Sistema de Informação na plataforma Android para abertura de chamados dentro do IFSP Campus Hortolândia

Matheus da Silva Vassoler, Daniela Marques

Instituto Federal de Educação, Ciência e Tecnologia de São Paulo - Campus Hortolândia Avenida Thereza Ana Cecon Breda, s/n, Vila São Pedro - Hortolândia – SP – Brasil

matheus.vassoler@aluno.ifsp.edu.br, marquesdaniela@ifsp.edu.br

Abstract. *This paper presents the development of an Android App, that allows the opening of tickets by scanning QR Codes near the assets of the IFSP Campus Hortolândia, in which may present a problem. Currently already exist solutions to open ticket and the project aims to provide one more option for teachers and students.*

Resumo. *Este artigo apresenta o desenvolvimento de um aplicativo Android que permita a abertura de chamados por meio do escaneamento de QR Codes próximos aos patrimônios do IFSP Campus Hortolândia, nos quais possam apresentar algum problema. Atualmente já existem soluções para abertura de chamado e o projeto visa disponibilizar mais uma opção para os docentes e discentes.*

1. Introdução

Sistema de Informação é um tema que está presente na sociedade desde o século XX, período este sendo considerado como advento da era da informação. Naquele período, notou-se uma grande importância da coleta, armazenamento e processamento de dados para embasamento de tomadas de decisões. Porém, seu uso era restrito a uma pequena parcela da sociedade e demandava muito esforço para a utilização dos sistemas, visto que os computadores da época eram pouco acessíveis e de complexa usabilidade [Cosentino 2006]. Atualmente, os Sistemas de Informação são utilizados por uma ampla camada da população, tendo diferentes finalidades de uso.

Sistemas de Informação, ou simplesmente SI, são sistemas capazes de coletar, armazenar, processar e transmitir dados entre usuários. Empresas e instituições têm cada vez mais adotado sistemas que facilitem a comunicação e a realização de tarefas. Um exemplo são os sistemas de chamados, uma ferramenta comum dentro do departamento de Tecnologia de Informação (TI) de uma organização. Nela, as pessoas solicitam demandas, apontam dificuldades ou erros, que serão recebidos pelo setor de TI. Essas demandas são os chamados, que serão avaliados e respondidos pelos responsáveis técnicos [Milvus 2019]. Os sistemas de chamados são importantes, pois são capazes de centralizar as informações em um único ponto, além de facilitar o contato entre o usuário e o departamento de TI.

Atualmente, o Instituto Federal de São Paulo (IFSP) campus Hortolândia possui um sistema para abertura de chamados, sendo utilizado por servidores e discentes quando os mesmos possuem problemas com os equipamentos do campus. O sistema de informação utilizado é o GLPI *(Gestionnaire Libre de Parc Informatique)*, em português, Gestor de Equipamentos de TI de Código Aberto. Ele é um sistema de gestão de serviços e gerenciamento de ativos 100% web, permitindo gerenciar chamados de TI, manutenção de equipamentos, instalações prediais, dentre outros recursos [GLPI 2020]. Além disso,

é possível registrar as ocorrências que acontecem com os patrimônios. O sistema é acessado através da página inicial do portal web da Instituição.

Existem duas maneiras de se abrir um chamado no GLPI do campus Hortolândia, a primeira forma é procurar manualmente o formulário, no qual possui as seguintes etapas:

1. Procurar a categoria do problema
2. Verificar dentro da categoria qual o item está com problema
3. Após navegar até o item, haverá um *link* que redireciona o usuário para o formulário

A segunda forma é escanear um QR Code *(Quick Response Code)* que leva o usuário diretamente para o formulário [Esteves 2021]. Nesta segunda opção, o QR Code possui o *link* do formulário com algumas informações adicionais, uma vez escaneado no celular, o dispositivo móvel abre o navegador e carrega o formulário preenchendo algumas informações de forma automática.

Diante dessas duas opções, essa pesquisa tem o objetivo de apresentar um protótipo de aplicativo Android como terceira forma para se abrir chamados dentro do campus Hortolândia. A ideia é que o aplicativo escanei o QR Code gerado no trabalho do Esteves (2021) e abra o formulário utilizando os componentes nativos do Android, isto é, ao invés de abrir o formulário da versão web, será aberto um formulário dentro do próprio aplicativo, utilizando os recursos do Android.

Nas próximas seções desse trabalho é possível conferir a justificativa, referencial teórico, materiais e métodos, desenvolvimento e conclusão. A justificativa visa elucidar o porque de desenvolver o projeto, apresentando dados e informações. O referencial teórico embasa o leitor sobre alguns temas abordados na pesquisa. Materiais e métodos apresentam quais ferramentas e estratégias foram utilizadas para o desenvolvimento do aplicativo, o desenvolvimento demonstra as etapas da criação da aplicação Android e a conclusão realiza o fechamento, repassando e relembrando alguns pontos abordados na pesquisa.

2. Justificativa

Nota-se que hoje a sociedade tem cada vez mais utilizado celulares. Para se ter uma ideia, o Brasil possui atualmente 234 milhões de *smartphones* em uso [FGV 2020]. Através desse dispositivo é possível realizar diversas tarefas de forma simples e rápida, por exemplo, consultar notícias, escrever mensagens, navegar em redes sociais, dentre outras funções. Pensando nos benefícios, facilidade e comodidade que este tipo de dispositivo possui, foi pensado no desenvolvimento de um aplicativo para abertura de chamado.

A proposta central é permitir que o usuário abra o chamado e o mesmo seja adaptado para o dispositivo móvel, fazendo com que campos e botões do formulário seja exibido corretamente no dispositivo do usuário. O uso do QR Code seria mantido, seguindo a ideia descrita no projeto de Esteves (2021), a diferença em relação a esse trabalho seria que a exibição dos chamados, seus detalhes e o formulário seriam feitos e exibidos através dos próprios componentes do Android. A plataforma Android foi escolhida pelo fato de ser a mais utilizada no Brasil. De acordo com a StatCounter (2020), a participação de mercado do Android no Brasil foi de 85,4% até setembro de 2020.

O QR Code é um um código de barras que pode ser escaneado utilizando a câmera do celular. O uso do QR Code está presente em diversas áreas, proporcionando ao público

acesso rápido às informações e aos mais diversos tipos de serviços [Ribas 2017]. Através do escaneamento é possível obter informações, ser direcionado para um determinado site ou realizar validações de ingressos de show, por exemplo.

Pesquisas indicam que o uso de QR Code vem aumentando. De acordo com a Mobile Time e Opinion Box, cerca de 35% dos internautas brasileiros com *smartphone* já fizeram pagamentos fotografando o QR Code [Time e Box 2020]. Ainda no ramo de pagamentos, em novembro de 2020 o Banco Central do Brasil lançou o PIX [Banco Central do Brasil 2020], um novo modelo de pagamento eletrônico, que permite efetuar transações por escaneamento de QR Codes, sem a necessidade de informar dados bancários. Portanto, percebe-se que cada vez mais essa tecnologia vem sendo difundida dentro da sociedade, com o objetivo de facilitar ações que necessitam o preenchimento de diversas informações de forma manual.

3. Referencial Teórico

3.1. Arquitetura

A arquitetura é o esqueleto do sistema e, por isso, torna-se o plano de mais alto-nível da construção de cada novo sistema [Krafzig, Banke e Slama 2004]. Definir um padrão arquitetural é uma maneira de estimular a reutilização de componentes e padronizar a estrutura através do uso rotineiro de soluções existentes, por meio de um ponto de referência comum *para* as demais atividades que são executadas posteriormente a sua definição [Bass, Clements e Kazman 2004].

3.2 API

Application Programming Interface ou em português, Interface de Programação de Aplicações, é uma aplicação que permite a conexão de softwares com sistemas providos por terceiros. Com uma API é possível expor serviços ou dados sem a necessidade de implementar métodos e procedimentos [Meng, Steinhardt e Schubert, 2021]. Neste caso, ao invés de um aplicativo prover seu sistema gerenciador de banco de dados para inserir, remover ou editar, a API disponibiliza uma interface que é possível se comunicar com o mesmo. Isso adiciona uma camada de segurança e evita de usuários terem acesso direto a base de dados.

3.3 Kotlin

Linguagem de programação voltada para a plataforma Java. Kotlin é conciso, seguro, pragmático e focado na interoperabilidade com o código Java [Flauzino 2019]. Pode ser utilizado em quase todos os lugares que o Java é usado: para desenvolvimento do lado do servidor e para aplicativos Android. Kotlin funciona bem com todas as bibliotecas Java existentes e *frameworks* e é executado com o mesmo nível de desempenho do Java [Jemerov e Isakova, 2017]. A linguagem Kotlin foi desenvolvida pela JetBrains em 2011 e possui o objetivo de ser mais simples e menos verbosa que o Java [Flauzino 2019]. Para o desenvolvimento de novos aplicativos Android o Google sugere considerar o uso dessa linguagem [Google Developers, 2020].

3.4 Sistema de Entrega e Integração Contínua

Sistema que permite automatizar o processo de testes e publicação da aplicação. De acordo com Humble (2014), Entrega Contínua é a capacidade de se obter mudanças de todos os tipos -incluindo novos recursos, mudanças de configuração, correções de bugs e experiências - em produção ou nas mãos dos usuários, de forma segura, rápida e sustentável [Lima e Vieira, 2018]. Portanto, a entrega contínua tem o objetivo de garantir

que uma nova versão da aplicação contendo novos recursos ou correções esteja disponível para o usuário de forma rápida, mas sem afetar a qualidade do sistema, já que o mesmo foi testado e validado durante a etapa de integração. Em relação a Integração Contínua, de acordo com Fowler (2006), é uma prática de desenvolvimento de software onde os membros de uma equipe integram seu trabalho com frequência, geralmente pelo menos uma vez ao dia. Cada integração é verificada por uma compilação automatizada (incluindo teste) para detectar erros de integração o mais rápido possível.

3.5 Metodologias ágeis

A metodologia ágil é um modelo de processo de planejamento gradual, tornando a construção do software flexível e aproximando o time de desenvolvimento do cliente. De acordo com Soares (2004), as metodologias ágeis para desenvolvimento de software são uma resposta às chamadas metodologias pesadas ou tradicionais. Segundo ele, a ideia é o enfoque nas pessoas e não em processos ou algoritmos. Além disso, existe a preocupação de gastar menos tempo com documentação e mais com a implementação. Uma característica das metodologias ágeis é que elas são adaptativas ao invés de serem preditivas. Com isso, elas se adaptam a novos fatores decorrentes do desenvolvimento do projeto, ao invés de procurar analisar previamente tudo o que pode acontecer no decorrer do desenvolvimento. [Soares 2004]. As ideias acima foram elaboradas por Beck et al. (2001), no quais sintetizaram 4 princípios:

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

3.5 Metodologia Scrum

Metodologia ágil que possui o objetivo de fornecer um processo conveniente para projeto e desenvolvimento orientado a objeto. O Scrum apresenta uma abordagem empírica que aplica algumas ideias da teoria de controle de processos industriais para o desenvolvimento de softwares, reintroduzindo as ideias de flexibilidade, adaptabilidade e produtividade. O foco da metodologia é encontrar uma forma de trabalho dos membros da equipe para produzir o software de forma flexível e em um ambiente em constante mudança [Soares 2004]. De acordo com Stopa e Rachid (2019), o desenvolvimento ocorre através de ciclos de atividades e durante o processo ocorrem reuniões para alinhamento e definição de expectativa entre cliente e o time de desenvolvimento. A metodologia ágil Scrum é fundamentada na teoria de controle de processo e tem por objetivo aperfeiçoar a previsibilidade e controlar os riscos de um projeto [Silva, Souza e 2013]. O Scrum estimula as equipes a aprenderem com as experiências, a se organizarem enquanto resolvem um problema e a refletirem sobre os êxitos e fracassos para melhorarem sempre [Atlassian 2022]. Dentro do Scrum existem alguns conceitos tais como:

- Tarefa: As tarefas são a decomposição de uma estória, indicando como a estória será concluída [Yeoh 2021]. As tarefas são divididas em pequenas unidades incrementais de trabalho, sendo geralmente feitas por uma pessoa da equipe e concluídas em um dia [Pivac 2019].
- Estória: Explicação informal e geral sobre um recurso de software escrita a partir da perspectiva do usuário final. Seu objetivo é articular como um recurso de software pode gerar valor para o cliente. [Atlassian 2022];

- Épico: Um épico é uma grande quantidade de trabalho que pode ser dividida em uma série de estórias menores. [Atlassian 2022];
- *Sprint*: Um *sprint* é um período curto e fixo em que uma equipe Scrum trabalha para concluir uma quantidade definida de trabalho. Os sprints estão no cerne das metodologias Scrum e ágil [Atlassian 2022];
- *Backlog*: O backlog do produto é uma lista de trabalho que a equipe de desenvolvimento deve realizar organizada em prioridades. Ela vem do roteiro do produto e seus requisitos. Os itens mais importantes são mostrados na parte superior do backlog do produto para que a equipe saiba o que fazer primeiro. [Atlassian 2022];
- *Sprint Review*: Demonstração do trabalho de toda a equipe: *designers*, desenvolvedores e o cliente. Os membros da equipe se reúnem para demonstrações informais e descrevem o trabalho que fizeram para cada iteração. É o momento de fazer perguntas, testar novos recursos e dar feedback. [Atlassian 2022];

4. Materiais e Métodos

Nesta Seção serão descritos os materiais e métodos utilizados durante o desenvolvimento deste projeto.

Para o desenvolvimento eficiente e correto do projeto, foi utilizada uma ferramenta de gerenciamento de projeto com o objetivo de definir prazos de entregas para pesquisa, desenvolvimento do *design*, e implementação das funcionalidades que o aplicativo deveria possuir. A ferramenta utilizada foi o Jira, na qual foi desenvolvida pela empresa Atlassian [Jira 2020]. O Jira permite construir quadros divididos em etapas, permitindo rastrear em que etapa uma determinada tarefa se encontra (Figura 1). No projeto foi utilizado as seguintes etapas:

- A fazer: a tarefa está pronta para ser iniciada
- Em desenvolvimento: a tarefa está sendo desenvolvida
- Pronto para reteste: utilizado para quando haver *bugs*, sendo necessário retestar a funcionalidade após a correção do problema.
- Finalizada: A tarefa foi finalizada

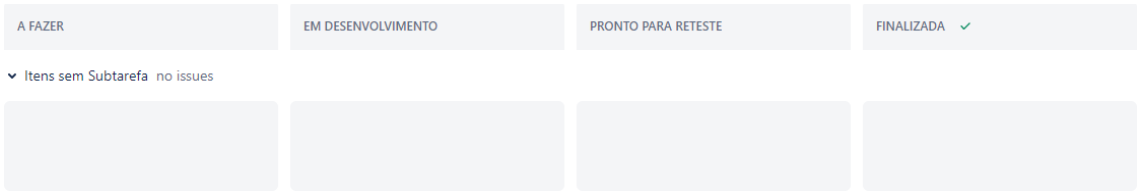


Figura 1. Board presente no Jira

O projeto utilizou o conceito de tarefa, estória, épico, sprint e backlog. Não foi adotado por completo todos os conceitos da metodologia Scrum, pois o aplicativo foi desenvolvido somente por uma pessoa.

O aplicativo foi desenvolvido para *smartphones* que rodem o sistema operacional Android [Google 2020a]. Para desenvolver o código foi utilizada a ferramenta Android Studio [Google 2020b] e a linguagem de programação Kotlin [Kotlin 2021]. Para testar o aplicativo foi utilizado a ferramenta de emulação disponibilizado dentro do Android

Studio e também o uso de um *smartphone* físico. O sistema operacional para desenvolver a aplicação foi o MacOS.

O projeto também realizou testes para garantir a integridade do aplicativo. Com os testes é possível verificar se a lógica da funcionalidade está funcionando. O projeto fez uso de dois tipos de testes, teste unitário e testes exploratórios. O teste unitário visa testar uma pequena funcionalidade da aplicação de forma isolada, verificando se a mesma está fazendo o que foi proposto [Bernardo e Kon 2008]. Este tipo de teste permite que o desenvolvedor identifique um problema durante a criação de uma determinada função, evitando falhas ou problemas no futuro. A biblioteca utilizada nesse teste foi a MockK [MockK 2020]. Já os testes exploratórios são testes manuais executados pelo desenvolvedor. O objetivo é testar o aplicativo rodando no emulador ou dispositivo físico.

Sobre a integração contínua, para fazer o uso desse conceito foi utilizado o Jenkins [Jenkins, 2022], um sistema web que permite testar o aplicativo após um código ir para o repositório. Além disso permite gerar e salvar versões do aplicativo para quando uma nova funcionalidade é implementada.

Para a hospedagem do código desenvolvido foi utilizada a ferramenta GitHub [GitHub 2020]. Ela é uma ferramenta que permite salvar, gerenciar e criar controles de versão do código. Todo o código desenvolvido fica guardado em um repositório. Outro sistema de hospedagem utilizado foi o sistema JFrog Artifactory [JFrog, 2022]. O objetivo desse sistema é armazenar as versões do aplicativo que o Jenkins gerou durante o desenvolvimento, criando uma espécie de histórico de versões do aplicativo.

Para se comunicar com o GLPI utilizado pelo IFSP foi utilizado a biblioteca Retrofit [Square 2020]. Ela é uma ferramenta que permite realizar comunicação com servidores e APIs [Egestor 2020] externas da aplicação Android.

5. Desenvolvimento

Nesta Seção será descrito o desenvolvimento do trabalho realizado. A primeira atividade foi entender o que já existia sobre o sistema e o que agregaria valor ao realizar a leitura por QRCode. Depois de entendido o contexto do sistema, foi realizada uma prova de conceito, em inglês *Proof of Concept* (POC) que é uma versão simples da aplicação, na qual não leva em consideração performance, arquitetura e estilização, o objetivo é verificar se a solução é tangível de se concretizar.

A ideia foi desenvolver um aplicativo que realizasse o intermédio entre o usuário e o sistema GLPI, para isso foi utilizada a API que o GLPI disponibiliza. O aplicativo coleta as informações que o usuário preencheu e envia as mesmas para o GLPI. O processo de geração dos formulários também é feito através das informações que o GLPI retorna. O aplicativo identifica qual formulário foi requisitado e solicita o mesmo. O retorno contem informações sobre o formulário, por exemplo, os campos que o mesmo possui, o tipo de campo, e se esse campo possui alguma regra que pode afetar a visibilidade de outros campos do formulário. Sobre a visibilidade, existem campos que de acordo com que o usuário digita ou seleciona, pode fazer com que outro campo do formulário aparece ou desapareça. Com as informações obtidas, o aplicativo constrói o formulário utilizando os componentes nativos do Android.

O desenvolvimento do aplicativo foi realizado em etapas. Primeiramente foi desenvolvida uma POC que tinha o objetivo de verificar se a comunicação com o GLPI iria funcionar, isto é, se seria possível realizar a comunicação com a API da plataforma, realizar a criação e a listagem de chamados. Estas três funções funcionaram perfeitamente

e diante desse fato foi possível avançar no desenvolvimento. A Figura 2 exemplifica a tela de login implementada durante a realização da POC.

Figura 2. Tela de login da POC

Para o desenvolvimento da versão final, foi criado o *design* das telas para poder validar o fluxo e o *layout* da aplicação. A ferramenta utilizada para construir o *layout* foi o Adobe XD.

Toda a validação foi realizada com o servidor que trabalha na CTI (Coordenadoria de Tecnologia de Informação) do Câmpus e também responsável por desenvolver a primeira solução de abertura de chamado através de QR Code. As validações feitas fazem parte dos princípios da metodologia ágil na qual o Scrum está inserido. Essas validações simulava a *Sprint Review* para garantir que o que estava sendo desenvolvido era o esperado pelo usuário da aplicação. Durante as validações foram verificadas se as informações e o fluxo do aplicativo estavam corretos.

Depois de validar o *layout*, as funcionalidades a serem desenvolvidas foram mapeadas em estórias, sendo que cada estória iria conter diversas *tasks*. Estórias que fazem parte de um mesmo fluxo ou que possuem semelhanças durante o desenvolvimento foram agrupadas em épicos. Com as estórias e épicos divididos, foi feito o cronograma de entrega das tarefas. Na Figura 3 é possível ver o cronograma dividido em trimestres. Os ícones em roxo são os épicos, conforme dito, eles abrigam um conjunto de estórias ou *bugs*.

Figura 3. Cronograma trimestral das entregas

O objetivo do desenvolvimento era que cada Sprint gerasse uma nova funcionalidade para o aplicativo. Primeiramente foi desenvolvida a manipulação da sessão do usuário. A ideia era poder realizar o login e o logout no GLPI através do aplicativo. As funcionalidades de login e logout estavam inseridas no épico “Manipulação da sessão do usuário”, divididas em duas estórias, “Implementação do Login” e “Implementação do Logout”. A Figura 4 exibe a tela de login, na qual possibilita o usuário preencher seu usuário e senha do GLPI. Caso as credenciais sejam válidas, o aplicativo redireciona o usuário para a tela com todos os chamados do usuário.

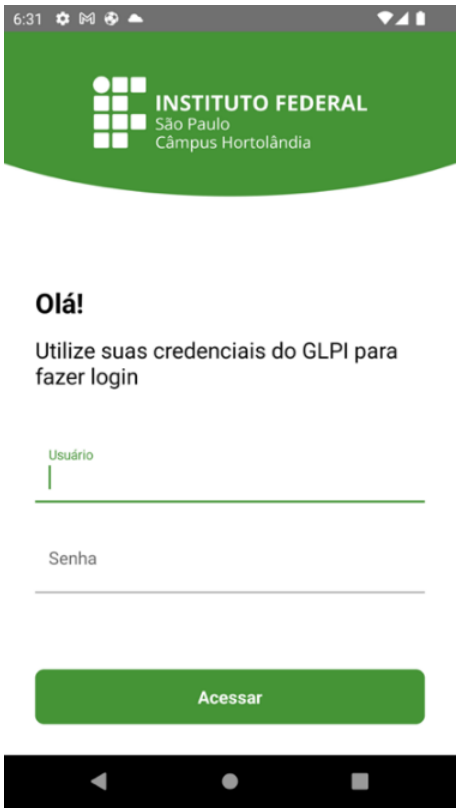


Figura 4. Tela de login do aplicativo

Vale ressaltar que o aplicativo não gerencia informações relacionadas ao cadastro do usuário, somente manipula as informações contidas no GLPI através do consumo de sua API. Para o login, por exemplo, é feita uma requisição com base no usuário e senha, o retorno é um *token* utilizado para realizar outras requisições dentro do aplicativo, caso o *token* seja válido, as requisições serão válidas. A Figura 5 mostra um exemplo de requisição de login.

Depois de realizar a função de login, foi criada a funcionalidade para permitir a exibição dos chamados. A construção foi feita através de duas estórias desenvolvidas ao longo de duas Sprints. O épico responsável por conter o desenvolvimento da exibição dos chamados foi o “Criação do chamado”. A primeira estória desenvolvida nessa Sprint foi a “Exibição na home”, o objetivo foi construir a lista que exibe de forma geral todos os chamados criados pelo usuário (Figura 6a). A segunda estória era a “Exibir detalhes do chamado”, com o intuito de exibir os detalhes de um chamado que o usuário selecionou na lista de chamados (Figura 6b).

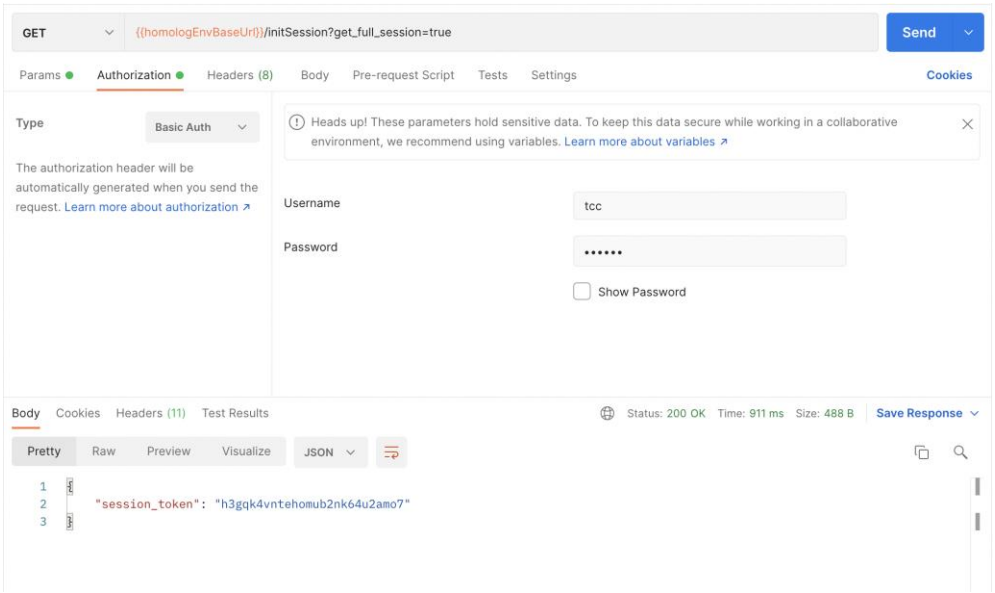
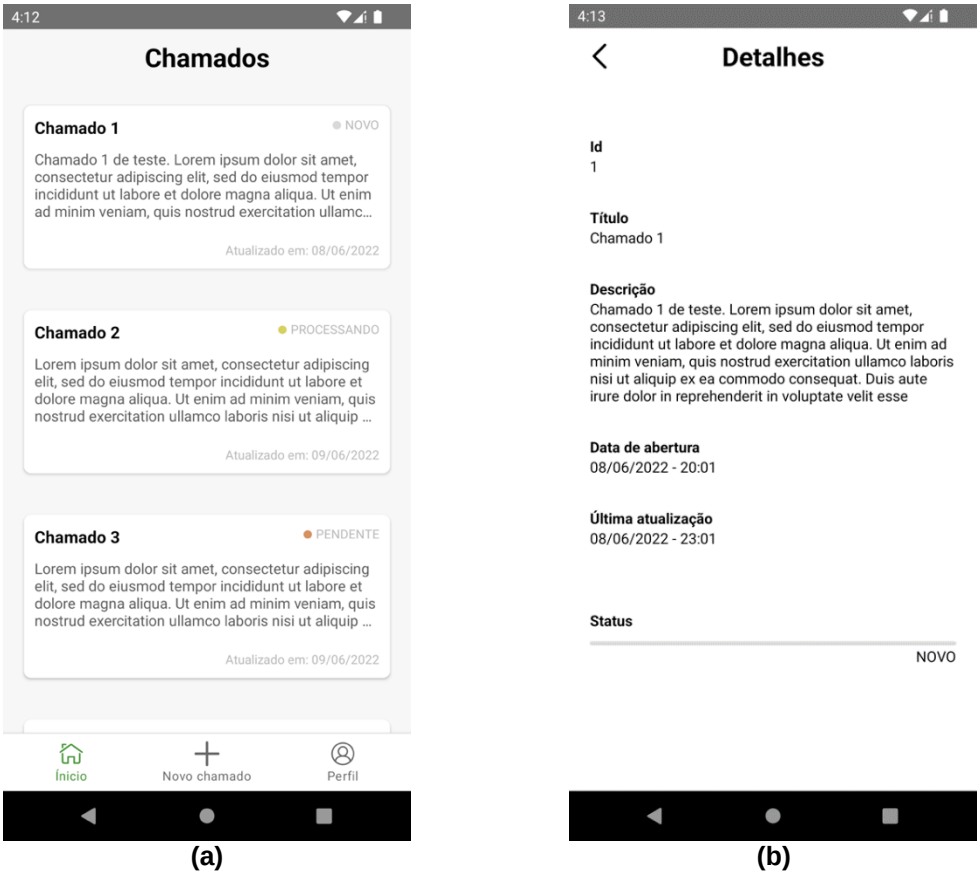


Figura 5. Exemplo de requisição para o caso de uso Fazer Login



(a) **(b)**

Figura 6. Tela para exibição dos chamados

Após ter desenvolvido as funcionalidades de realizar login e exibição dos chamados criados, a próxima funcionalidade a ser desenvolvida seria a criação do chamado direto pelo aplicativo. Essa Sprint foi a parte mais complexa do projeto, pois para realizar a criação do chamado através do aplicativo e escanear QRCodes, os formulários precisam ser gerados de forma dinâmica. Seria inviável construir

manualmente os formulários dentro do aplicativo, pois caso o mesmo sofra alguma modificação seria necessário atualizar o código do aplicativo.

Os formulários do GLPI possuem campos que são capazes de alterar o comportamento do formulário, por exemplo, de acordo com a opção selecionada em um campo, outro campo poderia aparecer ou desaparecer, ou o conteúdo dentro de um campo poderia ter seu valor modificado. Isso significa que os campos precisavam ser gerados de forma dinâmica e as regras que eles contêm precisavam ter esse mesmo comportamento. Essa dinâmica foi outra dificuldade para implementação.

A API do GLPI também não é amigável e não conta com uma documentação bem descritiva. Por isso, para desenvolver a criação do formulário foi necessário realizar pesquisas na internet e no fórum do próprio GLPI. Para verificar a viabilidade de gerar os formulários, foi criada outra POC para validar o escaneamento dos QRCodes (Figura 7) e a geração dos formulários (Figura 8).

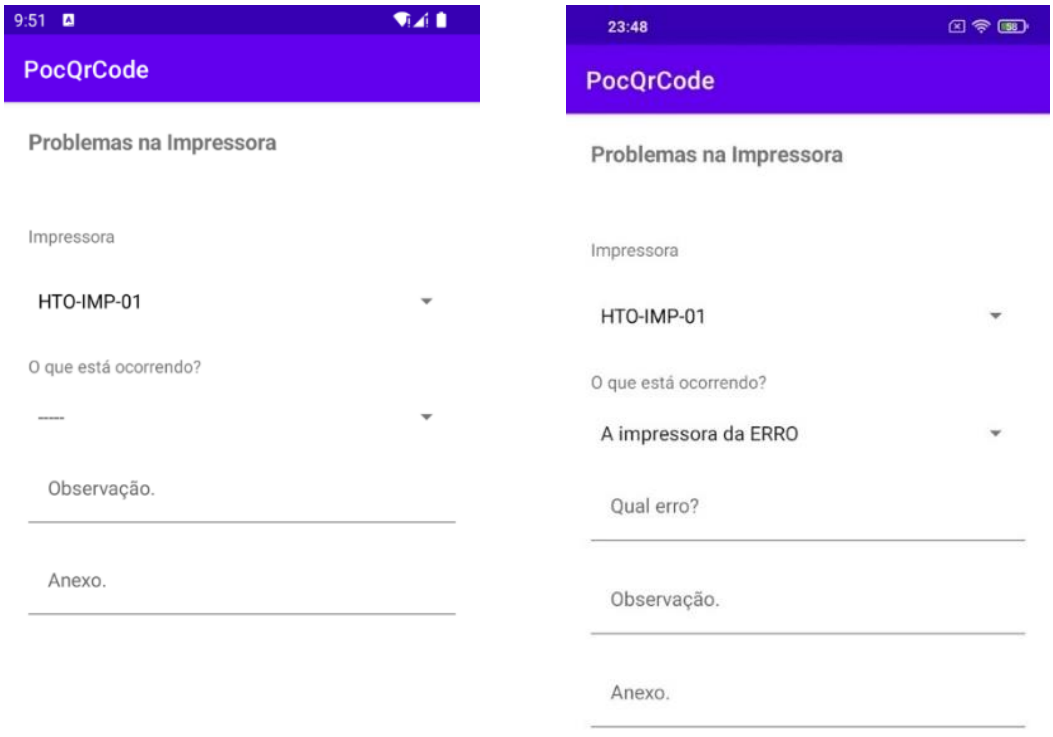
Figura 7. Formulário Aberto através do escaneamento do QRCode

A Figura 7 exibe a tela de leitura de QR Code da POC construída para validar a exibição do formulário. O objetivo da tela é escanear o código e redirecionar o usuário para a tela que exibe o formulário que o usuário irá precisar preencher para abrir o chamado. O QR Code contém o *link* do formulário, ao escanear, o aplicativo obtém o id do formulário presente no *link* e solicita para a API do GLPI o formulário. Com as informações do formulário retornado, o aplicativo consome os dados e constrói o formulário usando os componentes nativos do Android.

A Figura 8a contém a tela a ser exibida após o escaneamento, repare que no exemplo, o formulário possui 4 campos. Dependendo do valor selecionado no campo “O que está ocorrendo?”, um novo campo pode ser exibido. Na Figura 8b é possível ver

justamente esse comportamento, ao selecionar a opção “A impressora da ERRO” no campo citado anteriormente, um novo campo chamado “Qual é o erro” é exibido.

Com a POC observou-se que é possível lidar com a API do GLPI e gerar os formulários de forma dinâmica, incluindo a ocultação e exibição de formulários de acordo com as respostas do usuário. Toda a lógica de ocultação e exibição ficou por conta do aplicativo no qual identifica na API qual o tipo de lógica que está sendo utilizado no campo. Exemplificando essa situação, se a API indica que o campo de texto nome deve desaparecer quando um botão de seleção chamado “já tenho nome” for marcado, então o sistema é capaz de identificar que quando o campo de seleção for marcado, deve-se procurar pelo campo de nome texto e quando o mesmo for achado, deve-se ocultá-lo. Conforme dito anteriormente, está etapa foi a parte principal do projeto e a mais desafiadora, demandando muito tempo de estudo e implementação do código, pois foi necessário mapear as lógicas que a API poderia disponibilizar, e implementá-las dentro do aplicativo, garantindo que novos formulários possam ser criados, sem precisar alterar código da aplicação.



(a) **(b)**

Figura 8. Telas após escanear o QR Code

Com a POC desenvolvida e validada, foi iniciado o desenvolvimento das telas de formulário seguindo o *layout* proposto no AdobeXD. Foram criadas duas estórias para esse desenvolvimento, ambas ficaram contidas dentro do épico “Criação do chamado”. A primeira estória tinha o objetivo de implementar a leitura do QR Code (Figura 9a) e a segunda tinha o objetivo de implementar a geração, exibição e envio do formulário de abertura de chamado. A Figura 9b mostra a tela exibida após a criação de um chamado com sucesso. A Figura 9c apresenta uma tela de alerta ao usuário informando que houve uma falha ao abrir o chamado.

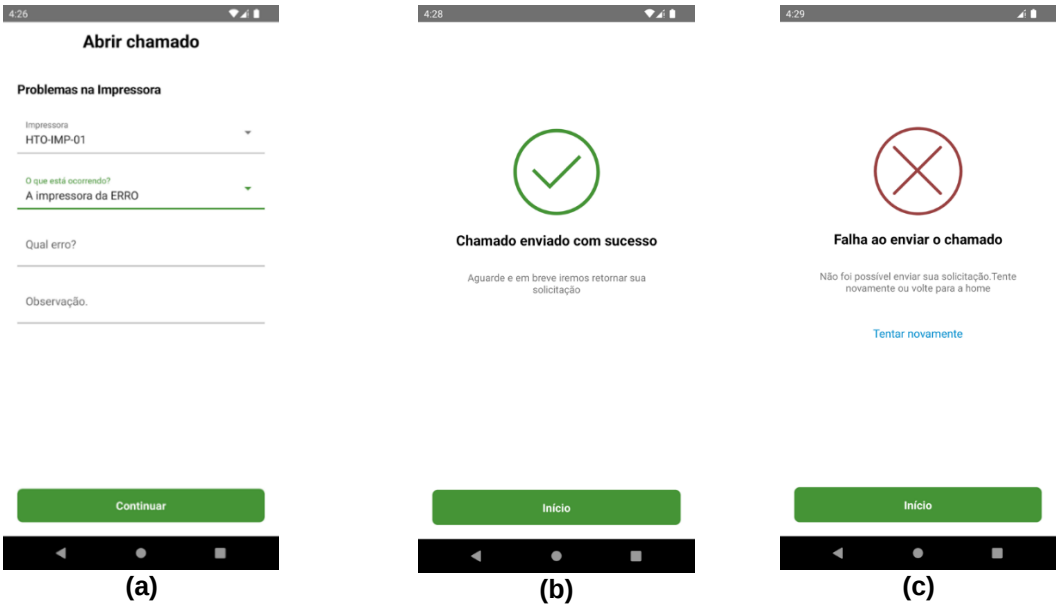


Figura 9. Telas de abertura de chamado

Com as telas desenvolvidas, o épico foi concluído e a codificação do aplicativo também foi finalizada. Para toda tarefa que possuísse alguma lógica foi executado um teste para verificar se a mesma estava correta. A ideia do teste era validar as funcionalidades de forma isolada e automática, sem precisar executar o aplicativo. Esses testes foram executados em uma base local com dados fictícios. No total foram desenvolvidos 51 testes unitários separados de acordo com seu cenário (Tabela 1).

Tabela 1. Quantidade de testes unitários desenvolvidos por cenário

Cenário de teste	**Quantidade**
Obter formulário e abertura de chamado	18
Login	17
Logout	4
Exibição dos chamados abertos	8
Lidar com a sessão do usuário após o login	4

Após finalizar a codificação, iniciou-se testes do aplicativo utilizando uma base que fosse parecida com a base GLPI em produção no câmpus. A mesma possuía apenas caráter de testes e não continha nenhuma informação sobre discente e servidores, apenas formulários com regras parecidas com as existentes na base em produção. O suporte para o uso da mesma foi oferecido pelo servidor da CTI. Todas as telas e funcionalidades foram testadas funcionalmente para garantir o seu comportamento.

Outro ponto do desenvolvimento a ser mencionado é a entrega de código. A cada tarefa concluída, um trecho de código era enviado para um repositório remoto, o GitHub. Porém, durante o envio do código era executado o Jenkins, um sistema para verificar se o aplicativo estava compilando corretamente, se o código seguia padrões de desenvolvimento para a linguagem Kotlin e o sistema Android, verificava se os testes unitários estavam executando com sucesso e ao final gerava uma publicação do aplicativo caso todos os passos anteriores não contivessem erros. A hospedagem das versões do aplicativo ficou no site chamado JFrog.

Neste projeto, o Jenkins foi baixado e configurado para rodar localmente na máquina em que o projeto foi desenvolvido. Foi preciso adicionar um arquivo no aplicativo chamado Jenkinsfile (Figura 10), este arquivo era o responsável por realizar as etapas de execução e validação do projeto. Os *stages* são os estágios de execução da integração contínua, cada *stage* pode conter diversas etapas que podem ser executadas sequencialmente ou de forma paralela por meio do comando *parallel*, linha 48 da Figura 10. Na Figura 10, linha 33, é possível ver instruções para compilar o código, verificando se não há nenhum erro em termos de codificação, depois (linha 40) é realizado a *build* para verificar se o aplicativo é capaz de ser instalado e no *Stage Static Analysis* (linha 47) é feito o processo de verificar se o código está seguindo os padrões recomendados da Google e da linguagem Kotlin, as 3 etapas desse estágio são executadas de forma independente e paralela. Se alguma etapa falhar, o processo de execução é encerrado, sendo necessário corrigir a falha e executar novamente o Jenkins.

Além disso, toda entrega deveria ser feita em uma *branch* que tinha o prefixo feature/. Exemplo, se a implementação fosse para realizar o sistema de login, a *branch* deveria conter o nome feature/login. Todo código enviado nesse tipo de *branch* tem uma versão do aplicativo hospedada no JFrog localizada no caminho feature/nome_do_recurso_a_ser_desenvolvido.

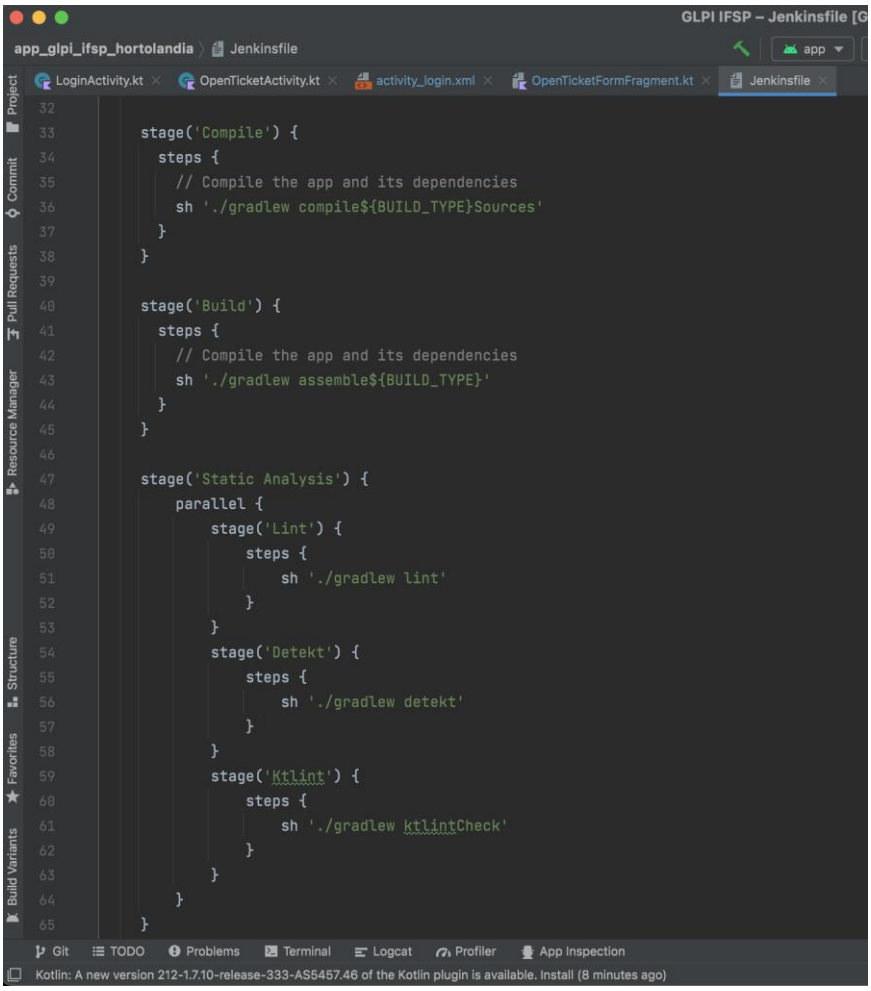


Figura 10. Trecho de código do arquivo de configuração para executar o aplicativo no Jenkins

O projeto também possui um padrão para envio de correção de *bugs*, chamado bugfix/descrição_do_bug_corrigido. Versões do aplicativo para correção de *bugs*

também são armazenadas no JFrog. Outras 2 *branchs* foram utilizadas no projeto, *main* e *develop*, a *main* contém todo o código do aplicativo testado e que já está funcionando normalmente, a *develop* contém o código de uma implementação que ainda está em andamento. No JFrog, existem pastas para armazenar versões do código que estão na *main* e *develop*, cada uma dessas pastas contém outras pastas que identificam as versões do aplicativo, exemplo, 1.0.0 e 1.0.1. Essas versões são configuradas na aplicação e quando enviadas para publicação, o JFrog identifica se a versão existe, caso sim, subscreve o aplicativo atual armazenado pelo novo, caso a versão ainda não exista, uma pasta com o número da versão é criada e o aplicativo é armazenado. A Figura 11 demonstra todas as etapas do sistema de integração e entrega contínua com status de sucesso. As etapas mostradas na Figura 11 são explicadas a seguir:

- *Detect build type*: verifica em qual *branch* o código está sendo executado, na *main*, *develop* o outra *branch*;
- *Compile*: verifica se o aplicativo não possui código que está quebrando;
- *Build*: verifica se o aplicativo consegue ser executado;
- *Static Analysis*: realiza uma análise do código para checar se segue os padrões do Android e da linguagem Kotlin;
    - *Detekt*: verifica questões relacionadas com o tamanho de método, classe e quantidade de parâmetros em uma classe. Em resumo, lida em verificar quão complexo um código está ficando;
    - *Ktlint*: verifica se está sendo utilizado corretamente os itens da linguagem Kotlin;
    - *Lint*: verifica se está seguindo os padrões definidos para o desenvolvimento Android;
- *Tests*: executa os testes unitários e verifica se todos estão passando com o status de sucesso.
- *Publish*: gera uma versão do aplicativo no JFrog.

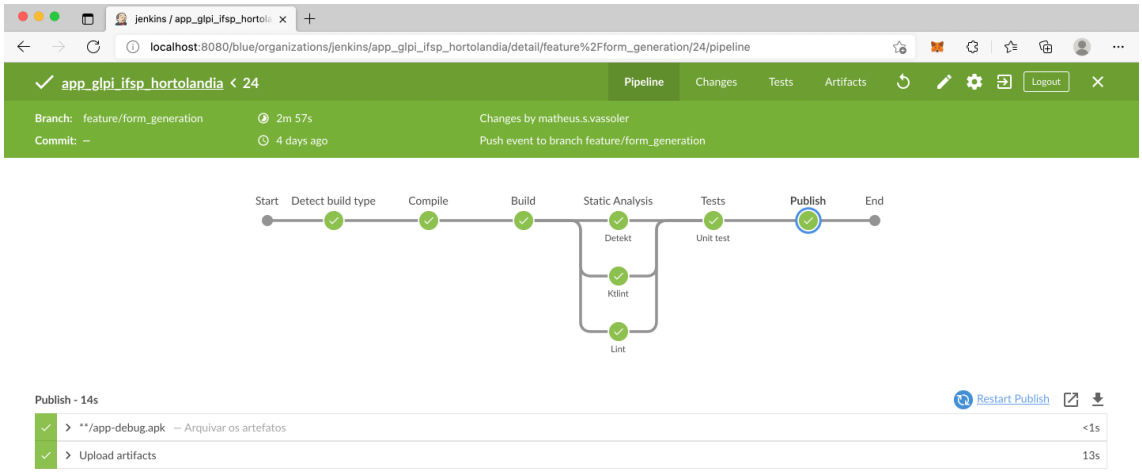


Figura 11. Exibição das etapas do sistema de Integração e Entrega Contínua

A Figura 12 mostra o armazenamento das publicações do aplicativo no JFrog. A seguir, resumo da explicação sobre as pastas do projeto:

- glpi-ifsp-hto-app: pasta onde fica armazenada as versões do aplicativo;
    - *debug*: pasta onde fica as versões do aplicativo antes de estar realmente pronto para uso;
        - PRs: onde fica as pastas que contém os códigos que são enviados para o GitHub e que antecedem a ida para a branch *develop*;
            - *Bugfix*: Armazena as versões onde há correção de bugs;
            - *Feature*: Armazena as versões que contém implementação de novos recursos;
        - *develop*: armazena as versões do aplicativo que contém um conjunto de recursos que estão em desenvolvimento e que em breve deverão ser disponibilizados para uso. As versões aqui são identificadas por pastas e esse controle é feito dentro de um arquivo localizado na aplicação;
    - *release*: não disponível na imagem, mas é a pasta onde ficam as versões que já foram disponibilizadas para os usuários utilizarem. As versões são identificadas por pastas e esse controle é feito dentro de um arquivo localizado na aplicação.

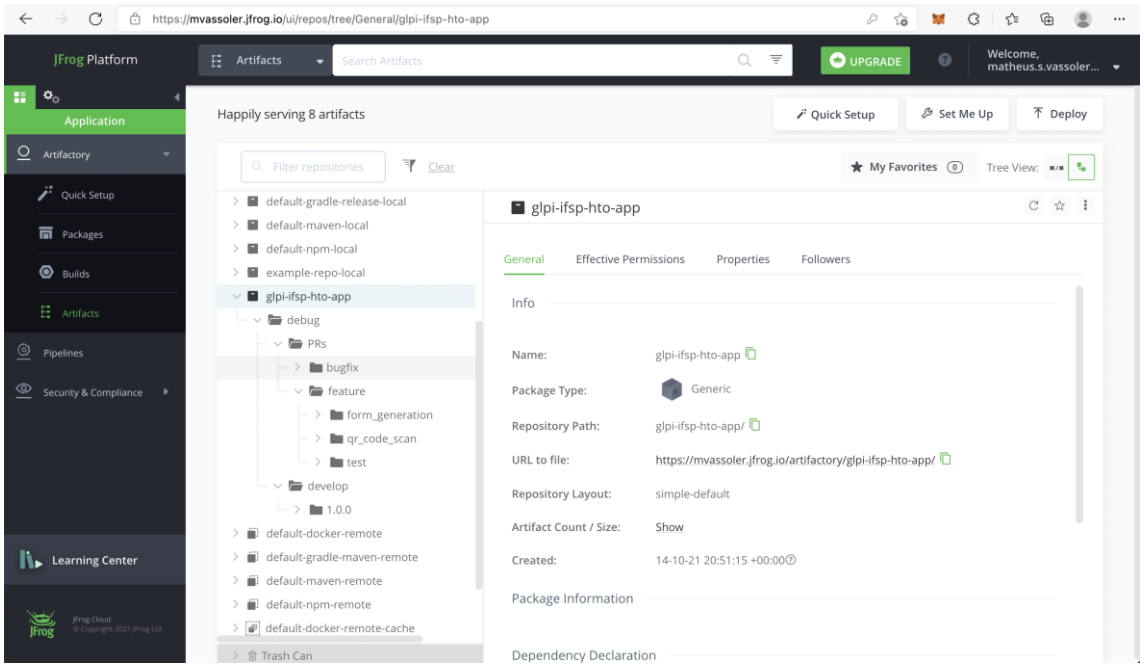


Figura 12. Exibição das publicações do app no JFrog

7. Conclusão

O mundo está cada vez mais dinâmico e ágil, exigindo que tarefas complexas se tornem mais simples. Com este pensamento em mente, este trabalho apresentou uma nova solução na forma que os chamados de problemas são abertos dentro do IFSP Campus Hortolândia. Permitindo a abertura de chamados por meio de um aplicativo Android que possibilita escanear o QR Code para o preenchimento das informações relativo ao problema enfrentado. Os códigos de barras (QR Code) utilizados são os mesmos desenvolvidos e apresentados no trabalho de Esteves (2021). Os QR Code estarão

espalhados por diversos pontos do Instituto, nos quais possam necessitar de algum suporte por parte do departamento de TI.

O aplicativo Android foi construído utilizando a linguagem de programação Kotlin. Além disso, para garantir o funcionamento correto da aplicação, testes unitários foram feitos.

Para seguir o cronograma e ter um desenvolvimento organizado, utilizou-se uma ferramenta de gerenciamento de projeto, permitindo rastrear as etapas das tarefas e as entregas a serem feitas ou que já foram concluídas.

Como aprendizado foi possível entender o que é um sistema de Integração e Entrega Contínua e também implementá-lo dentro do projeto, permitindo que o aplicativo fosse constantemente testado e que o histórico de versões pudesse ser armazenado.

Caso o discente ou servidor tenha problema com algum patrimônio, ele pode abrir o chamado através das três opções disponíveis: (i) procurando pelo formulário, ou (ii) escanear o QR Code para abrir diretamente o formulário web ou (iii) utilizar o aplicativo para escanear o QR Code e abrir o formulário dentro da própria aplicação. Utilizando o aplicativo, é necessário que o mesmo seja instalado no celular para então acessar a opção de abertura de chamados, apontar a câmera para o QRCode, preencher o formulário carregado e enviar o mesmo para a CTI averiguar o problema.

As disciplinas cursadas e que ajudaram no desenvolvimento deste trabalho foram: Engenharia de Software, Gestão de Projetos, Qualidade de Software e disciplinas que ensinaram lógica e linguagem de programação.

Referências

Atlassian (2022). Disponível em: https://www.atlassian.com/br/agile/scrum. Último acesso: 16 set. 2022.

Banco Central do Brasil (2020). Pix. Disponível em: https://www.bcb.gov.br/estabilidadefinanceira/pagamentosinstantaneos/. Último acesso: 26 set. 2020

Bass, L., Clements, P. e Kazman R. (2004). Software Architeture in Practice. 2nd ed. Boston- MA: Addison-Wesley, 2004. Último acesso: 30 set. 2020.

Beck, K. et al. (2001). Manifesto para Desenvolvimento Ágil de Software. Disponível em: http://agilemanifesto.org/iso/ptbr/manifesto.html. Último acesso: 16 set. 2022.

Bernardo, P., Kon, F. (2008). A importância dos testes automatizados. Engenharia de Software Magazine. v.1, n.3, p.54-57, 2008 Último acesso: 16 set. 2022.

Cosentino, L. (2006). Aspectos evolutivos da interação homem-máquina: tecnologia, computador e evolução humana. Psicologia & Informática: produções do III Psicoinfo II Jornada do NPPI. v.1, p.61-71, 2006. Último acesso: 16 set. 2022.

Egestor (2020). API ou Interface de Programação de Aplicativos: o que é e como funciona?. Disponível em: https://blog.egestor.com.br/api-o-que-e-e-como-funciona/. Último acesso: 01 out. 2020

Esteves, R. dos S. (2021). Manutenção do GLPI para Facilitar Abertura de Chamados no IFSP Câmpus Hortolândia. Disponível em: https://hto.ifsp.edu.br/portal/index.php/ensino/cursos/ads/ads-tcc

FGV (2020). Disponível em: https://portal.fgv.br/noticias/brasil-tem-424-milhoes-dispositivos-digitais-uso-revela-31a-pesquisa-anual-fgvcia/. Último acesso: 26 set. 2020.

Flauzino, M. (2019). Are you still smelling it? A comparative between Java and Kotlin. Disponível em: https://medium.com/@matheusflauzino/are-you-still-smelling-it-a-comparative-between-java-and-kotlin-a7372ecddb19. Último acesso em: 16 set. 2022.

Fowler, M. (2006). Disponível em: https://www.martinfowler.com/articles/continuousIntegration.html. Último acesso: 29 mar. 2022

GLPI (2021). Disponível em: http://www.glpibrasil.com.br/o-que-e-glpi/. Último acesso: 24 fev. 2021.

GitHub (2020). Disponível em: https://github.com/. Último acesso: 01 out. 2020

Google (2020a). Android. Disponível em: https://www.android.com/. Último acesso: 28 set. 2020.

Google (2020b). Android Studio. Disponível em: https://developer.android.com/studio/. Último acesso: 28 set. 2020.

Google (2020c). Espresso. Disponível em: https://developer.android.com/training/testing/espresso/. Último acesso: 28 set. 2020.

Google Developers (2020). Uso do Kotlin para aplicativos Android. Disponível em: https://developers-br.googleblog.com/2020/11/devo-aprender-usar-o-kotlin-para.html. Último acesso: 29 mar. 2022

Jenkins (2022). Disponível em: https://www.jenkins.io/. Último acesso: 29 mar. 2022.

Jemerov, D. e Isakova, S. (2017). Kotlin in Action. 1st ed. Shelter Island- NY: Manning, 2017. Último acesso: 29 set. 2020.

Jira (2020). Disponível em: https://www.atlassian.com/br/software/jira/. Último acesso: 28 set. 2020.

JFrog Artifactory (2022). Disponível em https://jfrog.com/. Último acesso: 29 mar. 2022.

Kotlin (2021). Disponível em: https://kotlinlang.org/. Último acesso: 24 fev. 2021.

Krafzig, D., Banke, K. e Slama D. (2004). Enterprise SOA, Service-Oriented Architeture: Best Pratices. 1st ed. Indianapolis- IN: Prentice Hall, 2004. Último acesso: 29 set. 2020.

Lima, M., Vieira, R. (2018). Continuous Delivery -Agilidade Entre Desenvolvimento e Entrega de Valor. v.4, n.2, p.68-85, 2018. Último acesso: 29 mar. 2022.

Meng, M., Steinhardt, S., Schubert A. (2018). Application Programming Interface Documentation: What Do Software Developers Want? Journal of Technical Writing and Communication. v48. Último acesso: 25 fev. 2021.

Microsoft (2020). Windows. Disponível em: https://www.microsoft.com/pt-br/windows/. Último acesso: 28 set. 2020.

Milvus (2019). Disponível em: https://milvus.com.br/sistema-de-chamados/. Último acesso: 26 set. 2020.

MockK (2020). Disponível em: https://mockk.io/. Último acesso: 29 set. 2020.

Pivac, L. (2019). Composing Meaningful Tasks. Disponível em: https://medium.com/agile-adapt/composing-meaningful-tasks-c1ca51064c1a. Último acesso em: 26 set. 2022.

Retrofit (2020). Square. Disponível em: https://square.github.io/retrofit/. Último acesso: 01 out.2020

Ribas, A. et al. (2017). O uso do aplicativo QR Code como recurso pedagógico no processo de ensino e aprendizagem. Ensaios Pedagógicos. v.7, n.2, p.12-21, jul/dez. 2017. Disponível em: http://www.opet.com.br/faculdade/revista-pedagogia/pdf/n14/n14-artigo-2-O-USO-DO-APLICATIVO-QR-CODE.pdf/. Último acesso: 26 set.2020.

Silva, D., Souza, I. e Camargo, T. (2013). Metodologias ágeis para o desenvolvimento de software: Aplicação e o uso da metodologia Scrum em contrates ao modelo tradicional de gerenciamento de projetos. Revista Computação Aplicada. v.2, n.1, 2013. Último acesso: 28 set. 2020.

Soares, M. (2004). Metodologias Ágeis Extreme Programming e Scrum para o Desenvolvimento de Software. Revista Eletrônica de Sistemas de Informação. v.3, n.1, p.1-8, 2004. Último acesso: 16 set. 2022.

Souto, T. (2018). Arquiteturas em Android: MVVM + Kotlin + Retrofit Parte 1. Disponível em: https://medium.com/@othiagosouto/arquiteturas-em-android-mvvm-kotlin-retrofit-parte-1-2ac77c8a26/. Último acesso: 01 out. 2020.

StatCounter (2020). Disponível em: https://gs.statcounter.com/os-market-share/mobile/brazil/. Último acesso: 28 set. 2020

Stopa, G., Rachid, C. (2019). Scrum: Metodologia ágil como ferramenta de gerenciamento de projetos. CES Revista. v.33, n.1, p.302-323, 2019 Último acesso: 16 set. 2022.

Time, M. e Box, O. (2020). Panorama Mobile Time/Opinion Box sobre comércio móvel no Brasil. Disponível em: https://www.mobiletime.com.br/noticias/29/04/2020/qr-code-supera-nfc-em-pagamentos-no-brasil/. Último acesso: 26 set. 2020.

Yeoh, J. (2021). Example for Epic, User Story, Task and Bug. Disponível em: https://medium.com/@Jia_Le_Yeoh/example-for-epic-user-story-task-and-bug-7d7b9d4b0931. Último acesso em: 26 set. 2022.

Zarelli, G. (2020). Descomplicando a Clean Architecture. Disponível em: https://medium.com/luizalabs/descomplicando-a-clean-architecture-cf4dfc4a1ac6/. Último acesso em: 01 out. 2020.

Documento Digitalizado Público

Anexo I - Entrega da versão final do artigo referente ao TCC

Assunto: Anexo I - Entrega da versão final do artigo referente ao TCC
Assinado por: Daniela Marques
Tipo do Documento: Projeto
Situação: Finalizado
Nível de Acesso: Público
Tipo do Conferência: Documento Digital

Documento assinado eletronicamente por:

- **Daniela Marques**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 26/09/2022 20:39:00.

Este documento foi armazenado no SUAP em 26/09/2022. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifsp.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 1111853
Código de Autenticação: e9fd8a324f