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

JOÃO FELIPE DA SILVA FREITAS

CONNECTBUG:
SISTEMA PARA GERAR E GERENCIAR BUG REPORT DE APLICATIVOS MOBILES

CAMPINA GRANDE - PB

2023

JOÃO FELIPE DA SILVA FREITAS

CONNECTBUG:
SISTEMA PARA GERAR E GERENCIAR BUG REPORT DE APLICATIVOS MOBILES

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. Franklin de Souza Ramalho

CAMPINA GRANDE - PB

2023

JOÃO FELIPE DA SILVA FREITAS

CONNECTBUG:
SISTEMA PARA GERAR E GERENCIAR BUG REPORT DE APLICATIVOS MOBILES

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. Franklin de Souza Ramalho

Orientador – UASC/CEEI/UFCG

Professor Dr. Rohit Gheyi

Examinador – UASC/CEEI/UFCG

Professor Tiago Lima Massoni

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 14 de Fevereiro de 2023.

CAMPINA GRANDE - PB

RESUMO (ABSTRACT)

On a daily basis, 3739 new applications are released on the Play Store, and with them, countless bugs, defined as an error or failure in software or a system, that causes unexpected or incorrect results, or behaves unintentionally. To control these bugs, bug reports were created, which are reports made by users or professionals hired to test an application. These reports help developers maintain the system and organize the priority order and how each bug will be tackled.

These reports are often created based on feedback and evaluations that point out a flaw or problem. This feedback is read and interpreted by more technical people, and then they become bug reports, which generates an extra layer between the final user and the developer. In this work, we propose an integration tool designed to connect the application with a bug tracking tool, bringing the relationship between the customer and the developer closer. The goal is to offer autonomy for the end user of the application to create a bug report, already filling in the most common data that does not require technical knowledge, and then generate a report, making it easier for users of an application to create bug reports and consequently investigate and correct them by responsible developers.

ConnectBug: sistema para gerar e gerenciar bug report de aplicativos mobiles

João Felipe da Silva Freitas
joao.freitas@ccc.ufcg.edu.br
Universidade Federal de Campina Grande
Campina Grande, Paraíba

Franklin Ramalho
franklin@computacao.ufcg.edu.br
Universidade Federal de Campina Grande
Campina Grande, Paraíba

Tiago Massoni
massoni@computacao.ufcg.edu.br
Universidade Federal de Campina Grande
Campina Grande, Paraíba

RESUMO

Por dia, 3739 novos aplicativos são lançados na playstore e com eles inúmeros bugs, definidos como um erro ou falha em um software ou sistema, que causa um resultado inesperado ou incorreto, ou se comporta de maneira não intencional. Para controlar esses bugs surgiram os bugs reports, relatórios feitos por usuários ou profissionais contratados para testar uma aplicação. Esses relatórios ajudam tanto aos desenvolvedores a dar manutenção no sistema como a organizar a ordem de prioridade e como cada bug será atacado. Esses relatórios são muitas vezes criados com base em feedbacks e avaliações, que apontam alguma falha ou problema. Esses feedbacks são lidos e interpretados por pessoas de cunho mais técnico, para aí sim se tornarem bug reports, o que gera uma camada a mais entre o cliente final e o desenvolvedor. Neste trabalho, propomos uma ferramenta de integração pensada em conectar o aplicativo com uma ferramenta de bug tracking, aproximando mais a relação entre o cliente e o desenvolvedor. O objetivo é oferecer autonomia para o usuário final do aplicativo criar um bug report, já preenchendo os dados mais comuns e que não demandam um conhecimento técnico, e após isso já gerar um relatório, dando facilidade ao fluxo de criação de relatórios de bugs por usuários de uma aplicação e consequentemente a sua investigação e correção pelos desenvolvedores responsáveis.

Palavras-chave: bugs, bug tracking, react-native, aplicativo, expo, ferramenta para react-native, open-source bug tracking, bug report react-native, bug report android

Mobile: https://github.com/ja1felipe/mobile-connectbug
Frontend: https://github.com/ja1felipe/frontend-connectbug
Backend: https://github.com/ja1felipe/backend-connectbug

ACM Reference Format:
João Felipe da Silva Freitas, Franklin Ramalho, and Tiago Massoni. 2023. ConnectBug: sistema para gerar e gerenciar bug report de aplicativos mobiles. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUÇÃO

O número de aplicativos no mercado está em constante crescimento, no último quarto de 2022 estão disponíveis 2.87 milhões de apps na Google Play, a maior plataforma do segmento, seguida da Apple Store com 1.96 milhões [4]. Uma das maneiras de um aplicativo se tornar relevante e alcançar sucesso em alguma dessas plataformas é sua avaliação geral, dada pelos usuários. Ser acessível, ter um bom design e ser constantemente atualizado e melhorado são pontos cruciais que refletem nessas avaliações, que acabam ditando como um app vai se sair no mercado. Avaliações podem trazer a fama e o sucesso, mas também podem fazer com que um grande investimento seja perdido, quando uma grande parte do público o avalia de forma negativa. Às vezes essas massa de avaliações negativas podem vir a ocorrer por conta de cenários que poderiam ser evitados, como por exemplo um mau update do app [10], que pode fazê-lo parar de executar ou trazer erros funcionais, que são umas das reclamações mais comuns registradas em avaliações [14].

Em alguns casos, os feedbacks relatados na verdade são bugs na aplicação, que podem demorar a serem resolvidos, por terem que passar por fluxos demorados de investigação. Um dos fluxos mais comuns é que, primeiramente, um profissional acompanhe as avaliações procurando nos feedbacks dos usuários potenciais erros que precisam ser corrigidos. Em seguida, inicia-se o rastreamento de bug, no qual outro profissional, geralmente um engenheiro de qualidade de software, irá tentar reproduzir o erro que o usuário citou, e após isso, preencher um relatório detalhado com informações claras sobre o que aconteceu, o que deveria ter acontecido, e todo o apanhado de etapas que culminou naquela situação [5]. Por fim, esse relatório é enviado diretamente para o time ou através de uma ferramenta de gerenciamento, onde a equipe poderá visualizar e investigar os bugs relatados na aplicação.

Em algumas situações, tanto o feedback negativo por parte do usuário, quanto todo o trabalho para criar um bug report poderiam ser evitados, por um meio que o usuário pudesse avisar de forma mais direta e facilitada ao time responsável. Com isso, prevenindo que a avaliação geral do app venha a cair, e dando ao time de desenvolvimento um feedback mais rápido de algum erro que pode estar acontecendo no aplicativo, aproximando a relação entre usuário e desenvolvedor, o que pode ter um efeito positivo em futuras avaliações de acordo com os estudos [19] e [16].

Nesse contexto, este trabalho visa a criação de uma ferramenta a fim de ajudar o usuário a preencher um formulário de bug report genérico podendo abranger problemas simples como um erro de gramática em algum texto do app ou algo mais alarmante como a falha de um funcionamento chave do aplicativo, levando em consideração

os campos mais imprescindíveis para um bug report satisfatório descritos no trabalho [5], diretamente no app.

A intenção é que o próprio usuário consiga preencher campos importantes de um bug report inicial, como o título do problema, uma descrição, como chegou no erro e provas visuais, como screenshots e vídeos. Outros campos muitas vezes necessários como o ambiente em que o bug foi relatado, a própria ferramenta ficaria encarregada de obter via a API do aparelho no qual o usuário está reportando o bug.

Com isso, a ferramenta ameniza o trabalho que o time técnico teria que fazer, a investigação dos feedbacks, e o preenchimento inicial do bug report, que será feito pelo usuário da aplicação, restando apenas filtrar bugs reports que interpretam como válidos e completar report com informações mais técnicas quando necessário, ganhando assim tempo de produtividade.

A ferramenta propõe recompensas aos usuários que delatarem um bug relevante, trazendo uma maneira de mostrar ao usuário que seu relato foi importante para o melhoramento do aplicativo e incentivá-lo a continuar reportando falhas e erros no futuro.

Todo o gerenciamento de bugs reportados, recompensas e desenvolvedores que irão agir sobre os relatos de bug serão feitos a partir de uma dashboard administrativa, dando assim total controle aos desenvolvedores de poder analisar os relatos e decidir se são realmente válidos ou não.

A ferramenta será voltada para o uso em aplicativos mobile construídos com a framework React-Native [15], um framework utilizado para criação de aplicativos Android e IOS. Este foi escolhido por ser um dos frameworks mais utilizados nos anos de 2019 a 2021 de acordo com a pesquisa realizada por Lionel Sujay Vailshery [27]. E o gerenciamento será feito a parte em um aplicativo web, sendo assim de fácil acesso aos desenvolvedores que iram analisar os bug reports.

2 TRABALHOS RELACIONADOS

Trabalho recente propõe uma ferramenta chamada WebBug, utilizada para gestão de erros em projetos de software, essa que é responsável por gerenciar defeitos de forma gratuita, auxiliando gestores de projetos na Gerência de Configuração de Software [1]. De forma parecida à WebBug, existem outras ferramentas disponíveis na internet, como a solução proprietária Instabug [11]. Elas servem tanto como meio de reportar bugs, como bug trackings onde se é possível gerenciar os bugs reportados.

Também existe a ferramenta paga Shakebug [22], onde sua principal função é apenas a criação de relatórios de bugs e não o gerenciamento deles. Também já existem ferramentas de bug tracking no mercado, como é o caso da BugZilla [3], ela serve como um gerenciador de relatórios de bugs, onde se pode acompanhar o trajeto de relatórios, desde a sua criação até a resolução do bug relatado no mesmo.

Similarmente das ferramentas WebBug [1] e Instabug [11], a ferramenta proposta neste trabalho funciona como um gerenciador de bugs, uma forma dos próprios usuários finais criarem relatórios de bugs diretamente no aplicativo, onde os desenvolvedores poderão gerenciar o relatório e trabalhar a fim de corrigi-lo. Nesse contexto inicial o ConnectBug é voltado para pequenos e médios projetos, por ser um projeto ainda inicial, onde a equipe responsável não terá que arcar com novos custos no seu projeto, e poderá ter um sistema completo totalmente gratuito e open source.

O ConnectBug se destaca por ter como diferencial as recompensas aos usuários que relatarem bugs que são considerados relevantes para o time de manutenção, com isso incentivando aos usuários continuarem a reportar bugs ou possíveis melhorias através da ferramenta ao invés de escrevem avaliações negativas sobre o aplicativo para delatar possíveis erros.

3 FUNDAMENTAÇÃO TEÓRICA

Para este trabalho foi importante entender a relação entre bugs achados em um aplicativo e como isso influência em avaliações negativas dadas por usuários. Com base no estudo [19] é perceptível que são duas coisas ligadas intrinsecamente, o que motiva ainda mais achar uma solução que além de facilitar a resolução de bugs, ajude a evitar que clientes deem avaliações negativas. Visto isso, veio a tona a ideia de trazer o usuário como uma ajuda direta a problemática e recompensa-lo caso seja impactante na resolução de um bug.

Bug é um problema inesperado com um software ou hardware, geralmente é um problema causado por alguma interferência externa que o desenvolvedor não previu. Bugs podem variar na sua severidade, pequenos bugs podem causar problemas pequenos como, congelamento de telas ou mensagens de erros inexplicáveis, já grandes bugs podem causar problemas severos, como, parada total de funcionamento ou danificar dados ou problemas de integração [9].

Com a consolidação do mercado de software e hardware, os bugs foram se tornando cada vez mais preocupantes, e com isso foi colocado em prática o conceito de bug reports para facilitar no rastreamento e na gerencia de bugs. Através do estudo de [5] temos que, um bom bug report é um relatório que indica à equipe técnica qual foi o problema experienciado pelo usuário e informações sobre em qual ambiente a problemática aconteceu e pistas de como reproduzi-lo. Portanto, faz-se necessário descobrir quais campos mais pertinentes e quais o próprio usuário, mesmo que sem conhecimento técnico, poderia preencher, a fim de ter autonomia para criar um bom bug report inicial.

Através do trabalho de [5], encontramos alguns campos em um bug report que na maioria das vezes podem ser facilmente preenchidos por usuários comuns, eles são: título de um relatório, sua descrição, um passo a passo e screenshots do problema. São campos simples e intuitivos o suficiente para serem pelo menos parcialmente preenchidos por usuários comuns de um aplicativo, e ainda assim criar um bom bug report inicial para que uma equipe técnica consiga trabalhar em cima.

4 CONNECTBUG

ConnectBug é uma ferramenta é proposta para ter sua interface para o usuário disponível primordialmente em aplicativos construídos usando react-native [15] e Expo [7] por serem duas frameworks para aplicativos mobile cross-platform, o que significa que os aplicativos construídos com essas frameworks funcionam tanto em ambientes *android* quanto *ios* dando assim mais possibilidades para o uso da ferramenta.. Sua principal função será a de ser uma forma rápida e acessível para clientes reportarem um bug na aplicação.

ConnectBug é acionada através de um botão, que por padrão fica sobreposto no canto inferior direito da aplicação. Ao clicar nesse botão abre-se um modal na tela, com um formulário contendo os principais campos que uma pessoa não técnica consegue preencher para um bug report, "título", "descrição", "passos para reproduzir" e "screenshots", seguidos de um botão que confirma a criação de um novo bug report com esses dados. A ferramenta também ficará responsável por coletar automaticamente as especificações técnicas do aparelho do cliente, via API, e pela coleta da versão do aplicativo em que a ferramenta está instalada.

Os bugs relatados poderão ser vistos em um painel de controle a parte acessada via navegador. No painel, além da visualização dos reportes, também será possível escolher um responsável por analisar o relato, escrever comentários sobre o mesmo para auxiliar futuros desenvolvedores que possam vim trabalhar no mesmo. Poderá também ser feito o gerenciamento de usuários que terão acesso ao dashboard e possíveis recompensas que poderão ser dadas aos usuários que relatarem algum bug importante.

4.1 Funcionalidades

As funcionalidades foram feitas com bases em casos de usos estruturados para cada entidade que faz parte do escopo do projeto, sendo elas: bug reports, usuários e recompensas. Os casos de uso estão ilustrados na Figura 1.

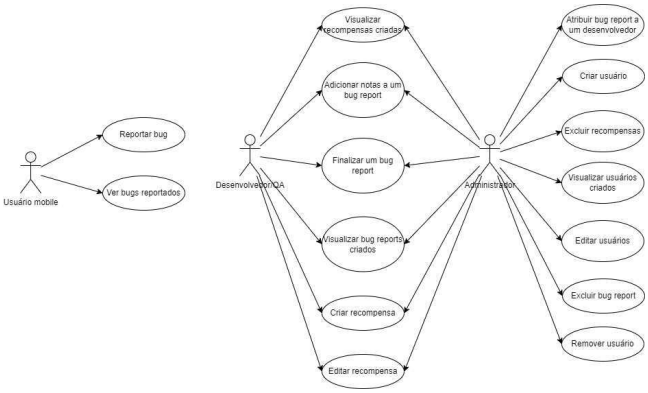

Figura 1: Diagrama de casos de uso do projeto

4.1.1 Bug reports.

Bug report é a entidade central da ferramenta. É a partir da criação deles no aplicativo que a interface mobile da ferramenta está instalada, por parte do usuário e que é dado o inicio do fluxo de análise, investigação e conserto de um bug pela da equipe técnica.

No que diz respeito aos bugs reports temos os seguintes casos de uso:

(1) Na interface mobile é possível, criar um novo bug report, preenchendo os dados no formulário.
(2) Na interface mobile é possível, o usuário pode visualizar os seus bug reports criados.
(3) No dashboard, é possível visualizar os bugs reports criados pelos usuários em forma de tabela.
(4) Abrir um modal que mostra detalhadamente todas informações concedidas pelo usuário que criou o bug report.
(5) No dashboard é possível, adicionar notas no bug report para comunicação interna dos desenvolvedores.
(6) No dashboard é possível, aprovar ou negar um bug report, caso seja aprovado o bug report tem que ser atribuido a um desenvolvedor cadastrado na dashboard, esse será responsável pela investigação e resolução do bug.
(7) No dashboard, é possível concluir a investigação de um bug report, tendo a opção de ceder uma recompensa ao usuário que reportou o bug.

As telas referentes a esses casos podem ser visualizadas nas Figuras 2 a 7.

Figura 2: Tela de criação de bug report

4.1.2 Recompensas.

Recompensas são gatilhos que disparam após um desenvolvedor finalizar de investigar um bug report. Em essência são uma chamada HTTP do tipo POST feitas para um webhook cadastrado por um administrador. Nessa chamada são enviadas informações como o ID do usuário que reportou o bug report e o ID do bug report em si. Com isso o time de desenvolvimento tem total liberdade para recompensar o usuário no contexto do seu aplicativo. Por exemplo, caso o aplicativo onde o ConnectBug esteja integrado seja um e-commerce uma possível recompensa poderia ser um cupom para o usuário que ajudou a solucionar o problema.

No que diz respeito às recompensas temos os seguintes casos de uso:

(1) No dashboard é possível, criar uma nova recompensa.
(2) No dashboard é possível, listar recompensas já criadas.
(3) No dashboard é possível, editar uma recompensa.
(4) No dashboard é possível, excluir uma recompensa.

As telas referentes a esses casos podem ser visualizadas nas Figuras 8 e 9.

Figura 3: Tela de bug reports criados no mobile

Figura 4: Tela de bug reports criados na dashboard

4.1.3 *Usuários.*

Usuários cadastrados no dashboard de administração podem acessar o mesmo, como mostrado na Figura 10. Quando instalada pela primeira vez, a dashboard já vem com um usuário padrão criado com o cargo de administrador, tendo assim todo poder sobre as ações de gerenciamento.

No que diz respeito aos usuários temos os seguintes casos de uso:

(1) No dashboard é possível, criar um novo usuário.
(2) No dashboard é possível, listar usuários já criadas.
(3) No dashboard é possível, editar informações básicas de um usuário.
(4) No dashboard é possível, excluir um usuário, caso a pessoa realizando a ação seja do cargo administrador.
(5) No dashboard é possível, o usuário logado possa alterar seus dados sensíveis.

As telas referentes a esses casos podem ser visualizadas nas Figuras 11 e 12.

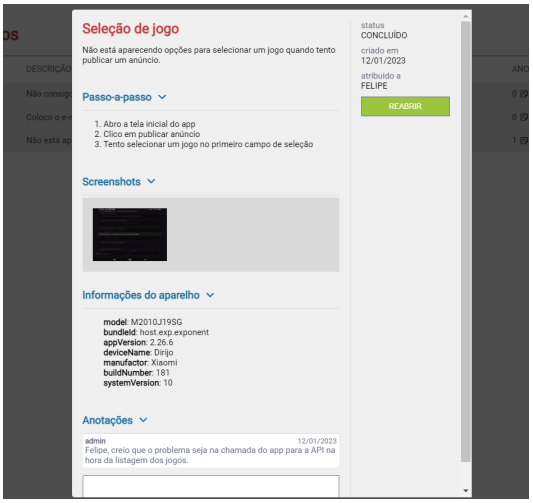

Figura 5: Tela de detalhes de um bug report

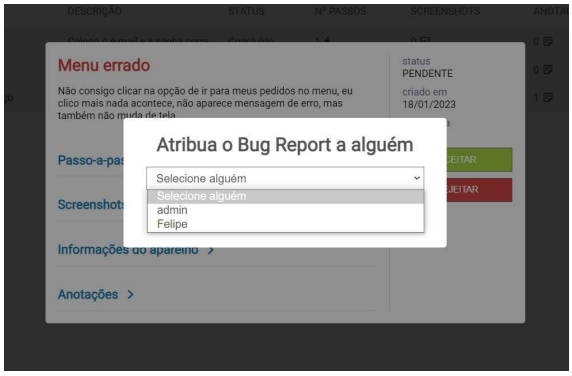

Figura 6: Tela de atribuição de bug report a um responsável

Figura 7: Tela de conclusão de um bug report e atribuição de recompensa

Figura 8: Tela de recompensas criadas

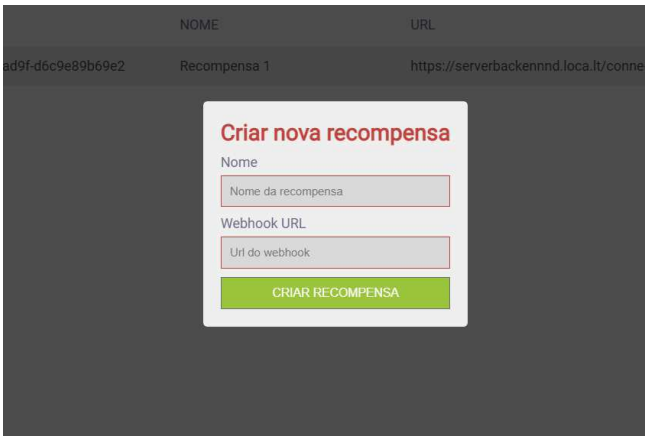

Figura 9: Tela de criação de recompensa

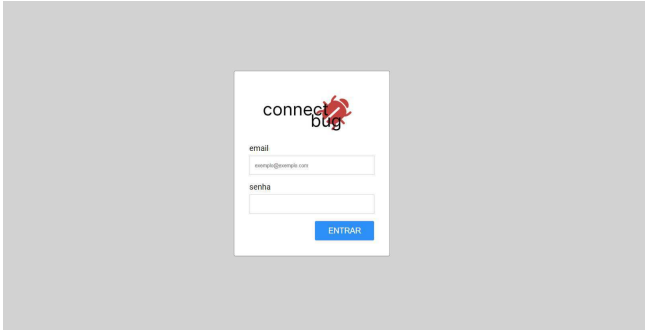

Figura 10: Tela de login da dashboard

4.2 Arquitetura

No projeto, foi utilizada a arquitetura Cliente-Servidor, umas das arquiteturas mais utilizadas atualmente no mundo web. Ela consiste em dois atores, um sendo o Cliente que faz requisições em busca de alguma informação e o Servidor, que é responsável por prover essas informações. Nesse contexto, quando o cliente necessita de alguma informação é estabelecido um protocolo entre ele o servidor, sendo o mais comum o HTTP [25], assim, o cliente envia uma requisição para o servidor utilizando uma linguagem comum para ambos e

Figura 11: Tela de usuários cadastrados na dashboard

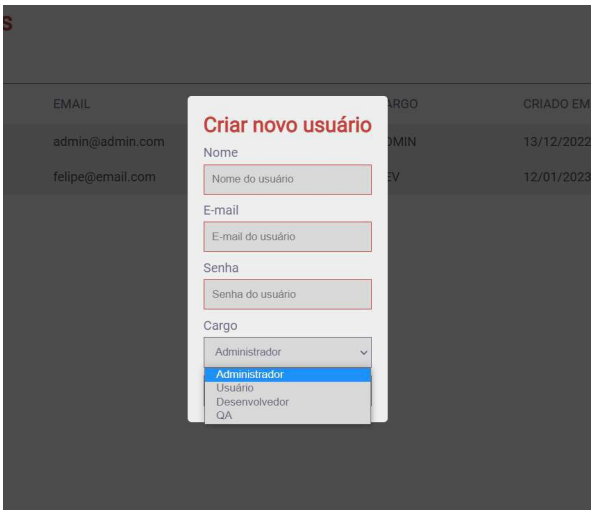

Figura 12: Tela de cadastro de novo usuário

o servidor o responde com uma resposta apropriada. Essa foi a arquitetura usada, tendo um servidor e dois clientes, a ferramenta mobile e o dashboard web. A comunicação utilizada entre eles foi o JSON.

4.2.1 Banco de dados.

Para haver persistência de dados foi utilizado o Postgres [20], um banco de dados relacional, escolhido por ser uma solução open source e gratuita, além de experiência prévia própria utilizando a linguagem de estrutura . Foi feito um levantamento de requisitos que proporcionou uma modelagem de dados, onde o sistema gira em torno da entidade de bug reports, acompanhada das tabelas *steps*, responsável por guardar os passos para se chegar ao bug, *screenshots* onde ficam a *url* das imagens cedidas pelo relator do bug report e *notes* onde ficam as anotações feitas por desenvolvedores e testadores durante a investigação do bug. Também temos a tabela de *rewards* onde fica guardado o método da chamada HTTP e o *endpoint* chamado quando uma recompensa é disparada. No mais, temos a tabela de *users* onde ficam as informações dos usuários que tem acesso a dashboard, os modelos estão ilustrados na Figura 13.

4.2.2 Backend.

O *backend* funciona como uma forma de fazer a conexão entre o

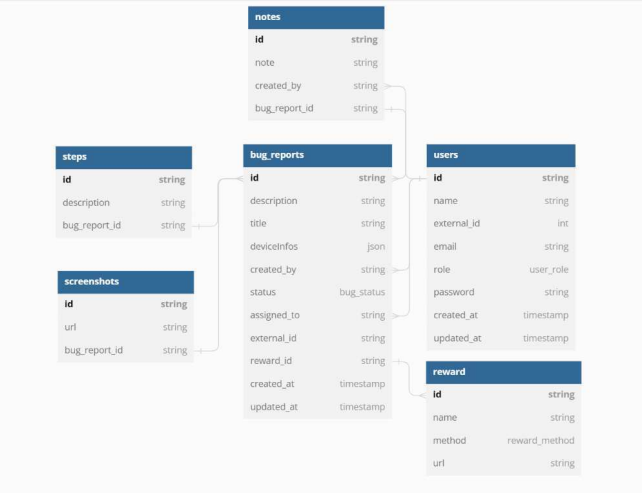

Figura 13: Modelo de dados utilizado

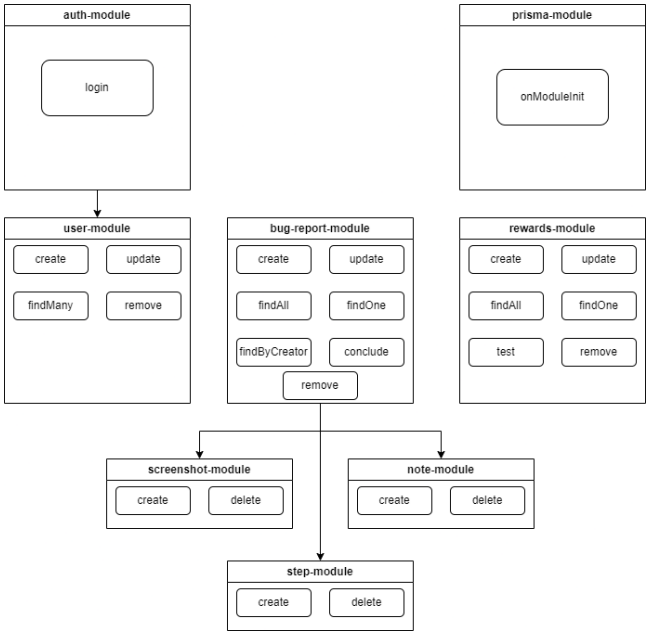

Figura 14: Diagrama de módulos

cliente e o banco de dados. O servidor é responsável por interações como adição, remoção, edição e recuperação de dados do *backend*. Também é onde fica lógicas de autenticação, assim orquestrando quem pode ou não pode acessar os diferentes dados, e além disso é encarregado pela maneira como esses dados serão tratados e entregues ao cliente. Dado o cenário, o servidor foi construído seguindo o conjunto de restrições de arquitetura REST [2], trazendo benefícios como escalabilidade, flexibilidade e independência, além da facilidade do uso por experiências prévias.

Na construção do servidor foi utilizado o NodeJs com auxílio da framework NestJs [17] para a estruturação do projeto seguindo o modelo REST. Além disso, para modelagem e conexão do NestJs com o banco de dados Postgres foi utilizado o ORM Prisma [21]. Já para o armazenamento de imagens foi utilizado o host gratuito ThumbSnap [26]. Por fim, para o processo de desenvolvimento e futuro deploy na cloud foi utilizado o Docker [6]. As tecnologias foram principalmente escolhidas por serem gratuitas e também por familiaridade com algumas e desejo de aprendizado de outras, por exemplo o NestJs.

Por se tratar de um projeto utilizando NestJs, foi seguida a arquitetura de pastas modulares, onde as entidades são separadas em módulos, onde cada módulo é responsável por seu *controller*, *service*, entidades e *dtos*, como ilustrado na Figura 16. Assim trazendo independência a cada módulo e facilitando eventuais manutenções. Os módulos utilizados no projeto estão ilustrados na Figura 14.

Cada módulo tem sua responsabilidade, como descrito a seguir:

- Auth-module: Módulo responsável por fazer a autenticação de usuários na API em geral. Ele é encarregado tanto por gerar tokens JWT [13] quanto pela validação dos mesmo, é esse módulo que faz a segurança de todas rotas privadas da aplicação. Ele se comunica diretamente com o módulo de usuários a fim de validar a autenticidade de uma requisição.
- User-module: Módulo responsável por gerir usuários da aplicação, ele é quem cria, faz updates, encontra e remove usuários do banco de dados.
- Prisma-module: Módulo responsável por controlar a conexão do ORM prisma [21] com o banco de dados postgres [20] e disponibilizar essa conexão aos demais módulos.
- Reward-module: Módulo responsável por gerir recompensas da aplicação. Ele é quem cria, faz updates, encontra e remove recompensas do banco de dados.
- Bug-report-module: Módulo responsável por gerir bug reports, a entidade central do projeto. Esse módulo controla tudo sobre bug reports, e conversa diretamente com os módulos de notas, screenshots e steps, pois essas entidades também compõe um bug-report final. Nesse módulo, são onde são criados os bug reports, onde ocorre toda lógica de atribuição e finalização de bug reports, além das funções básicas de editar um bug report e deleta-los.
- Note-module: Módulo responsável por gerir notas da aplicação. Ele é quem cria, faz updates, encontra e remove recompensas do banco de dados. Notas são na sua essência comentários que os desenvolvedores podem fazer sobre um bug report enquanto o investigam.
- Screenshot-module: Módulo responsável por gerir screenshots da aplicação. Ele é quem faz o upload das imagens enviadas pelo usuário para um serviço de hospedagem, além de poder deletar essas imagens caso requisitado.
- Step-module: Módulo responsável por gerir steps da aplicação. Ele é quem cria e deleta os passos que o usuário cadastra na criação de um bug report.

Uma exceção do padrão da figura 15 é o módulo de autenticação que conta com mais algumas pastas na sua estrutura, que são: *decorators*, *errors*, *guards*, *middlewares*, *models* e *strategies*, que são pastas exclusivas para o controle de autenticação e autorização das rotas do projeto, ilustradas na Figura 16. Outro módulo a parte da

estrutura padrão é o do Prisma ORM, é nele que são definidos os *schemas* das entidades, feito as *migrations* para o banco de dados e onde é encontrado o script *seed.ts* responsável por inserir dados iniciais ao banco, a estrutura é ilustrada pela Figura 17.

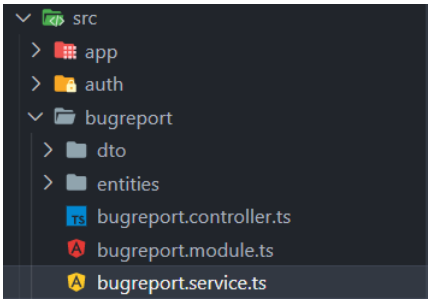

Figura 15: Estrutura de módulo padrão NestJs

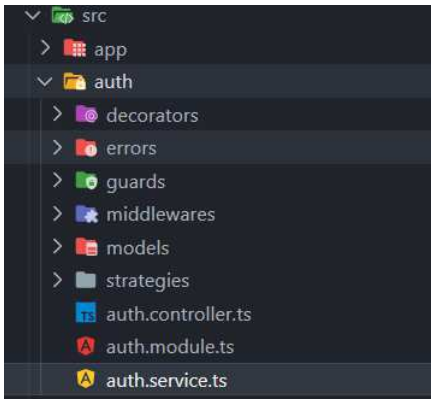

Figura 16: Estrutura do módulo auth

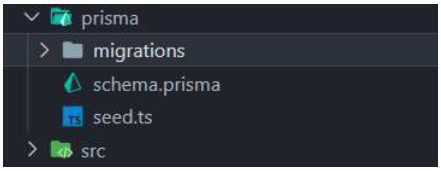

Figura 17: Estrutura de pastas do Prisma ORM

4.2.3 Frontend.

O frontend foi feito para ser utilizado pela equipe técnica de desenvolvimento e testes da aplicação onde a ferramenta for instalada. Também é o cliente principal do servidor, consumindo, alterando e provendo novos dados para a ferramenta no geral. Na sua construção foi utilizada a framework NextJs [18] que tem como objetivo adicionar mais funcionalidades à biblioteca de interface React, utilizando o TypeScript como a linguagem de programação principal. Esta foi escolhida por trazer uma maior facilidade no encapsulamento de componentes web e na facilidade de leitura e compreensão do código por terceiros, além de ter uma comunidade bem ativa, o que facilita futuras contribuições open-source da ferramenta. Também visando trazer mais agilidade na concepção dos componentes visualmente, foi utilizada a biblioteca styled-components [23] que integra o uso de CSS e TypeScript, trazendo mais utilidade e formas de criar componentes dinâmicos.

Na estruturação de pastas também foi utilizada uma arquitetura similar a modular, onde cada entidade tem sua pasta separada, e tudo que essas entidades precisam para existir, como serviços para fazer conexão com o *backend* e estrutura de tipagens, como ilustrado na Figura 18. Para os componentes visuais foi seguido o padrão do próprio NextJs, onde há uma pasta principal chamada *pages* onde ficam todos os arquivos de páginas da aplicação. Cada pasta dentro da *pages* é traduzida em um rota da aplicação, então por exemplo a pasta *home* simboliza o endpoint */home* da aplicação. Também existe a pasta *components* onde ficam agrupados os componentes menores da aplicação, a pasta *public* onde ficam concentrados os assets gerais de imagens, a pasta *styles* onde ficam as estilizações globais, a pasta *shared* onde ficam scripts de lógica utilizados por toda aplicação, *hooks* onde ficam funções feitas especialmente para o ambiente do React que podem ser reutilizadas por seus componentes e por fim a pasta *utils* onde ficam as funções utilitárias, essas pastas podem ser vistas destacadas na Figura 19.

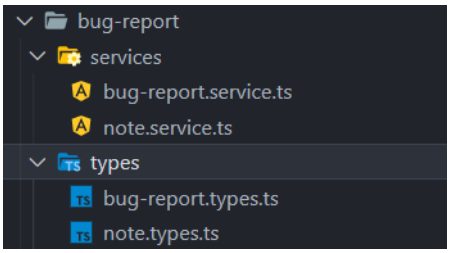

Figura 18: Estrutura de módulo padrão do frontend

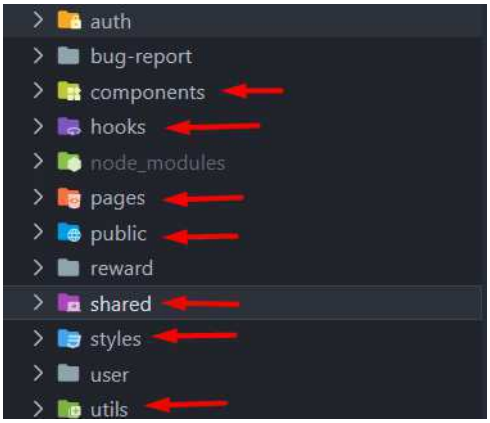

Figura 19: Demais pastas da estrutura do frontend

4.2.4 Mobile.

A interface da ferramenta em si para clientes reportar bugs foi feita utilizando *react-native*, utilizando o Expo [7], framework que

visa facilitar o desenvolvimento trazendo soluções para bundle em apps. Esta stack foi escolhida por ser nativa tanto para aplicações IOS quanto para Android, e por ser a framework mais utilizada no desenvolvimento de apps cross-platform [24].

A ferramenta desenvolvida é uma biblioteca que pode ser instalada em qualquer projeto expo ou react-native. Ela foi feita para ser intuitiva e fácil de criar um novo bug report, assim na intenção de incentivar usuários a criar bug reports e ajudar os desenvolvedores do app em que a ferramenta foi instalada a ter um maior vínculo com sua comunidade e seus usuários, além de ajudar na manutenção de falhas do app.

A estrutura de pastas dela é simples, conta apenas com um arquivo onde toda a lógica do componente é escrita o *App.tsx*. A demais temos o arquivo *tsconfig.ts* para configurar a transpilação do código feito em TypeScript em JavaScript. Conta também com o *package.json* onde ficam as informações para buildar o componente e transformá-lo em uma biblioteca, que pode ser publicada e instalada via o gerenciador de pacotes npm. Além disso, também temos a pasta *types* onde ficam as definições de tipos usadas pelo componente, a pasta *constants* onde ficam definidas constantes usadas e a pasta *assets* onde ficam as imagens usadas, como ilustrado na Figura 20.

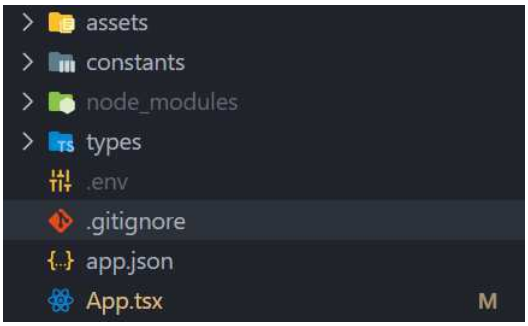

Figura 20: Estrutura de pastas mobile

4.3 Processo de desenvolvimento

O desenvolvimento foi feito utilizando a metodologia de desenvolvimento ágil Scrum, onde foram estabelecidas sprints de 15 dias, com tarefas criadas com base nos use cases da ferramenta e priorizadas com base na sua importância para o desenvolvimento do MVP (*Minimum Viable Product*).

Para o versionamento, foram iniciados três repositórios no *Github* um para cada aplicação a ser criada: *mobile*, *frontend* e *backend*. Para cada tarefa foi criada uma nova *branch* no seu respectivo repositório, assim facilitando o versionamento de código e possíveis *rollbacks* caso acontecesse algum imprevisto durante o desenvolvimento de uma nova *feature*.

A seguir a ordem em que ocorreu o fluxo de desenvolvimento:

(1) Iniciou-se na criação dos use cases que iriam ser abordados pela a ferramenta com um todo, a partir disso foi se estruturando quais seriam os fluxos do software.
(2) Com base nos fluxos do software, foram definidos quais seriam os casos de uso da ferramente.
(3) A partir dos casos de uso foi construído o modelo de dados a serem utilizados.
(4) Após isso, foi levantado um backlog de tarefas que seriam necessárias para conclusão do projeto. Com o backlog feito, começou a etapa de desenvolvimento. As tarefas foram catalogadas por prioridade (Baixa, Média ou Alta) onde foram ordenadas a fim que as com a prioridade Alta fossem trabalhadas primeiro.
(5) Devido a prioridade, as primeiras tarefas escolhidas foram: cadastrar bug reports na ferramenta, lista-los, atribui-los e conclui-los no dashboard administrativo, cadastrar, listar e autenticar usuários, e cadastrar, listar e editar recompensas no dashboard. Essas foram escolhidas como mais alta prioridade por serem necessárias para o funcionamento geral da ferramenta no seu propósito principal.
(6) Após isso, iniciou-se as configurações iniciais de cada projeto, utilizando em todos a ferramenta de análise estática ESLint e o formatador de código opinativo Prettier para certificar que a escrita do código seguisse uma estrutura e um padrão bem definido entre todas as aplicações.
(7) A implementação começou pelo backend, por se tratar do servidor e servir os dados a serem consumidos pelos clientes mobile e frontend, assim construindo toda a lógica que seria utilizada pelas interfaces gráficas.
(8) Na sequência foi desenvolvido o dashboard de administração por se tratar de ser web onde por experiência prévia pôde ser trabalhado com mais agilidade, sobrando assim mais tempo para trabalhar na aplicação mobile.
(9) Foi desenvolvido a parte mobile, que compõe o formulário onde o usuário do aplicativo reportará bugs que encontrar.
(10) Com a finalização do desenvolvimento, foi feito testes manuais tanto na ferramenta mobile quanto no dashboard administrativo.
(11) Após, foi gravado um vídeo demonstrativo da ferramenta.
(12) O vídeo demonstrativo foi apresentado a um grupo de pessoas, e após isso foi pedido para que respondessem um formulário com questões sobre a demonstração exibida.
(13) Com as respostas foi possível tirar alguns conhecimentos descritos na sessão a seguir: Avaliação.

O *backend* e o *frontend* tiveram seu *deploy* feito em um ambiente de testes para comprovar seu funcionamento em conjunto, enquanto a parte mobile por ser uma ferramenta instalável em aplicações *react-native*, foi configurada e testada em aplicativos open-source encontrados em repositórios públicos do *Github*.

5 AVALIAÇÃO

Antes do sistema ser colocado em uso aberto para o público foi feito uma série de testes manuais tanto no aplicativo quanto no dashboard de administração, foi testada cada feature, tanto sua adição, edição e remoção.

5.1 Metodologia

Foi feito um vídeo demonstrativo da ferramenta e da dashboard administrativa em uso [8]. Esse vídeo foi divulgado por meio de

convite para um grupo de 12 pessoas, entre elas profissionais da área de tecnologia e graduandos do curso de ciência da computação, onde 9 destas participaram, avaliando a interface e o uso da ferramenta. Após assistirem a demonstração, foi pedido para que esse grupo respondesse um questionário, criado no Google Forms. O formulário consiste em perguntas abertas e fechadas, onde as fechadas utilizavam a escala de resposta Likert [12], que apresenta uma faixa de números que vão de 1 a 5, onde cada número representa respectivamente opções como Ruim, Razoável, Bom, Muito bom e Excelente. As questões foram:

5.1.1 *Questões referentes à ferramenta mobile.*

(1) "Como você avalia a facilidade de uso da ferramenta de relatório de bugs?"
(2) "A ferramenta se mostrou útil para fornecer detalhes suficientes sobre o erro para que o time de desenvolvimento possa corrigi-lo?"
(3) "Você recomendaria a ferramenta de relatório de bugs para outros usuários do aplicativo?"
(4) "Você acha que a ferramenta de relatório de bugs foi projetada pensando nas necessidades dos usuários?"
(5) "Como você avalia a facilidade de navegação na interface da ferramenta de relatório de bugs?"

5.1.2 *Questões referentes à dashboard administrativa.*

(1) "Como você avalia o entendimento dos bug reports criados e suas informações?"
(2) "O fluxo de atribuição e conclusão de um bug report se mostrou de fácil compreensão?"
(3) "Como você avalia o fluxo para criação de novos usuários na dashboard?"
(4) "Como você avalia o seu entendimento do conceito de recompensas na conclusão de um bug report?"
(5) "Qual sua satisfação com a dashboard administrativa?"

5.1.3 *Questões referentes ao uso em conjunto.*

(1) "Em relação ao uso em conjunto da ferramenta com o dashboard administrativo, você acha que traz um auxílio no momento que um usuário encontra um bug até o momento em que o desenvolvedor precisa avaliar esse bug e resolvê-lo?"
(2) "Gostaria de deixar algum feedback adicional?": pergunta com resposta aberta.

5.2 Resultados

Foram coletadas no total 9 respostas. Após feita a análise, é possível comprovar que tanto a ferramenta mobile quanto o dashboard atendem ao que se sujeitam, como ilustrado na Figura 21, onde mostra a média das respostas das perguntas 1, 2, 3, 4 e 5, referentes tanto as perguntas sobre a ferramenta mobile (barra em azul), quanto as perguntas sobre a dashboard (barra em vermelho). Mas também fica claro, ao observar as respostas nos campos abertos que, há pontos de melhorias a serem desenvolvidos, os pontos mais citados foram:

- Adicionar notificações de atribuições de bugs a desenvolvedores na dashboard.
- Possibilidade de além de prints ser possível enviar vídeos do fluxo do bug report.
- Na dashboard é possível integração com softwares de desenvolvimento ágil como Jira e Trello.

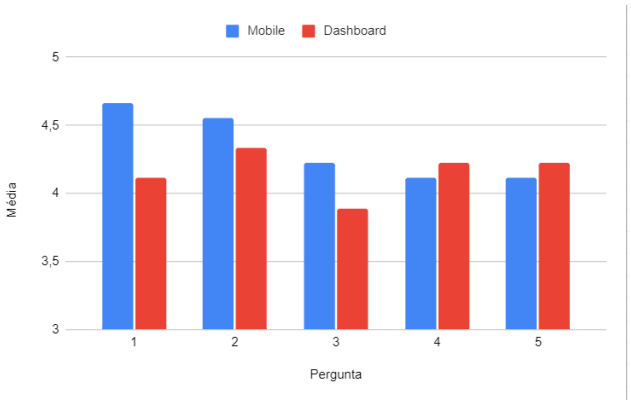

Figura 21: Gráfico vertical em barras da média entre as respostas das perguntas de 1 a 5, tanto para aplicação mobile, quanto para aplicação web

Quando observada a média das respostas de satisfação com a ferramenta no quesito de resolução e facilitamento de criar e administrar bug reports percebe-se que ela não resolve completamente o problema, mas ajuda em um nível significativo e tem muito espaço para evoluir a fim de cada vez auxiliar mais nesta problemática, como ilustra a Figura 22. Quanto à pergunta 2 da sessão 5.1.3, referente ao uso conjunto da ferramenta e da dashboard, foram coletados os seguintes feedbacks:

- "Achei legal, principalmente a parte de notificar quando o bug resolve e o usuário é notificado."
- "Acho que onde observei mais pontos de melhoria seria na interface gráfica da ferramenta, poderia ter mais opções de personalização, para desenvolvedores conseguissem deixar a ferramenta com o mesmo tema do aplicativo onde ela está instalado."
- "Gostei da ideia de dar recompensas quando um usuário reportar um bug."

6 CONSIDERAÇÕES FINAIS

Para o desenvolvedor o projeto completo foi necessário conhecimento de múltiplas áreas da criação de um software, desde a parte teórica com a idealização de modelos e diagramas de uso e fluxo, como conhecimento de partes mais práticas na implementação da ferramenta, dashboard e server. Foram explorados os aprendizados coletados em várias cadeiras do curso como: Programação 1 e 2, Laboratório de Programação 1 e 2, Projeto de Software, Engenharia de Software, Estrutura de Dados, Banco de Dados, Análise de Sistemas e Metodologia Científica.

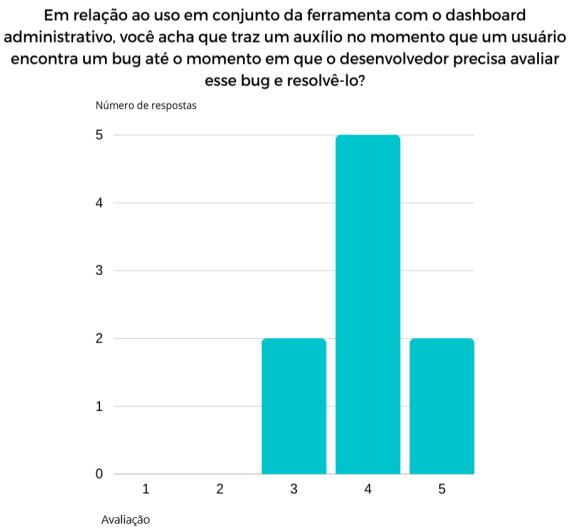

Figura 22: Gráfico em barras mostrando as respostas referentes a pergunta se a ferramenta ajudava ou não a solucionar o problema proposto

6.1 Desafios

O maior desafio desse trabalho foi o tamanho do projeto em si, por se tratar de três aplicações que trabalhavam em conjunto. O tempo foi um quesito chave, devido ao estudo necessário em cima do que se tratava um bom bug report e como achar uma maneira de um usuário sem conhecimento técnico cria-los. Também foi preciso uma parcela de tempo para levantar as especificações, casos de uso e modelar como seriam guardados os dados e como seguiria o fluxo da aplicação, além do tempo para implementar a ferramenta em si.

Outro desafio foi as tecnologias, pois mesmo já tendo experiência prévia com algumas, outras foram um grande desafio, como a *framework* NestJs em conjunto com o ORM Prisma, e mais ainda a parte *mobile* pois ela envolve ter que extrair dados técnicos do aparelho do usuário, o que é algo que foge um pouco da implementação de um aplicativo ou um formulário.

Por último, a forma de validação, pois a ferramenta foi feita para ser algo integrado a aplicativos em desenvolvimento ou já prontos, então fazer essa adaptação e testes em diversos aplicativos requereu um bom tempo de pesquisa e testagem. Além encontrar um grupo de pessoas que tivessem conhecimento técnico para poder avaliar e dar feedbacks tanto sobre a parte *mobile*, voltada para usuários comuns, quanto o dashboard administrativo, voltada para desenvolvedores e gestores técnicos.

6.2 Limitações e Trabalhos futuros

Embora o ConnectBug já traga uma base como facilitador e ponte dos usuários comuns com os devs, ficou claro que muitas coisas ainda podem ser melhoradas nesse processo, ideias como as coletadas nos feedbacks dos testadores:

- Melhor personalização da ferramenta para se integrar melhor com o aplicativo onde está sendo instalada.
- Integração com plataformas de gerenciamento de projetos ágeis.
- Forma mais direta da equipe técnica contactar os usuários que reportaram bugs, como por exemplo um chat entre eles.
- Expandir o escopo da ferramenta para atender outras plataformas além de aplicativos criados com react-native, como por exemplo uma versão que atenda sites webs ou aplicações desktop.
- Criar mais testes para aplicação e aumentar sua praticidade para ser instalada.

7 AGRADECIMENTOS

Gostaria de agradecer principalmente a meus amigos que me ajudaram com conhecimentos técnicos em áreas que eu não tinha tanta experiência, como a criação da aplicação mobile, e me incentivaram a seguir trabalhando no projeto em momentos que me via desmotivado, e a minha família por me deixar confortável para conseguir focar no desenvolvimento e escrita do trabalho. Por último, mas não menos importante, gostaria de agradecer aos meus orientadores, Franklin Ramalho e Tiago Massoni, que me ajudaram e me guiaram na idealização da ferramenta e na escrita desse trabalho.

REFERÊNCIAS

[1] Amorim, A. A. N. B. B., and de Sousa Pinto, F. Webbug: uma ferramenta para gestão de erros em projetos de software.
[2] AWS. O que é a api restful? – guia de api restful para iniciantes – aws.
[3] BugZilla. Bugzilla :: bugzilla.org.
[4] BuildFire. Mobile app download statistics usage statistics (2023) - buildfire.
[5] Davies, S., and Roper, M. What's in a bug report? *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 26 (2014), 1–10.
[6] Docker. What is a container?
[7] Expo. Expo.
[8] Freitas, J. Connectbug - vídeo demonstrativo.
[9] Gartner. Definition of bug - gartner information technology glossary.
[10] Hassan, S., Bezemer, C.-P., and Hassan, A. E. Studying bad updates of top free-to-download apps in the google play store. *IEEE Transactions on Software Engineering 46*, 7 (2020), 773–793.
[11] Instabug. Superior mobile app performance. improved user experience.
[12] Joshi, A., Kale, S., Chandel, S., and Pal, D. K. Likert scale: Explored and explained. *British journal of applied science & technology 7*, 4 (2015), 396.
[13] JWT. Json web tokens - jwt.io.
[14] Khalid, H., Shihab, E., Nagappan, M., and Hassan, A. E. What do mobile app users complain about? *IEEE Software 32*, 3 (2015), 70–77.
[15] Native, R. React native · learn once, write anywhere.
[16] Nayebi, M., Adams, B., and Ruhe, G. Release practices for mobile apps–what do users and developers think? 552–562.
[17] NestJs. Nestjs - a progressive node.js framework.
[18] NextJs. Next.js by vercel - the react framework.
[19] Palomba, F., Linares-Vásquez, M., Bavota, G., Oliveto, R., Di Penta, M., Poshyvanyk, D., and De Lucia, A. User reviews matter! tracking crowdsourced reviews to support evolution of successful apps. 291–300.
[20] PostgreSQL. Postgresql: The world's most advanced open source relational database.
[21] Prisma. Prisma | next-generation orm for node.js typescript.
[22] Shakebug. Free bug crash tracker tool for android ios - shakebug.
[23] Styled-components. styled-components.
[24] Technostacks. Top 10 mobile app development frameworks in 2023 | technostacks.
[25] TERRA, J. What is client-server architecture? everything you should know.
[26] ThumbSnap. Thumbsnap - free photo video hosting.
[27] Vailshery, L. S. Cross-platform mobile frameworks used by developers worldwide 2019-2021.