UNIVERSIDADE FEDERAL RURAL DO SEMIÁRIDO - UFERSA
CURSO DE CIÊNCIA DA COMPUTAÇÃO
Trabalho de Conclusão de Curso (2021).

AVALIAÇÃO DE ABORDAGENS E DEFINIÇÃO DE DIRETRIZES PARA O DESENVOLVIMENTO DE APLICATIVOS MOBILE: UM RELATO DE EXPERIÊNCIA UTILIZANDO A ABORDAGEM NÃO NATIVA DE DESENVOLVIMENTO

Vinicius George Carlos Nunes[1], Paulo Gabriel Gadelha Queiroz[2]

Resumo: O mundo do desenvolvimento *mobile* se torna cada vez mais relevante, e por consequência competitivo. Questões como o custo e o tempo de desenvolvimento, o desempenho, a interface de usuário e o número de usuários potenciais se tornaram cruciais em um mercado onde cada detalhe importa. A partir desse contexto, o presente trabalho tem como objetivo comparar as abordagens nativa e não nativa de desenvolvimento de aplicações móveis, e definir diretrizes para a escolha da melhor abordagem, de acordo com as vantagens e desvantagens de cada uma e levando-se em consideração as características da aplicação a ser desenvolvida. Ademais, objetiva-se desenvolver um aplicativo móvel utilizando o paradigma de desenvolvimento que se mostrou mais adequado, a partir das diretrizes definidas, que foi a abordagem não nativa *Progressive Web App* (PWA) .

Palavras-chave: desenvolvimento de *software*; dispositivos móveis; abordagens não nativas de desenvolvimento.

1. INTRODUÇÃO

O mercado de dispositivos móveis passou por um processo de grande popularização com o surgimento dos smartphones, com sistemas operacionais mais avançados, que permitiram o desenvolvimento de aplicativos cada vez mais complexos, com mais recursos e serviços ao usuário.

A partir dessa popularização, o mercado de aplicativos móveis passou por um rápido crescimento, sendo disputado por diversos sistemas, que atualmente se resumem ao Android do Google e iOS da Apple. Segundo a plataforma de análise de tráfego *web statcounter* [1], em outubro de 2021 o Android possuía 72,19 % do mercado e o iOS 27%. Frente a essa fragmentação, um dos principais desafios da área é a forma que se dá o desenvolvimento de um aplicativo para essas plataformas.

Em uma aplicação multiplataforma, aspectos como distribuição do aplicativo, tecnologias necessárias para o seu desenvolvimento, tempo, custo e manutenção, são pontos importantes que o diferenciam de uma aplicação criada para apenas uma plataforma [2].

O desenvolvimento de aplicações móveis é normalmente feito de forma nativa, ou seja, existe um ambiente de desenvolvimento para cada sistema operacional, com linguagens, bibliotecas e ferramentas diferentes. Entretanto, nos últimos anos, o paradigma de desenvolvimento não nativo vem ganhando espaço, tendo como objetivo tornar o desenvolvimento mais rápido e menos custoso, à medida que necessita de apenas um ambiente de desenvolvimento (uma *stack*) para gerar aplicativos para diversas plataformas alvo [3].

Acredita-se que cada abordagem tem suas vantagens, desvantagens e pode ser melhor aplicada para resolver determinados problemas. Dessa forma, neste artigo, propõe-se uma comparação entre as abordagens nativa e não nativa, de modo a destacar as vantagens e desvantagens de cada abordagem, além de apresentar um conjunto de diretrizes para a escolha da abordagem mais apropriada, de acordo com o problema a ser resolvido. Por fim, propõe-se apresentar um estudo de caso que consiste no desenvolvimento de um sistema que auxiliará agricultores familiares nas vendas de seus produtos. O sistema foi desenvolvido usando a abordagem mais adequada, *Progressive Web App* (PWA) , de acordo com as diretrizes propostas neste trabalho.

O restante deste artigo está organizado da seguinte forma: na seção 2 são apresentados os trabalhos relacionados; na seção 3, são apresentadas as abordagens de desenvolvimento não nativas de aplicativos, suas classificações e características; na seção 4, são definidas e apresentadas as diretrizes para que se possa definir a melhor abordagem para uma aplicação; na seção 5, apresenta-se um estudo de caso, com o desenvolvimento de um aplicativo móvel focado na interação entre clientes e agricultores familiares; e, por fim, na seção 6 apresenta-se a conclusão deste trabalho.

2. TRABALHOS RELACIONADOS

Há uma constante evolução no mercado de desenvolvimento móvel, o que se dá pelo relativamente recente *boom* desse mercado, que passa a ser o centro das inovações tecnológicas. Devido a esses fatores, a maioria dos estudos encontrados foram feitos entre os anos de 2010 e 2015, e uma minoria foi publicada entre os anos de 2016 e 2021. O que causa uma desatualização em relação às novas ferramentas de desenvolvimento mais populares. Afetando até mesmo a classificação dos tipos de abordagem para o desenvolvimento aplicativos móveis, que estão em constante evolução, de forma que vários desses trabalhos não possuem mais um forte respaldo na realidade atual.

O artigo de Khachouch et al. [2] classifica as abordagens de desenvolvimento de aplicativos móvel em: nativo, *web*, híbrido, compilados, máquina virtual, modelo, e baseado em nuvem. Para a escolha da melhor abordagem, os autores elaboraram uma série de questões com respostas binárias e deram um peso para cada uma.

Vilček et al. [4] trazem uma perspectiva diferente, ao categorizar as abordagens de forma que dentro de cada uma delas seja elencado um ambiente de desenvolvimento específico para realizar a comparação, da seguinte maneira: nativa android (*Android Studio*), nativa iOS (*Xcode*), nativa *Windows Phone* (*VS Community*), híbrida (*Ionic*), híbrida (*PhoneGap*), e híbrida (*NativeScript*). Na comparação, eles utilizam critérios como quais sistemas operacionais suportam cada ambiente de desenvolvimento, complexidade de instalação, documentação e comunidade.

Já Raj et al. [5] utiliza as seguintes categorias: *web*, híbrido, interpretado, e compilação cruzada, de forma a comparar somente as abordagens não nativas entre si. Para fazer a análise, os autores definem casos de uso e classificam os métodos mais adequados para cada uma. Os seguintes casos de uso são citados: aplicações focadas em servidor, aplicações baseadas em sensores, aplicações *standalone*, e aplicações cliente servidor.

3.ABORDAGEM DE DESENVOLVIMENTO DE APLICATIVOS

No universo de desenvolvimento mobile existem dois grandes grupos de abordagens, a abordagem nativa e a não nativa (ou multiplataforma). É necessário que seja feita uma apresentação dessas abordagens, para que depois ocorra uma avaliação das características de cada uma.

Aplicações nativas são aquelas que são desenvolvidas especificamente para uma plataforma, como Android ou iOS [6], sendo assim, é necessário um ambiente de desenvolvimento específico para cada plataforma em que se deseja ter o aplicativo, cada uma possui suas linguagens de programação, plataformas e outras especificidades. Por outro lado, as aplicações não nativas são aplicativos desenvolvidos de forma que apenas um projeto é necessário para que o aplicativo funcione em várias plataformas, utilizando-se, assim, somente um ambiente de desenvolvimento e linguagem de programação.

Dentre as principais vantagens do desenvolvimento não nativo pode-se citar:

- É necessário escrever o código uma única vez durante o desenvolvimento, o que resulta em uma significativa redução nos custos e no tempo de desenvolvimento.
- As ferramentas e linguagens estão cercadas por comunidades mais atuantes e geralmente orientadas ao código-aberto, que continuamente adicionam e atualizam as suas funcionalidades, ao invés de serem dependentes de um pequeno grupo ou uma única empresa. Dessa forma, existe um maior suporte.
- Por fim, as atualizações e correções de bugs podem ser lançadas para todas as plataformas de uma vez.

Entre as principais desvantagens, quando comparado ao desenvolvimento nativo, QUE et al. [7], destaca:

- Desempenho inferior por parte das aplicações não nativas, visto que elas não são projetadas para funcionar perfeitamente na sua plataforma alvo.
- Menor integração com o *hardware* e o sistema operacional do dispositivo, o que leva a um acesso dificultado aos sensores como GPS e câmeras.
- Além disso, aplicações muito complexas podem ter dificuldades em funcionar perfeitamente em todas as plataformas, pois quanto maior a complexidade do aplicativo, mais erros estão propensos a surgir.

Observa-se que entre as diferentes abordagens não nativas, também existem vantagens e desvantagens. Antes de apresentá-las é importante mostrar que, de acordo com SHAH et al. [6], aplicações não-nativas ou multiplataforma podem ser classificadas nas seguintes categorias baseadas em suas implementações:

Web Apps: são aplicações que utilizam o browser de internet para funcionar e tem acesso limitado ao *hardware* do dispositivo. As principais tecnologias usadas para o seu desenvolvimento são HTML e *Javascript* e

são geralmente desenvolvidos com o uso de bibliotecas e frameworks como *Angular*, *Vue*, *React*, *jQuery*, *Django*, *Ruby on Rails* entre outros [6].

Progressive Web Application (PWA): são considerados uma evolução das aplicações *web* comuns. Diferentemente de uma aplicação Web convencional, as PWAs podem ser executadas em modo offline, instaladas e possuem maior acesso ao *hardware* do dispositivo [8].

Apps Híbridos: aplicativos que também são desenvolvidos com HTML e *Javascript*, mas são embutidos num *software* chamado de *container*. *Apache Cordova* é um *container* popular usado para desenvolver aplicações híbridas [6].

Apps Interpretados: emulam apps nativos ao permitir que o usuário interaja com componentes de interface específicos daquela plataforma. Esses aplicativos podem ser desenvolvidos com Ruby, XML, *Javascript* e outras linguagens. O *React Native*, que é uma tecnologia criada pelo Facebook em 2015, é a mais popular dessa categoria. O framework fornece uma "ponte" que faz a comunicação, interpretando o código *web* para o sistema operacional [6].

Apps de Compilação Cruzada: os apps *cross-compiled* têm o seu código fonte compilado para cada plataforma alvo, de modo a ter um funcionamento parecido ao de aplicações nativas e, consequentemente, um desempenho similar. Como o código nativo é gerado automaticamente, pode haver dificuldades no acesso de *application programming interfaces* (APIs) específicas para uma plataforma. Uma das ferramentas mais populares nesta categoria é o *Xamarim* [5].

O kit de desenvolvimento *Flutter* é uma inovação na abordagem de compilação cruzada, pois abstrai a necessidade de implementações específicas para componentes da interface e de certas APIs. Todos os componentes desses apps são *widgets* que servem como blocos base que formam a interface da aplicação. Esses apps usam seus próprios motores gráficos para gerar código nativo. O *flute*r foi inicialmente lançado em maio de 2017 e vem se tornando cada vez mais popular entre os desenvolvedores multiplataforma [6].

Na figura a seguir, ilustra-se a classificação dessas abordagens de acordo com características como o seu desempenho, as que são baseadas em tecnologias *web*, e as que podem ser distribuídas nas lojas de aplicativos.

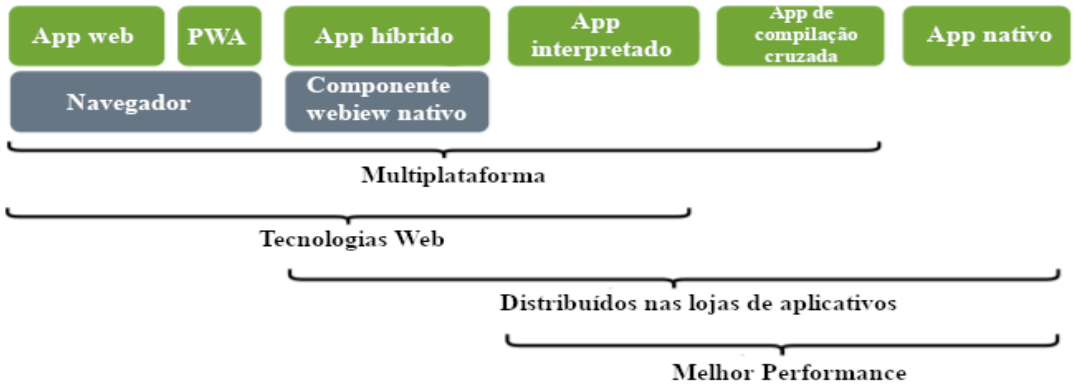

Fig. 1. Classificação das abordagens de desenvolvimento mobile [9].

3.1 Apps Web

A principal característica dos aplicativos *web* é que eles são feitos para serem executados em um navegador de internet. Essas aplicações são desenvolvidas usando tecnologias da *web* como HTML, CSS e *JavaScipt*. Nesta abordagem, nenhum dos componentes do aplicativo são instalados no dispositivo móvel. Em vez disso, o aplicativo é acessível a partir de uma *Uniform Resource Locator* (URL) e os dados do aplicativo são manipulados pelo servidor [2].

Entre suas vantagens, destaca-se a falta de necessidade de instalação do aplicativo. Por consequência, as atualizações não dependem do dispositivo móvel. Além disso, possui grande compatibilidade, estando disponível para praticamente qualquer dispositivo.

Entre as desvantagens, observa-se que esses apps necessitam de uma conexão de internet para que funcionem, apresentam dificuldade em acessar funcionalidades de *hardware* no dispositivo e poucos modos de monetizar os aplicativos. Outrossim, não é possível lançá-los nas lojas de aplicativos.

3.1.1 *Progressive Web Apps*

Os PWAs são uma evolução natural das aplicações *web* simples, unindo a versatilidade das tecnologias *web* com algumas funções presentes somente em aplicações nativas. Podem ser instaladas no dispositivo, exibir

notificações, e podem funcionar mesmo sem conexão com internet. Eles têm a garantia de estar funcionando sempre na última versão disponível, sem a necessidade de atualizar a aplicação a partir da loja de aplicativos, já que o aplicativo é atualizado via servidor pelo desenvolvedor.

Os PWAs são caracterizados como rápidas: geralmente renderizando o conteúdo no dispositivo do usuário em apenas alguns segundos. Estável: mesmo em conexões de internet de fraca qualidade ou em dispositivos mais antigos. Envolvente: ao ativar notificações, mesmo na Web App, os utilizadores podem receber notificações para serem alertados sobre alguma informação pertinente na aplicação, mesmo que o browser não esteja aberto [8]. Entre suas desvantagens, estão o desempenho reduzido, menor integração com o sistema operacional e acesso ao *hardware*, que apesar de ser mais completo do que em *apps web* tradicionais, ainda não atinge o nível de acesso ao *hardware* de aplicações nativas.

3.2 Aplicações Híbridas

A principal característica das aplicações híbridas é que elas combinam as funcionalidades das aplicações nativas com tecnologias usadas nas aplicações *web* [5], como HTML, CSS e *Javascript*. É uma das abordagens de desenvolvimento multiplataforma mais simples. Esse tipo de aplicação funciona ao injetar código HTML num *container web* nativo, possibilitando assim que eles sejam instalados no dispositivo, mesmo com os dados e lógica do programa vindo de um servidor remoto. O desenvolvimento se dá com o código HTML e *Javascript*, sem conhecimento de detalhes da plataforma alvo. O acesso ao *hardware* do dispositivo é feito por meio de APIs, que não são diretamente invocadas pelo programador, mas são abstraídas na forma de plugins, que executam uma funcionalidade de forma universal, e que internamente invocam as APIs específicas para cada plataforma [10].

Aplicativos assim desenvolvidos, tem como principais vantagens poderem ser publicados nas lojas de aplicativos, possuírem maior acesso ao *hardware* do dispositivo, se comparado à aplicações *web*s e ainda, possuem maior reúso da interface nas várias plataformas. Tem como desvantagens um desempenho reduzido se comparado às aplicações nativas, o que acarreta num certo prejuízo na experiência do usuário, além de não se integrarem tão bem com a interface nativa do sistema operacional.

3.3 Aplicações Interpretadas

As Aplicações Interpretadas recebem esse nome devido a sua habilidade de enviar o código fonte diretamente para o dispositivo, onde esse é interpretado por um motor *Javascript*. Esse motor também é chamado de interpretador, e seu tipo depende da plataforma utilizada. O *Javascriptcore* desenvolvido pela Apple é usado no iOS, e o V8, um motor de código aberto, é gerenciado pelo Android do Google [11].

Sua principal vantagem é resultar em aplicativos análogos aos nativos, que podem ser criados somente com *Javascript*. Dessa forma, essas aplicações têm uma performance melhor e interagem bem com o *hardware* do dispositivo, se comparado com outras abordagens multiplataforma. Além disso, possuem uma interface integrada ao sistema, já que há uma comunicação com os componentes nativos do sistema. Entre as desvantagens, destaca-se que o reúso da interface é limitado pelo nível de abstração da ferramenta utilizada, e que o desenvolvedor ficará preso às tecnologias usadas no framework de desenvolvimento escolhido. Pode ocorrer também, alguma perda de desempenho, devido a interpretação do código no momento da execução, que costuma ser mais lenta que a execução de código já compilado.

3.3.1 *React Native*

O *Reactive Native* vem se tornando a principal ferramenta da abordagem interpretada desde a sua criação em 2015 pelo Facebook. Ele é uma tecnologia de código aberto, na qual os aplicativos são criados com a linguagem *Javascript*, mas se portam como aplicações nativas.

Ela é construída com a popular biblioteca *ReactJS*, utilizando o modelo DOM (*Document Object Model*), que manipula o aplicativo de uma forma declarativa, o que fornece uma maior abstração para o desenvolvedor e torna o processo de criação do app mais simples. O DOM virtual armazena uma cópia do modelo de objetos e altera somente as partes que sofreram mudanças do aplicativo, o que traz a vantagem do desenvolvedor não necessitar esperar o aplicativo completo ser recompilado, bastando apenas recarregar a página para visualizar a mais nova versão.

A essência do *React Native* é a camada de abstração conhecida como “bridge” (ponte) , que permite o acesso de APIs específicas para cada plataforma que irão renderizar os componentes da interface. Enquanto nas aplicações *web ReactJS* a ponte acessa o DOM do *browser*, no *React Native* a ponte acessa APIs em *swift* ou *objective-C* no iOS, e *Java* ou *Kotlin* no Android. Um interpretador *Javascript* é utilizado em conjunto com a ponte para renderizar a interface de usuário, transformando as marcações típicas das linguagens *web* em componentes visuais da interface nativa do sistema operacional [12].

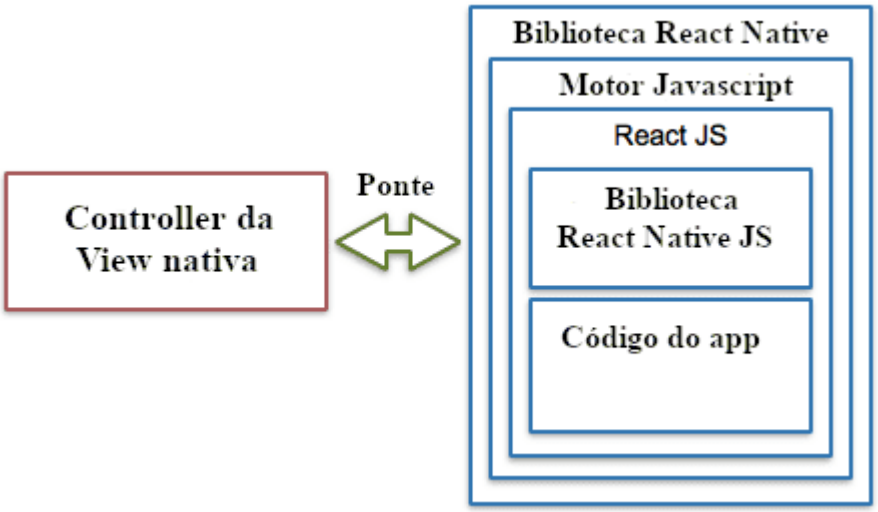

Fig. 2. Arquitetura de um *App React Native* [6].

3.4 Aplicações de Compilação Cruzada

Nesse tipo de abordagem o compilador da ferramenta de desenvolvimento converte o código fonte para binários executáveis de mais de uma plataforma, permitindo que o desenvolvedor utilize somente uma linguagem no código fonte, já que o compilador cruzado se encarregará de gerar o código executável específico de cada plataforma.

Tem como principais vantagens, possuir todas as capacidades que aplicações nativas possuem, como acesso ao *hardware* do dispositivo, e ainda possui a melhor performance entre as opções de desenvolvimento multi-plataforma. Sua principal desvantagem é que a interface do usuário não é reutilizável, como também o acesso às funcionalidades do *hardware* (câmera, localização, etc.) que devem ser implementadas de forma específica para cada sistema, pois esses aspectos da aplicação variam de acordo com a plataforma alvo [13].

3.4.1 *Flutter*

O *flutter* representa uma evolução na abordagem de compilação cruzada ao tratar todos os componentes como *widgets*, em vez de depender dos componentes nativos. Ele funciona criando seus próprios componentes (*widgets*) através de um motor de renderização próprio, diferentemente das aplicações de compilação cruzada tradicionais. Ao renderizar sua própria interface, o *flutter* resolve o problema de reusabilidade de interface, comum nessa abordagem, e torna necessário somente o aprendizado de uma única linguagem de programação.

Chamada de *Dart*, ela é uma linguagem que também foi criada pelo Google [14]. O *Flutter* usa o *Dart* para criar componentes juntamente com o motor gráfico 2D *Skia*. O código *Dart* é compilado em código nativo antes da execução do programa, o que é conhecido como compilação AoT (*Ahead of Time*).

4. DIRETRIZES PARA SELEÇÃO DA ABORDAGEM DE DESENVOLVIMENTO

Uma forma eficaz de se escolher a abordagem mais adequada é fazer uma avaliação dos requisitos não funcionais mais importantes de um aplicativo, e analisar quais critérios de cada abordagem mais contribuem para que esses requisitos sejam alcançados. Dessa forma, é possível tornar a escolha da abordagem mais objetiva de modo que atenda às necessidades do aplicativo a ser construído.

Para SHAH et. al. [6] a melhor forma de avaliar a abordagem a ser tomada se dá pelos seguintes critérios:

1) UI / UX (Interface do Usuário): a aparência do aplicativo, considerando a interface do usuário e a experiência do usuário, que é influenciada pelo grau de integração da aplicação com a interface do sistema operacional. Aplicações em que essa coesão é de muita importância devem considerar em um aplicativo nativo, pois nenhuma outro ambiente terá uma interface tão bem integrada.
2) Usuários potenciais: o público-alvo para qual o aplicativo é direcionado. Geralmente é um público que utiliza várias plataformas, mas também pode ser segmentado em apenas uma, como iOS, por exemplo. Dessa forma, abordagens não nativas são recomendadas quando esse

é um requisito relevante, principalmente as *web*, que funcionam quase em todos os dispositivos.

3) Custo de desenvolvimento: há diferenças no tempo e dinheiro necessários para construir o aplicativo, de acordo com a abordagem escolhida. Quanto mais plataformas alvo um método de desenvolvimento englobar, menor será o custo, pois serão necessárias menos pessoas e menos tempo para o desenvolvimento do aplicativo.
4) Segurança do aplicativo: a capacidade que pessoas não autorizadas têm de acessar ou invadir a lógica ou os dados do aplicativo. Aplicações nativas saem na frente, pois se integram com os mecanismos de segurança do próprio sistema operacional, que costumam ser bastante robustos.
5) Facilidade de atualização: o tempo necessário para que a atualização do aplicativo chegue em todas as plataformas, uma vez implantada. Aplicações *web* se tornam mais vantajosas, pois não precisam passar pelo crivo das lojas de aplicativos, que dependendo da companhia, pode ser bastante rigoroso.
6) Complexidade de implementação: quão difícil ou fácil é, para o desenvolvedor, criar o aplicativo usando um ferramenta multiplataforma específica. Utilizar um ambiente multiplataforma pode implicar no aprendizado de uma nova linguagem específica da plataforma, mas que geralmente é uma escolha vantajosa, já que no desenvolvimento nativo é necessário conhecer as linguagens e particularidades de cada sistema operacional alvo.
7) Acesso a APIs nativas: este critério mede a necessidade do aplicativo de acessar o *hardware* do dispositivo, como acelerômetro, câmera e outros sensores, através de APIs nativas. Quão mais próximo do desenvolvimento nativo a abordagem for, mais acesso a APIs nativas ele terá.
8) Licença: do ponto de vista do desenvolvimento, a Apple cobra uma taxa para desenvolver aplicativos para o iOS. Até mesmo seu ambiente de desenvolvimento é fechado por natureza. Nas plataformas do Google e nas baseadas em tecnologias Web, o desenvolvimento é geralmente de código aberto.
9) Linguagem: aplicativos nativos usam linguagens de programação conforme especificado pela plataforma, ou seja, *Java* / *Kotlin* para Android e *Swift* / *Object-C* para iOS. O Flutter usa sua própria linguagem , o *Dart*, e aplicações *web*, PWAs e *React Native* utilizam tecnologias *web* como *Javascript*.
10) Publicável na loja de aplicativos: a aprovação do aplicativo nas lojas de cada plataforma depende se o aplicativo segue as regras e padrões estabelecidos por cada empresa, além de cada plataforma ter o seu próprio processo para atualizar os aplicativos da loja. Aplicações Web e PWAs não passam por esse processo.

A tabela a seguir elenca esses critérios e os avalia perante a escala de Likert. É um tipo de escala usada em questionários, pesquisas e análises, em que cada item da escala corresponde a uma quantidade de pontos [15]. Na tabela abaixo utilizaremos a escala para medir o quão bem, cada abordagem cumpre os requisitos, ela está organizada da seguinte forma: 1- Ruim, 2 - Moderada, 3- Boa, 4- Muito boa, 5 - Excelente.

Tabela 1. mostra a classificação das abordagens em vários aspectos, baseada na tabela de SHAH et. al. [6].

Critérios de decisão	**Nativo**	**Apps Web**	**Híbrido**	**Interpretado (*React Native*)**	***Flutter***
Interface do usuário	Excelente (5)	Moderado (2)	Moderado (2)	Boa (3)	Muito Boa (4)
Usuários potenciais	Moderado (2)	Excelente (5)	Boa (3)	Muito Boa (4)	Boa (3)
Custo de desenvolvimento	Ruim (1)	Muito bom (4)	Muito bom (4)	Moderado (2)	Moderado (2)
Segurança do app	Excelente (5)	Ruim (1)	Moderada (2)	Boa (3)	Muito boa (4)

Facilidade de atualização	Moderada (2)	Excelente (5)	Boa (3)	Boa (3)	Moderado (2)
Complexidade de implementação	Ruim (1)	Excelente (5)	Moderada (2)	Moderada (2)	Moderada (2)
Acesso a APIs nativas	Sim	Sim, com HTML 5	Sim, através de plugins	Sim	Sim
Licença	Android: Open-source, iOS: Closed-source	*Open-source*	Geralmente *open-source*	*Open-source*	*Open-source*
Linguagem	*Java/Kotlin, Swift/Objective-C*, etc.	HTML, CSS, *Javascript, TypeScript*, etc.	HTML, CSS, *Javascript, Node.js* etc.	*Javascript, JSX, UX Markup*, etc.	*Dart*
Publicável na loja de aplicativos	Sim	Não	Sim	Sim	Sim

5. ESTUDO DE CASO

A partir de uma análise dessas diretrizes é possível determinar as melhores alternativas para o desenvolvimento de uma aplicação de acordo com os requisitos do aplicativo. Para ilustrar essa análise, apresenta-se um estudo de caso, que consiste no desenvolvimento de uma aplicação para venda de produtos cultivados por produtores da agricultura familiar.

Num contexto de pandemia, a comunicação desses trabalhadores com seus clientes se tornou complicada devido ao isolamento social, visto que antes da pandemia os agricultores participavam de feiras e conseguiam encontrar compradores com maior facilidade. Esse fator, aliado à chegada dessa população ao mundo dos smartphones, proporcionou a oportunidade ideal para que uma nova ferramenta fosse criada com o objetivo de melhorar a comunicação entre vendedor e clientes, permitindo uma maior agilidade nesse processo de vendas.

Entre os requisitos não funcionais desta aplicação, os mais importantes a se considerar foram:

- A disponibilidade do app, visto que o fato dele estar disponível na maioria das plataformas é considerado um fator essencial, a fim de aumentar a quantidade de possíveis usuários.
- Outro ponto importante é o baixo orçamento para o desenvolvimento da aplicação, uma vez que não se tratam de pessoas com grande poder aquisitivo.
- Por fim, é importante que a aplicação seja leve, de modo a ser executada sem engasgos na grande maioria dos dispositivos móveis.

A partir desses requisitos e da análise das diretrizes apresentadas na seção 3, optou-se pelo desenvolvimento de um *progressive web app.* Foi levado em consideração o fato dessa abordagem possuir a maior nota em dois dos seus principais requisitos: usuários potenciais, com nota 5, e custo de desenvolvimento, com nota 4, o que dá uma soma de 9, enquanto outras abordagens atingiram no máximo 7.

5.1 Desenvolvimento da Aplicação

Nesta seção apresenta-se o processo de criação do aplicativo desde os seus requisitos até a sua implementação, detalhando os métodos e as ferramentas utilizadas.

5.1.2 Requisitos

No decorrer de reuniões com os interessados na criação do aplicativo, foi possível conhecer o modelo de negócios praticado pelos envolvidos. A partir de uma análise desse modelo foi possível elencar os requisitos funcionais e não-funcionais.

Entre os requisitos funcionais destaca-se: criar uma conta de usuário; visualizar os estabelecimentos abertos; visualizar os produtos e preços de cada estabelecimento; adicionar produtos na sacola; remover produtos da sacola; selecionar o endereço de entrega; finalizar o seu pedido; editar endereços; adicionar endereços; visualizar pedidos feitos.

Entre os requisitos não funcionais destacam-se: a necessidade da aplicação funcionar em qualquer plataforma; ser uma aplicação rápida e responsiva; fornecer segurança aos usuários;

5.1.2 Protótipos

Foram desenvolvidos protótipos que são baseados em representações das telas da interface do sistema, feitos na plataforma Figma. A construção desses protótipos foi de grande importância para que se pudesse haver um melhor entendimento de como a aplicação funcionaria, bem como os requisitos que deveriam ser atendidos

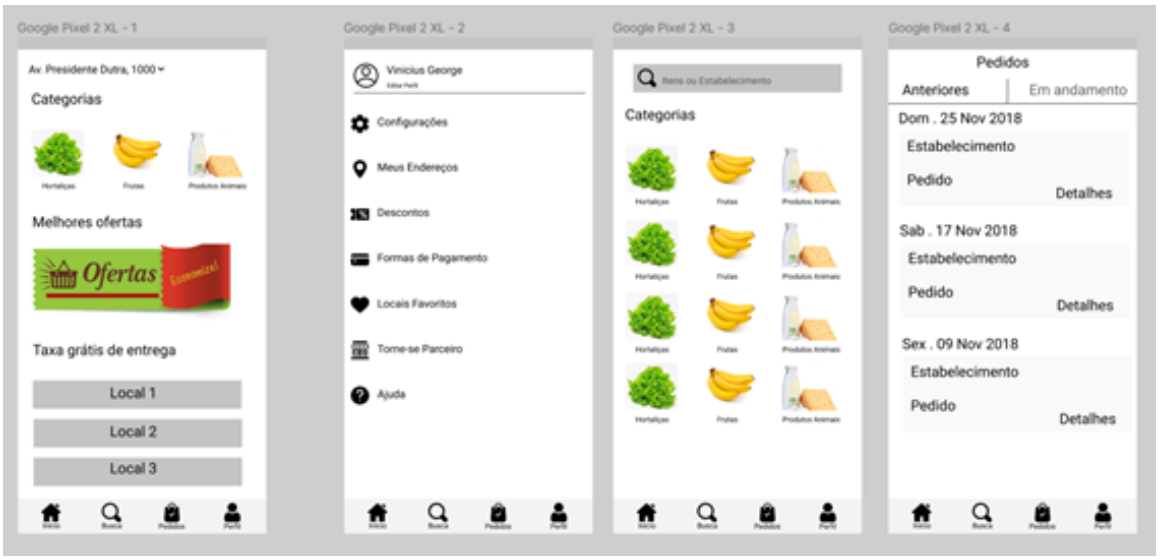

Fig 3. telas do protótipo do aplicativo. Disponível em https://figma.com/file/NnqcccufuoNxxdN6MqMrWy

5.1.3 Servidor

A parte do servidor, *server side* ou *backend* da aplicação consiste na parte do app responsável pela lógica que acontece por trás da interface. Ela foi construída utilizando a linguagem *Java*, que já é consolidada na área e possui amplo suporte e o *framework spring data* JPA, que facilita a conexão e gerenciamento do banco de dados SQL que foi utilizado para armazenar as informações do aplicativo.

Nesse sentido, a JPA (*Java Persistence API*) é uma API padrão da linguagem *Java* que descreve uma interface comum para frameworks de persistência de dados [16]. A JPA define um meio de mapeamento objeto-relacional para objetos *Java* simples e comuns (POJOs), denominados *beans* de entidade. O *spring data* JPA utiliza o JPA para abstrair ainda mais o acesso ao banco de dados e permitir uma implementação mais limpa e eficiente da interação com a camada do banco de dados na sua aplicação[17].

A partir dessas tecnologias foi criada uma API REST, utilizando-se o *Spring Data Rest*, que encapsula o JPA e entrega os dados através de uma interface REST [18]. Uma API (Interface de Programação de Aplicação) consiste em um conjunto de métodos e padrões estabelecidos e documentados por um *software* para a utilização das suas funcionalidades, sem que seja necessário entender como ocorreu a sua implementação[19]. REST é um estilo de arquitetura de *software* que define um conjunto de restrições a serem usadas para a criação de *web services* (serviços Web) [20].

Os *Web services RESTful* permitem que os sistemas solicitantes acessem e manipulem representações textuais de recursos da Web usando um conjunto uniforme e predefinido de operações sem estado, ou seja, cada requisição se comporta como uma transação independente que não está relacionada a qualquer requisição anterior.

5.1.4 Interface

Durante os estudos e análise do trabalho, constatou-se que um PWA, app *web* progressivo, seria a melhor abordagem para o desenvolvimento desse aplicativo. Para o desenvolvimento *web* foi utilizado o framework vue.js, que é um *framework Javascript* progressivo [21] para a construção de interfaces de usuário.

Ao contrário de outros *frameworks* monolíticos, Vue foi projetado desde sua concepção para ser adotável incrementalmente. A biblioteca principal é focada exclusivamente na camada visual (*view layer*), sendo fácil de implementar e integrar com outras bibliotecas ou projetos existentes, o que permitiu uma boa integração com a API REST desenvolvida na parte *backend*.

A escolha pelo *VueJS* se deu por sua facilidade de aprendizado, permitindo desenvolver a aplicação durante o prazo do trabalho de conclusão de curso, e também por ter um design responsivo, o que significa que a interface visual da aplicação se adapta bem aos mais variados dispositivos e tamanhos de telas.

Dentro do próprio framework *VueJS* optou-se pelo uso da biblioteca *Vuetify* para auxiliar na construção do visual da aplicação. O *Vuetify* é um framework de interface de usuário construído sobre o Vue.js. Ele foi projetado desde o início para ser fácil de aprender, com centenas de componentes gráficos elaborados a partir da especificação do *Material Design* [22].

5.2 Apresentação da aplicação

Ao iniciar a aplicação nos deparamos com a tela para que o número de celular seja inserido, visto que esse será o principal identificador único de cada usuário. Ao digitar o número e receber o código, o aplicativo vai para a tela de registro caso o usuário ainda não tenha cadastro no sistema ou para a tela inicial, caso contrário, conforme apresenta-se na Fig. 4 - parte A.

Na tela inicial, pode-se observar o endereço atual selecionado pelo usuário, bem como a lista de estabelecimentos abertos. Na parte de baixo há uma barra que é a principal ferramenta de navegação do aplicativo, permitindo alternar entre página inicial, pesquisa, lista de pedidos e o perfil, conforme apresenta-se na Fig. 4 - parte C.

Ao entrar num estabelecimento, o usuário visualiza a lista de produtos disponíveis, uma imagem, seu valor e a quantidade a ser adicionada na sacola. Ao clicar em “ver sacola” ou no ícone flutuante da sacola, é possível visualizar os itens que estão para ser comprados e o valor total do pedido. Após a finalização da compra, um histórico dos pedidos encontra-se disponível na aba “Pedidos”, conforme apresenta-se na Fig. 4 - parte F.

O aplicativo está disponível no endereço https://bit.ly/3wsTQxe

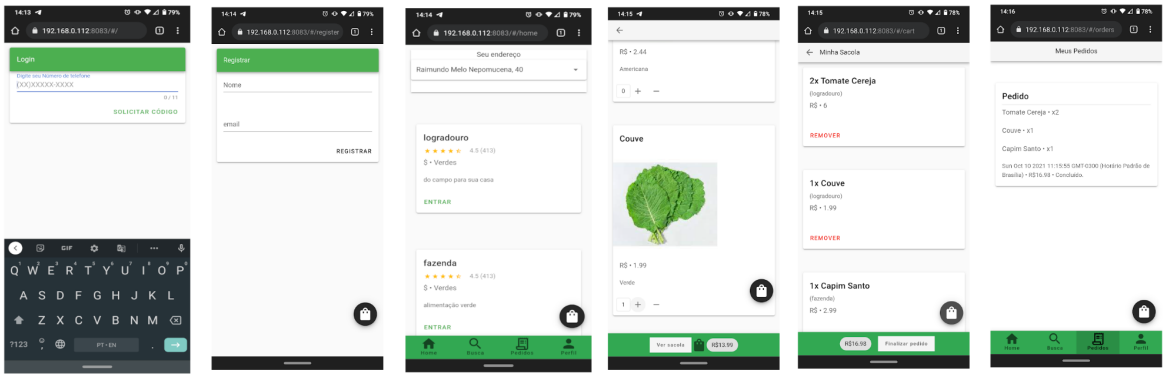

Fig 4. Principais telas da aplicação, respectivamente: A. tela de login, B. tela de cadastro, C. tela inicial, D. tela do estabelecimento, E. tela do carrinho, F. tela do histórico dos pedidos.

5.3 Validação

Para a validação da aplicação foi feito um questionário com potenciais usuários, em que após a utilização do aplicativo, os respondentes deveriam concordar ou não com afirmações sobre a experiência de uso, numa escala de Likert. A escala variava entre 0 e 5, que representam “discordo totalmente” e “concordo totalmente”, respectivamente.

Ao fim da coleta foram obtidas 21 respostas. O questionário está disponível em https://forms.gle/WRWEdj3uFnrRREfh7.

A elaboração do questionário foi baseada no modelo *Technology Acceptance Model* (TAM), que tem como objetivo tentar medir a aceitação dos usuários a determinados *softwares* de acordo com dois principais quesitos: a utilidade percebida e a facilidade de uso percebida [23]. A utilidade percebida mede o grau em que o usuário acredita que utilizar uma tecnologia ou sistema pode melhorar o seu desempenho sobre o objeto de uso. Já a facilidade de uso percebida se refere ao grau que uma pessoa acredita que o uso de um determinado sistema seria livre de esforço, ou seja, a facilidade de utilizar a tecnologia.

A partir do modelo TAM as seguintes afirmações foram feitas, com os seguintes resultados (em média):

Tabela 2. Resultado do questionário aplicado

Afirmação	Média de 1 a 5
O aplicativo é fácil de usar	4,71
O aplicativo facilitou o ato de comprar mercadorias	4,71
O aplicativo permitiu uma maior interação com os vendedores	4,47
O aplicativo é mais acessível quando comparado à forma tradicional de compras	4,66
O aplicativo possui uma interface amigável	4,66
Fiquei confuso com algum aspecto do aplicativo	2,09
É fácil criar uma conta	4,61
É fácil fazer um pedido	4,71
As funcionalidades do aplicativo são claras e objetivas	4,76
Tenho interesse em continuar usando o aplicativo	4,71

Ao avaliar as médias obtidas é possível perceber que a aplicação obteve uma boa aceitação, pois todas as afirmações de conotação positiva obtiveram uma média acima de 4,4, numa escala que vai de 0 a 5. A única afirmação de conotação negativa teve uma média de 2,09. Dessa forma, conclui-se que os usuários tiveram uma boa percepção de utilidade e de facilidade de uso, o que é mostrado na boa média de avaliação das afirmações mostradas no questionário, bem como nos comentários deixados por alguns participantes, como “O aplicativo torna uma atividade cotidiana mais simples e agradável” e “O aplicativo é fácil de usar”.

6. CONSIDERAÇÕES FINAIS

Este trabalho foi conduzido com o objetivo de definir diretrizes objetivas e práticas, para facilitar a escolha da abordagem de desenvolvimento móvel mais adequada, com base nos objetivos do aplicativo a ser desenvolvido. Adicionalmente, foram apresentados indícios de que essas diretrizes são efetivas, por meio do desenvolvimento de um aplicativo com uso da abordagem recomendada pelas diretrizes.

O aplicativo foi desenvolvido utilizando o método PWA, pois foi a abordagem que mais se encaixa nos requisitos necessários para aplicação, de alta portabilidade, rapidez e baixo custo de desenvolvimento. O aplicativo foi desenvolvido utilizando as ferramentas *Java, framework Spring, Javascript* e o *framework vue.js*.

Por fim, destaca-se que este trabalho não esgotou o assunto da pesquisa. Portanto, propõe-se como trabalhos futuros, a realização de testes mais profundos e específicos, a fim de estimar com maior precisão aspectos como o desempenho e a segurança das variadas abordagens.

7. REFERÊNCIAS

[1] STATCOUNTER. Mobile Operating System Market Share Worldwide. Disponível em: https://gs.statcounter.com/os-market-share/mobile/worldwide. Acesso em: 24 mai. 2021

[2] KHACHOUCH, Mohamed Karim et al. Framework Choice Criteria for Mobile Application Development. In: **2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE)**. IEEE, 2020. p. 1-5.

[3] DA FONSECA, Marcos Roberto; BEDER, Delano Medeiros. Aplicativos Android: desenvolvimento nativo x uso de ferramentas baseadas em padrões web. Revista TIS, v. 4, n. 1, 2016.
[4] VILČEK, Tena; JAKOPEC, Tomislav. Comparative analysis of tools for development of native and hybrid mobile applications. In: **2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)**. IEEE, 2017. p. 1516-1521.
[5] RAJ, CP Rahul; TOLETY, Seshu Babu. A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach. In: **2012 Annual IEEE India Conference (INDICON)**. IEEE, 2012. p. 625-629.
[6] SHAH, Kewal; SINHA, Harsh; MISHRA, Payal. Analysis of Cross-Platform Mobile App Development Tools. In: **2019 IEEE 5th International Conference for Convergence in Technology (I2CT)**. IEEE, 2019. p. 1-7.
[7] QUE, Peixin; GUO, Xiao; ZHU, Maokun. A comprehensive comparison between hybrid and native app paradigms. In: **2016 8th International Conference on Computational Intelligence and Communication Networks (CICN)**. IEEE, 2016. p. 611-614.
[8] FORTUNATO, David; BERNARDINO, Jorge. Progressive *web* apps: An alternative to the native mobile Apps. In: **2018 13th Iberian Conference on Information Systems and Technologies (CISTI)**. IEEE, 2018. p. 1-6.
[9] HJORT, Elin. Evaluation of React Native and Flutter for cross-platform mobile application development. Master's Thesis in Computer Engineering - Faculty of Science and Engineering, Åbo Akademi University. p. 5. 2020.
[10] DA SILVA, Marcelo Moro; SANTOS, Marilde Terezinha Prado. Os paradigmas de desenvolvimento de aplicativos para aparelhos celulares. **Revista TIS**, v. 3, n. 2, 2014.
[11] CHARKAOUI, Salma et al. Cross-platform mobile development approaches. In: **2014 Third IEEE International Colloquium in Information Science and Technology (CIST)**. IEEE, 2014. p. 188-191.
[12] HEITKÖTTER, Henning; HANSCHKE, Sebastian; MAJCHRZAK, Tim A. Evaluating cross-platform development approaches for mobile applications. In: **International Conference on Web Information Systems and Technologies**. Springer, Berlin, Heidelberg, 2012. p. 120-138.
[13] DE ANDRADE, Paulo RM et al. Cross platform app: a comparative study. **arXiv preprint arXiv:1503.03511**, 2015.
[14] XANTHOPOULOS, Spyros; XINOGALOS, Stelios. A comparative analysis of cross-platform development approaches for mobile applications. In: **Proceedings of the 6th Balkan Conference in Informatics**. 2013. p. 213-220.
[15] LIKERT, Rensis. A technique for the measurement of attitudes. **Archives of psychology**, 1932.
[16] Java Persistence API. FAQ. Disponível em: <https://www.oracle.com/java/technologies/javaee/java-persistence-api-faq.html>. Acesso em: 17 out. 2021.
[17] Spring Data JPA. Overview. Disponível em: <https://spring.io/projects/spring-data-jpa>. Acesso em: 17 out. 2021.
[18] Spring Data Rest. Disponível em <https://spring.io/projects/spring-data-rest>. Acesso em: 29 out. 2021.
[19] Application Programming Interface (API). Disponível em: <https://www.ibm.com/cloud/learn/api />. Acesso em: 17 out. 2021.
[20] Rest. What is. Disponível em: <https://restfulapi.net/>. Acesso em: 17 out. 2021.
[21] VUE. Guia. Disponível em: <https://br.vuejs.org/v2/guide/>. Acesso em: 17 out. 2021.
[22] Vuetify. Why you should be using vuetify. Disponível em: <https://vuetifyjs.com/en/introduction/why-vuetify/>. Acesso em: 27 out. 2021.
[23] DAVIS, Fred D. Perceived usefulness, perceived ease of use, and user acceptance of information technology. **MIS quarterly**, p. 319-340, 1989.

MINISTÉRIO DA EDUCAÇÃO
Universidade Federal Rural do Semi-Árido – UFERSA
Centro de Ciências Exatas e Naturais – CCEN

ATA DE DEFESA DE TRABALHO DE CONCLUSÃO DE CURSO

Às 08h45min do dia dezoito de novembro de dois mil e vinte e um, na sala virtual do Google Meet *https://meet.google.com/que-kaqv-aou*, reuniu-se a Banca Examinadora de defesa de trabalho de conclusão de curso de autoria do aluno **VINICIUS GEORGE CARLOS NUNES,** aluno do curso de Bacharelado em Ciência da Computação desta Universidade, N° de matrícula **2015010719**, com o título *"**AVALIAÇÃO DE ABORDAGENS E DEFINIÇÃO DE DIRETRIZES PARA O DESENVOLVIMENTO DE APLICATIVOS MOBILE: um relato de experiência utilizando a abordagem não nativa de desenvolvimento**".* A Banca Examinadora ficou assim constituída por três membros: Prof. Dr. Paulo Gabriel Gadelha Queiroz, presidente da banca e orientador do Trabalho de Conclusão de Curso; Prof. Dr. Bruno de Sousa Monteiro e B.ela Fernanda Ferreira do Nascimento, como membros. Concluída a defesa, procedeu-se o julgamento pelos membros da banca examinadora, tendo o aluno obtido as seguintes notas: 9; 9; 9. Apuradas as notas verificou-se que o aluno foi aprovado com média geral 9. E para constar, eu, Paulo Gabriel Gadelha Queiroz, lavrei a presente ata que, após lida e aprovada pelos membros da banca examinadora, será assinada por todos.

Mossoró, 18 de novembro de 2021.

Assinatura dos membros da Banca Examinadora.

Prof. Dr. Paulo Gabriel Gadelha Queiroz – UFERSA
Presidente e orientador

Prof. Dr. Bruno de Sousa Monteiro - UFERSA
Primeiro Membro

B.ela Fernanda Ferreira do Nascimento – UFERSA
Segundo Membro