UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
UNIDADE ACADÊMICA DE SISTEMAS E COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

THAYNAN ANDREY ROCHA NUNES

GIT DASHBOARD:
DASHBOARD DE PULL REQUESTS PARA O GITHUB

CAMPINA GRANDE - PB

2019

THAYNAN ANDREY ROCHA NUNES

GIT DASHBOARD:
DASHBOARD DE PULL REQUESTS PARA O GITHUB

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Professor Dr. Matheus Gaudencio do Rêgo.

CAMPINA GRANDE - PB

2019

N972g Nunes, Thaynan Andrey Rocha.
Git dashboard : dashboard de pull requests para o github. / Thaynan Andrey Rocha Nunes. – 2019.

13 f.

Orientador: Prof. Dr. Matheus Gaudencio do Rêgo
Trabalho de Conclusão de Curso - Artigo (Curso de Bacharelado em Ciência da Computação) - Universidade Federal de Campina Grande; Centro de Engenharia Elétrica e Informática.

1. Gerência de projetos. 2. Programação. 3. Controle de versão. 4. Pull requests. I. Rêgo, Matheus Gaudencio do. II. Título.

CDU:004(045)

Elaboração da Ficha Catalográfica:

Johnny Rodrigues Barbosa
Bibliotecário-Documentalista
CRB-15/626

THAYNAN ANDREY ROCHA NUNES

GIT DASHBOARD:
DASHBOARD DE PULL REQUESTS PARA O GITHUB

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Professor Dr. Matheus Gaudencio do Rêgo
Orientador – UASC/CEEI/UFCG

Professora Dr. Cláudio Elízio Calazans Campelo
Examinador – UASC/CEEI/UFCG

Professor Dr. Tiago Lima Massoni
Examinador – UASC/CEEI/UFCG

Trabalho aprovado em: 02 de julho de 2019.

CAMPINA GRANDE - PB

Git Dashboard: dashboard de Pull Requests para o Github

Thaynan Nunes
Orientador: Matheus Gaudencio
Departamento de Sistemas e Computação
Universidade Federal de Campina Grande
Campina Grande, Paraíba, Brasil
thaynanandrey@gmail.com

RESUMO

O gerente de projetos é responsável por organizar o desenvolvimento do sistema, definindo-se atividades como a criação de novas funcionalidades e correções de *bugs*. Para acompanhar tais atividades, o mesmo pode analisar pull requests (PR), pois estes apresentam qual tarefa está sendo realizada, o responsável e o estado atual das mudanças realizadas. Contudo, o gerente precisa por vezes acompanhar PRs em diferentes repositórios para gerenciar o desenvolvimento de uma funcionalidade, o que se caracteriza um trabalho manual complexo. Logo, o **Git Dashboard** propicia o acompanhamento de PRs de forma automatizada em uma plataforma web, onde o gerente poderá criar projetos que representam atividades do sistema, vinculando repositórios e PRs aos mesmos para acompanhar as informações desses e assim discernir o andamento no desenvolvimento do sistema. Para avaliar a qualidade do sistema, realizou-se um estudo da usabilidade do mesmo, no qual 7 alunos de graduação do curso de Ciência da Computação da Universidade Federal de Campina Grande com experiência em trabalhar com PRs do GitHub, testaram o sistema e responderam ao questionário PSSUQ (Post-Study System Usability Questionnaire), inferindo sua perspectiva no uso geral do sistema. Como resultado, foi-se calculada a média das respostas para cada item do questionário, as quais variaram no intervalo de 1,43 a 2,71, em uma escala de 1 a 7, onde 1 representa uma recepção bastante positiva em relação ao tópico avaliado, e 7 uma recepção bastante negativa. Logo, dada a média das resposta, inferiu-se que os usuários estavam satisfeitos por usar o sistema, concluindo-se que a plataforma se apresentou utilizável e aplicável para a gerência de atividades.

PALAVRAS-CHAVE

gerência de projetos, programação, controle de versão.

Repositório

https://github.com/ThaynanAndrey/tcc-git-dashboard

1 Introdução

O gerente de projetos tem a função de acompanhar o desenvolvimento de um sistema, mantendo o conhecimento sobre o estado atual do mesmo e definindo as próximas atividades. Quando o projeto está hospedado no GitHub, um meio de realizar essa gestão é através de pull requests (PR), pois os mesmos apresentam as atividades que estão sendo executadas, as mudanças realizadas e o estado atual das implementações, caracterizando-se como informações cruciais para o acompanhamento rápido e preciso.

Não obstante, a medida que são criados novos repositórios ou inseridas novas pessoas no desenvolvimento do sistema, aumenta-se a complexidade da gestão do projeto, pois são mais atividades sendo executadas ao mesmo tempo, as quais estão sendo realizadas em diferentes repositórios. Logo, o gerente precisa sempre está alinhando o grande número de PRs produzidos às suas respectivas atividades, para assim saber o estado atual do desenvolvimento e projetar os próximos passos.

Buscando sanar essa problemática, foi-se produzido o Git Dashboard, sistema que permite que o usuário crie projetos, nos quais ele poderá vincular repositórios e PRs, e acompanhe-nos de forma rápida e simples, pois são apresentadas as informações cruciais dos PR vinculados, como título, última atualização e estado dos mesmos, propiciando ao gerente, a capacidade de entender o estado atual de desenvolvimento das atividades e projetar os planos futuros.

Para avaliar a qualidade de uso do sistema, foi-se realizado um estudo da usabilidade do mesmo, a partir da sondagem de usuários através de um questionário PSSUQ (Post-Study System Usability Questionnaire), no qual 7 alunos de graduação do curso de Ciência da Computação da Universidade Federal de Campina Grande, os quais têm experiência com o GitHub e o envio de mudanças a partir de PRs, testaram as funcionalidades do sistema e responderam o questionário, avaliando se de fato o sistema é utilizável, possui as ferramentas esperadas e vai auxiliar um gerente no trabalho do seu dia a dia. Como resultado, as pessoas se demonstraram satisfeitas com o uso da aplicação e a avaliaram como uma possível solução para a problemática apresentada.

Nesta perspectiva, o presente artigo tem o objetivo de apresentar o desenvolvimento do Git Dashboard, aplicação que busca auxiliar o gerente de projetos a acompanhar os desenvolvedores em suas tarefas diárias.

1.1 Desenvolvimento de software no GitHub

O desenvolvimento de um software é composto por um time que pode crescer em tamanho à medida que o sistema cresce em funcionalidades e se torna mais importante para o público. As pessoas desse time se distribuem em várias responsabilidades, como desenvolvedores, testadores, líderes, gerentes e designers, e são a organização e realização de suas tarefas que propiciam o crescimento do sistema com fluidez, qualidade e manutenibilidade.

Um software é, em sua essência, um código que está organizado em vários arquivos que conjuntamente permitem a execução do sistema. No entanto, estes arquivos estão em constante mudança por conta, por exemplo, de novas funcionalidades sendo implementadas, correções de bugs ou refatorações de códigos. E são nestas alterações que duas ou mais pessoas podem estar lidando simultaneamente com o mesmo arquivo, o que acarreta versões distintas desse programa.

Nesta perspectiva, buscando-se facilitar a gerência e integração destas diferentes versões, foram desenvolvidos sistemas de controle de versões de arquivos, permitindo registrar mudanças feitas ao longo do tempo e, assim, propiciar a recuperação de versões específicas. Um desses sistemas foi o **Git**, o qual permite o controle de versões do código por meio de *commits*, os quais representam a consolidação (versionamento) de um conjunto de alterações realizadas para o desenvolvimento de uma tarefa.

A partir do Git, foi desenvolvido o **GitHub**[1], plataforma responsável pela hospedagem do controle de versões do Git em um repositório remoto. Esse repositório é estruturado como uma árvore, na qual existe a ramificação principal, onde está o código atual do sistema, e ramificações criadas a partir dessa, onde serão realizadas mudanças no código, que posteriormente serão mescladas com a principal, e, assim, propiciarão a estruturação das versões do código [1].

Contudo, com o crescimento do sistema, tornava-se difícil gerenciar todas as mudanças realizadas no código que entrariam na ramificação principal. Logo, o GitHub desenvolveu o **Pull Request**, uma nova metodologia para o envio de alterações do código [2], o qual permite que o responsável pelas alterações envie suas mudanças no formato de uma PR, e um outrem avalie essas mudanças, podendo sugerir alterações até o momento que ambos envolvidos entrarem em um acordo sobre todas estas modificações, e, assim, a ramificação criada será finalmente mesclada com a principal. Isso propicia um maior gerenciamento na evolução do código, buscando manter um padrão de desenvolvimento e tentando evitar a inserção de bugs no sistema.

1.2 Gerente de projetos

O gerente de um projeto é o responsável por acompanhar o desenvolvimento do sistema, lidando diariamente com as pessoas que estão trabalhando no mesmo. Ele tem de definir a prioridade das atividades e, em muitas situações, as equipes que as executarão. Logo, ele está sempre a par das mudanças que serão realizadas e os respectivos responsáveis por estas alterações.

Existem vários meios de realizar esta coordenação, e, no contexto de um software, o acompanhamento das mudanças a partir dos PRs é uma possível solução, pois os mesmos possuem dados que informam ao gerente quais atividades estão sendo realizadas e suas respectivas mudanças. Isso propicia que o gerente esteja ciente do que está sendo desenvolvido e permite que o mesmo possa definir uma ordem dessas atividades, como, por exemplo, definir a prioridade dos PR e a ordem para mesclá-los com a ramificação principal, além de estar ciente do trabalho de cada desenvolvedor.

Não obstante, Poo-Caamaño [3] suscita que existem muitas dificuldades para acompanhar todas as mudanças a serem enviadas ao código do projeto, descrevendo as problemáticas no gerenciamento do ecossistema GNOME. Logo, acompanhar o desenvolvimento de um projeto a partir dos PRs, pode se tornar bastante complexo quando a quantidade de PRs aumenta e o coordenador tem de definir qual a ordem de prioridade desses PR e como deve-se mesclá-los com a ramificação principal.

A complexidade pode se intensificar quando o coordenador tem de acompanhar vários repositórios que pertencem a um mesmo projeto, tendo de sincronizar os vários PR criados nestes repositórios, porque, em muitas situações, por exemplo, uma nova funcionalidade de alto nível implementada pode acarretar em mudanças em mais de um repositório. Logo, existirão PRs associados a esta funcionalidade criados nestes repositórios e que precisarão ser mesclados simultaneamente para que a funcionalidade esteja de fato disponível no sistema.

Nesta perspectiva, vislumbre a situação do gerente de projetos, Pedro, o qual gerencia um sistema de e-commerce e deseja introduzir a funcionalidade de realizar a compra de um combo de produtos. Para iniciar a implementação, existem duas equipes envolvidas, uma equipe responsável pelo repositório que fornece a funcionalidade de compra de produtos, e tem de propiciar a venda de itens combinados, e uma equipe responsável pelo repositório que permite a organização da entrega do produto, e terá a responsabilidade de possibilitar que itens possam ser enviados em conjunto.

Essas equipes, ao longo do desenvolvimento da funcionalidade, estarão submetendo PRs a seus respectivos repositórios, até o momento que concluírem as suas responsabilidades, e assim, concluírem o novo

[1] https://github.com/

requisito. Para coordenar a conclusão da funcionalidade, Pedro precisa está acompanhando a evolução desses repositórios, para saber as dificuldades enfrentadas nos mesmos, o seu estado atual e assim projetar as próximas atividades e até o lançamento desse novo recurso.

Tendo em vista a situação de Pedro, observa-se o quão é complexo para ele, acompanhar pull requests de dois repositórios para uma mesma atividade. Não obstante, essa complexidade pode crescer à medida que o número de repositórios, pessoas e PRs aumentam em um sistema, o que acarreta em um acompanhamento não adequado do desenvolvimento do sistema.

1.3 Soluções conhecidas

O site do GitHub possui funcionalidades a partir da url https://github.com/pulls que permitem obter pull requests por diversos filtros, como autor, status, visibilidade, etc., o que auxilia, em certa forma, o acompanhamento de PR de um determinado repositório. No entanto, sempre que o gerente precisa realizar um acompanhamento, o mesmo precisará montar novamente as consultas para busca. Outra problemática evidente é que o usuário não consegue buscar por PR de diferentes autores ou projetos ao mesmo tempo.

Outra forma de aperfeiçoar o trabalho com pull requests é o IGit[2], ferramenta que permite lidar com PR advindos de vários repositórios e com mais de um autor. Não obstante, este sistema foi construído para desenvolvedores que buscam aprimorar seu tempo nas revisões, o que acarreta que o usuário encontrará funcionalidades como: mudanças no código e envio de sugestões. Contudo, para um gerente, estas funções são desnecessárias, pois o mesmo precisa de dados cruciais para o acompanhamento, como: título, data de alteração e estado de uma PR. Além disso, o usuário não tem a possibilidade de unir vários PR em um projeto e acompanhar os dados desses simultaneamente.

1.4 Git Dashboard

O Git Dashboard, sistema desenvolvido neste projeto, busca sanar as dificuldades de gestão de um gerente de projeto, propiciando uma gestão mais sólida e simples a partir um sistema web para o acompanhamento de PRs. Esses PRs podem ser obtidos de qualquer repositório que o usuário tenha acesso no GitHub.

No sistema, um gerente pode se autenticar com uma conta do GitHub e criar vários Projetos, os quais funcionam como uma estrutura organizacional, onde serão vinculados repositórios e PRs do GitHub. Os repositórios vinculados ao Projeto auxiliam em uma busca mais rápida por PRs desses que o usuário deseja vincular e o usuário também pode buscar e vincular PRs de repositórios que ele ainda não acompanha.

O sistema tem o intuito de auxiliar o dia a dia do coordenador, para tanto, o mesmo fornece uma interface simples e amigável que lista os PRs e repositórios vinculados ao projeto, apresentando informações mais gerais como nome, *status*, data de atualização e responsável do PR, como apresentado na Figura 1. Além disso, o coordenador pode ter informações mais completas de cada PR, podendo acessar um PR em particular para obter dados como commits, comentários e revisores.

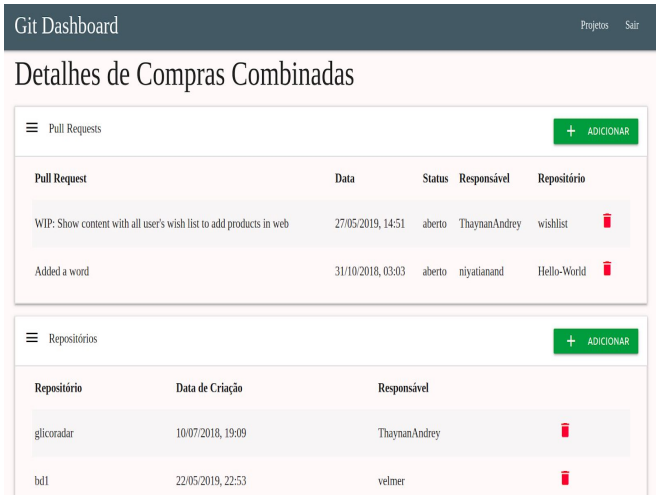

Figura 1: **Tela do Git Dashboard para acompanhamento dos pull requests e repositórios do projeto.**

Ademais, no momento em que o coordenador considerar que o PR ou o repositório não deve ser mais acompanhado, ele pode desvincular os mesmos do Projeto, ou até mesmo, se o Projeto criado não fizer mais jus às suas atividades, ele pode excluí-lo, mantendo apenas as informações necessárias para o seu trabalho.

Nesta perspectiva, o Git Dashboard é projetado para atender as necessidades de uma gerência com qualidade e fluidez, fornecendo um instrumento de acompanhamento simples que auxiliará no desenvolvimento dos Projetos de responsabilidade do coordenador. Vale ressaltar que o mesmo é desenvolvido para uma visão macro de coordenação e não para o desenvolvimento, sendo assim, não serão apresentadas no sistema, por exemplo, modificações de código.

Logo, suscitando-se o problema do coordenador Pedro, o mesmo ao utilizar o Git Dashboard, poderia criar o Projeto “Compras Combinadas” e vincular os repositórios responsáveis pelas compras e entregas de produtos, e a partir disso, vincular os PRs criados para a elaboração dessa nova funcionalidade. Com os repositórios e PRs vinculados, Pedro poderia ver de forma rápida e sucinta, o desenvolvimento das atividades, as últimas atualizações e estimar a entrega da “compra combinada”.

2 Arquitetura e Projeto da Solução

2.1 Visão geral do projeto

[2] https://igit.dev/

O Git Dashboard tem o intuito de propiciar uma gestão de qualidade para coordenadores de projetos que possuam sistemas armazenados no GitHub, permitindo que os usuários adicionem pull requests no sistema e acompanhem a evolução desses de forma rápida e sucinta.

O sistema é projetado para funcionar em plataformas web, permitindo que usuários o acessem de qualquer dispositivo com acesso à internet e com um navegador instalado.

Logo, esse software é estruturado em três componentes principais que se comunicam segundo o modelo REST. O primeiro componente é um cliente responsável pela interação com o usuário, o segundo é um servidor responsável pela autenticação de usuários e o armazenamento de dados, e o último é uma API externa que prover o acesso aos dados do GitHub.

2.2 Casos de Uso

Como já explicitado anteriormente, o sistema tem o objetivo de propiciar o acompanhamento de um projeto através de Pull Requests do GitHub, dispondo de funcionalidades que gerencie a adição e o detalhamento de PR e repositórios. Nesta perspectiva, são explicitadas as principais funcionalidades do Git Dashboard através de um diagrama de casos de uso, como mostra a Figura 2. Dentre estas funcionalidades, temos a autenticação do usuário, e a interação com projetos, repositórios e pull requests.

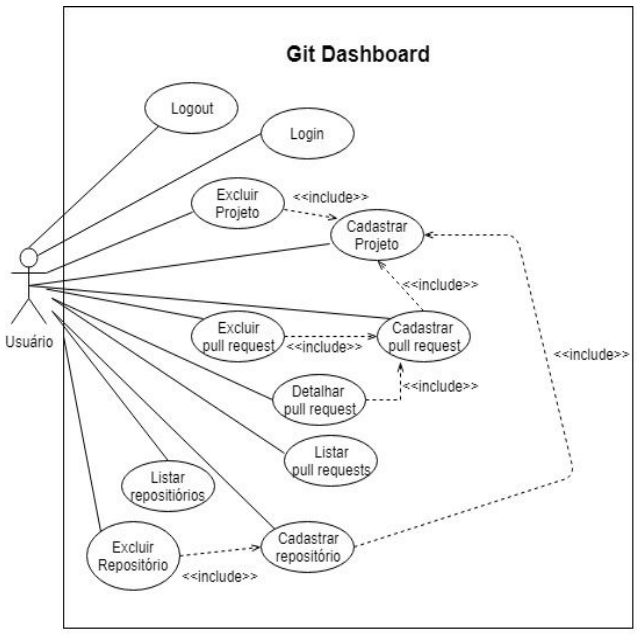

Figura 2: **Diagrama de Casos de Uso de um usuário**

A autenticação do usuário, realizada a partir do login, prover o acesso seguro ao sistema para a obtenção de dados do usuário, salvos no próprio sistema ou no GitHub. Além disso, o usuário pode sair do sistema com a função de logout.

A interação com o projeto provê o acesso aos projetos criados pelo o usuário, a criação de novos projetos, assim como, a exclusão desses.

A interação com os repositórios propicia o acesso aos repositórios vinculados ao projeto e a vinculação de novos repositórios, a qual pode ser realizada pela adição de repositórios do usuário autenticado ou repositórios externos que o usuário tenha acesso, mediante o nome do repositório e o do proprietário desse. Além disso, o usuário pode desvincular repositórios de seus projetos.

A interação com os pull requests proporcionam as principais funcionalidades do sistema, as quais permitem o acesso aos PR vinculados ao projeto e a vinculação de novos PR, como apresentado na Figura 3, onde podem ser adicionados PRs a partir de repositórios vinculados ao projeto ou buscados diretamente com os dados do PR: nome do proprietário do repositório, nome do repositório e número do PR. Ademais, o usuário pode detalhar as informações de um PR - nome, datas de criação e atualização, responsável, descrição, commits, etc. - ou desvinculá-lo do projeto.

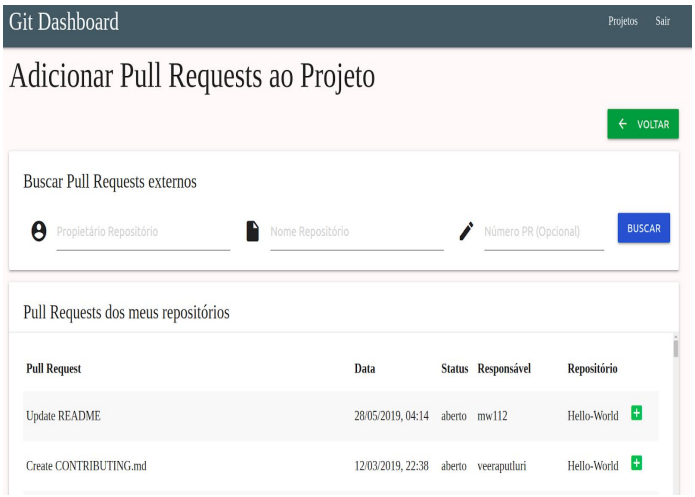

Figura 3: **Tela do Git Dashboard para a vinculação de novos pull requests ao projeto.**

2.3 Arquitetura

O Git Dashboard é projetado de acordo com a arquitetura Cliente-Servidor, na qual existem dois componentes do próprio sistema e um componente externo, a API do GitHub. Os componentes se comunicam segundo o modelo REST (Representational State Transfer), utilizando o JSON (Java Object Notation) como o modelo para a transferência de dados.

Este modelo de comunicação propicia a modularização e a escalabilidade [4]. Sendo assim, os componentes podem evoluir e se modificar constantemente e de maneira independente e, assim, oferecerem apenas seus serviços para os demais através de suas interfaces, as quais podem

ser acessadas por outros componentes e prover a transferência dos dados em formato JSON.

O modelo arquitetural Cliente-Servidor do sistema é organizado em um cliente e dois servidores, permitindo a distribuição de tarefas entre os mesmos, e é esboçado em um diagrama de componentes na Figura 4. O cliente é o responsável pela interação com os usuários do sistema, enquanto que os servidores provém as informações necessárias para o cliente, dos quais o servidor do sistema construído no Firebase possui dados dos projetos e informações identificadoras dos repositórios e PR, enquanto que a API do GitHub prover todas as informações mais correntes dos repositórios e PR vinculados.

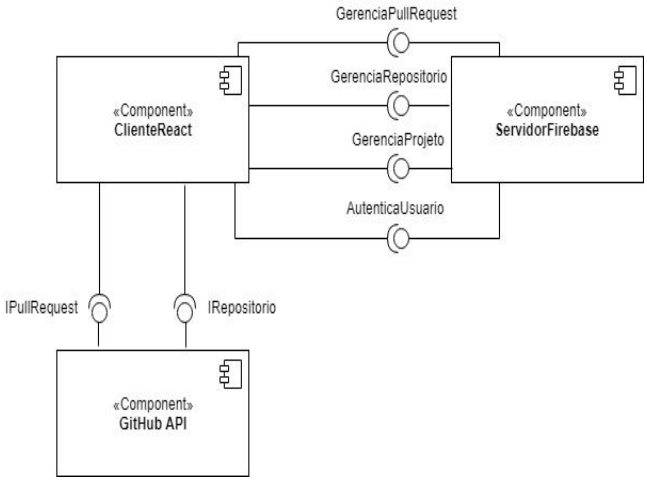

Figura 4: **Diagrama de Componentes do Git Dashboard**

2.3.1 Cliente

O cliente do sistema tem o objetivo de fornecer uma interface para a interação com os usuários que provê serviços de acesso aos projetos criados e aos repositórios e pull requests vinculados a esses projetos. O mesmo foi desenvolvido a partir das bibliotecas de código aberto: React[3] e Redux[4].

O React é uma biblioteca JavaScript de código aberto que permite a criação de interfaces de usuário, ou seja, propicia a construção de uma camada de visualização em um sistema web. A mesma é, até o presente momento, uma das principais tecnologias utilizadas no desenvolvimento de aplicações web, possuindo uma comunidade muito forte e ampla, o que acarreta em constantes evoluções no seu código.

Esse arcabouço segue um modelo arquitetural baseado em componentes, organizando sua estrutura em uma árvore de componentes, os quais, em conjunto, definem a camada de visualização do sistema. Os componentes funcionam como funções puras, possuindo um estado, e recebem propriedades através de propriedades (*props*), as quais poderão ser utilizadas na camada de visualização.

Esta tecnologia foi utilizada no sistema com o intuito de desenvolver a interface do usuário, de maneira a prover um sistema com módulos reusáveis e escaláveis. Logo, buscou-se a construção de componentes altamente coesos, ou seja, componentes que possuam uma função bem específica, podendo serem reutilizados sempre que necessitarem de suas funções. Além disso, essa estrutura bem modularizada propicia a escalabilidade do software, podendo crescer em funcionalidades com muita fluidez, a partir da reutilização ou inserção de novos componentes.

Além do React, utilizou-se o Redux, uma biblioteca JavaScript para gerenciar o estado da aplicação. Essa biblioteca proporciona um novo modelo arquitetural para a aplicação que se baseia no Flux[5], e se distingue do modelo MVVM (Model-View-ViewModel) que era o dominante em arquiteturas de clientes web. O Redux busca complementar a estrutura do React com a camada de lógica de negócios.

Essa biblioteca funciona seguindo três princípios:

1. **Um único ponto da verdade:** o estado da aplicação é mantido em um único objeto chamado Store.
2. **O estado é imutável:** a única maneira de alterar o estado é emitindo uma Action para enviar as mudanças.
3. **Funções puras realizam as mudanças:** as funções puras chamadas de Reducers são as responsáveis por receber as mudanças das Actions e alterar o estado.

Esse modelo permite a manutenção do estado da aplicação (conjunto de dados mantidos na aplicação) chamado de Store, o qual será modificado apenas mediante o disparo de uma Action que notificará o Reducer, e essa modificação se refletirá em toda a aplicação, como é apresentado na Figura 5. Essa estrutura auxilia o React na organização do sistema para as mudanças realizadas nos estados que serão utilizados nos componentes.

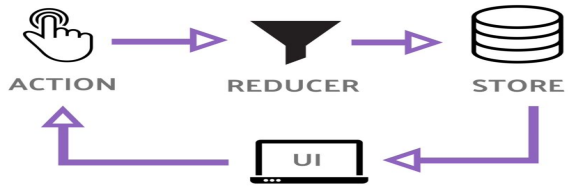

Figura 5: **Ilustração do gerenciamento do estado da aplicação realizado no Redux (Fonte: https://itnext.io/integrating-semantic-ui-modal-with-redux-4df36abb755c)**

Nesta perspectiva, o Git Dashboard realiza toda a comunicação com os servidores e todas as alterações feitas em estados que serão exibidos na visualização, por intermédio de Actions, as quais enviam as mudanças para os Reducers, e esses alteram o Store. Isso propicia a

[3] https://reactjs.org/
[4] https://redux.js.org/
[5] https://facebook.github.io/flux/

organização da lógica de negócios do cliente e a manutenção correta do estado que se refletirá entre todos os componentes.

2.3.2 Servidor

O servidor do sistema foi desenvolvido com o intuito de fornecer serviços de armazenamento em banco de dados e autenticação para o componente cliente. O mesmo foi construído no Firebase[6].

O Firebase é uma plataforma mantida atualmente pelo Google e fornece diversas ferramentas e serviços como: Realtime Database, Hosting, Cloud Firestore, Cloud Functions, Authentication, Google Analytics, etc. Dentre esses serviços, o servidor do sistema utilizou o Cloud Firestore para o armazenamento em banco de dados e o Authentication para realizar a autenticação do usuário.

Essa plataforma fornece uma interface de serviços para sistemas web, permitindo a comunicação segundo o modelo REST, o qual é usado no Git Dashboard. Ademais, o Firebase propicia o desenvolvimento de serviços com velocidade, praticidade e escalabilidade, em uma interface simples e amigável para o usuário. Fatores esses foram determinantes para escolhê-lo como o ambiente para a construção do componente servidor.

2.3.2.1 Autenticação

O sistema lida com repositórios e pull requests de usuários do GitHub, os quais podem ser públicos ou privados. Logo, para realizar o acesso aos recursos privados, a API do GitHub requer que o usuário possua um token de acesso, o que caracteriza que o mesmo esteja autenticado na plataforma.

Sendo assim, o Git Dashboard necessita que o usuário esteja autenticado no GitHub, porque a partir disso obtém-se um token de acesso para utilizar os recursos dessa plataforma, além de propiciar a verificação da existência de uma conta do usuário no GitHub.

Para realizar esta autenticação, o sistema utiliza o serviço Firebase Authentication, o qual permite o cadastro e a autenticação de usuários por meio de email e senha, assim como, possibilita a autenticação por meio do Social Auth, o qual possibilita que o usuário logue no sistema, sem a necessidade de um cadastro prévio, usando sua conta do Facebook, Instagram, GitHub, dentre outras mídias sociais.

Logo, utilizando do recurso do Social Auth, o Git Dashboard realiza a autenticação através do autenticador do GitHub disponibilizado no Authentication, e assim, o sistema utiliza o token de acesso do GitHub obtido na autenticação para validar o acesso ao Git Dashboard, assim como para obter os recursos da API do GitHub.

2.3.2.2 Banco de Dados

Um usuário no sistema pode criar projetos e vincular repositórios e pull requests aos mesmos, para que assim possa acompanhá-los no seu dia a dia. Sendo assim, o usuário lida com modelos de objetos que configuram três entidades do sistema: Project, Repository e Pull Request. As entidades Repository e Pull Request são representações das estruturas dos repositórios e pull requests do GitHub, enquanto que o Project é uma estrutura na qual estão vinculados os repositórios e pull requests. Além disso, o usuário por si só também é uma entidade para o sistema e é representado por User.

Nesta perspectiva, o Git Dashboard apresenta os dados dos repositórios e pull requests do GitHub para que o usuário possa realizar os seus acompanhamentos, no entanto, esses dados estão em constantes modificações dentro do GitHub. Sendo assim, como forma de evitar inconsistência desses dados no acompanhamento, as entidades do banco de dados guardam informações apenas de referências aos repositórios e pull requests, como o ID (identificador), número de pull request e nome dos proprietários dos pull requests e repositórios, para que a partir desses, obtenha-se os dados mais atuais de cada entidade, fazendo-se uma busca a API do GitHub.

Na Figura 6 é apresentado o Diagrama Entidade Relacionamento que modela as entidades do banco de dados e como elas se relacionam. Logo, observa-se que a entidade User, que modela os usuários, possui as propriedades UID e email, além de possuir zero ou mais projetos. O Project modela os projetos do sistema, os quais devem possuir as propriedades name e creationDate, além de conter zero ou mais repositórios e pull requests. O Repository modela os repositórios do sistema, os quais possuem as propriedades name, idGitHub e ownerName, as quais serão utilizadas na busca na API do GitHub. Por fim, existe o Pull Request que modela os pull requests do sistema, os quais devem conter as propriedades idGitHub, ownerName, idRepositoryGithub, repositoryNameGitHub e pullRequestNumber, e essas propriedades serão também utilizadas para as consultas à API do GitHub.

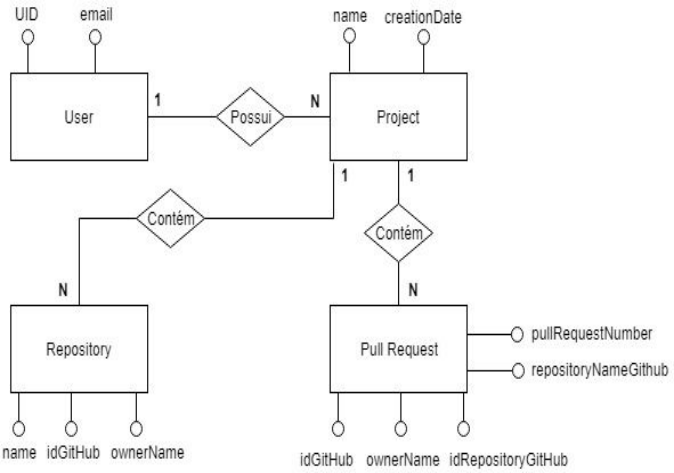

Figura 6: **Diagrama Entidade Relacionamento do modelo de dados do Git Dashboard**

[6] https://firebase.google.com/

O banco de dados do sistema foi criado através do serviço Cloud Firestore[7] fornecido pelo Firebase. Esse serviço propicia o armazenamento em um modelo de banco de dados NoSQL orientado a documentos e fornece o acesso aos dados a partir de uma camada REST, a qual também permite a utilização de consultas com cláusulas WHERE.

Os bancos de dados NoSQL fornecem mecanismos de armazenamento e recuperação de dados, e são característicos pela sua modelagem e o seu relacionamento de dados se diferir das relações tabulares utilizadas por bancos de dados relacionais. Esse modelo busca simplificar a representações de dados no projeto, além de ser escalável e proporcionar o escalonamento horizontal [5].

O modelo NoSQL orientado a documentos é utilizado pelo Cloud Firestore e organiza a estrutura em coleções e documentos. Os documentos representam entidades do sistema, como por exemplo, um repositório, e são estruturados conforme um objeto javascript, seguindo-se o padrão chave-valor. Enquanto que a coleção representa o conjunto de documentos, como por exemplo, o conjunto de todos os repositórios que está na coleção “repositories”.

Nesta perspectiva, tomando-se conhecimento da modelagem do NoSQL utilizada pelo Cloud Firestore, o banco de dados do Git Dashboard foi desenvolvido, no qual existem as coleções users, projects, repositories e pullRequests para reunir as entidades dos sistemas que são representados por documentos. E esses dados podem ser acessados ou inseridos a partir da interface REST fornecida pelo Firebase.

2.3.3 GitHub API

A API do GitHub fornece serviços para acessar e manipular recursos do próprio sistema, como por exemplo, acessar pull requests de um determinado repositório, criar novos repositórios e editar pull requests existentes. Para realizar estas ações, o sistema deve se comunicar com a interface fornecida pela API, e para essa interface, a API dispõe de algumas versões, das quais as versões 3 e 4 são as mais utilizadas.

A terceira versão[8] da API do GitHub propicia o acesso aos recursos a partir de uma camada REST, enquanto que a quarta versão permite que o acesso ocorra segundo o modelo GraphQL[9]. Logo, como o Git Dashboard mantém o padrão de comunicação REST entre os seus componentes, a versão 3 da API foi a utilizada no sistema.

Nesta perspectiva, como apresentado no modelo arquitetural do sistema, a API do GitHub é tratada como um componente externo, e se caracteriza como um pilar para o sistema, pois a mesma fornece os dados necessários para realizar as atividades do sistema. Logo, a API é acessada pelo componente cliente para se obter todos os dados correntes dos repositórios e pull requests, e assim propiciar aos usuários, o acompanhamento dos projetos.

Não obstante, apesar de fornecerem diversos recursos para manipulação dos pull requests e repositórios, como edições e exclusões dos mesmos, o sistema lida, até então, apenas com o acesso desses, fornecendo o estado atual dessas entidades.

2.3.4 Outras tecnologias utilizadas

Dentre outras tecnologias utilizadas neste projeto, a biblioteca Materialize CSS[10] foi uma que pode tomar um pouco de destaque, pois a mesma foi utilizada para a estilização das páginas do Git Dashboard, seguindo os padrões do Material Design[11], o qual foi desenvolvido por designers do Google, e fomenta um padrão de design para a interface de um sistema, com a proposta de proporcionar ao usuário, recursos que sejam fluidos e intuitivos. Nesta perspectiva, essa biblioteca, que está disponível para o React, foi utilizada para seguir os padrões e propiciar uma melhor interface para o usuário do sistema.

3 Sistema em Uso

Produtos de software que agradam o usuário são aqueles que deixam o mesmo satisfeito durante e após a sessão de uso do sistema, sendo assim, softwares que frustram o usuário só serão utilizados quando não existirem outras opções, como a falta de disponibilidade de sistemas concorrentes. Logo, o desenvolvimento de produtos baseados na satisfação do usuário propiciam a aceitação desses sistemas no mercado, além de proporcionarem a qualidade do trabalho, pois pessoas satisfeitas tendem a ter mais eficiência e melhor concentração durante sua rotina.

Nesta perspectiva, existem várias estratégias para mensurar o grau de satisfação de um usuário ao utilizar um sistema computacional, dentre as quais tornou-se muito aderido o uso de questionários. Os mesmos funcionam como testes que propiciam o delineamento de um perfil de satisfação do usuário, do qual pode-se obter valores quantitativos e qualitativos, e assim permitem avaliar o que está sendo estudando, que para o caso em questão é a satisfação do usuário.

Não obstante, avaliar a satisfação do usuário por si só, é algo muito amplo, complexo e subjetivo, e pode ser de difícil obtenção dentro de um questionário. Logo, definem-se focos para avaliar a satisfação do usuário, dentre os quais existe a mensuração da usabilidade do sistema, o qual pode ser considerado como um fator importante para inferir que o usuário está satisfeito em usá-lo [6].

A partir dessa mensuração de usabilidade, foram fomentados questionários que buscavam avaliar a usabilidade de um sistema e, assim, inferir o grau de satisfação do usuário ao utilizá-lo. Dentre esses questionários, pode-se citar: QUIS (Questionnaire for User

[7] https://firebase.google.com/docs/firestore
[8] https://developer.github.com/v3
[9] https://graphql.org/
[10] https://materializecss.com/
[11] https://material.io/

Interaction Satisfaction), ASQ (After-Scenario Questionnaire), PSSUQ (Post-Study System Usability Questionnaire) e CSUQ (Computer System Usability Questionnaire). Sabendo da qualidade proporcionada pelos mesmos para avaliação da usabilidade de um sistema, foi-se escolhido para avaliar o Git Dashboard, o PSSUQ, o qual foi desenvolvido por pesquisadores da IBM e se caracteriza por fornecer uma forma de avaliação global da usabilidade do sistema.

O PSSUQ é formado por 19 itens destinados a abordarem as seguintes características de usabilidade do sistema: conclusão rápida do trabalho, facilidade de aprendizado, documentação de alta qualidade, adequação funcional e rápida aquisição do sistema. Além disso, a resposta para cada item está associada a uma escala de 1 a 7, na qual o avaliador colocará o quanto concorda com o item, onde 1 significa concordar completamente e 7 discordar completamente [7].

3.1 Metodologia

Para realizar a avaliação do Git Dashboard, por meio do PSSUQ, foram submetidos a responderem ao questionário, 7 alunos de graduação do curso de Ciência da Computação da Universidade Federal de Campina Grande, os quais estão próximos de concluírem o curso, ou seja, faltam 3 períodos ou menos para se formarem, além disso os mesmo têm 1 ano ou mais de experiência com o desenvolvimento no GitHub, utilizando PRs para executarem suas atividades diárias.

Esses alunos, separadamente, receberam durante um tempo médio de 10 minutos, informações gerais sobre o sistema, como: os objetivos e os problemas que o sistema busca enfrentar. Posteriormente, foi apresentada aos mesmos, a maneira de acessar o sistema a partir de um navegador web, não sendo necessária a configuração prévia de um ambiente na máquina, além disso, o aluno ficou livre para escolher o navegador que se sentisse mais confortável, os quais em suma tomaram preferência por usar o Google Chrome ou o Mozilla Firefox.

Todos seguiram um mesmo roteiro[12] para execução das tarefas, as quais englobavam todas as funcionalidades do sistema, navegando entre todas as páginas do mesmo, o que durou um tempo médio de 20 minutos para executá-las. Logo, para tal execução, os alunos, primeiramente, tiveram de criar um projeto no sistema para representar uma atividade que o gerente gostaria de acompanhar. Depois disso, vincularam novos repositórios a esse projeto, os quais deveriam ser tanto repositórios que eles contribuíam, quanto repositórios públicos que eles não tinham vínculo nem contribuíam, e assim testaram todas as possibilidades de vincularem repositórios que o sistema permite. Posteriormente, os mesmos tiveram de inserir PRs no projeto, os quais deveriam possuir comentários, commits, descrições e advirem dos repositórios que eles vincularam ao projeto, assim como, de repositórios externos, os quais eles contribuíam ou eram públicos, e assim englobaram todas as possíveis maneiras de vincularem um PR. Por fim, os alunos deveriam observar todas as informações dos PRs vinculados, como título, descrição, data de atualização, commits, etc., e assim concluírem sua sessão de testes do sistema.

Finalmente, após a execução do roteiro, os alunos responderam ao questionário[13] PSSUQ disponibilizado em um formulário online no Google Forms, avaliando sua experiência em usar o sistema, o que durou um tempo médio de 5 minutos para concluírem-no.

3.2 Resultados

Para calcular o nível de satisfação dos usuários em utilizar o sistema, foi-se necessário sumarizar as respostas dos alunos de forma a representarem um resultado geral. Logo, resumiu-se as repostas a partir do cálculo das médias das respostas em cada item do questionário, ou seja, para cada item do questionário, obteve-se a resposta de cada aluno relacionada ao mesmo e calculou-se esta média. O resultado final desta estatística sumarizante é apresentado na Figura 7.

Figura 7: **Gráfico de barras com a resposta média dos usuários para o questionário PSSUQ**

Neste tocante, avaliando-se a satisfação dos usuários, observou-se a média das respostas para cada item, as quais variaram no intervalo de 1,43 a 2,71, deduzindo-se, de acordo com a escala proposta, que os usuários concordam com os itens. Sendo assim, concluiu-se que os mesmos estavam satisfeitos em utilizar o sistema, concordando com os critérios apontados no questionário, e assim, inferiu-se que para esta versão inicial do Git Dashboard, o sistema é de fato utilizável e pode propiciar qualidade no dia a dia de gerentes de projeto.

[12] Roteiro disponível em: https://github.com/ThaynanAndrey/tcc-git-dashboard/blob/master/anexos/Roteiro%20de%20execu%C3%A7%C3%A3o%20para%20avalia%C3%A7%C3%A3o%20do%20PSSUQ.pdf

[13] Questionário PSSUQ traduzido para português, disponível em: https://github.com/ThaynanAndrey/tcc-git-dashboard/blob/master/anexos/Post-Study%20System%20Usability%20Questionnaire%20(PSSUQ).pdf

Pode-se destacar os itens 3, 14 e 15 como os mais positivos, os quais apresentaram uma resposta média de 1,43. Dentre esses, o item 3 corresponde a avaliar a capacidade de completar as tarefas desse sistema de forma efetiva, enquanto que os itens 14 e 15 contemplam a apresentação e organização das informações no sistema, avaliando-se a clareza na apresentação das mesmas e se essas foram eficazes para auxiliarem o usuário a completar suas tarefas. Logo, suscita-se que o sistema se apresentou bem para a execução das tarefas disponíveis pelo mesmo, nas quais os usuários conseguiram completar essas tarefas de forma eficiente, e suas dúvidas ao decorrer da realização das mesmas foram sanadas com informações claras sobre a maneira de prosseguir com a execução dela.

Além disso, destacam-se os itens 9, 16, 17 e 18 como os que receberam as piores avaliações, dos quais o item 16 obteve uma média de respostas de 2,71, e os demais itens obtiveram 2,57 como respostas médias. Dentre esses itens, o item 9 contempla a exibição clara de mensagens de erros e como o usuário deve agir para corrigir o problema, os itens 16 e 17 avaliam a agradabilidade da interface do sistema e a facilidade no uso, por fim o item 18 avalia se o sistema possui todas as funções que são esperadas do mesmo. Nesta perspectiva, a avaliação desses itens propiciaram a análise de que o sistema de fato precisa melhorar sua interação com o usuário, na qual tem de se aperfeiçoar a organização de sua interface para tornar a atividade do usuário mais eficaz e agradável, além de propiciar mensagens de erros mais claras que ajudem os usuários a entender os problemas ocorridos e como podem ser solucionados. Além disso, pode-se inferir que é necessário aprimorar o conjunto de funcionalidades disponíveis pelo sistema para auxiliar a atividade do gerente de projetos.

Por fim, os avaliadores fomentaram sugestões de melhorias no projeto, as quais vão de encontro aos itens com piores avaliações, e propõem no geral, mudanças nas interfaces do Git Dashboard para deixá-lo mais claro, a inserção de novas funcionalidades como filtragens e ordenações nas tabelas apresentadas, e a melhoria na exibição dos erros ocorridos e como proceder quando ocorrer um desses.

4 Experiências e lições aprendidas

O desenvolvimento do Git Dashboard uniu várias características de um engenheiro de software para propiciar a construção do sistema. Sendo assim, foram necessárias habilidades para elencar os requisitos do software, definir o ambiente onde o mesmo seria executado, projetar o arcabouço do sistema para a execução em uma plataforma web, planejar o processo de desenvolvimento segundo alguns princípios da metodologia Scrum, definir as entregas das funcionalidades, testando-as e validando-as, e assim, a partir dessa organização, consolidar a entrega de um software com qualidade e manutenibilidade.

Além disso, foi-se necessário saber lidar com tecnologias externas como a API do GitHub e o Materialize CSS, manuseando as funcionalidades fornecidas por essas para a construção da aplicação. Não obstante, foi-se essencial desenvolver novos conhecimentos para tratar as limitações dessas tecnologias para as necessidades do sistema.

Nesta perspectiva, produzir uma aplicação web que se encaixasse com as necessidades de um gerente de projetos se apresentou bastante desafiador, pois buscou-se fornecer mecanismos que melhorassem o seu trabalho diário e auxiliassem no desenvolvimento de outros sistema com velocidade, qualidade e robustez. Ademais, enfrentar os limites das tecnologias externas sem comprometer o desenvolvimento do sistema fomentou a aquisição de novas habilidades para um engenheiro de software.

4.1 Processo de desenvolvimento

O processo de desenvolvimento adotado seguiu a ideia do **Scrum**, uma forma de organização do processo que se adequa segundo as metodologias ágeis. Essa metodologia fomenta a entrega rápida de pequenas funcionalidades requeridas no sistema, e a evolução constante no desenvolvimento do software.

Para entender melhor, no modelo Scrum, o ciclo é definido como **Sprint**, e pode ser entendido como um intervalo de tempo bem definido, que pode ser semanal ou mensal, onde será implementada uma série de atividades planejadas, as quais estão listadas no **Sprint Backlog**. As atividades do Sprint Backlog são retiradas de uma lista previamente planejada, chamada de **Product Backlog**, o qual contém todas as funcionalidades requeridas para o sistema, e a priorização para a execução das mesmas. Para se definir a priorização das atividades e quais serão realizadas em cada Sprint, existe o **Product Owner (PO)**. Logo, sendo definido todas essas partes, são organizados todos os ciclos de desenvolvimento até ser concluído o projeto [8].

Sendo assim, para o desenvolvimento do Git Dashboard, o PO foi representado por Matheus Gaudencio, o qual em reuniões comigo, definimos inicialmente o Product Backlog do sistema para representar todos as funcionalidades do sistema e qual era a ordem de prioridade para a implementação dessas. Posteriormente, o desenvolvimento começou em Fevereiro de 2019, onde foi definido antes de cada Sprint, a qual tinha duração de 15 dias, o Sprint Backlog para estabelecer as atividades a serem executadas durante esse período. Por fim, ao finalizar cada ciclo, as entregas dessas novas funcionalidades planejadas, como por exemplo, a busca por repositórios do usuário, eram testadas e avaliadas pelo PO, recebendo-se um *feedback* da implementação da mesma e de como estava o andamento geral do sistema, além de avaliar as perspectivas futuras para finalização do sistema. As sprints continuaram até a finalização do projeto com a implementação de todas as funcionalidades esperadas, o que ocorreu em Maio de 2019.

Este modelo de processo auxiliou a entrega pontual do sistema, de forma organizada e consensual. Ademais, com esse metodologia mais flexível de entregas, pôde-se adicionar, alterar ou excluir requisitos durante o

desenvolvimento para seguir em conformidade com as necessidades do gerente de projetos.

Por fim, pode-se destacar que o cliente por estar sempre em contato com o sistema, produziu *feedbacks* mais precisos de suas necessidades, fomentando a conclusão de uma aplicação robusta.

4.2 Principais desafios e limitações

Lidar com API externas, por mais consolidadas que essas sejam, requer desafios, pois nem sempre encontram-se as funcionalidades adequadas para serem utilizadas. No contexto do Git Dashboard, o sistema lida com entidades do GitHub, os repositórios e os pull requests, dos quais são apresentadas suas informações constantemente no sistema.

No entanto, para que o sistema possa apresentar as informações de um pull request ou de um repositório vinculado a um determinado projeto, ele tem de armazenar as mesmas em um banco de dados para acessá-las quando necessário. Todavia, a descrição de um PR, por exemplo, está sujeita a mudanças constantes no GitHub, e salvá-la no banco de dados do Git Dashboard poderia levar à inconsistência da informação com o que está salvo no GitHub.

Sendo assim, como forma de lidar com a problemática, foi desenvolvida a estratégia de armazenar no banco de dados, apenas as identificações do PR e do repositório no GitHub, ou seja, o ID dos mesmos, para a partir desses, acessar essas entidades diretamente da API do GitHub e assim manter a consistência de todas as informações das mesmas.

Além disso, apresentar Pull Requests de um determinado repositório que ainda não foram vinculados ao projeto foi outro desafio, pois uma requisição para obter os PR de um determinado repositório a partir da API retorna uma resposta paginada, ou seja, são enviados 30 PR por vez. Logo, sabendo-se que podem existir PR desse repositório que já foram vinculados ao projeto, é necessário filtrar a resposta para remover os que já estão vinculados ao projeto, e assim apresentar apenas os que são propícios a serem adicionados. Não obstante, esse número de PR vinculados pode variar bastante de um projeto para o outro, sendo assim, como forma de lidar com isso, foram realizadas requisições em paralelos por todas as páginas de PR do repositório, concatenando-se todas as respostas como uma única, e assim filtrando-se todos os PR já vinculados, para enfim exibir a resposta correta ao usuário.

Observe que para ambas as soluções apontadas anteriormente existem limitações, pois essas resoluções requerem maior custo de memória e processamento dos dados. Para o escopo do desenvolvimento desse projeto, o qual vislumbra apenas a produção de um MVP, concluiu-se que essas limitações seriam aceitáveis. Contudo, para o desenvolvimento do projeto para uma escala de produção, projeta-se usar recursos de *cash* para manter os dados mais atuais e diminuir as requisições, além de estudar outras maneira de lidar com a paginação, para assim tentar superar esses desafios.

4.3 Trabalhos futuros

Este artigo apresenta um sistema que auxilia um gerente de projetos a acompanhar o desenvolvimento de sistemas de software a partir de uma plataforma web, na qual o gerente poderá realizar a supervisão do desenvolvimento a partir da análise dos PR e repositórios que estão vinculados aos projetos criados pelo mesmo.

Todavia, o Git Dashboard é até então um MVP (Minimum Viable Product), ou seja, um produto produzido para se avaliar a sua importância para a gestão do desenvolvimento de projetos de software. A partir disso, visa-se de fato desenvolver o sistema para uma escala de produção, inserindo-se novas funcionalidades no sistema que correspondam às necessidades do usuário, além de melhorar o desempenho e performance desse, o que já foi suscitado, na Seção 4.2, como limitações apresentadas até o presente momento no software.

Ademais, a partir dos testes de usabilidade do sistema apresentados na Seção 3 do artigo, foram cogitadas melhorias a serem inseridas no sistema, como mudanças na interface do sistema, para torná-la mais sucinta, agradável e sem dualidade de informações, além de organização da apresentação de erros realizados pelo usuário. Nesta perspectiva, as propostas de melhorias serão analisadas e projetadas para serem desenvolvidas futuramente, e assim, proporcionar mais qualidade no uso do Git Dashboard.

REFERÊNCIAS

[1] GitHub - Maintaining a Project, https://git-scm.com/book/en/v2/GitHub-Maintaining-a-Project.

[2] Creating Pull Requests, https://www.kernel.org/doc/html/v4.17/maintainer/pull-requests.html.

[3] G.P.C Poo-Caamaño, Release Management in Free and Open Source Software Ecosystems, https://dspace.library.uvic.ca/bitstream/handle/1828/7648/Poo-Caamano_German_PhD_2016.pdf?sequence=1.

[4] Sulyman, Shakirat. (2014). Client-Server Model. IOSR Journal of Computer Engineering. 16. 57-71. 10.9790/0661-16195771.

[5] Dave, Meenu. (2012). SQL and NoSQL Databases. International Journal of Advanced Research in Computer Science and Software Engineering.

[6] JERQ Queiroz, Sondagem da Satisfação Subjetiva e Mensuração do desempenho do Usuário, http://www.dsc.ufcg.edu.br/~rangel/ihm/downloads/Capitulo6.pdf.

[7] JRL Lewis (1993). IBM Computer Usability Satisfaction Questionnaires: Psychometric Evaluation and Instructions for Use. International Journal of Human-Computer Interaction.

[8] HLNA Almeida. Gestão e planejamento de projetos de software com scrum. Revista Científica Multidisciplinar Núcleo do Conhecimento. Ano 04, Ed. 03, Vol. 09, pp. 114-123. Março de 2019.