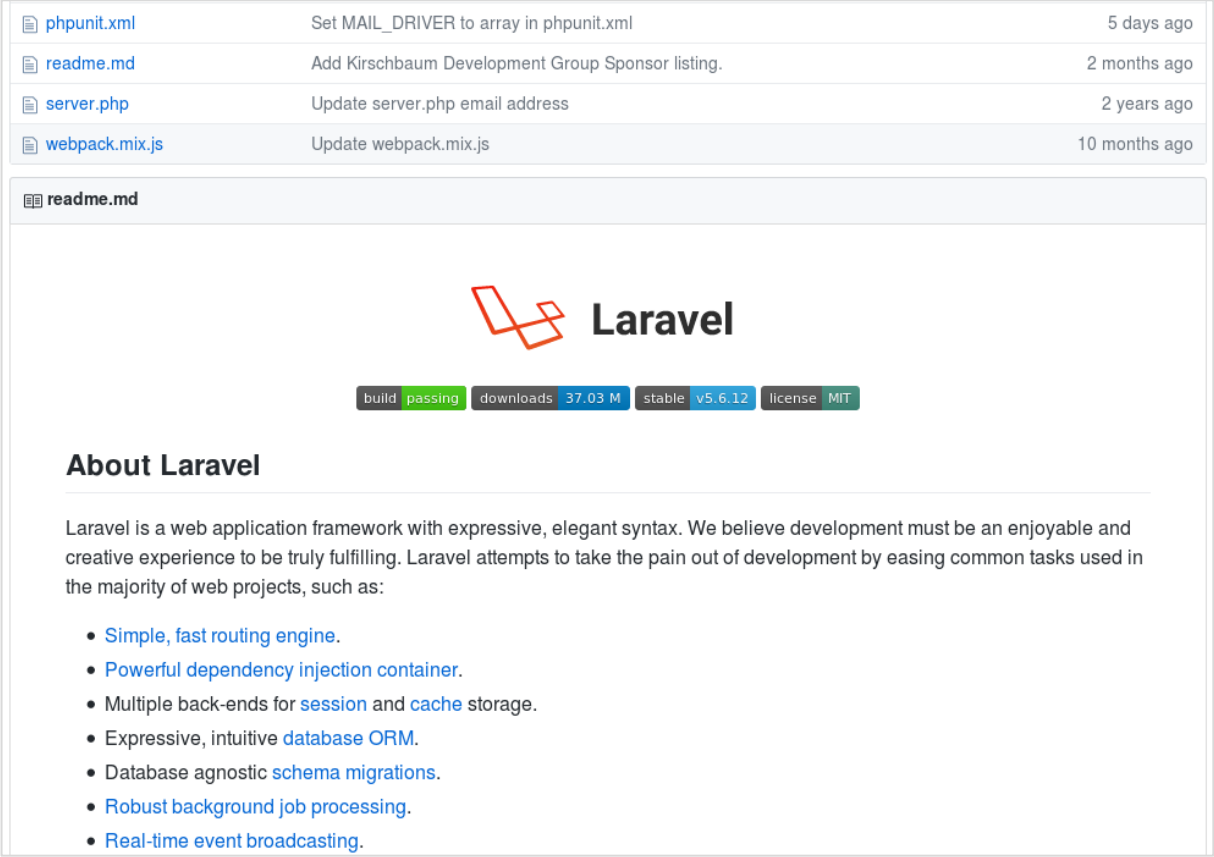
A aplicação da linguagem *Markdown* para construir os arquivos de instruções dos *softwares* disponibilizados em repositórios do *GitHub* e em outros sites de armazenamento e compartilhamento de código, como o *Gitlab*[13] e o *Bitbucket*[14], talvez seja o primeiro contato de muitos desenvolvedores 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

[13] https://about.gitlab.com/
[14] https://bitbucket.org/

exemplo a área de *issues,* já citada, e a área de *wiki* (um local para publicação de arquivos como mais detalhes do *software*).

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

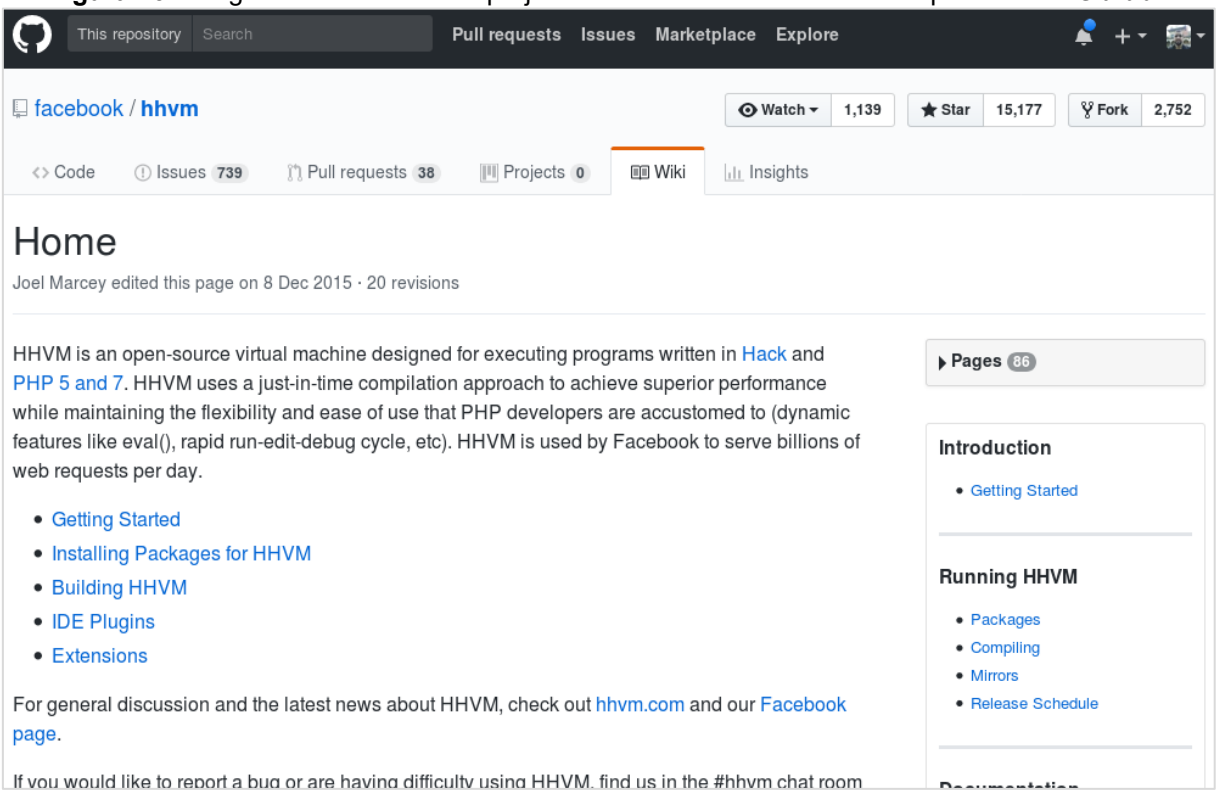


Fonte: Autoria própria.

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.

4.5.2 Geradores de sites estáticos

Por conseguinte, após toda a explicação sobre o 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.

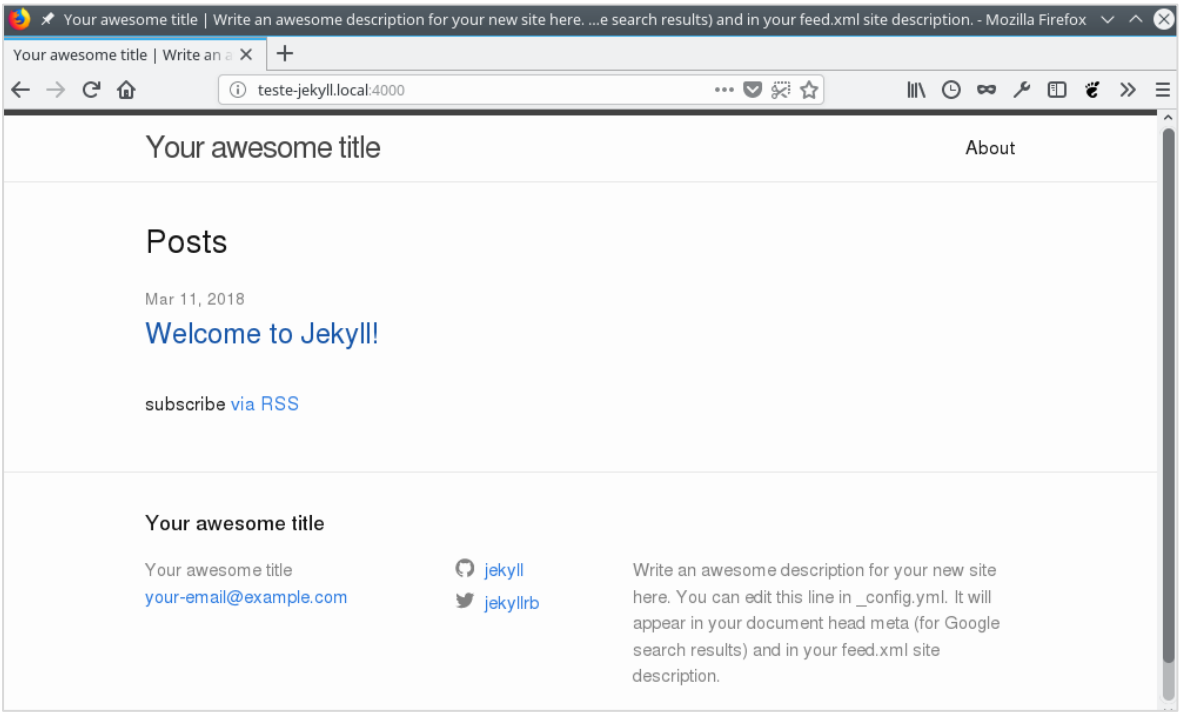
Assim, existem aplicações capazes de converter arquivos escritos em linguagem *Markdown* para sites estáticos compostos por arquivos HTML e folhas de estilo CSS – *Cascading Style Sheet* (Folha de Estilo em Cascata), com a possibilidade de utilização de *templates*, que neste caso, representam modelos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

O site *StaticGen* (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, *online*).

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 traz da renderização das páginas no *Github Pages*[15], um serviço oferecido pelo *GitHub* para servir sites estáticos que apresentam os softwares armazenados em repositórios do próprio *GitHub* (JEKYLL, 2018, *online*).

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

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



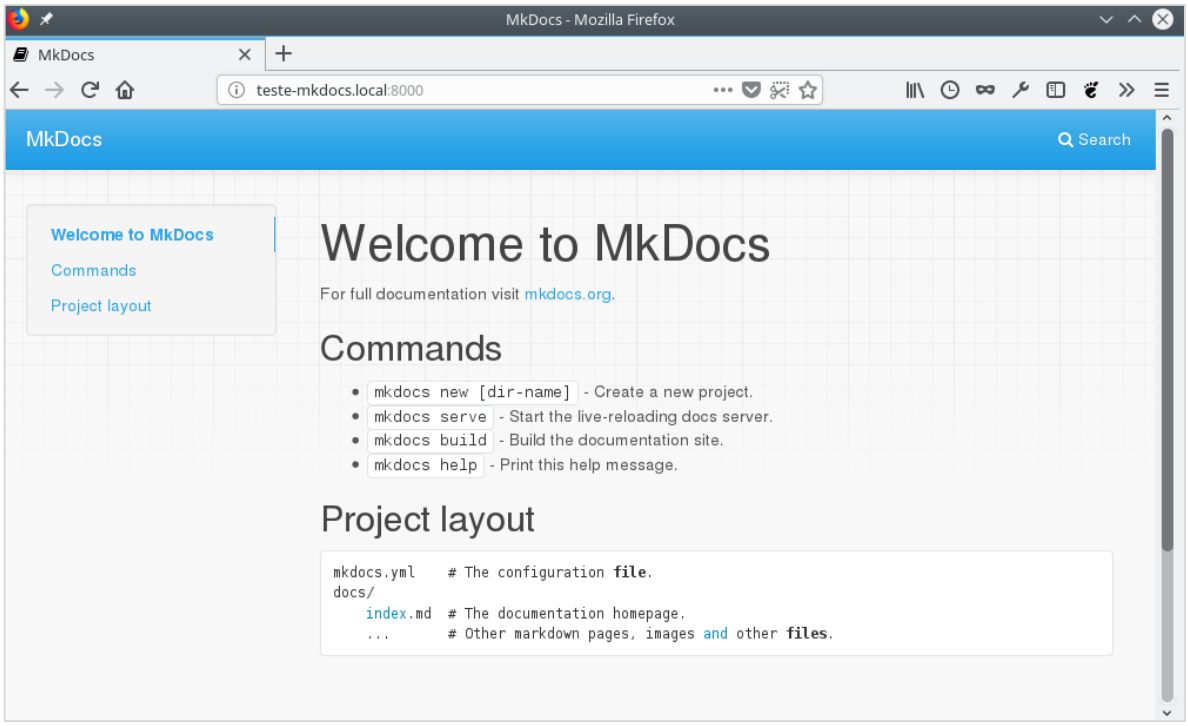
Fonte: Autoria própria.

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, *online*). 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*.

[15] https://pages.github.com

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

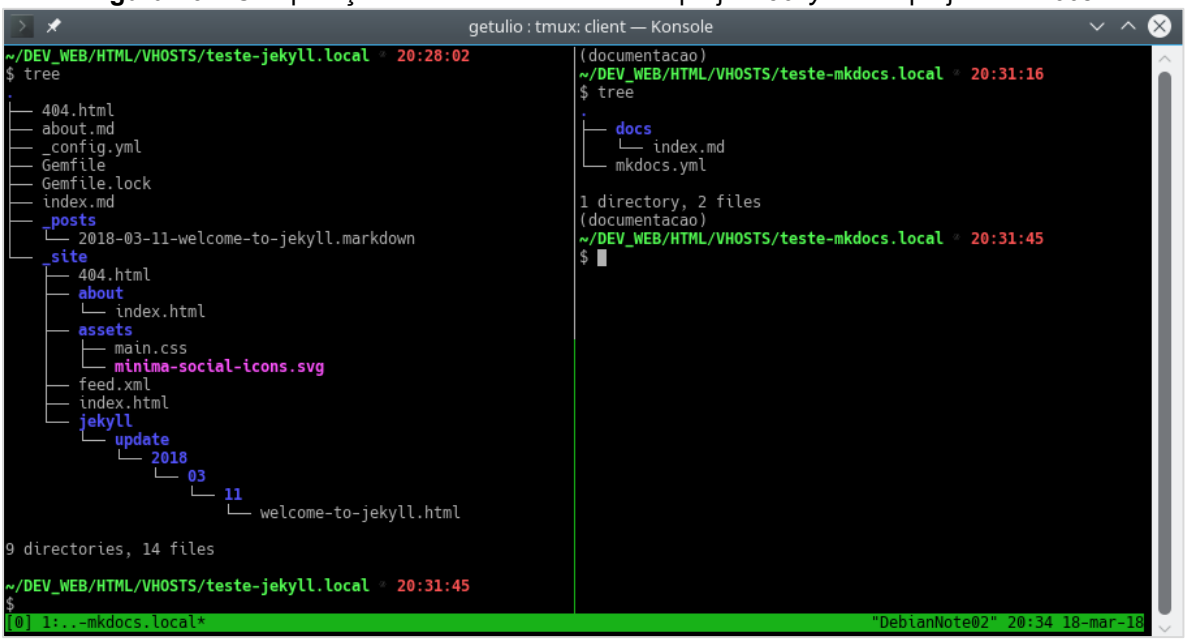


Fonte: Autoria própria.

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.

4.5.2.1 *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 *MkDocs*
1. Fazer o *download* do instalador do gerenciador de pacotes "*pip*". *$ wget https://bootstrap.pypa.io/get-pip.py* 2. Instalar o gerenciador de pacotes "*pip*". *$ sudo python get-pip.py* 3. Instalar o pacote "*virtualenv*". *$ sudo pip install virtualenv* 4. Instalar o pacote "*virtualenvwrapper*". *$ sudo pip install virtualenvwrapper* 5. Editar o arquivo "*.bashrc*" na localizado no diretório "*home*" do usuário atual e adicionando as linhas abaixo no final do arquivo. *# 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 paramento “*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* (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 *mkdocs.yml*

```
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/
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
14. # Conta do Google Analytics
15. google_analytics: ['UA-YXYXYXYX-YX', 'auto']
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: "_"
28. # Tema
29. theme:
30.     name: readthedocs
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
```

Fonte: Autoria própria.

No arquivo de configurações exposto no Quadro 22, os números à esquerda representam a contagem de linhas, e, 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* (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.

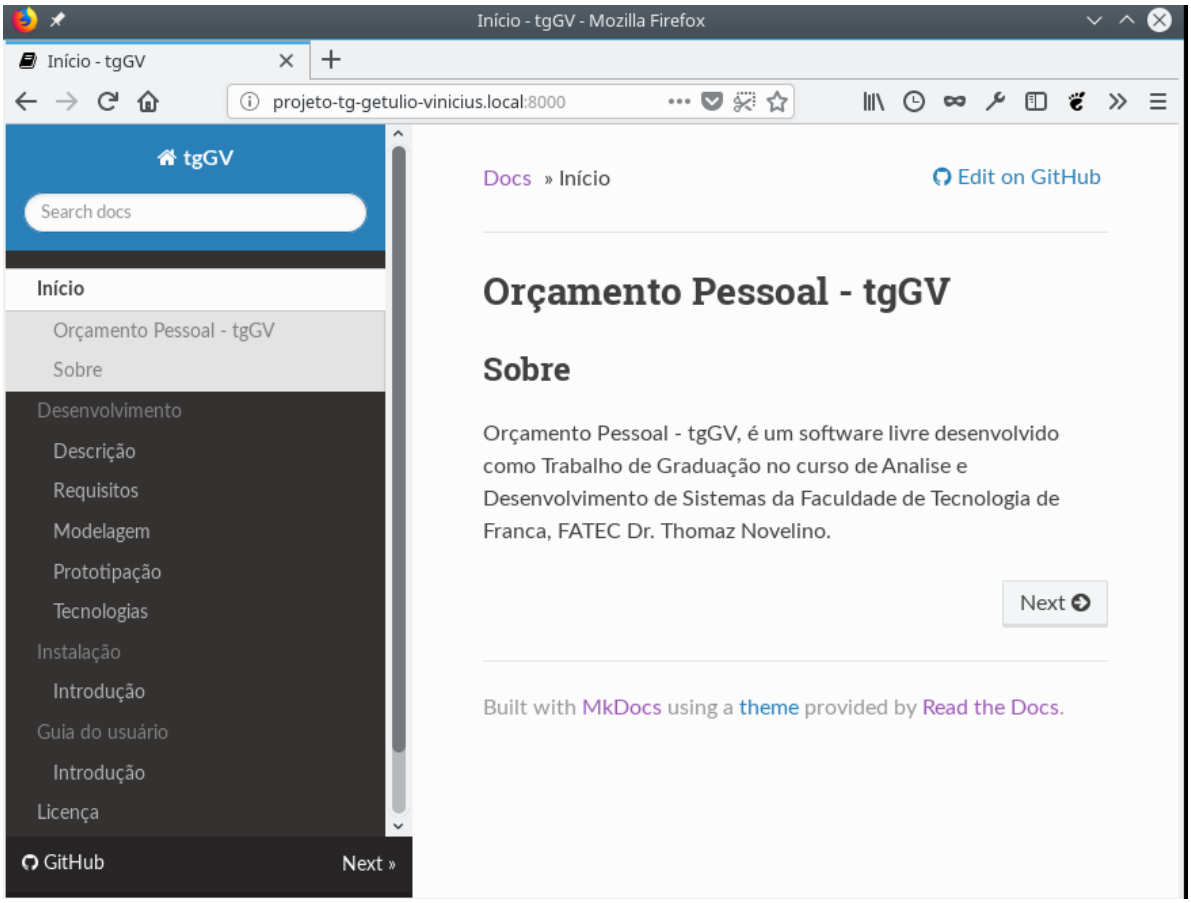
O primeiro parâmetro da sessão Mapa do Site, define o subdiretório do projeto onde serão armazenados os arquivos *Markdown*, que possuem a extensão ".md". A estrutura de diretórios e arquivos definida no exemplo constante no Quadro 22 pode ser observada na Figura 47.

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

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, onde foi utilizada a linguagem *Markdown* e o gerador de sites estáticos *MkDocs* para produzir, gerenciar e apresentar o conteúdo documental.

5 APLICAÇÃO PRÁTICA – PROJETO

Como estudo de caso para o trabalho sobre documentação de *software* elaborado até momento, foi iniciado o desenvolvimento de um projeto de aplicação para finanças e orçamento pessoal.

A seguir serão relatados os processos de Concepção, Planejamento, Execução e Fechamento do projeto, etapa por etapa, sendo que o objetivo é a entrega de uma versão 1.0.0 da aplicação para finanças e orçamento pessoal.

5.1 CONCEPÇÃO

Neste ponto será realizada uma explicação sobre a etapa de concepção do projeto de desenvolvimento da aplicação, acompanhada por uma explanação em linhas gerais acerca da abordagem tecnológica.

O projeto foi concebido a partir da constatação de uma necessidade comum a muitas pessoas, observada a partir de diálogos, 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 *smatphone* e *tablet*. Alguns deles são: Guia Bolso, Minhas Economias, *Mobills* 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, para o controle de finanças pessoais, existem características do universo contábil como, por exemplo, as partidas dobradas (método para lançamento de valores que prevê, sempre, ao menos uma conta de origem e uma conta de destino) 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 desenvolvida a referida aplicação, e que ela receberia o nome de “tgGV”.

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 seus recursos, tais como: aplicação *Web*, aplicação *Desktop* – para diversos sistemas operacionais – e aplicativo *mobile* – para *smartphones*.

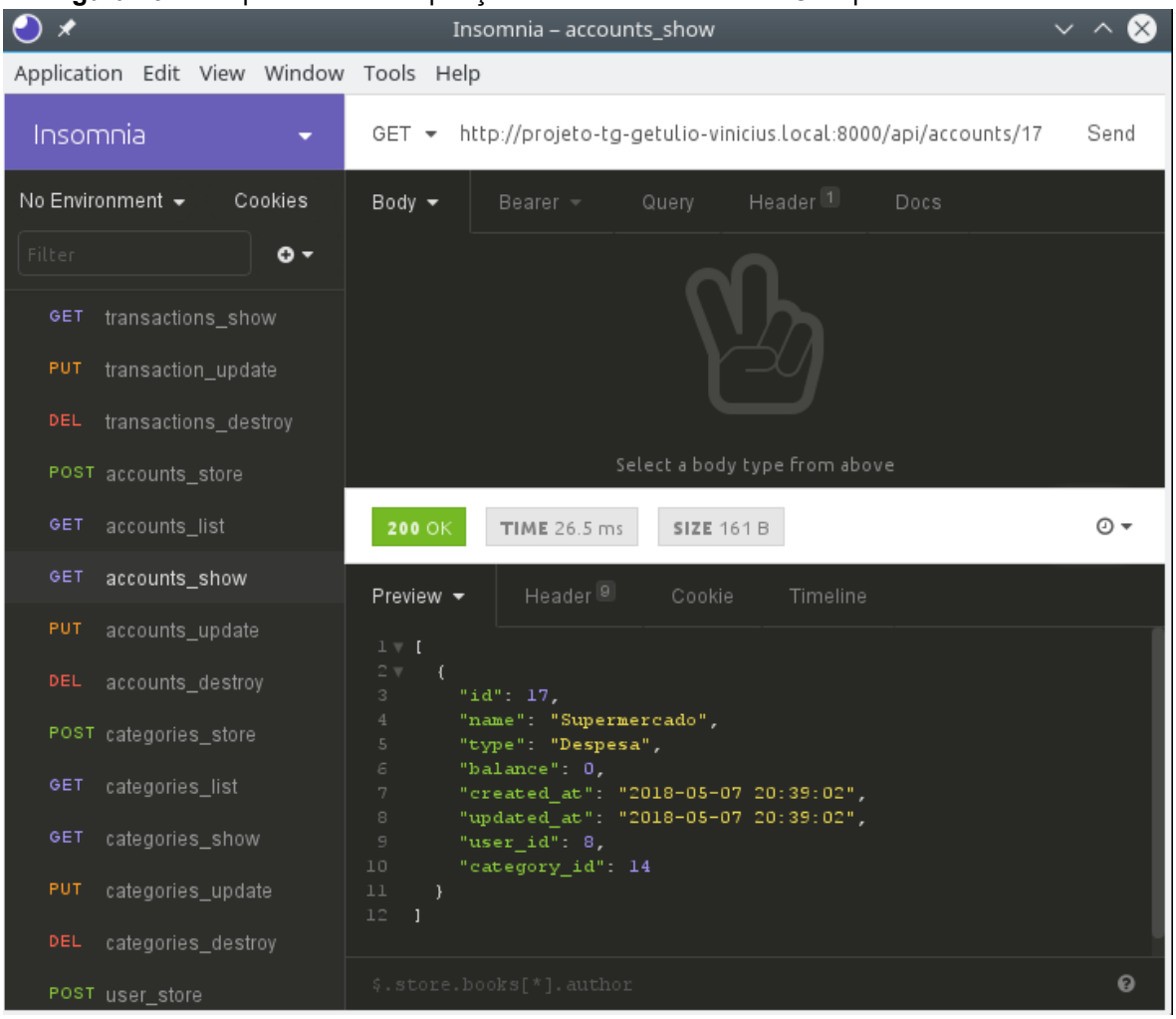
Deste modo também é possível manter as partes da aplicação desacopladas e possivelmente integrar o código com outros sistemas mais facilmente, pois em se tratando de REST, que é um acrônimo para *Representational State Transfer* (Transferência de Estado Representativo), todos os dados são considerados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

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

Assim, através do protocolo HTTP a aplicação web requisita o recurso a uma URI da API, por exemplo: http://dominio-da-api.com.br/api/accounts/17 (o número 17 no final do endereço representa a identificação da conta) e, após processar a requisição, caso o recurso tenha sido localizado, a API retorna os dados em formato de arquivo JSON – *Javascript Object Notation* (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 reposta 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 um determinado dado; e completando a lista dos verbos mais utilizados existe o *DELETE,* cujo objetivo é requisitar a remoção de dados. Os outros verbos HTTP são: *PATCH*, *HEAD, OPTIONS, TRACE e CONNECT* (LOPEZ, 2016, p. 315 - 322).

Quanto a segurança da aplicação, o acesso aos recursos da API se dá mediante o envio de *tokens* emitidos por uma implementação do protocolo *OAuth2*, um padrão para autorização em sistemas que possui especificações desenvolvidas pela IETF (OAUTH 2.0, 2018, *online)*.

5.2 PLANEJAMENTO

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

5.2.1 Tecnologias

Diversas tecnologias foram utilizadas para o desenvolvimento e possivelmente a 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 *script open source* de uso comum, especialmente adequada para o desenvolvimento web, que foi utilizada como linguagem *server side* para a API.
Laravel	*Framework* para desenvolvimento em PHP, utilizado para desenvolvimento da API REST.
Laravel Passport	Complemento para o *framework Laravel* que prove uma implementação do protocolo de autorização *OAuth2.*
predis	Cliente *Redis* flexível e com recursos completos para PHP.
Nginx	Servidor *Web* utilizado para servir os recursos da API e as páginas da aplicação *Web*.
Maria DB	Servidor SGDB utilizado para persistir os dados da aplicação.

Redis	Serviço de armazenamento de estrutura de dados na memória, usado como banco de dados, *cache* e *message broker*.
Supervisor	Sistema cliente/servidor que permite aos usuários monitorar e controlar vários processos em sistemas
Javascript	Linguagem de scripts utilizada para criação das páginas da aplicação *Web.*
HTML	Linguagem de marcação utilizada para criação das páginas da aplicação *Web*.
CSS	Folhas de estilos para páginas HTML.
Vue.js	*Framework* para desenvolvimento *Javascript* utilizado para criação da aplicação *Web cliente side* ou *front-end*.
Bootstrap	*Framework* para criação de componentes visualmente estilizados com CSS para as páginas HTML.

Fonte: autoria própria.

Os *frameworks* utilizados dependem de outros softwares 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
Composer	Gerenciador de pacotes para PHP.
Git	Sistema de versionamento utilizado para manter organizadas as versões e incrementos da aplicação.
Git Flow	Complemento para o sistema de versionamento *Git* utilizado para simplificar o trabalho com ramificações das versões.
VSCode	Editor de código.
Node.js	Plataforma para desenvolvimento de aplicações utilizando *Javascript*.
Insomnia	Cliente para testes de APIs REST.

Fonte: Autoria própria.

Além das tecnologias citadas no Quadro 24, o desenvolvimento dessa aplicação utilizou o serviço de repositórios do *GitHub* para manter as versões de código e a aplicação online *Trello* para planejamento e controle das atividades *do projeto.*

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
Python	Linguagem de programação sob a qual as ferramentas de conversão de *Markdown* para HTML foi escrita.
pip	Ferramenta para instalar pacotes *Python*.
Python virtualenv	Ferramenta para criar ambientes *Python* isolados.
Python virtualenvwrapper	Conjunto de extensões para a ferramenta *Python virtualenv.*
MkDocs	Ferramenta geradora de sites estáticos a partir de texto estruturado em *Markdown* escrito em linguagem *Python*.
MkDocs Material	Extensão de tema visual para o *MkDocs.*
Draw.io	Ferramenta para criação de diagramas.

Fonte: Autoria própria.

Os *softwares* utilizados dependem de outros *softwares* que não necessitam serem citados em sua totalidade.

Além das tecnologias citadas acima, foi utilizado o serviço *GitHub Pages* para hospedar as páginas do site estático da documentação e o serviço de análise de desempenho de sites *Google Analytics* para monitorar os acessos ao site da documentação.

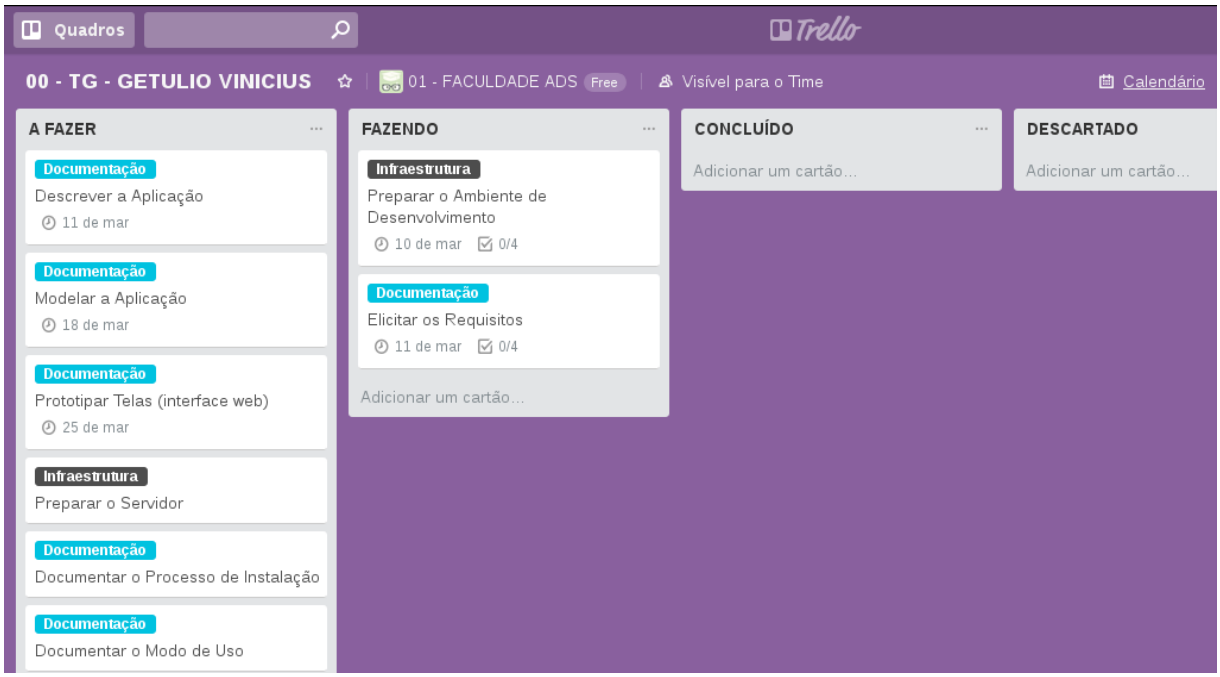
5.2.2 Metodologia

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 MPCT – Metodologia de Pesquisa Cientifica e Tecnológica.

A aplicação *online Trello* foi utilizada para elaboração do quadro *Kanban*, onde foram planejadas as atividades do projeto. A Figura 50 traz uma visão das tarefas planejadas inicialmente em 23 de fevereiro de 2018.

Figura 50 – *Trello*: tarefas planejadas incialmente em 23/02/2018.



Fonte: Autoria própria.

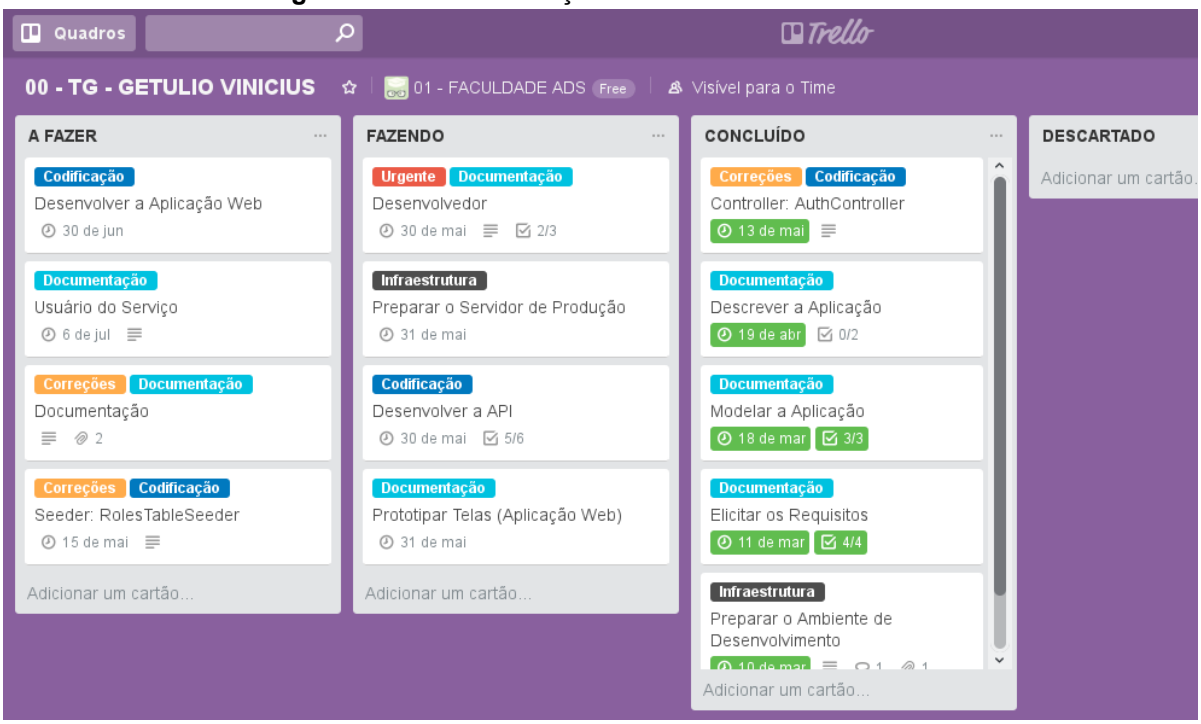
Já a Figura 51 traz uma visão das tarefas em 30 de março de 2018, demonstrando a evolução do processo de desenvolvimento.

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 13 de maio de 2018 em um cenário que restavam ainda algumas tarefas a serem entregues.

Figura 52 – *Trello*: evolução das tarefas em 13/05/2018.



Fonte: Autoria própria.

Nota-se, nas figuras 50, 51 e 53, que as atividades foram inseridas ou modificadas no decorrer da execução do projeto conforme é característico em abordagens de desenvolvimento que utilizam métodos ágeis.

Também inerente as metodologias, foi decidido que o desenvolvimento da aplicação teria principalmente influência das características de métodos ágeis, no que diz respeito a priorização das entregas de funcionalidades em detrimento a documentação abrangente, principalmente em razão do tempo escasso para a entrega do TG.

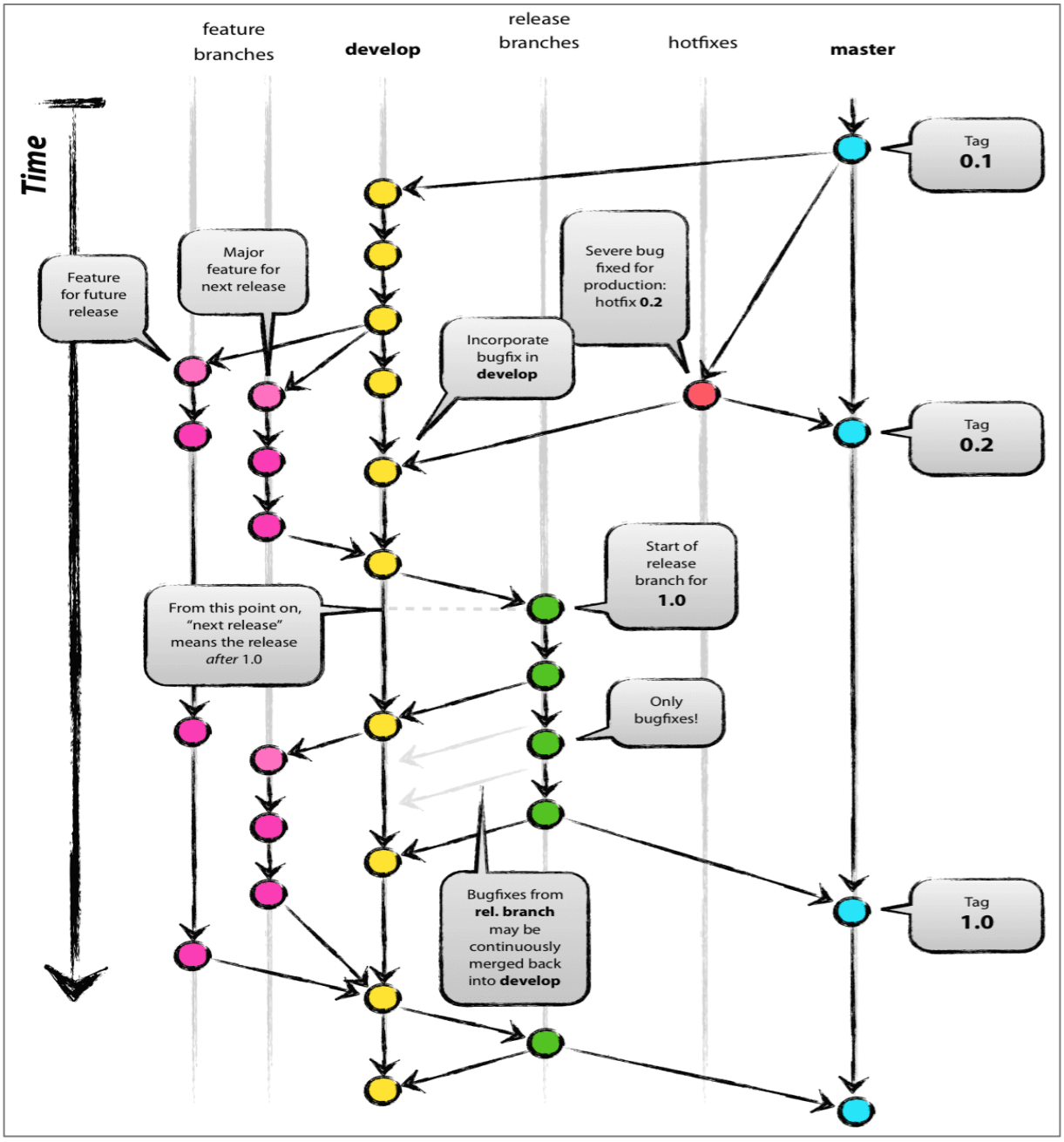
Todavia, a documentação não foi posta em segundo plano, pois essa era desde o princípio a meta do TG, sendo, portanto, produzidos documentos à medida que a relevância dos mesmos para desenvolvimento do projeto era superior ao custo/esforço dispensado para obtenção.

Por último, foi definida uma estratégia a ser adotada para o versionamento da aplicação, sendo escolhido o modelo de versionamento semântico de Preston-Werner (2015, *online*) conforme segue:

> Um número de versão normal DEVE ter o formato de X.Y.Z, onde X, Y, e Z são inteiros não negativos, e NÃO DEVE conter zeros à esquerda. X é a versão Maior, Y é a versão Menor, e Z é a versão de Correção. Cada elemento DEVE aumentar numericamente. Por exemplo: 1.9.0 -> 1.10.0 -> 1.11.0.

Por conseguinte, complementando a adoção do modelo de versionamento semântico, foi escolhido um método para utilização das *branches* (ramificações) – um recurso do *Git* -, que consiste no modelo de Driessen (2010, *online*) apresentado na Figura 53.

Figura 53 – Modelo de trabalho com *branches* para versionamento com o *Git*.



Fonte: Driessen, 2010, *online*.

Nesse cenário trabalha-se com duas *branches* permanentes denominadas *master* e *develop* (principal e desenvolvimento). Na *branch master*, mantem-se somente o código de versões finais que estão estáveis, ou seja, aquelas que acabaram de serem lançadas ou que porventura tenham acabado de receber uma correção. Na ramificação d*evelop* mantem-se as versões que estão em desenvolvimento, ou seja, recebendo novas funcionalidades.

As demais *branches* são temporárias, utilizadas durante o período de desenvolvimento de uma nova funcionalidade (em inglês, *feature*); de testes de uma versão candidata a versão final (em inglês, *release*); e de correção de falhas em versões que já haviam sido lançadas (em inglês, *hotfix*).

Para viabilizar a aplicação desse método de versionamento e trabalho com as *branches*, o *software Git* teve suas funcionalidades estendidas com o complemento *Git-flow,* que provê operações de alto-nível para repositórios através de um conjunto de ferramentas que simplificam as tarefas, reduzindo a quantidade de comandos necessários para criação e posteriormente a mesclagem – ou *merge,* como é costume se dizer referindo-se ao termo em inglês -, das *branches feature, release* e *hotfix* com as *branches master* e *develop* (KUMMER, 2017, *online*).

Ilustrando esse processo, incialmente pode-se mostrar o desenvolvimento de uma correção para a versão 0.7.0 da aplicação deste projeto, utilizando a *branch hotfix,* conforme segue na Figura 54.

Figura 54 – Início da *branch hotfix/0.7.1* para desenvolvimento de uma correção na aplicação.

```
~/DEV_WEB/PHP/VHOSTS/projeto-tg-getulio-vinicius.local on  develop  19:58:12
$ git flow hotfix start 0.7.1
Switched to a new branch 'hotfix/0.7.1'

Summary of actions:
- A new branch 'hotfix/0.7.1' was created, based on 'master'
- You are now on branch 'hotfix/0.7.1'

Follow-up actions:
- Start committing your hot fixes
- Bump the version number now!
- When done, run:

     git flow hotfix finish '0.7.1'


~/DEV_WEB/PHP/VHOSTS/projeto-tg-getulio-vinicius.local on  hotfix/0.7.1  19:59:08
$ ls
```

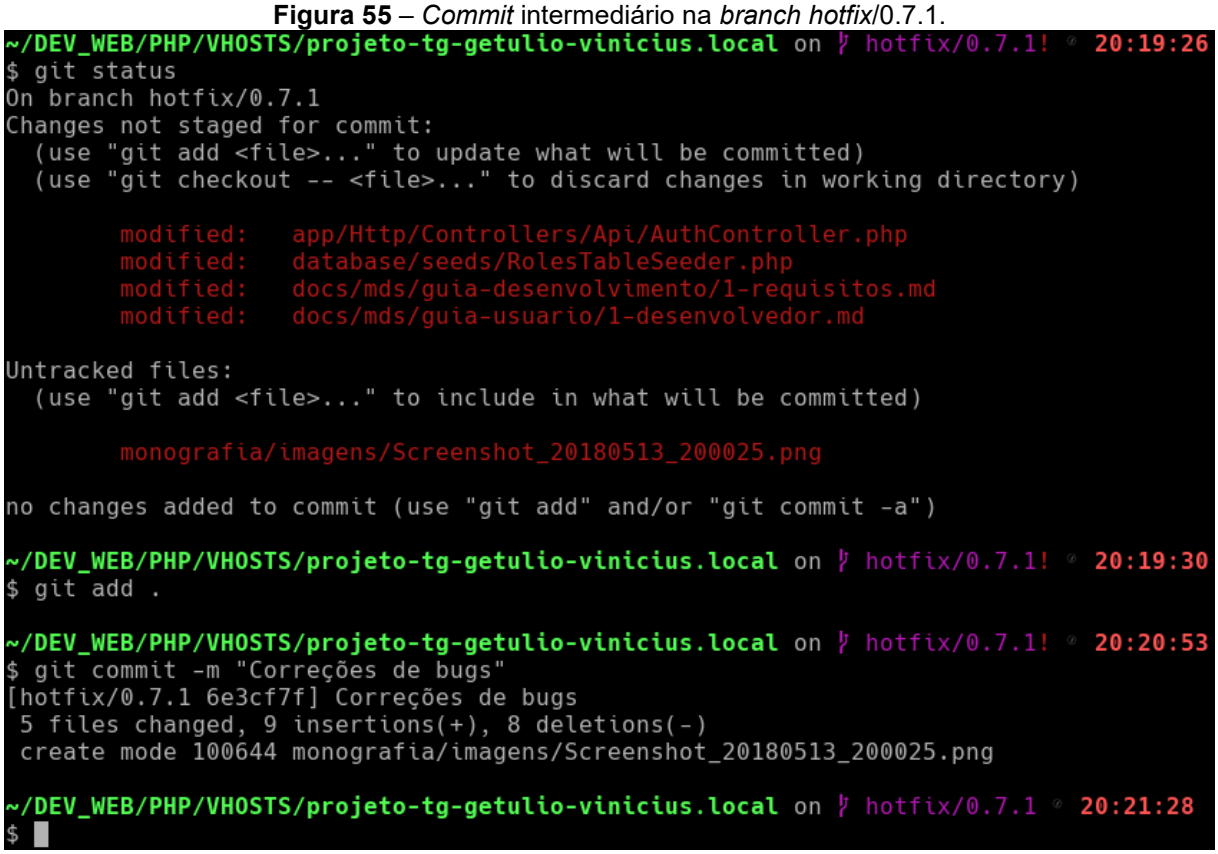
Fonte: Autoria própria.

O comando para criação da *branch*, é visto na segunda linha, *git flow hotfix start 0.7.1.*

Nota-se que por se tratar de uma correção, o último algarismo da versão foi alterado de 0 para 1, conforme preconiza o sistema de versionamento semântico.

A Figura 55 traz a visão do processo de criação de um *commit* (em português pode ser compreendido como confirmação da alteração) intermediário, que ocorreu durante o desenvolvimento da correção feita na *branch hotfix/0.7.1.*

Figura 55 – *Commit* intermediário na *branch hotfix*/0.7.1.



Fonte: Autoria própria.

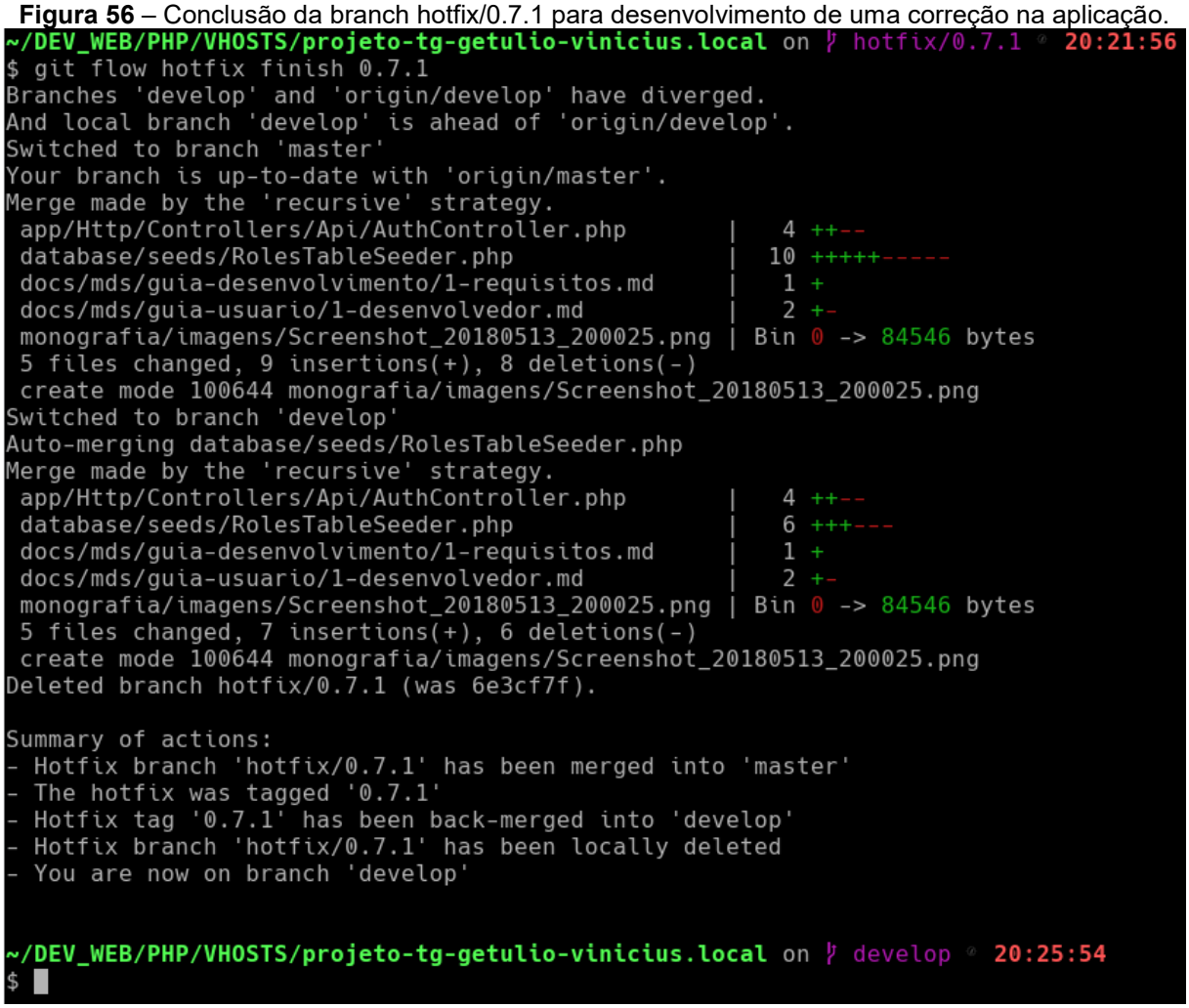
Os comandos inseridos para realização do commit, em ordem foram:

- *git status* – para checagem dos arquivos alterados*;*
- *git add .* – para incluir todos arquivos que foram alterados a partir do diretório corrente, representado pelo "." ponto, na área de *commit*;
- *git commit -m "Correção de ..."* – para efetuar o commit, sendo o conteúdo passado entre aspas, após o parâmetro -m uma mensagem.

Uma *branch* pode receber quantos *commits* forem necessários.

Na Figura 56, tem-se o fechamento do processo de correção, que ocorre após a realização dos *commits* intermediários, inserindo o comando *git flow hotfix finish 0.7.1.*

Figura 56 – Conclusão da branch hotfix/0.7.1 para desenvolvimento de uma correção na aplicação.



Fonte: Autoria própria.

Observa-se, nas últimas linhas da saída do comando, em *Summary of actions* (resumo das ações), que a *branch hotfix/0.7.1*, foi fundida (em inglês, *merged*) com as *branches master* e *develop*, e, logo após ela foi excluída, pois cumpriu a função para a qual foi criada e não tinha mais serventia.

Assim, a versão estável passa a ser a 0.7.1 e tanto a *branch develop* quanto a *branch master* receberam as correções.

O processo para criação de uma nova funcionalidade (*branch feature*) ou para a realização de testes (*branch release*), é semelhante ao processo demonstrado para o desenvolvimento de correções, mudando apenas o terceiro parâmetro dos comandos, ou seja, em lugar de *hotfix* no comando *git flow hotfix* ..., sendo "..." a sequência dos parâmetros, usa-se *feature* ou *branch*.

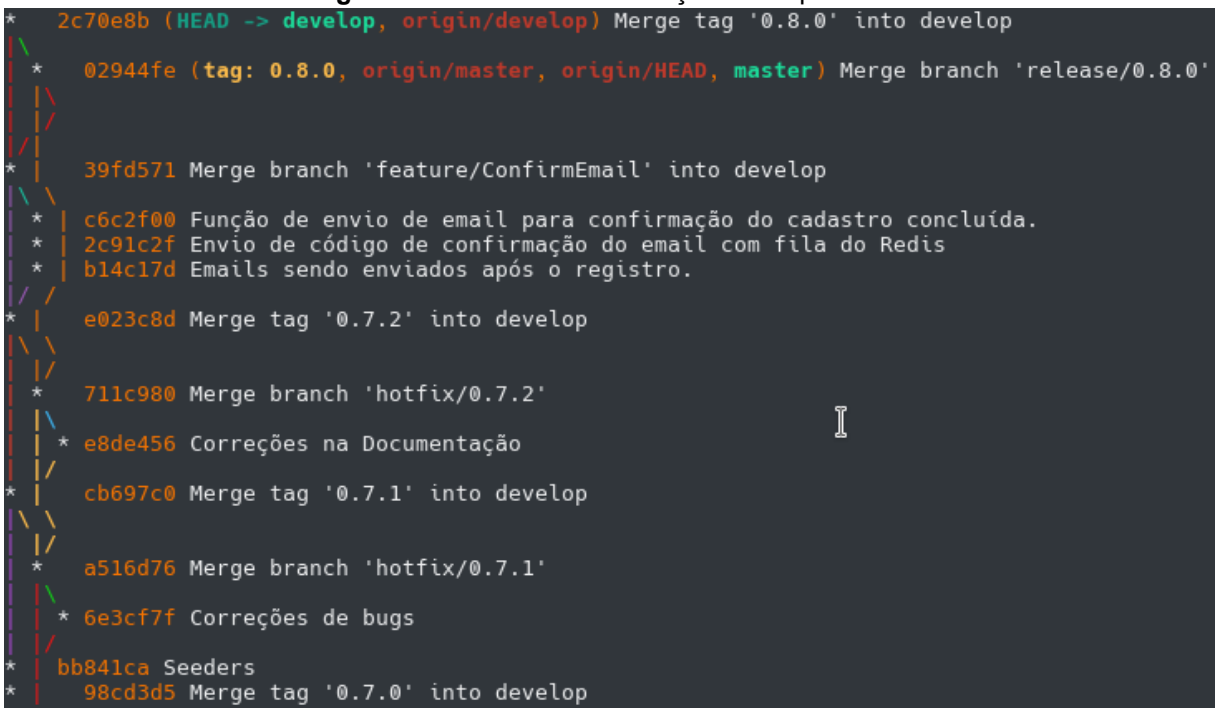
A conclusão de uma *feature* se dá com uma fusão com a *branch develop*, sem que ocorra alterações no número da versão, ao passo que a conclusão de uma *release* se dá com a fusão com as *branches master e develop*, porém, com a alteração no

primeiro ou no segundo algarismo da versão, de acordo com o tipo de versão que se está gerando:

- Maior – altera-se o primeiro algarismo para o número inteiro seguinte e os segundo e terceiro algarismos passam a ser 0; e
- Menor – altera-se o segundo algarismo para o número inteiro seguinte, o terceiro algarismo passa a ser 0 e primeiro algarismo é mantido.

A Figura 57 traz uma visão linear das alterações que o repositório do projeto recebeu em seu versionamento, vistas em ordem de baixo para cima, desde a versão 0.7.0 até a versão 0.8.0, de acordo com o modelo de (DRIESSEN, 2010, *online*).

Figura 57 – Histórico de alterações no repositório.



Fonte: Autoria própria.

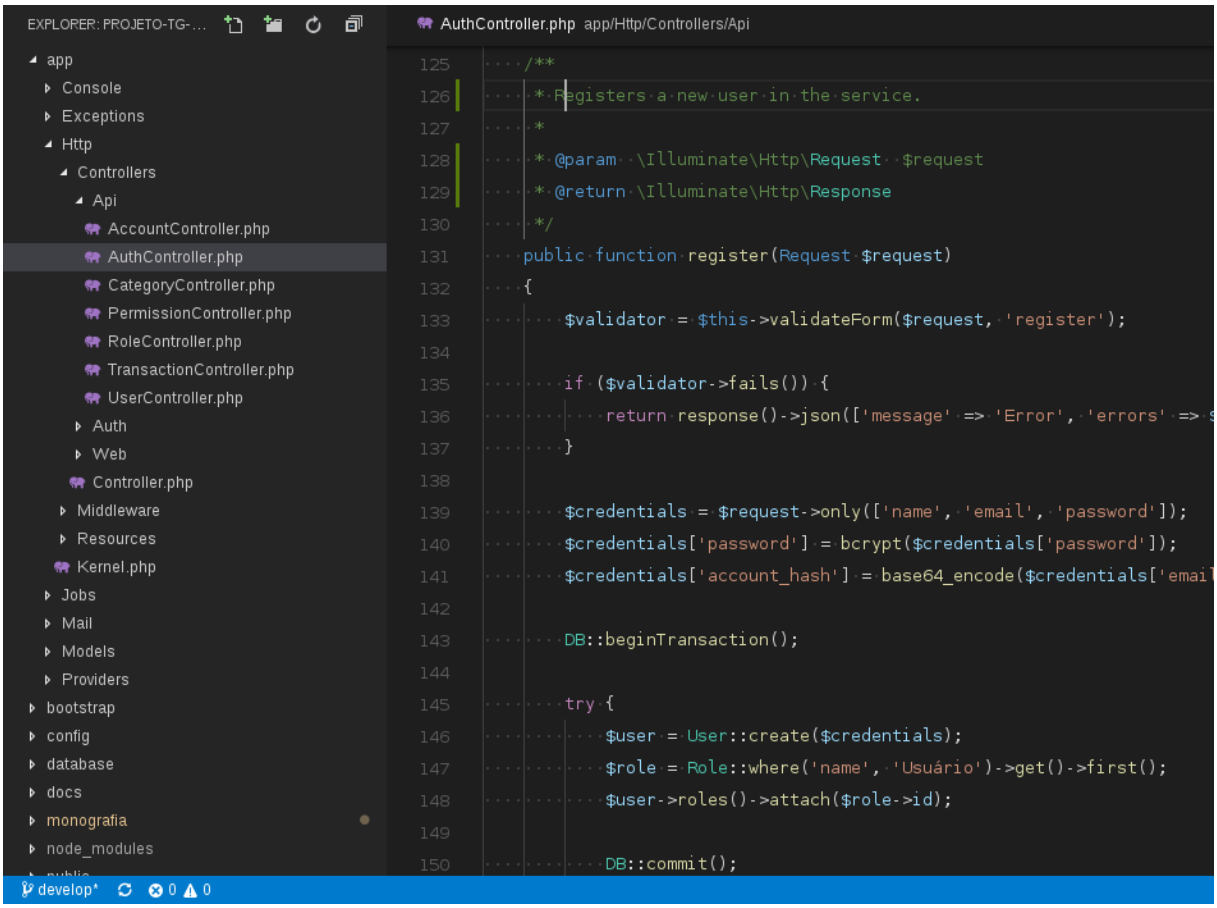
5.3 EXECUÇÃO

Excetuando-se as configurações dos serviços externos, do ambiente de desenvolvimento, além do desenho dos diagramas de modelagem e a prototipagem de telas da interface da aplicação Web, o trabalho concentrou-se na maior parte do tempo na transição entre o terminal do SO Debian *GNU/Linux*, o editor de código *Visual Studio Code* e o aplicativo para testes da API *Insomnia*

Um navegador de internet foi utilizado para verificar o resultado da conversão dos arquivos de documentação, escritos em *Markdown*, para HTML.

A Figura 58 traz uma visão de parte do código da função de registro de usuários na aplicação, bem como uma parte da arvore de diretórios e arquivos que compõem a aplicação.

Figura 58 – Parte do código e estrutura de diretórios e arquivos da aplicação.



Fonte: Autoria própria.

O processo de desenvolvimento da aplicação, até a sua versão 1.0.0 não havia sido concluído quando excedeu o prazo para entrega do TG, todavia, foram executadas as seguintes tarefas:

- **Preparação do ambiente de desenvolvimento:** Foi instalada toda a pilha de software que está listada entre as tecnologias citadas nos quadros da sessão 5.2.1 em um computador com as seguintes características: *Notebook* com processador *Intel Core i 5 i 4x2,7 GHz,* 8 GB de memória RAM, HD de 120 GB SSD, SO *Debian GNU/Linux* e interface gráfica KDE Plasma 5.

- **Configuração de serviços no *GitHub* e *Google Analytics*:** Foi preparado o repositório para publicação do código no *GitHub* e da documentação no

GitHub Pages, além da integração entre as páginas da documentação e o serviço *Google Analytics.*

- **Documentação:** Foi escrita a documentação da aplicação contemplando os recursos que já haviam sido concluídos. Além disso, foi concluído o desenvolvimento da plataforma de documentação com uso do gerador de sites estáticos *MkDocs* e publicada no serviço *GitHub Pages*, podendo ser acessada no endereço http://bit.ly/2IJR93Y. A documentação escrita em *Markdown* pode ser visualizada no Apêndice A.
- **Elicitação dos requisitos para a versão 1.0.0:** Foi realizado o levantamento de requisitos de tecnologia, da aplicação do usuário do serviço e do administrador do serviço. O documento de requisitos pode ser acessado no endereço http://bit.ly/2rTt3cL, e o texto em *Markdown* pode ser visto no Apêndice A
- **Modelagem da aplicação:** Foram desenvolvidos os modelos de banco de dados e casos de uso para a aplicação. A modelagem foi adicionada a documentação que está disponível no endereço http://bit.ly/2rLsd27. O texto em *Markdown* pode ser visto no Apêndice A e as figuras que representam os modelos no Apêndice B.
- **Prototipação da interface da aplicação web:** Foram desenhadas duas telas para a aplicação baseadas no conceito de modelos de média fidelidade. Os protótipos estão disponíveis no endereço http://bit.ly/2k4fhAv, o texto em *Markdown* está no Apêndice A e as figuras que representam as telas podem ser vistas no Apêndice B.
- **Guia para o usuário desenvolvedor:** Foi escrito um guia de utilização para usuários desenvolvedores que tiverem interesse em contribuir com o projeto de software livre. O guia traz todas as informações necessárias para instalação da aplicação em um ambiente local de desenvolvimento e está disponível no endereço http://bit.ly/2rMj5ue. O texto em *Markdown* está no Apêndice A.
- **Desenvolvimento da API:** Foi desenvolvida a API com o auxílio do *framework* PHP *Laravel*, versão 5.6. A API é estável na versão 0.8.0, tendo sido implementados parcialmente os requisitos da aplicação e do usuário

serviço. Não foram implementados os requisitos do administrador do serviço.

Para conclusão da entrega da versão 1.0.0 da aplicação restam as seguintes tarefas a serem cumpridas:

- **Desenvolvimento da API:** Implementação dos requisitos faltantes para a aplicação e para o usuário do serviço e a implementação total dos requisitos para o administrador do serviço.
- **Aplicação Web para consumir a API:** Desenvolvimento de uma aplicação web com o auxílio do *framework Javascript Vue.js* para consumir a API.
- **Guia para o usuário do serviço:** Documentar o uso da aplicação web para usuários que irão se cadastrar no serviço.

5.4 FECHAMENTO

Ao optar pela adoção de princípios de desenvolvimento característicos de metodologias ágeis, a rigor o projeto pode incorporar novos requisitos a qualquer momento, no entanto, considerando a possibilidade do não descobrimento de novas regras de negócio, requisitos para aceitação e outros fatores não previstos até o momento, o projeto caminha para o seu fechamento com a execução das três tarefas que ficaram pendentes na etapa de execução.

Além da conclusão das tarefas pendentes para entrega da versão 1.0.0 da aplicação tgGV, o fechamento do projeto incluirá a apresentação de requisitos descobertos que poderão compor uma futura versão 2.0.0 da aplicação e, possivelmente, marcará o início de um novo projeto objetivando a implantação da aplicação em modo de produção.

CONSIDERAÇÕES FINAIS

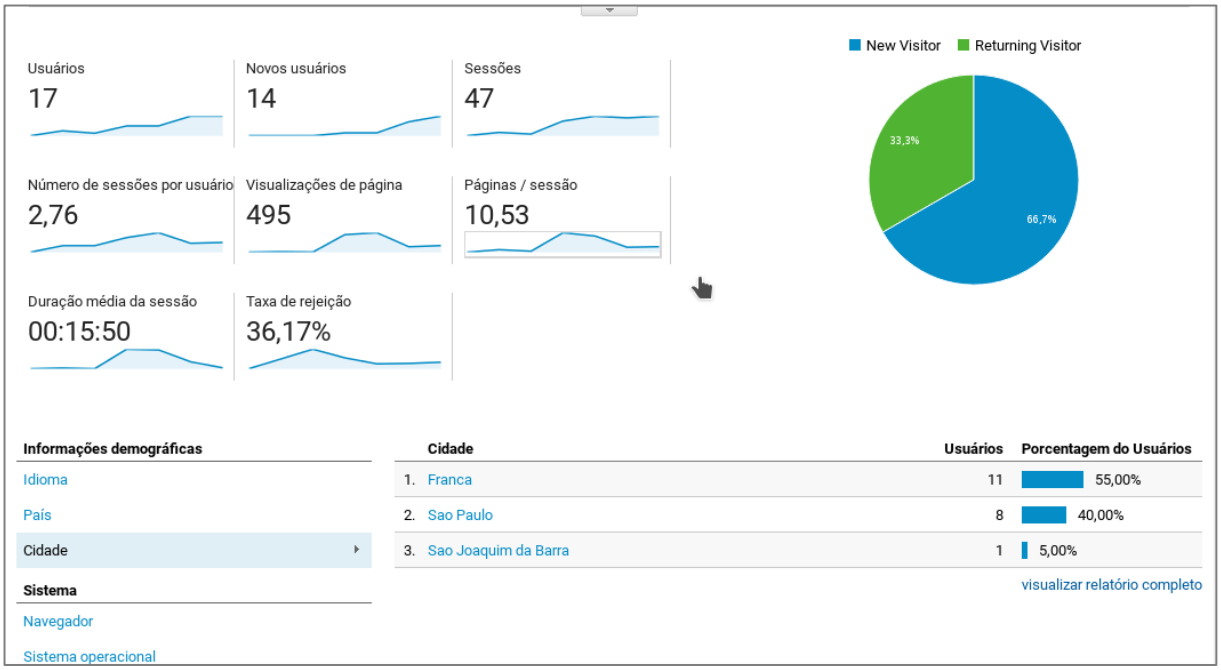
O objetivo inicial do TG, que era apresentar uma forma viável de promover a gestão da documentação produzida em um projeto de *software*, foi atingido, sendo a escolha da linguagem *Markdown* juntamente com gerenciador de sites estáticos *MkDocs*, uma opção realmente viável para desenvolvedores promoverem a elaboração da documentação de seus projetos.

Outra vantagem desse método de trabalho são as integrações que se pode fazer. Por exemplo, além de propiciar uma gestão eficiente da documentação, o *MkDocs* permite a publicação da mesma em serviços de repositórios baseados no sistema de versionamento *Git*, que são amplamente utilizados e torna fácil a distribuição do *software* juntamente com a sua documentação.

Não obstante, tudo isso já possa ser considerado enorme vantagem para o desenvolvedor de *software*, ainda existe a possibilidade de integrar facilmente o site estático gerado pelo *MkDocs* ao serviço de análise de desempenho de sites *Google Analytics*, que permite verificar como os usuários utilizam a documentação do *software*, quais páginas são mais consultadas, quanto tempo os usuários dedicam a leitura das instruções, de onde são esses usuários, entre outros recursos.

A Figura 59 traz uma amostra de relatório para o site da documentação do projeto tgGV gerado pelo *Google Analytics*.

Figura 59 – Visão geral dos acessos ao site da documentação da aplicação tgGV.



Fonte: Autoria própria.

Por fim, há que se ressaltar que as tecnologias que foram citadas ao longo deste TG e que permitiram a realização dessa abordagem para documentação de *software*, são pouco difundidas na instituição de ensino, de modo que, a partir desse trabalho é de se esperar que mais alunos possam recorrer ao uso de softwares como *Git, MkDocs* e a linguagem *Markdown* para desenvolverem suas atividades acadêmicas.

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.

BARE BONES. BBEdit 12. **Bare Bones**, 2017. Disponível em: <https://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 a. 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 b. 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.

DRIESSEN, V. A successful Git branching model. **nvie.com**, 05 jan. 2010. Disponível em: <http://nvie.com/posts/a-successful-git-branching-model/>. Acesso em: 16 maio 2018.

FELDMAN, I. setext_concepts_Aug92.etx. **Docutils:** Documentation Utilities, 16 ago. 1992. Disponível em: <http://docutils.sourceforge.net/mirror/setext/setext_concepts_Aug92.etx.txt>. Acesso em: 16 mai. 2018.

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.

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**, 2018. 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 a. Disponível em: <https://daringfireball.net/2004/03/dive_into_markdown>. Acesso em: 09 nov. 2017.

GRUBER, J. Introducing Markdown. **Daring Fireball**, 15 mar. 2004 b. Disponível em: <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.

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_AreHm4dlKP6QQCekuIPky1CiwmdI6>. 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.

KUMMER, D. cheatsheet do git-flow. **cheatsheet do git-flow**, 30 out. 2017. Disponível em: <https://danielkummer.github.io/git-flow-cheatsheet/index.pt_BR.html>. Acesso em: 16 mai. 2018.

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

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

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

LOPEZ, A. **Learning PHP 7**. Birmingham: Packt Publishing Ltd, 2016.

LUSA, D. A. 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.

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

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

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

OAUTH 2.0. **OAuth 2.0**, 2018. Disponível em: <https://oauth.net/2/>. Acesso em: 16 mai. 2018.

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 Processs Model and Notation (BPMN) - Version 2.0**. [S.l.]: [s.n.], 2011. Disponível em: <http://www.omg.org/spec/BPMN/2.0>.

PAULA FILHO, W. D. P. **Engenharia de Software - Fundamentos, Métodos e Padrões**. 3. ed. ed. Rio de Janeiro: LTC, 2013.

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

PRESTON-WERNER, T. Versionamento Semântico 2.0.0. **Versionamento Semântico 2.0.0**, 17 nov. 2015. Disponível em: <https://semver.org/lang/pt-BR/>. Acesso em: 16 mai. 2018.

RESTRUCTUREDTEXT - Markup Syntax and Parser Component of Docutils. **Docutils:** Documentation Utilities, 24 mai. 2016. Disponível em: <http://docutils.sourceforge.net/rst.html>. Acesso em: 16 mai. 2018.

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

SCHWBER, K.; SUTHERLAND, J. **Guia do Scrum**. [S.l.]: [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>.

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.

STUMM JR, V. Documentando um programa Python com docstrings e Pydoc. **Python Help**, 14 fev. 2011. Disponível em: <https://pythonhelp.wordpress.com/2011/02/14/docstrings/>. Acesso em: 16 mai. 2018.

SWARTZ, A. atx, the true structured text format. **Aaron Swartz**, 2002. Disponível em: <www.aaronsw.com/2002/atx/intro>. Acesso em: 16 mai. 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.

WIKIPEDIA. Help:Wikitext. **WIKIPEDIA - The Free Encyclopedia**, 02 mai. 2018. Disponível em: <https://en.wikipedia.org/wiki/Help:Wikitext>. Acesso em: 16 mai. 2018.

APÊNDICE A

Será apresentado o texto, escrito em *Markdown,* para cada uma das páginas do site estático que foi gerado para documentar o projeto da aplicação Orçamento Pessoal - tgGV.

Na sequência do texto de cada uma das páginas, será apresentada uma captura de tela exibindo uma parte da página renderizada pelo navegador *web*.

Arquivo: /mds/index.md

```
Orçamento Pessoal – tgGV
========================

**Versão:** 0.7.2 – 13/05/2018

Sobre o projeto
---------------

Olá!

Meu nome é [Getúlio Vinícius](https://github.com/getuliovinicius "Perfil no
GitHub").

Orçamento Pessoal – tgGV, é um software livre cujo desenvolvimento foi iniciado
como parte do meu Trabalho de Graduação (monografia) no curso de Análise e
Desenvolvimento de Sistemas da Faculdade de Tecnologia de Franca, FATEC Dr.
Thomaz Novelino.

!!! important "MONOGRAFIA"
    **Ficou curioso sobre o trabalho?**

    Você pode ler a [MONOGRAFIA](downloads/monografia.pdf "Ainda em
desenvolvimento").

Descrição
---------

Trata-se de uma aplicação para realização de orçamento pessoal arquitetada da
seguinte forma:

+ API REST desenvolvida com o framework PHP Laravel, e
+ Aplicação Web desenvolvida com o framework Javascript Vue.js.

O tgGV tem como objetivo permitir ao usuário a criação de um plano de contas
(orçamento pessoal) para controlar suas finanças.
```

Através de lançamentos realizados com o método das partidas dobradas, o tgGV permite a visualização das receitas e despesas de modo que o usuário possa compreender como utiliza seu dinheiro.

Página: Orçamento Pessoal – tgGV

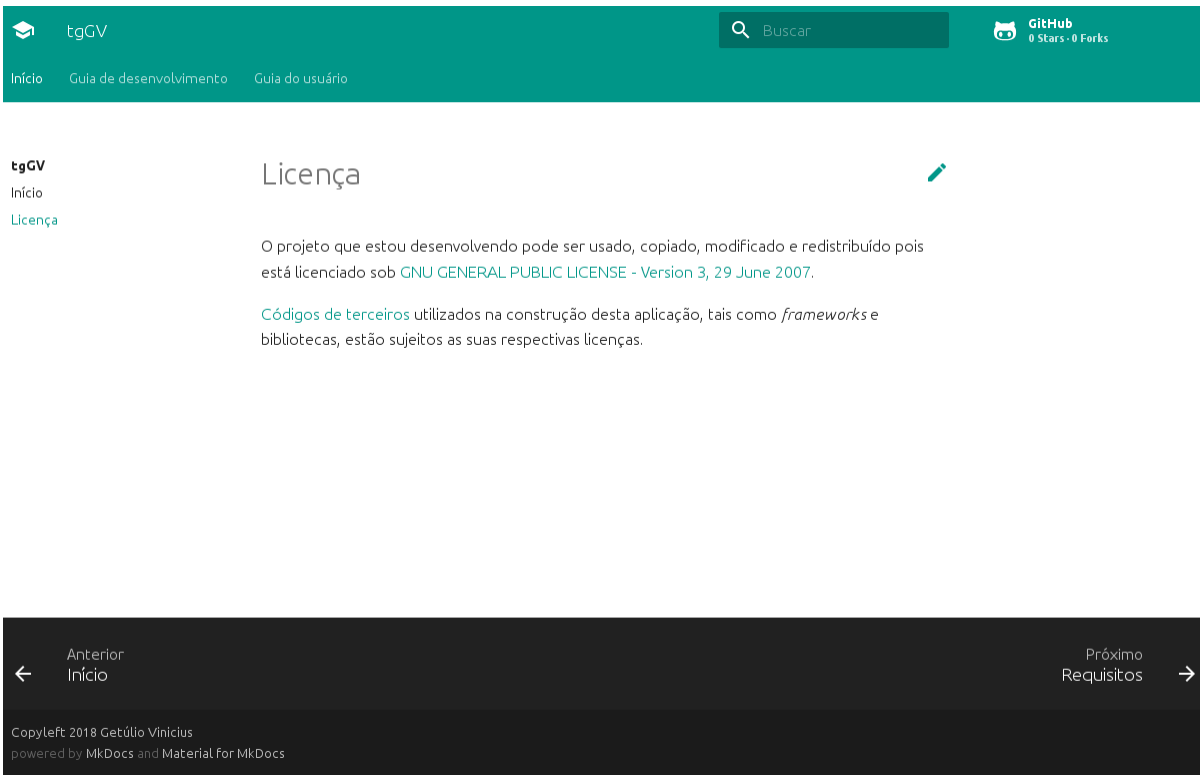
Arquivo: /mds/licence.md

```
Licença
=======

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](downloads/LICENSE "Download da Licença").

[Códigos de terceiros](guia-desenvolvimento/4-tecnologias.md "Lista de softwares
utilizados") utilizados na construção desta aplicação, tais como _frameworks_ e
bibliotecas, estão sujeitos as suas respectivas licenças.
```

Página: Licença



Arquivo: /mds/guia-desenvolvimento/1-requisitos.md

```
Documento de Requisitos
=======================

**Atualizado em:** 12/05/2018

Requisitos para a versão 1.0.0
------------------------------

### Requisitos de tecnologia

- [X] Arquitetura orientada a serviço.
- [X] API – Web service REST.
- [ ] Aplicação Web para consumir a API.

### Requisitos da aplicação

Abaixo segue a tabela com a lista de requisitos gerais da aplicação.

| API | Web | Requisito |
|:---:|:---:|:--------- |
| OK  |  -  | Autenticação de usuários. |
```

```
| OK  |  -  | A aplicação deve possuir quatro papéis de usuário (Super Usuário, Administrador, Gerente e Usuário). |
|  -  |  -  | Deve existir apenas um Super Usuário denominado `root` que não poderá ser excluído. |
| OK  |  -  | Nenhum usuário pode visualizar, alterar, excluir ou criar uma categoria, conta ou lançamento de outro ou para outro usuário. |
|  -  |  -  | A aplicação deve gerar relatórios do tipo lista de usuários (ativos, inativos, bloqueados) |
|  -  |  -  | A aplicação deve gerar relatório sintético de contas criadas por período |

### Requisitos do usuário do serviço

Esses são os requisitos e regras de negócio da aplicação para usuários que irão criarão uma conta no serviço a fim de realizarem o controle de suas finanças pessoais.

| API | Web | Requisito |
|:---:|:---:|:--------- |
| OK  |  -  | As pessoas que tiverem interesse em usar o serviço devem se cadastrar em um formulário público informando o nome completo, endereço de e-mail e senha. |
|  -  |  -  | Após realizar seu cadastro na aplicação o usuário deve receber um link no e-mail informado, apontando para o endereço de ativação da conta. |
| OK  |  -  | Todas as contas para usuários do serviço cadastradas através do formulário público devem ser atribuídas ao papel Usuário. |
|  -  |  -  | O usuário cadastrado pode alterar sua senha de acesso a aplicação e outros dados pessoais. |
|  -  |  -  | O usuário cadastrado pode remover seu cadastro do serviço. |
| OK  |  -  | O usuário cria categorias para organizar as contas. |
| OK  |  -  | O usuário pode alterar o nome das categorias. |
| OK  |  -  | O usuário pode excluir uma categoria desde que não existam contas vinculadas a esta categoria. |
| OK  |  -  | O usuário cria as contas observando os grupos Ativo, Passivo, Patrimônio Líquido, Receita e Despesa. |
| OK  |  -  | O usuário pode alterar o nome e a categoria de uma conta. |
| OK  |  -  | O usuário pode excluir uma conta desde que não existam lançamentos vinculados a esta conta. |
| OK  |  -  | O usuário faz lançamentos nas contas. |
| OK  |  -  | O usuário pode realizar correções nos lançamentos. |
| OK  |  -  | O usuário lista as contas com seus respectivos saldos. |
| OK  |  -  | O usuário visualiza as contas com seus respectivos lançamentos. |
| OK  |  -  | O saldo das contas somente é atualizado a partir de lançamentos onde são identificadas as contas de origem e de destino da movimentação. |

### Requisitos do administrador do serviço
```

```
Esses são os requisitos e regras de negócio para usuários que gerenciarão a
aplicação.

| API | Web | Requisito |
|:---:|:---:|:--------- |
|  -  |  -  | Apenas o usuário `root` pode gerenciar o cadastro de usuários com o
papel Administrador. |
|  -  |  -  | O usuário `root` e os demais usuários com papel Administrador podem
gerenciar o cadastro de usuários com o papel Gerente. |
|  -  |  -  | O usuário `root` e os demais usuários com papel Administrador podem
bloquear o cadastro de usuários com o papel Usuário. |
|  -  |  -  | O usuário com papéis de Administrador ou Gerente podem acessar
relatórios gerenciais. |

Requisitos da versão 2.0.0
--------------------------

Até o momento todos os requisitos elencados para a versão 1.0.0 são também
requisitos para a versão 2.0.0 e não sofreram alterações.

### Requisitos da aplicação

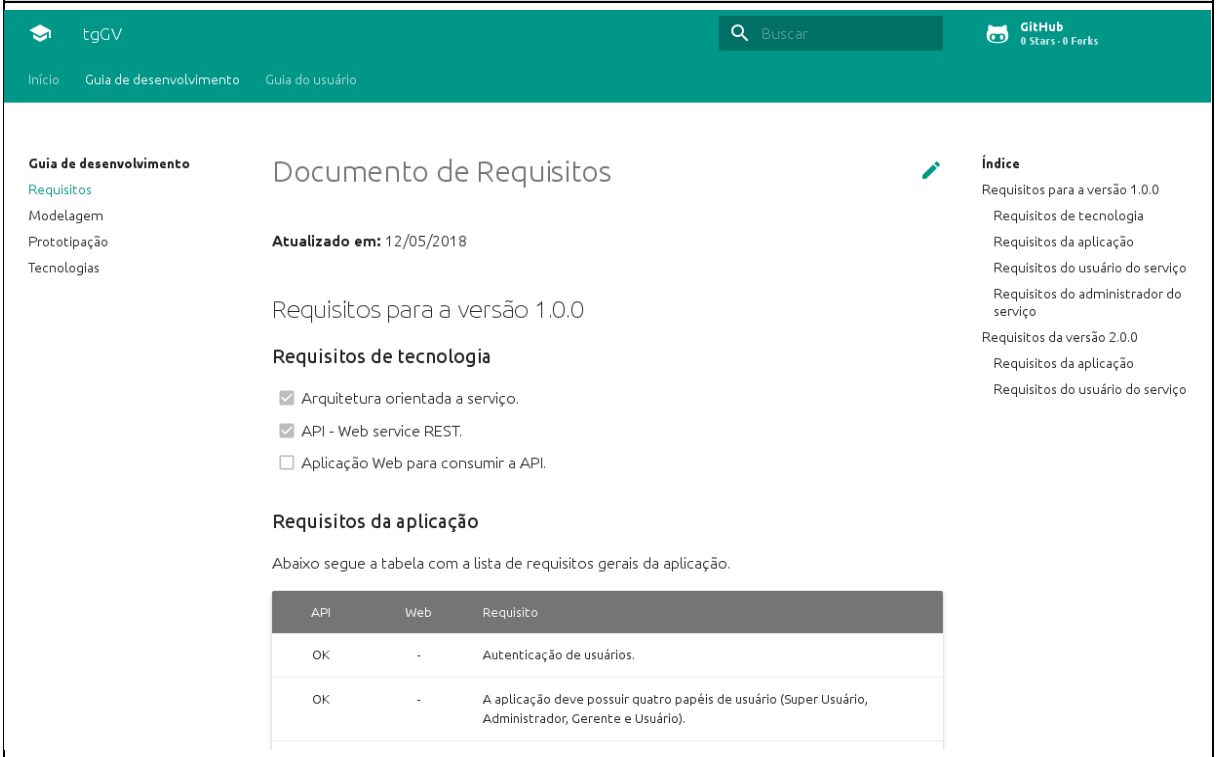
Abaixo segue a tabela com a lista de requisitos gerais da aplicação.

| API | Web | Requisito |
|:---:|:---:|:--------- |
|  -  |  -  | A aplicação deve registrar em log as ações dos usuários (criação,
alteração e exclusão de lançamentos e acessos) |

### Requisitos do usuário do serviço

| API | Web | Requisito |
|:---:|:---:|:--------- |
|  -  |  -  | O usuário faz lançamentos nas contas de: despesa, receita ou
pagamento que são recorrentes, de modo a criar uma agenda. |
|  -  |  -  | Os lançamentos agendados não alteram os saldos das contas enquanto
não forem concretizados. |
```

Página: Guia Desenvolvimento > Requisitos



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

```
Modelagem
=========

**Atualizado em:** 08/05/2018

Banco de dados
--------------

### Modelo conceitual

Descreve todo o modelo de entidades utilizado para desenvolvimento da versão
1.0.0 da aplicação.

![DER – Diagrama Entidade Relacionamento](diagramas/der-1.png)

### Modelo lógico

Descreve as tabelas do Banco de Dados da aplicação.

```
Permission (* id, name, description, created_at, updated_at)
```
```

```
Role (* id, name, description, created_at, updated_at)

Permission_Role (* id, created_at, updated_at, + permission_id, + role_id)
    permission_id reference Permission
    role_id reference Role

User (* id, name, email, password, account_activated, account_hash,
remember_token, created_at, updated_at, deleted_at)

Role_User (* id, created_at, updated_at, + role_id, + user_id)
    role_id reference Role
    user_id reference User

Category (* id, category, created_at, updated_at, + user_id)
    user_id reference User

Account (* id, account, type, balance, allow_negative_balance, created_at,
updated_at, + user_id, + category_id)
    user_id reference User
    category_id reference Category

TrasactionLog (* id, date, confirmed, description, value, created_at, updated_at,
+source_account_id, +destination_account_id)
    source_account_id reference Account
    destination_account_id reference Account
```

Caso de uso

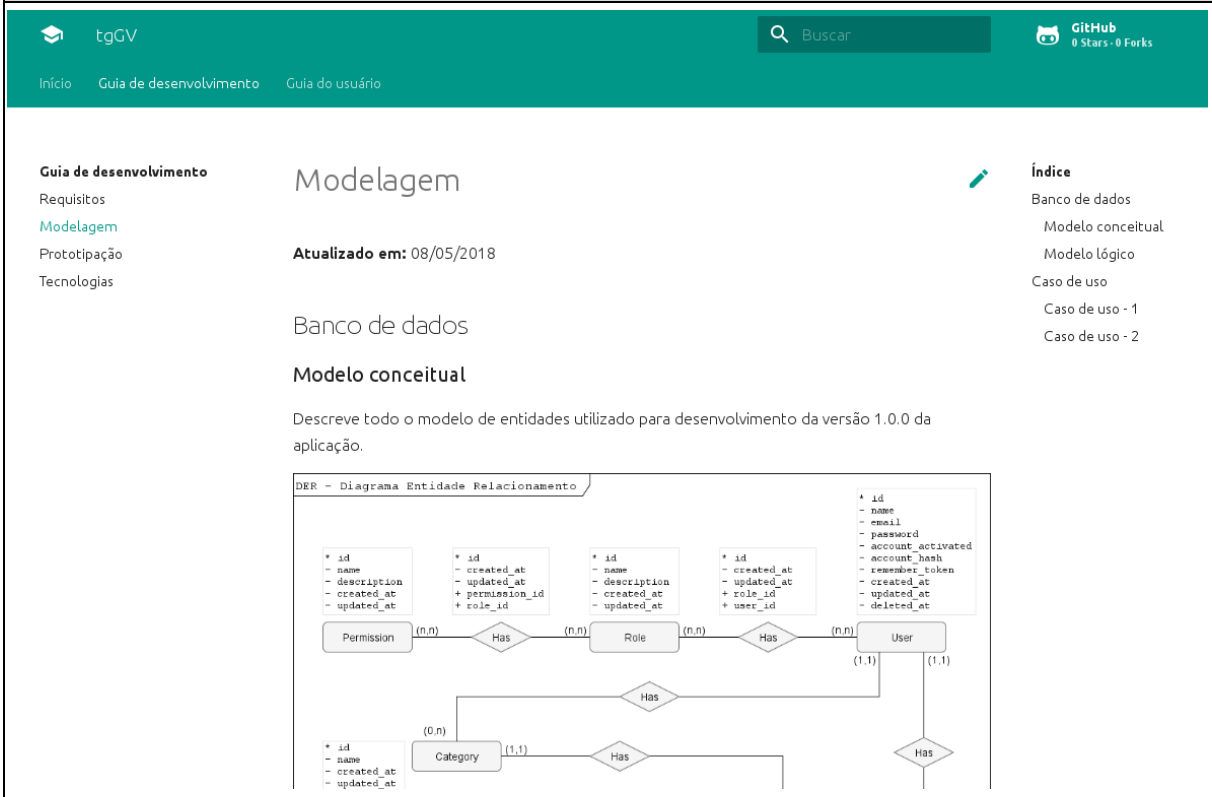
Caso de uso – 1

Demonstra as funcionalidades disponibilizadas para os usuários do serviço oferecido pela aplicação.

Caso de uso – 2

Demonstra as funcionalidades disponibilizadas para os administradores de sistema que porventura possam usar o serviço como negócio.

Página: Guia Desenvolvimento > Modelagem



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

```
Prototipação
============

**Atualizado em:** 30/03/2018


Tela 1 – Agenda
---------------

Listagem das movimentações efetivadas e programadas.

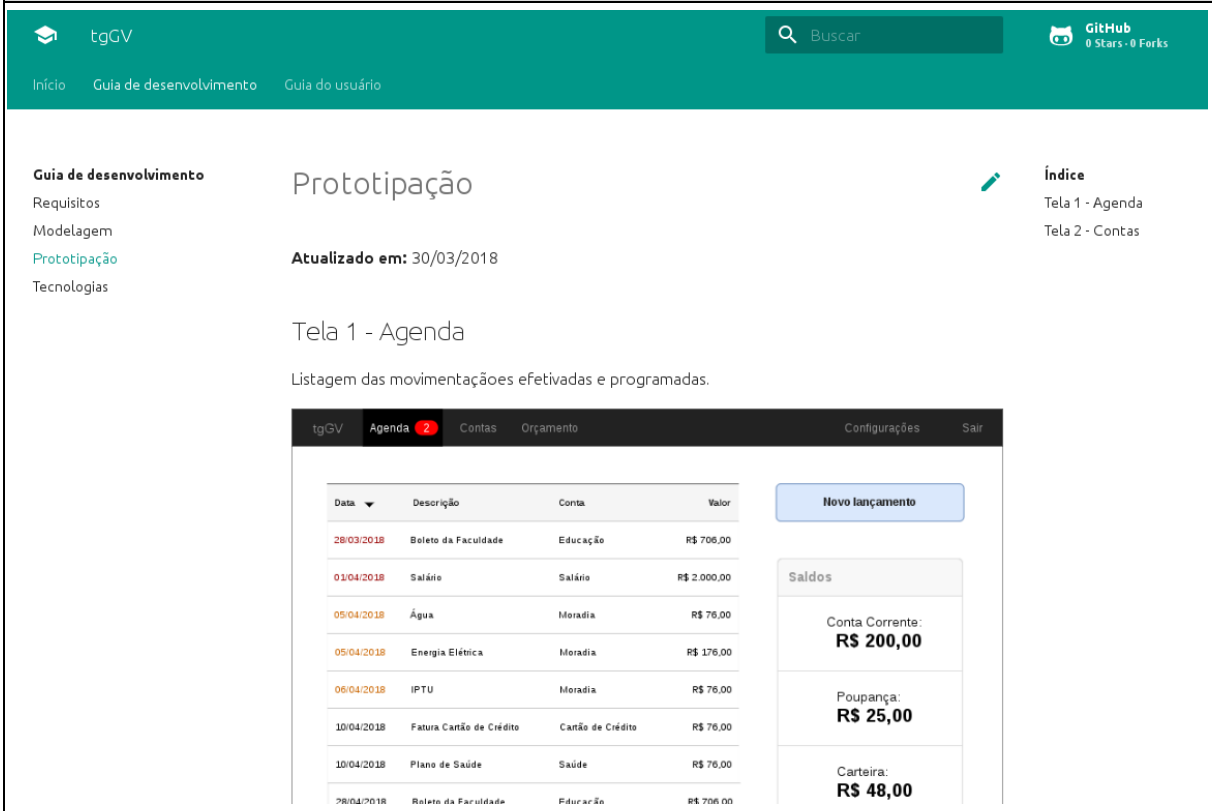
![Tela 1](prototipos/prototipo-1.png)

Listagem das contas com os respectivos saldos.

Tela 2 – Contas
---------------

![Tela 2](prototipos/prototipo-2.png)
```

Página: Guia Desenvolvimento > Prototipação



Arquivo: /mds/guia-desenvolvimento/4-tecnologias.md

```
Tecnologias
===========

**Atualizado em:** 12/05/2018

Desenvolvimento e produção
--------------------------

Pilha de software – tecnologias – utilizadas para o desenvolvimento e implantação
da aplicação em modo de produção.

| Software | Versão | Descrição | Licença |
|:----- |:-----:|:----- |:----- |
| PHP | 7.2.5 | Linguagem de script open source de uso comum, especialmente
adequada para o desenvolvimento Web, que foi utilizada como linguagem server side
para a API. | open source |
| Laravel | 5.6 | Framework para desenvolvimento em PHP, utilizado para
construção da API REST. | [MIT license](https://opensource.org/licenses/MIT) |
| Nginx | 1.10.3 | Servidor Web utilizado para servir os recursos da API e as
páginas da aplicação Web. | [2-clause BSD-like license](http://nginx.org/LICENSE)
|
```

```
| Maria DB | 10.1.33 | Servidor SGDB utilizado para persistir os dados da
aplicação. | [GNU General Public License
(GPLv2)](https://github.com/MariaDB/server#license) |
| Javascript | – | Linguagem de scripts utilizada para criação das páginas da
aplicação Web. | – |
| HTML | 5 | Linguagem de marcação utilizada para criação das páginas da
aplicação Web. | – |
| CSS  | 3 | Folhas de estilos para páginas HTML. | – |
| Vue.js | 2.5.7 | Framework para desenvolvimento Javascript utilizado para
criação da aplicação Web client side ou front-end. | [MIT
license](https://opensource.org/licenses/MIT) |
| Bootstrap | 4.0.0 | Framework para criação de componentes visualmente
estilizados com CSS para as páginas HTML. | [MIT
license](https://github.com/twbs/bootstrap/blob/master/LICENSE) |
| Laravel Passport | 6.0.0 | Complemento para o framework Laravel que prove uma
implementação do servidor de autorização token OAuth2. | [MIT
license](https://opensource.org/licenses/MIT) |

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

Desenvolvimento
---------------

Também foram utilizadas diversas tecnologias apenas para o desenvolvimento e
versionamento da aplicação.

| Software | Versão | Descrição | Licença |
|:----- |:-----:|:----- |:----- |
| Composer | 1.4.2 | Gerenciador de pacotes para PHP | [MIT
license](https://github.com/composer/composer/blob/master/LICENSE) |
| Git | 2.11.0 | Sistema de versionamento utilizado para manter organizadas as
versões e incrementos da aplicação. | [GNU General Public License version
2.0](http://opensource.org/licenses/GPL-2.0) |
| Git Flow | 1.10.2 | Complemento para o sistema de versionamento Git utilizado
para simplificar o trabalho com ramificações das versões. |
[BSD](https://github.com/nvie/gitflow/blob/develop/LICENSE) |
| VSCode | 1.23.1 | Editor de código. | [MIT
license](https://github.com/Microsoft/vscode/blob/master/LICENSE.txt). |
| Node.js | 8.11.1 | Plataforma para desenvolvimento de aplicações utilizando
Javascript. | [Licence](https://github.com/nodejs/node/blob/master/LICENSE) |
| Insomnia | 5.16.2 | Cliente para testes de APIs REST. |
[MIT](https://github.com/getinsomnia/insomnia/blob/develop/LICENSE) |

Além das tecnologias citadas acima, o desenvolvimento deste aplicativo utilizou o
serviço de repositórios do [GitHub](https://github.com).

Documentação
```

```
------------

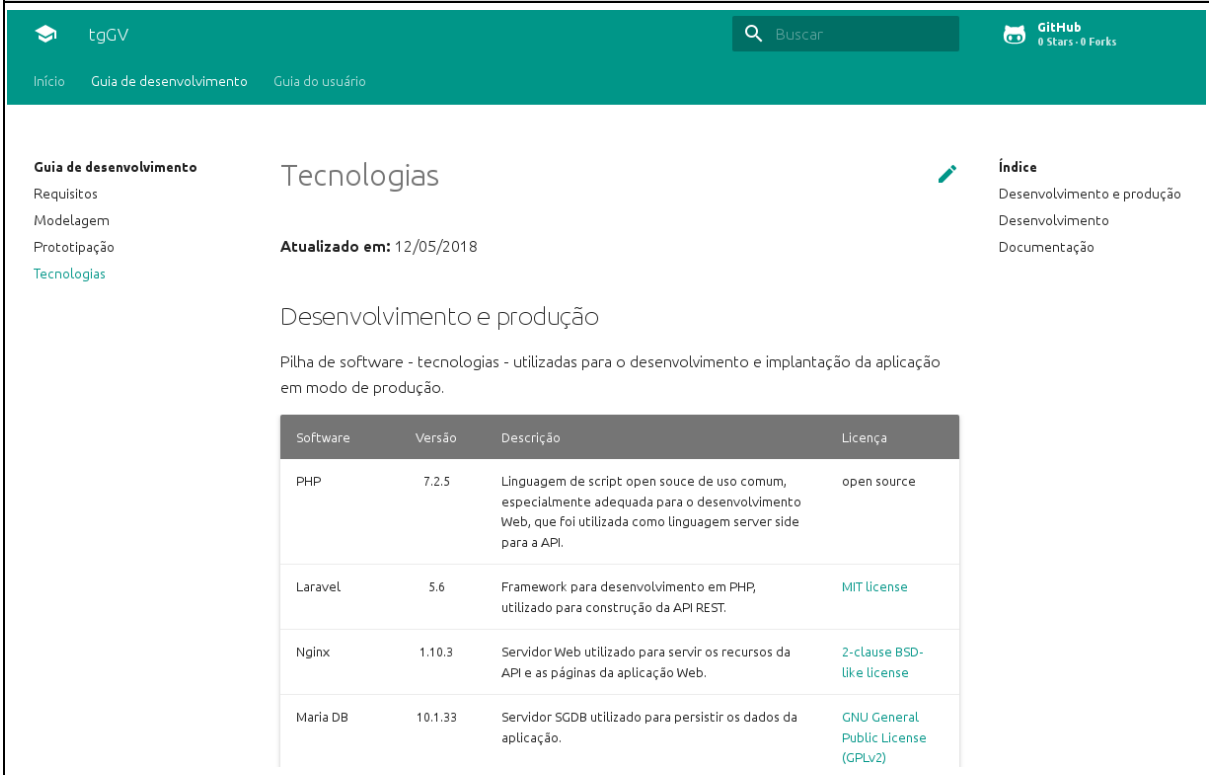
E por fim, as tecnologias utilizadas para produzir a documentação da aplicação.

| Software | Versão | Descrição | Licença |
|:----- |:-----:|:----- |:----- |
| Python | 2.7.13 | Linguagem de programação sob a qual as ferramentas de
conversão de Markdown para HTML foi escrita. | [PSF LICENSE AGREEMENT FOR PYTHON
2.7.15](https://docs.python.org/2.7/license.html).|
| pip | 10.0.1 | Ferramenta para instalar pacotes Python. | [MIT License
(MIT)](https://github.com/pypa/pip/blob/master/LICENSE.txt) |
| Python virtualenv | 15.2.0 | Ferramenta para criar ambientes Python isolados. |
[MIT license](https://github.com/pypa/virtualenv/blob/master/LICENSE.txt) |
| Python virtualenvwrapper | 4.8.2 | Conjunto de extensões para a ferramenta
Python virtualenv |
[Licence](https://virtualenvwrapper.readthedocs.io/en/latest/#license) |
| MkDocs | 0.17.2 | Ferramenta geradora de sites estáticos a partir de texto
estruturado em Markdown escrito em linguagem Python. | [MkDocs License
(BSD)](http://www.mkdocs.org/about/license/) |
| MkDocs Material | 2.6.6 | Extensão de tema visual para o MkDocs | [MIT
license](https://squidfunk.github.io/mkdocs-material/license/) |
| Draw.io | 8.6.3 | Ferramenta para criação de diagramas. | [Apache
v2](https://github.com/jgraph/drawio) |

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

Além das tecnologias citadas acima, foi utilizado o serviço [GitHub
Pages](https://pages.github.com/) para hospedar essa e as demais páginas da
documentação.
```

Página: Guia Desenvolvimento > Tecnologias



Arquivo: /mds/guia-usuario/1-desenvolvedor.md

```
Desenvolvedor
=============

**Atualizado em:** 13/05/2018

Este guia de usuário  para o desenvolvedor descreve o passo a passo para instalar
em um ambiente local o projeto tgGV – Orçamento Pessoal, a fim de possibilitar
que o usuário faça alterações no projeto.

Requisitos
----------

Para usar a aplicação, certifique-se de que possui instalado em seu computador as
[Tecnologias](../guia-desenvolvimento/4-tecnologias.md) descritas na página do
Guia de Desenvolvimento, de acordo com o uso que irá fazer.

Para rodar o código é imprescindível que tenha instalado:

+ [PHP 7.1.3 ou superior](http://php.net/manual/pt_BR/install.php "Link para
documentação do PHP")
+ [Composer](https://getcomposer.org/doc/00-intro.md#installation-linux-unix-osx
"Link para a documentação do Composer")
```

```
+ [Git](https://git-scm.com/download/linux "Link para a documentação do Git")
+ [Node.js](https://nodejs.org/en/download/package-manager/ "Link para a
documentação do Node.js")
+ Um gerenciador de banco de dados podendo ser:
    - MySQL ou Maria DB
    - PostgreSQL
    - SQLite
    - SQL Server

Para suportar o desenvolvimento da documentação é preciso ter instalado:

+ [Python 2.7 ou superior](https://www.python.org/downloads/ "Link para a
documentação do Python")
+ [pip](https://pip.pypa.io/en/stable/installing/ "Link para a documentação do
pip")
+ [Python virtualenv](https://pypi.org/project/virtualenv/ "Link para a página do
pacote no PyPI")
+ [Python virtualenvwrapper](https://pypi.org/project/virtualenvwrapper/ "Link
para a página do pacote no PyPI")
+ [MkDocs](http://www.mkdocs.org/#installation "Link para a documentação do
MkDocs")
+ [MkDocs - Material](https://squidfunk.github.io/mkdocs-material/getting-
started/ "Link para a documentação do MkDocs - Material")

Veja como [configurar o ambiente](1-
desenvolvedor.md#desenvolvendo_a_documentacao).

Instalação
----------

Com todos os requisitos para o desenvolvimento do projeto instalados, clone o
repositório que está armazenado no
[GitHub](https://github.com/getuliovinicius/trabalho.graduacao "Link para o
repositório").

```
$ git clone https://github.com/getuliovinicius/trabalho.graduacao.git
```

Após o término do clone, acesse o diretório `trabalho.graduacao`.

```
$ cd trabalho.graduacao
```

Para configurar o ambiente, renomeie o arquivo `.env.exemple` para `.env` e
edite-o ajustando os parâmetros conforme sua necessidade. Mais informações podem
ser obtidas em:
```

````
+ [Configuration – https://laravel.com/docs/5.6/configuration](https://laravel.com/docs/5.6/configuration "Link para a documentação do Laravel")
+ [Database Configuration – https://laravel.com/docs/5.6/database#configuration](https://laravel.com/docs/5.6/database#configuration "Link para a documentação do Laravel")

Neste documento, vamos assumir que tenha optado por uma instalação do [SQLite](https://sqlite.org/download.html "Link para a página de Donwload do SQLite") como gerenciador de banco de dados. Embora não tenha sido este o gerenciador utilizado no desenvolvimento do projeto, ele serve perfeitamente para a realização de uma instalação rápida do tgGV.

Crie um arquivo chamado `database.sqlite` no diretório `database`.

```
$ touch database/database.sqlite
```

Após, remova as linhas abaixo no arquivo `.env`:

```env
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=
DB_USERNAME=
DB_PASSWORD=
```

Então, adicione as duas linhas abaixo no lugar das linhas removidas anteriormente:

```env
DB_CONNECTION=sqlite
DB_DATABASE=/absolute/path/to/trabalho.graduacao/database/database.sqlite
```

Lembre-se de alterar `/absolute/path/to` pelo nome correto dos diretórios acima do diretório `trabalho.graduacao`.

Em seguida, use o comando `composer install` para fazer o download das dependências de back-end do projeto; o comando `php artisan key:generate` para gerar a chave de criptografia da aplicação; e o comando `php artisan migration` para criar as tabelas no banco de dados.

```
````
````

```
$ composer install
$ php artisan key:generate
$ php artisan migrate --seed
```

Além das tabelas para o banco de dados, o comando `php artisan migration --seed` cria:

+ o usuário `root`, que inicialmente terá como senha "123456" (calma, no futuro não será assim),
+ os papéis de usuário – tabela `roles`, e
+ o vínculo entre o usuário `root` e o papel `Super Usuário` na tabela `role_user`.

Para concluir a instalação do back-end, execute o comando `php artisan passport:install` para gerar as chaves de clientes para a criação de tokens para acesso a API.

```
$ php artisan passport:install
```

Copie o `CLIENT ID` e a `CLIENT SECRET` gerados para `Password grant` e insira-os no arquivo `.env` após o sinal de igual (sem espaços) nas variáveis `PASSWORD_CLIENT_ID` e `PASSWORD_CLIENT_SECRET`.

```env
PASSWORD_CLIENT_ID=
PASSWORD_CLIENT_SECRET=
```

Por fim, use o comando `npm install` para fazer o download das dependências de front-end do projeto, e o comando `npm run dev`, para gerar os arquivos `.css` e `.js` do front-end da aplicação.

```
$ npm install
$ npm run dev
```

Para subir a aplicação execute:

```
$ php artisan serve
```

```
Agora você pode [testar a API](1-desenvolvedor.md#uso_da_api) ou acessar a
aplicação web através do navegador: [http://localhost:8000](http://localhost:8000
"Link para a aplicação web").

Desenvolvendo a documentação
----------------------------

Com todos os requisitos para suporte ao desenvolvimento da documentação já
instalados, você pode editar os arquivos que são armazenados por padrão no
diretório `docs`.

Para executar o servidor web para visualizar o resultado é preciso configurar o
ambiente. Para isso, presumindo que já tenha instalado o Python, pip, virtualenv
e virtualenvwrapper, siga os seguintes passos:

Crie o ambiente virtual.

```
$ mkvirtualenv documentacao
```

Instale o MkDocs Material

```
$ pip install mkdocs-material
```

Todas as dependências, inclusive o MkDocs, serão instaladas automaticamente.

No diretório do projeto `trabalho.graduacao`, vá para o diretório `docs`.

```
$ cd docs
```

Então, execute:

```
$ mkdocs serve -a localhost:8080
```

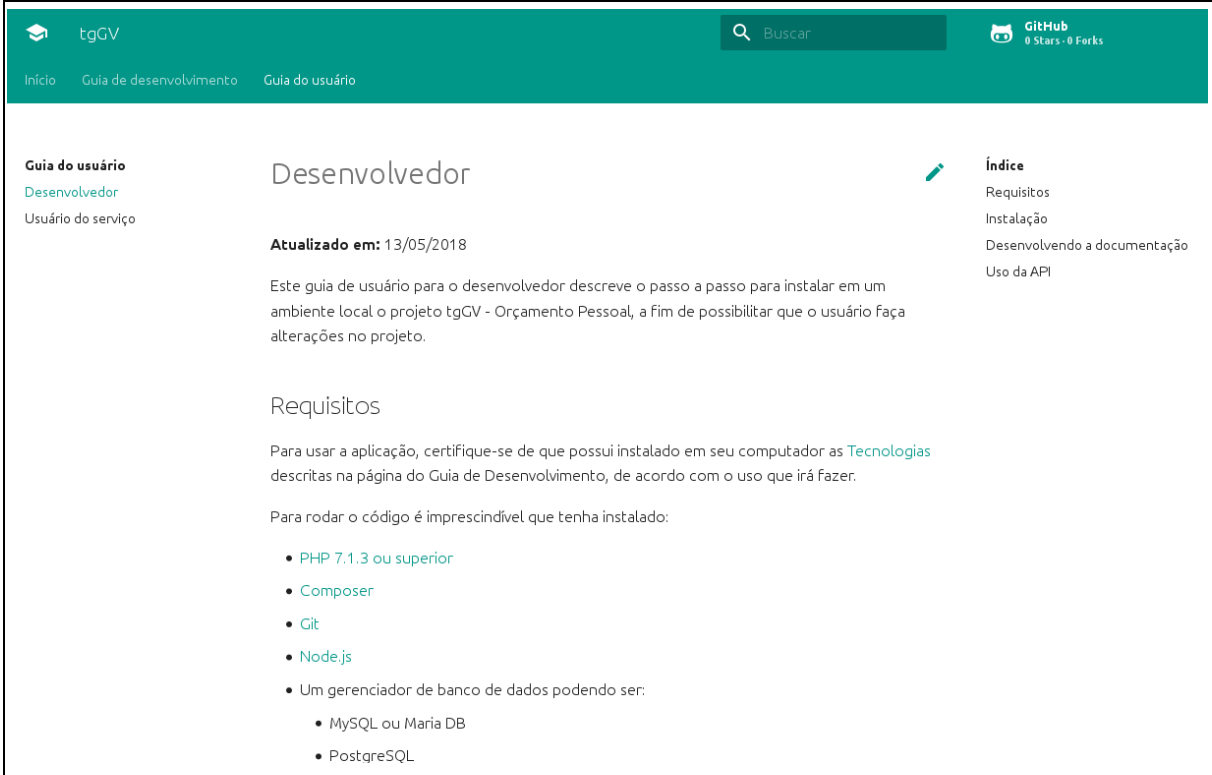
Agora você pode visualizar a documentação no navegador:
[http://localhost:8080](http://localhost:8080 "Link para a documentação do
projeto").

Uso da API

```

```
_**Observação:** Nessa sessão será documentado o modo de uso da API, para que outras aplicações de acesso possam ser desenvolvidas e integradas a aplicação.
```

Página: Guia do usuário > Desenvolvedor



Arquivo: /mds/guia-usuario/2-usuario-servico.md

```
Usuário do Serviço.
====================

**Atualizado em:** 13/05/2018

Esse guia do usuário do serviço descreverá o funcionamento da aplicação web para gestão de orçamento pessoal, demonstrando como o usuário poderá executar as funções de: cadastro no serviço, autenticação por usuário e senha, cadastro de categorias, cadastro de contras e lançamento de movimentações.

_**Observação:** A aplicação web ainda não está concluída, dessa forma ainda não foi escrita a documentação de uso do serviço._

Cadastro
--------

Login
-----
```

```
Categorias
----------

Contas
------

Movimentações
-------------
```

Página: Guia do usuário > Usuário do Serviço

APÊNDICE B

As figuras, apresentadas neste apêndice, representam a modelagem da aplicação cujo desenvolvimento foi iniciado para fins de estudo de caso deste TG.

Banco de Dados – Modelo Conceitual

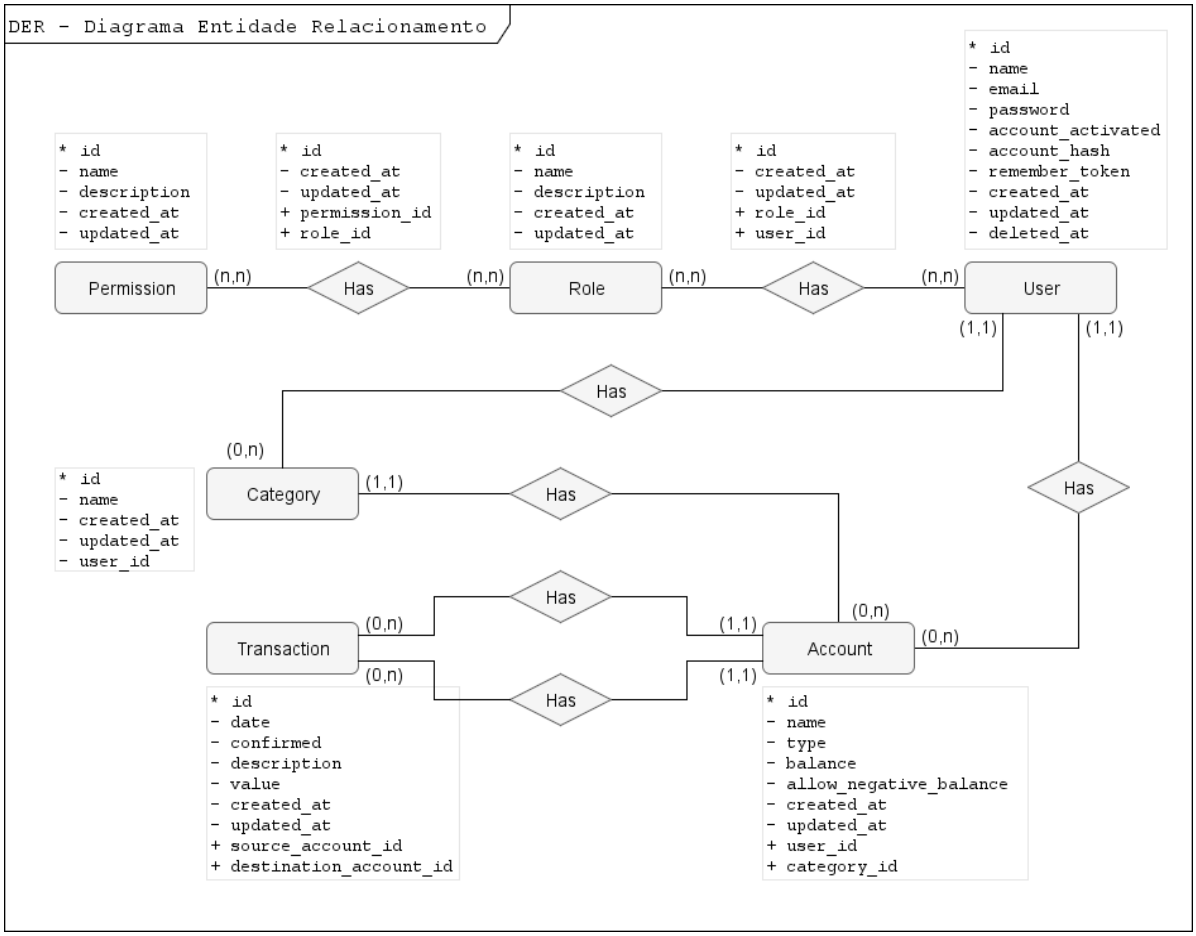


Diagrama de Caso de Uso 1

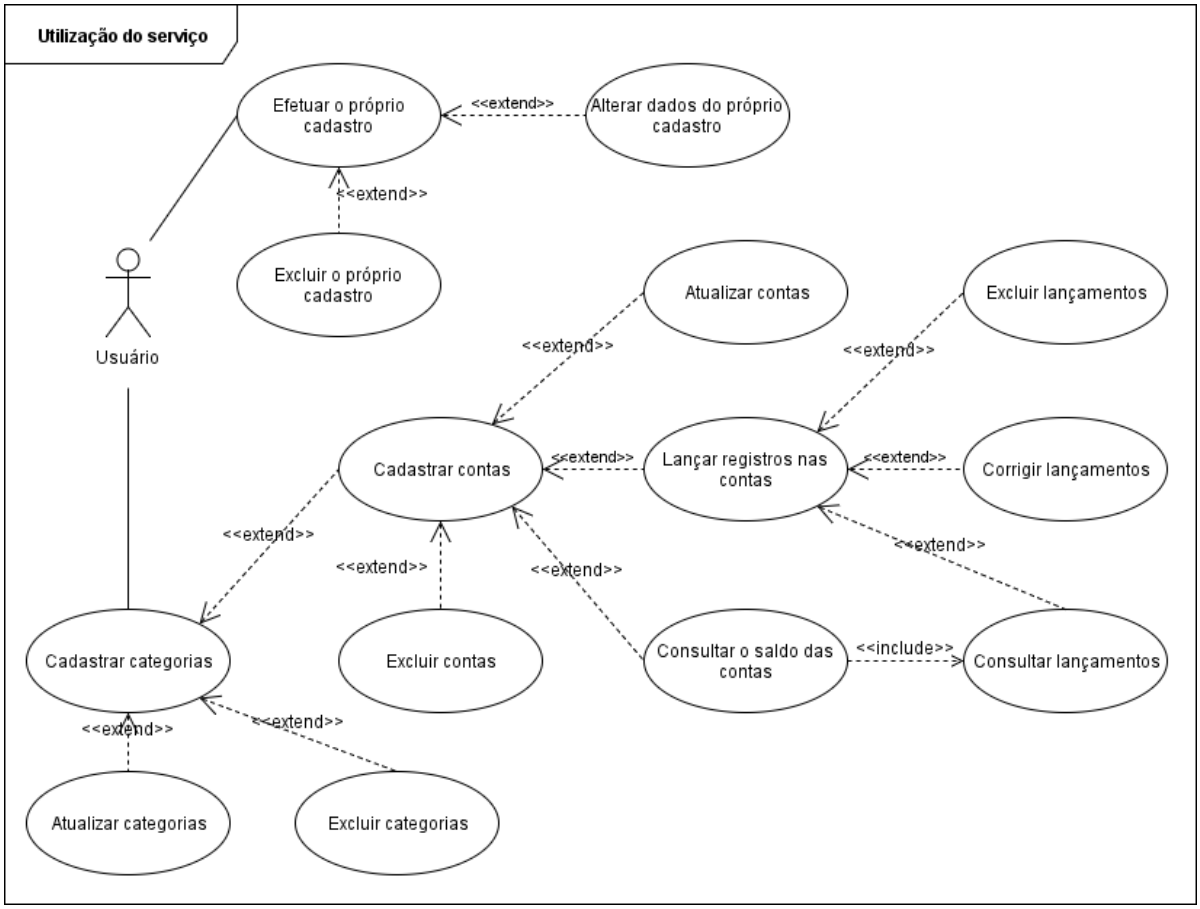
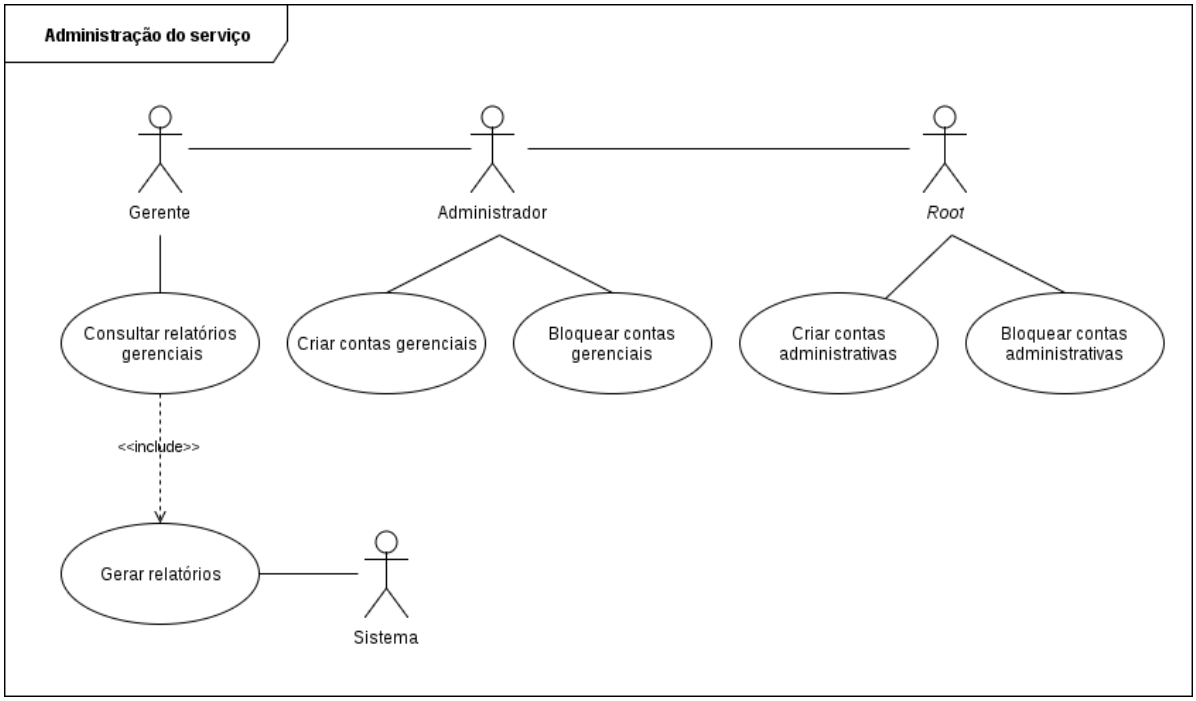
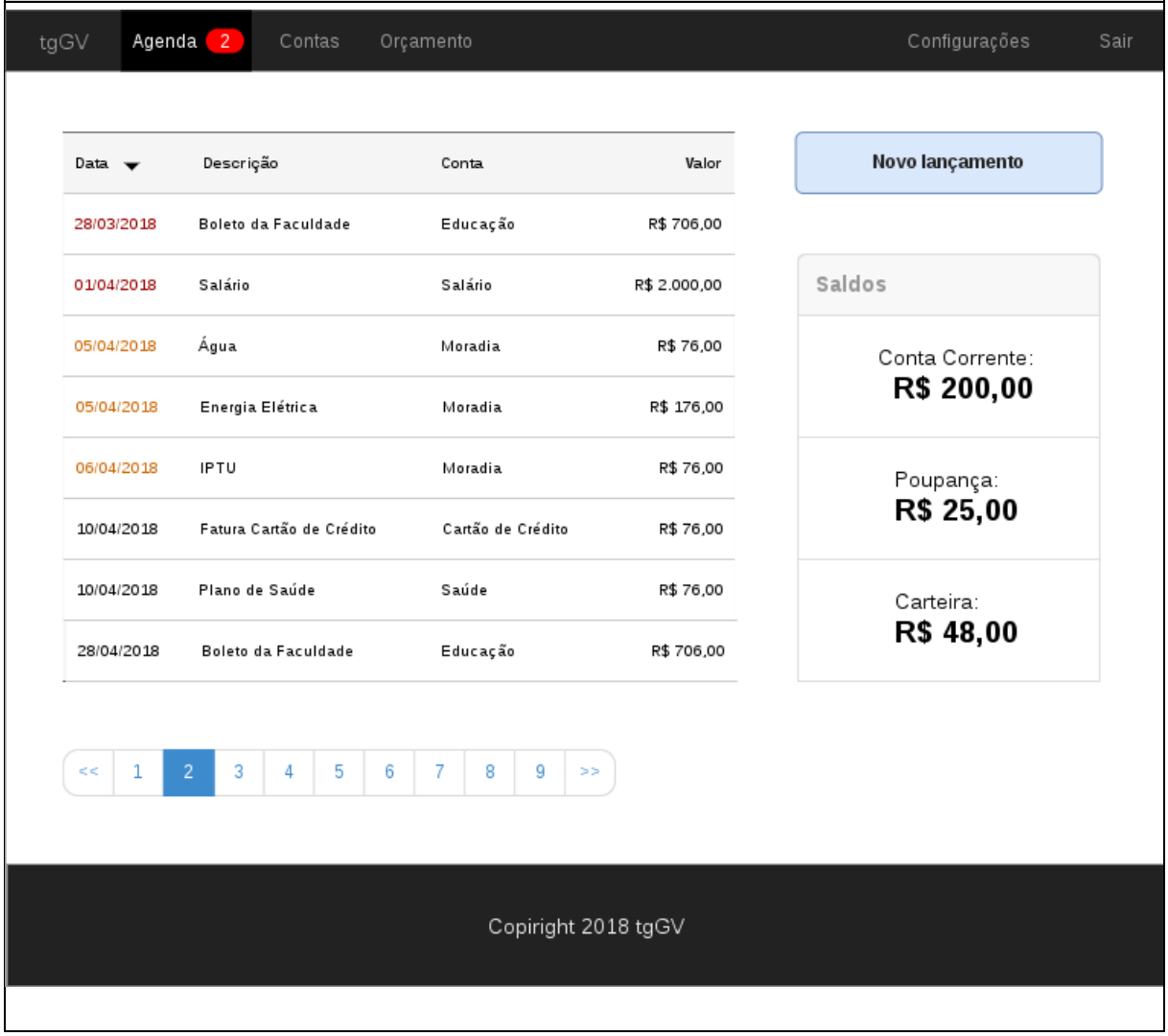


Diagrama de Caso de Uso 2



Tela 1 - Agenda



Tela 2 - Contas