UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

Murilo Vicentin Giordano

Desenvolvimento de um aplicativo mobile para aplicação no mercado de segurança do trabalho

São Carlos
2020

Murilo Vicentin Giordano

Desenvolvimento de um aplicativo mobile para aplicação no mercado de segurança do trabalho

Monografia apresentada ao Curso de Engenharia Mecatrônica, da Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Engenheiro Mecatrônico.

Orientador: Prof. Dr. Maíra Martins da Silva

VERSÃO CORRIGIDA

São Carlos

2020

AUTORIZO A REPRODUÇÃO TOTAL OU PARCIAL DESTE TRABALHO, POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

Vd

Vicentin Giordano, Murilo
Desenvolvimento de um aplicativo mobile para aplicação no mercado de segurança do trabalho / Murilo Vicentin Giordano; orientadora Maíra Martins da Silva. São Carlos, 2020.

Monografia (Graduação em Engenharia Mecatrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2020.

1. aplicativo mobile. 2. automatização. 3. segurança do trabalho. 4. Flutter. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE AVALIAÇÃO

Candidato: Murilo Vicentin Giordano

Título: Desenvolvimento de um aplicativo mobile para aplicação no mercado de segurança do trabalho

Trabalho de Conclusão de Curso apresentado à Escola de Engenharia de São Carlos da Universidade de São Paulo Curso de Engenharia Mecatrônica.

BANCA EXAMINADORA

Professora Associada Maíra Martins da Silva (Orientadora)

Nota atribuída: 9,0 (NOVE, ZERO) Maíra M. de Silva

(assinatura)

Professor Associado Adriano Almeida Gonçalves Siqueira

Nota atribuída: 9,0 (NOVE, ZERO) P/ Maíra M. de Silva

(assinatura)

Doutor Guilherme Serpa Sestito

Nota atribuída: 9,0 (NOVE, ZERO) P/ Maíra M. de Silva

(assinatura)

Média: 9,0 (NOVE, ZERO)

Resultado: APROVADO

Data: 02 / 12 / 2020 .

Este trabalho tem condições de ser hospedado no Portal Digital da Biblioteca da EESC

SIM ☒ NÃO ☐ Visto do orientador Maíra M. de Silva

DEDICATÓRIA

Dedico este trabalho ao meu pai, Paulo Cesar Giordano, à minha mãe, Rita de Cassia Vicentin Giordano, e à minha irmã, Vitória Vicentin Giordano, que tanto me incentivaram e me ensinaram durante minha trajetória até aqui. Sem vocês, esse trabalho não poderia ter sido desenvolvido.

AGRADECIMENTOS

Agradeço à professora doutora Maíra Martins da Silva por toda a disposição em me ajudar ao longo dos anos em que estive na Universidade, sendo acessível sempre que precisei esclarecer dúvidas técnicas e burocráticas.

RESUMO

Giordano, Murilo Vicentin. **Desenvolvimento de um aplicativo mobile para aplicação no mercado de segurança do trabalho**. 2020. 43 p. Monografia (Trabalho de Conclusão de Curso) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2020.

O presente trabalho apresenta o desenvolvimento de um aplicativo em Flutter para dispositivos móveis para uma empresa do segmento de segurança do trabalho. A empresa é uma prestadora de serviço cujos clientes são outras empresas que necessitam realizar análise de segurança dos riscos aos quais seus trabalhadores estão submetidos. Para executar esse serviço, ela realiza uma vistoria, coletando informações para elaborar, posteriormente, um plano de melhoria para os setores que necessitem. O processo de obtenção das informações do cliente era realizado manualmente, com papel e caneta, preenchendo as folhas impressas com as informações necessárias. Posteriormente, os dados eram inseridos em uma planilha para então serem processados e gerar os relatórios finais. Com o desenvolvimento do aplicativo proposto nesse trabalho, esse processo foi automatizado com o objetivo de agilizá-lo, facilitá-lo e acelerá-lo. O desenvolvimento foi realizado de maneira iterativa com um representante da empresa, com o objetivo de atender aos requisitos dos processos envolvidos. A coleta dos dados é realizada digitalmente através de tablets ou celulares, e o próprio aplicativo é capaz de gerar as planilhas e os relatórios automaticamente, reduzindo, em torno de 85%, o tempo gasto em todo o procedimento.

Palavras-chave: aplicativo mobile; automatização; segurança do trabalho; Flutter.

ABSTRACT

Giordano, Murilo Vicentin. **Development of a mobile app with application in the occupational safety market.** 2020. 43 p. Monografia (Trabalho de Conclusão de Curso) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2020.

This document presents the development of a Flutter application for mobile devices for a company from the occupational safety segment. It is a service provider company whose clients are other companies that need to analyse the safety risks from which their employees are subjected to. In order to do that, the company conducts an inspection, collecting information to elaborate, subsequently, an improvement plan for the sectors that need it. The process of obtaining client information used to be performed manually, using pen and papers, filling printed sheets with the necessary information. Afterwards, that data had to be inserted in a spreadsheet to, then, be processed and generate final reports. With the development of the application proposed in this document, this process was automated in order to streamline, facilitate and accelerate it. The development was done interactively with a representant from the company, and the objective was to attend the requirements of involved processes. Data collecting is digitally done using tablets or smartphones, and the application itself is capable of generating all spreadsheets and reports automatically, reducing the amount of time spent over the complete procedure by approximately 85%.

Keywords: mobile application; automation; occupational safety; Flutter.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1. INTRODUÇÃO

Graças à popularização do acesso aos tablets, smartphones e redes de internet, a sociedade global observa uma tendência cada vez maior na utilização dos aplicativos móveis. Esses apps, popularmente chamados, têm a capacidade de resolver inúmeros problemas em variadas áreas de aplicação, desde os mercados alimentícios e farmacêuticos até os mercados de transporte e entretenimento, por exemplo. Essa abrangente área de atuação comprova a eficiência com que os apps agregam valor aos consumidores e explica os bilhões de downloads nas lojas virtuais todos os anos.

Com as pessoas progressivamente mais acostumadas a utilizarem apps na sua rotina diária, estes viram um excelente meio pelo qual atividades empresariais podem ser automatizadas, barateadas e inovadas. Esse documento foca, portanto, na criação de uma solução em aplicativo para uma empresa de segurança do trabalho, realizando uma reformulação de seus processos na prática. Esse aplicativo foi construído com o framework Flutter, que é uma ferramenta gratuita, multiplataforma e de alto desempenho, ao invés das tecnologias tradicionais de construção nativa.

1.1 Motivação e Objetivos

Uma empresa do segmento de segurança do trabalho realiza visitas a seus clientes, com a intenção de coletar diversos dados referentes aos possíveis riscos que podem estar ou não ameaçando a segurança dos trabalhadores que lá atuam. O técnico de segurança do trabalho efetua o preenchimento de checklists que foram impressos anteriormente à visita de inspeção, com o objetivo de adquirir os dados que serão utilizados mais tarde. Posteriormente, essas informações são inseridas em planilhas nos computadores da empresa para a realização de análises e para que sejam exportados relatórios necessários ao cliente. Dessa forma, viu-se uma oportunidade de melhoria através de um aplicativo que tem o objetivo de automatizar digitalmente esses processos, buscando otimizar o tempo e os recursos financeiros.

Com isso, o técnico que faz a visita de inspeção poderá utilizar seu próprio celular ou tablet para carregar informações anteriores dos seus clientes, caso existam, e fará o preenchimento dos checklists e tabelas para gerar as planilhas e relatórios, tudo de forma automática no aplicativo.

Portanto, essa solução provocará uma redução no tempo gasto durante o processo global, pois o uso do aplicativo traz agilidade no momento da coleta e dispensa-se a etapa de transferência de dados para planilhas e geração manual de relatórios; redução no custo do processo à longo prazo, pois não será necessário gasto com material físico para cada etapa de coleta, mas sim um investimento inicial com os tablets ou celulares; atração de novos clientes, pois a empresa será vista no mercado como moderna, digital, atualizada e inovadora.; e padronização dos relatórios entregues, facilitando as interpretações dos resultados e aumentando a nota de satisfação do cliente.

1.2 Estrutura do Trabalho

No 2º capítulo, os conceitos básicos de Flutter são apresentados, ressaltando sua estrutura de criação de componentes e renderização nas telas, e os principais “packages” terceirizados utilizados.

No 3º capítulo, é explicado a aplicação do Flutter nesse app, mostrando as interfaces de usuário que foram construídas e como os requisitos da empresa foram atendidos. Além disso, é discutida a maneira com que o app armazena as informações salvas pelo usuário.

Por fim, o 4º capítulo é destinado aos resultados e conclusões obtidos com esse trabalho.

2. FLUTTER

Nessa seção serão apresentadas as características básicas de Flutter, incluindo os conceitos de Widgets e Packages, e como essa ferramenta possibilita a criação de aplicativos de forma bem organizada e intuitiva.

2.1 Conceitos básicos

Flutter é um kit de desenvolvimento de software (do inglês, “SDK”) criado pelo Google que possibilita a criação de aplicativos compilados nativamente, através da linguagem Dart (criada, também, pelo Google). Isso significa que todos os componentes de visualização são renderizados usando o próprio mecanismo de alto desempenho do Flutter (CHEON, Y; CHAVEZ, C, et. al 2020).

Ademais, todos os projetos podem ser compilados para Android, iOS, Windows, Mac, Linux e Web, podendo ser chamado, por isso, de uma ferramenta multiplataforma. Essas características oferecem ao desenvolvedor, uma maneira muito poderosa de construir aplicativos que podem ser executados em quase todos os dispositivos móveis da atualidade, com performance muito próxima da máxima que o hardware é capaz de proporcionar. Assim, é possível atingir a maior parte dos usuários de smartphones do mundo, já que 99,9% dos aparelhos vendidos em 2017 foram Android ou iOS (FAYZULLAEV, J, et. al 2018).

O Dart é uma linguagem de programação orientada a objetos, ou seja, cada dado possui um “tipo”, e cada “tipo” possui suas próprias atribuições (propriedades) e métodos (funções). Os “tipos” são também chamados de “classes”. Uma característica importantíssima do Dart é a hereditariedade, que significa que uma classe pode herdar todas as atribuições e métodos de outra classe.

O principal concorrente do Flutter no mercado é o React Native, que é uma tecnologia também multiplataforma pertencente ao Facebook e utiliza o JavaScript como linguagem de desenvolvimento dos apps. Essa ferramenta é alguns anos mais velha e possui, portanto, uma maior comunidade de desenvolvedores. No entanto, o Flutter

entrega um desempenho superior e um mesmo aplicativo ocupa menos espaço na memória do dispositivo quando comparado à sua versão feita em React Native (STENDER, S, et. al 2020). Por isso, o Flutter foi a tecnologia escolhida para desenvolver o app apresentado nesse trabalho.

2.2 Widgets

No Flutter, existem classes que herdam de uma classe Widget. Uma classe Widget (StatelessWidget ou StatefullWidget), possui um método chamado “build()”. Esse método é responsável por renderizar a interface do usuário no dispositivo em que está sendo utilizado. Dessa forma, basta que a função “build()” de um widget seja invocada para que os pixels sejam carregados de acordo. Em um aplicativo construído com Flutter, tudo é widget. Os widgets são os componentes básicos que compõe a estrutura organizacional, visual e funcional das telas. Um widget pode ser, por exemplo, um botão, um texto, uma região da tela, uma imagem, uma lista, e até mesmo uma tela inteira. Dessa forma, para a construção do design do aplicativo, basta criar uma relação pai-filho entre diversos widgets e ajustar as suas propriedades de acordo com os requisitos do projeto, o que torna o desenvolvimento super customizável.

Existem dois tipos de widgets:

- Stateless (sem estados);
- Statefull (com estados).

Quando a função “build()” de um pai de um widget é invocado, a “build()” do filho também é invocada, e a tela será construída de acordo com as propriedades do widgets naquele instante (quando o objeto é instanciado). Isso significa que, caso a interface visual tenha que ser atualizada devido à mudança dos atributos do widget, a função construtora deve ser novamente chamada. Porém, como os StatelessWidgets não possuem estado (ou seja, possuem apenas o estado inicial do instanciamento do objeto), não há como invocar novamente a “build()”. Já os StatefullWidgets possuem o método “setState()”, que executa uma função qualquer criada pelo desenvolvedor (geralmente para mudar propriedades do objeto) e invoca o método construtor novamente. Com a utilização dos estados, portanto, faz-se possível as atualizações da interface visual (WU, W, et. al 2018).

2.3 Packages

Alguns requisitos estabelecidos pela empresa foram:

- Aplicativo capaz armazenar todas as informações na própria memória do dispositivo, sendo capaz de operar de maneira “offline”.
- Ter uma tela para recolher a assinatura do representante do cliente na vistoria.
- Carregamento de um arquivo PDF para que o técnico tenha fácil acesso de leitura durante o preenchimento do checklist.
- Carregamento de planilhas em formato “.xlsx” com informações anteriores de riscos e setores do cliente.
- Criação de planilhas em formato “.xlsx” contendo as novas informações de riscos e setores do cliente.
- Acesso à câmera fotográfica e galeria de fotos para registro em cada pergunta do checklist.

Como pode-se notar, a maioria desses requisitos utilizam funcionalidades nativas do dispositivo, sendo necessário acessar utilidades como câmera, arquivos e armazenamento interno. Esse acesso só é possível através da execução de código nativo, ou seja, Java ou Kotlin (para Android) e Swift ou Objective-C (para iOS), e o próprio Flutter disponibiliza uma ferramenta otimizada para executar tais instruções (DAGNE, L, et. al 2019). Porém, isso dificulta e atrasa muito o desenvolvimento, já que o domínio em mais de uma linguagem torna-se necessário. Felizmente, é possível reaproveitar roteiros escritos por qualquer pessoa.

Como o Flutter possui uma comunidade de desenvolvedores bem estabelecida, é possível utilizar códigos que já foram escritos por outros a fim de otimizar o tempo de desenvolvimento. Assim, existem “packages” que são oferecidos gratuitamente, contendo códigos que realizam tarefas específicas, eliminando a necessidade de que cada pessoa tenha que construir todas as funcionalidades do app.

Por isso, para atender aos requisitos, foi necessário o uso dos seguintes “packages”:

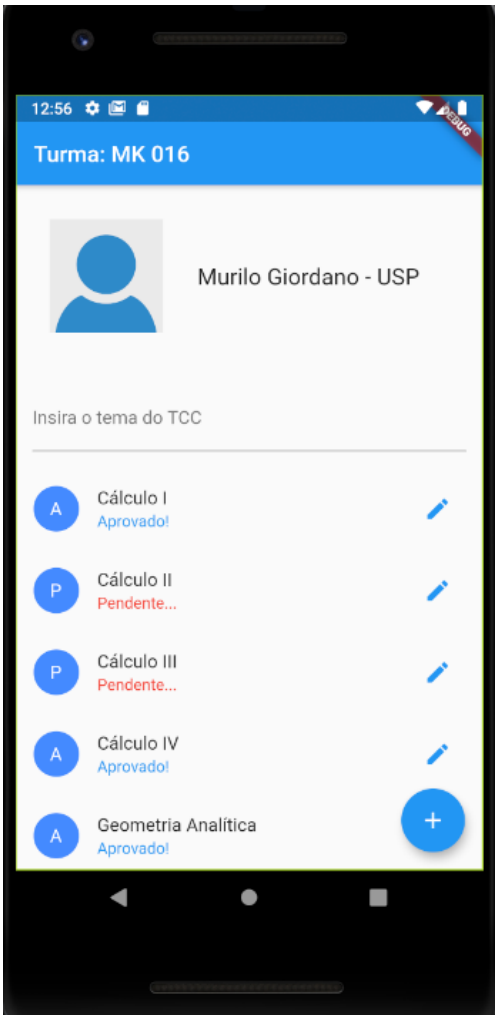
- Signature (versão 3.2.0), para a criação de um widget capaz de recolher a assinatura.

- Sqflite (versão 1.3.0), para a criação de banco de dados com sistema SQLite.
- File_picker (versão 1.13.3), para a tarefa de selecionar arquivos externos ao app.
- Excel (versão 1.1.5), para a manipulação de planilhas.
- Flutter_plugin_pdf_viewer (versão 1.0.7), para a visualização do PDF importado.
- Intl (versão 0.16.1), para a formatação de datas nos campos de “inputs”.
- Image_picker (versão 0.6.7+10), para o acesso à câmera e galeria de imagens.
- Path_provider (versão 1.6.16), para encontrar a localização de documentos nos arquivos do dispositivo.
- Path (versão 1.7.0), para a manipulação das localizações de documentos.
- Pdf (versão 1.11.2), para a criação de documentos PDF.
- Printing (versão 3.6.1), para a exportação de documentos PDFs criados pelo app.
- Downloads_path_provider (versão 0.1.0), para conseguir acesso à pasta de downloads do dispositivo.
- Permission_handler (versão 5.0.1+1), para gerenciar os estados de permissões que app pode ter no dispositivo.

Mais informações podem ser encontradas em (GOOGLE, Flutter packages documentation).

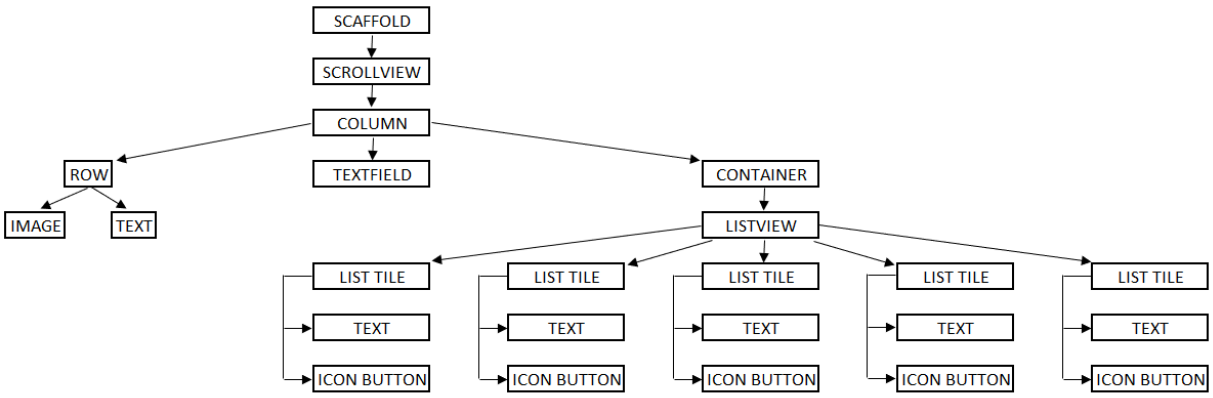
Conforme mencionado anteriormente, os widgets são instanciados num relação pai-filho. Isso torna possível a construção de interfaces extremamente customizáveis e detalhadas. Porém, muitas vezes os dados utilizados para renderizar uma tela são passados para o widget pai global, isto é, o objeto “mais superior na hierarquia”. Nas Figuras 1 e 2 há um exemplo de interface e sua respectiva “widget tree” (árvore com as relações de parentesco entre widgets):

Figura 1 – Exemplo de interface com disciplinas de um aluno.



Fonte: autor próprio.

Figura 2 – Exemplo da árvore de widgets.



Fonte: autor próprio.

Se essa tela possuísse suas informações variáveis de acordo com o aluno escolhido (neste caso, foi escolhido o Murilo Giordano), a instância do aluno deveria ser passada por parâmetro para a criação dos widgets que exibem as suas propriedades. Ou seja, caso os IconButton tivessem que acessar algum método do objeto aluno, esse deveria ser passado sequencialmente para todos os pais dos IconButton, mesmo que o widget Container, por exemplo, não utilize essa função.

Para contornar esse problema, o package “Provider” (versão 4.3.2) pode ser utilizado, permitindo que um widget cujo pai não recebe o objeto por parâmetro consiga acessar os atributos e métodos necessários. Ou seja, qualquer widget que está abaixo, na hierarquia, de uma instância de aluno poderia acessá-lo, sem a necessidade de passá-lo por todos os widgets-pai por parâmetros.

3. CONSTRUÇÃO DO APP

Nesta seção será discutido como a organização dos dados foi estruturada para atender aos requisitos que o app deve atender, além de serem apresentadas todas as interfaces visuais que recebem os inputs do usuário.

3.1 Visão Geral

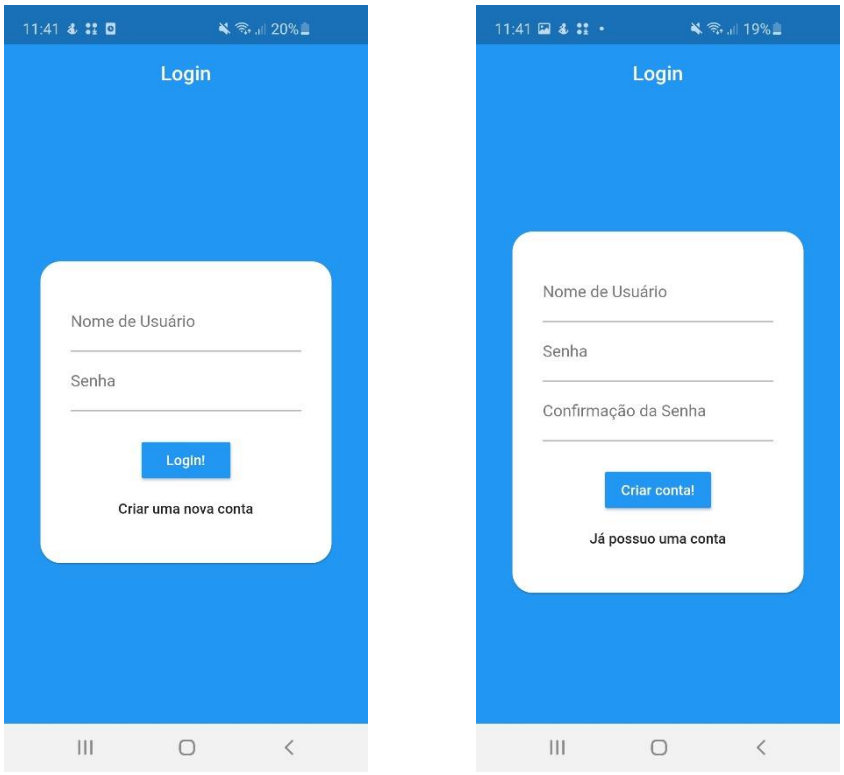
Os requisitos feitos pela empresa para funcionalidades que deveriam estar presentes no app são: sistema de autenticação com nome de usuário e senha, preenchimento das informações já existentes nas folhas utilizadas para elaboração do checklist, recolher assinatura do acompanhante do cliente na visita, importação e visualização de um arquivo PDF, importação de dados de planilhas externas de visitas prévias, e exportação dos resultados coletados em formato de planilha e relatório PDF.

Além disso, como o Flutter possibilita desenvolvimento multiplataforma, algumas interfaces foram construídas em função do tamanho da tela do dispositivo. Assim, o app consegue renderizar diferentes “layouts” caso esteja sendo executado em um celular ou um tablet, carregando sempre uma interface bonita e de fácil usabilidade.

3.2 Autenticação

A tela de autenticação foi estabelecida como a “HomeScreen” do app. Ou seja, quando o aplicativo é executado, o usuário sempre será levado para a tela de autenticação. Dessa maneira, o usuário deverá realizar o “LogIn” na sua conta para poder continuar com o uso do app através da interface representada na Figura 3.

Figura 3 – Interface de autenticação



Fonte: autor próprio.

Inicialmente, as contas devem ser criadas. Clicando-se sobre o botão “Criar uma nova conta”, um terceiro campo para input é mostrado para que a senha seja confirmada. Foi criado um sistema que valida as informações, exigindo que, para que uma conta seja criada com sucesso, o campo “nome de usuário” esteja preenchido e inexistente no banco de dados, e os dois campos de senha contenham inputs idênticos. Com isso, os dados de nome de usuário e senha são registrados no banco de dados, e a tela de autenticação é substituída pela tela de gerenciamento de projetos.

No caso de Login de um usuário já existente, o banco de dados é lido em busca do nome de usuário fornecido. Se for encontrado, verifica-se se a senha está inserida corretamente e, em caso afirmativo, o banco de dados de projetos é buscado para que sejam carregados possíveis projetos que já foram criados anteriormente.

Feita a autenticação, as informações do usuário ativo serão salvas em um objeto para que os resultados tenham a identificação de qual funcionário da empresa foi responsável pela elaboração do checklist correspondente.

3.3 Gerenciamento dos Projetos

Cada usuário cadastrado no aplicativo possui uma lista de projetos, onde cada projeto pode ser criado dinamicamente conforme a necessidade, conforme mostrado na Figura 4. Foi definido um projeto como uma classe que possui todas as propriedades acerca de um cliente. Ou seja, toda a informação coletada de um serviço realizado à um cliente é armazenado em um objeto do tipo Projeto, que por sua vez é associado à determinado nome de usuário.

Figura 4 – Interface de projetos

Fonte: autor próprio.

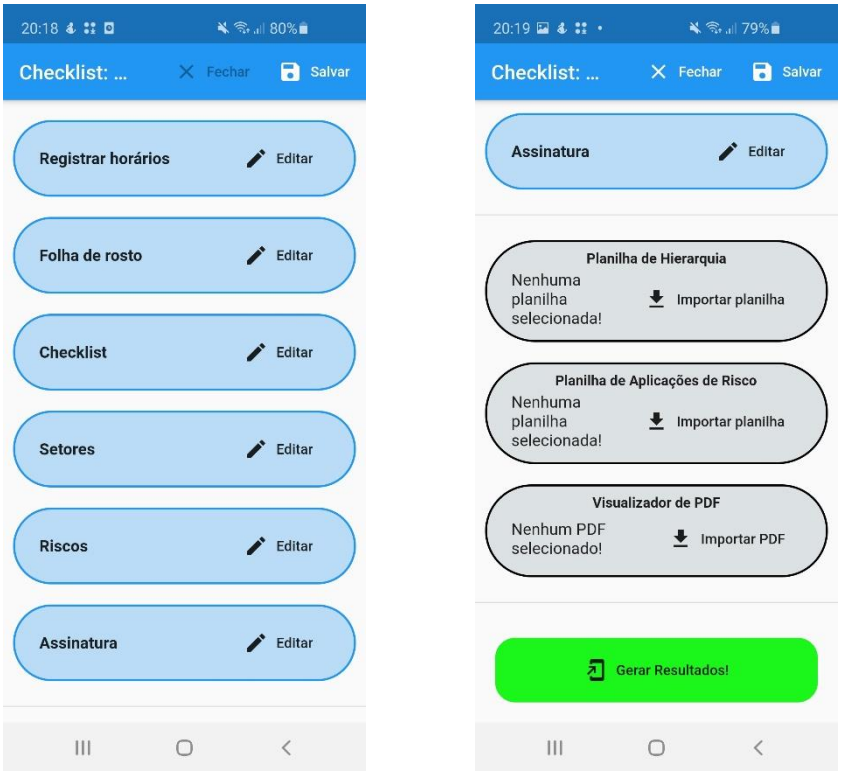
Um projeto possui seu próprio registro de horários, folha de rosto, arquivos importados, checklist, setores, riscos e assinatura, ou seja, tudo aquilo que obtido em uma visita ao cliente. Por isso, foi desenvolvida uma tela de gerenciamento de projetos, onde

o usuário autenticado pode acessar projetos criados anteriormente para editá-los ou criar um novo.

3.4 Menu

Uma vez selecionado o projeto de trabalho, o usuário é levado para uma tela de Menu, vide Figura 5, onde todas as propriedades daquele projeto podem ser criadas, acessadas e editadas. O Menu é a tela responsável por gerenciar o ambiente de trabalho de um técnico da empresa durante a visita a um cliente, além de ser o meio pelo qual as informações lá inseridas serão salvas num banco de dados e exportados como resultados.

Figura 5 – Interface do menu de projeto



Fonte: autor próprio.

Dessa forma, caso o usuário deseje realizar a exportação dos resultados de maneira a gerar a planilha dos setores e dos riscos, e o relatório das respostas inseridas no

checklist, basta salvar o projeto e clicar sobre “Gerar Resultados”. Assim, será possível escolher quais checklists devem ser exportados no PDF do relatório.

3.5 Importações

Com a finalidade de atender aos requisitos de importações de documentos, foi elaborado um conjunto de três objetos que são atributos da classe “Projeto”. Assim, cada um desses objetos são os responsáveis por administrar seu respectivo arquivo externo.

A classe “HierarquiaExcel” é a que armazena o arquivo “.xlsx” selecionado pelo usuário. No momento de sua importação, um método dessa classe é executado, realizando operações com as células da planilha. Tomado como referência o molde da planilha utilizada, a função percorre todas as células da coluna F, que é onde estão registrados os cargos da empresa cliente do projeto. O algoritmo ignora repetições de cargo e células vazias e cria uma lista com os valores adquiridos.

Por sua vez, a classe “SetoresRiscosExcel” gerencia o arquivo “.xlsx” selecionado pelo usuário. O processo que acontece na importação é similar ao descrito no parágrafo anterior com a seguinte diferença: as colunas percorridas são F e N e duas listas são criadas, armazenando as informações de setores e grupos de risco, respectivamente.

Vale lembrar que esses arquivos não são obrigatórios para o funcionamento do app, porém é o meio pelo qual um técnico pode importar possíveis informações obtidas anteriormente (como uma visita de vistoria antiga, por exemplo).

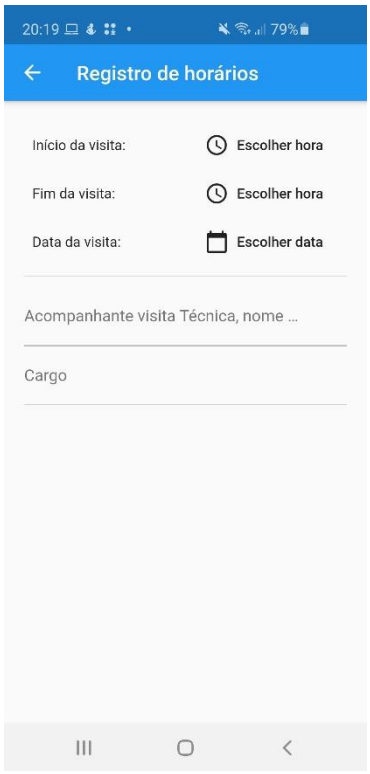
Por último, a classe “MyPDF” é responsável pelas propriedades de um arquivo PDF qualquer que pode ser importado pelo usuário e ficará disponível para a visualização a qualquer momento.

3.6 Registro de horários

Toda visita realizada por representantes da empresa deve registrar informações básicas, como: horário de início e horário de término, data, nome do(a) funcionário(a) do cliente que foi o acompanhante, e o cargo dessa pessoa. Para isso, foi desenvolvido um layout que utiliza “pickers” nativos do Flutter, encurtando o tempo gasto para elaborar as interfaces de seleção de horário e data.

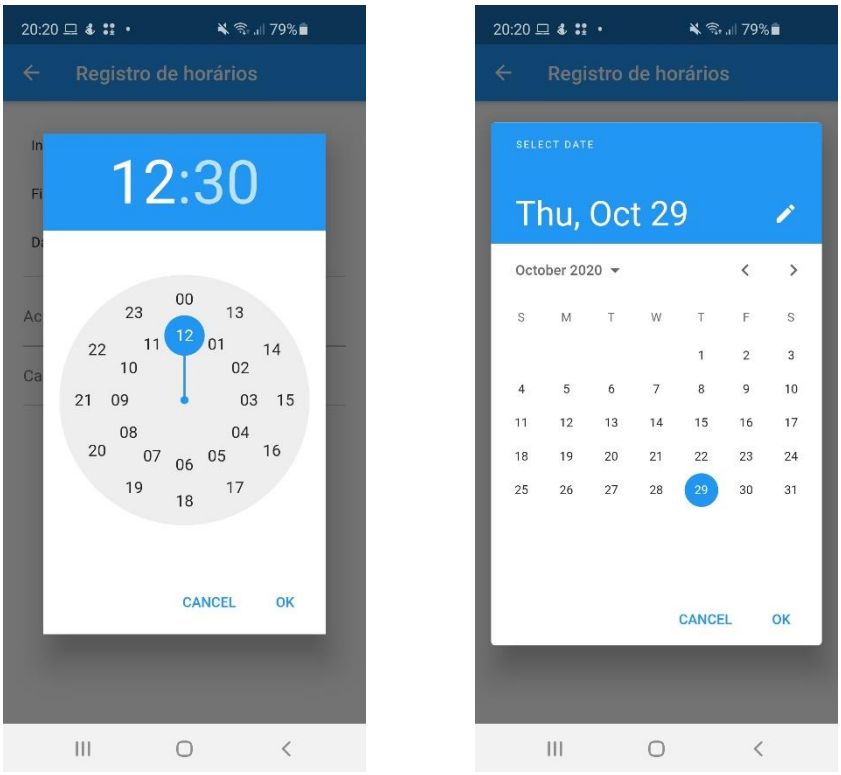
As informações inseridas pelo usuário são armazenadas no objeto “RegistroDeHorários”, que é um atributo do projeto que está sendo editado. A inserção dessas informações é feita na interface da Figura 6, e na Figura 7 é mostrado a aparência de funcionamento dos “pickers”.

Figura 6 – Interface dos registros de horários



Fonte: autor próprio.

Figura 7 – DatePicker e TimePicker

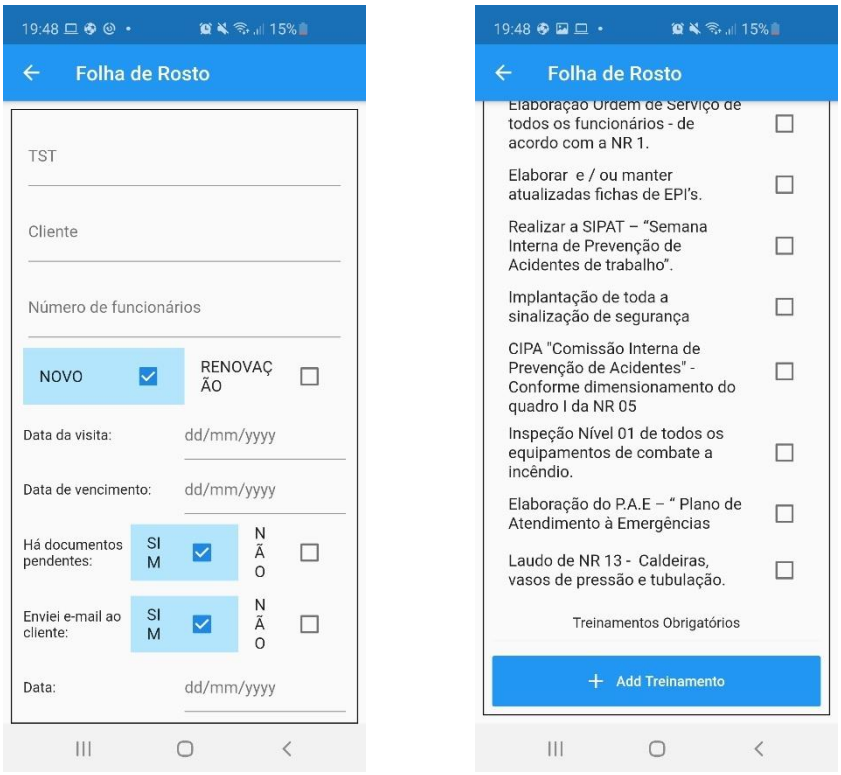


Fonte: autor próprio.

3.7 Folha de Rosto

A Folha de Rosto é um dos documentos utilizados para obter as informações necessárias na elaboração do serviço contratado. Ela coleta dados mais gerais do cliente, armazenando cada input em um atributo do objeto “FolhaDeRosto” do projeto. A interface visual desse item foi feita com base no documento que já é utilizado hoje para preenchimento, e é mostrado na Figura 8.

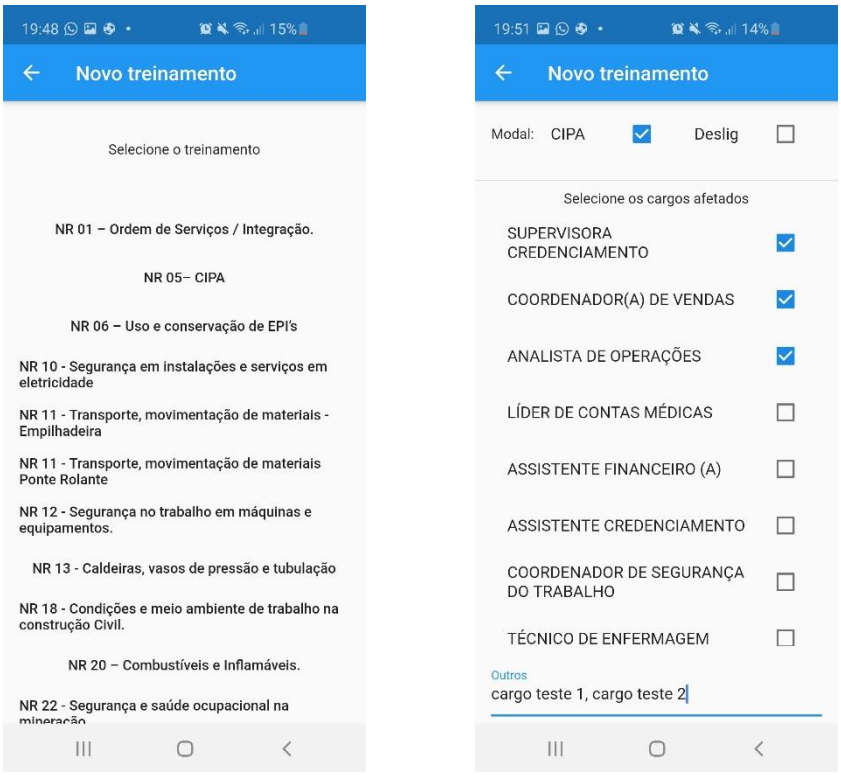
Figura 8 – Interface da folha de rosto



Fonte: autor próprio.

Um desses atributos é uma lista de treinamentos, onde cada treinamento pode ser criado dinamicamente na parte final da Folha de Rosto, conforme Figura 9. Na criação, devem ser selecionados o tipo de treinamento (que é uma lista constante), a modalidade e os cargos afetados. Nessa etapa, caso o usuário tenha importado uma planilha externa de hierarquia conforme mencionado anteriormente, haverá uma lista com todos os cargos obtidos disponíveis para serem selecionados. Caso contrário, apenas a região de escrita manual será exibida.

Figura 9 – Interface de novos treinamentos



Fonte: autor próprio.

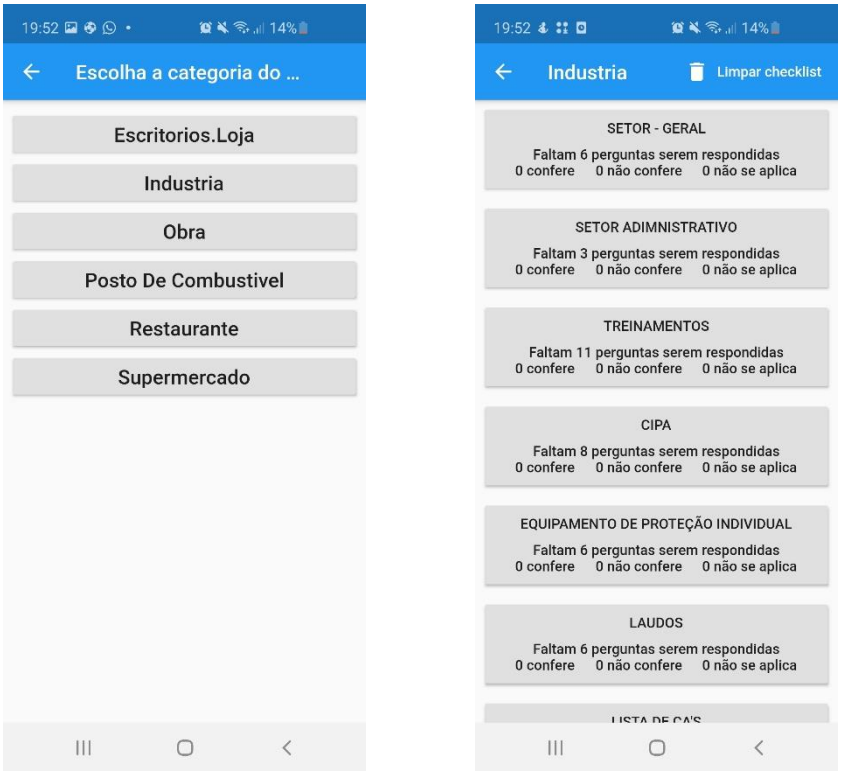
3.8 Checklist

Cada projeto possui um objeto da classe “Checklist”, que possui uma lista de categorias que a empresa utiliza. Ou seja, o técnico possui 6 opções de checklist para aplicar no projeto de acordo com o nicho de atuação do cliente, sendo eles: lojas, indústrias, obras, postos de combustível, restaurantes e supermercados. Isso significa que, para cada categoria, existem diferentes perguntas que devem ser respondidas. Além disso, o usuário pode trabalhar com quantas categorias for necessário por projeto, permitindo a utilização de 1 só projeto para cada área (caso o cliente possua uma indústria e uma loja, por exemplo).

Escolhida a categoria que será preenchida, o usuário encontrará uma tela, conforme Figura 10, com os grupos das perguntas, sendo possível analisar de maneira

rápida quantas perguntas de cada grupo ainda faltam ser respondidas. Isso também facilita o controle do técnico e reduz a probabilidade de alguma pergunta ter sido esquecida.

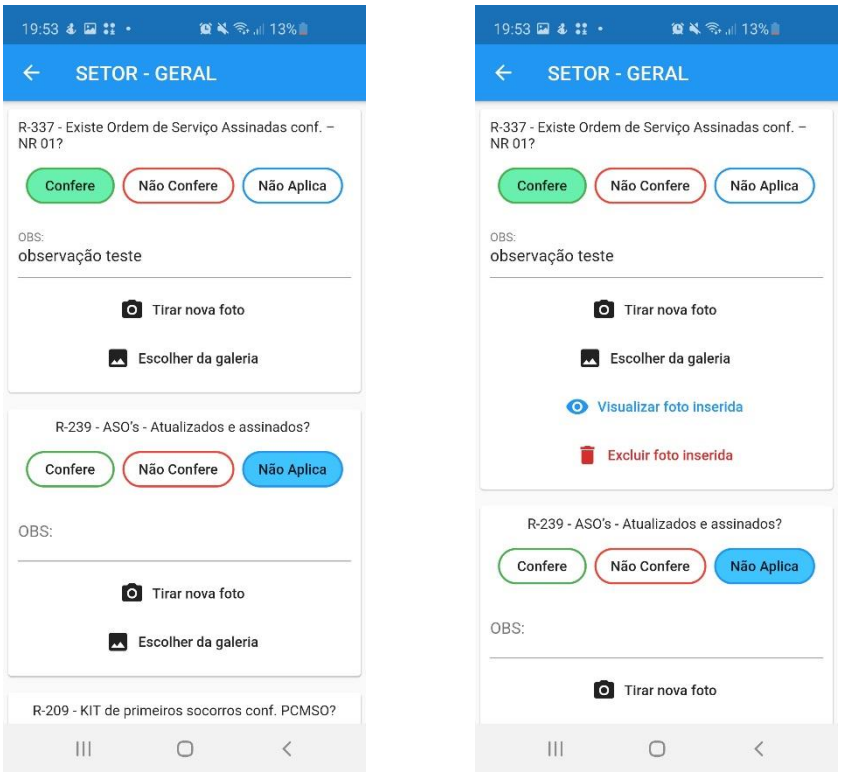
Figura 10 – Escolha das categorias e dos grupos de perguntas



Fonte: autor próprio.

Quando selecionado, as perguntas daquele grupo referentes à categoria escolhida anteriormente são apresentadas em formato de lista. Cada uma recebe um input de resposta, sendo possível marcar a opção “Confere”, “Não Confere” e “Não Aplica”, e um input de imagem, exibido na Figura11. O usuário tem a opção de acessar a câmera do dispositivo ou a galeria de fotos para anexar uma fotografia e, caso seja feito, 2 botões serão adicionados para que a imagem seja visualizada ou excluída. Além disso, há uma opção de inserção de texto, caso seja necessário incluir alguma observação escrita. Todas essas informações vão sendo armazenadas na lista de categorias do objeto “Checklist” do projeto, conforme mencionado anteriormente.

Figura 11 – Interface do checklist



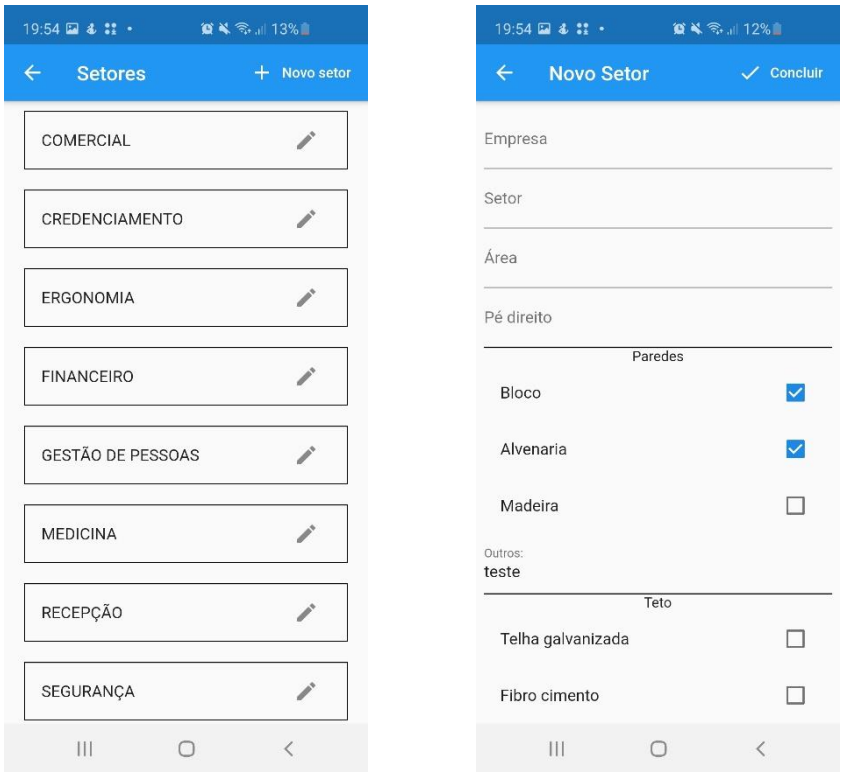
Fonte: autor próprio.

3.9 Setores

A classe “Setores” agrupa todas as informações de setores do cliente, onde cada um possui uma série de atributos que devem ser criados e editados dinamicamente pelo usuário. Informações físicas das paredes, teto, piso, e iluminação são algumas das propriedades adquiridas para os setores do cliente.

Caso uma planilha de riscos tenha sido importada anteriormente, o aplicativo terá criado todos os setores existentes na coluna F da planilha. Esses já serão exibidos na tela de gerenciamento dos setores e poderão ser editados normalmente, conforme exibido na Figura 12.

Figura 12 – Interfaces dos setores



Fonte: autor próprio.

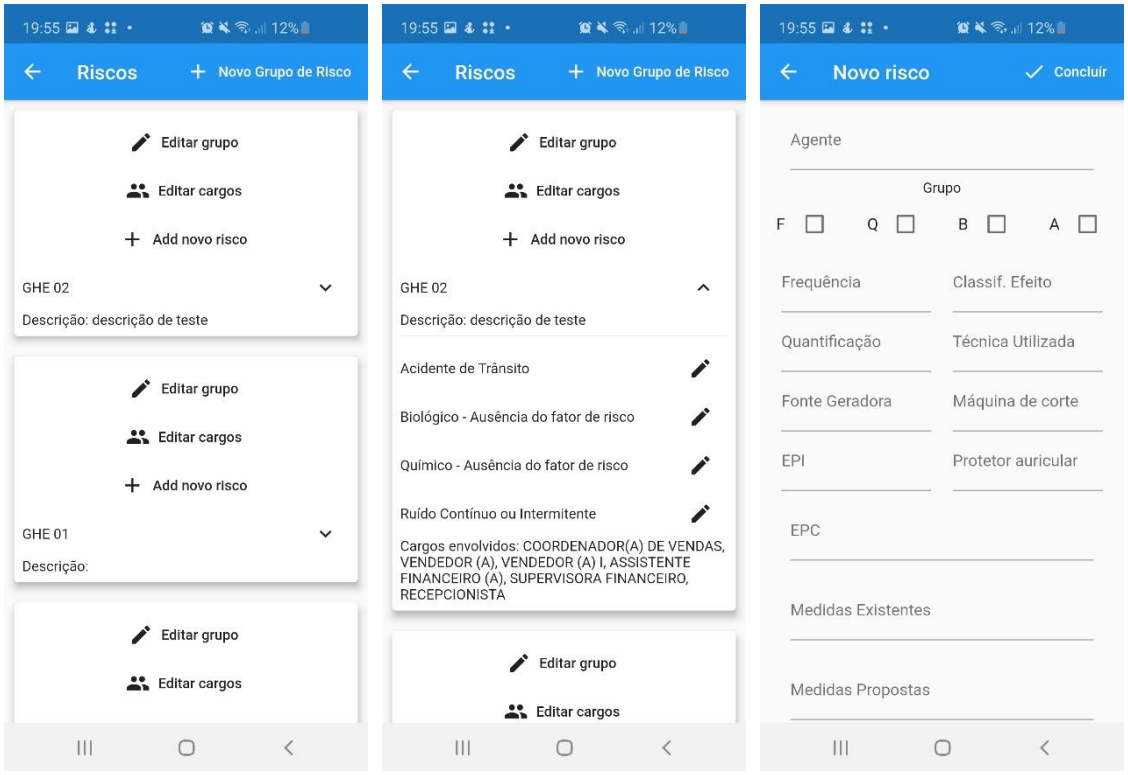
3.10 Riscos

No atributo “Riscos”, são manipulados todos os grupos de riscos que o técnico vai estabelecer de forma dinâmica. Determinados riscos podem ser agrupados em um mesmo objeto, sendo possível atribuir-lhe os cargos do cliente que são afetados por esses riscos. Essa é a maneira pela qual a empresa poderá fazer as análises de quais atitudes podem ser tomadas de acordo com os inputs obtidos nessa parte da vistoria. Assim, os grupos de risco podem ser editados a qualquer instante, sendo possível incluir novos riscos e cargos através dos botões especificados na interface da Figura 13.

Cada risco possui uma série de propriedades que foi criada com base nos documentos já utilizados pela empresa e, por isso, foi criada uma interface semelhante para a obtenção desses dados. Caso uma planilha de hierarquia tenha sido anteriormente importada para o app, uma lista dos cargos presentes no cliente será apresentada para que

o técnico possa incluir quais serão os afetados para o respectivo grupo de risco. Caso contrário, apenas a opção de entrada de texto estará disponível.

Figura 13 – Interfaces dos riscos



Fonte: autor próprio.

3.11 Assinatura

A tela de assinatura foi construída com o auxílio do “package” Signature, criando a interface da Figura 14, onde o acompanhante da visita pode desenhar a sua assinatura. O desenho é, então, convertido em pontos, que são armazenados na propriedade “Assinatura” do projeto.

Figura 14 – Interface da assinatura

Fonte: autor próprio.

3.12 Armazenamento

Como o software pode ser considerado uma aplicação embarcada, é recomendado o uso de um sistema de armazenamento SQLite (BHOSALE, et. al 2015). Todas as informações adquiridas para um projeto serão armazenadas em um banco de dados no próprio dispositivo em que o app está sendo executado. Por isso, não é necessário acesso à conexão de internet para que o app funcione e cada dispositivo não consegue acessar informações de projetos criados em outros dispositivos, atendendo, assim, aos requisitos estabelecidos incialmente. Na verdade, vários bancos de dados SQL são utilizados, sendo que cada um recebe informações sobre um atributo de um projeto.

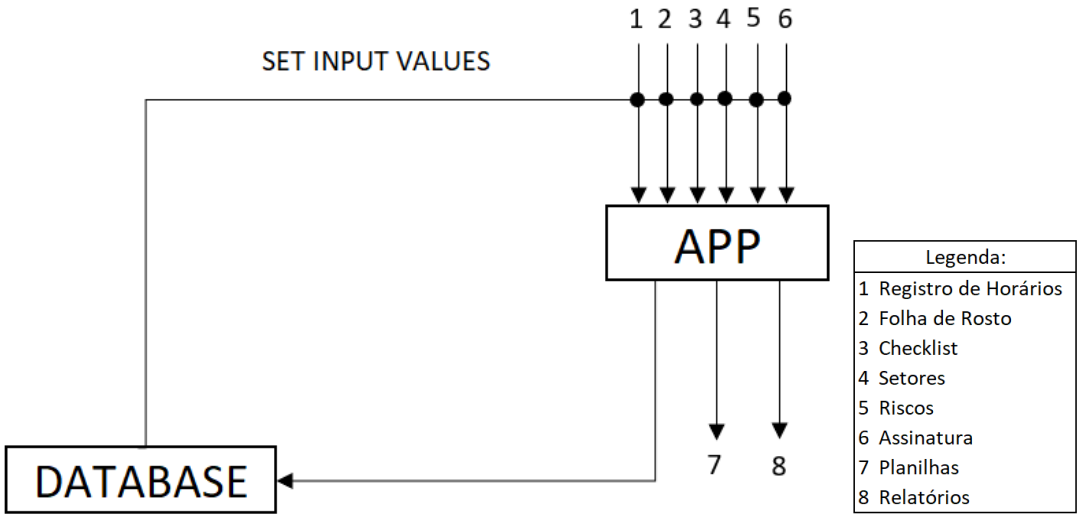
Todos os usuários cadastrados são salvos em um banco de dados e esse conteúdo é sempre lido no momento em que o aplicativo é executado. Dessa forma, algumas validações podem ser feitas durante o uso da tela de login, como por exemplo, a

verificação se um nome de usuário já existe ou se a senha inserida coincide com a senha cadastrada.

Para dados que são fixos para um determinado usuário e projeto, foi necessário a criação de uma coluna de identificação no banco de dados. Isto foi feito concatenando o nome do usuário com o nome do projeto. Para acessar, por exemplo, a data em que a visita foi feita ou a resposta para uma pergunta do checklist, basta ler no respectivo banco de dado o valor que corresponde ao usuário e ao projeto que estão sendo executados. Já para os dados que são criados de maneira dinâmica, o valor de identificação é um ID único correspondente à hora exata de criação do objeto. Dessa forma, elementos como treinamentos, setores, riscos e grupos de riscos são armazenados com o número de ID sendo a referência de busca nos seus bancos de dados. Assim, permite-se a criação de um número ilimitado de objetos para um determinado projeto de usuário, sendo apenas necessário armazenar os valores concatenados dos IDs dos objetos correspondentes.

O armazenamento dessas informações é importante para que o técnico possa editar e acessar as informações de qualquer projeto quando quiser. Portanto, quando o técnico cria um novo projeto, todos os objetos são instanciados sem nenhum valor inicial. Já quando um projeto existente é aberto, os bancos de dados servem como uma fonte de informações que "alimentam" as instâncias do projeto, que as utilizam como um valor inicial. Essa lógica é mostrada de maneira mais sistêmica na Figura 15.

Figura 15 – Inicialização de dados



Fonte: autor próprio.

A Figura 15 ilustra de maneira simplificada quais são as entradas e as saídas de um projeto. Quando o usuário pressiona o botão “Salvar” no topo da interface “Menu”, todos os dados dos inputs são salvos nos bancos de dados e há a possibilidade de se exportarem as planilhas e os relatórios. Quando um usuário acessa aquele projeto posteriormente, todas as informações que haviam sido salvas servem como valores iniciais dos inputs do mesmo projeto e, assim, novas alterações podem ser feitas.

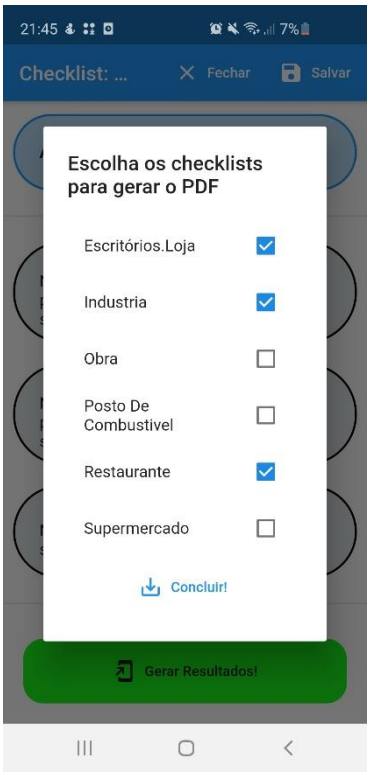
3.13 Exportações

Conforme mencionado anteriormente, a exportação é possível apenas após os dados terem sido salvos. Ou seja, o botão verde no final da página “Menu” fica indisponível até que o usuário realize o armazenamento nos bancos de dados.

Quando pressionado, o botão “Gerar Resultados” mostra uma mensagem de diálogo para que o técnico selecione quais checklists este deseja exportar, conforme Figura 16. Ao concluir, um PDF será criado para cada checklist escolhido, contendo todas as perguntas, respostas, observações e imagens que foram atribuídas a cada um, respectivamente. Esses são os relatórios que a empresa precisa gerar ao final de cada visita, e, por isso, é incluído uma imagem da assinatura recolhida no final de cada PDF, junto com o nome e cargo da pessoa que acompanhou o técnico durante a visita.

Além disso, o app gera duas planilhas em formato “.xlsx” contendo todas as informações de setores e grupos de risco que foram criados e editados durante o preenchimento do projeto do cliente.

Figura 16 – Seleção de relatórios



Fonte: autor próprio.

4. RESULTADOS E CONCLUSÕES

Atualmente a inovação é cada vez mais exigida no mercado. Novos procedimentos e novos produtos são capazes de trazer melhorias, sejam eles otimização de tempo, redução de custos, aumento de práticas ambientalmente sustentáveis, entre outros. O engenheiro possui papel fundamental nesses progressos, pois possui a capacidade técnica e criativa necessárias para desenvolver tais avanços e, desta forma, contribuir com a evolução da sociedade em que a humanidade vive.

Nesse trabalho foi possível analisar o projeto de uma solução automatizada para substituir alguns processos de uma empresa de segurança do trabalho. Houve uma demanda para substituir as etapas de coleta de dados e formulação de relatórios, com o intuito de torná-los digitais, padronizados e mais ágeis.

Devido à presente popularidade do uso de dispositivos móveis como celulares e tablets, a construção de um aplicativo foi considerada uma maneira ideal de implementação de tal mudança. Por isso, foi elaborado o software apresentado nesse trabalho como uma solução própria e inovadora e que, após implementado, foi responsável por:

- Reduzir drasticamente o tempo gasto desde o momento da visita ao cliente até a produção dos documentos resultantes. Antes, as visitas levavam em torno de 2 horas em média e a empresa possuía a capacidade de gerar de 4 a 5 relatórios por dia. Com o uso do aplicativo, é possível realizar a coleta dos dados em menos de 1 hora e 30 minutos e gerar de 10 a 12 relatórios por dia;
- Armazenar digitalmente todos os dados dos clientes, o que, no longo prazo, corresponderá a uma redução no custo do processo já que não será necessário investimento em todos os materiais utilizados para cada serviço. Antes da implementação do aplicativo, eram utilizadas, em média, aproximadamente 100 folhas impressas para cada visita. Por isso, além de contribuir com a diminuição dos custos, há o aumento das práticas ambientalmente sustentáveis;
- Agregar valor para o setor de marketing, pois a empresa será reconhecida no mercado por investir em modernizações produtivas e sustentáveis, aumentando as chances de ampliar a quantidade de clientes;
- Padronizar relatórios e planilhas finais.

Dessa forma, pode-se observar que o aplicativo apresentado nesse trabalho deve reduzir em aproximadamente 25% o tempo médio das visitas aos clientes, aumentar por volta de 144,5%, em média, a capacidade produtiva dos documentos finais, e reduzir quase 100% do consumo de papéis pela visita à campo.

Além disso, esse projeto como um todo deve ser estendido continuamente, uma vez que novas características deverão ser implementadas com o passar do tempo. Assim, conforme novas necessidades surgirem, os representantes da empresa reportarão tais requisitos para que futuras atualizações do aplicativo sejam desenvolvidas.

5. REFERÊNCIAS BIBLIOGRÁFICAS

BHOSALE, S. T; PATIL, T; PATIL, P. **SQLite: Light Database System.** April, 2015.

CHEON, Y; CHAVEZ, C. **Creating Flutter Apps from Native Android Apps.** University of Texas at El Paso. September, 2020.

DAGNE, L. **Flutter for cross-platform App and SDK development.** Metropolia University of Applied Sciences. May, 2019.

FAYZULLAEV, J. **Native-like Cross-Platform Mobile Development Multi-OS Engine & Kotlin Native vs Flutter.** May, 2018.

GOOGLE. **Flutter packages documentation.** Disponível em: https://pub.dev/

STENDER, S; ÅKESSON, H. **Cross-platform Framework Comparison. Flutter & React Native.** Blekinge Institute of Technology. May, 2020.

WU, W. **React Native vs Flutter, cross-platform mobile application frameworks.** Metropolia University of Applied Sciences. March, 2018.