UNICESUMAR – UNIVERSIDADE CESUMAR
CENTRO DE CIÊNCIAS EXATAS TECNOLÓGICAS E AGRÁRIAS
CURSO DE GRADUAÇÃO EM ENGENHARIA DE SOFTWARE

JOÃO PEDRO ROSADA VOLPONI

COMPARATIVO DAS ABORDAGENS DE DESENVOLVIMENTO DE INTERFACES MÓVEIS EM SWIFT

MARINGÁ - PR
2023

JOÃO PEDRO ROSADA VOLPONI

COMPARATIVO DAS ABORDAGENS DE DESENVOLVIMENTO DE INTERFACES MÓVEIS EM SWIFT

Trabalho de Conclusão de Curso apresentado para obtenção do grau de bacharel em Engenharia de Software, do Centro Universitário Cesumar — Unicesumar

Orientador: Prof. Thiago Bussola

MARINGÁ - PR

2023

FOLHA DE APROVAÇÃO

JOÃO PEDRO ROSADA VOLPONI

COMPARATIVO DAS ABORDAGENS DE DESENVOLVIMENTO DE INTERFACES MÓVEIS EM SWIFT

Trabalho de Conclusão de Curso apresentado para obtenção do grau de bacharel em Engenharia de Software, do Centro Universitário Cesumar — Unicesumar, sob orientação do Prof. Thiago Bussola.

Aprovado em: ____ de _____ de _____.

BANCA EXAMINADORA

_____

Nome do professor – (Titulação, nome e Instituição)

_____

Nome do professor - (Titulação, nome e Instituição)

_____

Nome do professor - (Titulação, nome e Instituição)

AGRADECIMENTOS

A Deus, por ter me dado força e perseverança durante todos esses anos, permitindo que eu chegasse ao fim desse curso.

A meus pais, Maurício e Mônica, minha eterna admiração e gratidão. Vocês nunca mediram esforços para prover tudo o que era necessário para minha educação, sempre colocando o aprendizado como prioridade, o apoio e incentivo foram pilares essenciais em cada conquista alcançada durante a minha formação.

A todos os membros da minha família que acreditaram em mim e me apoiaram durante todos esses anos.

À Eloah que por todo apoio, amor e compreensão,

Ao Professor Thiago Bussola, por me ajudar durante todos os processos de criação deste trabalho, disponibilizando seu tempo e, também, por seu conhecimento, que foi fundamental para tornar este trabalho possível.

COMPARATIVO DAS ABORDAGENS DE DESENVOLVIMENTO DE INTERFACES MÓVEIS EM SWIFT

João Pedro Rosada Volponi

RESUMO

O desenvolvimento de aplicativos móveis se tornou uma prática crucial na era digital atual. Dentro deste contexto, a linguagem Swift, criada pela Apple, destaca-se como uma ferramenta robusta e versátil para desenvolvedores. Este estudo tem como foco analisar as diferentes abordagens disponíveis na linguagem Swift para a construção de interfaces, são elas: Storyboard, ViewCode e SwiftUI. Cada uma destas abordagens apresenta suas particularidades, vantagens e limitações. Por meio de um questionário dirigido a desenvolvedores e estudantes familiarizados com Swift, foram coletadas opiniões e feedbacks sobre cada abordagem. Os resultados mostram que o Storyboard é apreciado por sua natureza visual, o ViewCode é valorizado por sua flexibilidade, enquanto o SwiftUI, mais recente, é notável por sua sintaxe declarativa e integração com as últimas funcionalidades da Apple. Contudo, ele ainda apresenta limitações em termos de compatibilidade e curva de aprendizado. Um protótipo de aplicação desenvolvido no FIGMA também foi utilizado para avaliar a inclinação da comunidade em relação a estas abordagens, revelando uma preferência notável pelo ViewCode. Em essência, a escolha da abordagem ideal varia de acordo com as necessidades do projeto, experiência da equipe e especificações técnicas. A pesquisa ressalta a importância de escolher a abordagem correta para otimizar o desenvolvimento e garantir aplicações móveis eficientes e atraentes.

ABSTRACT

The development of mobile applications has become a crucial practice in the current digital era. Within this context, the Swift language, created by Apple, stands out as a robust and versatile tool for developers. This study focuses on analyzing the different approaches available in the Swift language for building interfaces, namely: Storyboard, ViewCode, and SwiftUI. Each of these approaches presents its particularities, advantages, and limitations. Through a questionnaire directed at developers and students familiar with Swift, opinions and feedback on each approach were collected. The results show that Storyboard is appreciated for its visual nature, ViewCode is valued for its flexibility, while SwiftUI, the most recent, is notable for its declarative syntax and integration with the latest Apple functionalities. However, the latter is presents limitations. A prototype application developed in FIGMA was also used to evaluate the community's inclination towards these approaches, revealing a notable preference for ViewCode. Essentially, the choice of the ideal approach varies according to the needs of the project, the team's experience, and technical specifications. The research highlights the importance of choosing the right approach to optimize development and ensure efficient and attractive mobile applications.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1. INTRODUÇÃO

Nos últimos anos, o desenvolvimento de aplicativos para dispositivos móveis se tornou um pilar fundamental na era digital, permitindo que empresas e indivíduos interajam com seu público-alvo de maneiras inovadoras. Com a crescente demanda por aplicações móveis, a eficiência e eficácia das ferramentas e linguagens de programação utilizadas para criar essas aplicações são essenciais. Entre as linguagens e frameworks disponíveis, a linguagem Swift, da Apple, tem ganhado destaque e popularidade devido à sua robustez e versatilidade. Contudo, com várias abordagens disponíveis para a construção de interfaces nessas linguagens – como Storyboard, ViewCode e SwiftUI – surge um dilema para desenvolvedores e equipes de design: qual abordagem escolher e quando?

Para este trabalho, foi feita a proposta de uma análise minuciosa das diferentes abordagens oferecidas pela linguagem Swift para a construção de interfaces. Foram avaliados seus pontos fortes, limitações e cenários ideais de aplicação, com base na experiência de desenvolvedores e estudantes que trabalham com a linguagem Swift. Além disso, por meio de um protótipo desenvolvido no FIGMA, buscamos entender as preferências da comunidade de desenvolvedores em relação a essas abordagens, avaliando os critérios de decisão para a escolha de um método em detrimento de outro.

Espera-se que, ao final deste estudo, possamos fornecer perspectivas valiosas para desenvolvedores, equipes de design e interessados em desenvolvimento iOS, auxiliando na tomada de decisão sobre qual abordagem utilizar de acordo com as necessidades específicas de cada projeto. Por meio desta pesquisa, buscamos não apenas analisar tecnicamente cada método, mas também compreender o panorama atual e as tendências emergentes no ecossistema Swift de desenvolvimento de interfaces.

2. OBJETIVO GERAL

Este estudo teve como objetivo analisar as possibilidades de desenvolvimento de interfaces para dispositivos móveis utilizando a linguagem de programação Swift, destacando as vantagens e desvantagens de cada método de criação.

2.1 Objetivos específicos

Para se alcançar o objetivo geral deste trabalho faz-se necessário efetuar os seguintes objetivos específicos.

1. Realizar uma análise detalhada da linguagem de programação Swift, abrangendo suas características, trajetória histórica e seu processo de evolução;
2. Identificar e descrever as principais ferramentas, recursos e frameworks disponíveis para o desenvolvimento de interfaces em Swift;
3. Desenvolver um questionário com perguntas relacionadas a vantagens e desvantagens dos métodos analisados;
4. Encaminhar o questionário para comunidades focadas em desenvolvimento na linguagem de programação Swift;
5. Realizar a análise dos dados coletados no formulário;
6. Desenvolver a aplicação demonstrada no questionário com a metodologia com maior porcentagem de escolha.

3. FUNDAMENTAÇÃO TEÓRICA

3.1 Lançamento primeiro iPhone

Segundo Ricardo R. Lecheta (2014, p. 23):

> A primeira versão do iPhone foi lançada em 2007, durante o WWDC (Apple Worldwide Developers Conference), que é um evento para desenvolvedores organizado em 2008, junto com a abertura da App Store, a loja que facilita o processo de distribuição de aplicativos pelo mundo.

O lançamento do primeiro iPhone, em 2007, marcou um ponto de virada revolucionário na indústria de tecnologia e comunicações. Conforme mencionado por Ricardo R. Lecheta em seu livro "Desenvolvendo para iPhone e iPad", de 2014, esse marco se deu durante a Apple Worldwide Developers Conference (WWDC), um evento anual crucial para desenvolvedores, organizado pela Apple. Este evento serviu como uma plataforma estratégica para apresentar ao mundo o dispositivo que redefiniu a maneira como interagimos com a tecnologia móvel.

O iPhone introduziu uma abordagem totalmente nova para os smartphones, combinando um design elegante e minimalista com uma interface de usuário intuitiva e um ecossistema altamente integrado. Ao lançar o iPhone, a Apple não apenas apresentou um dispositivo, mas sim uma experiência completamente nova para os consumidores.

Entretanto, é importante destacar que, embora o iPhone tenha sido lançado em 2007, a capacidade de desenvolver aplicativos para essa plataforma só foi anunciada no ano seguinte, em 2008, durante o mesmo evento WWDC. Além disso, a Apple anunciou a criação da App Store, um componente fundamental que democratizou o processo de distribuição de aplicativos em escala global. A App Store simplificou a maneira como os desenvolvedores podiam oferecer suas criações para um amplo público, permitindo que usuários de todo o mundo tivessem fácil acesso a uma vasta gama de aplicativos diretamente de seus dispositivos iPhone.

Esse lançamento não apenas estabeleceu a Apple como líder na indústria de smartphones, mas também inaugurou uma nova era na computação móvel, impulsionando a inovação e estabelecendo um padrão elevado para a concorrência. A introdução do iPhone e da App Store transformou fundamentalmente a forma como interagimos com a tecnologia e desencadeou uma revolução na maneira como os

aplicativos são desenvolvidos, distribuídos e utilizados no cenário digital contemporâneo.

3.2 Origem do Objetive-C

De acordo com Ricardo R. Lecheta (2014, p. 24).

> A linguagem Objective-C foi criada por Brad Cox e Tom Love no início da década de 1980, e, anos mais tarde, em 1988, a NeXT de Steve Jobs adquiriu a linguagem e licenciou-se. O Objective-C é derivado da linguagem C tradicional e obteve uma forte influência da linguagem Smalltalk(Linguagem de programação Orientada a Objetos onde todos os dados ou informações são objetos).

A linguagem Objective-C é um marco significativo na história da programação para dispositivos Apple, tendo sua origem no início da década de 1980, quando foi criada por Brad Cox e Tom Love. Entretanto, seu verdadeiro potencial se tornou aparente anos depois, em 1988, quando a empresa NeXT, de propriedade de Steve Jobs, adquiriu a linguagem e obteve uma licença para seu uso. Essa aquisição abriu as portas para o desenvolvimento de software inovador para a crescente linha de produtos Apple, inaugurando uma era de criatividade e inovação na indústria da tecnologia.

O Objective-C se destaca por ser uma evolução da linguagem C tradicional, mantendo a eficiência e o desempenho da linguagem subjacente, enquanto incorpora fortemente os princípios da programação orientada a objetos. Sua forte influência da linguagem Smalltalk, que é conhecida por tratar todos os dados e informações como objetos, adicionou uma camada de abstração e flexibilidade que tornou o desenvolvimento de aplicativos para dispositivos Apple uma experiência única e poderosa. Essa combinação de elementos permitiu a criação de interfaces de usuário elegantes e a integração perfeita de funcionalidades avançadas, solidificando a posição do Objective-C como a linguagem de escolha para desenvolvedores Apple por muitos anos.

3.3 Xcode

Segundo a Apple Developer [2023], Xcode é um conjunto de ferramentas que os desenvolvedores usam para criar aplicativos para plataformas Apple. Dessa

forma, o Xcode gerencia todo o seu fluxo de trabalho de desenvolvimento, desde a criação do aplicativo até o teste, otimização e envio para a App Store.

O Xcode é um conjunto de ferramentas essenciais para os desenvolvedores que buscam criar aplicativos destinados às diversas plataformas da Apple, como iOS, macOS, watchOS e tvOS. O Xcode é uma suíte abrangente que abrange todo o ciclo de vida de desenvolvimento de um aplicativo, simplificando o processo para os programadores. Por meio do Xcode, os desenvolvedores podem criar, testar e otimizar suas aplicações, permitindo que alcancem a máxima qualidade e desempenho antes de disponibilizá-las na App Store, o mercado oficial de aplicativos da Apple.

Além disso, o Xcode se destaca como uma ferramenta de gerenciamento de projeto crucial, pois auxilia os desenvolvedores a organizar seus códigos e recursos, o que é fundamental para manter a eficiência e a produtividade durante o desenvolvimento. O fato de abranger todo o fluxo de trabalho, desde a fase inicial de criação até a etapa final de envio para a App Store, demonstra como o Xcode é uma solução abrangente e integrada para a criação de aplicativos de alta qualidade que atendem aos rigorosos padrões da Apple e proporcionam aos usuários uma experiência excepcional.

3.4 Origem do Swift

Ricardo R. Lecheta (2014, p. 24) afirma que "Na WWDC 2014, a Apple introduziu a linguagem Swift, com uma sintaxe moderna e simples, visando aumentar a produtividade no desenvolvimento de aplicativos e facilitar o aprendizado dos iniciantes."

Em 2014, durante a Worldwide Developers Conference (WWDC) da Apple, a empresa de tecnologia apresentou ao mundo a linguagem de programação Swift. Nas palavras de Ricardo R. Lecheta (2014, p. 24) “essa nova linguagem foi projetada com uma sintaxe simples e moderna, com o objetivo de tornar o desenvolvimento de aplicativos mais produtivo”. Um dos principais atrativos da Swift era a sua acessibilidade, tornando-a uma escolha ideal para iniciantes que desejavam ingressar no mundo do desenvolvimento de software. A simplicidade da linguagem também facilitava o aprendizado para os novatos no campo.

Lançado em 2014, o Swift é a linguagem de programação com crescimento mais rápido da história e combina o desempenho e a eficiência de linguagens compiladas com a simplicidade e a interatividade das populares linguagens de script. A Apple também lançou o website Swift.org, que traz informações detalhadas e documentação técnica.

Conforme relata a Apple Cupertino (2015), a linguagem de programação Swift é um código livre. De acordo com o site da organização Swift [2023] "Swift é uma linguagem de programação de uso geral acessível para iniciantes e poderosa para especialistas. É rápido, moderno, seguro e uma alegria escrever".

Em dezembro de 2015, a Apple deu um passo adiante ao anunciar que a Swift se tornaria uma linguagem de código aberto, abrindo suas portas para a comunidade de desenvolvedores. A Swift combinava a eficiência e o desempenho das linguagens de programação compiladas com a facilidade de uso e interatividade das linguagens de script, tornando-a uma opção versátil para programadores.

O lançamento do site Swift.org forneceu documentação técnica e informações detalhadas, consolidando o compromisso da Apple com o desenvolvimento e aprimoramento contínuo da Swift. A linguagem tornou-se conhecida por sua rapidez, modernidade e segurança, tornando-a uma escolha atraente tanto para iniciantes quanto para especialistas na programação.

3.5 UIKit

Como afirma a Apple Developer [2023, s/p],

> A estrutura UIKit fornece os objetos principais necessários para criar aplicativos para iOS e tvOS. Você usa esses objetos para exibir seu conteúdo na tela, para interagir com esse conteúdo e para gerenciar interações com o sistema. Os aplicativos dependem do UIKit para seu comportamento básico, e o UIKit fornece muitas maneiras de personalizar esse comportamento para atender às necessidades específicas.

O UIKit é um framework de interface do usuário desenvolvido pela Apple para o desenvolvimento de aplicativos iOS. Esse framework desempenha um papel crítico no ecossistema de desenvolvimento da Apple, pois foi lançado em 2007, juntamente com o primeiro iPhone e o sistema operacional móvel da Apple. Isso marca o início da era dos smartphones da Apple e, por extensão, o início do desenvolvimento de aplicativos móveis para a plataforma iOS. O lançamento do iPhone e do UIKit

representou uma revolução na forma como as pessoas interagem com a tecnologia, introduzindo uma experiência de toque e gestos intuitivos. Desde então, o UIKit tem sido a base para o desenvolvimento de aplicativos iOS, permitindo que desenvolvedores criem interfaces de usuário interativas e responsivas.

Como menciona a Apple Developer [2023, s/p], "o UIKit é um framework de interface do usuário para o desenvolvimento de aplicativos iOS, o qual foi lançado em 2007 junto com o lançamento do primeiro iPhone e primeiro sistema operacional móvel da Apple."

A estrutura UIKit fornece os componentes fundamentais necessários para criar aplicativos tanto para iOS quanto para tvOS. Os objetos disponibilizados pelo UIKit são essenciais para a exibição de conteúdo na tela dos dispositivos, a interação dos usuários com esse conteúdo e o gerenciamento das interações com o sistema operacional.

Em outras palavras, o UIKit oferece as ferramentas e recursos que permitem que os desenvolvedores criem a interface de usuário de seus aplicativos e controlam como os usuários interagem com ela. Os aplicativos móveis dependem do UIKit para estabelecer seu comportamento básico, tornando-o uma parte crucial do desenvolvimento iOS. Assim como destaca a Apple Developer [2023], "a estrutura dos aplicativos UIKit é baseada no padrão de design Model - ViewController (MVC), em que os objetos são divididos por sua finalidade."

O UIKit também oferece flexibilidade e personalização, permitindo que os desenvolvedores adaptem o comportamento padrão de acordo com as necessidades específicas de seus aplicativos. A estrutura dos aplicativos UIKit é baseada no padrão de design Model-View-Controller (MVC). O padrão MVC é uma abordagem amplamente aceita para organizar o código de um aplicativo, na qual os objetos são separados de acordo com suas funções.

O "Model" representa a lógica de negócios e os dados do aplicativo, o "View" cuida da apresentação e exibição da interface de usuário, e o "Controller" age como um intermediário que facilita a comunicação entre o Modelo e a Visão. Essa separação de responsabilidades é fundamental para a manutenção, escalabilidade e organização do código de um aplicativo. O UIKit incentiva os desenvolvedores a adotar esse padrão de design, o que torna mais fácil criar aplicativos iOS bem estruturados e gerenciáveis.

3.5.1 Storyboard

Conforme descrito por Apple Developer [2023, s/p] "Storyboard é uma representação visual da interface de um aplicativo iOS, que mostra telas e suas conexões". Assim, fica claro que ele é formado por várias cenas. No Xcode, há um editor para projetar essa interface e adicionar elementos gráficos.

Storyboard é uma ferramenta essencial no desenvolvimento de aplicativos iOS, permitindo uma visualização clara da interface do usuário (UI). Dentro do ambiente de desenvolvimento integrado (IDE) da Apple, o Xcode, os desenvolvedores têm a capacidade de projetar a aparência do aplicativo de forma intuitiva, visualizando as interações entre diferentes telas e elementos. Neste contexto, uma "cena" é uma tela específica do aplicativo, e um Storyboard compreende uma coleção destas cenas, mapeando a estrutura e o fluxo do aplicativo.

O Xcode disponibiliza um editor visual específico para trabalhar com Storyboards. Através deste editor, os desenvolvedores podem integrar elementos gráficos como botões, campos de texto e imagens, além de estabelecer conexões e transições entre as cenas. Esta ferramenta gráfica facilita e agiliza o processo de design e desenvolvimento, proporcionando uma representação visual abrangente da experiência do usuário e tornando a criação de aplicativos iOS mais acessível e eficiente.

3.5.2 ViewCode

De acordo com Giovanna Moeller (2022, p.15), "Quando criamos um layout com view code, significa que estamos criando via código. Não usando storyboards e nem xibs, portanto, não utilizamos ferramentas como interface builder, que permite arrastar os elementos para dentro da tela."

Criar layouts de interface de usuário usando "view code" é uma abordagem alternativa ao desenvolvimento de aplicativos iOS em relação ao uso de storyboards ou xibs. A expressão "view code" se refere ao processo de criar a interface do usuário do aplicativo por meio da escrita de código de programação, em oposição à criação visual de interface de usuário oferecida por ferramentas como storyboards e xibs.

Quando os desenvolvedores optam por seguir a abordagem "view code," eles estão criando todas as interfaces do aplicativo programaticamente, geralmente em

arquivos de código-fonte, como arquivos Swift. Em vez de usar uma interface visual para arrastar e soltar elementos, eles definem cada elemento da interface, como botões, etiquetas, caixas de texto e outros, em código. Isso oferece um maior nível de controle sobre o layout e o comportamento da interface, bem como a flexibilidade para criar interfaces personalizadas.

A abordagem "view code" é preferida por alguns desenvolvedores por várias razões, incluindo a capacidade de versionar o layout da interface no controle de código-fonte, facilitando a colaboração em equipe e a resolução de conflitos. Além disso, ela pode ser particularmente útil para aplicativos com interfaces complexas e personalizadas.

No entanto, é importante notar que a escolha entre "view code" e o uso de storyboards/xibs depende das preferências do desenvolvedor e das necessidades específicas do projeto. Algumas equipes podem preferir uma abordagem híbrida, usando storyboards para algumas partes da interface e "view code" para outras. Em última análise, a decisão é influenciada pela eficiência, manutenibilidade e requisitos do aplicativo.

3.6 SwiftUI

Conforme Apple Developer [2023, s/p], "SwiftUI é uma forma moderna de declarar interfaces de usuários para qualquer plataforma Apple."

SwiftUI é um framework revolucionário que simplifica a criação de interfaces de usuário para aplicativos em várias plataformas da Apple. Com SwiftUI, os desenvolvedores podem definir a interface de seus aplicativos usando uma abordagem declarativa, o que significa que eles descrevem como a interface deve ser, em vez de programar manualmente cada elemento. Isso permite a criação de aplicativos visualmente atraentes e dinâmicos de forma mais eficiente do que as abordagens anteriores, tornando o desenvolvimento mais rápido e intuitivo.

De acordo com Apple Developer [2023, s/p], "SwiftUI fornece visualizações, controles e estruturas de layout para declarar a interface do usuário do seu aplicativo. A estrutura fornece manipuladores de eventos para fornecer toques, gestos e outros tipos de entrada ao seu aplicativo."

SwiftUI oferece um conjunto abrangente de recursos para a criação de interfaces de usuário. Isso inclui visualizações (como botões e listas), controles (como

deslizadores e caixas de texto) e estruturas de layout que permitem organizar os elementos na tela. Além disso, o framework simplifica a manipulação de eventos, como toques e gestos, facilitando a resposta do aplicativo às interações do usuário.

Além disso, o SwiftUI possui umSwiftUI oferece um conjunto abrangente de recursos para a criação de interfaces de usuário. Isso inclui visualizações (como botões e listas), controles (como deslizadores e caixas de texto) e estruturas de layout que permitem organizar os elementos na tela. Além disso, o framework simplifica a manipulação de eventos, como toques e gestos, facilitando a resposta do aplicativo às interações do usuário.

E ainda mais, o SwiftUI possui um sistema de ligação de dados eficiente que ajuda a gerenciar o fluxo de dados dos modelos do aplicativo para as visualizações e controles, mantendo a interface sempre sincronizada com os dados subjacentes. sistema de ligação de dados eficiente que ajuda a gerenciar o fluxo de dados dos modelos do aplicativo para as visualizações e controles, mantendo a interface sempre sincronizada com os dados subjacentes.

De acordo com a Apple Developer [2023, s/p],

> Você pode integrar visualizações SwiftUI com objetos das estruturas UIKit, AppKit e WatchKit para aproveitar ainda mais a funcionalidade específica da plataforma. Podendo também personalizar o suporte de acessibilidade no SwiftUI e localizar a interface do seu aplicativo para diferentes idiomas, países ou regiões culturais.

SwiftUI é altamente flexível e permite que os desenvolvedores integrem visualizações criadas com SwiftUI a objetos das estruturas UIKit (iOS), AppKit (macOS) e WatchKit (watchOS). Isso facilita a migração gradual de aplicativos existentes para a nova tecnologia ou a combinação de SwiftUI com abordagens mais antigas. Além disso, o framework oferece suporte para acessibilidade, o que significa que os aplicativos podem ser desenvolvidos para serem acessíveis a uma ampla variedade de usuários. Além disso, o suporte à localização permite que os desenvolvedores adaptem facilmente a interface de seus aplicativos para diferentes idiomas, países ou regiões culturais, tornando-os verdadeiramente globais.

4. TRABALHOS RELACIONADOS

De acordo com o autor Brenno Giovanini de Moura (2020), que apresenta uma proposta de implementar um framework para construção de interfaces de usuário usando o paradigma de programação declarativa nos dispositivos iOS. O desenvolvimento é composto de vários conceitos e recursos de programação, discutindo a integração com a linguagem de programação Swift e com o sistema operacional iOS. Os objetos declarativos encapsulam os objetos de interface do framework UIKit, sendo implementados os essenciais para permitir o uso prático em aplicações reais.

A avaliação sobre framework desenvolvido por este trabalho consistiu em aplicar casos de teste que simulam uma tela ou um aplicativo prático, coletando e analisando dados de desempenho. Com isso, o trabalho mostrou-se consistente e aplicável, necessitando do desenvolvimento de novos conceitos para integralizar o framework.

5. METODOLOGIA

Nesta seção, serão detalhadas as categorias de classificação da pesquisa, relacionadas à abordagem adotada para a condução do estudo de caso. Abordaremos a natureza da pesquisa, seus objetivos, as fontes de informação utilizadas, os procedimentos técnicos empregados, as limitações da pesquisa e as considerações éticas subjacentes. Para uma visão resumida da classificação da pesquisa, consulte o Quadro 1 que segue abaixo:

Quadro 1 – Classificação da pesquisa

Pesquisa	**Classificação**
Natureza	Aplicada
Objetivos	Descritiva e Comparativa
Abordagem	Descritiva e Comparativa
Fonte de Informação	Campo
Procedimentos técnicos	Estudo bibliográfico Descrição dos métodos Questionário Análise dos dados coletados Construção da aplicação

Fonte: elaborado pelo autor (2023).

5.1 Quanto à natureza e aos objetivos da pesquisa

Quanto à natureza, a pesquisa se classifica como aplicada, a qual visa melhorar processos ou atender às necessidades da sociedade e da indústria. Este trabalho gera informações para empresas de tecnologias e estudantes que possam conhecer e entender as possibilidades de desenvolvimento de interfaces no desenvolvimento de software.

Quanto aos objetivos, a pesquisa se classifica como Descritiva e Comparativa, pois tende a descrever detalhadamente a linguagem de programação em questão, suas características, vantagens e desvantagens e Comparativa pelo fato de comparar os métodos de desenvolvimento da própria linguagem de programação.

5.2 Quanto à abordagem da pesquisa

A presente pesquisa utiliza uma abordagem de métodos comparativos e descritivos, onde o objetivo principal é descrever e documentar o caso em questão. Isso envolve coletar informações sobre o caso, como histórico, contexto, características, eventos e significados. Já na abordagem comparativa, estudamos para fins de comparação, utilizado para identificar semelhanças e diferenças entre casos e tirar conclusões com base nessas comparações.

5.3 Quanto aos procedimentos técnicos

Durante o processo de construção dessa pesquisa, foram elaborados objetivos específicos para a análise e compreensão do assim objeto, que foram delimitados:

Para o objetivo específico 1 foi realizado um estudo bibliográfico aprofundado sobre a linguagem de programação Swift, esse estudo foi realizado em bases de dados científicas como. Google Scholar, ACM Digital Library e documentações e livros oficiais da Apple (APPLE DEVELOPER [2023]).

Para o objetivo específico 2: Os resultados coletados do objetivo específico 1 foram organizados em duas principais seções sendo UIKit e SwiftUI, descrevendo seus recursos disponíveis até o momento.

Para o objetivo específico 3: Desenvolver um questionário utilizando o Google Forms, com perguntas relacionadas a vantagens e desvantagens de cada método de desenvolvimento.

Para o objetivo específico 4: Divulgação do questionário em comunidades de desenvolvimento iOS como para programadores atuantes no mercado de trabalho.

Para o objetivo específico 5: Se refere a análise dos dados coletados pelo questionário, os dados coletados foram tratados e organizados na seção 6 para apresentação.

Para o objetivo específico 6: Desenvolvimento das interfaces da aplicação demonstrada no questionário, utilizando todas as vantagens do método que obteve a maior porcentagem de respostas.

6. APRESENTAÇÃO DE RESULTADOS

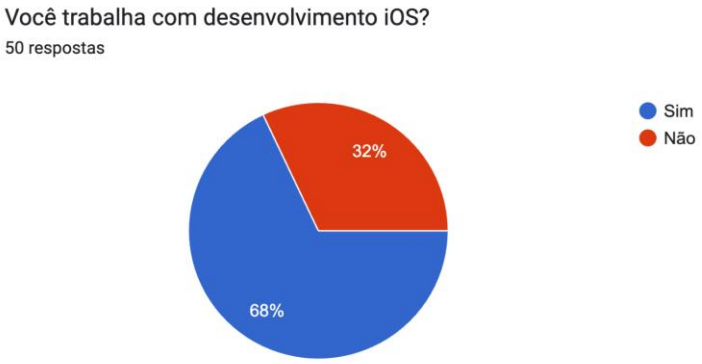
6.1 Construção do questionário

O questionário é composto por 17 perguntas, das quais 15 são obrigatórias. Seu principal objetivo é compreender as vantagens e desvantagens dos métodos de criação de interfaces, fundamentado nas experiências de desenvolvimento na linguagem Swift. O formulário foi destinado apenas aqueles que estão estudando desenvolvimento iOS ou programadores já inseridos no mercado de trabalho.

6.2 Resultados do questionário

O questionário foi respondido por 50 indivíduos, conforme apresentado na Figura 1 sendo 32% (16 indivíduos) que ainda não trabalham com desenvolvimento iOS portanto são estudantes da linguagem de programação Swift, e 68% (34 indivíduos) que atuam como desenvolvedores iOS na linguagem de programação Swift

Figura 1 – Atuante ou não como desenvolvedor iOS



Fonte: Elaborado pelo autor (2023).

6.2.1 Experiência com Desenvolvimento

Foram contabilizadas um total de 50 respostas ao questionário, o qual se propôs a entender as vantagens e desvantagens dos métodos de criação de interfaces na linguagem Swift. Este retorno é um indicativo valioso, pois dá uma perspectiva sobre a experiência e trajetória dos desenvolvedores iOS, sendo estes tanto novatos quanto veteranos na área. Através desta pesquisa, eles esperam contribuir para essa diferença no mundo do desenvolvimento iOS

Dentre as respostas recebidas, foi possível identificar uma distribuição interessante quanto ao tempo de atuação no mercado de desenvolvimento iOS. Conforme ilustrado na Figura 2, que apresenta um gráfico dos resultados, 36% dos respondentes, correspondendo a 18 indivíduos, estão na área há menos de um ano, indicando uma onda recente de profissionais ingressando no campo. Seguindo essa análise, 32% (ou 16 indivíduos) possuem uma jornada de trabalho que se estende entre 1 e 3 anos, mostrando um nível intermediário de experiência. Finalmente, o mesmo percentual de 32%, também representado por 16 indivíduos, têm uma bagagem de 3 anos ou mais na área, sugerindo uma sólida compreensão e domínio dos desafios e nuances do desenvolvimento iOS. Essa divisão nos ajuda a compreender a diversidade de experiências e perspectivas entre os participantes, o que pode enriquecer a análise dos resultados do questionário.

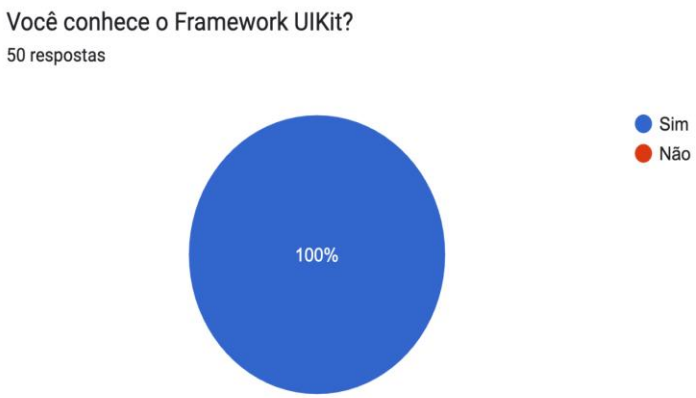
Figura 2 – Experiência em desenvolvimento.

Fonte: Elaborado pelo autor (2023).

6.2.2 Conhecimento sobre o Framework UIKit?

Nesta pesquisa, buscamos entender o grau de familiaridade dos desenvolvedores com o framework UIKit, fundamental para a criação de interfaces gráficas em aplicativos iOS. Como destacado por Christian Keur e Aaron Hillegass (2016) em seu livro seminal, 'iOS Programming: The Big Nerd Ranch Guide'. o UIKit não apenas fornece um conjunto robusto de classes para a concepção de interfaces gráficas, mas também abraça uma variedade de funções interativas que definem a relação do usuário com o aplicativo. No contexto atual, onde a experiência do usuário é fundamental para o sucesso de um aplicativo, entender o domínio e aplicação eficaz do UIkit por parte dos desenvolvedores se torna mais crucial do que nunca. Surpreendentemente, o feedback foi unânime: 100%(ou 50 indivíduos) dos respondentes afirmaram conhecer o UIKit, conforme apresentado na Figura 03. Esta resposta evidencia a centralidade e relevância do UIKit no universo do desenvolvimento iOS, sendo uma ferramenta indiscutivelmente reconhecida e utilizada pelos profissionais da área.

Figura 3 – UIKit



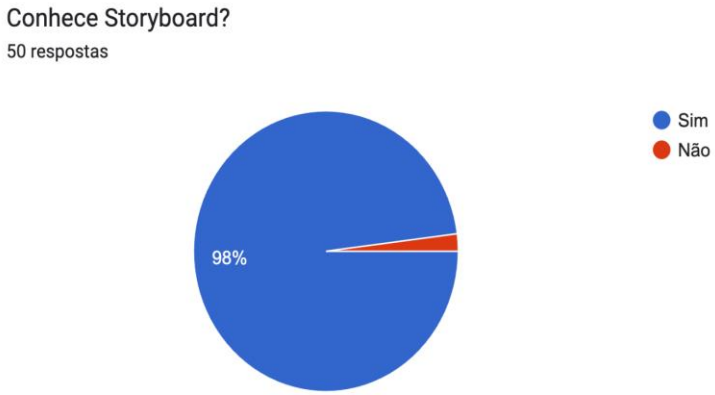
Fonte: Elaborado pelo autor (2023).

6.2.3 Conhecimento sobre Storyboar

A respeito do Storyboard, uma ferramenta visual utilizada para construir interfaces de usuário no iOS, os resultados também foram significativos. Conforme ilustrado na Figura 04, 98% dos respondentes, representando 49 indivíduos, afirmaram estar familiarizados com ela. No entanto, houve uma pequena parcela, de 2% ou um único indivíduo, que indicou não conhecer o Storyboard.

Esta predominância indica o quanto o Storyboard é amplamente aceito e usado entre os desenvolvedores, enquanto também destaca a presença de profissionais que podem preferir abordagens alternativas ou ainda não tiveram a oportunidade de explorá-lo. Como Christian Keur e Aaron Hillegass (2016) observam, "O verdadeiro poder de um aplicativo iOS vem da sua capacidade de ser intuitivo, interativo e envolvente"

Figura 4 – Conhecimento sobre Storyboard.



Fonte: Elaborado pelo autor (2023).

6.2.3.1 Vantagens e desvantagens de se utilizar o método Storyboard.

Ao questionar os desenvolvedores sobre as vantagens do uso do Storyboard, após confirmarem seu conhecimento sobre a ferramenta, eles tiveram a oportunidade de expressar suas percepções e experiências de maneira aberta e detalhada. Esta pergunta proporcionou um insight valioso sobre o que os profissionais realmente valorizam ao adotar tal método em seus processos de desenvolvimento.

A diversidade das respostas recebidas foi notável, refletindo as diferentes jornadas e experiências individuais com o Storyboard. Para oferecer uma visão mais clara dessas perspectivas, foram selecionadas algumas das respostas mais representativas e esclarecedoras que destacam as principais vantagens e desvantagens percebidas pelos desenvolvedores ao utilizar este método.

6.2.3.1.1 Vantagens

Segundo os participantes da pesquisa, a utilização do Drag and Drop (Arrastar e soltar), que simplifica a compreensão do desenvolvimento iOS, permitindo uma visualização intuitiva do fluxo da interface. Muitos desenvolvedores relatam que conseguem aprender a construir interfaces de maneira rápida, graças à possibilidade de adicionar elementos de maneira lúdica e simples.

Além disso, para quem está entrando agora no mundo iOS, a experiência com ferramentas como o Storyboard é extremamente valiosa, esse método ajuda a entender vários processos vinculados ao Interface Builder, facilitando bastante a curva de aprendizado para novatos. Isso faz do desenvolvimento iOS uma área acessível, onde mesmo indivíduos com pouco ou nenhum conhecimento prévio podem criar um app básico com relativa facilidade.

Outras vantagens incluem as animações integradas, segues para transição entre telas, o auto Layout e a funcionalidade do Interface Builder que permite arrastar e soltar elementos diretamente no código. Essas características não apenas potencializam a eficiência do desenvolvimento, mas também enriquecem a experiência do usuário final, resultando em aplicativos mais atraentes e funcionais, esses recursos tornam o desenvolvimento de aplicativos iOS uma escolha atrativa para os desenvolvedores iniciantes.

6.2.3.1.2 Desvantagens

De acordo com os participantes existem algumas desvantagens notáveis quando se trata de desenvolvimento iOS, especialmente quando usando a ferramenta Storyboard. Em primeiro lugar, a escalabilidade pode ser um problema. A performance, muitas vezes, não atende às expectativas e a manutenção principalmente em termos de layout e versionamento, pode ser desafiadora. Há relatos de perda de referências dos elementos durante as atualizações de verão, o que complica ainda mais o processo.

Outra questão crítica é a dificuldade em trabalhar em equipe em projetos que utilizam Storyboard, devido aos frequentes conflitos de merge. Isso se torna ainda mais pronunciado em projetos maiores, onde tais conflitos podem se tornar comuns e dificultar a colaboração entre desenvolvedores.

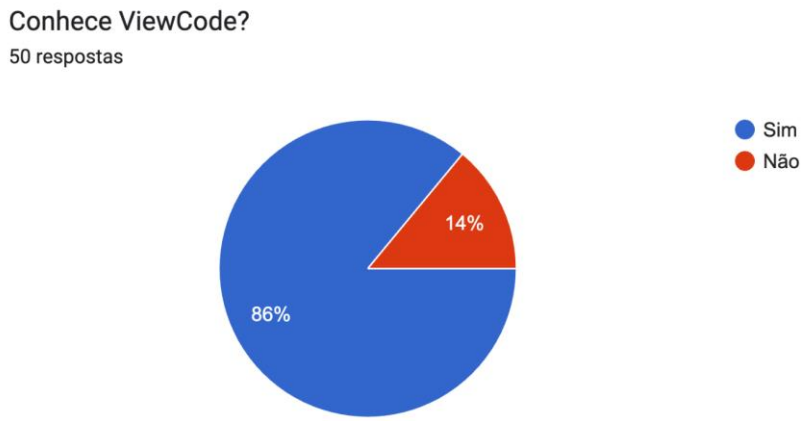
Além disso, embora o versionamento seja uma ferramenta essencial para o desenvolvimento, sua manutenção e escalabilidade, em alguns cenários iOS, não são ideais, e a performance pode ser insuficiente, mas como se esses desafios não fossem mais suficientes, à medida que um aplicativo cresce e possui muitas telas, o processo de manutenção se torna mais demorado e complicado, exigindo uma atenção extra por parte dos desenvolvedores. Portanto, enquanto o desenvolvimento

com Storyboard oferece várias vantagens, também vem com seu conjunto de desafios que precisam ser cuidadosamente gerenciados.

6.2.4 Conhecimento sobre ViewCode

Em relação ao ViewCode, uma abordagem programática para construção de interfaces no iOS, os dados coletados mostraram um cenário interessante. Conforme detalhado na Figura 05, 86% dos respondentes, equivalente a 43 indivíduos, afirmaram estar familiarizados com esta técnica. Contudo, uma porcentagem de 14%, que se traduz em 7 indivíduos, mencionou não ter conhecimento sobre o ViewCode. Esses resultados sugerem que, embora o ViewCode seja amplamente reconhecido e adotado por uma grande parte dos desenvolvedores, ainda existe uma parcela que não teve contato ou preferiu outras abordagens para a criação de interfaces.

Figura 5 – Conhecimento sobre ViewCode.



Fonte: Elaborado pelo autor (2023).

6.2.4.1 Vantagens e desvantagens de se utilizar o método ViewCode.

Quando os desenvolvedores foram questionados sobre as vantagens e desvantagens do uso do ViewCode, com base na confirmação de seu conhecimento prévio sobre o método, eles tiveram a chance de compartilhar suas opiniões fundamentadas em suas vivências profissionais. Esta indagação proporcionou uma

compreensão profunda não apenas sobre os aspectos positivos, mas também sobre os desafios associados ao uso do ViewCode em projetos reais.

As respostas variaram, evidenciando uma ampla gama de experiências e pontos de vista. Para sintetizar e apresentar uma visão equilibrada, foram selecionadas algumas das opiniões mais significativas e elucidativas. Essas destacam tanto as vantagens quanto as desvantagens percebidas pelos profissionais ao optar pelo método ViewCode em suas atividades de desenvolvimento.

As escolhas de desenvolvimento e design vão além da estética, elas impactam a funcionalidade e a experiência do usuário. Neste contexto, o debate em torno do ViewCode não é apenas sobre sua aparência ou praticidade, mas sobre sua eficácia e utilidade em projetos reais.

6.2.4.1.1 Vantagens

Como os participantes relatam, uma das principais vantagens do ViewCode é a facilidade na manutenção do código. Sem a dependência de arquivos de interface gráfica volumosos, os desenvolvedores têm menos conflitos durante o processo de merge, o que simplifica a colaboração e agiliza o fluxo de trabalho.

O ViewCode se destaca pela reusabilidade e flexibilidade que oferece. Os componentes são facilmente reutilizáveis em diferentes partes do projeto, tornando o código mais limpo e modular. Além disso, a refatoração torna-se mais ágil, e a documentação do código é facilitada, proporcionando uma compreensão clara do funcionamento das funções e classes.

Adotar o ViewCode significa ter um controle muito maior sobre todos os aspectos do código. Desde a documentação até a manutenção, passando pela utilização de Pods(Cocoa Pods) e adoção de padrões de arquitetura, os desenvolvedores têm uma visão mais clara e direta do que está acontecendo, permitindo uma modularização eficaz.

Com ViewCode, os problemas comuns associados ao versionamento principalmente em relação a layout e interações, são minimizados. A reusabilidade do código também é uma característica marcante, permitindo que os desenvolvedores aproveitem o trabalho anterior em novos contextos, reduzindo o trabalho.

Por fim, o ViewCode é notavelmente escalável, atendendo tanto pequenos projetos quanto a grandes aplicações. A possibilidade de trabalho em equipe é

otimizada, com cada membro tendo a capacidade de contribuir de maneira eficaz. A manutenção e documentação tornam-se tarefas mais simples, permitindo que a equipe se concentre em inovação e aprimoramento contínuos.

6.2.4.1.2 Desvantagens

Conforme descrito pelos participantes, uma desvantagem notória do ViewCode é sua curva de aprendizado elevada. Diferente de abordagens que utilizam interfaces gráficas, como o Storyboard, o ViewCode exige que os desenvolvedores tenham uma compreensão mais profunda de como o código se traduz visualmente na interface do usuário. Isso representa um desafio significativo para aqueles que estão acostumados a ferramentas visuais, podendo prolongar o período de adaptação à nova metodologia.

Outro ponto de atenção no uso do ViewCode é a necessidade de compilar todo o projeto frequentemente para ver as alterações realizadas nas views. Esse processo pode ser demorado e menos eficiente em comparação com métodos que permitem a visualização instantânea das mudanças, como o Storyboard.

Além disso, antes de iniciar o desenvolvimento com ViewCode, é preciso configurar o projeto adequadamente, o que pode ser uma etapa adicional e às vezes complicada. A maior desvantagem, no entanto, reside na necessidade de mentalizar a interface do usuário (UI) enquanto se lê o código, ao invés de visualizá-la diretamente no storyboard. Isso exige um nível de abstração e compreensão visual que pode aumentar significativamente a curva de aprendizado.

Projetos desenvolvidos em ViewCode geralmente requerem desenvolvedores com maior especialização e experiência. Devido à complexidade e às nuances da escrita e manutenção do código de interface sem o auxílio visual direto, profissionais mais qualificados são necessários para garantir a qualidade e a eficiência do desenvolvimento.

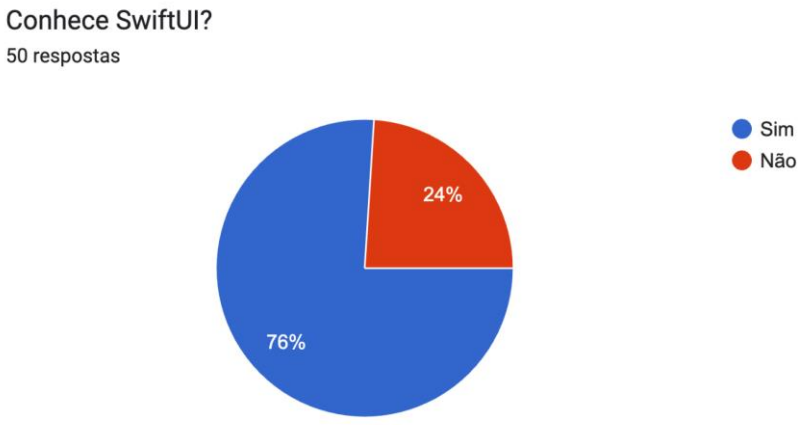
Por fim, desenvolver um projeto robusto e bem estruturado em ViewCode pode demandar mais tempo em comparação com métodos que utilizam interfaces gráficas como o storyboard. A escrita manual de cada elemento da interface e a necessidade constante de compilação para visualização das alterações contribuem para uma maior duração no ciclo de desenvolvimento do projeto.

6.2.5 Conhecimento sobre SwiftUI

Ao abordar o tema SwiftUI, uma moderna interface toolkit que permite a construção de UIs de forma declarativa para todas as plataformas da Apple, como Paul Hudson relata, "SwiftUI é um kit de ferramentas de interface de usuário que nos permite projetar aplicativos de forma declarativa. Essa é uma maneira elegante de dizer que dizemos ao SwiftUI como queremos que nossa interface de usuário seja e funcione, e ele descobre como fazer isso acontecer à medida que o usuário interage com ela" foram constatados resultados interessantes sobre sua popularidade entre os desenvolvedores.

De acordo com os dados representados na Figura 05, 76% dos entrevistados, o que corresponde a 38 indivíduos, afirmaram ter conhecimento e experiência com o SwiftUI. Por outro lado, 24%, ou 12 indivíduos, indicaram não estar familiarizados com esta ferramenta. Estes números demonstram que, apesar da ascensão e das inovações proporcionadas pelo SwiftUI, ainda há profissionais que não se aventuraram ou tiveram a oportunidade de explorar essa nova abordagem no universo do desenvolvimento iOS.

Figura 6 – Conhecimento sobre SwiftUI.



Fonte: Elaborado pelo autor (2023).

6.2.5.1 Vantagens e desvantagens de se utilizar o método SwiftUI.

6.2.5.1.1 Vantagens

Os participantes enfatizaram a importância de poder visualizar em tempo real o que está sendo desenvolvido. Esse recurso acelera o processo de desenvolvimento e permite correções imediatas, melhorando a eficiência e a qualidade do código final.

Outra vantagem relatada é a natureza multiplataforma da ferramenta, que permite o desenvolvimento para diversos dispositivos e sistemas. Além disso, a função livePreview proporciona alterações em tempo real, e a customização de componentes é simplificada, oferecendo flexibilidade ao desenvolvedor.

Grande parte dos participantes concordaram que esta abordagem reúne as vantagens do ViewCode, mas com benefícios adicionais. Não há necessidade de configurações prévias antes de iniciar, tornando o processo mais rápido e direto. Com menos código requerido para criar interfaces e uma curva de aprendizado mais suave que o ViewCode, os desenvolvedores conseguem alcançar resultados de alta qualidade de forma mais eficiente.

Finalmente, o recurso de hot-reload foi amplamente elogiado, já que permite que as alterações sejam refletidas em tempo real. A integração mais profunda com componentes nativos garante uma experiência de usuário mais fluida e consistente, destacando-se como uma vantagem fundamental relatada pelos participantes.

6.2.5.1.2 Desvantagens

Conforme apontado pelos participantes, uma das desvantagens sentidas é a percepção de que o código pode parecer mais desorganizado quando comparado ao UIKit. Essa sensação pode surgir da diferença de estruturação ou dos padrões adotados por essa nova abordagem.

Grande parte dos participantes relataram que, por ser uma tecnologia ainda recente, muitas integrações não são suportadas ou não funcionam tão eficientemente quanto esperado. Isso pode representar um obstáculo para projetos que dependem de ferramentas ou bibliotecas específicas.

A juventude deste framework é uma preocupação para alguns participantes. Como foi destacado, o SwiftUI ainda não oferece todas as funcionalidades e recursos que o UIKit, já consolidado no mercado, proporciona. Além disso, foi mencionado que

seu uso só começa a ser verdadeiramente vantajoso a partir do iOS 15, limitando seu uso em versões anteriores.

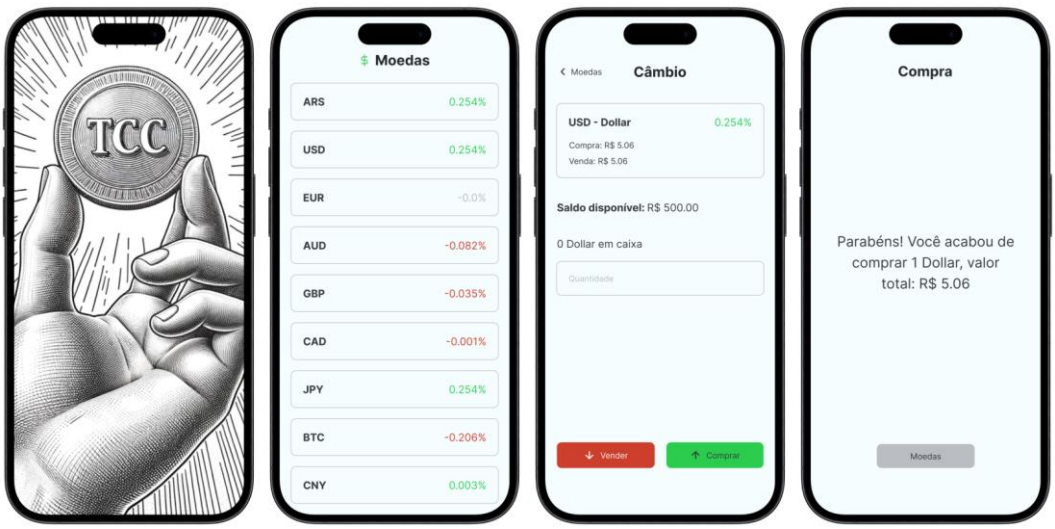
A natureza incipiente dessa tecnologia gera preocupações sobre sua compatibilidade, especialmente com versões mais antigas do iOS. Os participantes expressaram que, embora mostre potencial, pode haver desafios ao tentar implementá-la em ambientes mais antigos.

Um ponto comum entre os feedbacks é a ênfase no nível de maturidade da tecnologia. Sendo ainda muito nova, pode apresentar limitações e desafios que tecnologias mais estabelecidas já superaram. Essa juventude pode influenciar a decisão de adotá-la em projetos de maior escala ou em ambientes mais críticos.

6.2.6 Escolha de metodologia a partir de um Mockup de aplicação.

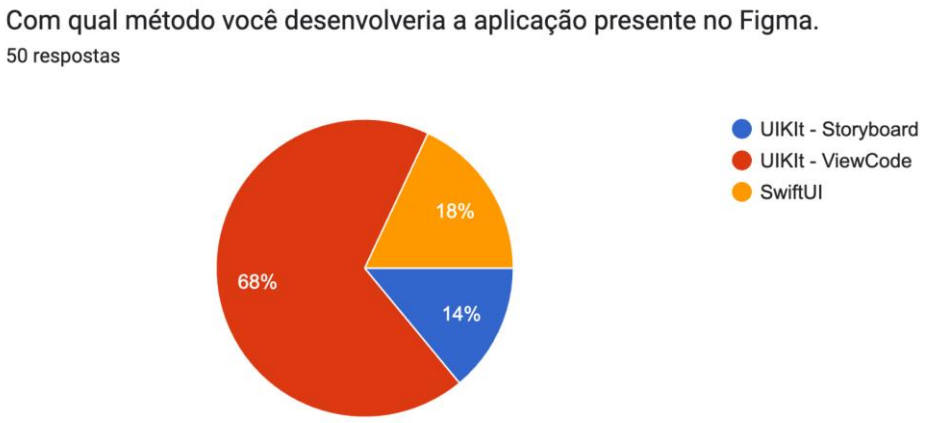
Diante de uma aplicação ilustrada no Figma, uma plataforma de destaque para design de interfaces, apresentamos na Figura 06 o exemplo dessa aplicação. Os desenvolvedores foram questionados sobre qual abordagem eles adotariam para concretizar esse design em um aplicativo iOS. As respostas, evidenciadas na Figura 07, proporcionaram uma visão reveladora sobre as tendências atuais e a flexibilidade dos profissionais perante as variadas técnicas de desenvolvimento

Figura 7 – Exemplo da aplicação



Fonte: Elaborado pelo autor (2023).

Figura 8 – Escolhas de métodos



Fonte: Elaborado pelo autor (2023).

A maior parte, 68% ou 34 indivíduos, optou pela combinação UIKit - ViewCode, sugerindo a popularidade e confiança nessa abordagem consolidada para trazer designs do Figma à vida. Em contrapartida, 18%, equivalente a 9 participantes, mostrou inclinação pelo SwiftUI, refletindo a crescente aceitação deste toolkit moderno e sua aplicabilidade em projetos recentes. Curiosamente, outro grupo, que constitui 14% ou 7 respondentes, também preferiu o método UIKit - Storyboard, reiterando a relevância contínua dessa técnica no desenvolvimento de interfaces.

6.2.6.1 Justificativa da escolha dos métodos

A decisão de optar por determinados métodos de programação não se dá de forma aleatória. Vários fatores influenciam essa escolha, e a análise das respostas dos entrevistados destaca algumas nuances interessantes. O nível de experiência do desenvolvedor, por exemplo, é crucial. Desenvolvedores mais novatos podem se inclinar por métodos que oferecem uma curva de aprendizado mais suave, enquanto veteranos podem buscar técnicas mais avançadas ou específicas.

Além disso, a exposição a projetos em mercados avançados que utilizam a linguagem de programação Swift também desempenha um papel fundamental. Trabalhar em ambientes que exigem soluções mais sofisticadas ou inovadoras pode

moldar a perspectiva do desenvolvedor e influenciar suas preferências. Dessa forma, compreender os motivadores por trás das escolhas metodológicas oferece insights valiosos não apenas sobre o perfil e a trajetória do profissional, mas também sobre as tendências e demandas do mercado de desenvolvimento atual.

6.2.6.1.1 ViewCode

Uma das notáveis vantagens do ViewCode é sua capacidade de integrar-se perfeitamente com elementos de Storyboard. Por exemplo, ele permite o uso da launchScreen, uma tela essencial para a introdução dos aplicativos. Essa flexibilidade proporciona aos desenvolvedores um leque de opções enquanto eles decidem a estrutura de suas interfaces. Além disso, ViewCode está se tornando uma tecnologia cada vez mais presente no mercado, o que a estabelece como uma oportunidade promissora para desenvolvedores que desejam estar em sintonia com as tendências atuais.

Considerando especificidades de design, como a tela de Câmbio, o ViewCode mostra-se uma escolha acertada. O posicionamento de elementos se torna mais simplificado e direto, e a capacidade de reutilizar botões e outros componentes é altamente benéfica. Essa reutilização não se limita apenas a botões; diversos componentes podem ser replicados com facilidade, otimizando o processo de desenvolvimento. O ViewCode também se destaca na construção de layouts, onde sua simplicidade e compatibilidade com todas as versões do iOS se tornam pontos fortes.

Outro ponto a ser considerado é a performance do ViewCode. Como uma tecnologia de mercado consolidada, ela é conhecida por sua eficiência e pela capacidade de reutilização de componentes. Isso não apenas acelera o tempo de desenvolvimento, mas também facilita a manutenção posterior do aplicativo, tornando as atualizações e correções mais diretas. Além disso, a capacidade de personalizar e misturar elementos é um atrativo adicional.

Por exemplo, os desenvolvedores podem optar por adicionar uma animação de carregamento específica para a primeira tela, enquanto utilizam o ViewCode para as subsequentes. Essa combinação oferece uma abordagem híbrida, permitindo uma atenção detalhada à estética e funcionalidade, enquanto ainda se foca na reutilização e definição clara dos elementos.

6.2.6.1.2 SwiftUI

Tenho uma excelente oportunidade de expandir meus conhecimentos na área de programação, pois ainda não trabalhei com SwiftUI. Estou considerando seriamente o desenvolvimento de um aplicativo utilizando essa tecnologia. Acredito que criar um app inovador, pensando nas tendências de mercado, seria um ótimo pretexto para me aventurar pelo SwiftUI. Esse método parece se adequar perfeitamente às minhas necessidades, principalmente por sua capacidade de criar interfaces simples e funcionais, o que reflete a praticidade e a eficiência da tecnologia.

Ao considerar as funcionalidades requeridas, percebo que muitas delas já são bem atendidas pelo Swift, como as condições ternárias. O SwiftUI, especificamente, disponibiliza uma gama de recursos que se encaixam perfeitamente às minhas necessidades: desabilitação de botões, criação de designs variados, além de textfields, imagens e systemimages. Tudo isso, a meu ver, pode ser implementado de forma simples e intuitiva com o SwiftUI.

No entanto, é importante ressaltar que minha escolha pelo SwiftUI pode estar sujeita a revisões, já que ainda não coloquei o projeto em prática e sabemos que muitos desafios e surpresas podem surgir no decorrer do desenvolvimento. Mesmo assim, o SwiftUI se apresenta como uma opção bastante promissora. Para este projeto específico, que é um aplicativo simples, com uma lista de funcionalidades não muito extensas, escolheria utilizar SwiftUI juntamente com a arquitetura MVVM. Isso porque a criação de listas é significativamente mais simples com SwiftUI, e essa arquitetura tende a ser mais eficiente para apps com um escopo mais restrito e sem muitas complexidades.

6.2.6.1.3 Storyboard

A minha escolha pelo uso do Storyboard em projetos de desenvolvimento de aplicativos é majoritariamente motivada pela sua simplicidade e pelo fato de ser a única metodologia com a qual estou atualmente familiarizado. A facilidade no uso do Storyboard reside na sua capacidade de permitir a construção rápida e prática de layouts. Isso se estende à criação de interfaces e transições de tela, que podem ser

executadas de forma muito ágil, uma característica particularmente útil em projetos com prazos apertados ou recursos limitados.

Além disso, a simplicidade no desenvolvimento de interfaces usando Storyboard é notável. Ele oferece um ambiente visual intuitivo, o que reduz a complexidade na criação de interfaces gráficas, tornando-se uma opção atraente especialmente para aqueles que estão começando na área de desenvolvimento iOS. Essa abordagem permite que um desenvolvedor com menos experiência consiga criar e manipular elementos de UI de maneira eficaz e sem um aprofundamento extensivo em programação de interfaces.

No entanto, apesar de todas as vantagens, sou consciente de que a utilização exclusiva do Storyboard não é considerada por muitos como a melhor prática ou a abordagem mais avançada em termos de arquitetura e manutenção de aplicativos. Essa consciência me leva a reconhecer a importância de expandir meus conhecimentos para incluir outras metodologias e práticas mais avançadas no desenvolvimento de software. Embora o Storyboard atenda às minhas necessidades atuais e me proporcione uma curva de aprendizado menos íngreme, estou ciente da necessidade de explorar e dominar outras técnicas e ferramentas para evoluir profissionalmente na área de desenvolvimento de aplicativos.

6.2.7 Desenvolvimento da Aplicação

Para garantir a transparência e permitir uma validação detalhada das informações e recursos discutidos neste artigo, convido todos a acessarem os repositórios complementares disponíveis no GitHub e no Figma.

No GitHub, você encontrará o código-fonte, documentações técnicas, e outros materiais que embasam as técnicas e as metodologias apresentadas. Este repositório é uma oportunidade para os interessados em tecnologia entenderem profundamente os processos e as ferramentas utilizadas.

O Figma, por sua vez, oferece uma visão interativa dos designs e protótipos que foram essenciais para o desenvolvimento do projeto. Esta plataforma é especialmente útil para designers, desenvolvedores de UI/UX e entusiastas que desejam explorar os aspectos visuais do nosso trabalho.

Acesse os links abaixo para explorar estes recursos:

GitHub: [https://github.com/JoaoPedroVolponi/TCC]

Figma: [Figma/joaopedrovolponi/tcc]

7. CONCLUSÃO

Após uma análise aprofundada da linguagem de programação Swift e suas abordagens para a criação de interfaces – Storyboard, ViewCode e SwiftUI – chegou-se a diversas considerações e insights. A linguagem Swift, desenvolvida pela Apple, apresenta robustez, versatilidade e uma gama de métodos para construir interfaces visuais para aplicações iOS.

Por meio de um questionário divulgado para desenvolvedores e estudantes familiarizados com Swift, foi possível reunir dados relevantes sobre as vantagens e desvantagens percebidas em cada abordagem.

Storyboard foi destacado por sua abordagem visual e interativa, permitindo que designers e desenvolvedores colaborem eficazmente. Contudo, enfrenta críticas quanto à sua escalabilidade e complexidade em projetos maiores, além de potenciais conflitos ao trabalhar com controle de versão.

ViewCode foi elogiado por oferecer um controle mais detalhado sobre a UI e a lógica da aplicação. Seu ponto forte é a flexibilidade, permitindo adaptações precisas na interface. Contudo, a ausência de um preview visual imediato pode ser considerada uma barreira para alguns desenvolvedores.

SwiftUI, a mais recente adição ao ecossistema, destaca-se por sua sintaxe declarativa e capacidade de criar interfaces de forma mais concisa. Ele integra-se perfeitamente com as funcionalidades mais recentes dos sistemas operacionais da Apple e é altamente adaptável a diferentes dispositivos. No entanto, seu uso ainda é limitado a versões mais recentes dos sistemas da Apple, e sua curva de aprendizado pode ser desafiadora para aqueles acostumados com abordagens mais imperativas.

Ao apresentar um protótipo de aplicação no FIGMA, foi observada uma tendência interessante na escolha dos desenvolvedores sobre qual método utilizar para implementá-lo ao.ViewCode

Isso reflete a percepção e o conforto atual da comunidade com essa abordagem, bem como os benefícios percebidos em relação às alternativas. Ao

desenvolver a aplicação com o método mais votado, foi possível consolidar as vantagens e desvantagens previamente mencionadas e observar a eficácia do método em um cenário real.

Em conclusão, não há uma abordagem "única" ou "melhor" para todos os cenários. Cada método de criação de interfaces em Swift tem suas próprias forças e fraquezas. A escolha depende das necessidades do projeto, da equipe e do nível de familiaridade com a abordagem. Contudo, é inegável que a linguagem Swift, em conjunto com suas ferramentas para design de interface, posiciona-se como uma solução poderosa para o desenvolvimento de aplicativos modernos e atraentes.

REFERÊNCIAS BIBLIOGRÁFICAS

ALURA. **IOS e Swift: Diferenças na construção de layouts com Storyboard, XIB e View Code**. Disponível em: https://www.alura.com.br/artigos/ios-swift-diferencas-construcao-layouts-storyboard-xib-view-code#:~:text=Como%20o%20nome%20j%C3%A1%20sugere,elementos%20para%20dentro%20da%20tela.. Acesso em: 14 out. 2023.

APPLE DEVELOPER [2023]. **Apple Developer**. Disponível em: https://developer.apple.com/. Acesso em: 12 out. 2023.

APPLE DEVELOPER [2023]. **Sobre o desenvolvimento de aplicativos com UIKit**. Disponível em: https://developer.apple.com/documentation/uikit/about_app_development_with_uikit. Acesso em: 14 out. 2023.

APPLE DEVELOPER [2023]. **SwiftUI**. Disponível em: https://developer.apple.com/documentation/swiftui/. Acesso em: 14 out. 2023.

APPLE DEVELOPER [2023]. **Xcode**. Disponível em: https://developer.apple.com/xcode/. Acesso em: 31 out. 2023.

APPLE DEVELOPER. **Documentation Archive (Storyboard)**. Disponível em: https://developer.apple.com/library/archive/documentation/General/Conceptual/Devpedia-CocoaApp/Storyboard.html. Acesso em: 14 out. 2023.

APPLE. **Newsroom**. Disponível em: apple.com. Acesso em: 12 out. 2023.

FIGMA. **Figma**. Disponível em: https://www.figma.com/. Acesso em: 31 out. 2023.

GITHUB. **GitHub**. Disponível em: https://github.com/. Acesso em: 1 nov. 2023.

GOOGLE FORMS. **Forms Google**. Disponível em: https://www.google.com/intl/pt-BR/forms/about/#overview. Acesso em: 19 out. 2023.

HACKING WITH SWIFT. **Hacking With Swift**. Disponível em: https://www.hackingwithswift.com/. Acesso em: 28 out. 2023.

HILLEGASS, C. K. A. A. **IOS Programming** : The Big Nerd Ranch Guide. 1. ed. [S.l.]: Big Nerd Ranch, 2016. p. 1-747.

INC, Appe. **Desenvolva em Swift Exporações**: Xcode 11. 1. ed. [S.l.]: Apple Inc, 2020. p. 1-502.

INC, Appe. **Desenvolva em Swift: Explorações** : Guia do professor Xcode 11. 1. ed. [S.l.]: Apple Inc, 2020. p. 1-709.

INC, Appe. **The Swift Programming Language**: Swift 5.7 Edition. 1. ed. [S.l.]: Apple Inc, 2023. p. 1-953.

INC, Apple. **Develop in Swift Fundamentals**: Xcode 11. 1. ed. [S.l.]: Apple Inc, 2020. p. 1-603.

LECHETA, R. R. **Desenvolvendo para iPhone e iPad**: Aprenda a desenvolver aplicações utilizando iOS SDK. 3. ed. São Paulo - BR: Novatec, 2014. p. 1-624.

MASKREY, M. **Beginning iPhone Developmente with Swift 3**: Exploring the iOS SDK. 3. ed. [S.l.]: Apress, 2016. p. 1-759.

MOELLER, G. **Opcionais em Swift**. Medium, 2022. Disponível em: https://medium.com/@giovannamoeller/opcionais-em-swift-81762bc319. Acesso em 12 nov. 2023.

MOURA, B. G. D. **Framework para desenvolvimento de aplicativos ios com interface de usuário programada de forma declarativa**. TCC, GOIÂNIA, v. 1, n. 1, p. 1-140, 2020. Disponível em: https://repositorio.pucgoias.edu.br/jspui/bitstream/123456789/509/1/TCC_BRENNO_GIOVANINI_DE_MOURA_v09dez2020.pdf. Acesso em: 10 out. 2023.

NAHAVANDIPOOR, V. **IOS 10 Swift Programming Cookbook**: Solutions e examples for iOS Apps. 1. ed. [S.l.]: O'Reilly, 2017. p. 1-427.

NAHAVANDIPOOR, V. **IOS 11 Swift Programming Cookbook**: Solutions e examples for iOS Apps. 1. ed. [S.l.]: O'Reilly, 2018. p. 1-609.

NOLAN, G. **Agile Swift**: Swift Programming Using Agile Tools and Techniques. 1. ed. [S.l.]: Apress, 2017. p. 1-1

SWIFT ORG. **Swift Org**. Disponível em: https://www.swift.org/. Acesso em: 12 out. 2023.