Desenvolvimento de aplicativo para a capacitação de trabalhadores agrícolas

CAIO SOUZA LIMA
LUCAS EDUARDO DESSY
ELOIZA MARTINS PRIMO CAPELOCI

Fatec – Faculdade de Tecnologia de Pompeia

Resumo

Este trabalho tem como objetivo a utilização de estratégias e técnicas já introduzidas pelas empresas, para desenvolver um aplicativo direcionado a capacitar os trabalhadores agrícolas que padronize e facilite o acesso às informações de modo a assegurar a capacitação dos trabalhadores de forma rápida e efetiva. Seguindo o padrão de modelagem de *software Domain Driven Design* (DDD) foi utilizado o *framework* Flutter, Firebase como *Backend as a Service* (BaaS) e GitHub para a hospedagem do código. Assim como proposto pelo DDD o software foi separado por camadas, sendo elas: *Domain*, *Infrastructure*, *Application* e *Presentation* de modo a definir boas práticas e padrões de design, assim facilitando a manutenção e protegendo as regras de negócios. Durante o desenvolvimento as tecnologias escolhidas se mostraram muito eficientes facilitando a implementação de funcionalidades como a autenticação de usuário e o armazenamento das informações do aplicativo como o próprio código. O uso do aplicativo para empresas pode reduzir custos referentes a preparação do ambiente que ocorrerá o treinamento e ao mesmo tempo facilita o acesso e disponibilização dos treinamentos e materiais. Já para os trabalhadores, o aplicativo torna-se uma facilidade na realização dos treinamentos, pois as videoaulas podem ser assistidas de acordo com o tempo disponível, onde desejar e ainda revê-las quando quiser.

Palavras-chave: *Domain Driven Design*; Flutter; *Business Logic Components*; Firebase; Treinamento.

App development for the training of agricultural workers

Abstract

This research aims to use strategies and techniques already introduced by companies, to develop an application aimed at empower agricultural workers, seeking to standardize and facilitate access to information in order to ensure the training of workers quickly and effectively. Following the pattern of software modeling Domain Driven Design (DDD) the Flutter framework was used, Firebase as Backend as a Service (BaaS) and GitHub for code hosting. As proposed by DDD, the software was separated by layers, these being: Domain, Infrastructure, Application and Presentation in order to define good practices and design patterns, thus facilitating maintenance and protecting business rules. During the development the chosen technologies proved to be very efficient, facilitating the implementation of functionalities such as user authentication and storing both the application information and the code itself. The use of the application for companies can reduce costs referring to the preparation of the environment where the training will take place and at the same time facilitates the access and availability of training and materials. For the workers, on the other hand, the application becomes a facility for carrying out the training, because they can watch the video classes according to their time, wherever they wish, and still review them whenever they want.

Keywords: *Domain Driven Design*; Flutter; *Business Logic Components*; Firebase; *Traning*.

---

1 INTRODUÇÃO

Ao longo dos anos a Agricultura vem sendo alvo de diversos avanços tecnológicos, porém por melhor que isso seja muitos trabalhadores ainda não estão prontos para utilizar as novas máquinas e tecnologias que vem sendo utilizadas. Segundo Almeida et al. (2006) diante a chegada de fenômenos como a introdução de tecnologias poupadoras e o surgimento de novas culturas, surge a necessidade de substituir a mão de obra de menor qualificação por uma de qualificação maior. Neste contexto, tem sido observado um aumento na rotatividade dos trabalhadores agrícolas, o que se faz necessário a capacitação de maneira mais rápida e eficiente com orientações e treinamentos específicos.

O uso de sites para disponibilização de cursos de forma online já vem sendo utilizados por muitas empresas como, por exemplo, a utilização da plataforma Rocketseat (2022) que possui um ambiente onde disponibiliza seus cursos gratuitos e pagos, porém empresas como a Qualifica que buscam o foco no uso de aplicativos e disponibilização de conteúdos exclusivos é um fenômeno recente (QUALIFICA, 2022). De acordo com a rede social LinkedIn (2018), 94% dos funcionários permaneceriam por mais tempo em empresas que os ajudassem a se desenvolver e 58% dos funcionários preferem aprender em seu próprio ritmo.

Atualmente o acesso à Internet tem crescido, inclusive nas zonas rurais que estão cada vez mais conectadas, como retratado na pesquisa da Cetic (Centro Regional de Estudos para o Desenvolvimento da Sociedade da Informação) com dados de 2021, o número percentual de pessoas que tinham acesso à internet em 2019 era de 53% e em 2021 73%, tendo um aumento de 20% na quantidade da população com acesso, entretanto dentre os dispositivos utilizados com a finalidade se conectar à rede o celular tomou destaque pelo fato de 83% das pessoas disporem dele como a única forma de se conectar à internet(YUNES, 2022).

Ao analisar as tecnologias utilizadas pelas empresas desenvolverem seus aplicativos, tem sido utilizado o *framework* Flutter baseado na linguagem de programação *Dart.* Como é retratado em seu site Flutter (2022), este *framework* foi criado pela Google e ela ainda está oferecendo suporte e se utiliza desta ferramenta e outras empresas como a Nubank, BMW, Toyota e Ebay também utilizam o Flutter como *framework* no desenvolvimento de seus produtos.

Este trabalho busca desenvolver um aplicativo voltado para capacitação e treinamento de trabalhadores da área agrícola, utilizando de estratégias e técnicas já aplicadas pelas empresas, facilitando o acesso e padronizando as informações, visando garantir a qualificação dos profissionais de maneira ágil e efetiva.

Visando melhor desempenho durante o processo de desenvolvimento foi escolhido o padrão de modelagem *Domain-Driven Design* (DDD), pois a forma como ele realiza a divisão das camadas possibilita o trabalho em conjunto de diversas pessoas evitando conflitos e facilitando manutenções futuras.

2 MATERIAIS E MÉTODOS

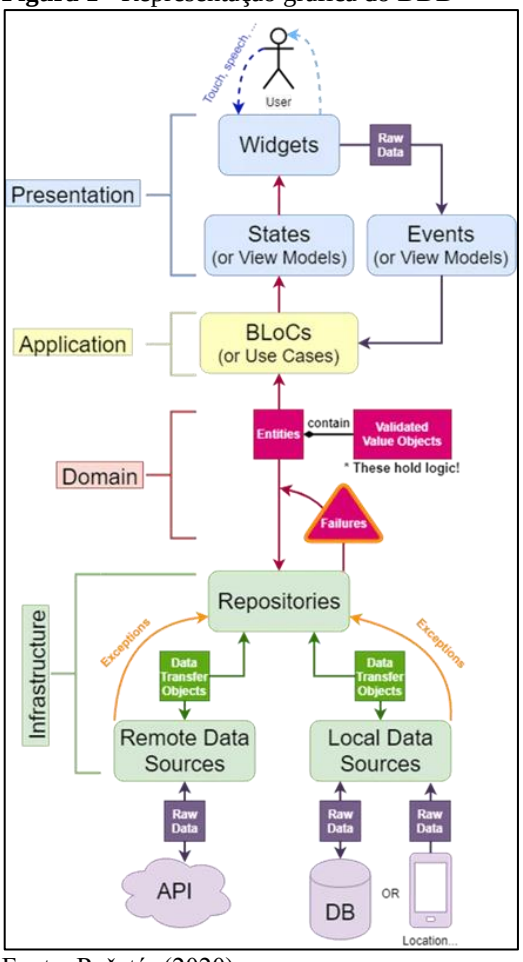
Foi usado o Flutter como *framework* para a criação do aplicativo, pois conforme mostrado por Demedyuk e Tsybulskyi (2020) o Flutter possui um desempenho semelhante às tecnologias nativas além de possuir suporte a multiplataformas.

Para iniciar o processo de desenvolvimento foi selecionado o padrão de modelagem *Domain-Driven Design* (DDD). Segundo Masotti (2022) o DDD combina práticas do desenvolvimento e design de software, oferecendo uma estrutura que auxilia nas tomadas de decisões. Por ser uma estrutura focada na camada de Domínio que funciona como o núcleo da aplicação, é nela que ficarão todas as regras de negócios. Esta camada é independente de todas as outras, entretanto todas as camadas dependem dela, ou seja, caso haja alterações nas camadas

de *Presentation*, *Application* ou *Infrastructure*, essas mudanças não impactarão em nada a camada de *Domain*, na verdade ocorrerá o oposto e a *Domain* as influenciará.

Como observado na Figura 1, segundo Rešetár (2020), o DDD é dividido em quatro camadas, cada uma possuindo sua própria responsabilidade dentro da estrutura, sendo elas: *Presentation*, que é responsável por cuidar dos *Widgets* e a parte visual do aplicativo, não abrangendo lógicas relacionadas ao envio de dados para o servidor ou banco de dados; *Application*, que tem a função de gerenciar a conexão entre as camadas, armazenando o estado da aplicação e regras de negócio, sendo a camada onde será aplicado o Cubit. Angelov (2022) descreve o Cubit da seguinte forma: "Cubit *is a lightweight state management solution. It is a subset of the bloc package that does not rely on events and instead uses methods to emit new states*.", ou seja, ele é uma solução mais simples para o gerenciamento de estados através de métodos, sendo um subconjunto do pacote *Business Logic Components* (BLoC); *Domain* é a camada principal que será responsável por validar e transformar os dados, executar a lógica de negócios, agrupar e identificar exclusivamente os dados pertencentes a classe Entidade; e *Infrastructure*, encarregada de administrar as conexões com banco de dados, *Application Programming Interface* (API) e sensores.

Figura 1 - Representação gráfica do DDD



Fonte: Rešetár (2020).

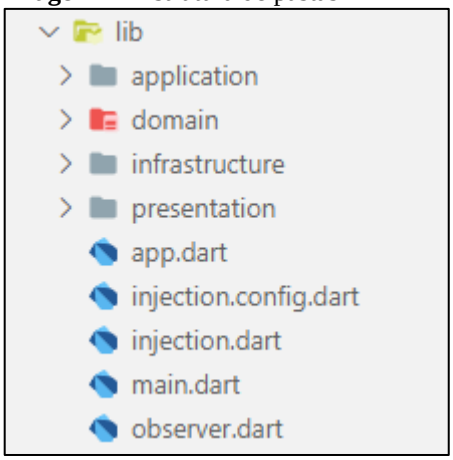
Também é necessário ter uma forma de armazenar as informações tanto das videoaulas como dos usuários, com o intuito de guardar esses dados foi utilizado o Firebase pois como é apresentado por Remessa Online (2021*)* "O Firebase funciona como um *Backend as a Service* (BaaS). Nesse sentido, ele oferece diversos recursos e ferramentas que permitem o desenvolvimento simples de aplicações.".

Dentre as diversas opções oferecidas foram escolhidas as seguintes ferramentas e recursos: *Firestore*, para consulta e armazenamento dos dados como o nome e *e-mail* de um usuário; *Storage*, buscando armazenar na nuvem os arquivos de vídeo e imagem; e *Authentication*, por fornecer serviços de autenticação, além de facilitar o gerenciamento dos usuários. Algumas empresas como Duolingo, Trivago e Gameloft utilizam essa plataforma do Google no desenvolvimento de suas aplicações (REMESSA ONLINE, 2021).

Além de escolher uma forma para armazenar os dados do aplicativo, é necessário selecionar uma maneira de compartilhar o código em desenvolvimento entre os membros da equipe, visto que cada membro irá trabalhar remotamente. Buscando solucionar este problema foi utilizado o GitHub como plataforma para hospedar o código. O Github trabalha em conjunto com o Git um sistema de controle de versão (VCS) permitindo a colaboração entre os desenvolvedores e ao mesmo tempo guarda registros das alterações possibilitando restaurar um código alterado ou removido caso necessário (LONGEN, 2022).

Com o *framework*, banco de dados e padrão de modelagem escolhidos inicia-se a construção de estrutura de pastas. Na Imagem 1 observa-se a estrutura de pastas criada de acordo com a separação de camadas propostas no modelo DDD, vale ressaltar que no Flutter a estruturação se inicia na pasta lib.

Imagem 1 - Estrutura de pastas



Fonte: Elaborado pelos autores (2022).

Após a construção da estrutura de pastas iniciou-se a criação das telas que irão compor o aplicativo. Nesta etapa, o DDD finalmente foi aplicado de fato, para acompanhar este processo do desenvolvimento e a aplicação do DDD foi utilizada na tela de treinamentos como exemplo. Começando pela pasta *domain*, foi definido as propriedades dos modelos e após as funcionalidades que serão requisitadas em cada tela. Na Imagem 2 é definido as propriedades do modelo dentro do arquivo *training_model.dart*, ou seja, as variáveis, seus tipos e se podem ou não serem vazias.

Imagem 2 - Propriedades do modelo de treinamento

```dart
@freezed
class TrainingModel with _$TrainingModel {
  const factory TrainingModel({
    String? id,
    required String category,
    @TimestampConverter() required DateTime createdAt,
    required String thumb,
    required String videoName,
    required String videoUrl,
  }) = _TrainingModel;

  factory TrainingModel.fromJson(String id, Map<String, dynamic> json) =>
      _$TrainingModelFromJson(json).copyWith(id: id);
}
```

Fonte: Elaborado pelos autores (2022).

A Imagem 3 contém os serviços que a tela de treinamentos utiliza para funcionar, por exemplo, a exibição das aulas disponíveis necessitando de um serviço que retorne os vídeos de treinamento cadastrados no banco de dados. Diante dessa necessidade foi criado um arquivo chamado *training_service.dart* contendo os métodos *getTrainings* e *getCategories* visando mostrar para o usuário as aulas separadas de acordo com sua categoria.

Imagem 3 - Serviços consumidos pela tela de treinamentos

```dart
import 'package:sales_platform_app/domain/training/category_model.dart';
import 'package:sales_platform_app/domain/training/training_model.dart';

abstract class TrainingService {
  Future<List<CategoryModel>> getCategories();
  Future<CategoryModel> getCategory(String id);

  Future<List<TrainingModel>> getTrainings({String? categoryId});
  Future<TrainingModel> getTraining(String id);
  Future<TrainingModel> getLatestTraining();

  Future<String> getMainVideoUrl({required String url});
}
```

Fonte: Elaborado pelos autores (2022).

Seguindo a estrutura DDD após definir as propriedades e serviços na camada de *domain* foi implementado a camada de *infrastructure*, local onde ocorre consultas e envio de informações para um Banco de dados ou API, o que a torna o *backend* da aplicação, ou seja, nesta camada foi feito a lógica dos serviços definidos anteriormente.

O exemplo de implementação da lógica do serviço *getCategories* na Imagem 4, o *getCategories* é uma função assíncrona, pois é necessário esperar que o sistema receba os dados alocados no Firestore antes de prosseguir. Pensando em tratar possíveis erros é feito um *try*

catch, sendo criado dentro do *try* duas variáveis a *docs* e *categories*, sendo a *docs* responsável por armazenar um objeto com todas as categorias registradas na coleção *categories* do Firestore. Essas informações são passadas através de um laço de repetição para a variável *categories* que será retornada como resposta desta função, porém caso ocorra um erro o retorno da função será o que está dentro do *catch*, neste caso retornando uma mensagem com o erro.

Imagem 4 - Lógica do serviço *getCategories*

```
14    @override
15    Future<List<CategoryModel>> getCategories() async {
16      try {
17        final docs = await _firestore.collection('categories').get();
18        final categories = <CategoryModel>[];
19        for (final doc in docs.docs) {
20          categories.add(CategoryModel.fromJson(doc.id, doc.data()));
21        }
22        return categories;
23      } catch (e, s) {
24        log("Error getting categories: $e");
25        log(s);
26        throw AppFailure("Ocorreu um erro ao buscar as categorias");
27      }
28    }
```

Fonte: Elaborado pelos autores (2022).

Com o *domain* e *infrastructure* feitos iniciou-se a construção da *application*, conectando as camadas e apresentando as informações presentes em cada tela do aplicativo. Na Imagem 5 encontra-se um trecho do arquivo *trainings_cubit.dart* localizado dentro da pasta *application* que consome os serviços getCategories e getTrainings da camada *infrastructure*, recebendo as informações definidas na Imagem 4 e estruturando de acordo com as necessidades da camada *presentation* com a finalidade de exibir da maneira planejada, neste caso, os treinamentos separados por categoria.

Imagem 5 - Consumindo o getTrainings do infrastructure

```
36    Future<void> getTrainings() async {
37      log('getTrainings...');
38      emit(const TrainingsState());
39      try {
40        emit(state.copyWith(isLoading: true));
41        final categories = await _trainingService.getCategories();
42        final trainings = await _trainingService.getTrainings();
43        log('got ${trainings.length} trainings and ${categories.length} categories');
44        final Map<String, List<TrainingViewModel>> trainingsByCategory = {};
45
46        // Make first category "Latest". Limit to 10 trainings
47        final latestCat = CategoryModel.latest();
48        trainingsByCategory[latestCat.id!] = trainings
49            .map((e) => TrainingViewModel(
50                  data: e.copyWith(category: latestCat.id!),
51                  categoryName: latestCat.name,
52                ))
53            .take(10)
54            .toList();
55        for (final training in trainings) {
56          for (final category in categories) {
57            if (training.category == category.id) {
58              trainingsByCategory.putIfAbsent(category.id!, () => []);
59              trainingsByCategory[category.id!]!.add(TrainingViewModel(
60                categoryName: category.name,
61                data: training,
62              ));
63            }
64          }
65        }
```

Fonte: Elaborado pelos autores (2022).

A finalização do desenvolvimento se deu na camada *application* onde foram criadas as telas de acordo com o *layout* planejado, recebendo os dados já coletados do Firebase e tratados para preencher os campos que o usuário irá interagir.

Com o código pronto ainda é necessário gerar o *Android Application Package* (APK) que pode ser feito manualmente através do comando *flutter build apk.* Também é possível a opção de melhorar e automatizar esse processo através do uso de *Continuous Delivery (CD)* do GitHub *Actions.* Conforme descrito por GitHub (2021) "*Continuous Delivery (CD) automatically delivers code changes to production-ready environments for approval*", ou seja, o CD é responsável por realizar automaticamente as entregas de mudanças no código para o ambiente de produção ou aprovação.

3 RESULTADOS E DISCUSSÃO

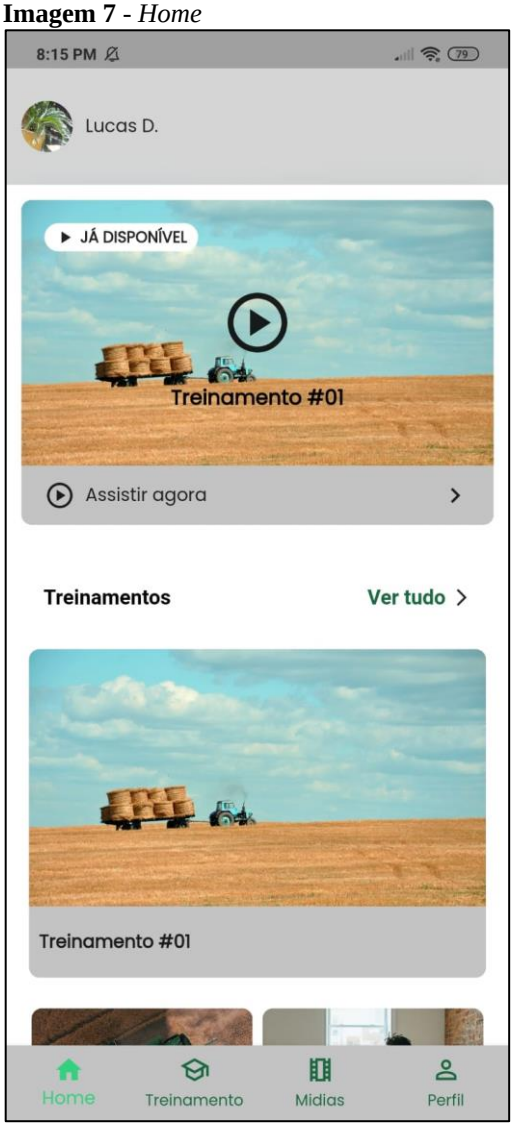
Todo o processo de construção do aplicativo resultou nas telas de *Login*, *Home*, Treinamentos, Assistir treinamentos e Perfil. Estas telas estão exibindo informações presentes no banco de dados, além de atualizarem suas informações caso ocorra alguma mudança nos dados como o vídeo ou nome de um treinamento, por exemplo. Na Imagem 6 é mostrada a primeira tela do aplicativo, após a empresa cadastrar o trabalhador este poderá realizar o *login* e caso necessário acessar a opção "Esqueci minha senha"

Imagem 6 - *Login*

Fonte: Elaborado pelos autores (2022)

Uma vez conectado ao abrir o ambiente de treinamento, o trabalhador encontrará a página inicial da aplicação onde ele poderá visualizar as últimas aulas postadas pela empresa e os vídeos em destaque. As videoaulas contidas na plataforma serão elaboradas pela própria empresa, sendo todo o conteúdo de responsabilidade dela. Na Imagem 7 é mostrado a versão final da tela *Home* e vale destacar que em todas as telas exceto a de *login*, "Redefinição de senha" e "Esqueci minha senha" possuem um menu de navegação na parte inferior.

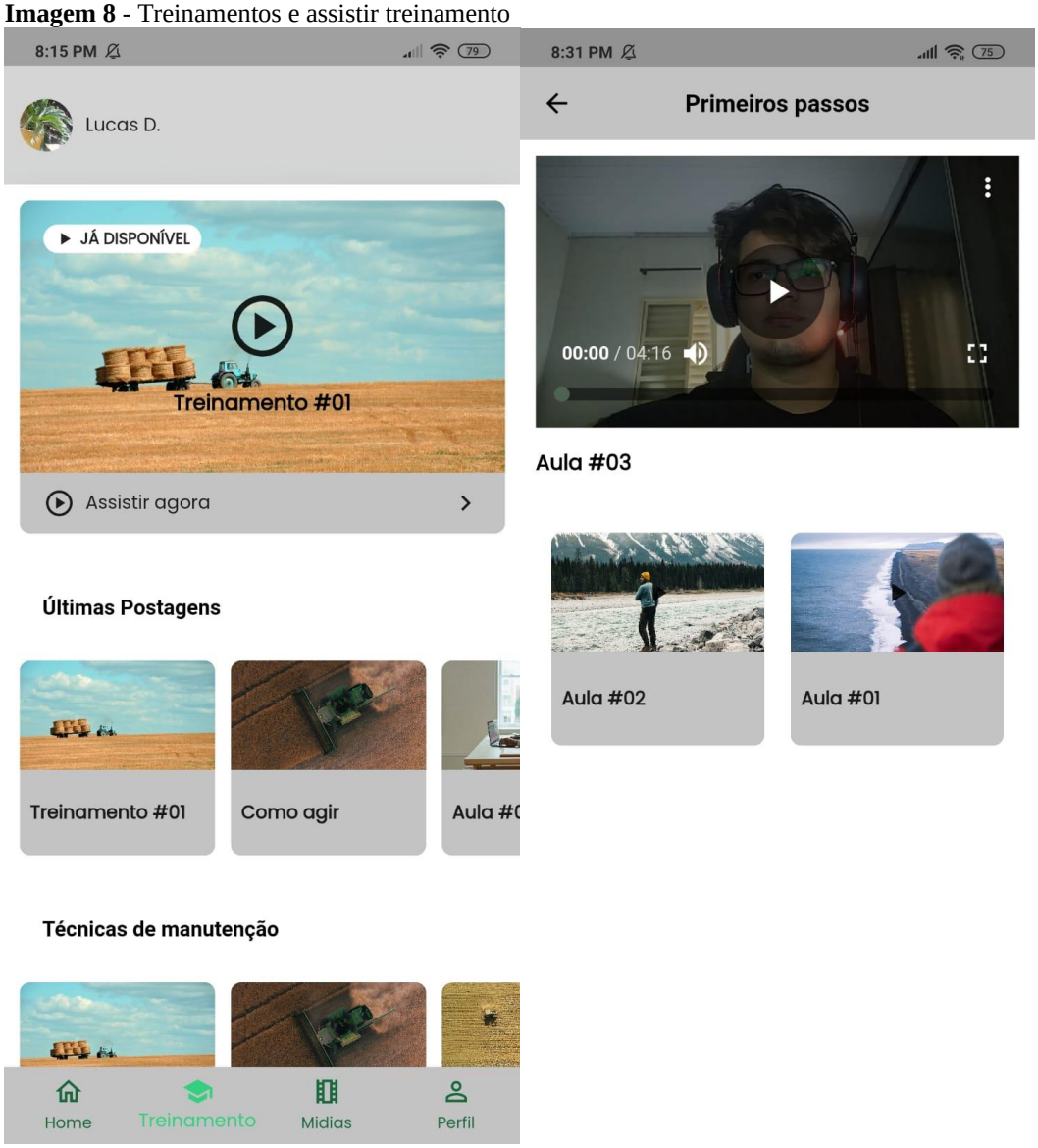
Imagem 7 - *Home*



Fonte: Elaborado pelos autores (2022)

A partir da *Home* ou menu na parte inferior é possível acessar uma das principais telas da aplicação e a tela de treinamentos onde se encontram todos os treinamentos disponibilizados pela empresa com as videoaulas separadas por categoria. A Imagem 8 retrata esta categorização das aulas na tela de treinamento onde também encontra-se a tela para acessar os vídeos, o local onde o usuário conseguirá assistir as videoaulas, além do *player* de vídeo onde é mostrado as outras aulas pertencentes à mesma categoria.

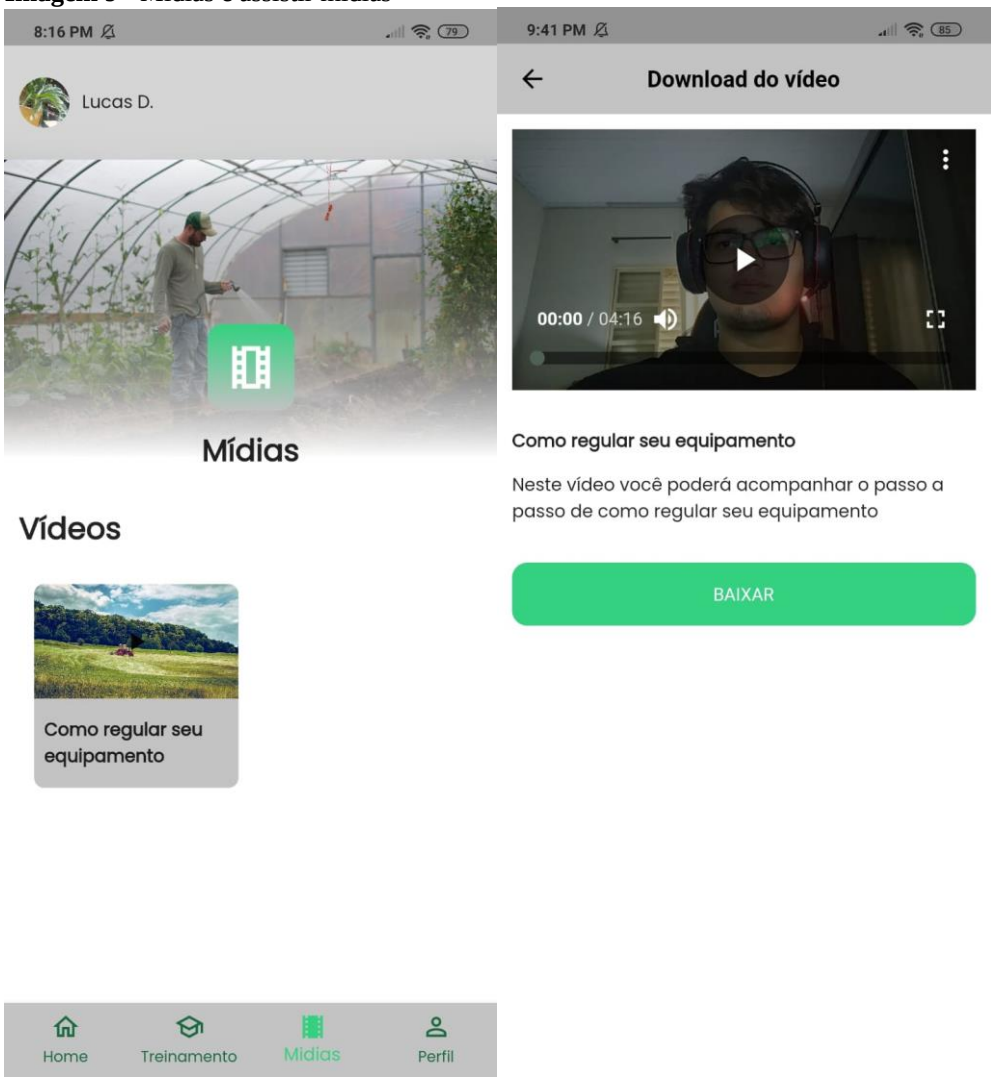
Imagem 8 - Treinamentos e assistir treinamento



Fonte: Elaborado pelos autores (2022)

A tela de mídias é muito importante para o *software*, pois as empresas as utilizarão para oferecer manuais e passo a passo em formato de vídeos curtos que poderão ser baixados possibilitando uma visualização *offline* do conteúdo. Os vídeos são curtos por terem a opção de *download*, para que não ocupem muito espaço na memória do dispositivo utilizado.

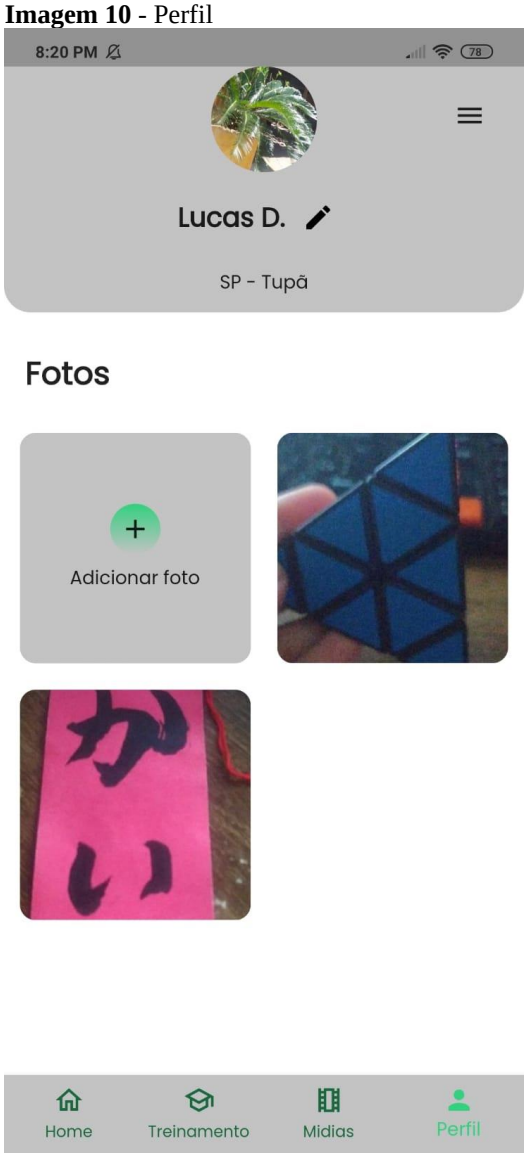
Imagem 9 - Mídias e assistir mídias



Fonte: Elaborado pelos autores (2022)

Foi construída a tela de perfil buscando trazer os dados do usuário cadastrado e possibilitar a personalização de seu perfil. A Imagem 10 demonstra o perfil do trabalhador exibindo seu nome, foto de perfil e a cidade em que reside. Essas informações podem ser alteradas ao clicar no ícone de lápis ao lado do nome, no canto superior direito encontra-se o menu de hambúrguer responsável por abrir mais opções como: *Logout*; Alterar os dados pessoais; Alterar senha; Acessar grupo no Telegram; e a central de ajuda. Ainda nesta tela é possível fazer *uploads* de fotos.

Imagem 10 - Perfil



Fonte: Elaborado pelos autores (2022)

4 CONCLUSÕES

Durante o desenvolvimento do aplicativo, a modelagem proporcionada pelo DDD mostrou-se eficiente na organização no projeto do aplicativo, evitando conflitos entre os arquivos manipulados pelos programadores e facilitando a manutenção. Utilizando a aplicação foi possível verificar a melhoria de desempenho devido ao uso do Flutter. A utilização do Firebase como BaaS mostrou-se altamente viável devido a sua facilidade de implementar funcionalidades como o sistema de autenticação por *e-mail*, dessa forma agilizando o desenvolvimento. O GitHub se provou essencial para o desenvolvimento por oferecer a hospedagem do código, fator importante tanto para salvar o código como o compartilhar entre os membros da equipe, além de gerar o APK através do CD do GitHub Actions, deixando-o disponível para download nas *releases*, vale ressaltar que as *releases* são as versões do projeto.

Através da implementação do aplicativo para a capacitação dos trabalhadores a empresa conseguirá reduzir custos, pois não precisará fazer locação de um espaço adequado, ter gastos com transporte, alimentação e materiais que se fazem necessários em treinamentos presenciais. E para os trabalhadores, as vantagens da utilização deste aplicativo se dão na comodidade de

poder assistir as videoaulas em qualquer momento e lugar, seguindo seu próprio ritmo de aprendizagem, sendo possível rever os treinamentos quando houver necessidade.

Visando agregar mais valor e funcionalidade ao produto desenvolvido existe a possibilidade da utilização do Google Analytics para coletar diversas informações sobre a interação do usuário com o *software*, desde acompanhar o caminho feito por ele dentro da aplicação ou até mesmo saber o dispositivo utilizado e o tempo médio de utilização, o que pode facilitar o monitoramento da empresa. Diante dessas possibilidades seria possível implementar funções para acompanhar o desempenho do trabalhador verificando quais treinamentos já foram assistidos.

REFERÊNCIAS

ALMEIDA, A. N.; BALSADI, O. V.; FERREIRA, B.; FREITAS, R. E;. **Ocupações agrícolas e não agrícolas: trajetória e rendimentos no meio rural brasileiro**. Disponível em: https://ageconsearch.umn.edu/record/146406. Acesso em: 28.ago.2022.

ANGELOV, F. **felangel/cubit**. Disponível em: https://github.com/felangel/cubit. Acesso em: 04.set.2022.

DEMEDYUK, IHOR; TSYBULSKYI, NAZAR. **Flutter vs Native vs React-Native: Examining performance.** 2020 Disponível em: https://medium.com/swlh/flutter-vs-native-vs-react-native-examining-performance-31338f081980. Acesso em: 12.set.2022.

FLUTTER. **Flutter - Beautiful native apps in record time**. Disponível em: https://flutter.dev/. Acesso em: 12.set.2022.

GITHUB, 2021. **Continuous Integration and Continuous Delivery (CI/CD) Fundamentals.** Disponível em: https://resources.github.com/ci-cd/. Acesso em: 24.out.2022.

LINKEDIN, 2018. **2018 Workplace Learning Report**. Disponível em: https://learning.linkedin.com/resources/workplace-learning-report-2018. Acesso em: 09.out.2022.

LONGEN, A. **O Que é GitHub e Para Que é Usado?**. Hostinger, 2022 Disponível em: https://www.hostinger.com.br/tutoriais/o-que-github. Acesso em: 02.out.2022.

MASOTTI, D. **Domain-Driven Design: guia básico sobre DDD**. Zup, 2022 Disponível em: https://www.zup.com.br/blog/domain-driven-design-ddd. Acesso em: 23.out.2022.

QUALIFICA, 2022. **Qualifica para empresas**. Disponível em: https://www.qualifica.com.br/qualifica/qualifica-para-empresas. Acesso em: 09.out.2022.

REMESSA ONLINE, 2021. **Firebase: descubra para que serve, como funciona e como usar - Remessa Online**. Disponível em: https://www.remessaonline.com.br/blog/firebase-descubra-para-que-serve-como-funciona-e-como-usar/#:~:text=O%20Firebase%20Authentication%20facilita%20o. Acesso em: 14.out.2022.

REŠETÁR, MATEJ. **Flutter Firebase & DDD Course [1] – Domain-Driven Design Principles**. Disponível em: https://resocoder.com/2020/03/09/flutter-firebase-ddd-course-1-domain-driven-design-principles/. Acesso em: 28.ago.2022.

ROCKETSEAT, 2022. **Evolua rápido como a tecnologia**. Disponível em: https://www.rocketseat.com.br/. Acesso em: 09.out.2022.

YUNES, C. **Acesso à internet no campo avança no Brasil, aponta pesquisa**. Jovem Pan, 2022. Disponível em: https://jovempan.com.br/programas/jornal-da-manha/acesso-a-internet-no-campo-avanca-no-brasil-aponta-pesquisa.html. Acesso em: 02.out.2022.