JOSÉ DANILO GUARIZZO NETO

DESENVOLVIMENTO DE APLICATIVO PARA DISPOSITIVOS MÓVEIS COM A TECNOLOGIA FLUTTER: UM ESTUDO DE CASO COM UMA PLATAFORMA DE NOTÍCIAS

LAVRAS – MG

2019

JOSÉ DANILO GUARIZZO NETO

DESENVOLVIMENTO DE APLICATIVO PARA DISPOSITIVOS MÓVEIS COM A TECNOLOGIA FLUTTER: UM ESTUDO DE CASO COM UMA PLATAFORMA DE NOTÍCIAS

Relatório técnico apresentado à Universidade Federal de Lavras, como parte das exigências do curso de Ciência da Computação, para a obtenção do título de Bacharel.

Prof. Neumar Costa Malheiros
Orientador

LAVRAS – MG

2019

Ficha catalográfica elaborada pela Coordenadoria de Processos Técnicos

da Biblioteca Universitária da UFLA

Neto, José Danilo Guarizzo

Desenvolvimento de aplicativo para dispositivos móveis com a tecnologia Flutter: um estudo de caso com uma plataforma de notícias / José Danilo Guarizzo Neto. 2^a ed. rev., atual. e ampl. – Lavras : UFLA, 2019.

49 p. : il.

Tese(graduação)–Universidade Federal de Lavras, 2019.

Orientador: Prof. Neumar Costa Malheiros.

Bibliografia.

1. TCC. 2. Monografia. 3. Dissertação. 4. Tese. 5. Trabalho Científico – Normas. I. Universidade Federal de Lavras. II. Título.

CDD-808.066

JOSÉ DANILO GUARIZZO NETO

DESENVOLVIMENTO DE APLICATIVO PARA DISPOSITIVOS MÓVEIS COM A TECNOLOGIA FLUTTER: UM ESTUDO DE CASO COM UMA PLATAFORMA DE NOTÍCIAS

Relatório técnico apresentado à Universidade Federal de Lavras, como parte das exigências do curso de Ciência da Computação, para a obtenção do título de Bacharel.

APROVADA em 5 de dezembro de 2019.

Prof. Dr. Raphael Winckler de Bettio UFLA
Bruno Lucinda Ribeiro CRIA Tecnologia e Inovação

Prof. Neumar Costa Malheiros
Orientador

LAVRAS – MG
2019

Dedico este trabalho a minha mãe, que sempre esteve, está e estará comigo.

AGRADECIMENTOS

Agradeço a minha família, em especial meus pais, Joseane e Evandro, por me apoiarem e acreditarem em mim, sempre fazendo o possível e o impossível para que eu conseguisse concluir está etapa. Agradeço a minha "família"de Lavras, Julia, Bruno e Victor e amigos, Gustavo, Ivan e Bruno, por viverem e compartilharem dessa fase comigo. Agradeço ao professor Neumar, pela oportunidade deste projeto, orientação e conhecimento passado.

RESUMO

A quantidade de usuários que utilizam dispositivos móveis para acesso a serviços e aplicações na Internet está cada vez maior. Desta forma, o desenvolvimento de aplicativos para dispositivos móveis é uma área muito relevante. Há diversos desafios no projeto e implementação de soluções para dispositivos móveis, por exemplo, a heterogeneidade de dispositivos e plataformas. As diferenças entre os dispositivos e suas linguagens nativas tornam difícil a implementação de aplicativos e sua portabilidade. Neste trabalho, foi desenvolvido um aplicativo para uma plataforma de notícias na Web. O aplicativo foi implementado com uma tecnologia denominada Flutter e ele interage com um serviço web RESTful para acessar os dados da plataforma. Este relatório contemplará as atividades de projeto e implementação do aplicativo.

Palavras-chave: Aplicativos. Flutter. RESTful.

ABSTRACT

The number of users using mobile devices to access services and applications on the Internet is increasing. Thus, mobile app development is a very relevant area. There are several challenges in designing and implementing mobile device solutions, for example, device and platform heterogeneity. Differences between devices and their native languages make application deployment and portability difficult. In this work, we have implemented a mobile application as part of a web news platform. The application was developed using a technology called Flutter and it interacts with a RESTFul web service to access the platform data. This report will cover the design and implementation of the mobile application.

Keywords: Mobile application. Flutter. RESTful.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

O mundo está cada vez mais conectado e a utilização de dispositivos móveis para acesso a serviços e aplicações na Internet cresce cada vez mais. Em 2018, houve duas vezes mais acessos à Internet por dispositivos móveis do que via computadores (CETIC.BR, 2018). Portanto, é de grande importância que exista uma alternativa para que as aplicações disponíveis na Web possam ser acessadas de forma conveniente e eficiente a partir de dispositivos móveis.

Embora as aplicações web responsivas possam ser acessadas pelo navegador web nos dispositivos móveis, em muitos casos, é essencial o desenvolvimento de um aplicativo próprio para esses dispositivos, devido a sua praticidade e desempenho. Neste cenário de expansão do mercado de aplicativos para dispositivos móveis, as empresas têm o objetivo de aumentar ou fidelizar um número maior de clientes por meio das facilidades de um aplicativo próprio para os dispositivos móveis mais populares, como telefones celulares e *tablets*. Porém, há diversos desafios perante o desenvolvimento de um aplicativo. Deve ser levado em consideração o número de plataformas abrangentes, custo e tempo de desenvolvimento e custo de manutenção. A experiência do usuário ao utilizar o produto final também é um desafio, como a fluidez da utilização, consumo de rede, consumo de bateria, entre outros.

Para lidar com a heterogeneidade de plataformas para dispositivos móveis, uma abordagem comum no mercado é o desenvolvimento multi-plataforma. Essa abordagem consiste em desenvolver um aplicativos para diversas plataformas desejadas utilizando apenas um ambiente de desenvolvimento.

Atualmente, uma das tecnologias mais importantes de desenvolvimento multi-plataforma é o Flutter. O objetivo deste trabalho foi utilizar a tecnologia Flutter no desenvolvimento de um aplicativo para uma sistema Web de canais de notícias. Esse sistema, denominado ColabNews [1], é uma aplicação de gerenci-

[1] https://www.colabnews.com.br/

amento de canais de notícias. Ela permite ao usuário criar canais, enviar e ler notícias, priorizar as notícias de canais favoritos, curtir notícias, entre outros. Este relatório abordará as atividades de projeto e desenvolvimento do aplicativo da plataforma ColabNews utilizando o Flutter.

Este relatório tem a seguinte organização. O Capítulo 2 apresenta o referencial teórico, que aborda os conceitos básicos sobre o desenvolvimento multiplataforma de aplicações para dispositivos móveis, o framework Flutter e o estilo arquitetural REST. O Capítulo 3 descreve as atividades de projeto e implementação, como o consumo de RESTful API e a utilização de recursos nativos utilizando Flutter. Por fim, o Capítulo 4 apresenta comentários e considerações finais sobre o projeto e a utilização do Flutter em seu desenvolvimento.

2 REFERENCIAL TEÓRICO

Neste capítulo, são descritos os principais conceitos e ferramentas utilizadas para o desenvolvimento desse projeto, como os desafios e abordagens no desenvolvimento de aplicativo para dispositivos móveis e uma visão geral sobre o Flutter e sua estrutura. Este capítulo também aborda HTTP e a utilização da arquitetura REST.

2.1 Desenvolvimento de Aplicativos Multi-Plataforma

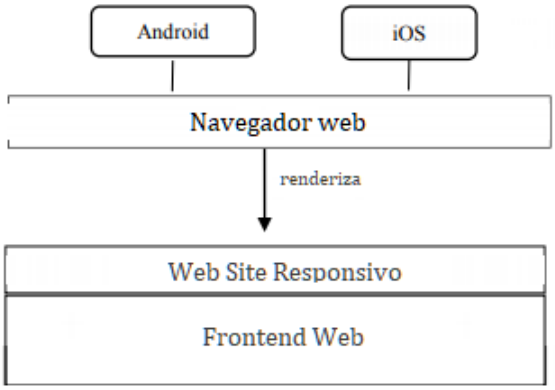
O desenvolvimento para dispositivos móveis envolve vários desafios como a heterogeneidade dos dispositivos, seja por parte do sistema operacional ou pelo tamanho dos dispositivos, capacidade de processamento, memória, disponibilidade de rede, manutenção, entre outros. Atualmente, os dois principais sistemas operacionais para dispositivos móveis são o Android e o iOS. No desenvolvimento de aplicações para esses dispositivos, deve ser utilizado o SDK (kit de desenvolvimento de software) específico para cada plataforma. Neste caso, para desenvolver um aplicativo que esteja disponível para ambos os sistemas operacionais, seria necessário a utilização de dois SDKs diferentes, com suas respectivas linguagens nativas, Java e Swift.

É de se esperar que as empresas que utilizam essa forma de desenvolver, tenham duas equipes especializadas, uma para cada plataforma, aumentando assim o custo e o tempo do projeto. Como forma de amenizar este problema, existe a metodologia de desenvolvimento multi-plataforma (EL-KASSAS et al., 2017). Esta metodologia tem por finalidade utilizar apenas um código fonte para gerar um aplicativo para cada plataforma. Existem diferentes abordagens de desenvolvimento multi-plataforma, entre as quais pode-se destacar: web, híbrida, interpretada e multi-compilada (LATIF et al., 2016).

2.1.1 Abordagem Web

A abordagem web é baseada em sites responsivos, que utilizam o próprio navegador web do dispositivo para disponibilizar seu conteúdo. Esta abordagem é limitada pelo fato de não ter acesso a recursos nativos do dispositivo. Além disso, o dispositivo só consegue executar a aplicação mediante disponibilidade de acesso à Internet. Esse modelo é explicado no esquema apresentado na Figura 2.1.

Figura 2.1 – Abordagem web

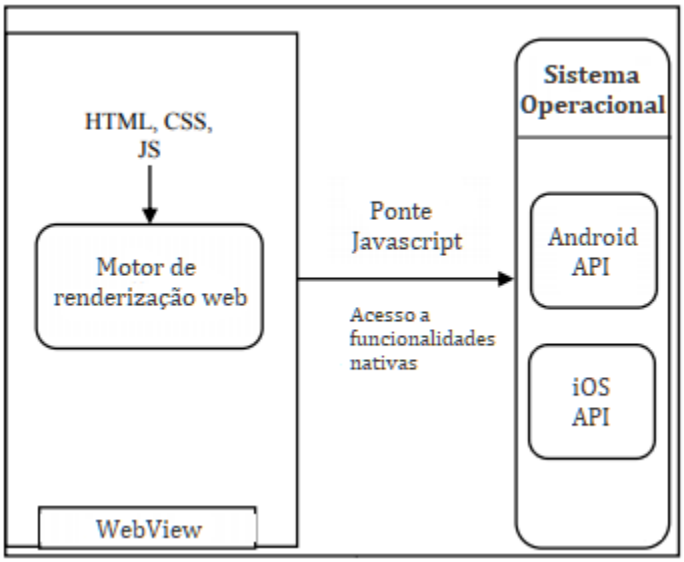


Fonte: Latif et al. (2016), adaptado pelo autor.

2.1.2 Abordagem Híbrida

A abordagem híbrida mistura o desenvolvimento web com funcionalidades nativas do sistema dos dispositivos. Neste caso, a interface de usuário dos aplicativos utiliza uma WebView para ser renderizada e as funções nativas ficam acessíveis por uma API em Javascript. Diferente da abordagem web, que necessita do navegador web para disponibilizar a aplicação, a abordagem híbrida disponibiliza esse acesso através de um aplicativo. Porém, por utilizarem do motor do navegador web, são menos performáticos, o que acarreta numa sensação menos fluida e nativa para os usuários, se comparado aos aplicativos nativos. Essa abordagem é ilustrada na Figura 2.2.

Figura 2.2 – Abordagem híbrida

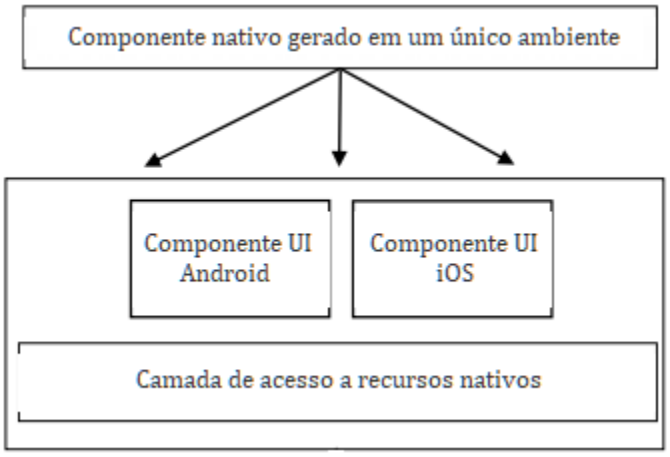


Fonte: Latif et al. (2016), adaptado pelo autor.

2.1.3 Abordagem Interpretada

A abordagem interpretada tem por finalidade usar uma linguagem de programação para implementar uma interface de usuário e gerar o componente nativo equivalente de cada plataforma. As funcionalidades nativas são acessadas por uma camada abstrata que interpreta o código em tempo de execução. Esse modelo é descrito no esquema da Figura 2.3.

Figura 2.3 – Abordagem interpretada



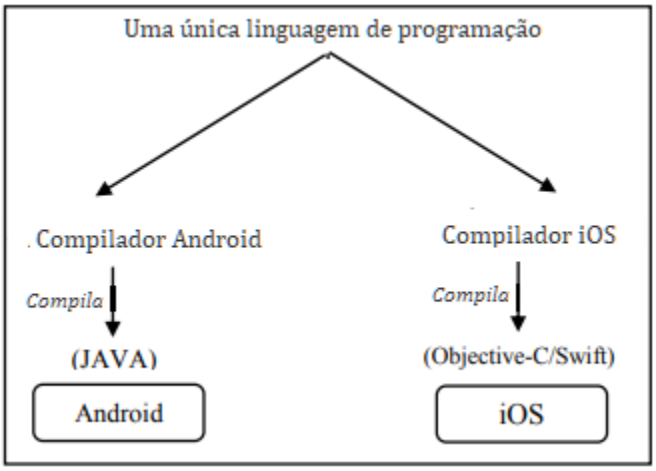
Fonte: Latif et al. (2016), adaptado pelo autor.

2.1.4 Abordagem Multi-compilada

A abordagem multi-compilada utiliza uma linguagem de programação específica para implementar o aplicativo. Este único código fonte é compilado em códigos nativos de cada plataforma.

O benefício desta abordagem é obter uma performance melhor e acesso abrangente aos recursos nativos nas aplicações. A desvantagem é que alguns recursos específicos de cada plataforma, como por exemplo o acesso a uma câmera, não pode ser utilizado, por diferirem entre as plataformas. Entretanto, este problema pode ser solucionado por APIs que encapsulem o comportamento de cada recurso em sua respectiva plataforma. Na Figura 2.4, é ilustrada a abordagem multi-compilada.

Figura 2.4 – Abordagem multi-compilada



Fonte: Latif et al. (2016), adaptado pelo autor.

2.2 Flutter

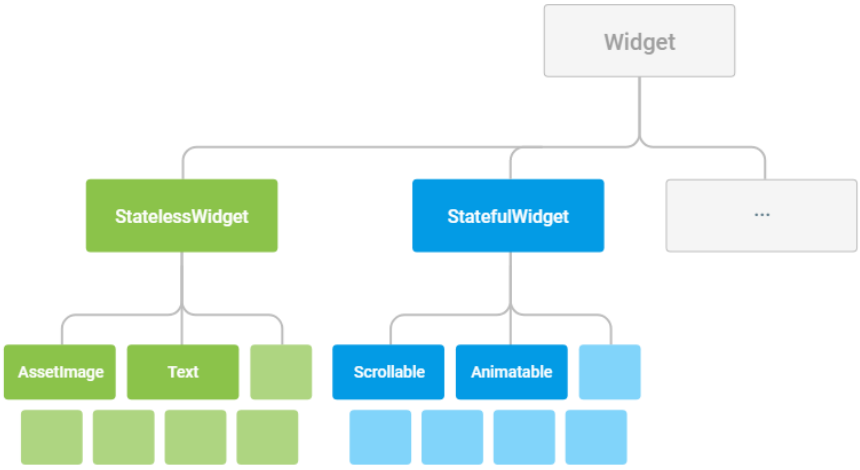
Flutter é um SDK desenvolvido pela Google, que utiliza a abordagem multi-compilada para construir aplicativos de alta performance e fidelidade para as plataformas Android e iOS, a partir um único código fonte (FLUTTER, 2019a).

O Flutter utiliza a linguagem Dart, que foi desenvolvida pela Google. Dart é uma linguagem orientada a objeto e tipada, que utiliza verificação de tipo estático e em tempo de execução para garantir a integridade de tipos. Porém, caso haja a necessidade da flexibilidade das linguagens dinâmicas, há a disponibilidade do tipo de variável dinâmica. Dart foi escolhida pelo Flutter por ser considerada uma linguagem de alta performance, ter capacidade multi-plataforma, alta e consistente performance, ter um eficiente alocador de memória, *hot reload* e baixa curva de aprendizado (FLUTTER, 2019b).

Diferentemente de outros frameworks que têm *views, controllers, layouts* e outras propriedades, a interface de usuário do Flutter é composta de Widgets. Os Widgets estabelecem um modelo de objeto unificado. Eles podem ser desde pequenos componentes como um texto, estilos de outros Widgets (cor, margem e

outros), ou até mesmo uma tela inteira (composta por outros Widgets). O Flutter garante a customização e criação de Widgets utilizando herança e agrupamento. Além disso, há vários Widgets padrões disponibilizados pelo framework. A Figura 2.5 demonstra a base da hierarquia dos Widgets do Flutter.

Figura 2.5 – Hierarquia dos Widgets



Fonte: Flutter (2019a).

Existem dois tipos de Widgets, os sem estado e os com estado. Como o nome sugere, eles se diferem pela ausência ou não de estado.

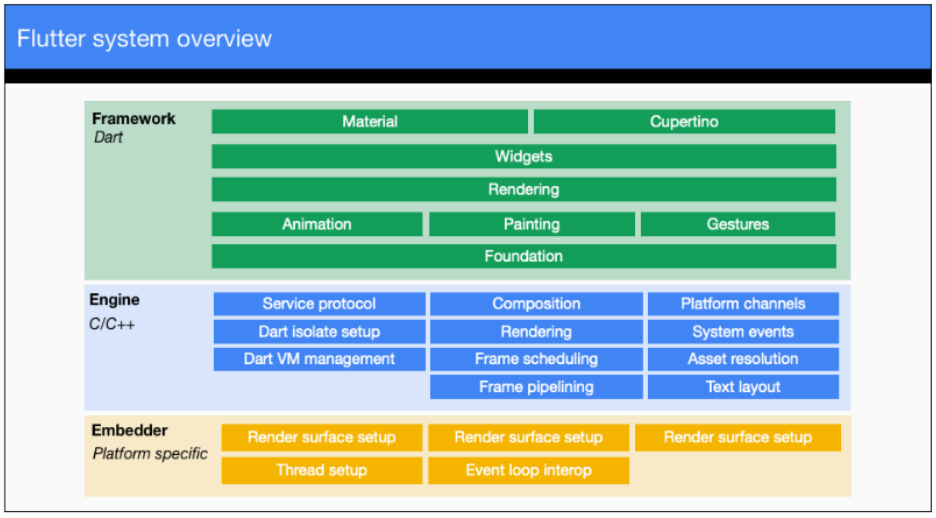
- Widgets sem estado não sofrem mudanças, são Widgets mais simples, como uma foto estática ou um texto.
- Widgets com estado, em instância também são imutáveis, mas guardam sua parte mutável em um objeto de estado.

O Flutter tem uma função responsável pela renderização dos Widgets. A função é chamada toda vez que o Flutter é notificado que há alguma mudança no valor de uma variável de estado. O desempenho não é afetado devido à disposição em árvore dos Widgets, causando uma atualização apenas na sub-árvore pertencente àquele Widget. Dessa forma, atualizações no estado de nós folha são menos

custosas, além de que Widgets constantes são mantidos, sendo apenas reutilizados. Um Widget do tipo constante têm seus estados totalmente determinados em tempo de compilação e imutável em tempo de execução.

O Flutter é organizado em uma série de camadas, de maneira que cada camada é baseada na camada anterior. Esse modelo em camadas é mostrado na Figura 2.6. As camadas mais acima geralmente serão mais utilizadas por serem componentes de alto nível com funcionalidades das camadas abaixo implementadas, seguindo a meta de design: *"fazer mais com menos código"* (FLUTTER, 2019a).

Figura 2.6 – Camadas do Flutter



Fonte: Flutter (2019a).

Um dos pontos que transformam o desenvolvimento em Flutter mais produtivo e rápido, é o suporte a *hot reload*. Este mecanismo injeta as mudanças do código fonte diretamente na máquina virtual que está executando o aplicativo, fazendo com que o Flutter refaça a árvore de Widgets com as mudanças necessárias, sem perder os estados atuais, mantendo o fluxo de execução do aplicativo. Isso contribui para maior produtividade no desenvolvimento de aplicativos.

2.3 RESTful API

2.3.1 HTTP e Serviços Web

HTTP (HyperText Transfer Protocol) é o protocolo que especifica a comunicação na Web (GOURLEY et al., 2002). O HTTP situa-se na camada de aplicação e utiliza do protocolo TCP para o transporte dos dados.

A comunicação é feita de forma cliente-servidor, na qual os conteúdos ou recursos estão localizados em um servidor HTTP. Essa comunicação é feita através de diferentes mensagens HTTP. A mensagem que parte do cliente é denominada requisição HTTP e transporta informações sobre o recurso requerido. Quando recebida pelo servidor, este retorna os dados em uma mensagem denominada resposta HTTP.

As mensagens HTTP consistem de três partes:

- Linha de começo, que para requisições HTTP indica qual o método, URI e versão do HTTP e para respostas HTTP indica a versão do HTTP e código da resposta.

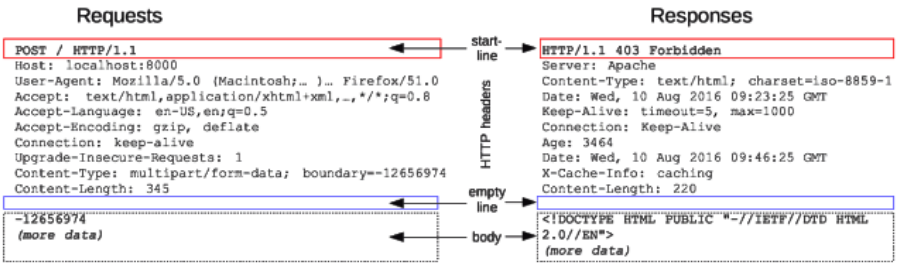
- Cabeçalho, que tem informações da mensagem HTTP, como tipo de dado, conteúdos aceitos, línguas aceitas, entre outros.

- Corpo, que contém os dados que serão enviados pelo cliente, ou dos dados retornados pelo servidor, no caso da resposta HTTP. Estes dados podem ser de diversos tipos, como imagens, textos simples e linguagem de marcação. Nem todos os métodos HTTP utilizam do corpo da mensagem, como é o caso do GET e DELETE.

Na Figura 2.7, é apresentado um exemplo dessas mensagens, onde há uma requisição POST para a url "localhost:8000/". O servidor devolveu uma resposta com código "403 - Forbidden", demonstrando que o servidor recebeu a requisição, mas se recusa a autorizá-la. Neste exemplo, pode-se determinar que a requisi-

ção foi feita de um navegador web Mozilla de um Macintosh, devido ao item de cabeçalho "User-agent".

Figura 2.7 – Exemplo de requisição e resposta HTTP



Fonte: MDN (2019)

O HTTP suporta diferentes tipos de requisições, os quais são chamados de métodos. Toda mensagem HTTP tem um método, e este, especifica ao servidor qual ação deve ser realizada. A Tabela 2.1 demonstra alguns dos principais métodos HTTP disponíveis.

Tabela 2.1 – Métodos HTTP

Método	Descrição
GET	Retorna um recurso do servidor para o cliente.
POST	Envia um recurso ao servidor.
PUT	Atualiza um recurso do servidor.
DELETE	Exclui um recurso do servidor.

Fonte: Autor

Toda resposta HTTP tem um código de estado. Os estados são números de três dígitos e estão divididos em cinco classes (FIELDING; RESCHKE, 2014):

- 1xx (Informativo), informa que a requisição foi recebida mas ainda está sendo processada.

- 2xx (Sucesso), informa que a requisição foi recebida e aceita.

- 3xx (Redirecionamento), informa que ações devem ser tomadas para completar a requisição.

- 4xx (Erro no cliente), informa que a requisição tem erros de sintaxe ou não pode ser completada.

- 5xx (Erro no servidor), informa que o servidor falhou ao completar a requisição.

A Tabela 2.2 contém alguns exemplos de códigos de estado HTTP, com a respectiva descrição.

Tabela 2.2 – Exemplos de códigos de estado HTTP

Código	Descrição
200	OK. Requisição bem sucedida.
400	Bad Request. Requisição rejeitada pelo servidor por haver um erro do cliente.
403	Forbidden. Servidor entendeu a requisição mas não autoriza.
404	Not Found. O servidor não achou o recurso alvo.
405	Method Not Allowed. O método da requisição não está disponível no recurso alvo.
500	Internal Server Error. Servidor encontrou condições inesperadas que impedem de completar a requisição.

Fonte: Autor

2.3.2 Estilo arquitetural REST

REST (Representational State Transfer) é um estilo arquitetural para sistemas Web distribuídos (FIELDING; TAYLOR, 2000). Ele foi definido perante algumas restrições no âmbito de comunicação entre componentes de aplicações distribuídas e manipulação de dados.

A primeira restrição do estilo é a separação da interface do usuário dos dados e controles do sistema, seguindo o estilo cliente-servidor, facilitando a portabilidade e escalabilidade do sistema.

Outra restrição importante é manter a comunicação sem estado, sendo assim, cada requisição feita pelo cliente devem conter todas as informações necessárias para que possa ser compreendida pelo servidor. Isso resulta em um menor acoplamento de requisições e liberação de recursos mais rápida, portanto, contribui para melhor desempenho do sistema.

Respostas podem ser rotuladas como cacheáveis, dando o direito do cliente reutilizar os dados, melhorando a eficiência das requisições e uso dos recursos do servidor.

No modelo REST, as informações são abstraídas como "recursos". Qualquer informação que pode ser nomeada pode ser um recurso: um documento, um serviço ou uma coleção de outros recursos, por exemplo. Neste caso, pode-se utilizar a representação textual do recurso, para padronizar o seu consumo. Isto pode ser alcançado, por exemplo, com a utilização de JSON[1] (Javascript Object Notation). Como o REST é construído nos conceitos de HTTP, recursos são identificados por meio de um URL, e as interações disponíveis com o recurso são definidas pelos métodos HTTP, como exemplificado no Quadro 2.1. Neste exemplo, o recurso notícia é identificado no servidor através da rota api/noticia e pode-se manipular este recurso utilizando os métodos HTTP. Como se trata da coleção de recursos notícias, rotas que trabalham com uma notícia específica necessitam utilizar como parâmetro o identificador da notícia correspondente. A Figura 2.8 apresenta um exemplo de uma requisição para consumo desse recurso e os dados da resposta no formato JSON.

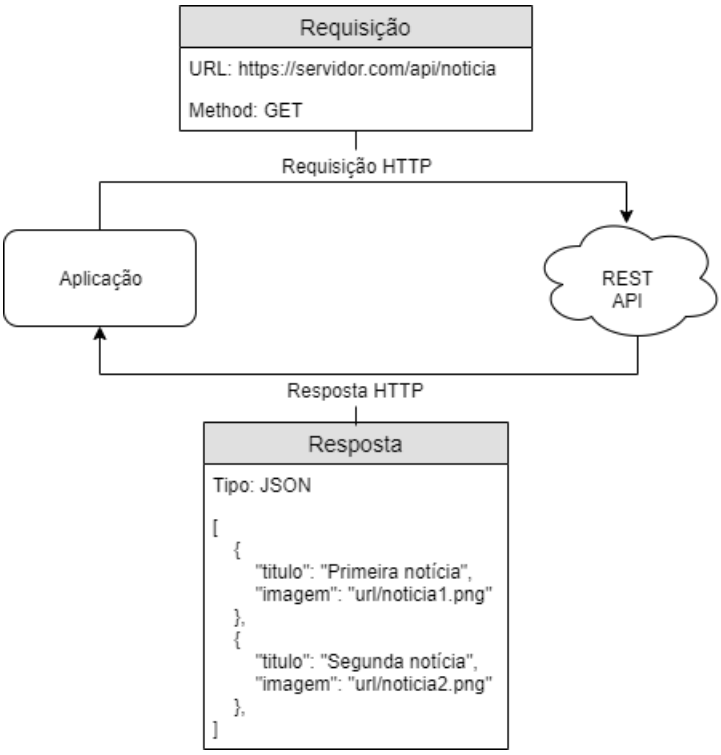
[1] https://www.json.org/

Quadro 2.1 – Exemplo de representação de recurso REST

Identificação/URL	Método HTTP	Descrição
api/noticia	GET	Retorna todas as notícias
api/noticia	POST	Cria uma notícia
api/noticia/:id	PUT	Atualiza a notícia pelo id
api/noticia/:id	DELETE	Deleta a notícia pelo id

Fonte: Autor

Figura 2.8 – Exemplo de consumo de recurso REST



Fonte: Autor.

3 ATIVIDADES REALIZADAS

Neste capítulo, são descritas as principais atividades realizadas no aplicativo da plataforma ColabNews. A plataforma ColabNews tem como finalidade reunir canais de notícias em um único sistema. Ela possibilita publicar notícias, criar e gerenciamento de canais de notícias, classificar canais como favoritos, curtir notícias, entre outros. O desenvolvimento da plataforma é o MVP de uma startup em programa de pré aceleração. Não foi feito um documento formal de análise de requisitos, apenas um protótipo de alta fidelidade.

A Figura 3.1 apresenta uma visão geral da arquitetura da plataforma ColabNews. A plataforma é composta pelos seguintes componentes:

- API RESTful desenvolvida com Node.js, sobre o serviço AWS Lambda;
- AWS API Gateway, receptor e orquestrador de requisições à API;
- Banco relacional PostgreSQL provido pelo serviço AWS RDS;
- Repositório de imagens provido pelo serviço AWS S3;
- Aplicação Web desenvolvida com Vue.js;
- Aplicativo desenvolvido com Flutter.

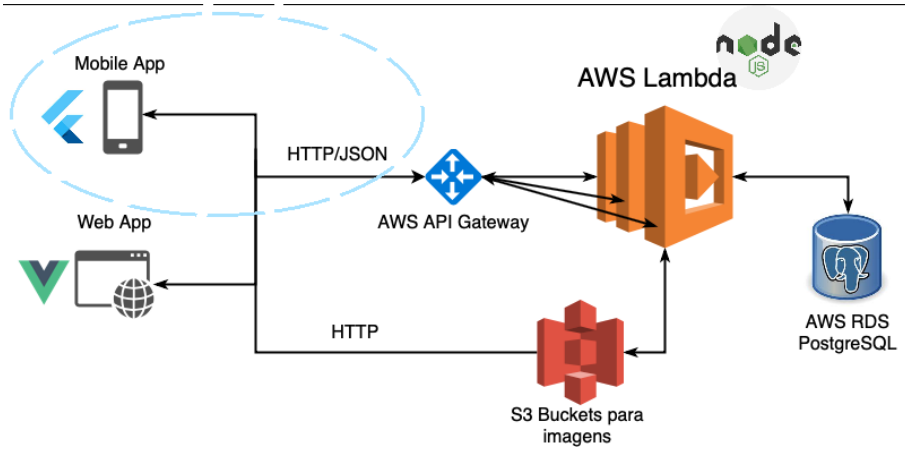
Este trabalho é referente ao aplicativo de dispositivo móvel e sua comunicação com a RESTful API, destacados em azul na Figura 3.1. O modelo de dados e a arquitetura do sistema foram feitas por outros integrantes da equipe da plataforma.

3.1 Estrutura do aplicativo

O aplicativo foi estruturado em torno de quatro partes:

- As telas, que são as telas do aplicativo, responsáveis pela renderização da interface do usuário, apresentação dos dados e chamada dos serviços;

Figura 3.1 – Esquema da arquitetura da plataforma ColabNews



Fonte: Autor

- Os componentes, que são Widgets que compõem as telas, tem as mesmas funcionalidades das telas, mas não existem por si próprios e não há navegação para os mesmos.

- Os serviços, que são as classes responsáveis pela comunicação com a API, responsáveis pelo retorno dos recursos para as telas.

- Os modelos, que são as classes responsáveis por encapsular os dados no sistema, contam com métodos de *parse* de JSON, para transformação de dados recebidos da API em objetos do sistema.

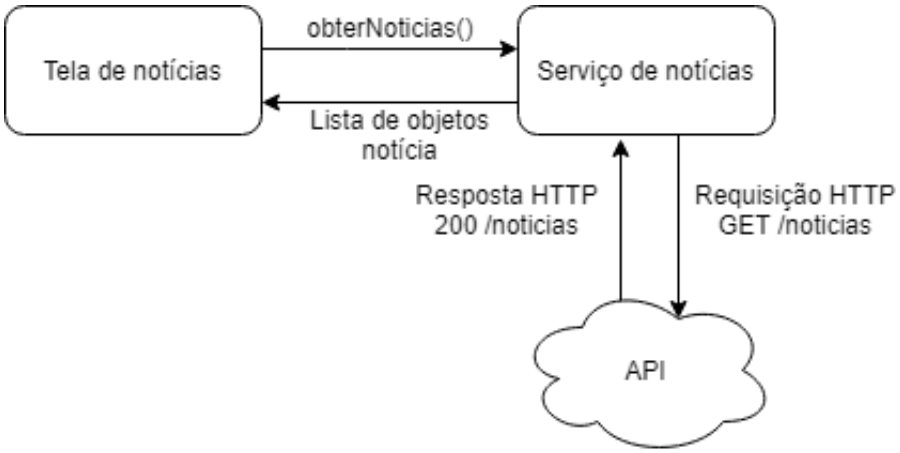
3.2 Requisições HTTP

Para facilitar o consumo dos recursos da API, foi utilizada a biblioteca "http"da linguagem Dart. Essa biblioteca fornece funções de alto nível para realizar requisições HTTP.

Para melhor organização, foram implementadas classes de serviço para cada recurso da API. Estas classes são responsáveis pelo controle das requisições e respostas HTTP, o que torna as requisições centralizadas e abstraídas nas telas.

Na Figura 3.2, é apresentado um exemplo da utilização de um método da classe de serviço de notícias para obtenção de notícias que serão apresentadas para o usuário. Neste exemplo, a tela chama a função de obtenção de notícias do serviço e fica esperando por uma lista de notícias. O serviço realiza a requisição para a API e utiliza os dados da resposta para retornar uma lista de notícias para a tela. Neste caso, retornar o objeto pronto para a tela ajuda na abstração da requisição, porém atrapalha na notificação visual do usuário caso a requisição falhe.

Figura 3.2 – Utilização de uma classe de serviço



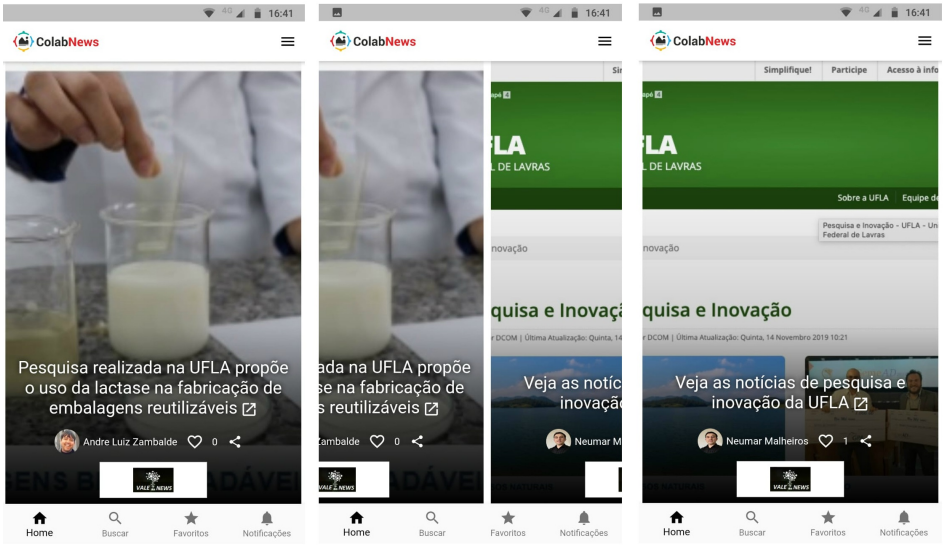
Fonte: Autor

3.3 Visualização de notícias

A principal funcionalidade do aplicativo é a visualização das notícias, sejam elas na página inicial (notícias de todos os canais), na página de um canal específico (contendo apenas as notícias do mesmo), ou na tela de busca (apenas com as notícias que correspondem ao filtro de busca). As notícias são dispostas em forma de um carrossel e a navegação é feita com a ação de deslizar para o lado, como demonstrado na Figura 3.3.

Para evitar duplicação de código devido a utilização em diversas telas, o carrossel foi desenvolvido como um componente. Este componente permite mu-

Figura 3.3 – Demonstração do carrossel no aplicativo



Fonte: Autor

dar a disposição de seus conteúdos (título, autor e logo do canal) e é responsável pelo consumo do recurso "notícia"na API. O componente permite a obtenção das notícias de três formas:

- Obter todas as notícias;
- Obter notícias que contenham uma string de busca.
- Obter apenas as notícias de um canal;

A diferenciação da obtenção das notícias é feita através da utilização de um parâmetro de busca nas duas primeiras formas e na utilização de uma rota diferente na terceira.

3.4 Autenticação de usuário

Algumas rotas da API estão disponíveis apenas para usuários que efetuaram autenticação no sistema. A autenticação é feita no backend via JWT (JSON Web Token), uma forma compacta e segura de transmissão de dados entre partes,

codificada como um objeto JSON (JONES; BRADLEY; SAKIMURA, 2015). O token é recebido no consumo das rotas de cadastro ou login e salvo em uma variável global. Este token é enviado no cabeçalho das requisições que necessitam de autenticação.

Como há a opção do usuário manter-se logado na plataforma, foi implementada uma solução utilizando a biblioteca "shared preferences". Essa biblioteca agrega as funções nativas NSUserDefaults (iOS) e SharedPreferences (Android), que proveem persistência de dados simples. Assim, o token é salvo tanto na variável global, quanto no dispositivo e sempre que o aplicativo é inicializado, verifica-se a existência do token.

Caso o usuário não queira mais permanecer autenticado, foi implementada uma função para deletar este token do dispositivo e chamar o serviço de notificação do backend para registrar que o usuário já não está mais utilizando o token em questão.

A persistência dos dados com o sharedPreference dispõe de funções de set, get e remove para os tipos bool, int, double e String. O armazenamento desses dados primitivos segue o padrão chave e valor. A Figura 3.4 demonstra a utilização desta biblioteca na persistência do token, onde a função getToken tem a função de recuperar o token da memória, a setToken de salvá-lo e a função logout de excluí-lo.

3.5 Enviar notícias

A plataforma ColabNews também permite que um usuário possa enviar uma notícia para um canal. Se aprovada pela equipe de moderação, a notícia será publicada no canal. Cada notícia tem os seguintes atributos:

- Título, um pequeno texto que explica o assunto da notícia.
- Imagem, uma foto que represente a notícia;

Figura 3.4 – Exemplo da utilização do SharedPreference

```
Future<String> getToken() async{
  await SharedPreferences.getInstance().then((session){
      String token = session.getString(_token);
      return token;
  };
}

void setToken(String token) async{
  await SharedPreferences.getInstance().then((session){
    session.setString(_token, token);
  });
}

void logout() async{
  await SharedPreferences.getInstance().then((session){
    session.remove(_token);
  });
}
```

Fonte: Autor

- Url externa, uma url para acesso da notícia completa;
- Autor, usuário responsável pelo envio da notícia;
- Canal, identificador do canal que a notícia pertence;
- Número de curtidas, representa o número de usuários que gostaram da notícia.

Para a implementação do envio de notícias, foram utilizadas duas bibliotecas: "image picker"e "image cropper". A primeira inclui as funções nativas de acesso à galeria de fotos e à câmera do dispositivo para ambas as plataformas. Esta biblioteca também fornece compressão de imagem, que no cenário de aplicativos para dispositivos móveis é de extrema importância, pelo consumo de dados e velocidade das redes móveis.

A segunda biblioteca faz comunicação com duas bibliotecas nativas de corte e rotação de imagem, uCrop e TOCropViewController. Esta biblioteca tam-

bém disponibiliza a interface de ajuste de imagens, como demonstra a Figura 3.5. No escopo deste projeto, foi necessário a implementação desta função, pois era necessário que todas as imagens fossem padronizadas na proporção 16:9.

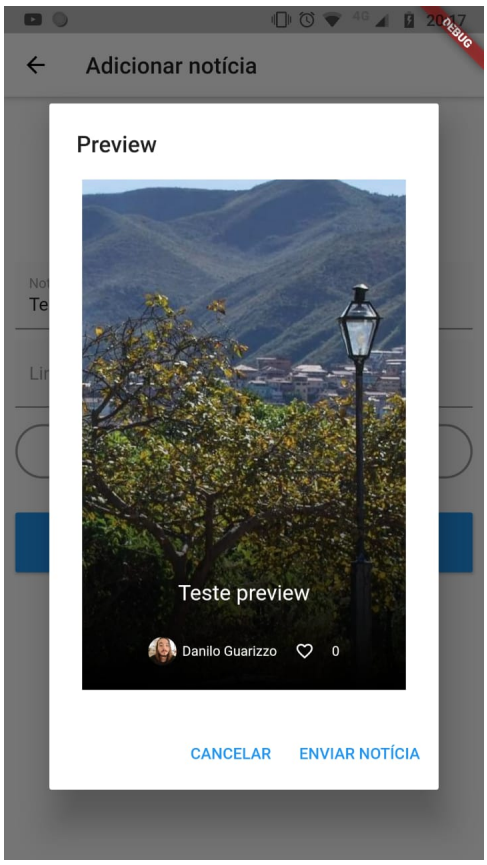
Figura 3.5 – Interface de ajuste de imagem

Fonte: Autor

Com estas duas bibliotecas foi implementado o processo de seleção de imagem, que é a escolha e compressão de uma imagem e a utilização desta foto comprimida no recortador de fotos para deixa-lá do tamanho necessário.

Foi implementada uma interface de pré-visualização de notícia (Figura 3.6). Após os dados da notícia serem preenchidos, antes do usuário enviar, uma modal com a notícia montada (como aparece no visualizar notícia) é mostrada a ele, para a validação da aparência e dados.

Figura 3.6 – Interface de preview de notícia



Fonte: Autor

3.6 Paginação de dados

Há recursos na aplicação que podem ter centenas de kilobytes. Uma requisição que retorne uma lista de *n* destes valores, pode ter uma performance ruim, devido ao tamanho total desta lista. Isto pode causar desconforto ao usuário, uma vez que ele terá que esperar mais tempo para visualizar um conteúdo. Como solução para este problema, é possível fragmentar esta lista em páginas. No aplicativo, são exemplos de requisições paginadas as notícias e as notificações. A requisição destes recursos contém atributos para definir como a paginação será feita, por exemplo, número da página requisitada e quantidade de itens por página. Enquanto

a resposta contém os dados do recurso requisitado, número total deste recurso no servidor e o número destes recursos que estão nesta consulta.

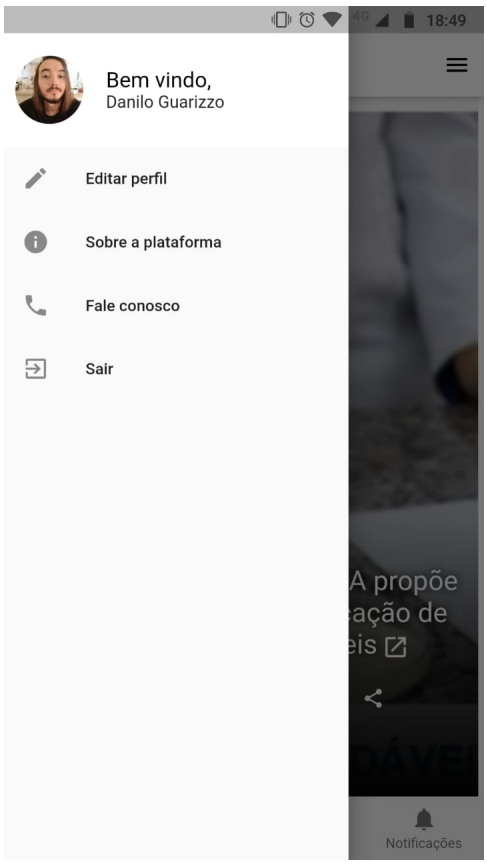
No entanto, com o uso de paginação, o usuário pode perceber um novo carregamento de dados sempre que o conteúdo de uma página acabar. Para que o usuário não perceba que o acesso aos dados é paginado, foi implementado um controle que começa o carregamento da próxima página sempre que o usuário estiver próximo do fim da página atual. Isso torna a leitura do usuário fluida e diminui requisições desnecessárias.

3.7 Menu lateral (Navigation Drawer)

O Flutter disponibiliza um componente que contém os controles padrões de um menu lateral, como o ícone de abertura do menu e as ações de deslizar para abrir e fechar. Cabe ao desenvolvedor estilizar e alocar as informações contidas no menu.

No aplicativo, o menu lateral foi utilizado para conter os menus de navegação referentes às informações do usuário, informações sobre a plataforma e ação de deslogar da plataforma, como demonstra a Figura 3.7.

Figura 3.7 – Resultado menu lateral



Fonte: Autor

3.8 Análise do desenvolvimento em Flutter

A curva de aprendizado da linguagem Dart é realmente pequena, por contemplar vários estilos de programação como a orientação à objeto, dinâmica e funcional. O começo do desenvolvimento com Flutter pode ser desafiador para desenvolvedor que tem experiência em Web, devido a sua estilização e composição da interface de usuário baseada em Widgets. Porém, é apenas uma dificuldade de mudança de estilo de desenvolvimento, pois o Flutter fornece diversos Widgets de estilização, como por exemplo: cores de fontes, fundos, sombras para fontes, posicionamento e tamanho de Widgets, disposição em linhas e colunas, entre ou-

tros. Neste âmbito, um erro por parte do desenvolvedor que pode acontecer é a não utilização dos Widgets de responsividade (Expanded e Flexible) na composição do layout, o que ocasiona em não ter a aplicação funcional em todos os dispositivos.

Outro ponto a se tomar cuidado é na utilização de programação assíncrona. Diversas funções do Flutter tem seu retorno do tipo "Future", o que significa que o retorno dessa função é assíncrono e pode não devolver o resultado esperado na ordem de execução. Para resolver esse problema, caso o retorno de uma função assíncrona necessite ser utilizado logo após a sua chamada, pode-se usar a chave "await"para tornar esta função sequencial. Caso o retorno possa ser utilizado apenas quando estiver pronto, deve-se utilizar o método "then"para capturar o resultado.

A navegação do Flutter permite passar dados entre as telas, tanto na ida, quanto na finalização de uma tela, o que torna a utilização de ferramentas de controle de estado, como o Redux, necessárias apenas quando se tem uma aplicação com muitos estados compartilhados entre diversas telas.

Os métodos de alto nível ("get", "post", "put"e "delete") da biblioteca "http"do Dart resolvem a maior parte das requisições, porém para requisições do tipo Multipart (onde há envio de dados simples e arquivos juntos), o desenvolvedor deve montar a requisição utilizando a classe "MultipartRequest"dessa biblioteca. Caso isto não seja feito, a requisição pode ter seus arquivos não reconhecidos pelo servidor, o que acarretará em um resultado diferente do planejado.

4 CONCLUSÃO

O objetivo deste relatório foi destacar a importância da abordagem multiplataforma no desenvolvimento de aplicações para dispositivos móveis, conceitos básicos sobre esta área e a maior facilidade gerada ao utilizar um framework de geração de aplicativos nativos.

Este projeto contribuiu para o conhecimento sobre o estilo arquitetural REST e a utilização de RESTful API, que são fundamentais no desenvolvimento de serviços web modernos e que facilitam a distribuição dos dados entre as plataformas. Também contribuiu para o conhecimento sobre desenvolvimento multiplataforma de aplicações para dispositivos móveis, que estão em alta no mercado, por terem a capacidade de criar aplicações em menor tempo e também com menor custo.

O Flutter cumpriu seu propósito, pois atendeu as necessidades do projeto. A criação do aplicativo com apenas um código fonte para ambas as plataformas (Android e iOS) foi bem sucedida. O layout do aplicativo conseguiu atingir a proposta inicial, devido à grande quantidade de Widgets disponíveis e a capacidade de cria-los. Todos os recursos nativos necessários para o funcionamento do aplicativo foram providos pelo Flutter. O *hot reload* foi um aspecto satisfatório em relação ao desenvolvimento utilizando esta tecnologia. Por possibilitar a atualização do aplicativo sem a necessidade de compila-lo novamente, permitiu que as interfaces fossem desenvolvidas com maior velocidade.

Entre as dificuldades encontradas durante o desenvolvimento do aplicativo, pode-se citar a diferença da estilização e criação de interface no Flutter, se comparado às tecnologias que utilizam os padrões do desenvolvimento Web (HTML, CSS e Javascript). Outra dificuldade encontrada foi a integração de autenticação utilizando redes sociais, que foi implementada na plataforma Android, mas resultou em erros no aplicativo gerado para iOS. Fato que acarretou no adi-

amento da implementação dessa funcionalidade, mesmo que ela seja importante nas aplicações atuais, devido à praticidade fornecida aos usuários.

REFERÊNCIAS BIBLIOGRÁFICAS

CETIC.BR. **TIC Domicílios**. 2018. Disponível em: <https://www.cetic.br/tics/domicilios/2018/individuos/C16/>.

EL-KASSAS, W. S. et al. Taxonomy of cross-platform mobile applications development approaches. **Ain Shams Engineering Journal**, Elsevier, v. 8, n. 2, p. 163–190, 2017.

FIELDING, R.; RESCHKE, J. Rfc 7231-hypertext transfer protocol (http/1.1): Semantics and content. **Internet Engineering Task Force (IETF)**, 2014.

FIELDING, R. T.; TAYLOR, R. N. **Architectural styles and the design of network-based software architectures**. [S.l.]: University of California, Irvine Doctoral dissertation, 2000. v. 7.

FLUTTER. **Technical overview**. 2019. Disponível em: <https://flutter.dev/docs/resources/technical-overview>.

FLUTTER. **Why did Flutter choose to use Dart?** 2019. Disponível em: <https://flutter.dev/docs/resources/faq>.

GOURLEY, D. et al. **HTTP: the definitive guide**. [S.l.]: "O'Reilly Media, Inc.", 2002.

JONES, M.; BRADLEY, J.; SAKIMURA, N. **RFC 7519: JSON Web Token (JWT). Internet Engineering Task Force (IETF)**. 2015.

LATIF, M. et al. Cross platform approach for mobile application development: A survey. In: IEEE. **2016 International Conference on Information Technology for Organizations Development (IT4OD)**. [S.l.], 2016. p. 1–5.

MDN, C. **HTTP Messages**. 2019. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages>.