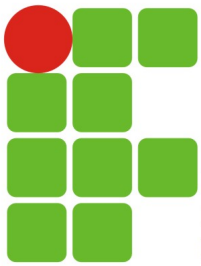


INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
RIO GRANDE DO NORTE

SPM: Uma Ferramenta para Gerenciamento de Projetos do GitHub Baseada em Técnicas de Mineração de Repositório de Software

Herlan Assis Pereira da Silva

Pau dos Ferros, Brasil

Julho, 2019

Herlan Assis Pereira da Silva

SPM: Uma Ferramenta para Gerenciamento de Projetos do GitHub Baseada em Técnicas de Mineração de Repositório de Software

Monografia apresentada para obtenção do título de Tecnólogo à banca examinadora do Curso de Análise e Desenvolvimento de Sistema, do Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte.
Orientador: Prof. Me. João Helis Junior de Azevedo Bernardo.

Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte

Campus Pau dos Ferros

Orientador: **João Helis Junior de Azevedo Bernardo**

Pau dos Ferros, Brasil

Julho, 2019

A minha mãe.

Agradecimentos

Em primeiro lugar, quero agradecer a minha mãe Alcileide Iracema de Assis por tudo que ela fez por mim, desde o meu nascimento até hoje, não existe nenhuma forma de descrever o quando sou grato a você. Eu te amo mãe.

Quero agradecer ao meu irmão Henzio Assis Pereira da Silva e a minha irmã Heloise Assis Pereira da Silva, por preocupar-se com o meu rendimento acadêmico, isso foi o gatilho motivacional para terminar esta graduação.

Agradeço ao meu orientador João Helis Junior de Azevedo Bernardo por me motivar, dia após dia, com palavras e gestos simples, porém poderosos, Obrigado!

Quero agradecer em especial a Leila Maria de Freitas Souza, por me cobrar todo dia a escrita deste trabalho, muito obrigado.

Quero agradecer aos meus professores e colegas de curso, vocês me proporcionaram uma vida acadêmica bastante animada.

Por último, agradeço a instituição de ensino a qual faço parte, o Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte campus Pau dos Ferros por me dar inúmeras oportunidades, me proporcionar grandes experiência e por me conectar com dezenas de pessoas maravilhosas.

“O verdadeiro homem mede a sua força quando se defronta com o obstáculo.”.

Saint-Exupéry, Antoine

Abstract

With the increasing demand for software and the need to constantly change the market to meet customer expectations, it is essential to use tools that help in the management of projects in order to meet deadlines and established goals. Researches on project management point out that the use of tools improves the quality of the product delivered to the client while applying effective methods of managing software projects. In this way, the present work had as objective to develop a free tool for the management of projects of GitHub, unifying the management of tasks, communication with the client and monitoring of the individual contribution of each developer of the project. A bibliographic study on the subject was carried out, with this it was possible to identify mining software repository techniques for extraction of relevant information on the individual contribution of each developer and project health. The results obtained with the construction of the tool were satisfactory, since it was possible to create a platform capable of unifying the management of tasks, communication with the client and analysis of the individual contribution of each collaborator of the project. With this, the project manager can use a tool free to help you deform useful to manage your software development team from a quantitative perspective.

Keywords: Software Project Management; Software Repository Mining; Analysis of Individual Contribution; Communication with the Customer.

Resumo

Com o crescente aumento da demanda de software e a necessidade do mercado manter-se em constante mudança para atender as expectativas dos clientes, torna-se essencial o uso de ferramentas que auxiliem na gestão de projetos a fim de cumprir os prazos e metas estabelecidas. Pesquisas sobre gestão de projetos apontam que a utilização de ferramentas melhoram a qualidade do produto entregue ao cliente quando aplicado métodos eficazes de gerenciamento de projetos de software. Desta forma, o presente trabalho teve como objetivo desenvolver uma ferramenta gratuita para o gerenciamento projetos do GitHub, unificando a gestão de tarefas, comunicação com o cliente e monitoramento da contribuição individual de cada desenvolvedor do projeto. Foi realizado um estudo bibliográfico sobre o assunto, com isso foi possível identificar técnicas de mineração de repositório de software para extração de informações relevantes sobre a contribuição individual de cada desenvolvedor e a saúde do projeto. Os resultados obtidos com a construção da ferramenta mostraram-se satisfatórios, pois foi possível criar uma plataforma capaz de unificar a gestão de tarefas, comunicação com o cliente e análise da contribuição individual de cada colaborador do projeto. Com isso, o gerente de projetos pode utilizar uma ferramenta gratuita para ajuda-lo de forma útil a gerenciar sua equipe de desenvolvimento de software sob uma perspectiva quantitativa.

Palavras-chave: Gestão de Projetos de Software; Mineração de Repositórios de Software; Análise da Contribuição Individual; Comunicação com o Cliente.

Lista de ilustrações

Figura 1 – Grupos de processos. 17
Figura 2 – Exemplo de quadro Kanban. 21
Figura 3 – Comparando *Containers* e Máquinas Virtuais. 31
Figura 4 – Diagrama de casos de uso. 36
Figura 5 – Adicionar nova tarefa. 37
Figura 6 – Enviar código de convite. 38
Figura 7 – Adicionar novo projeto. 38
Figura 8 – Habilitar notificações. 39
Figura 9 – Diagrama de entidade relacionamento. 40
Figura 10 – Diagrama de arquitetura. 42
Figura 11 – Tela de Login. 46
Figura 12 – Tela Principal. 46
Figura 13 – Estrutura da aplicação. 47
Figura 14 – Página **Home**. 48
Figura 15 – Página **Pesquisar**. 48
Figura 16 – "Caixa"com informações de um repositório. 49
Figura 17 – Repositório do projeto no Github. 50
Figura 18 – Página **Tarefas do Projeto**. 50
Figura 19 – Página **Contribuidores**. 51
Figura 20 – Página **Commits**. 51
Figura 21 – Página **Projetos Assistidos**. 52
Figura 22 – Página **Tarefas**. 52
Figura 23 – Página **Agenda**. 53
Figura 24 – Página **Relatórios**. 53
Figura 25 – Documentos gerados. 54
Figura 26 – Página **Notificações**. 54
Figura 27 – Códigos de monitoramento. 55
Figura 28 – Tarefas do Projeto. 55
Figura 29 – Inserindo uma nova tarefa. 56
Figura 30 – Detalhando uma tarefa. 56
Figura 31 – Inserindo uma nova nota. 57
Figura 32 – Tela de Login. 58
Figura 33 – Tela principal. 58
Figura 34 – Estrutura da aplicação. 59
Figura 35 – Página **Detalhes do Projeto**. 60
Figura 36 – Tabela de tarefas. 60

Figura 37 – Detalhes da tarefa. 61
Figura 38 – Inserindo uma nova nota. 61
Figura 39 – Editar uma tarefa. 62
Figura 40 – Página **Tarefas**. 62
Figura 41 – Página **Agenda**. 63
Figura 42 – Página **Notificações**. 63
Figura 43 – Bot Telegram **@MeninoDeRecado_Bot**: Início. 64
Figura 44 – Bot Telegram **@MeninoDeRecado_Bot**: Começar. 65
Figura 45 – Bot Telegram **@MeninoDeRecado_Bot**: Comandos. 66
Figura 46 – Bot Telegram **@MeninoDeRecado_Bot**: Notificação. 66

Lista de tabelas

Tabela 1 – Requisitos funcionais. 34
Tabela 2 – Requisitos não-funcionais. 35

Lista de abreviaturas e siglas

API	Application Programming Interface
MRS	Mineração de Repositório de Software
PDF	Portable Document Format
VM	Virtual Machine
UML	Unified Modeling Language
XP	Extreme Programming
XP	Extreme Programming
NBR	Norma Técnica Brasileira
ISO	International Organization for Standardization
IEC	International Electrotechnical Commission
LTS	Long Term Support
CPU	Central Processing Unit
RAM	Random Access Memory
VCS	Version Control System
HTML	Hypertext Markup Language
UI	User Interface
Bot	Roboot
WIP	Work in Progress
ABNT	Associação Brasileira de Normas Técnicas
CI	Continuous Integration

Sumário

1 INTRODUÇÃO 13
1.1 Objetivos 14
1.1.1 Objetivo Geral 14
1.1.2 Objetivos Específicos 14
1.2 Estrutura do Trabalho 15

2 REFERENCIAL TEÓRICO 16
2.1 Gerenciamento de Projetos de Software 16
2.2 Manifesto Ágil 18
2.3 Mineração de Repositório de Software 22
2.3.1 Métricas para análise da contribuição individual de desenvolvedores 23
2.4 Trabalhos relacionados 25
2.5 Considerações do Capítulo 26

3 METODOLOGIA 27

4 MODELAGEM DA FERRAMENTA PROPOSTA 29
4.1 Visão Geral da SPM 29
4.2 Tecnologias Utilizadas 29
4.2.1 Google Cloud Platform 30
4.2.2 Contêineres Linux 30
4.2.3 Versionamento de código 32
4.2.4 Ferramentas 32
4.3 Requisitos 33
4.4 Diagramas 35
4.4.1 Diagramas UML 35
4.4.1.1 Diagrama de casos de uso 35
4.4.1.2 Diagrama de Atividades 37
4.4.2 Diagrama de entidade relacionamento 39
4.4.3 Diagrama de arquitetura 41
4.5 Considerações do Capítulo 44

5 APRESENTAÇÃO DA APLICAÇÃO 45
5.1 Manager 45
5.2 Client 57

5.3 MeninoDeRecado_Bot 63
5.4 Considerações do Capítulo 67

6 CONCLUSÃO 68
6.1 Trabalhos Futuros 68

REFERÊNCIAS 70

1

Introdução

O aumento da competitividade no âmbito do desenvolvimento de software tem impulsionando a adoção de novas técnicas e métodos de gerência de projetos por parte das empresas deste setor, a fim de manterem-se ativas dentro do mercado. Nesse contexto, a informação vem tornando-se um ativo cada vez mais valorizado no processo decisório destas organizações, uma vez que essa embasa de forma efetiva decisões de projeto de cunho estratégico e operacional.

No âmbito do desenvolvimento de software, a utilização de ferramentas **dashboards** que apresentam informações relevantes de projetos de software para gerentes e desenvolvedores, bem como, que proveem meios eficientes de comunicação com o cliente, podem ser consideradas um importante diferencial competitivo para as organizações. De acordo com Costa et al. (2014), é crucial que gerentes de projetos utilizem ferramentas que provejam informações adequadas sobre as contribuições individuais dos desenvolvedores que coordenam, possibilitando, de forma proativa, melhorar o processo de gerenciamento destes projetos.

A coleta manual de métricas de contribuição de desenvolvedores, por sua vez, é caracterizada por um alto consumo de tempo e pode elevar os custos do projeto (COSTA et al., 2014). Alternativamente, sistemas de controle de versão como Github, sistemas de comunicação (E.g. Slack e Telegram), e-mails, bancos de dados, logs, dentre outros repositórios, guardam diariamente ações desempenhadas pelos desenvolvedores, bem como dados valiosos sobre a evolução dos projetos (HASSAN, 2006). A transformação dos inúmeros dados contidos nestes repositórios em informações úteis tem sido o foco da disciplina de Mineração de Repositório de Software (MRS) nos últimos anos. Como exemplo, pode-se citar os trabalhos de Wang, Lo e Jiang (2013), Zhang e Lee (2013) e Bernardo, Costa e Kulesza (2018), que utilizam métricas de MRS para criar ferramentas que coletam e apresentam, de forma automatizada, informações gerenciais de projetos de software.

Apesar de já existirem diversas ferramentas disponíveis para gerência de projetos, em especial projeto de software, um pequeno número de gerentes de projeto ainda hesitam em adotar tais tecnologias. Alguns dos motivos pelos quais estes gerentes ainda apresentam resistência em incorporar ferramentas desse gênero ao seu

ambiente de trabalho estão relacionados com o seu custo elevado de aquisição, alta complexidade e a dificuldade em atender às suas necessidades, sendo necessário utilizar várias ferramentas simultaneamente (E.g. ferramenta de captura de métricas de contribuição dos desenvolvedores, ferramenta de comunicação e ferramenta de gerenciamento de tarefas).

1.1 Objetivos

Nesta sessão será discutido os objetivos gerais e específicos deste trabalho monográfico.

1.1.1 Objetivo Geral

Este trabalho tem o objetivo de desenvolver uma plataforma gratuita de gerenciamento em tempo real de projetos do GitHub, que possibilite o gerenciamento de tarefas da equipe, o monitoramento da contribuição individual dos desenvolvedores por meio de métricas de Mineração de Repositório de Software, e o estreitamento da comunicação da equipe de desenvolvimento com o(s) cliente(s) do projeto.

1.1.2 Objetivos Específicos

Dentre os objetivos específicos deste trabalho, pode-se citar:

- Realizar um levantamento bibliográfico sobre técnicas de mineração de repositório de software (MSR) que possam ser uteis para recuperar informações sobre as atividades desempenhadas por um desenvolvedor em um dado intervalo de tempo;
- Realizar uma exploração bibliográfica sobre métodos e técnicas eficientes para o gerenciamento de tarefas em um projeto de software;
- Realizar uma pesquisa sobre ferramentas atuais que possam auxiliar na comunicação entre a ferramenta de gerenciamento de projetos proposta, e os clientes envolvidos em um determinado projeto;
- Apresentar uma abordagem metodológica que permita apoiar o desenvolvimento da solução proposta;
- Pesquisar tecnologias que suportem o desenvolvimento da aplicação;
- Estruturar e implementar a abordagem proposta;

- Mostrar resultados satisfatórios gerados pela ferramenta, utilizando dados de um projeto real do GitHub.

1.2 Estrutura do Trabalho

Este trabalho está organizado em seis capítulos. Além do capítulo introdutório, o Capítulo 2 apresenta a fundamentação teórica utilizada para embasar o desenvolvimento do sistema. O Capítulo 3 retrata a metodologia utilizada na concepção deste trabalho. O Capítulo 4 descreve o desenvolvimento do sistema, apresentando os seus principais diagramas UML, além de abordar as tecnologias e ferramentas que foram utilizadas na sua construção. No Capítulo 5, o sistema é apresentado com o uso de imagens e descrição de suas funcionalidades. Por fim, no Capítulo 6, é anunciado as principais conclusões e planos para trabalhos futuros.

2

Referencial Teórico

Este capítulo elenca os principais conceitos e definições utilizadas para embasar teoricamente o desenvolvimento da ferramenta resultante deste trabalho monográfico.

2.1 Gerenciamento de Projetos de Software

Atualmente, o mercado de software tem se caracterizado como um ambiente altamente competitivo. As empresas, há pouco tempo, perceberam que o ciclo do produto deve iniciar a partir do consumidor, e com esse, deve-se estabelecer um relacionamento direto e intenso (ADOLPHO, 2011), buscando-se efetividade na construção do produto, que deve atender claramente as expectativas dos seus usuários finais.

A alta competitividade e as demandas do mercado exigem veementemente que os produtos de software sejam entregues dentro da qualidade e prazo pré-definido, bem como, dentro do orçamento previsto. De acordo com Sommerville (2003 apud BROOKS, 1975), percebeu-se no final da década de 60 e início da década de 70 que grandes projetos de software eram frequentemente entregues com atrasos, desempenho inferior ao esperado e custando várias vezes o valor estipulado no seu planejamento, o que culminou com um período denominado dentro da Engenharia de Software como: “Crise do Software”.

De acordo com Sotille (2014), a má gestão de um projeto de software se caracteriza como o principal fator para o seu fracasso. Neste sentido, para que se pudesse superar a Crise do Software, a Engenharia de Software passou a criar uma série de processos, metodologias, ferramentas e técnicas, cujo objetivo estava ligado ao empoderamento do processo de gerenciamento de projetos de software, de forma a garantir a efetividade do desenvolvimento desses projetos.

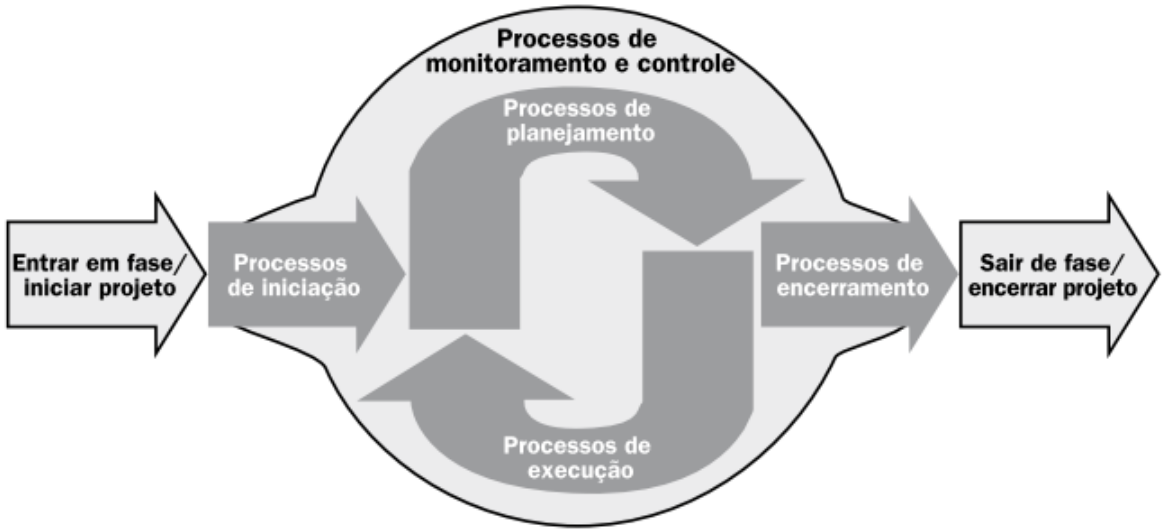
Um projeto é uma atividade com inicio e fim definidos, que tem como propósito gerar um produto, serviço ou resultado único. Um projeto pode envolver uma ou várias pessoas ou organizações. Além do processo de criação de produtos e serviços, projetos também podem ser utilizados para estabelecer melhorias a um determinado produto ou serviço, de forma que seja possível potencializar a capacidade de realizar

um serviço, linhas de produto ou um resultado único (PMBOK, 2012).

Com o conceito de projeto definido, podemos falar sobre o gerenciamento de projetos. Segundo PMBOK (2012, p. 5), o gerenciamento de projetos é "a aplicação do conhecimento, habilidades, ferramentas e técnicas às atividades do projeto para atender aos seus requisitos". Os processos utilizados no gerenciamento de projetos geralmente são organizados em cinco grupos, que são: iniciação, planejamento, execução, monitoramento e controle, e encerramento.

A Figura 1 ilustra o ciclo de interação entre os processos de gerenciamento de projetos. Normalmente o gerente de projeto assume a responsabilidade de elaborar propostas, planejar, definir custos, monitorar, selecionar e avaliar os desenvolvedores, além de criar relatórios e apresentações (SOMMERVILLE, 2003).

Figura 1 – Grupos de processos.



Fonte: (PMBOK, 2012).

No que concerne o gerenciamento de projetos de software, é necessário garantir produtividade nas equipes que compõem tais projetos, para isso é importante avaliar a maior quantidade de dados disponíveis para obter informações sobre custo, tempo e recursos utilizados. Desta forma, a mineração de informações sobre produtividade de equipes de software e de desenvolvedores individuais tem se caracterizado como uma ferramenta importantíssima para o gerenciamento de projetos desta natureza, uma vez que possibilita o melhor planejamento da ações, bem como, possibilita agir proativamente com base em dados reais do projeto. De acordo com (DEMARCO, 1982, p. 3), "Não se pode controlar o que não se pode medir".

Durante o desenvolvimento de um projeto de software é essencial que haja comunicação periódica com o cliente, tornando possível a sua colaboração com o

desenvolvimento do projeto. Caso não seja possível trabalhar presencialmente com ele, é imprescindível a utilização ou criação de ferramentas que auxiliem nesse processo (RAYMUNDO; LACERDA,).

Levando-se em consideração as fragilidades encontradas nas atividades de gestão de projetos de software, ainda pode-se elencar a comunicação da equipe de desenvolvimento e demais *stakeholders* como uma das principais causas para que projetos de software não obtenham êxito. Alinha-se a isso ao uso de técnicas incorretas de gerenciamento de projetos (SOMMERVILLE, 2003; SOTILLE, 2014). O bom gerenciamento de um projeto não garante o seu sucesso, por outro lado, o mal gerenciamento quase sempre resulta no fracasso (SOMMERVILLE, 2003).

Os processos de software, bem como as metodologias ágeis, são esforços da engenharia de software para garantir que o desenvolvimento de projetos de software atendam o prazo, custo e qualidade esperada. Neste sentido, o bom gerenciamento de projetos de software deve utilizar-se de processos de software, uma vez que estes utilizam de etapas e iterações bem definidas, fazendo uso de técnicas que perpassam todo o ciclo de desenvolvimento de um produto de software. Além disso, os processos de software determinam estratégias para o estabelecimento de interações com os clientes envolvidos no projeto de software e, com isso, buscam a eficácia em entregas de produtos que agreguem o máximo de valor ao negócio desses clientes.

2.2 Manifesto Ágil

No contexto da engenharia de software, um processo de software não se estrutura como uma determinação hirta e imutável sobre como desenvolver um software. Na realidade, o processo é passível de modificações a depender da sua finalidade, tornando possível a utilização de apenas um conjunto de tarefas e práticas deste processo, levando em consideração o contexto de desenvolvimento no qual o projeto de software está inserido. Segundo (SOMMERVILLE, 2003), o processo de software pode ser descrito como um conjunto de atividades relacionadas que fomentam a produção de um produto de software. Adicionalmente, Sommerville (2003) evidencia que apesar da existência de vários processos de software, com suas próprias práticas e definições, quatro atividades se destacam por incorporar todo e qualquer processo de software, são elas:

- **Especificação de software** – As funcionalidades e restrições a respeito do funcionamento do software devem ser definidas;
- **Projeto e implementação do software** – O software deve ser produzido para tentar atendes às especificações definidas;

- **Validação do software** – O software deve ser validado para verificar se atende às demandas do cliente;
- **Evolução do software** – O software deve evoluir à medida que as necessidades do cliente mudam;

Os processos de software são categorizados como dirigidos a planos ou processos ágeis. No primeiro caso, todas as atividades são previamente definidas e o progresso é avaliado a partir da comparação com o projeto inicial. No caso dos processos ágeis o planejamento é estruturado de forma gradativa. A representação do processo de software é feita através do modelo de processo de software, em que cada modelo externa uma perspectiva particular a respeito do processo e, dessa forma, fornece informações de forma parcial.

No cenário atual, as empresas, que atuam em um ambiente global, lidam com mudanças rápidas e sucessivas, o que demanda softwares que possam ser desenvolvidos rapidamente para lograrem vantagens sobre as oportunidades e responder às pressões do mercado. Dessa forma, a abordagem de desenvolvimento dirigido a planos, quando aplicada aos sistemas corporativos de pequeno e médio porte, acaba por gerar um *overhead* tão grande que compromete o processo de desenvolvimento do software. Gasta-se mais tempo realizando análises acerca do desenvolvimento do sistema do que no desenvolvimento e testes do programa (SOMMERVILLE, 2003).

Com essas mudanças persistentes é quase impossível delimitar todos os requisitos de um software, os requisitos inicialmente definidos serão alterados de forma inevitável e imprevisível. Nesse sentido, objetiva-se o desenvolvimento rápido de software, o que torna inviável a utilização de processos de software que visam a completa especificação de requisitos e, em seguida, projetar, construir e testar (SOMMERVILLE, 2003).

Em fevereiro de 2001, metodologistas leves, terminologia utilizada para definir pessoas que eram adeptas a adoção de metodologias mais flexíveis de desenvolvimento de software, em detrimento às metodologias tradicionais, se reúnem no estado de Utah para discutir ideias em comum. A partir desse encontro surgiu o Manifesto Ágil, que tem como objetivo priorizar indivíduos e interações ao invés de processos e ferramentas, software funcional ao invés de documentação extensa, colaboração do cliente no lugar de contrato, e resposta rápida a mudanças mesmo depois de um plano já definido (MANIFESTO, 2001).

As metodologias ágeis tem prioridades bem definidas, de forma que a interação, colaboração, entrega de resultados concretos e mudanças rápidas devem ser priorizadas para garantir a entrega de valor ao cliente.

Na criação do manifesto ágil, os membros refinaram suas ideias em doze princípios, os quais estão descritos a seguir:

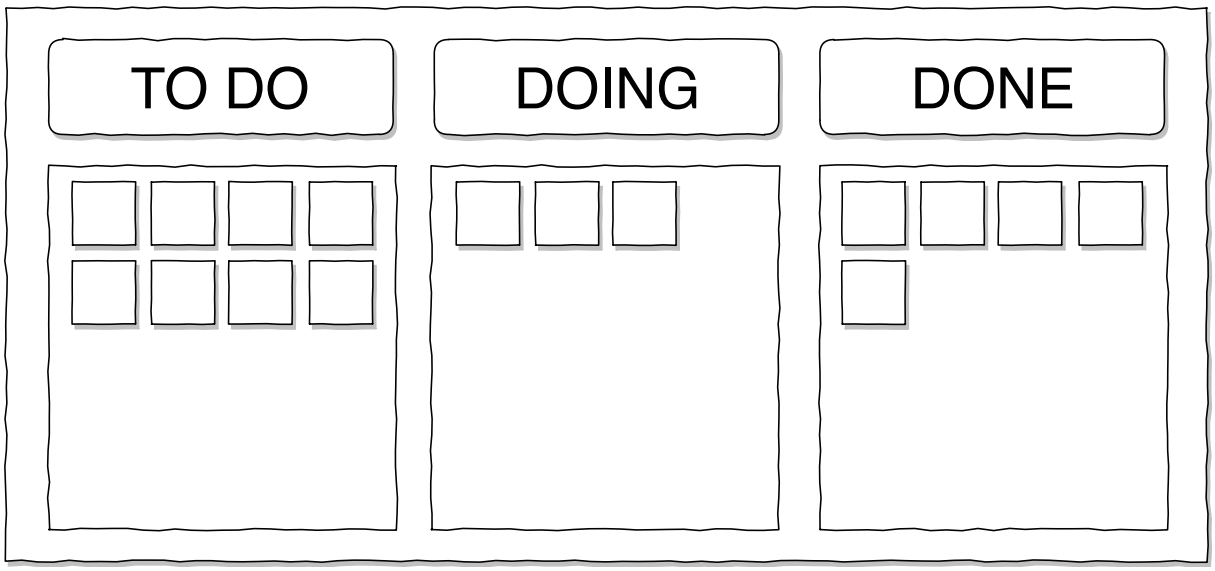
- Satisfazer o cliente através de entregas contínuas;
- As mudanças de escopo são aceitas, mesmo no final do projeto;
- Entrega de resultados frequentes com intervalos de alguns meses ou semanas, preferindo-se pela menor escala de tempo possível;
- Todos os envolvidos no projeto devem colaborar juntos durante todo o tempo de desenvolvimento;
- Trabalhe com indivíduos motivados e dê a eles total suporte para realizar o seu trabalho;
- A melhor maneira de trocar informações dentro de uma equipe é através de conversas cara a cara;
- O resultado entregue é a principal medida de progresso;
- Os envolvidos devem manter um ritmo constante de desenvolvimento a fim de garantir a sustentabilidade do projeto;
- A atenção nos detalhes técnicos do projeto e o bom design aumenta a agilidade;
- Simplifique e não complique;
- Equipes auto-organizadas garantem as melhores arquiteturas, requisitos e projetos;
- Reuniões periódicas sobre resultados e comportamentos devem ser realizadas com o objetivo de refletir sobre as ações e melhorar os resultados obtidos.

De acordo com Sommerville (2011), o Scrum é um dos métodos ágeis mais conhecidos cujo foco norteia o gerenciamento do desenvolvimento iterativo. Esse método não prescreve o uso de práticas de programação, como programação em pares e o desenvolvimento *test-first*, possibilitando sua utilização vinculada a abordagens ágeis mais técnicas, como a *extreme Programming* (XP), que provê um framework de gerenciamento do projeto.

No Scrum destacam-se três fases: a primeira constitui o planejamento geral, no qual os objetivos gerais do projeto e da arquitetura do software são estabelecidos; a segunda é marcada por uma série de ciclos de *sprint* – unidade de planejamento na qual o trabalho a ser feito é avaliado –, em que cada ciclo determina o desenvolvimento de um incremento do sistema; a última fase demarca o encerramento do projeto,

completa a documentação exigida e pondera acerca das lições aprendidas com o projeto.

Figura 2 – Exemplo de quadro Kanban.



Fonte: O Autor.

No cenário de desenvolvimento ágil, além dos métodos ágeis supracitado (Scrum e XP), o Kanban se mostra uma ferramenta útil por acompanhar o fluxo de trabalho e o trabalho em desenvolvimento. O quadro Kanban é dividido, normalmente, em *to do* (a fazer), *doing* (fazendo) e *done* (feito), exemplificado na Figura 2, permitindo uma melhor visualização sobre o que está em andamento e, dessa forma, permite uma limitação sobre trabalho em progresso, pois expõe variabilidade e desperdícios. A divisão *doing* pode ser fragmentada, estabelecendo etapas, para melhor permitir o acompanhamento do seu progresso (BENZECRY, 2017).

Segundo Mariotti (2012), um quadro representando o sistema Kanban pode conter as etapas: análise, desenvolvimento, aceitação e implantação. Esse modelo tem, como linha de produção, a regra de limitar o processo em andamento (WIP). Cada coluna terá um WIP estabelecido e representados pelo número máximo de cartões em cada fase. O cartão é constituído por uma breve história do usuário, descrevendo seus requisitos. Todo cartão entra na fila de *backlog* e aguarda a liberação de capacidade para ser inserido nas colunas seguintes. Quando as atividades relacionadas ao cartão na coluna em andamento são finalizadas, o mesmo é passado para a coluna seguinte, liberando espaço para a entrada de um novo cartão. A priorização para inicialização dos cartões deve ser coerente com as exigências e estratégias estabelecidas para o projeto.

O uso do sistema Kanban, então, tem como base a preparação e limitação do trabalho em andamento para uma capacidade suportada pela equipe de desenvolvimento, proporcionando o equilíbrio entre a demanda da equipe e seu rendimento,

possibilitando, dessa forma, uma melhor produção. No processo de desenvolvimento de software, a implementação do modelo Kanban resulta em códigos de alta qualidade, ciclo de desenvolvimento curto, e controle do desempenho de produção (MARIOTTI, 2012).

Além do uso de metodologias ágeis, técnicas de mineração de repositório de software podem ser utilizadas para acompanhar e monitorar a evolução do projeto. Através delas, é possível obter informações sobre o seu progresso, sua estabilidade e o desempenho de cada colaborador.

2.3 Mineração de Repositório de Software

A mineração de repositório de software (MSR) é um conjunto de técnicas utilizadas para extrair dados pertinentes à evolução de um projeto de software. Esses dados podem ser extraídos, por exemplo, a partir de sistemas descentralizados de controle de versão como o Git ou SVN, Issues Trackers, e-mails, chats, dentre outros repositórios utilizados no processo de desenvolvimento de software (MEIRELLES, 2013).

De acordo com Hassan (2008 apud LIMA, 2014), esses repositórios têm sido utilizados pelos pesquisadores da área de MSR, com base em dois propósitos principais, que consistem em elaborar técnicas para automatizar e melhorar a extração de dados dos repositórios, e também, descobrir e validar novas técnicas e abordagens para minerar informações importantes de tais repositórios.

Com a popularização dos sistemas descentralizados de controle de versão, a área de Mineração de Repositório de Software tem se proposto, de forma cada vez mais frequente, a analisar os dados tanto de contribuidores individuais, como também de equipes de software. Isso porque estes sistemas online como GitHub, BitBucket e GitLab tem sido amplamente utilizados pelas equipes de software para o versionamento dos seus projetos, tanto Open-Source quanto projetos privados, e, atrelado a isso, tais plataformas disponibilizam sofisticadas APIs de acesso aos seus dados, o que facilita o processo de mineração.

A normativa NBR ISO/IEC 9126-1 da Associação Brasileira de Normas Técnicas (ABNT) define um conjunto de parâmetros para a análise e avaliação da qualidade de um produto de software, uma forma de medir é através de métricas de software. Uma métrica é a definição de métodos e medidas para a avaliação da qualidade do software (ABNT, 2003).

Medir um software é necessário para obter informações sobre qualidade, produtividade e estimativas (custos, prazos e recursos). O gerente de projeto deve garantir ao cliente entregas regulares assim como metas e prazos cumpridos rigorosamente.

Além disso, o software é um produto intangível, ou seja, não pode ser visto ou tocado; para examinar o progresso do software o gerente depende de artefatos produzidos pelos desenvolvedores (SOMMERVILLE, 2003).

Ferramentas que fornecem informações visuais, como gráficos e tabelas, auxiliam os gerentes de softwares na tomada de decisão. Esses softwares, com telas intuitivas ou *dashboards*, ajudam no monitoramento do desempenho das suas equipes e dos seus projetos. Essas telas fornecem ferramentas que facilitam a análise de dados, seja ela por período de tempo ou outra variável, que possa ser utilizada como parâmetro de medida.

2.3.1 Métricas para análise da contribuição individual de desenvolvedores

A fim de estabelecer critérios claros e objetivos para mensurar a evolução de um projeto de software, bem como, para verificar a contribuição individual de cada desenvolvedor de uma equipe de software no desenvolvimento de um determinado projeto, estabeleceu-se dentro da literatura uma série de métricas de software. A exemplo do trabalho de Costa (2013), que propôs a avaliação da contribuição dos desenvolvedores de uma equipe de software com base na qualidade e tamanho dos *commits* e sob a ótica da prioridade dos *bugs* solucionados. No tratamento de qualidade avalia-se a capacidade dos *commits* em gerar *bugs*. No ponto de vista de tamanho, utilizam-se métricas que verificam a quantidade de linhas de código, métodos e número de classes. Sob essa perspectiva do tamanho dos *commits*, as seguintes métricas podem ser levadas em consideração:

- **Quantidade de linhas adicionadas:** quantifica o número de linhas de código adicionadas por cada desenvolvedor. As linhas possuem o símbolo "+" na descrição de alteração do *commit*;
- **Quantidade de linhas removidas:** quantifica o número de linhas de código removidas por cada desenvolvedor. As linhas possuem o símbolo "-" na descrição de alteração do *commit*;
- **Classes adicionadas:** classes adicionadas a cada *commit*;
- **Classes removidas:** classes removidas por *commit*;
- **Classes modificadas:** classes modificadas pelo *commit*;
- **Métodos adicionados:** métodos completos adicionados no *commit*;

- **Métodos modificados:** métodos modificados pelo *commit*, havendo adições ou remoções de linhas no método;

Dessa forma, a quantidade de *commits* é proporcional aos valores extraídos pelas métricas. No tratamento de *bugs*, a contribuição dos desenvolvedores é avaliada com base na resolução de *bugs* de prioridade urgente, alta e média.

O número de linhas adicionadas e removidas ou quantidade de *commits* feitos em um determinado período de tempo revelam dados sobre a saúde do projeto, em especial a estabilidade do código fonte (MEIRELLES, 2013).

Nesta monografia, a avaliação utilizada para medir a contribuição individual do desenvolvedor foi o ***churn*** – métrica de software obtida pela soma da quantidade de linhas removidas e adicionada no *commit*. Sendo assim, a análise de contribuição está sendo monitorada pelo ponto de vista de tamanho.

Na proposição feita por Lima (2014), torna-se possível ao gerente de software averiguar valores baixos, moderados ou altos para a contribuição dos desenvolvedores com base no volume de contribuição, complexidade média por método, introdução a *bugs* e correção destes. No que tange o volume de contribuição, a análise é feita com base em seis métricas: quantidade de linhas de código adicionadas (desconsiderando comentários e linhas em branco), quantidade de linhas de código removidas, quantidade de linhas de código modificadas, quantidade de métodos adicionados, quantidade de métodos removidos e quantidade de métodos modificados.

Para averiguar a complexidade dos métodos, Lima (2014) fez uso de duas métricas:

- **Complexidade média em métodos modificados:** somatório das diferenças dos valores da complexidade antes da modificação e depois da modificação, dividido pela quantidade de métodos modificados pelo desenvolvedor no período;
- **Complexidade média dos métodos adicionados:** somatório da complexidade dos métodos adicionados pelo desenvolvedor no período, dividido pela quantidade de métodos adicionados pelo desenvolvedor no referido período;

Na introdução de *bugs*, a avaliação de qualidade de contribuição é realizada com base na quantificação de *bugs* incluídos pelo desenvolvedor. A medição desse atributo é parametrizado por duas métricas:

- ***Commits* que introduziram *bugs***: quantidade de *commits* realizados por um desenvolvedor em que suas alterações acarretam em correções posteriormente;

- **Falha em testes:** quantidade de vezes que tarefas retornaram para desenvolvimento por ventura de falha em testes.

O trabalho de Lima (2014) ainda fez uso da métrica de quantificação de *logs* de desenvolvimento para correção de *bugs*, que consiste na quantidade de *logs* de desenvolvimento em tarefas cujo objetivo é a correção de *bugs* de cada desenvolvedor, num dado período, dividido pela quantidade de *logs* de desenvolvimento em tarefas de correção de erro dos desenvolvedores de sua equipe. Tal análise é vinculada ao atributo de contribuição em correção de *bugs*.

2.4 Trabalhos relacionados

Esta sessão compara este trabalho com outras pesquisa que tem como propósito resolver ou amenizar problemas relacionados a desempenho, comunicação ou análise da contribuição individual.

O trabalho de Lima (2014) propõe uma abordagem de apoio à gerencia de projetos para acompanhamento da contribuição de desenvolvedores com base na extração de métricas através da MRS. Tendo como resultado informações que são úteis para a análise da contribuição individual de cada desenvolvedor do projeto de software. Lima (2014) ressalta que essas métricas não devem ser usadas de forma isolada, tendo que utilizar de outras informações além das obtidas pela MRS.

O estudo realizado por Carvalho et al. (2014) investigou a unificação de metodologias ágeis em uma ferramenta online para melhoria do desempenho e gerenciamento de projetos de softwares. Como resultado, o trabalho comprovou a efetividade da sua ferramenta no auxílio da construção do software por meio de um estudo avaliativo.

Além das ferramentas citadas anteriormente, os repositórios de software online – Github, Bitbucket ou Gitlab são alguns dos principais repositórios de software no mercado – fornecem um painel *dashboard* de gerenciamento de software, sendo possível gerenciar as etapas de construção, desenvolvimento e implantação. Essas ferramentas oferecem suporte a integração com aplicações de CI, como por exemplo o Circle CI, Travis CI ou Jenkins.

Ao analisar às aplicações mencionadas previamente, percebe-se que o grande diferencial da pesquisa deste trabalho está em criar uma ferramenta que unifique a gestão, comunicação com o cliente e análise individual da atuação do colaborador no projeto. Adicionalmente, todo esse processo acontece levando em consideração projetos do GitHub, onde métricas de software são extraídas e apresentadas de forma automatizada por meio de uma aplicação web.

2.5 Considerações do Capítulo

Este capítulo apresentou os principais conceitos e termos necessários para o entendimento deste trabalho monográfico. Primeiramente, foi descrito o conceito de Gerenciamento de Projetos de Software, onde apresentou-se o que é um projeto de software e como este pode ser gerenciado (Sessão 2.1). Em seguida, evidenciou-se o conceito de Processo de Software, bem como, definiu-se o que são metodologias ágeis, apresentando alguns dos métodos ágeis mais utilizados no mercado (Sessão 2.2). Depois disso, definiu-se o que é Mineração de Repositório de Software, apresentando algumas métricas que podem ser utilizadas para verificar a contribuição individual de desenvolvedores de software (Sessão 2.3). Por fim, na Sessão 2.4 foi comentado os trabalhos relacionados onde é evidenciado a importância do uso de ferramentas para gestão de projetos de software.

3

Metodologia

A concepção deste trabalho se deu a partir de uma proposta para estudar as dificuldades encontradas no gerenciamento de equipes de software. Este trabalho pode ser dividida em tês partes distintas. Na parte I, realizou-se uma pesquisa exploratória com o propósito de identificar quais às dificuldades que estão sendo enfrentadas pelos gestores de projetos de software. Na parte II, foi feita uma pesquisa bibliográfica de autores da área de engenharia de software e gestão de projetos, além da busca por trabalhos atuais que tem como foco a gestão e monitoramento de projeto de software. Por fim, na parte III, deu-se início ao desenvolvimento da ferramenta.

Na parte I deste trabalho, foram identificadas dificuldades em monitorar o desempenho da equipe, gerenciamento eficaz das tarefas em um determinado projeto de software e a falta de comunicação constante com o cliente sobre o andamento do projeto. Neste contexto, realizou-se uma análise de viabilidade para a construção de uma ferramenta que atendesse as necessidades criadas pelos problemas mencionados anteriormente.

Posteriormente, na parte II, realizou-se de uma pesquisa bibliográfica, por meio da leitura de artigos e livros, a fim de embasar teoricamente a construção da ferramenta. A partir desta pesquisa, percebeu-se que a má gestão de projetos de software é o principal fator para o seu fracasso (SOMMERVILLE, 2003; SOTILLE, 2014).

Depois disso, foi feita uma busca sobre formas de mensurar a construção de um software e qual a contribuição dos envolvidos. Os trabalhos de Lima (2014), Meirelles (2013) abordam o uso de técnicas de mineração de repositório de software para extração de métricas para o monitoramento de projetos de software.

Portando, Na parte III, foi decido a criação de uma ferramenta que auxiliasse o gestor na gerência de projetos de software e inspeção de desempenho dos seus desenvolvedores, assim como estabelecer uma comunicação profícua com os seus clientes.

Em seguida, foi feita a modelagem da ferramenta proposta (tecnologias utilizadas, levantamento de requisitos e elaboração de diagramas). Foi determinado o uso de tecnologias que garantissem a alta escalabilidade e performance da ferramenta,

tendo como principais escolhas o Docker, Python e JavaScript.

A princípio, decidiu-se que deveria existir uma integração com alguma plataforma online de repositório de software, neste caso foi escolhido o Github, levando em consideração a sua popularidade no mercado. Após isso, foi realizado o estudo da API do Github com o propósito de entender o seu funcionamento e quais informações agregariam maior valor ao gerenciamento de um projeto.

Para melhorar a interação entre o membros do projeto foi decidido que a ferramenta teria *dashboard* sobre o progresso dos projetos monitorados. Também foi decidido que a métrica *churn*, obitida pela soma da quantidade de linhas de código adicionadas e removidas, seria inicialmente a métrica utilizada para análise da contribuição individual de cada desenvolvedor do projeto.

Por fim, a ferramenta deveria oferecer uma abstração do quadro de Kanban para garantir uma forma prática de acompanhamento das tarefas (BENZECRY, 2017).

4

Modelagem da Ferramenta Proposta

Este capítulo retrata uma visão geral sobre a ferramenta proposta por este trabalho, bem como, apresenta uma série de diagramas (UML e Arquitetural) e tecnologias utilizadas no desenvolvimento da ferramenta.

4.1 Visão Geral da SPM

A ferramenta proposta tem como finalidade promover auxílio ao processo de gerenciamento de projetos de software, auxiliar na análise do desempenho de cada colaborador e melhorar a comunicação com o cliente. Esta ferramenta pode ser vista sob duas perspectivas: (i) gerente de projeto ou colaborador e (ii) cliente.

Sob a ótica do gerente de projeto ou colaborador, a ferramenta oferece uma visão voltada ao desenvolvimento de software, em que é possível enxergar repositórios no Github e dados importantes sobre estes, a exemplo do número de *commits* no projeto ou por contribuidor. Também é possível gerar relatórios de progresso e compartilhar informações do projeto com o cliente ao enviar para ele um código de convite que vai permitir o seu acesso ao painel de tarefas.

No contexto do cliente é possível realizar o acompanhamento de vários projetos, seu progresso e como as tarefas estão sendo distribuídas e realizadas. Vale lembrar que o cliente é um membro do projeto, podendo ter tarefas atribuídas a ele.

Ambos os contextos mencionados tem suporte a notificações, ou seja, para cada alteração no projeto todos os envolvidos serão notificados pelo Telegram ou e-mail.

4.2 Tecnologias Utilizadas

Nesta sessão será discutida quais tecnologias foram utilizadas na construção da aplicação proposta neste trabalho.

4.2.1 Google Cloud Platform

O Google Cloud Platform é uma plataforma ofertada pela Google, com um conjunto de soluções e produtos de *Cloud Computing*, podendo se adaptar à necessidade de qualquer empresa, transmitindo segurança, praticidade e tranquilidade, seja para empresas que estão no início da jornada ou rumando para a transformação digital (GOOGLE, 2019).

Um dos produtos disponibilizados pela plataforma supracitada é o Google Compute Engine. Este recurso é capaz de criar e executar máquinas virtuais (VMs) com recursos de escalonamento, desempenho e valor com os quais é possível inicializar facilmente grandes *clusters* de computação na infraestrutura do Google.

O Google Compute Engine fornece instâncias de máquinas virtuais altamente personalizáveis. Uma máquina virtual (VM, na sigla em inglês) hospedada na infraestrutura do Google. Nas instâncias do Compute Engine, é possível implantar contêineres, que são iniciados automaticamente nas instâncias que executam a imagem pública do SO otimizado para contêineres.

Neste trabalho foi utilizado uma instância virtual hospedada no *data center* da Gooogle na região de São Paulo/SP com as configurações:

- Sistema operacional Ubuntu 14.04.6 LTS;
- Processador Intel *single core* com arquitetura de 64 bits;
- CPU com 2200 MHz;
- Memória RAM de 3.75 GB;
- Disco virtual de armazenamento de 10GB;

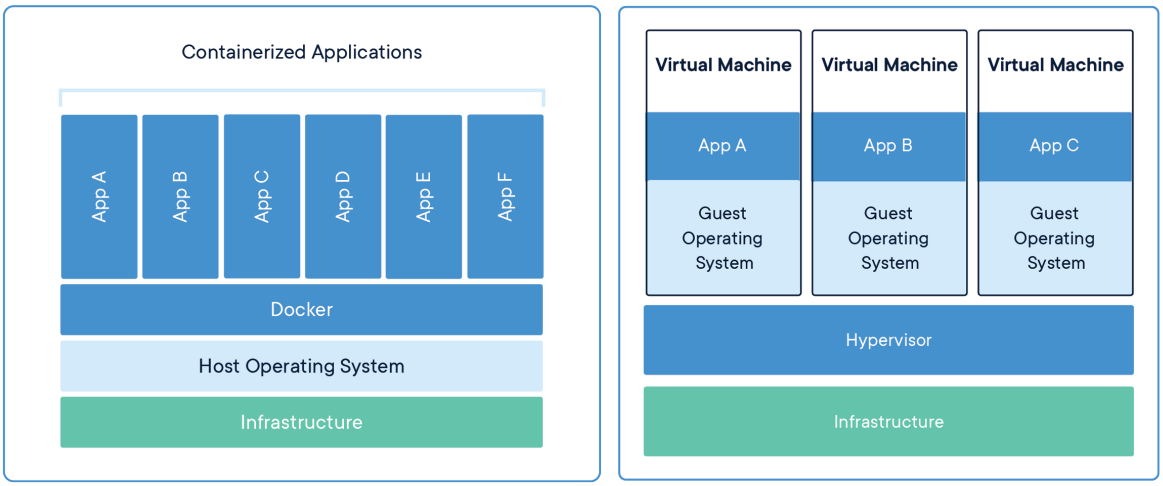
4.2.2 Contêineres Linux

Um contêiner é uma unidade padrão de software que empacota o código e todas as suas dependências para que o aplicativo seja executado de maneira rápida e confiável de um ambiente de computação para outro. Em outras palavras, os contêineres isolam o software de seu ambiente e garantem que ele funcione de maneira uniforme, apesar das diferenças, por exemplo, entre desenvolvimento e preparação (DOCKER, 2019).

Uma imagem de contêiner do Docker é um pacote de software leve, autônomo e executável que inclui tudo o que é necessário para executar um aplicativo: código, tempo de execução, ferramentas do sistema, bibliotecas do sistema e configurações

(DOCKER, 2019).

Figura 3 – Comparando *Containers* e Máquinas Virtuais.



(a) *Containers* ***(b)*** Máquinas virtuais

Fonte: (DOCKER, 2019).

A diferença entre um contêiner e uma máquina virtual pode ser notada na camada que realiza a comunicação com a infraestrutura da máquina hospedeira (Figura 3). Um *container* compartilha o *kernel* do sistema operacional hospedeiro, já a máquina virtual cria uma nova camada de comunicação chamada *Hipervisor* para fazer a tradução do sistema "convidado" para infraestrutura do "hospedeiro". Apesar de ambos serem métodos de virtualização, eles atuam de maneiras distintas, sendo que o *docker* tem maior vantagem sobre a virtualização porque dispensa a utilização de um novo sistema operacional.

Com o propósito de orquestrar a criação e o gerenciamento de um conjunto de *containers* a partir do uso de um simples arquivo de configuração, foi criado o Docker Compose. O Compose é uma ferramenta utilizada para definir e executar aplicativos Docker com vários contêineres. Com o auxílio desta ferramenta, todos os serviços do seu aplicativo são configurados facilmente com um arquivo no formato YAML. Então, com um único comando, você cria e inicia todos os serviços da sua configuração (DOCKER, 2019). O Compose funciona em todos os ambientes: produção, teste, desenvolvimento, bem como em fluxos de trabalho de *continuous integration*(CI)[1] (DOCKER, 2019).

[1] Prática de software para mescar trabalhos de vários desenvolvedores em um único repositório compartilhado.

4.2.3 Versionamento de código

Um sistema de controle de versão (VCS – Version Control System) é um sistema que registra as mudanças feitas em um arquivo ou um conjunto de arquivos ao longo do tempo, de forma que você possa recuperar versões específicas. (CHACON, 2014).

O Git é um sistema de controle de versão distribuído gratuitamente e de código aberto. Este sistema foi projetado para lidar, de forma eficiente, com todo tipo de projeto, que vão desde projetos pequenos a muito grandes. O Git é fácil de aprender e possui recursos como ramificação local barata, áreas de preparação convenientes e vários fluxos de trabalho.

O Github é uma plataforma que hospeda projetos de código fonte com controle de versão na nuvem. Ou seja, enquanto o Git é uma ferramenta de versionamento de código, o Github é um repositório de código online criado para promover a interação entre os desenvolvedores de diferentes lugares do mundo.

4.2.4 Ferramentas

Além das tecnologias já comentadas na Subseção 4.2.1, Subseção 4.2.2 e Subseção 4.2.3, foram utilizadas ferramentas para armazenamento de dados, *web framework*, cache, bibliotecas para abstração da API do Github, bibliotecas visuais para criar interfaces elegantes, servidor HTTP e *Bot* para notificações.

Para armazenamento de dados foi utilizado o PostgreSQL como banco de dados objeto relacional. Sua escolha foi motivada pela sua rapidez, ser um software de código aberto, fácil implantação com a arquitetura de *containers*, atendendo assim as necessidades do projeto.

Para a construção da camada lógica foi utilizado o Web *framework* Django. Suas principais características está no desenvolvimento rápido, modular e limpo. Outro fator importante para seu uso é ele ser uma ferramenta de código aberto.

Para facilitar a comunicação da aplicação desenvolvida em Django com a API do Github foi utilizado uma biblioteca de código aberto chamada PyGithub. Ele provê uma interface em Python para acesso aos *endpoints* da API.

Um detalhe importante a ser mencionado é que a API do Github limita o número de requisições por hora, ou seja, caso esse limite seja ultrapassado o acesso aos seus dados será interrompido até que este limite seja renovado. Para evitar que isso ocorra, foi utilizada uma ferramenta de cache chamada Memcached, ela permite que o número de requisições sejam reduzidas, uma vez que armazena dados já solicitados pelo usuário na memória da máquina.

Na construção das aplicações Web foi utilizado a biblioteca de código aberto ReactJS. Ela agiliza o desenvolvimento ao fornecer um conjunto de ferramentas que facilitam a manipulação de páginas HTML. Para auxiliar o ReactJS foi utilizado o AntDesign como biblioteca de UI (User Interface), ela contém componentes prontos que podem ser utilizados de maneira rápida e simples.

para notificar o usuário de maneira eficiente, foi criado um Bot no Telegram Bot API. Essa plataforma provê um conjunto de ferramentas para interagir com o usuário através de mensagens e comandos simples.

4.3 Requisitos

O levantamento de requisitos é uma atividade que precede a construção e existe durante toda a vida útil do software. A seguir, será mostrado os requisitos funcionais (tabela 1) e não-funcionais (tabela 2) utilizados na elaboração desta ferramenta.

Tabela 1 – Requisitos funcionais.

	Descrição	**Entradas**	**Saídas**
RF0001	login com aplicações externas.	dados de login no provedor de autenticação.	novo usuário cadastrado no sistema.
RF0002	pesquisar projetos no Github.	nome do projeto.	lista de repositórios que correspondem ao termo buscado.
RF0003	listar projetos do usuário autenticado autenticado que estão no Github.		repositórios do usuário.
RF0004	gerar projeto com base no repositório do Github.		novo projeto gerenciado no sistema.
RF0005	enviar notificações por email.		email de notificação de nova tarefa criada.
RF0006	enviar notificações por Telegram.		nova mensagem gerada a partir da alteração de uma tarefa no projeto.
RF0007	gerar relatório de progresso.	tipo de relatório e projeto.	documento em PDF.
RF0008	convidar o cliente para projeto.	envio do código de convite para o cliente.	projeto compartilhado com o cliente.
RF0009	cadastrar tarefa.	dados da tarefa.	tarefas cadastrada no projeto.
RF00010	editar tarefa.	dados da tarefa.	tarefa com os dados atualizados.
RF00011	cadastrar nota em uma tarefa.	dados da nota.	nota cadastrada para uma tarefa.
RF00012	listar *commits* de um repositório.		informações sobre os *commits*.
RF00013	visualizar o *churn* de um repositório		gráfico de *churn*.
RF00014	visualizar o *churn* por contribuidor do repositório.		gráfico de churn individual.
RF00015	monitorar projetos abertos.	projeto que será monitorado	projeto favoritado.

Fonte: O Autor.

A tabela 1 exibe os principais requisitos funcionais utilizados na concepção inicial da ferramenta.

Tabela 2 – Requisitos não-funcionais.

	Descrição
NF0001	utilizar bibliotecas de interface modernas e reativas para uma melhor experiência de navegação para o usuário.
NF0002	construir a plataforma utilizando tecnologia de virtualização altamente escalável.
NF0003	hospedar o software na nuvem.
NF0004	realizar cache de dados para diminuir a demanda de acesso a API's externas.
NF0005	armazenar os dados em um banco de dados open-source para evitar custos e despesas.

Fonte: O Autor.

A tabela 2 lista os requisitos não-funcionais que foram levantados no início da elaboração do software.

4.4 Diagramas

Um diagrama é uma representação visual cujo propósito consiste em transmitir um conceito ou ideia de forma simples. A seguir será exposto os diagramas UML e o diagrama arquitetural do projeto.

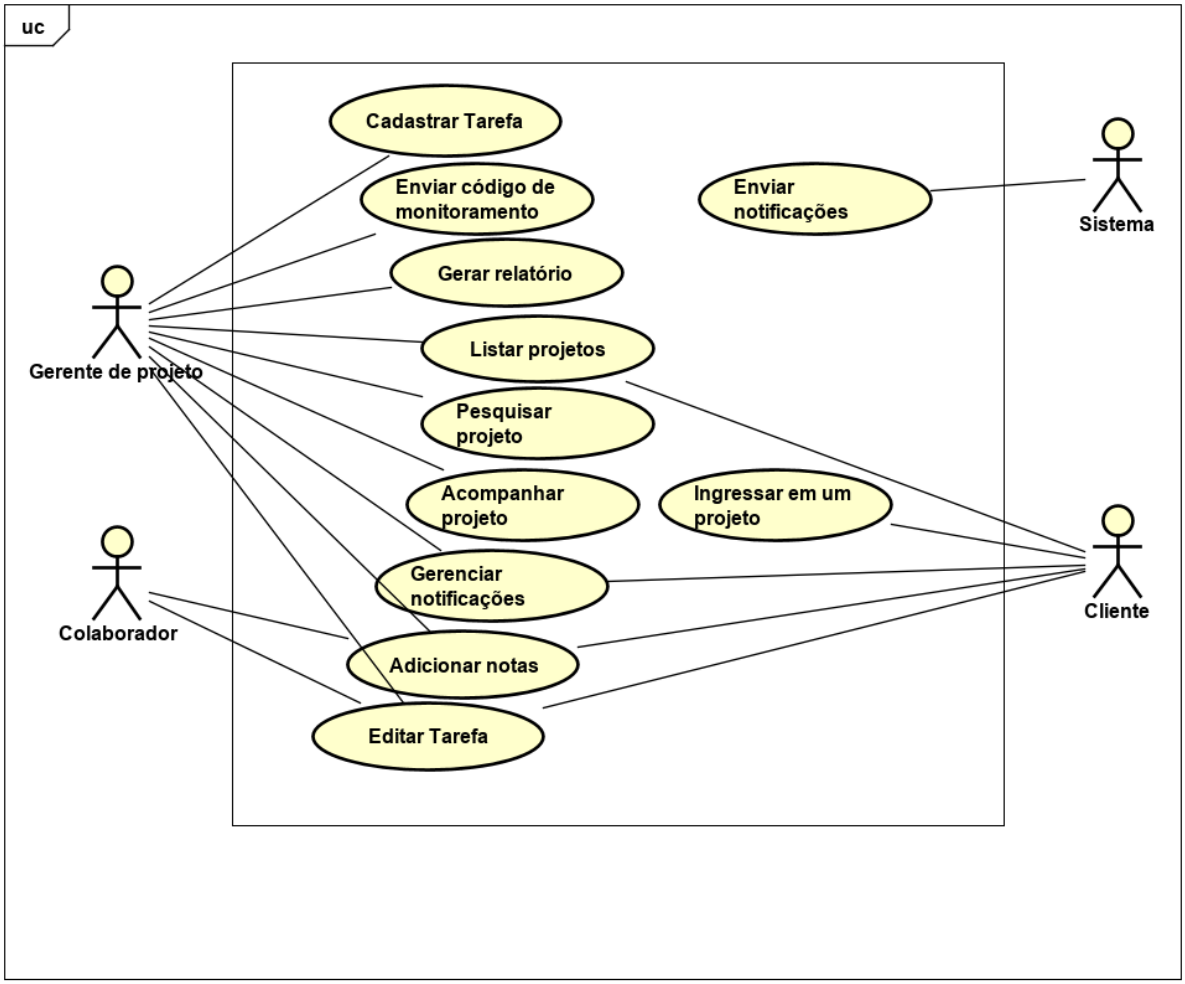
4.4.1 Diagramas UML

O diagrama UML é uma representação gráfica padrão para elaboração da estrutura de projetos de software. Para este trabalho, produziu-se os diagramas de casos de uso (Seção 4.4.1.1), atividades (Seção 4.4.1.2), e de entidade relacionamento (Seção 4.4.2). Estes diagramas são essenciais para entender os requisitos do sistema.

4.4.1.1 Diagrama de casos de uso

O diagrama de casos de uso é utilizado para descrever as principais funcionalidades do sistema. A partir deste diagrama é possível compreender o papel de cada ator do sistema. A Figura 4 mostra o diagrama de casos de uso da ferramenta SPM.

Figura 4 – Diagrama de casos de uso.



Fonte: O Autor.

No diagrama da Figura 4 são propostos 10 (dez) casos de uso que representam as principais funcionalidades do sistema. Algumas dessas funcionalidades são compartilhadas entre os atores gerente de projeto, cliente e colaborador.

O gerente de projetos é o único que pode cadastrar novas tarefas no projeto. Ele também pode enviar o código de convite para o cliente, para que este passe a ter acesso ao andamento da execução das tarefas propostas para um determinado projeto de software. Adicionalmente, o gerente de projetos é capaz de gerar relatórios de produtividade individual dos colaboradores (desenvolvedores) de um projeto de software, além de poder acompanhar projetos públicos no Github.

O gerente de projeto, colaborador e cliente podem adicionar notas em qualquer tarefa dos projetos em que eles fazem parte. Também é permitido editar qualquer tarefa que lhes for atribuída.

O gerente de projeto e cliente podem listar os projetos em que eles fazem parte e gerenciar as notificações que desejam receber.

O cliente pode ingressar em um projeto através do código de convite enviado pelo gerente de projeto.

O sistema enviará notificações para os usuários sempre que uma tarefa for cadastrada ou atualizada.

4.4.1.2 Diagrama de Atividades

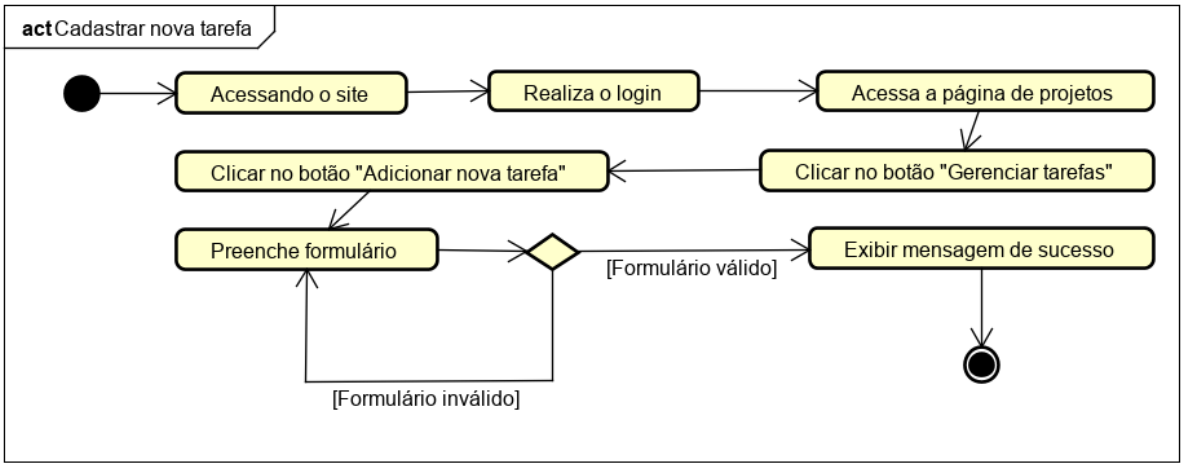
Um diagrama de atividade documenta o fluxo de atividades/ações em um único processo, exibindo a sequência de passos que podem ou não ocorrer naquele processo.

As Figuras 5, 6, 7 e 8 representam algumas das principais funcionalidades do sistema, que são:

- Adicionar nova tarefa;
- Enviar código de convite para o cliente;
- Adicionar um novo projeto;
- Habilitar notificações pelo Bot [2];

Os diagramas serão apresentados na mesma ordem que foram citados anteriormente.

Figura 5 – Adicionar nova tarefa.



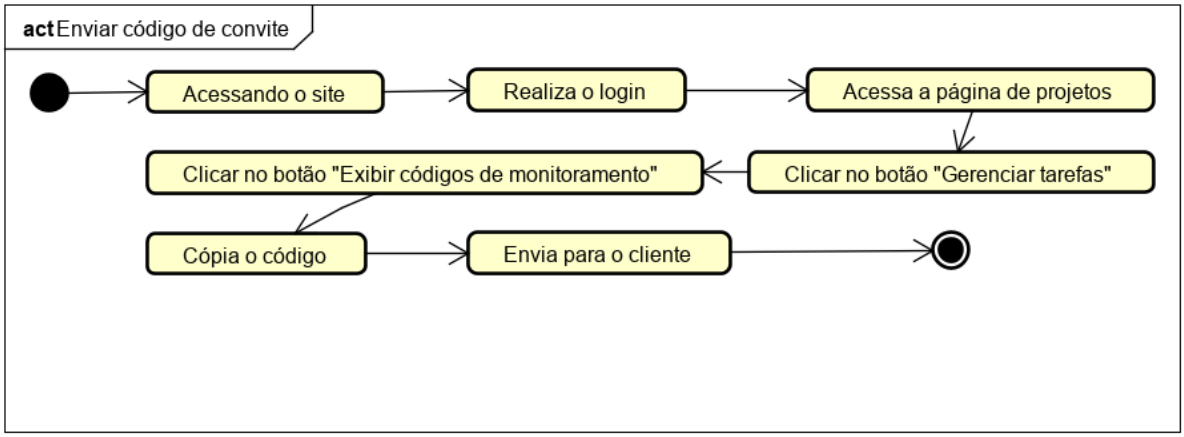
Fonte: O Autor.

O diagrama da Figura 5 mostra o fluxo de atividades que o gerente de projeto terá que fazer para adicionar uma nova tarefa. Para isso, ele deve acessar o site, realizar

[2] software capaz de simular ações humanas repetidas vezes obedecendo padrão, semelhante a um robô.

o login, ir até a página de projetos, clicar no botão "Gerenciar tarefas"do projeto desejado, clicar no botão "Adicionar nova tarefa", preencher o formulário e salvar, depois disso uma mensagem de sucesso será exibida e a tarefa estará salva.

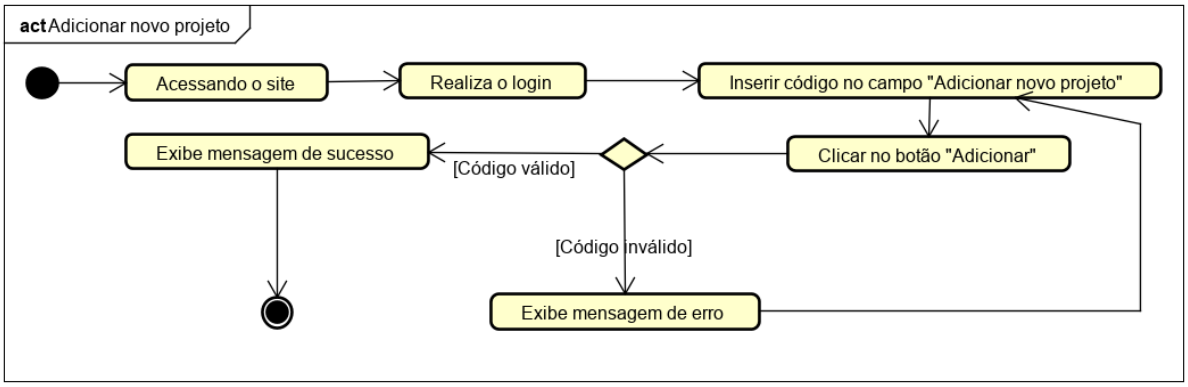
Figura 6 – Enviar código de convite.



Fonte: O Autor.

O diagrama da Figura 6 mostra o fluxo de atividades que o gerente de projeto terá que fazer para adicionar enviar o código de convite para o seu cliente. Para isso, ele deve acessar o site, realizar o login, ir até a página de projetos, clicar no botão "Gerenciar tarefas"do projeto desejado, clicar no botão "Exibir códigos de monitoramento", copiar o código de monitoramento e enviar para cliente.

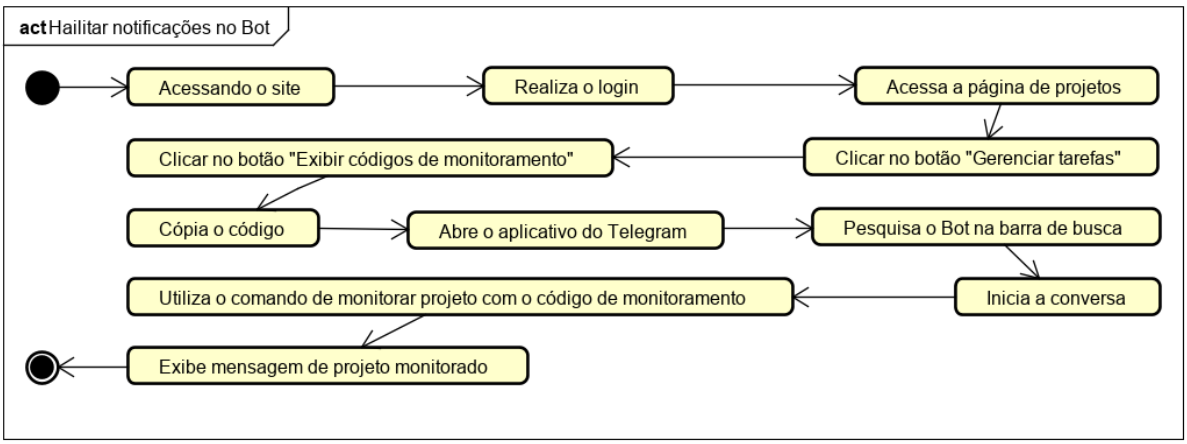
Figura 7 – Adicionar novo projeto.



Fonte: O Autor.

O diagrama da Figura 7 mostra o fluxo de atividades que o cliente terá que fazer para monitorar um novo projeto. Para isso, ele deve acessar o site, realizar o login, inserir o código de monitoramento no campo "Adicionar novo projeto"e clicar no botão "Adicionar", depois disso o sistema irá fazer uma consulta na base de dados, caso o código seja válido uma mensagem de sucesso será exibida ao cliente e o projeto adicionado estará sendo monitorado.

Figura 8 – Habilitar notificações.



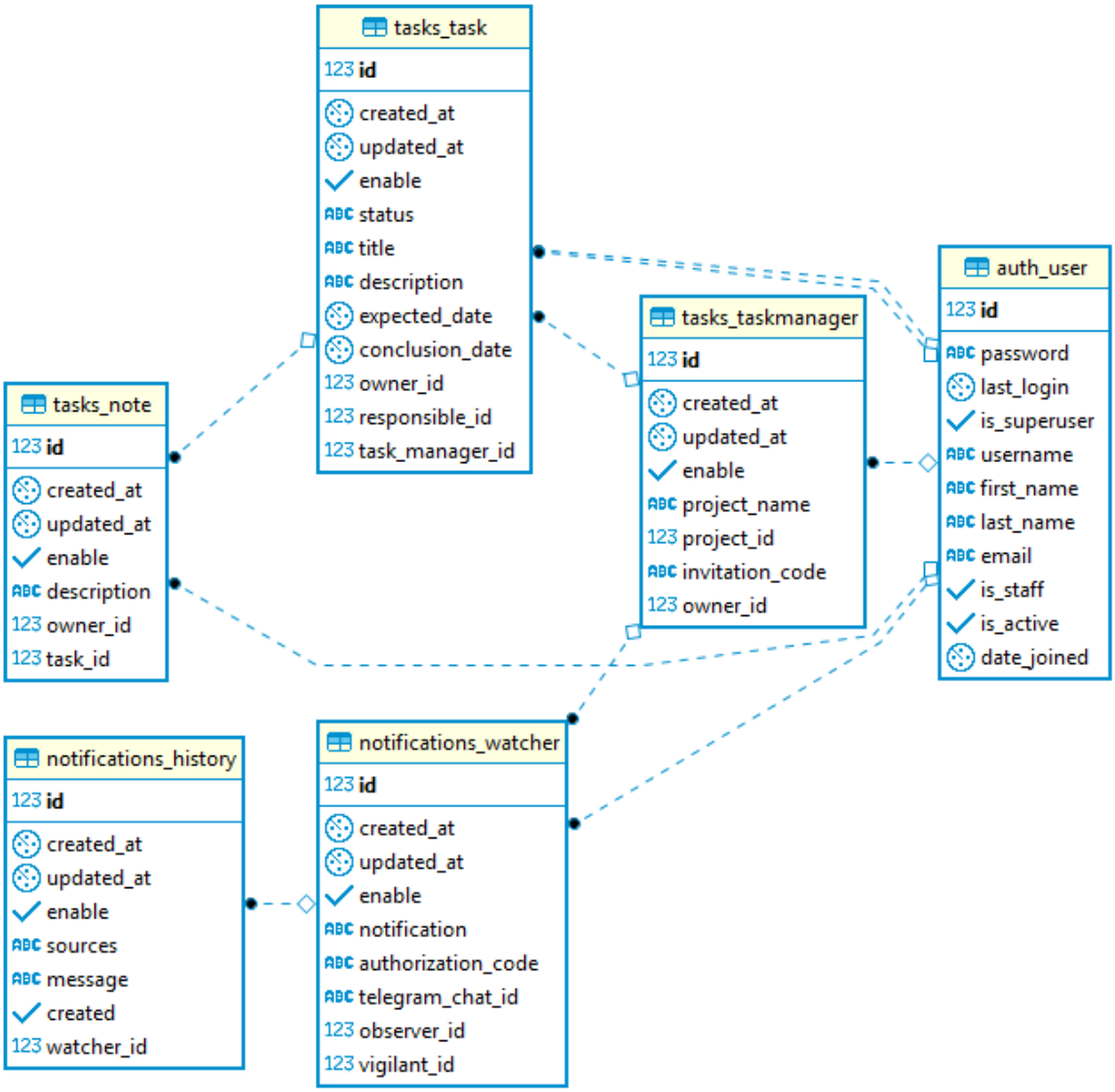
Fonte: O Autor.

O diagrama da Figura 8 mostra o fluxo de atividades que o cliente, gerente de projeto ou colaborador terá que fazer para habilitar as notificações pelo Bot. Para isso, ele deve acessar o site, realizar o login, acessar a página de projetos, clicar no botão "Gerenciar tarefas", clicar no botão "Exibir códigos de monitoramento"e copir o código de monitoramento do Bot. Depois de fazer as ações anteriores, o usuário deve abrir o aplicativo do Telegram, pesquisar o bot na barra de busca da aplicação, iniciar a conversa, utilizar o código de monitoramento no comando de monitorar projeto, a partir desse ponto uma mensagem de sucesso será exibida.

4.4.2 Diagrama de entidade relacionamento

O diagrama de entidade relacionamento é utilizado para representar as relações entre as entidades que foram modeladas no sistema. A Figura 9 mostra o diagrama de entidade relacionamento utilizado para modelar a ferramenta proposta neste trabalho monográfico.

Figura 9 – Diagrama de entidade relacionamento.



Fonte: O Autor.

O diagrama da Figura 9 é composto pelas 6 (seis) principais entidades do sistema, que são: *tasks_taskmanager, tasks_task, tasks_note, notifications_history, notification_watcher* e *auth_user*. O nome da entidade é gerado com a regra *underscore_case*, sendo o prefixo do nome da aplicação e nome do modelo. As entidades exibidas representam 3 aplicações internas do sistema, *tasks, notifications* e *auth* – essas aplicações serão exploradas com mais detalhes no Capítulo 5.

A entidade *auth_user* armazena os dados de um usuário do sistema, relacionando-se de 1 para muitos com *tasks_taskmanager*, 1 para muitos com *tasks_task*, 1 para muitos com *tasks_note* e 1 para muitos com *notifications_watcher*.

A entidade *notification_watcher* representa os usuários que estão participando ou monitorando um projeto de software. Ela relaciona-se de 1 para muitos com *notifica-*

tion_history, de muitos para 1 com *auth_user*, de muitos para 1 com *tasks_taskmanager*.

A entidade *notification_history* representa o histórico de notificações de um usuário. Ela relaciona-se de muitos para 1 com *notification_watcher*.

A entidade *tasks_taskmanager* é responsável por gerenciar as tarefas de um projeto do Github. Ela relaciona-se de muitos para 1 com *auth_user*, 1 para com muitos *tasks_task* e 1 para com muitos com *notification_watcher*.

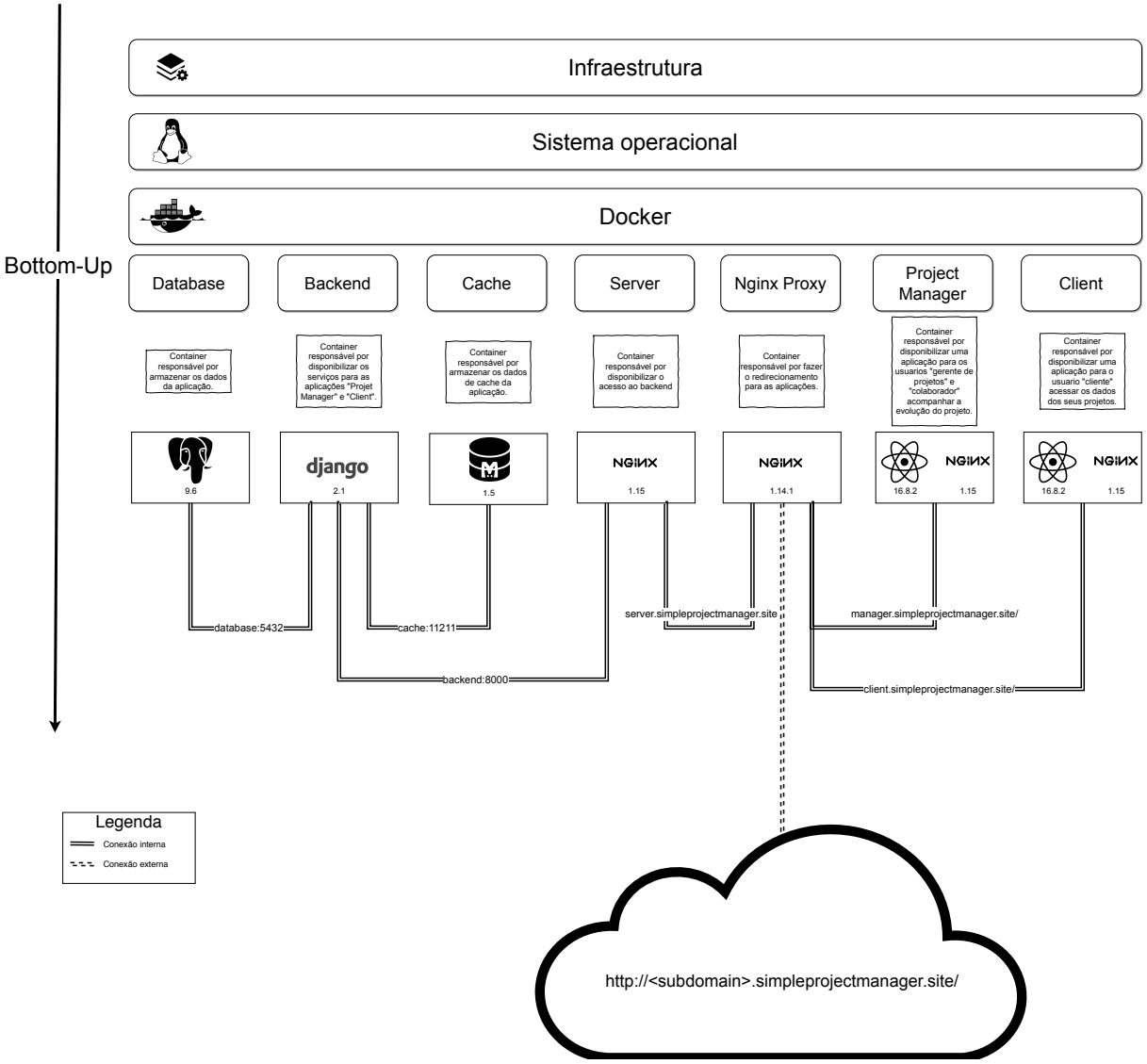
A entidade *tasks_task* é responsável por representar uma tarefa. Ela relaciona-se de muitos para 1 com *tasks_taskmanager*, 1 para muitos com *tasks_note* e de muitos para 1 com *auth_user*.

A entidade *tasks_note* representa uma nota de uma tarefa. Ela relaciona-se de muitos para 1 com *tasks_tasks* e de muitos para 1 com *auth_user*.

4.4.3 Diagrama de arquitetura

Nesta subseção será comentado sobre o diagrama de arquitetura do sistema. Ele ilustra, através de uma abordagem “Bottom-up”, a arquitetura final do sistema desenvolvido e como ocorre o seu funcionamento interno.

Figura 10 – Diagrama de arquitetura.



Fonte: O Autor.

A Figura 10 exibe as três camadas do sistema, que são:

- **Infraestrutura:** Máquina utilizada para hospedar a aplicação, será comentada na Subseção 4.2.1.
- **Sistema operacional:** Sistema operacional utilizado na máquina da aplicação, ele será comentado na Subseção 4.2.1.
- **Docker:** Tecnologia de virtualização utilizada. Ela será comentada na Subseção 4.2.2.

Na Figura 10, a camada Docker envolve 7 aplicações que foram construídas para trabalharem juntas na composição final da ferramenta. As definições de cada

aplicação e suas respectivas interações estão dispostas como segue:

- **Aplicação Database**: é um contêiner responsável por fornecer a persistência de dados para a aplicação Backend. Sua comunicação é feita através da rede local pelo endereço “database” na porta 5432.
- **A aplicação Cache**: consiste em um contêiner responsável por fornecer o cache de dados para a aplicação Backend. Sua comunicação é feita através da rede local pelo endereço “cache” na porta 11211.
- **A aplicação Backend**: é um contêiner responsável por fornecer acesso a camada lógica do sistema. Sua comunicação é feita através da rede local pelo enderece “backend” na porta 8000.
- **A aplicação Server**: é um contêiner responsável por fornecer uma interface de comunicação entre o servidor HTTP que se comunica com o mundo exterior e a aplicação “Backend”. Sua comunicação é feita através da rede local pelo endereço “server.simpleprojectmanager.site” na porta 80.
- **A aplicação Nginx Proxy**: constitui-se de um contêiner responsável por permitir acesso as aplicações internas do sistema com o mundo exterior, para isso ela realiza o *proxy* de acordo com o subdomínio solicitado pelo navegador. Sua comunicação e feita através da rede pelo endereço “simpleprojectmanager.site” na porta 80.
- **A aplicação Projet Manager**: é um contêiner responsável por fornecer acesso a aplicação **Manager** para o mundo exterior através da aplicação **Nginx Proxy**. Sua comunicação é feita através da rede local pelo endereço “manager.simpleprojectmanager.site” na porta 80.
- **A aplicação Client**: é um contêiner responsável por fornecer acesso a aplicação **Client** para o mundo exterior através da aplicação **Nginx Proxy**. Sua comunicação é feita através da rede local pelo endereço “manager.simpleprojectmanager.site” na porta 80.

A Figura 10 mostra a comunicação entre os contêineres. As linhas sem tracejado representam as ligações internas, essas não se comunicam com o mundo exterior. As linhas com tracejado representam a comunicação com o mundo exterior.

4.5 Considerações do Capítulo

Este capítulo apresentou a modelagem realizada para o desenvolvimento da ferramenta proposta por esse trabalho monográfico, dando detalhe de cada um dos artefatos que subsidiaram o planejamento do desenvolvimento do projeto. A Sessão 4.1 mostrou a visão geral da ferramenta. Na Sessão 4.2 foi abordado as tecnologias utilizadas para o desenvolvimento da ferramenta, onde foi explicado o conceito de *cloud platform*, *containers* Linux e versionameto de código, além de outras ferramentas empregadas neste software. Depois disso, a Sessão 4.3 foi abordado os principais requisitos funcionais e não-funcionais. Por fim, foram apresentados os diagramas criados (Sessão 4.4), sendo eles: UML, entidade-relacionamento, e arquitetural.

5

Apresentação da Aplicação

Antes de iniciar a apresentação da aplicação é necessário ressaltar que a ferramenta foi dividida em dois contextos para atender as necessidades do gerente de projeto/colaborador e as necessidades do cliente – umas das grandes vantagens de trabalhar com contêineres *docker* no linux é a facilidade de criar novas aplicações, de forma modulada, evitando assim a complexidade de criar uma grande e única aplicação com complexas regras de acesso. Além disso, para auxiliar os usuários no monitoramento de projetos, foi criado um Bot no Telegram para enviar notificações (caso o usuário habilite essa funcionalidade) das alterações feitas em cada projeto monitorado.

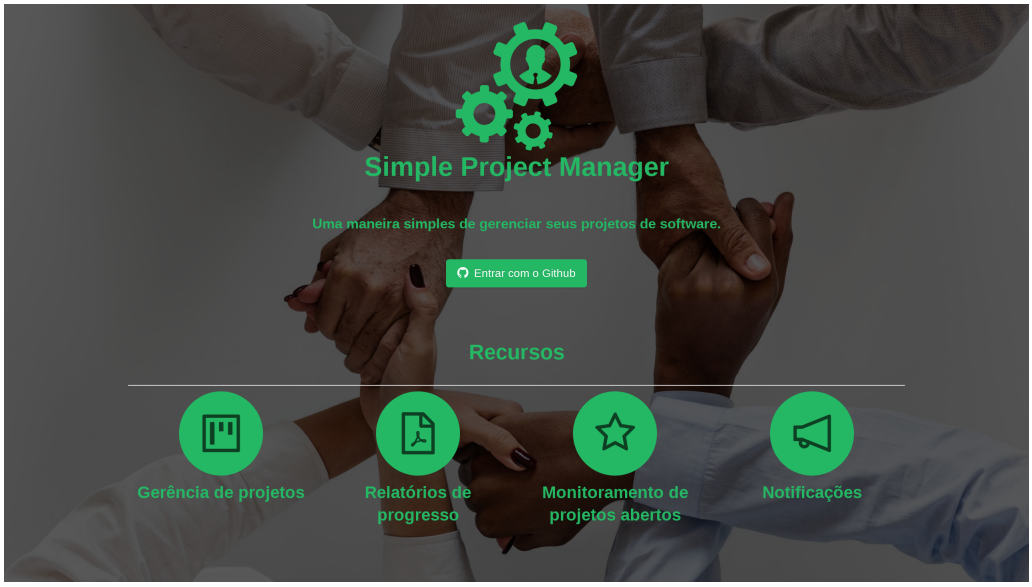
Nas sessões seguintes será detalhado as aplicações e o Bot do Telegram, a primeira será chamada de **Manager** e será apresentada na Sessão 5.1, essa aplicação tem como objetivo atender as necessidades do gerente e colaborador do projeto. A segunda aplicação será nomeada de **Client** e será apresentada na Sessão 5.2, tendo como objetivo principal oferecer a capacidade de monitoramento de projetos de software para o(s) cliente(s). O Bot será chamado de **MeninoDeRecado_Bot** e será comentado na Sessão 5.3.

A seguir será mostrado os resultados obtidos após a fase de construção do software.

5.1 Manager

Nesta sessão será comentado as telas da aplicação **Manager**, ela tem o propósito de atender as necessidades do gerente de projeto e do colaborador.

Figura 11 – Tela de Login.

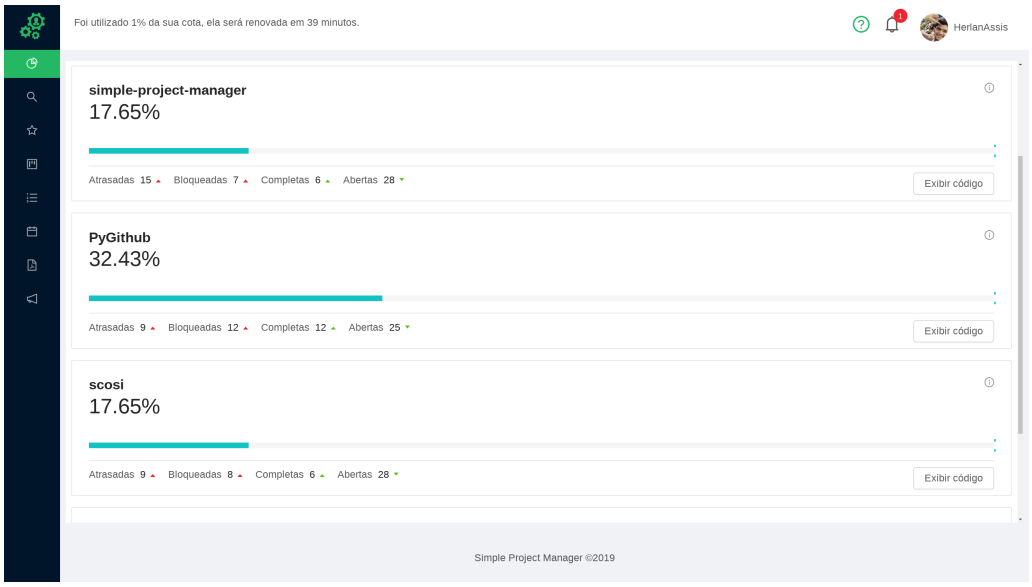


Fonte: O Autor.

A Figura 11 mostra a tela de login do aplicativo **Manager**, exibindo os recursos da aplicação e o login social pelo Github. Os principais recursos apresentados são: gerência de projetos, relatório de progresso, monitoramento de projetos abertos e notificações.

Para utilizar a aplicação é necessário possuir uma conta ativa no Github, isso é obrigatório porque todos os projetos gerenciados no sistema serão associados aos repositórios do usuário no Github – o uso de login social com o Github permite que os dados de repositórios de software existentes na plataforma sejam acessíveis pela aplicação **Manager**.

Figura 12 – Tela Principal.



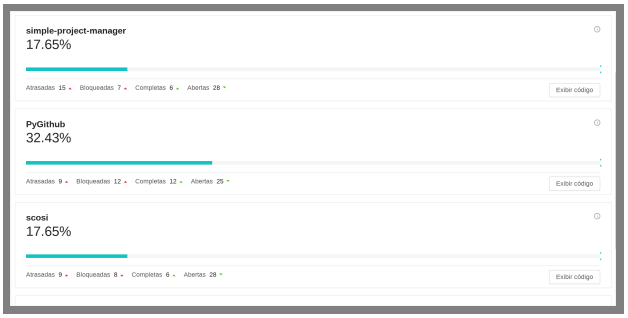
Fonte: O Autor.

A Figura 12 apresenta a tela inicial da aplicação para um usuário autenticado. Ela contém informações resumidas de todos os projetos monitorados e notificações importantes.

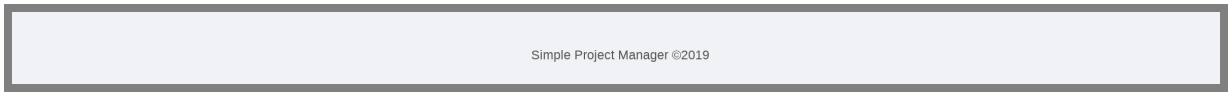
Figura 13 – Estrutura da aplicação.

(a) Cabeçalho

(b) Menu.



(c) Corpo.



(d) Rodapé.

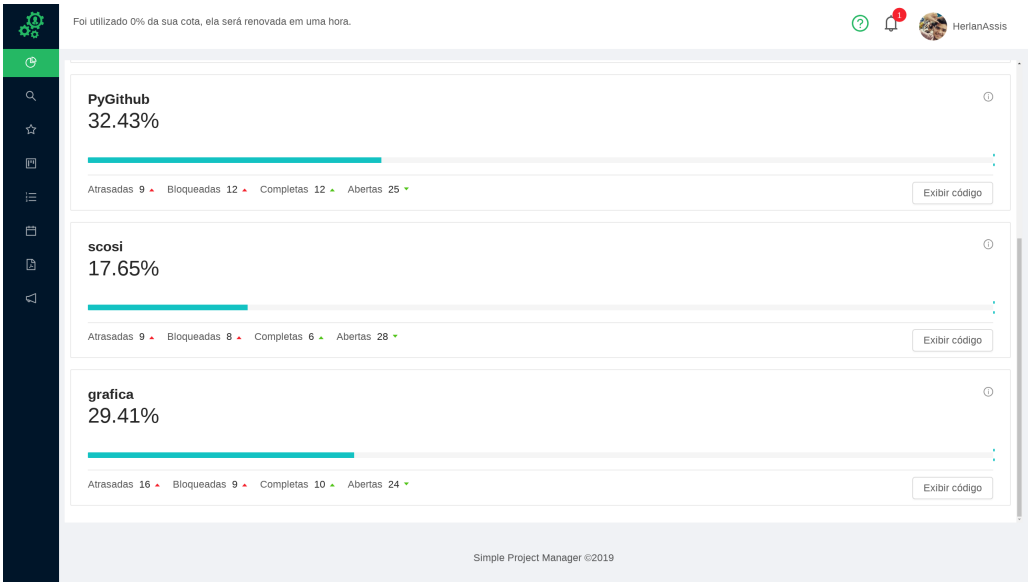
Fonte: O Autor.

Conforme mostrado na Figura 13, a aplicação **Manager** pode ser dividida em 4 partes:

- O cabeçalho (Figura 13a) informa a cota disponível para acesso a *Aplication Programming Interface* (API) do Github – o Github API para acesso as suas informações, a cota de acesso dela é renovada a cada hora –, as notificações do sistema e dados do usuário logado. Também é logout da aplicação ao clicar no botão com o seu nome;
- O menu (Figura 13b) contém as páginas: Home, Pesquisar, Projetos Assistidos, Projetos, Tarefas, Agenda, Relatórios e Notificações – nos próximos parágrafos essas páginas serão comentadas com mais detalhes;
- O corpo da página (Figura 13c) mostra informações referentes a cada página da aplicação;
- O rodapé (Figura 13d) apresenta informações de direitos autorais;

As páginas da aplicação serão apresentadas seguindo a ordem em que foram citadas anteriormente.

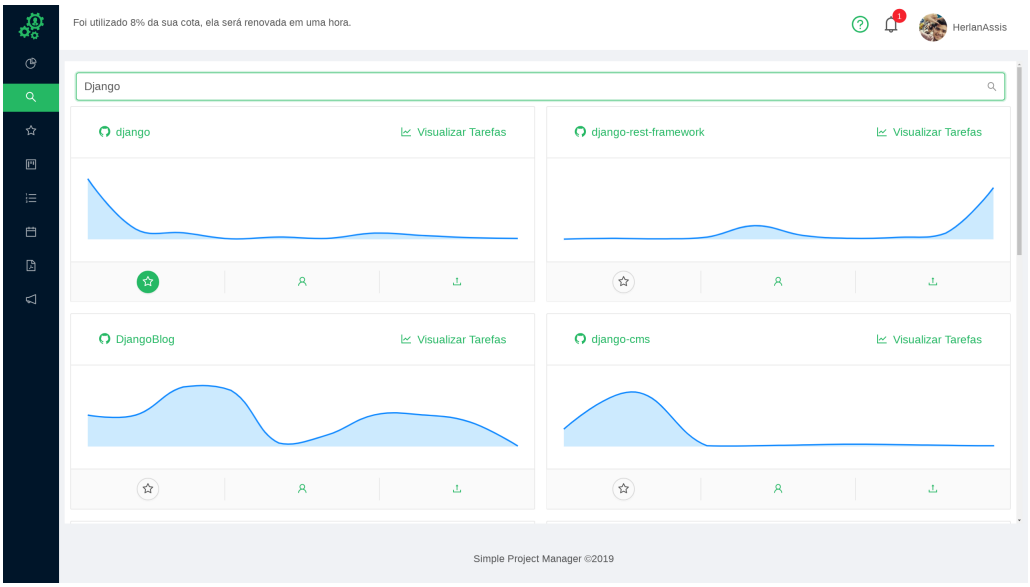
Figura 14 – Página **Home**.



Fonte: O Autor.

A página **Home** (Figura 14) lista todos os projeto gerenciados, exibindo informações sobre o seu progresso [1], quantidade de tarefas atrasadas, bloqueadas, completas e abertas. Além dessas informações, o código de convite está disponível através do botão "Exibir código".

Figura 15 – Página **Pesquisar**.



Fonte: O Autor.

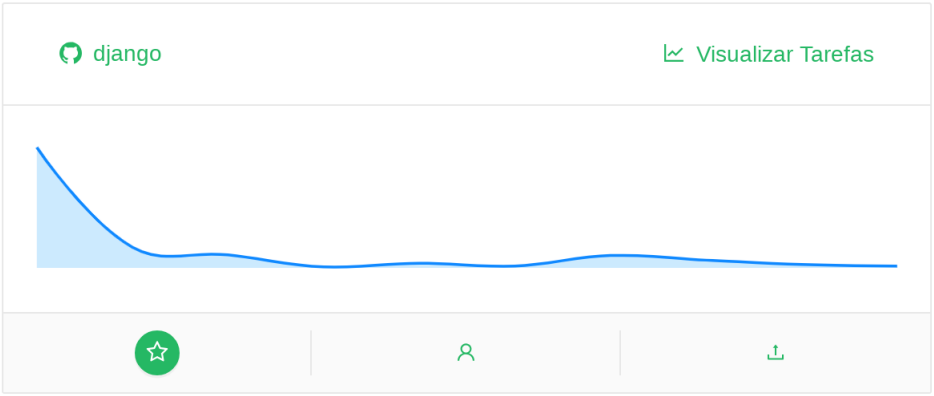
Na página **Pesquisar** (Figura 15) o gerente de projetos pode realizar uma busca na base de dados do Github, somente os repositórios abertos serão retornados como

[1] O progresso é calculado pela divisão do total de tarefas concluídas pelo total de tarefas do projeto multiplicado por 100.

resultado da pesquisa. O usuário terá acesso a uma lista de repositórios, cada um em uma "caixa"com informações sobre o número de *commits*, números de contribuidores, botão de favoritar, link do projeto no Github, gráfico com o *churn*, botão para visualizar ou gerenciar tarefas – se o usuário for o proprietário do projeto o botão a ser exibido será o "Gerenciar tarefas", caso contrário "Visualizar tarefas".

Neste ponto da apresentação, se faz necessário explicar de forma detalhada a caixa que contém as informações de um repositório.

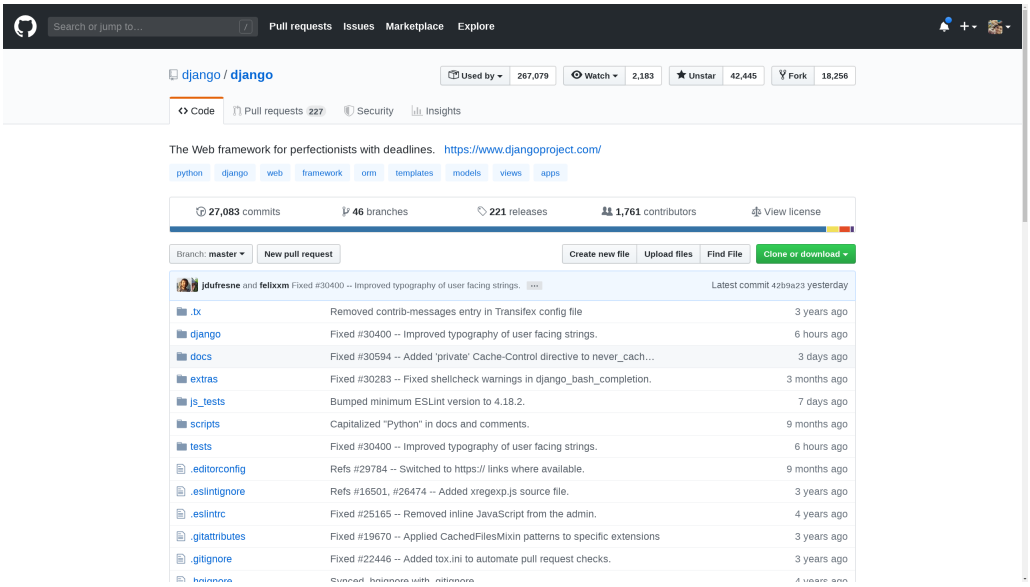
Figura 16 – "Caixa"com informações de um repositório.



Fonte: O Autor.

Na Figura 16, o *link* localizado no canto superior esquerdo abre uma nova aba no navegador com a página principal do repositório no Github. O *link* no canto superior direito será exibido como "Gerenciar tarefas"para o gerente de projetos, caso contrário irá exibir o texto "Visualizar tarefas"e ao ser clicado redireciona o usuário (gerente de projeto ou colaborador) para a página de tarefas do projeto. Já no centro da figura é mostrado o gráfico de *churn*, ele é obtido com a soma de linhas adicionadas e removidas dos últimos *commits* da última semana. No espaço inferior encontram-se os botões "Favoritar/Desfavoritar", "Contribuidores"e "*Commits*". O botão de "Contribuidores"irá levar o usuário a página de contribuidores e o botão de "*Commits*", à página de *"Commits"*.

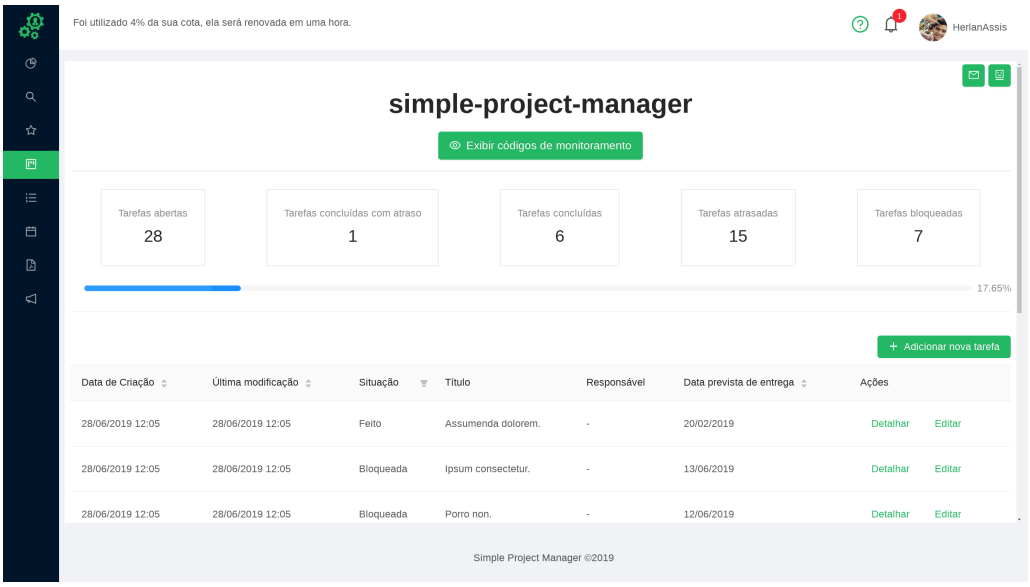
Figura 17 – Repositório do projeto no Github.



Fonte: O Autor.

Ao clicar no título do projeto, o usuário será redirecionado para a página principal do projeto no github (Figura 17).

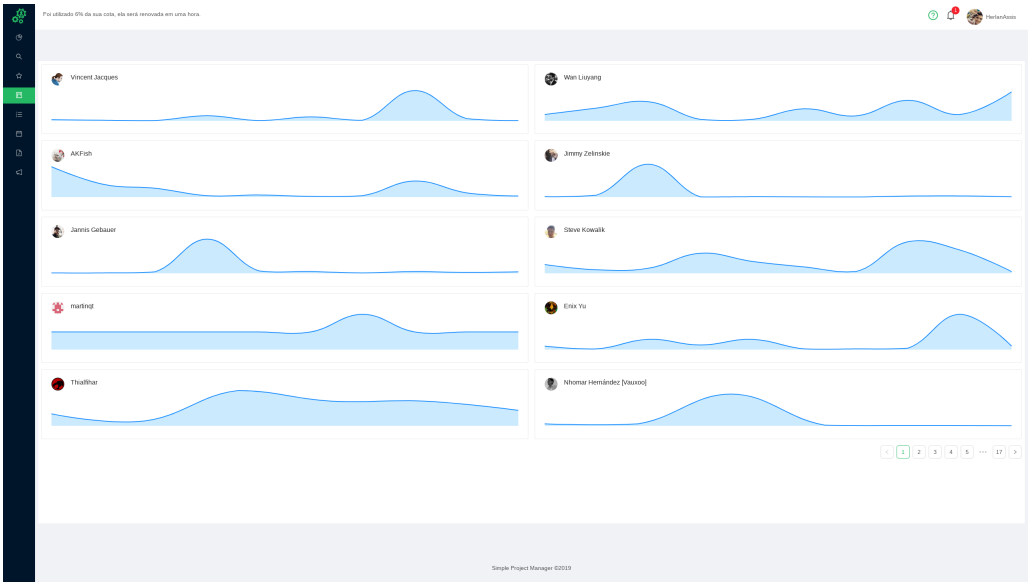
Figura 18 – Página **Tarefas do Projeto**.



Fonte: O Autor.

Na Figura 18, ao clicar no botão "Gerenciar/Visualizar Tarefas", o usuário será redirecionado para a página de **Tarefas do Projeto**. Essa página será detalhada nos próximos parágrafos.

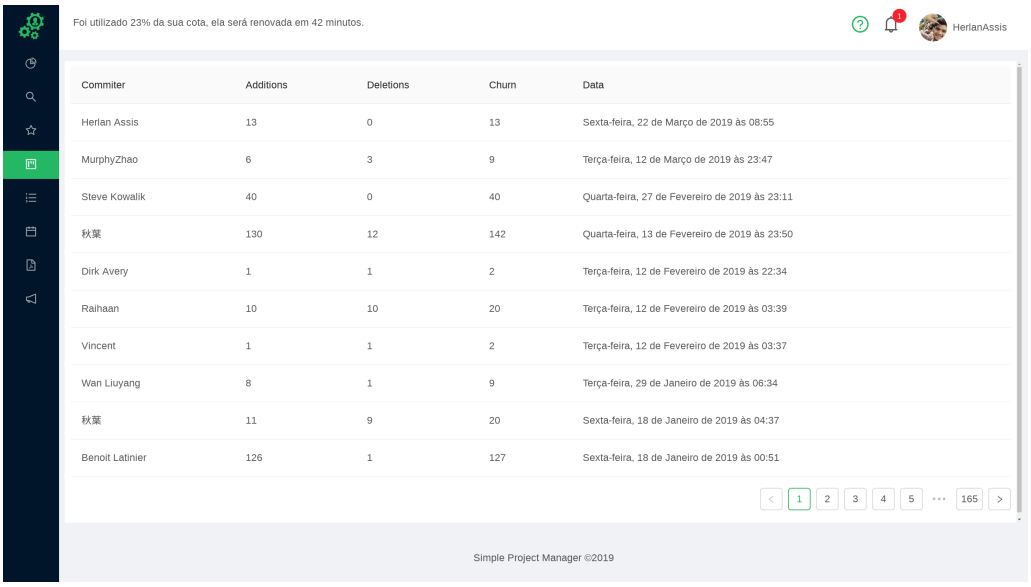
Figura 19 – Página **Contribuidores**.



Fonte: O Autor.

Conforme exibido na Figura 19, a página **Contribuidores** exibe uma lista paginada com gráficos de *churn* de todos os contribuidores do projeto.

Figura 20 – Página **Commits**.



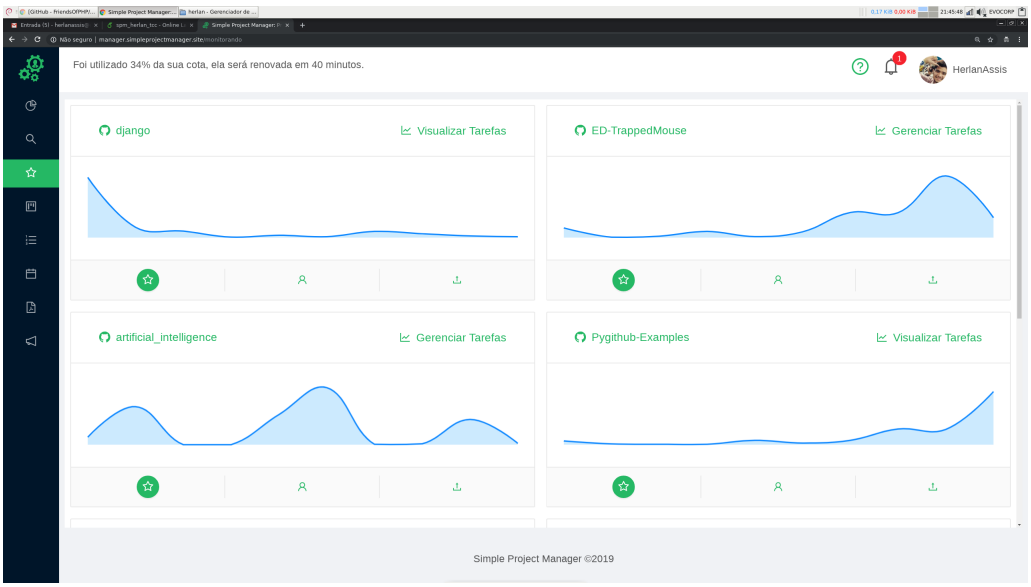
Fonte: O Autor.

Já na Figura 20, a página ***Commits*** exibe uma tabela com 5 colunas, que são:

- Commiter: responsável pelo *commit*;
- Additions: número de linhas adicionadas;
- Deletions: número de linhas removidas;

- *Churn*: soma das linhas adicionadas e removidas;
- Data: data em que o *commit* foi feito;

Figura 21 – Página **Projetos Assistidos**.

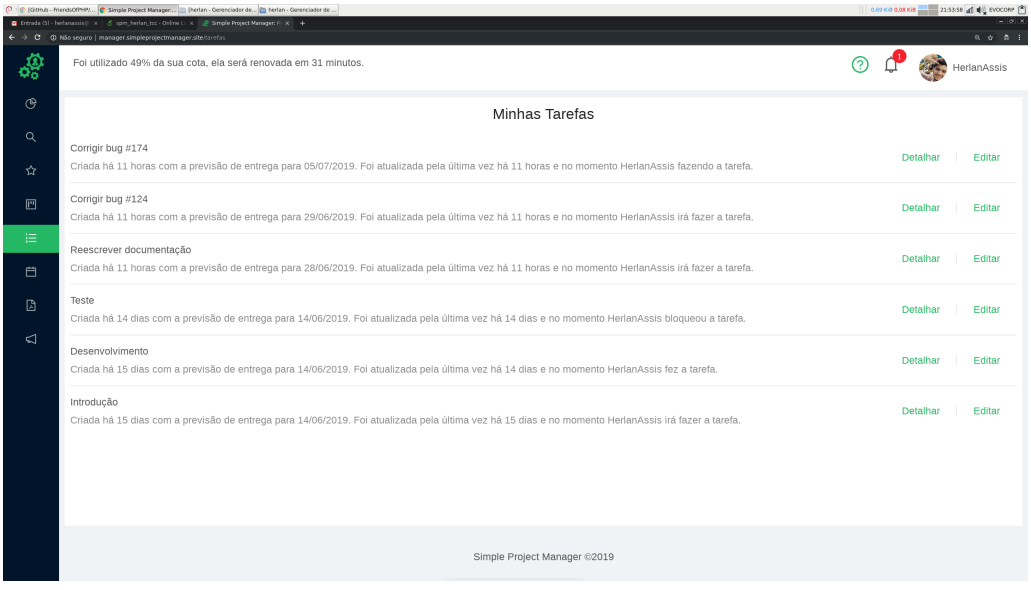


Fonte: O Autor.

Na página **Projetos Assistidos** (Figura 21), o gerente pode acessar a lista de projetos marcados como "favoritos", tendo acesso às informações já citadas no parágrafo anterior sobre a listagem de projetos.

A página **Projetos**, Figura 18, exibe uma lista com os repositórios hospedados no Github que pertencem ao gerente de projeto.

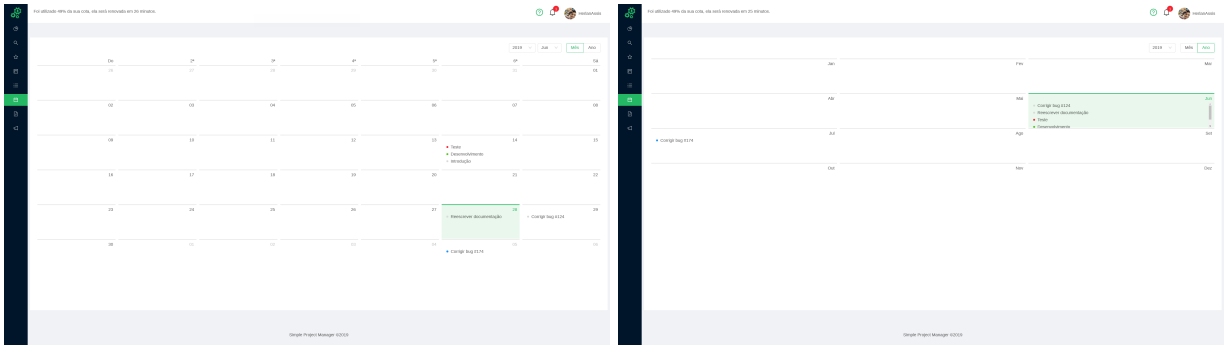
Figura 22 – Página **Tarefas**.



Fonte: O Autor.

A página **Tarefas** (Figura 22) contém uma lista de todas as tarefas pelas quais o usuário é responsável. Nesta página são exibidas informações sobre a data de criação, última atualização, previsão de entrega e status da tarefa. Para cada tarefa é permitido ver os detalhes e edita-la.

Figura 23 – Página **Agenda**.

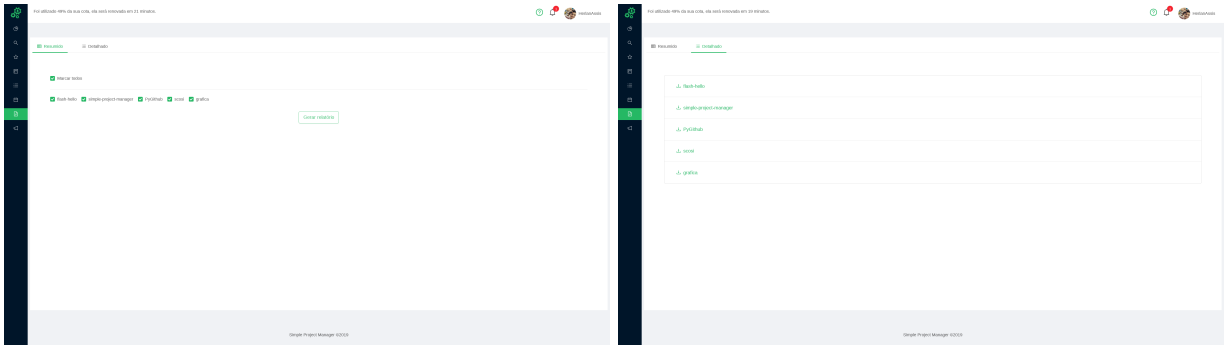


(a) Visualização por mês. ***(b)*** Visualização por ano.

Fonte: O Autor.

A página **Agenda** (Figura 23) exibe uma agenda que pode ser visualizada por mês (Figura 23a) ou ano (Figura 23b). Ela contém a lista de tarefas que foi atribuída ao usuário, sendo exibida no dia previsto de entrega. Também é permitido a edição rápida da tarefa ao clicar nela.

Figura 24 – Página **Relatórios**.



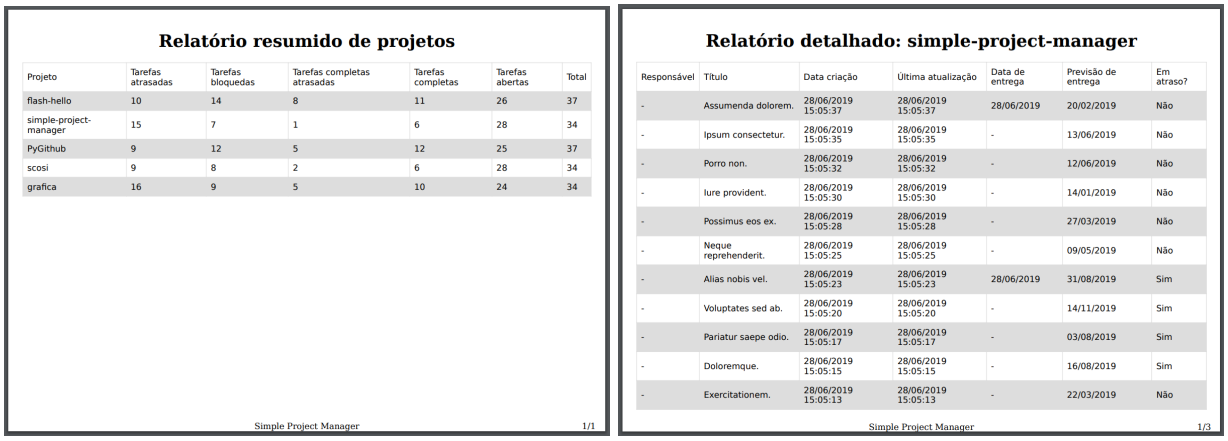
(a) Relatório resumido. ***(b)*** Relatório detalhado.

Fonte: O Autor.

A página **Relatórios** (Figura 24) permite que o gerente de projeto gere um relatório resumido (Figura 24a) de um ou mais projetos ou um relatório detalhado (Figura 24b) por projeto.

Foi implementado dois tipos de relatórios no sistema, o resumido e o detalhado (Figura 25). Ao gerar um relatório resumido (Figura 25a) será disponibilizado um documento PDF contendo uma listagem de cada projeto selecionado, apresentando as seguintes informações: tarefas atrasadas, bloqueadas, completas atrasadas, completas, abertas e total. Já na opção de relatório detalhado (Figura 25b), será gerado um

Figura 25 – Documentos gerados.



Relatório resumido de projetos

Projeto	Tarefas atrasadas	Tarefas bloquedas	Tarefas completas atrasadas	Tarefas completas	Tarefas abertas	Total
flash-hello	10	14	8	11	26	37
simple-project-manager	15	7	1	6	28	34
PyGithub	9	12	5	12	25	37
scosi	9	8	2	6	28	34
grafica	16	9	5	10	24	34

Simple Project Manager 1/1

Relatório detalhado: simple-project-manager

Responsável	Título	Data criação	Última atualização	Data de entrega	Previsão de entrega	Em atraso?
-	Assumenda dolorem.	28/06/2019 15:05:37	28/06/2019 15:05:37	28/06/2019	20/02/2019	Não
-	Ipsum consectetur.	28/06/2019 15:05:35	28/06/2019 15:05:35	-	13/06/2019	Não
-	Porro non.	28/06/2019 15:05:32	28/06/2019 15:05:32	-	12/06/2019	Não
-	Iure provident.	28/06/2019 15:05:30	28/06/2019 15:05:30	-	14/01/2019	Não
-	Possimus eos ex.	28/06/2019 15:05:28	28/06/2019 15:05:28	-	27/03/2019	Não
-	Neque reprehenderit.	28/06/2019 15:05:25	28/06/2019 15:05:25	-	09/05/2019	Não
-	Alias nobis vel.	28/06/2019 15:05:23	28/06/2019 15:05:23	28/06/2019	31/08/2019	Sim
-	Voluptates sed ab.	28/06/2019 15:05:20	28/06/2019 15:05:20	-	14/11/2019	Sim
-	Pariatur saepe odio.	28/06/2019 15:05:17	28/06/2019 15:05:17	-	03/08/2019	Sim
-	Doloremque.	28/06/2019 15:05:15	28/06/2019 15:05:15	-	16/08/2019	Sim
-	Exercitationem.	28/06/2019 15:05:13	28/06/2019 15:05:13	-	22/03/2019	Não

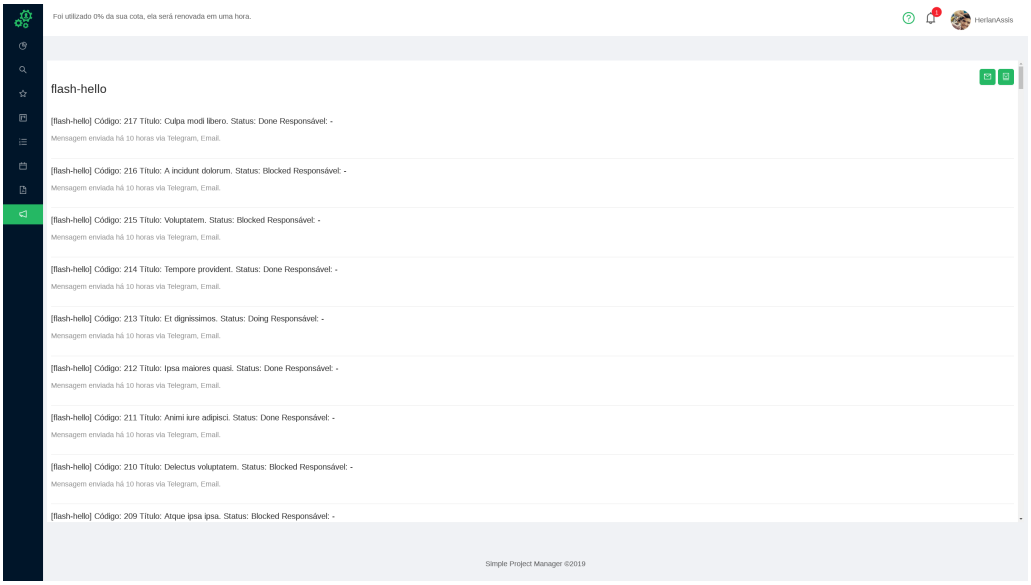
Simple Project Manager 1/3

(a) PDF Relatório resumido. ***(b)*** PDF Relatório detalhado.

Fonte: O Autor.

documento PDF contendo a listagem da situação de cada tarefa no projeto, com as informações: título, data de criação, última atualização, data de entrega, previsão de entrega e em atraso.

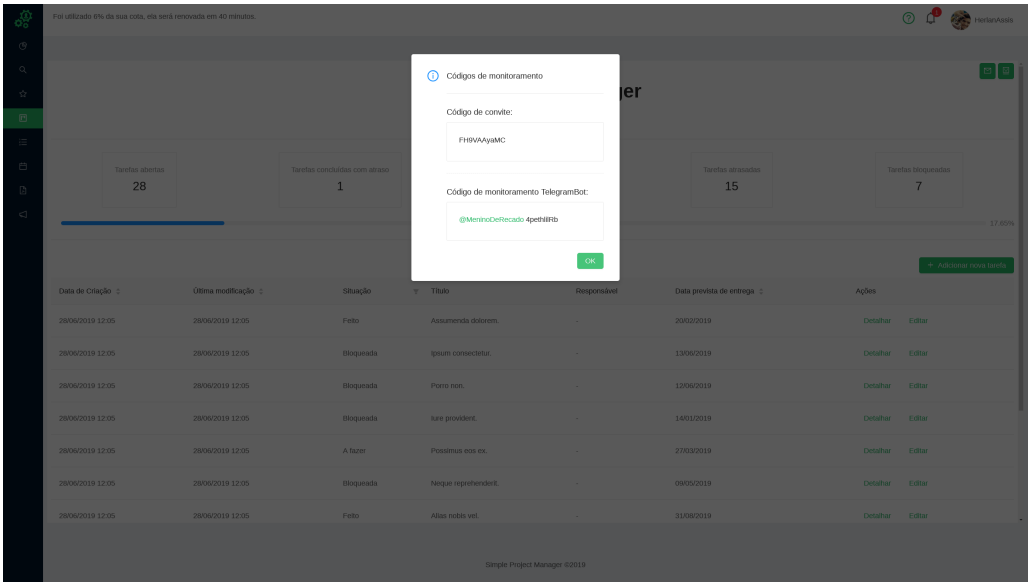
Figura 26 – Página **Notificações**.



Fonte: O Autor.

A página **Notificações** (Figura 26) exibe todas as notificações enviadas pelo email ou telegram para cada projeto monitorado ou gerenciado pelo usuário. Também é possível habilitar ou desabilitar os canais de notificações disponíveis para monitoramento de cada projeto.

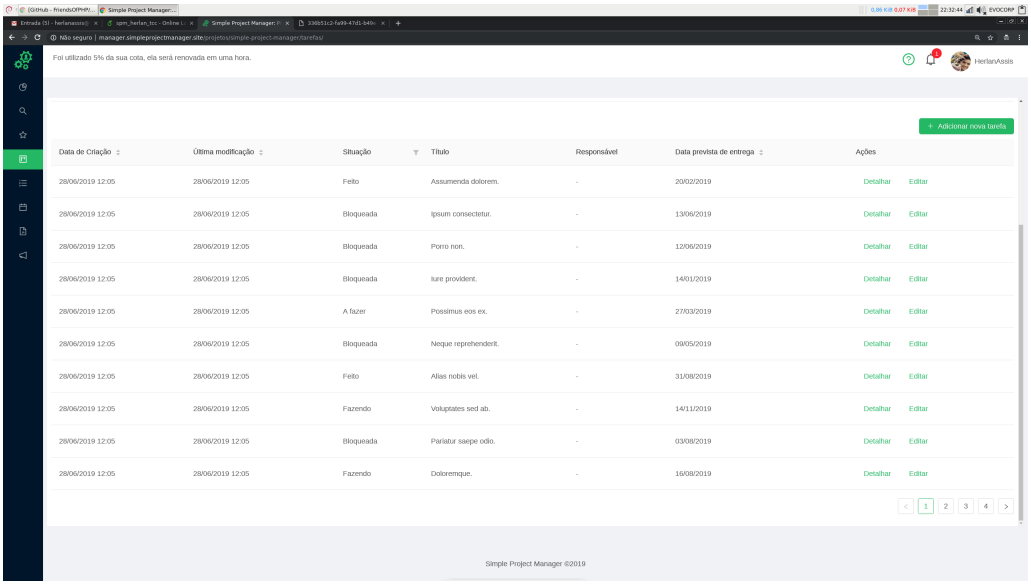
Figura 27 – Códigos de monitoramento.



Fonte: O Autor.

Os dados de monitoramento do projeto (Figura 27) podem ser acessados pela página **Tarefas por projeto** (Figura 18). Informações sobre código de convite, código de acesso do Bot do Telegram, além do número de tarefas abertas, concluídas com atraso, concluídas, atrasadas, bloqueadas e uma barra de progresso do projeto. Também é possível adicionar uma nova tarefa.

Figura 28 – Tarefas do Projeto.

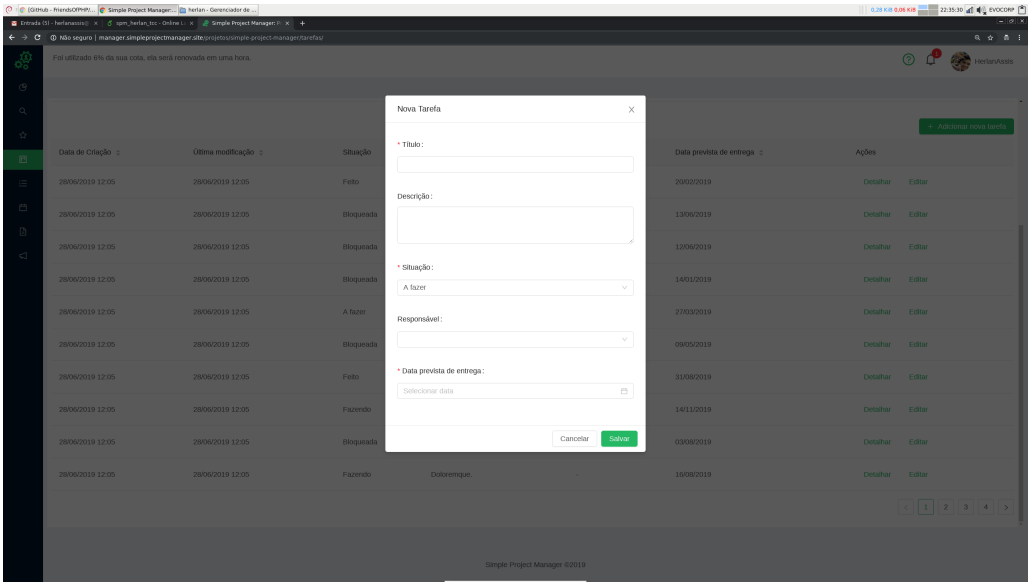


Fonte: O Autor.

Para adicionar uma nova tarefa ao projeto, basta clicar no botão adicionar nova tarefa (Figura 28) e preencher o formulário (Figura 29) com o título da tarefa, descrição, situação, responsável e data prevista de entrega. Caso o usuário deseje realizar alguma

alteração na tarefa cadastrada, basta clicar na opção de "editar"na tarefa selecionada, preencher/modificar os dados no formulário e salvar.

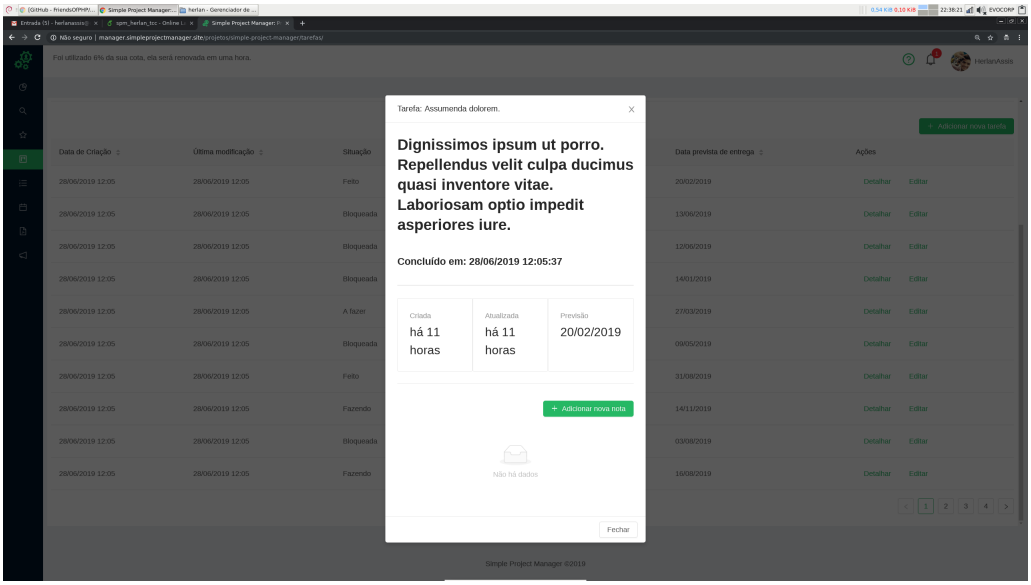
Figura 29 – Inserindo uma nova tarefa.



Fonte: O Autor.

A Figura 28, por sua vez, exibe uma tabela com todas as tarefas cadastradas para um determinado projeto, com os seguintes detalhes: data de criação, última modificação, situação, título, responsável, data prevista de entrega e acesso as ações de detalhar e editar.

Figura 30 – Detalhando uma tarefa.

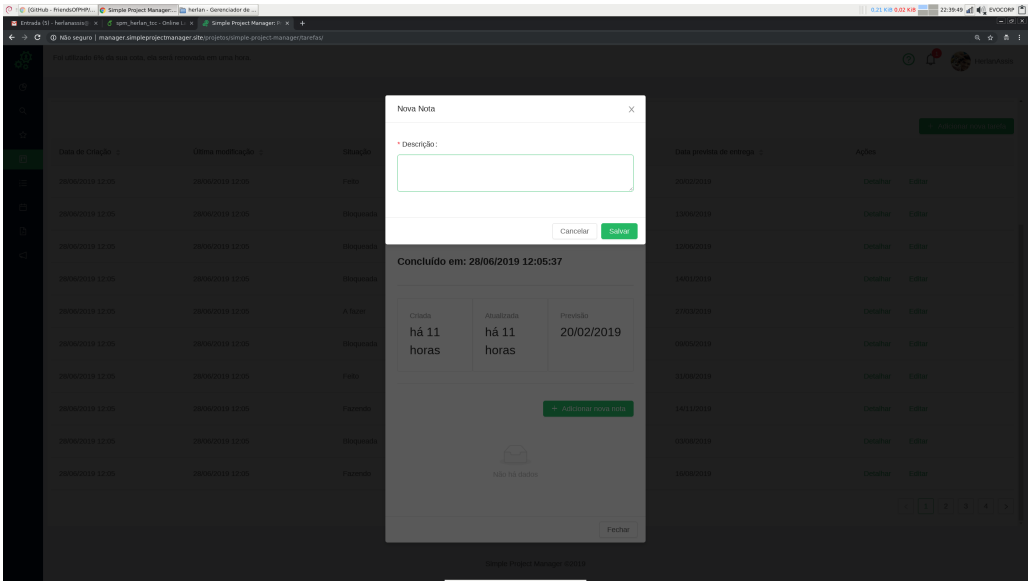


Fonte: O Autor.

Ao clicar na ação de detalhar tarefa, uma janela flutuante será exibida na tela

(Figura 30), nela é possível visualizar todas as informações da tarefa. Além disso, é permitido cadastrar uma nota (lembrete ou anotação) e ler todas as notas já cadastradas.

Figura 31 – Inserindo uma nova nota.



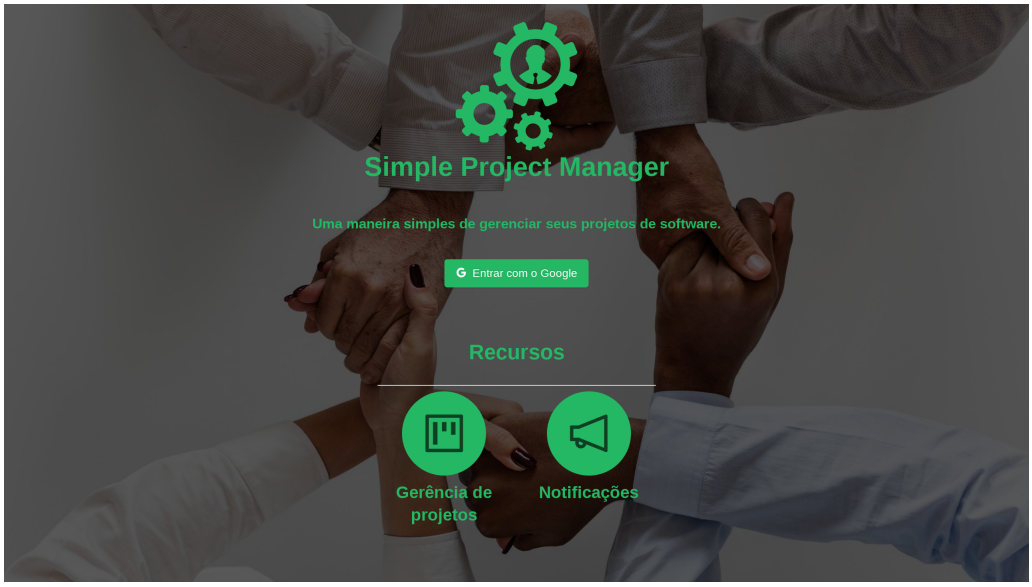
Fonte: O Autor.

Para adicionar uma nova nota basta clicar no botão "adicionar nova", após isso uma janela flutuante será exibida com o formulário de cadastro (Figura 31).

5.2 Client

Nesta sessão serão comentadas as capturas de telas feitas da aplicação **Client**. Esta aplicação tem o propósito de aproximar o cliente do gerente de projetos, ao fornecer uma maior transparência sobre o que está sendo feito no projeto.

Figura 32 – Tela de Login.

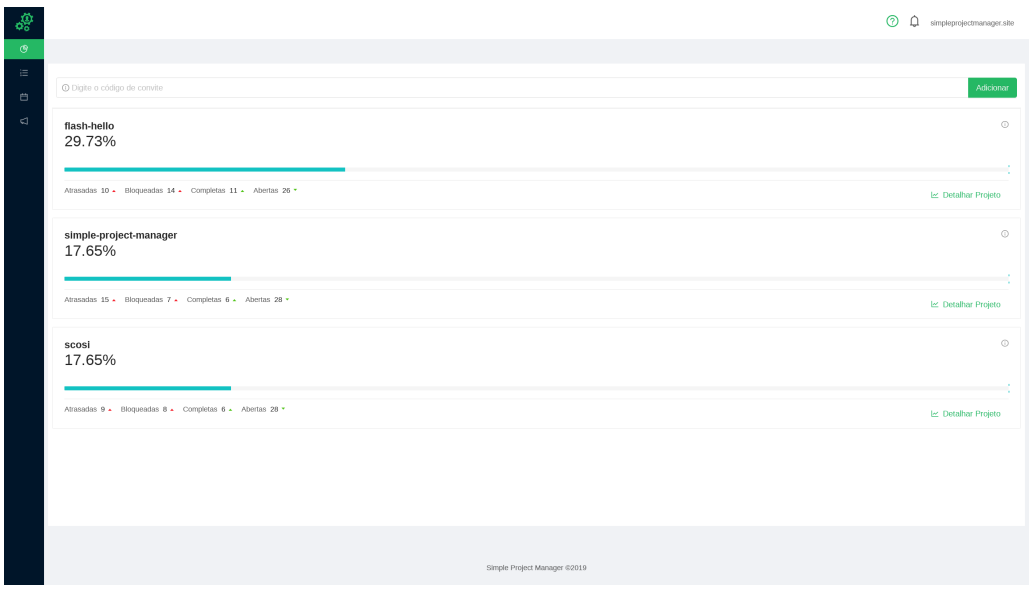


Fonte: O Autor.

A Figura 32 mostra a tela de login do aplicativo **Client**, os recursos da aplicação e o login social pelo Google. Os principais recursos apresentados são: monitoramento de projetos e notificações.

Para utilizar a aplicação é necessário possuir uma conta ativa no Google, isso é obrigatório porque garante informações de contato (email) entre o cliente e o gerente de projeto e agiliza o ingresso do usuário no sistema. O uso de login social com o Google dispensa o preenchimento de, por exemplo, um formulário de cadastro para começar a utilizar a aplicação.

Figura 33 – Tela principal.



Fonte: O Autor.

A Figura 33 apresenta a tela inicial da aplicação para um usuário que contém as principais informações dos projetos monitorados e notificações importantes.

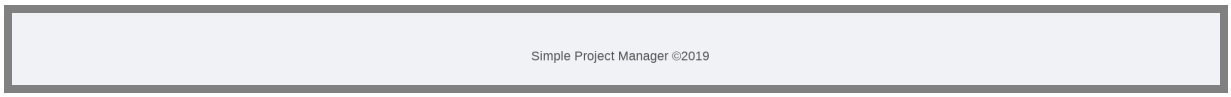
A aplicação **Client** pode ser dividida em 4 partes: menu, cabeçalho, corpo da página e rodapé. A Figura 34 ilustra essa divisão.

Figura 34 – Estrutura da aplicação.

(a) Cabeçalho.

(b) Menu.

(c) Corpo.



(d) Rodapé.

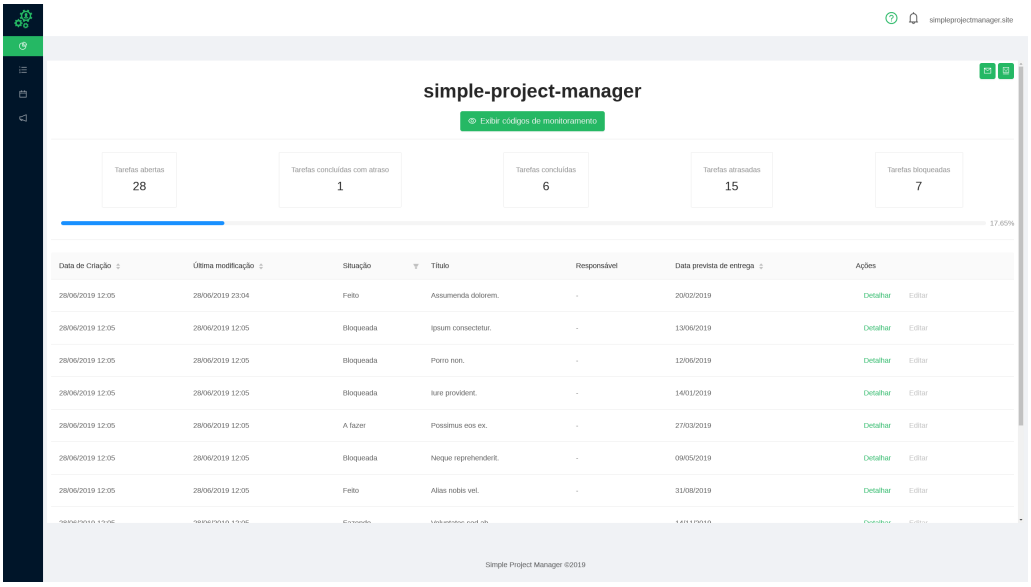
Fonte: O Autor.

O menu marcado na Figura 34 contém as páginas: Home, Tarefas, Agenda e Notificações – nos próximos parágrafos essas páginas serão comentadas com mais detalhes. O cabeçalho selecionado na Figura 34a contém as notificações do sistema e dados do usuário logado. A partir desta cabeçalho o usuário pode realizar o logout da aplicação. O menu exibe os botões de navegação (Figura 34b). O corpo da página exibido na Figura 34c mostra informações referentes a cada página da aplicação. Por último, o rodapé (Figura 34d) apresenta informações de direitos autorais.

As páginas da aplicação serão apresentadas na ordem em que foram mencionadas anteriormente.

De forma similar à página **Home** do gerente de projetos (Figura 33), o cliente tem acesso a lista de todos os projetos monitorados, exibindo informações sobre o seu progresso, quantidade de tarefas atrasadas, bloqueadas, completas e abertas.

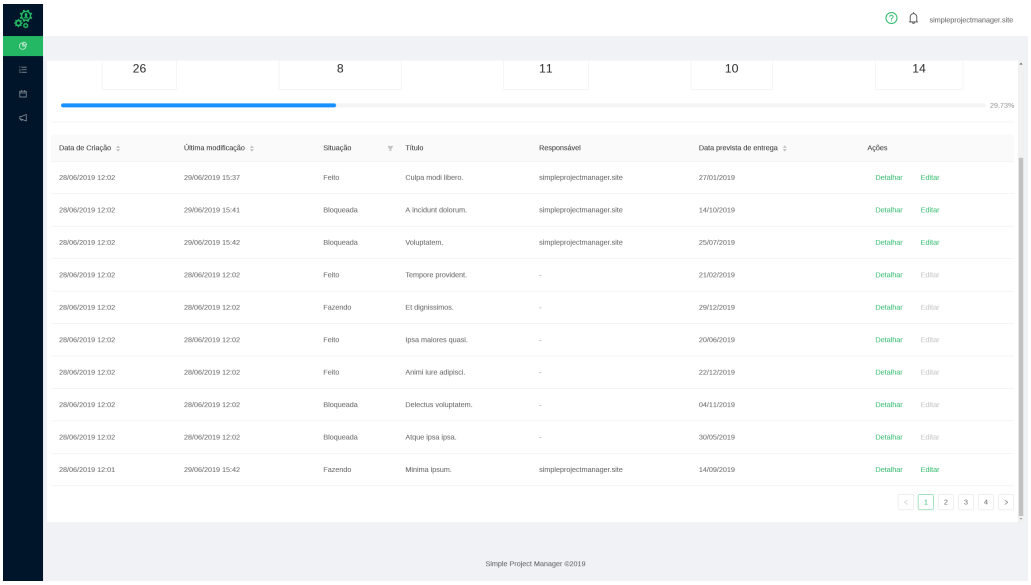
Figura 35 – Página **Detalhes do Projeto**.



Fonte: O Autor.

Para acessar a página **Detalhes do Projeto** (Figura 35) basta clicar no botão detalhes do projeto (Figura 33). A partir dessa página é possível visualizar o código de monitoramento do Telegram, quantidade de tarefas abertas, concluídas com atraso, completas, atrasadas, bloqueadas e o progresso do projeto.

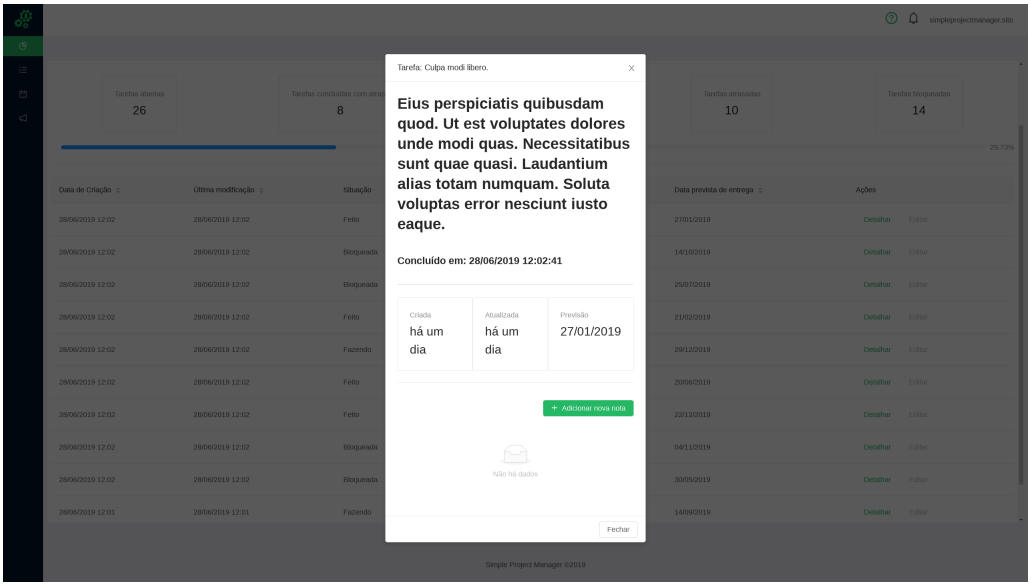
Figura 36 – Tabela de tarefas.



Fonte: O Autor.

Ainda na página **Detalhes do Projeto** (Figura 35), é mostrada uma tabela com todas as tarefas do projeto (Figura 36). Nessa tabela é exibida informações da data de criação da tarefa, última modificação, situação, título, responsável, data prevista de entrega e ações de detalhar e editar tarefa.

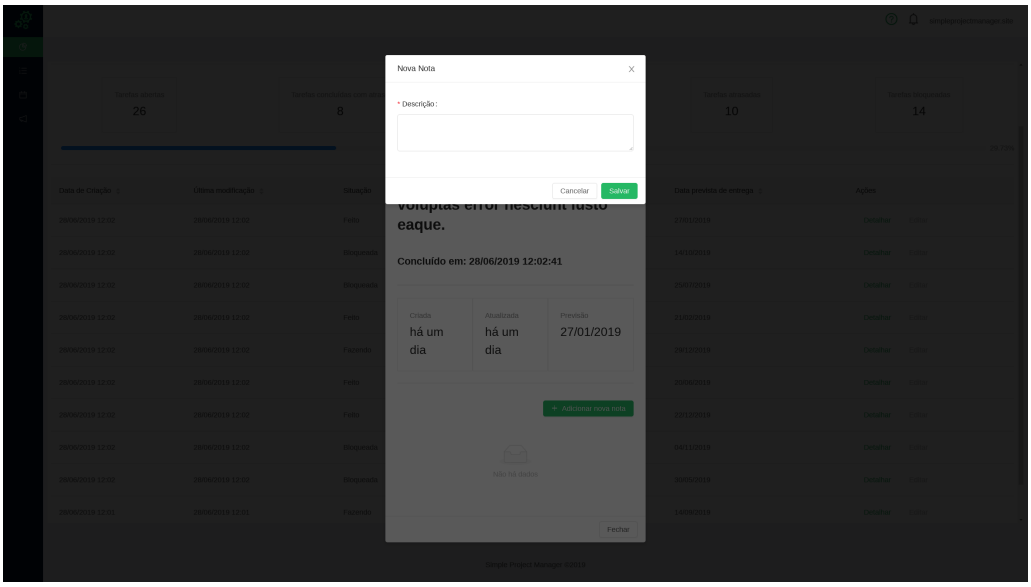
Figura 37 – Detalhes da tarefa.



Fonte: O Autor.

Ao detalhar uma tarefa (Figura 37), as informações sobre responsável, *status*, descrição, data de criação, prazo estimado de entrega, data de conclusão (caso esteja completa) e notas estarão visíveis.

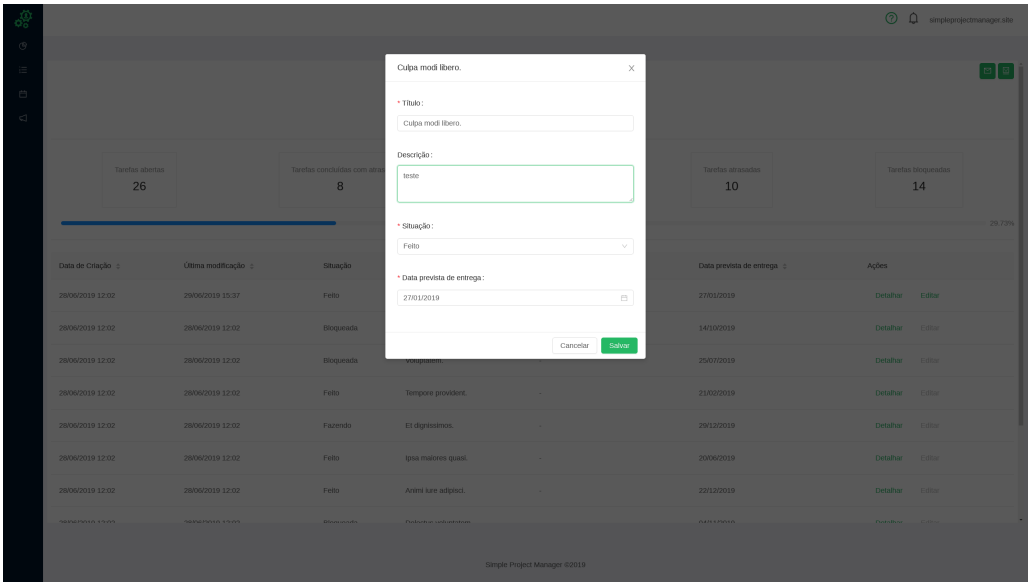
Figura 38 – Inserindo uma nova nota.



Fonte: O Autor.

Ao clicar em adicionar uma nova nota, o cliente pode cadastrar uma nova nota para a tarefa ao preencher o campo descrição no formulário (Figura 38).

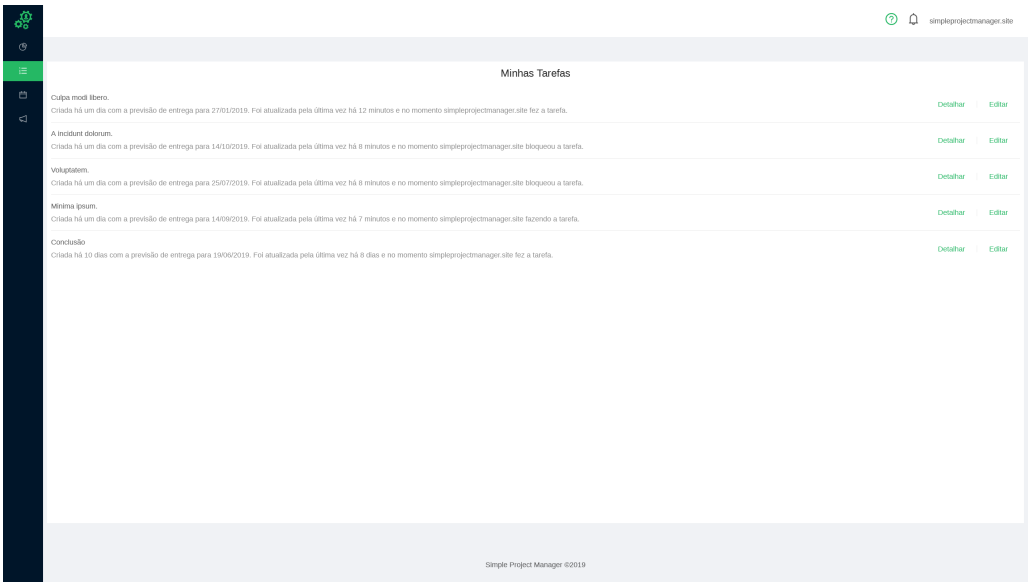
Figura 39 – Editar uma tarefa.



Fonte: O Autor.

Para atualizar os dados de uma tarefa atribuída ao cliente, basta clicar na ação de editar e preencher/modificar os dados no formulário e salvar (Figura 39).

Figura 40 – Página **Tarefas**.

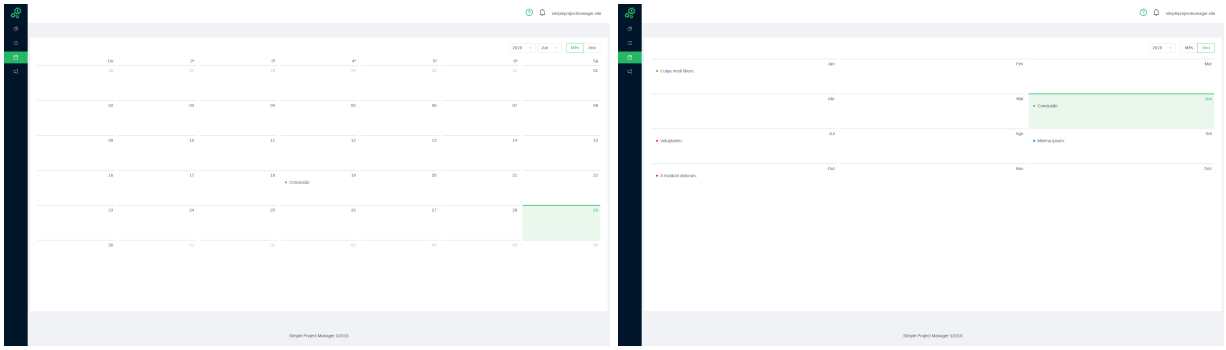


Fonte: O Autor.

A página **Tarefas** (Figura 40) contém uma lista de todas as tarefas do cliente, onde são exibidas informações sobre a data de criação, última atualização, previsão de entrega e status da tarefa. Para cada tarefa é permitido edita-la.

A página **Agenda** (Figura 41) exibe uma agenda que pode ser visualizada por mês (Figura 41a) ou ano (Figura 41b). Ela contém a lista de tarefas que foram atribuídas

Figura 41 – Página **Agenda**.

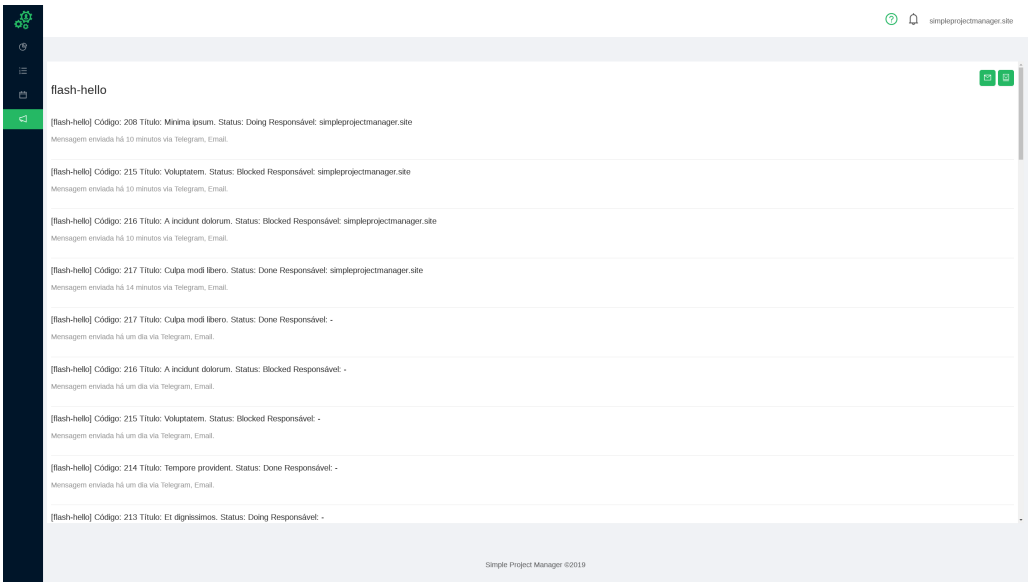


(a) Menu. ***(b)*** Corpo.

Fonte: O Autor.

ao cliente, sendo exibida no dia previsto de entrega. Também é permitido a edição rápida da tarefa ao clicar na mesma.

Figura 42 – Página **Notificações**.



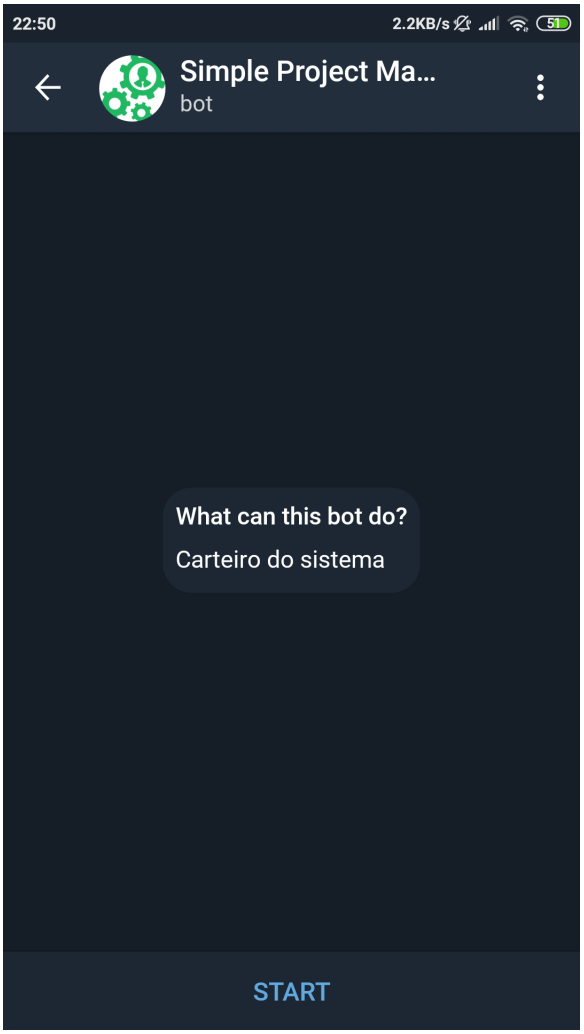
Fonte: O Autor.

A página **Notificações** (Figura 42) exibe todas as notificações enviadas pelo email ou telegram para cada projeto monitorado. Também é possível habilitar ou desabilitar os canais de notificações disponíveis para monitoramento de cada projeto.

5.3 MeninoDeRecado_Bot

Nesta sessão serão comentadas as capturas de telas feitas do Bot **MeninoDeRecado_Bot**, ele tem o propósito de notificar os usuários do sistema.

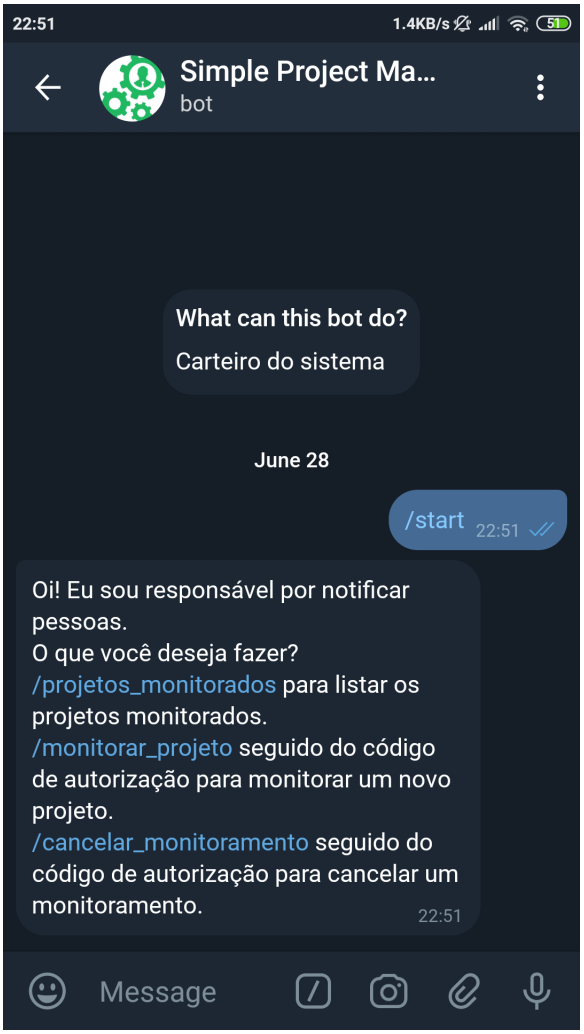
Figura 43 – Bot Telegram **@MeninoDeRecado_Bot**: Início.



Fonte: O Autor.

O **MeninoDeRecado_Bot** pode ser utilizado pelo gerente, colaborador ou cliente para auxiliar no monitoramento de projetos. Para utiliza-lo é preciso possuir uma conta ativa no Telegram, buscar por @MeninoDeRecadoBot_Bot na barra de busca do aplicativo e clicar no botão "começar/start"(Figura 43).

Figura 44 – Bot Telegram **@MeninoDeRecado_Bot**: Começar.



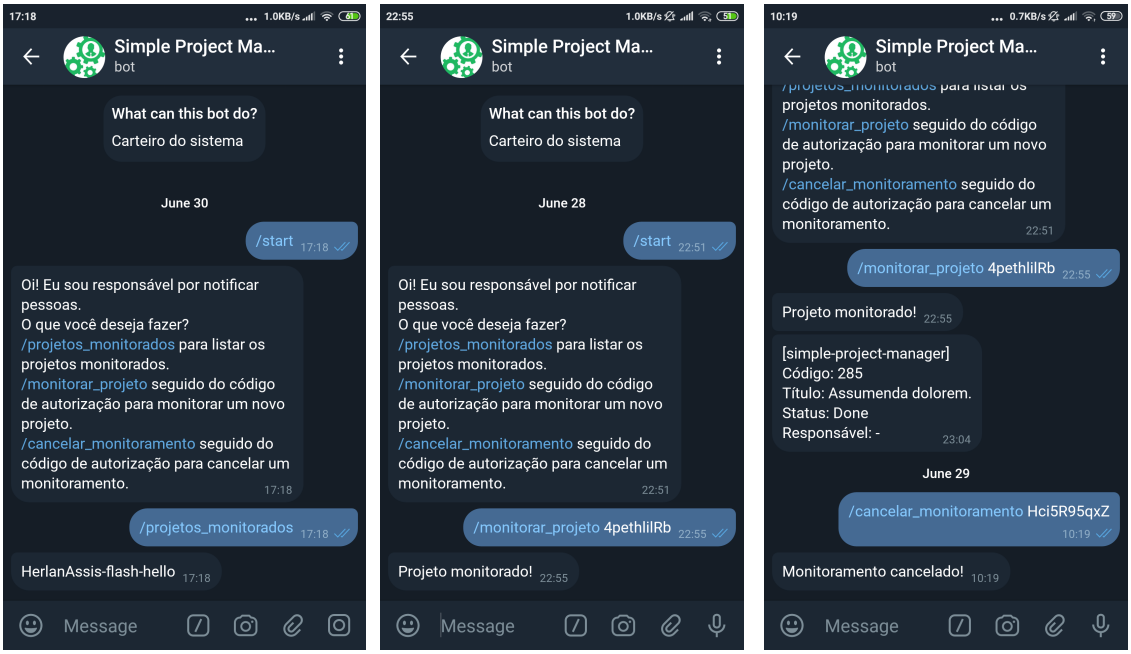
Fonte: O Autor.

Uma vez habilitado (Figura 44), uma mensagem de ajuda vai ser enviada ao usuário explicando o funcionamento do Bot e instruindo como utilizar as funcionalidades presentes nele.

O Bot possui 3 comandos, que são:

- "projetos_monitorados": exibe uma lista com o nome de todos os projetos monitorados (Figura 45a);
- "monitorar_projeto": permite monitorar um novo projeto ao informar o código de monitoramento do Telegram (Figura 45b);
- "cancelar_monitoramento": permite cancelar o monitoramento de um projeto ao informar o seu código (Figura 45c);

Figura 45 – Bot Telegram **@MeninoDeRecado_Bot**: Comandos.



(a) Comando começar. ***(b)*** Monitorando projeto. ***(c)*** Monitoramento cancelado.

Fonte: O Autor.

Figura 46 – Bot Telegram **@MeninoDeRecado_Bot**: Notificação.

Fonte: O Autor.

Quando uma nova tarefa é cadastrada ou alterada, uma nova notificação é enviada ao usuário (Figura 46). O corpo da mensagem contém as informações do código da tarefa, título, status e responsável da tarefa.

5.4 Considerações do Capítulo

Neste capítulo foi apresentado o resultado deste trabalho: as aplicações **Manager** e **Client**, e o Bot **@MeninoDeRecado_Bot**. Na Sessão 5.1 foi apresentado as principais funcionalidades da aplicação **Manager** e como ela pode ajudar o gerente de projetos a gerir sua equipe de desenvolvedores. De forma semelhante, na Sessão 5.2 foi detalhado os recursos que a aplicação **Client** dispõe, além de explicar como ela pode melhorar a comunicação com o gerente de projetos. Por fim, na Sessão 5.3, foi mostrado os comandos do Bot e como ele pode melhorar o conhecimento do estado do projeto pelos envolvidos através de notificações pelo Telegram.

6

Conclusão

Este trabalho monógrafo propôs o desenvolvimento de uma plataforma gratuita para auxiliar o gerente de projetos na gestão de projetos de software. Além disso, a ferramenta possibilita uma comunicação mais estreita entre o gerente e o cliente.

O objetivo deste trabalho foi cumprido ao fornecer uma ferramenta capaz de unificar o gerenciamento de tarefas, comunicação com o cliente e a análise da contribuição individual de cada desenvolvedor de projetos de software do GitHub.

Neste contexto, é importante enfatizar que a utilização de métodos e ferramentas para avaliar, interagir e analisar projetos de software é de grande importância, pois além de melhorar o desempenho e comunicação entre os envolvidos no projeto, possibilita ainda identificar gargalos e outros empecilhos durante o desenvolvimento do produto.

6.1 Trabalhos Futuros

Durante o desenvolvimento deste trabalho, surgiu a necessidade de implementar novos recursos para aperfeiçoar a finalidade da aplicação, tais como:

- Utilizar a ferramenta desenvolvida neste trabalho em um cenário real, com o propósito de avaliar se ela atende as necessidades do gerente de projeto, desenvolvedores e melhora a comunicação com o cliente além de estreitar a sua comunicação com gestor;
- Adicionar suporte a outros sistemas online de controle de versão, como o Bitbucket e Gitlab;
- Aprimorar a ferramenta para analisar outras métricas de software, como a análise da complexidade ciclotemática do código e *commits* que introduziram *bugs*, falha em testes, além da quantidade de dívidas técnicas introduzidas em cada *commit*;
- Realizar a integração com ferramentas de CI (*Continuous Integration)*, como o Travis CI ou Circle CI – serviços de integração contínua tem informações relevan-

tes sobre o número de *deploys* feito em um determinado período de tempo ou o número de *commits* com *bugs* que foram enviados para o repositório *master*;

- Vincular a tarefa criada a uma *Issue* no repositório de software;
- Criar aplicativos para smartphones.

Referências

ABNT, A. B. d. N. T. *Engenharia de software - Qualidade de produto Parte 1: Modelo de qualidade*. Rio de Janeiro, RJ, 2003. v. 2003. Citado na página 22.

ADOLPHO, C. *Os 8 Ps do Marketing Digital: O guia estratégico de marketing digital*. [S.l.]: Novatec Editora, 2011. Citado na página 16.

BENZECRY, F. S. Metodologias àgeis para gerenciamento de projetos de inovação e pesquisa e desenvolvimento. 2017. Citado 2 vezes na(s) página(s) 21 e 28.

BERNARDO, J. H.; COSTA, D. A. da; KULESZA, U. Studying the impact of adopting continuous integration on the delivery time of pull requests. In: IEEE. *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*. [S.l.], 2018. p. 131–141. Citado na página 13.

BROOKS, F. P. *The Mythical Man-Month: Essays on Software Engineering*. [S.l.]: Addison-Wesley, 1975. ISBN 9780201006506. Citado na página 16.

CARVALHO, P. R. F. de et al. Desenvolvimento de uma ferramenta web para o gerenciamento de projeto de software utilizando metodologias 100% ágeis. *RE3C-Revista Eletrônica Científica de Ciência da Computação*, v. 9, n. 1, 2014. Citado na página 25.

CHACON, S. *Pro Git*. Berkeley, CA New York, NY: Apress,Distributed to the Book trade worldwide by Spring Science+Business Media, 2014. ISBN 978-1484200773. Citado na página 32.

COSTA, D. A. d. *Avaliação da contribuição de desenvolvedores para projetos de software usando mineração de repositórios de software e mineração de processos*. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Norte, 2013. Citado na página 23.

COSTA, D. A. da et al. Unveiling developers contributions behind code commits: an exploratory study. In: ACM. *Proceedings of the 29th Annual ACM Symposium on Applied Computing*. [S.l.], 2014. p. 1152–1157. Citado na página 13.

DEMARCO, T. *Controlling Software Projects*. [S.l.]: Yourdon Press, 1982. ISBN 0917072324. Citado na página 17.

DOCKER. *What is a Container?* 2019. <https://www.docker.com/resources/what-container>. Accessado: 26/06/2019. Citado 2 vezes na(s) página(s) 30 e 31.

GOOGLE. *Why Google Cloud*. 2019. <https://cloud.google.com/why-google-cloud/>. Accessado: 26/06/2019. Citado na página 30.

HASSAN, A. E. Mining software repositories to assist developers and support managers. In: IEEE. *2006 22nd IEEE International Conference on Software Maintenance*. [S.l.], 2006. p. 339–342. Citado na página 13.

HASSAN, A. E. The road ahead for mining software repositories. In: IEEE. *2008 Frontiers of Software Maintenance*. [S.l.], 2008. p. 48–57. Citado na página 22.

LIMA, J. R. F. D. *Uma abordagem de apoio à gerência de projetos de software para análise da contribuição de desenvolvedores*. Dissertação (Mestrado) — Brasil, 2014. Citado 4 vezes na(s) página(s) 22, 24, 25 e 27.

MANIFESTO, A. Manifesto for agile software development. In: ____. [s.n.], 2001. Disponível em: <https://agilemanifesto.org/>. Acesso em: 24 abr. 2019. Citado na página 19.

MARIOTTI, F. S. Kanban: o ágil adaptativo. *Engenharia de Software Magazine*, v. 45, n. 4, p. 6–10, 2012. Citado 2 vezes na(s) página(s) 21 e 22.

MEIRELLES, P. R. M. *Monitoramento de métricas de código-fonte em projetos de software livre*. Tese (Doutorado) — Universidade de São Paulo, 2013. Citado 3 vezes na(s) página(s) 22, 24 e 27.

PMBOK, G. *Um Guia de Conhecimento em Gerenciamento de Projetos*. [S.l.]: Saraiva, 2012. ISBN 8502223720. Citado na página 17.

RAYMUNDO, R.; LACERDA, G. Estudo e implementação de uma ferramenta para gerência de projetos baseado em metodologias ágeis. Citado na página 18.

SOMMERVILLE, I. *Engenharia de software*. São Paulo: Addison Wesley, 2003. ISBN 978-8588639072. Citado 6 vezes na(s) página(s) 16, 17, 18, 19, 23 e 27.

SOMMERVILLE, I. *Engenharia de software*. São Paulo: Pearson Prentice Hall, 2011. ISBN 978-8579361081. Citado na página 20.

SOTILLE, M. Gerenciamento de projetos na engenharia de software. *DOI: http://www. pmtech. com. br/artigos/Gerenciamento_Projetos_Software. pdf*, 2014. Citado 3 vezes na(s) página(s) 16, 18 e 27.

WANG, S.; LO, D.; JIANG, L. An empirical study on developer interactions in stackoverflow. In: ACM. *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. [S.l.], 2013. p. 1019–1024. Citado na página 13.

ZHANG, T.; LEE, B. A hybrid bug triage algorithm for developer recommendation. In: ACM. *Proceedings of the 28th annual ACM symposium on applied computing*. [S.l.], 2013. p. 1088–1094. Citado na página 13.