MOBILE APPLICATION FOR MARKETING FAMILY FARMING PRODUCTS

APLICATIVO MOBILE PARA COMERCIALIZAÇÃO DE PRODUTOS DA AGRICULTURA FAMILIAR

GARRONI, Leonardo Caixeta; CRUZ, Lucas Pereira; FIDELIS, Matheus Cardoso Francisco; CARVALHO, Marcos Alberto; CARVALHO, Jaqueline Corrêa Silva; SANTOS, Flávia Aparecida Oliveira Silva; BASTOS, Camila; SOUZA, Patrícia Carolina; SILVA, Vinícius Duarte Esteves; RAMOS, Celso de Ávila

Leonardo Caixeta Garroni, UNIFENAS, Brasil

Lucas Pereira Cruz, UNIFENAS, Brasil

Matheus Cardoso Francisco Fidelis, UNIFENAS, Brasil

Marcos Alberto Carvalho, UNIFENAS, Brasil

Jaqueline Corrêa Silva Carvalho, UNIFENAS, Brasil

Flávia Aparecida Oliveira Santos, UNIFENAS, Brasil

Camila Bastos, UNIFENAS, Brasil

Patrícia Carolina Souza, UNIFENAS, Brasil

Celso de Ávila Ramos, UNIFENAS, Brasil

Vinícius Duarte Esteves Silva, UNIFENAS, Brasil

Revista Científica da UNIFENAS
Universidade Professor Edson Antônio Velano, Brasil
ISSN: 2596-3481
Publicação: Trimestral
vol. 6, nº. 5, 2024
revista@unifenas.br

Recebido: 08/07/2024
Aceito: 28/08/2024
Publicado: 09/09/2024

URL: https://revistas.unifenas.br/index.php/revistaunifenas/issue/view/52

DOI: 10.29327/2385054.6.5-5

ABSTRACT: The commercialization of agricultural products from family farming faces several problems in marketing, from fierce competition with large producers to the problems generated by the COVID-19 pandemic. Based on these problems, it was proposed in this project, the development of a web/mobile application to help the producer in the dissemination and commercialization of their products, in an agile and assisted way. The work is part of the activities of the UNIFENAS Talent Laboratory project. It was proposed to create a hybrid mobile application, which could advertise and sell products from family farming. For the execution of the project, it was defined that Flutter, created and maintained by Google, will be used as a programming tool, which facilitates the execution and creation of hybrid applications. Dart, extremely similar to the C/C++ and Javascript language, being strongly typed. For the development of the application's back-end, an API developed in .NET will be used, which will provide the data stored in the MySQL database for the application. The administration of the entire structure will be carried out by the site that will be developed in React and it will be possible to add new products, properties and farmers. Store (Android) and App Store (IOS). The result of using the application is very beneficial and increases sales because the customer can buy much more easily from a farmer they already know. Conclusions - It is concluded that an application with this approach is of total relevance to the present day, where people have less and less time to attend fairs and shops in their city.

KEYWORDS: Flutter, Development, Farmers, Family Farming, Mobile App, Urban Fairs, .NET, COVID-19.

RESUMO: A comercialização de produtos agrícolas oriundos da agricultura familiar enfrenta diversos problemas na comercialização, desde a acirrada concorrência com os grandes produtores até os problemas gerados pela pandemia da COVID-19. Tendo por base tais problemas, foi proposto neste projeto, desenvolvimento de uma aplicação web/mobile para o auxílio ao produtor na divulgação e comercialização de seus produtos, de forma ágil e assistida. O trabalho faz parte das atividades do projeto Laboratório de Talentos da UNIFENAS. Foi proposto a criação de um

aplicativo móvel híbrido, que conseguisse anunciar e vender os produtos oriundos da agricultura familiar. Para a execução do projeto foi definido que será utilizado como ferramenta de programação o Flutter, criado e mantido pelo Google, que facilita a execução e criação de aplicações híbridas. Juntamente com o Flutter é utilizado a linguagem de programação Dart, extremamente parecida com a linguagem C/C + + e Javascript, sendo fortemente tipada. Para o desenvolvimento do back-end da aplicação, será utilizado uma API desenvolvida em .NET, que fornecerá os dados armazenados no banco de dados MySQL, para o aplicativo. A administração de toda a estrutura ficará por parte do site que será desenvolvido em React e nele será possível adicionar novos produtos, propriedades e agricultores. Com enfoque na facilidade da aplicação ser distribuída, será adicionado o executável nas principais lojas virtuais de aplicações móveis, Play Store (Android) e App Store (IOS). O resultado da utilização do aplicativo é muito benéfico e aumenta as vendas pois o cliente pode comprar com muito mais facilidade de um agricultor que já conhece. Conclusões - Conclui-se que uma aplicação com esse enfoque é de total relevância para os dias atuais, onde as pessoas possuem cada vez menos tempo para frequentar feiras e comércios em sua cidade.

PALAVRAS-CHAVE: Flutter, Desenvolvimento, Agricultores, Agricultura familiar, Aplicativo Móvel, Feiras Urbanas, .NET, COVID-19.

1 INTRODUÇÃO

A agricultura familiar é um dos grandes responsáveis pela produção de alimentos em Alfenas. Essa classe de trabalhadores têm dificuldades frente à concorrência com o Ceasa, que faz com que necessitem diminuir o valor de seus produtos e acabem saindo prejudicados. Isso ocorre devido à falta de contato por parte dos trabalhadores com seus clientes, já que em sua grande maioria, preocupam-se mais com sua produção do que com a sua comercialização, segundo responsáveis da AFFLA (Associação dos Feirantes das Feiras Livres de Alfenas).

O bom desempenho da agricultura familiar, já antes citado no meio acadêmico, foi reconhecido pelas agências governamentais. Porém reconhece-se que o Brasil, tem historicamente, contemplado de forma desproporcional a elite agrária com base na agricultura familiar [1].

A comercialização de produtos agrícolas, produzidos por agricultores da agricultura familiar, sempre sofreram com a constante concorrência de grandes agricultores, e esse âmbito aumentou no período da pandemia do Covid-19 nos anos de 2020 e 2021, onde apresentaram uma grande dificuldade ao exibir e realizar a venda de seus produtos. O projeto em questão teve a função de construir um aplicativo de celular para a venda dos produtos e também uma página de gestão web, para que sejam um mercado digital, assim ajudando agricultores, inicialmente da cidade de Alfenas MG, a expor e vender seus produtos com mais facilidade. A ideia foi criar uma vitrine que além de ajudar na venda, traga para o produtor uma visão geral de quanto ele ganhou, quanto vendeu e seus pedidos futuros. Além disso, deve auxiliar o cliente a conhecer a fazenda do agricultor, observar a qualidade do produto e assim aumentar a divulgação do produtor[2].

A construção de um aplicativo para fazer a intermediação do produtor, que precisa vender seus produtos e do cliente, que muitas vezes não tem a possibilidade de ir a feira no dia que ocorre, ou ao mercado durante sua semana, tem uma importância muito grande no âmbito de acesso, anúncio, controle e venda dos produtos.

Vale destacar também que dos cerca de 5,1 milhões de terras agropecuárias no Brasil, mais de 80% é caracterizado como agricultores familiares. Isso confirma a existência do problema já levantado e o que os impede de avançar nas produções ou impede os compradores de encontrá-los é a falha na comunicação e a falta de exposição dos seus produtos[3].

O termo delivery diz respeito ao serviço de entrega de produtos em endereço solicitado, e o Brasil é o maior mercado de entrega de comida da América Latina, isso pelo fato do delivery oferecer mais praticidade e comodidade a seus consumidores. A crescente demanda de clientes que pedem suas refeições em casa ou até mesmo no trabalho proporcionou um avanço enorme na utilização do delivery e dos aplicativos de comida.

A pergunta norteadora da pesquisa, é, portanto: um aplicativo mobile pode auxiliar Produtores locais a venderem seus produtos de forma ágil e eficiente? Acredita-se que, com um software completo para delivery de produtos da agricultura familiar, é muito mais fácil e rápido um produtor exibir, realizar promoções, vender para mercados, manter seu controle monetário e observar quais produtos estão sendo mais procurados e vendidos em certas épocas do ano. Do lado do cliente, há também uma grande facilidade na hora de escolher, pesquisar e reservar suas mercadorias além da possibilidade de receber o produto em seu próprio lar, já que o aplicativo tem como premissa uma feira com entregas delivery duas vezes por semana.

No desenvolvimento do aplicativo de e-commerce, buscou-se aumentar a procura e a venda de produtos agrícolas, além de facilitar o contato entre produtor e cliente mesmo em momentos de distanciamento social.

2 METODOLOGIA

Este tópico irá abordar todas as etapas de desenvolvimento deste projeto, desde a parte dos requisitos, até a elaboração do aplicativo e do painel web.

2.1 API

Nesse tópico será apresentado os materiais e os métodos que foram utilizados para desenvolver a API para controle e acesso a dados desenvolvida em C# .NET.

2.1.1 Etapas de Desenvolvimento

Durante todo o projeto foi seguido alguns passos para a concretização do trabalho, dentre eles é possível destacar:

- Estudo e aprofundamento das tecnologias referentes ao projeto, com foco na tecnologia responsável pelo desenvolvimento de aplicações móveis escolhida: .NET;
- Escolha do banco de dados para a persistência dos dados da aplicação, o MySQL foi definido como o banco da aplicação;
- Desenvolvimento do projeto utilizando sprints e entrega de melhorias a cada reunião realizada;
- Realização de testes e análise da aplicação.

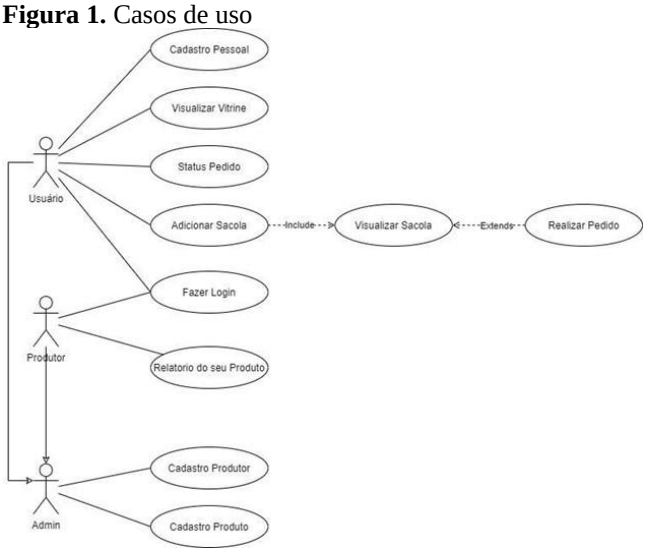
2.1.2 Desenvolvimento

O início desenvolvimento do projeto se deu com o levantamento de requisitos, onde foi possível conhecer o problema, com a visita a uma das propriedades onde é produzido frutas, legumes e verduras, foi capaz de entender melhor o âmbito das adversidades que são enfrentadas pelos produtores na hora de expor seus produtos. Com isso foi possível observar que, para resolver o problema, era necessário a construção de um e-commerce, onde facilitaria a exposição dos produtos e onde são produzidos, desta forma, agregando valor aos trabalhos realizados pela agricultura familiar.

Após o levantamento dos requisitos foi montado um diagrama de caso de uso, que definiu quais eram os papéis de cada usuário na aplicação. Com o diagrama desenvolvido, a avaliação das funcionalidades que deveriam ser implementadas ficará mais nítida. Sendo assim, se deu início ao desenvolvimento da API com as funcionalidades projetadas para que o aplicativo funcione corretamente e atenda aos requisitos.

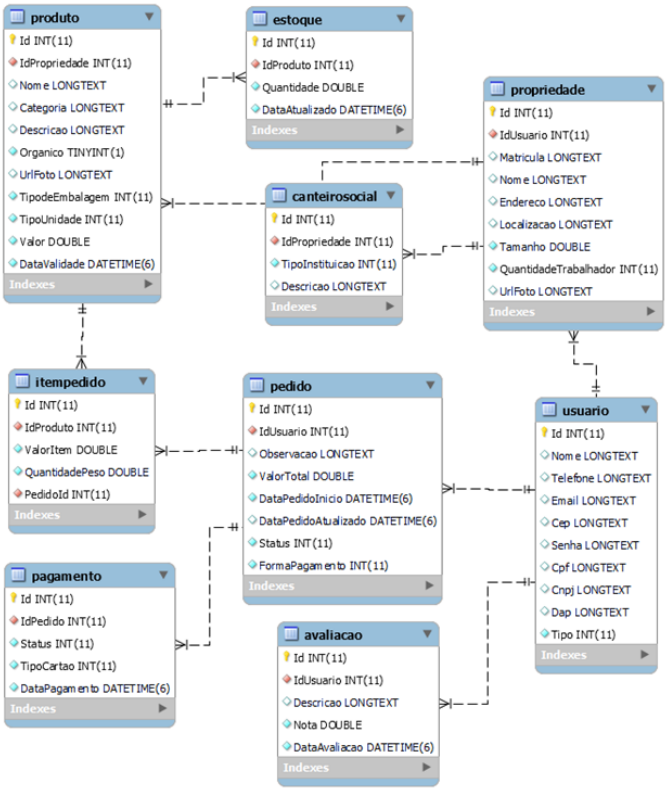
O diagrama de classes foi desenvolvido de acordo com as funcionalidades definidas no caso de uso (FIGURA 1), além de como os usuários irão utilizá-las.

Figura 1. Casos de uso



O diagrama de classes (FIGURA 2) é a base para a implementação do banco de dados que se utilizou para armazenar todos os dados da aplicação. O esquema a seguir mostra todas as entidades presentes no sistema e todos os relacionamentos entre as tabelas.

Figura 2. Diagrama de classes



O banco de dados selecionado foi o MySQL, disponibilizado pela Oracle, é um banco de dados gratuito e de código aberto, onde é exequível de se realizar consultas utilizando a linguagem por SQL, permitindo que os dados possam ser facilmente recuperados na aplicação.

O banco de dados e a api, no momento de execução da construção do aplicativo necessitam de ser colocados em repositórios para a execução em nuvem, para isso utilizou-se o Heroku, site que disponibiliza uma certa quantidade de recursos para a execução em nuvem.

A construção do banco de dados foi realizada a partir de um ORM chamado Entity Framework, a partir dele a criação das tabelas do banco de dados fica por conta das classes definidas de acordo com o diagrama de classe. A criação do banco usado foi realizada no projeto de uma API que utilizou a linguagem C#.

Com a construção da api o projeto passa a ter mais segurança, um fluxo melhor dos dados e aumenta a produtividade do aplicativo que foi criado, pois o mesmo apenas fará a utilização da api, desta forma possibilitando a integração de sistemas e linguagens diferentes. Para a criação da API foi utilizado a IDE Visual Studio, desenvolvida pela Microsoft, com o framework do C# .Net Core 5.0.

Outro ponto importante é a documentação gerada a partir da API com todos os métodos implementados no sistema, todas as chamadas para a api estão disponíveis para testes implementados com a ferramenta Swagger, que facilita a implementação dos métodos e testes rápidos realizados na própria interface.

2.2 Aplicativo Mobile

Nesse tópico será apresentado os materiais e os métodos que foram utilizados para desenvolver o aplicativo mobile em Flutter.

2.2.1 Prototipação

Antes de codificar as telas foi desenvolvido um protótipo de alto nível, utilizando o software de design Figma, onde definiu-se como as telas seriam implementadas e como seria o fluxo de utilização do aplicativo. A criação das telas partiu com o conceito de visualização simples e direta, focando sempre na melhor experiência do usuário que acabou de entrar no aplicativo. Esse preceito partiu do estudo e compreensão de UX.

O software dispõe de cadastro do cliente, sacola de compras, visualização de produtos em destaque, uma vitrine de produtos e propriedades, entre outros. Para que possa ser realizado a venda do produto a aplicação possui as operações necessárias, como inserção de dados, exclusão, atualização e a possibilidade de leitura das mesmas, o conhecido CRUD (Create, Read, Update e Delete) em inglês.

2.2.2 Utilização da aplicação

O software realiza a venda online de produtos oriundos da agricultura familiar, a princípio da cidade de Alfenas, Minas Gerais. O aplicativo é utilizado por dois tipos de usuários, sendo um deles o cliente que fará seus pedidos, e o outro o produtor, este que será cadastrado pelo administrador, que irá ver os pedidos que chegaram além de obter um panorama geral de como estão suas vendas.

Como ilustrado na Figura 3, o cliente tem acesso imediato a tela home, onde é demonstrado um resumo dos principais conteúdos do software. Como a maioria dos aplicativos o que foi desenvolvido tem uma barra inferior, que possui abas para fácil acesso a outras funcionalidades da aplicação, nela existe uma tela de vitrine de produtos e propriedades (Figura 4), uma para a visualização (após o login) de pedidos que estão a caminho e que já foram concluídos, uma tela para a visualização do carrinho de compras e uma tela para a visualização do perfil do usuário logado.

Figura 3. Telas do Aplicativo

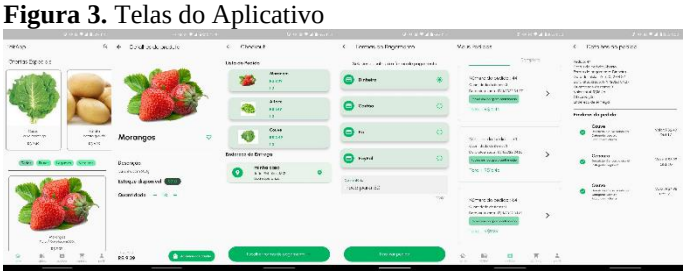


Figura 4. Telas de Vitrine e Propriedades do Produto

2.2.3 Tecnologias empregadas

Para o desenvolvimento do aplicativo foi utilizado como tecnologia de codificação o Flutter e sua linguagem de programação Dart, que auxilia por sua facilidade e provém de muitas bibliotecas para a criação de aplicativos móveis híbridos e também softwares web.

A linguagem de programação Dart, criada pelo Google para ser a linguagem do Flutter, sendo uma linguagem fortemente tipada, facilitou a criação de novas funcionalidades e visualização de erros no código. Utilizada durante o desenvolvimento do aplicativo, tem suas origens destacadas no C++, porém com todas as funcionalidades das atuais linguagens do momento, como o Javascript.

Com o foco na melhor exequibilidade, organização, produtividade e performance foi utilizado o pacote GetX,

disponibilizado nos servidores Google como pacote de apoio principal. O GetX faz o Flutter ser programável com a arquitetura MVC, sem ele a codificação e organização do projeto é dificultada.
Como IDE foi utilizado durante todo o processo de construção o VSCode (Visual Studio Code), editor de código criado pela Microsoft, disponibiliza incontáveis extensões para a codificação de diversas linguagens de programação.

2.2.4 Estrutura do Aplicativo

O padrão de estrutura de pastas definido para o aplicativo foi o padrão MVC, onde é separado cada pasta por camada da aplicação. A pasta Controllers é onde estão localizados todos os controladores que fazem com que os dados sejam buscados e exibidos corretamente nas Screens, a pasta Data é onde está localizada todas as classes de acesso a dados através da aplicação, sendo dividida em duas subpastas. Na pasta Api está localizada a implementação base para realizar as chamadas para api, seja uma chamada get, post ou put, no Repository está implementado todas as entidades do banco de dados para que o controlador realize a requisição dos dados que precisa.
No diretório Helper está implementado as dependências do Getx que são iniciados juntamente com o aplicativo, a pasta Models é responsável por definir todos os modelos de objetos utilizados para manipular os dados vindo da api ou para utilização dentro do aplicativo. A pasta Routes é onde estão centralizadas todas as rotas pertencentes ao aplicativo, sendo possível acessar de qualquer tela da aplicação, já no diretório Screens estão todas as telas implementadas. O diretório Utils é responsável por conter todas as constantes utilizadas no aplicativo, como estilo de cores e URLs para chamadas da api e por fim os Widgets é onde se encontram todos os componentes visuais que podem ser reutilizados na aplicação.

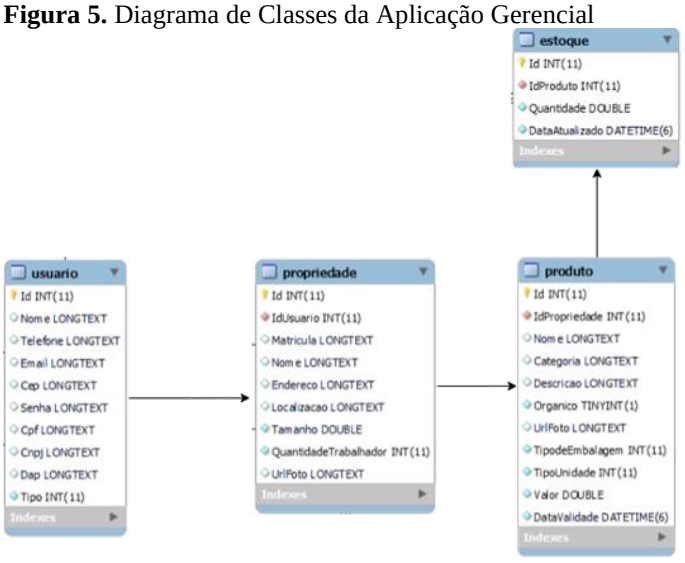
2.2.5 Codificação do Aplicativo

O início da codificação se deu replicando o layout das telas definidas no protótipo, com o layout já desenvolvido fica mais fácil dar andamento com as integrações e estados que o aplicativo deve ter. A base para todas as requisições para a api está implementada dentro da classe ApiClient que é onde os principais métodos como get, post e put estão implementados facilitando o acesso dos repositórios, não sendo necessário replicar o código base da api para todas as requisições.
Após ter a base das requisições codificadas, pode-se dar início na criação dos repositórios, que é a camada responsável por realizar a chamada para os métodos da api. No Repository da entidade de Login está implementado todas as requisições necessárias para realizar o login do usuário e também para realizar a redefinição da senha. É possível notar como fica simples e limpo as funções com a estruturação planejada para o projeto.
A implementação dos controllers é o último passo para o funcionamento da aplicação, pois é nele que estão as funções que fazem a integração dos dados vindo da api com a interface exibida para o usuário. O controlador de login está controlando a sessão do usuário, assim como o token que é retornado da api é utilizado para realizar as demais requisições, como realizar um pedido ou buscar todas as informações do usuário.
Para que os controladores e repositórios estejam sempre instanciados para serem usados a qualquer momento, é necessário inicializar eles juntamente com o aplicativo, por isso foi criado a classe Dependences, que é responsável por inicializar todas classes necessárias para o funcionamento do aplicativo, ficando centralizada esta responsabilidade única para a classe e caso seja necessário a criação de novos repositórios e controladores, basta adicionar sua instância na classe que eles estarão prontos para serem utilizados pelo aplicativo.

2.3 Aplicação Gerencial WEB

Neste tópico serão abordados todos os passos para a construção da aplicação de gerenciamento do aplicativo, um painel de controle, conhecido também como dashboard, que será executado na web. A criação da aplicação web surgiu com o intuito de facilitar o cadastro de produtores, propriedades e produtos, por um administrador interno da associação dos produtores rurais. A associação é responsável por toda a gestão das entregas e cadastro dos produtores.
Juntamente com o levantamento dos requisitos do aplicativo, foi definido como seria a construção de toda a lógica da aplicação web, para que a mesma já fosse conectada com o aplicativo. Levando em consideração o diagrama de classes desenvolvido e do caso de uso construído, ficou definido que a manutenção das tabelas: usuário (sendo cadastrado o tipo de usuário de número 1 - tipo do produtor), propriedade, produto e estoque será realizada através do painel de controle do projeto, como é possível observar no diagrama reduzido (Figura 5).

Figura 5. Diagrama de Classes da Aplicação Gerencial



O diagrama de classes é o mesmo que foi levantado após a

reunião com o cliente no momento da entrevista, porém com uma redução em sua exibição, para a melhor visualização de como a dashboard funciona. Para iniciar o desenvolvimento da dashboard, utilizando o VSCode como IDE, foi criado um novo projeto Flutter. Após o término do download das dependências iniciais da aplicação, já é possível executar e verificar que já possui um app de exemplo.

O primeiro passo, é colocar o padrão de projeto definido em ação, o MVC. Além das pastas Model (modelo), View (páginas) e Controller (Controladores), para manter o projeto organizado foi adicionado mais pastas sendo: a Constants (contendo objetos que nunca mudam como a paleta de cores), a Helpers (contendo classes de conexão com o banco de dados e dependências locais), Routing (responsável pelas rotas do projeto, direcionamento de cada botão) e uma pasta de Widgets (que é responsável por separar os widgets que foram usados no projeto).

No Flutter, para que seja possível a utilização do padrão de projeto e que seja mais facilmente organizado, é necessário instalar e importar a biblioteca Getx, que proporciona métodos e classes para o melhor desenvolvimento no padrão. Juntamente com o Get, também foram adicionadas mais algumas bibliotecas que serão de grande auxílio no desenvolvimento da aplicação, como o "syncfusion flutter charts", essa por sua vez ficando responsável por criar gráficos e também o "data table 2" que é uma biblioteca para a criação de tabelas.

Com os arquivos separados e as bibliotecas principais instaladas iniciou-se a criação das telas da dashboard. Como ponto de partida os primeiros widgets adicionados eram os que repetem por toda a aplicação com o cabeçalho, menu lateral esquerdo e a tela que deve mudar conforme clicado no menu lateral. Junto com os widgets também se adicionou às operações de roteamento do projeto, redirecionando para as telas corretas utilizando os recursos da biblioteca Get. A primeira tela criada foi a de visão geral da dashboard, onde é possível verificar todas as operações realizadas dentro do painel de controle e da aplicação.

Para que as ações necessárias da dashboard fossem possíveis, ficaram divididas em duas telas cada, sendo uma tela de exibição da lista dos dados já cadastrados por meio de tabelas (pages) e uma tela de formulário (forms) responsável pela inserção e edição de um dado. Além disso, dentro de cada tela existem widgets necessários para construção e execução da lógica da página.

Todas as operações da dashboard também estão relacionadas com o banco de dados escolhido, consequentemente dependendo da API criada para o projeto. Com isso a conexão da dashboard com o micro serviço é de grande importância, seja com a finalidade de criar-se os métodos de inserção, edição e deleção, como também para a exibição dos dados que já foram cadastrados pelo administrador anteriormente.

Com o intuito de fazer a conexão da dashboard com a API, elaborou-se uma nova classe responsável por conexões, sendo esta uma implementação da GetConnect e da GetxService, duas classes disponíveis dentro da biblioteca Get. Dentro da classe nomeada como ApiClient é iniciado o token de quem está logado, qual é a URL base da API, além de indicar o cabeçalho que irá transitar as informações requisitadas e enviadas.

Dentro da ApiClient é necessário que todas as requisições básicas sejam pré implementadas para a busca e salvamento dos dados, onde se for uma busca se passa o token, mas sem parâmetro de pesquisa, ou um salvamento/pesquisa que necessita de parâmetro. Todas as requisições são "async" (assíncronas), ou seja, requisições que demandam um certo tempo para enviar do cliente, pesquisar no servidor e trazer de volta para o ponto de partida.

O CRUD são as quatro operações básicas usadas em banco de dados relacionais. Na dashboard dividiu-se da seguinte forma: os arquivos form fazem as operações de criação (C) e edição (U), da mesma forma os arquivos "page" com seus widgets "table" fazem a leitura dos dados (R) exibindo-os na tabela, além de adicionar botões para a realização de deleção (D) imediata do dado e o redirecionamento para a realização da edição na página de form (FIGURA 6).

Figura 6. Telas da Aplicação Gerencial

Uma aplicação web nos dias atuais necessita de ser totalmente responsiva dependendo da plataforma que está sendo utilizada, na dashboard não poderia ser diferente. Para a administração da responsividade criaram-se algumas classes de apoio, onde as mesmas, após serem definidas, realizam cálculos para a verificação do tamanho da tela do dispositivo que está à acessando. A classe "ResponsiveWidget" funciona em segundo plano com a barra lateral e faz o cálculo de que tamanho é a tela, enviando para os widgets filhos de que tamanho devem se exibir.

Já a classe "Dimensions", serve para calcular em momento de execução como será exibido o tamanho do componente que está sendo construído. Seu cálculo é feito manualmente, porém após indexado dentro de uma classe abstrata, pode ser chamado em qualquer componente realizando a mudança dependendo do tamanho da tela.

Com a finalidade de manter a estilização das cores da aplicação, o arquivo "style" recebeu as cores fixas da aplicação, desta forma manteve a aplicação linear, ou seja,

sem fugir das cores definidas como tema do projeto. Como medida de organização, o arquivo de cores foi dividido em cores de fundo (bg), cores primárias (main), cor de textos (text) e cores terciárias (tertiary), além de algumas cores situacionais, todas estas definidas com o cliente.
A implantação da dashboard se deu com o site heroku, que disponibiliza a forma de colocar sites online para testes, utilizando o github como gerenciador de versões e atualizações do painel administrativo. Para que a experiência ficasse mais imersiva para o administrador real, a aplicação também foi implantada no github pages, que disponibiliza de forma mais rápida o acesso para a dashboard. Porém como a aplicação é de tamanho grande o site heroku parou de suprir esse tipo de aplicação. Com esse advento buscou-se a criação de um programa instalável para o windows, a princípio, que foi a melhor solução para a execução do software pelos administradores da AFFLA.

3 RESULTADOS E DISCUSSÃO

No aplicativo, foram pesquisadas cerca de 28 pessoas que testaram a aplicação para a verificação de usabilidade e procura de bugs, após utilizarem e verificarem como estava o software responderam a um questionário de satisfação das funcionalidades. Por meio da pesquisa foi possível identificar possíveis falhas e planejar possíveis melhorias que podem ser realizadas futuramente. Na avaliação de como o usuário achou do aplicativo teve uma boa aceitação onde 75% dos entrevistados colocaram que a aplicação supre muito bem grandes aplicações de venda de produtos, ficando 21,4% dizendo que foi aceitável e apenas 3,6% colocando que a aplicação é mediana.
Dentre as perguntas havia uma voltada para problemas que as pessoas encontraram ao utilizar a aplicação e foi identificado por 7,7% dos entrevistados alguns bugs na utilização, principalmente na hora de fazer um pedido. Sobre o problema da experiência do usuário apenas 3,8% dos entrevistados colocaram que acabaram se sentindo perdidos no momento em que utilizaram a aplicação, o restante dos interrogados não tiveram problemas de execução.
Agora no quesito de apreciação do aplicativo, 32,1% dos entrevistados colocaram que as funcionalidades da aplicação são ótimas, 28,6% que a aparência é boa, 25% que a velocidade é o que se destaca, já o restante ficou bem dividido entre navegação e conteúdo. Sobre sugestões de melhorias ficou destacado a recomendação de se adicionar mais funcionalidades para o software, como a adição de um sistema de pagamento integrado, além de melhorias na aparência focando nas cores da aplicação e correção dos maiores bugs.
Já na dashboard foram feitos testes entre os integrantes do grupo, que testaram a usabilidade e a velocidade de cada ação do painel de controle criado. Foram identificadas algumas melhorias que poderiam deixar a aplicação mais rápida e concisa com a demanda que foi estipulada, porém todas as funcionalidades atuais já satisfazem todas as necessidades do projeto.

CONCLUSÃO

O aplicativo foi desenvolvido com o intuito de facilitar a administração de compra e venda de produtos dos produtores participantes da AFFLA durante a pandemia, possibilitando a criação de uma vitrine para os seus itens onde os clientes possam efetuar suas compras, sem sair de suas casas.
Mesmo após o fim da pandemia, acredita-se que o aplicativo pode ser usado da mesma forma, visto que o modelo de e-commerce tornou-se cada vez mais relevante no mercado. Outro ponto, seria que não existem outros aplicativos com o mesmo objetivo para o público local, o que o torna uma exclusividade na região.
O objetivo proposto para o aplicativo, que é o fluxo de rotas para que o usuário consiga se cadastrar, autenticar e efetuar compra de produtos, podendo acompanhar os detalhes do seu pedido, além da construção do painel de gerenciamento interno da aplicação foram bem satisfeitas. Ambos os softwares desenvolvidos precisam ser polidos e melhorados porém até o momento é suficiente para a associação.
A primeira versão encontra-se finalizada e é uma versão construída com os pontos mais importantes para o funcionamento básico que foi proposto, porém existem correções a serem feitas no futuro. Acredita-se que o projeto possa ser continuado por outros acadêmicos, onde será melhorado e possivelmente implantado em todas as plataformas atuais.

REFERÊNCIAS

[1] CARNEIRO, M. J. (1997). Política pública e agricultura familiar: uma leitura do Pronaf. In Estudos Sociedade e Agricultura. Rio de Janeiro, v. 5, n. 1, pp. 70-82. https://revistaesa.com/ojs/index.php/esa/issue/view/9.

[2] SANTOS, Lucas Domingos dos. MANEJE CHAT: PROJETO DE UM APLICATIVO DE ASSISTÊNCIA TÉCNICA À AGRICULTURA FAMILIAR. Trabalho de Curso - Universidade Federal de Santa Catarina. Florianópolis, v.1, n.1, p. 1-116, Jan/Jun. 2021.

[3] CODAF. A importância da Agricultura Familiar. https://codaf.tupa.unesp.br/agricultura-familiar/a-importancia-da-agricultura-familiar