UNIVERSIDADE EVANGÉLICA DE GOIÁS - UNIEVANGÉLICA
ENGENHARIA DE COMPUTAÇÃO/ENGENHARIA DE SOFTWARE

ATIRSON FABIANO BARBOSA DE OLIVEIRA
IAGO GONÇALVES DE ASSIS

As vantagens e desvantagens do React Native comparado ao Kotlin

Anápolis
Dezembro, 2021

UNIVERSIDADE EVANGÉLICA DE GOIÁS - UNIEVANGÉLICA
ENGENHARIA DE COMPUTAÇÃO/ENGENHARIA DE SOFTWARE

ATIRSON FABIANO BARBOSA DE OLIVEIRA
IAGO GONÇALVES DE ASSIS

As vantagens e desvantagens do React Native comparado ao Kotlin

Trabalho apresentado ao Curso de Engenharia de Software da Universidade Evangélica de Goiás – UniEVANGÉLICA, da cidade de Anápolis-GO como requisito parcial para obtenção do Grau de Bacharel em Engenharia de Software.

Orientador (a): Prof. Me William Pereira Dos Santos Júnior

Anápolis
Dezembro, 2021

UNIVERSIDADE EVANGÉLICA DE GOIÁS - UNIEVANGÉLICA
ENGENHARIA DE COMPUTAÇÃO/ENGENHARIA DE SOFTWARE

ATIRSON FABIANO BARBOSA DE OLIVEIRA
IAGO GONÇALVES DE ASSIS

As vantagens e desvantagens do React Native comparado ao Kotlin

Monografia apresentada para Trabalho de Conclusão de Curso de Engenharia de Software da Universidade Evangélica de Goiás - UniEVANGÉLICA, da cidade de Anápolis-GO como requisito parcial para obtenção do grau de Engenheiro(a) de Software.

Aprovado por:

_____
Prof. Me. William Pereira Dos Santos Júnior

_____
Prof. Esp. Eduardo Ferreira de Souza

_____
Esp. Kleber Silvestre Diogo

Anápolis, 03 de dezembro de 2021.

FICHA CATALOGRÁFICA

ASSIS, Iago. **As vantagens e desvantagens do React Native comparado ao Kotlin. Anápolis,** 2021. (Universidade Evangélica de Goiás – UniEVANGÉLICA, Engenheiro(a) de Software, 2021).
Monografia. Universidade Evangélica de Goiás, Curso de Engenharia de Software, da cidade de Anápolis-GO.

1. Frameworks, Multiplataforma, Desenvolvimento, React Native e Kotlin.

OLIVERA, Atirson. **As vantagens e desvantagens do React Native comparado ao Kotlin. Anápolis,** 2021. (Universidade Evangélica de Goiás – UniEVANGÉLICA, Engenheiro(a) de Software, 2021). Monografia. Universidade Evangélica de Goiás, Curso de Engenharia de Software, da cidade de Anápolis-GO.
1. Frameworks, Multiplataforma, Desenvolvimento, React Native e Kotlin.

REFERÊNCIA BIBLIOGRÁFICA

ASSIS, Iago, OLIVEIRA, Atirson. **As vantagens e desvantagens do React Native comparado ao Kotlin. Anápolis**. Anápolis, 2021. 48 p. Monografia - Curso de Engenharia de Software Universidade Evangélica de Goiás - UniEVANGÉLICA.

CESSÃO DE DIREITOS

NOMES DOS AUTORES: Atirson Fabiano Barbosa de Oliveira e Iago Gonçalves de Assis
TÍTULO DO TRABALHO: As vantagens e desvantagens do React Native comparado ao Kotlin
GRAU/ANO: Graduação / 2021

_____

Iago Gonçalves de Assis

_____

Atirson Fabiano Barbosa de Oliveira

Anápolis, 03 de dezembro de 2021

AGRADECIMENTOS

Agradecemos ao Professor William, por ter sido nosso orientador e ter desempenhado tal função com dedicação e amizade dentro e fora dos ambientes acadêmicos.

Agradecemos também pelos professores que fizeram parte da nossa formação e que sempre se empenharam para atingir nossos melhores resultados acadêmicos, em especial a Professora Luciana Nishi que em todo momento esteve presente para nos ensinar e tornar esse momento possível.

Por fim, a Deus pela capacitação, força e saúde por todo momento percorrido na nossa formação.

RESUMO

Com o aumento da demanda de aplicações para dispositivos móveis, que foi impulsionada no ano de 2020 mediante a um cenário de pandemia no qual os usuários necessitam de software que ajudem a enfrentar tal adversidade, se fez necessário o uso de frameworks que atendam às necessidades dos desenvolvedores para que realizem um bom trabalho. O objetivo deste estudo é demonstrar as vantagens e desvantagens do desenvolvimento multiplataforma usando React Native quando comparado com o Java/Kotlin para plataforma nativa Android. E assim evidenciar os custos, mão-de-obra, oportunidades de trabalho, qualidade das ferramentas para desenvolvimento, tempo de esforço empregado na construção de funcionalidades deixando tais informações de forma clara e objetiva. Portanto dando a possibilidade de auxiliar empresas e profissionais que em algum momento irão precisar fazer uma análise de mercado e selecionar um framework que o atenda às suas necessidades e usando critérios objetivos e com base em dados para usar ou descartar tais ferramentas.

Palavras-chave: Frameworks, Multiplataforma, Desenvolvimento, React Native e Kotlin.

LISTA DE TABELAS

LISTA DE GRÁFICOS

LISTA DE ABREVIATURAS E SIGLAS

Siglas	Descrição
APIS	Application Programming Interface
IDE	Integrated Development Environment
GQM	Goal/Question/Metric
PCU	Pontos de Caso de Uso
CPU	Central Processing Unit
iOS	iPhone Operating System
CEO	Chief Executive Officer
JS	Javascript
NPM	Node Package Manager
GPS	Global Position System
PCU	Pontos de Casos de Uso
PCUNA	Pontos de Casos de Uso Não Ajustados

LISTA DE IMAGENS

SUMÁRIO

INTRODUÇÃO

O profissional desenvolvedor de software moderno vive em uma constante atualização de seus conceitos, técnicas e objetivos profissionais visando é claro, a manutenção do seu alto nível e se atentar as tendências de mercado para buscar sempre as melhores opções. Buscar uma especialização intensa em uma determinada tecnologia para atender suas necessidades e expectativas de mercado parece um roteiro de praxe para os programadores, mas acaba consumindo um tempo considerável, e existindo é claro a situação em que esse tempo não foi bem aproveitado.

O desenvolvimento de software é uma proposta arriscada e cara. Conversar com desenvolvedores de software ilustra rapidamente a situação complicada que o desenvolvimento pode se tornar. Entregas atrasadas, com possíveis defeitos e alterações no produto final, orçamentos, custos e funcionários e partes interessadas frustradas são alguns dos detritos resultantes de uma falha ou menos do que bem-sucedida situação do planejamento de um projeto (ADOLPH; KRUCHTEN; HALL, 2012)

Observando este cenário, enfrentar essa situação de investimento no Brasil, acaba se tornando financeiramente complicada, para uma clareza de valores, de acordo com Bartels (2006, p 1269, apud, ADOLPH; KRUCHTEN; HALL, 2012), “O software à nível global em seu desenvolvimento é uma indústria de 1,6 trilhão de dólares”. Ou seja, é uma parte da indústria que movimento um capital considerável no mercado e que vem ganhando mais espaço a cada dia. Contudo, é válido ressaltar a necessidade de minimizar os custos do desenvolvimento, pois, segundo Singh, Singh e Mishra (2018) “Avaliar o custo estimado do projeto, duração e custo de manutenção durante o desenvolvimento do software é uma meta valiosa que precisa ser alcançada para reduzir o custo amplo do desenvolvimento.”

Levando em conta esse custo do desenvolvimento algumas empresas adotam o uso de framework híbrido como por exemplo o React Native, e segundo Sabino e Leão (2016) “aplicações híbridas são aplicações multiplataforma. Elas permitem, por exemplo, que você construa um produto iOS e Android simultaneamente, em vez de desenvolver duas vezes usando a linguagem nativa de cada plataforma”. O que pode ajudar a manter o orçamento

dentro do estabelecido já que será necessário apenas uma ferramenta. Porém existe a possibilidade de fazer o desenvolvimento nativo usando Java/Kotlin para Android ou Objective-C/Swift para iOS.

Além do conflito financeiro mencionado, por se tratar de planejamento de software que envolve pessoal e trabalho em equipe intenso, Boehm (1984) afirma: “Atributos de pessoal e atividades de fornecimento de relações humanas são de longe a maior fonte de oportunidade para melhorar a produtividade do software”.

Portanto, como foi citado existe alguns fatores que devem ser levados em consideração no planejamento do desenvolvimento de software e analisando essa situação quais são as vantagens e desvantagens do framework React Native para a construção de software multiplataforma quando comparado a plataforma Nativa Android usando Java/Kotlin?

OBJETIVOS DA PESQUISA

OBJETIVO GERAL

Analisar as vantagens e desvantagens do framework React Native para desenvolvimento multiplataforma buscando auxiliar no planejamento dos projetos de software e discernir entre suas características os motivos para sua implementação em massa como opção forte e consistente de desenvolvimento.

OBJETIVOS ESPECÍFICOS

- Analisar os custos para softwares multiplataforma;
- Identificar as limitações dos frameworks de desenvolvimento multiplataforma;
- Mensurar as ofertas de profissionais para as tecnologias híbridas;
- Investigar a manutenibilidade e suporte dos frameworks híbrido e nativo;
- Quantificar a produtividade do desenvolvimento híbrido e nativo;
- Identificar a oferta de API (Application Programming Interface) para as ferramentas de desenvolvimento híbrido;

JUSTIFICATIVA

A demanda no desenvolvimento de aplicativos vem crescendo consideravelmente e com a Covid-19 e isolamento social impulsionou criando um aumento significativamente nouso de dispositivos moveis.

> As instalações de aplicativos pagos e orgânicos definitivamente aumentaram durante a pandemia; consumidores em todo o mundo instalaram cerca de 4 bilhões de aplicativos no primeiro semestre de 2020. Sem dúvida, eles estavam ocupando todo o tempo livre que ganhavam por não se deslocar para o trabalho ou escola ouparticipar de atividades pessoais. As instalações orgânicas e pagas aumentaram 21% e 15%, respectivamente, provando que quando as pessoas têm mais tempo livre, a melhor maneira de preenche-lo são os aplicativos em seus dispositivos (ABEINFO, 2021).

Um dos frameworks híbridos utilizado por grandes empresas é o React Native umabiblioteca desenvolvida com JavaScript criado em 2015 e usado atualmente no Instagram, Uber Eats, Discord, Tesla e etc. (Facebook, 2021). E como essa ferramenta é bastante difundida e com uma linguagem de programação entre as mais populares do mundo segundo Stack Overflow (2020) o JavaScript está em primeiro lugar em popularidade. E devido a esses fatores será usado como objeto de estudo o React Native.

O desenvolvimento Nativo foi feito na plataforma Android um sistema operacional construído pelo Google em 2008 isso porque segundo a Statista (2021) os smartphones com o sistema operacional Android detêm 87% do mercado global em 2019 e espera-se que aumente nos próximos anos. O sistema operacional móvel desenvolvido pela Apple (iOS) tem uma participação de mercado de 13%. O que permite uma boa base de comparação com esse volume de usuários do Android e por isso será usado como objeto de estudo.

Com tantas opções de ferramentas e particularidades entre as plataformas como escolher a melhor para o seu projeto e contexto seja empresa ou profissional diante de tantas incógnitas este estudo visa colaborar evidenciando vantagens e desvantagens entre as ferramentas mais usadas no mercado para mostrar de maneira clara e objetiva dados relevantes para auxiliar na escolha e planejamento.

1. FUNDAMENTAÇÃO TEÓRICA

O desenvolvimento para plataformas híbridas consiste em fazer um código para funcionar nos principais sistemas operacionais. Já o nativo implica em construir aplicações apenas para um tipo de sistema.

Será feita a introdução dos itens que irão compor este trabalho no qual inclui plataformas híbridas e nativas, Java/Kotlin, React Native e os métodos utilizados para quantificar os esforços empregados em desenvolvimento de funcionalidades.

> Abordagens de desenvolvimento de plataforma híbrida oferecem aos desenvolvedores de aplicativos móveis a oportunidade de implementar aplicativo em várias plataformas móveis a partir de uma única base de código. Eles permitem escrever código uma vez e executá-lo em qualquer lugar. Sem essas abordagens, os desenvolvedores têm que desenvolver e manter bases de código separadas para cada plataforma que eles querem oferecer seus produtos. Essa abordagem é conhecida como desenvolvimento nativo. Atualmente, o mercado de telefonia móvel é dominado por duas plataformas Google Android e Apple iOS (DORFER; DEMETZ; HUBER, 2020).

O desenvolvimento nativo em Android usa Java e/ou Kotlin são duas linguagens de programação a Oracle sendo responsável pelo Java e a JetBrains pelo Kotlin e podem ser usadas simultaneamente no desenvolvimento de aplicações para Android usando a IDE Android Studio.

> O Android Studio oferece compatibilidade total com o Kotlin. Assim, você pode adicionar arquivos Kotlin ao seu projeto e converter o código da linguagem Java em Kotlin. Em seguida, você poderá usar todas as ferramentas atuais do Android Studio com o código Kotlin, como preenchimento automático, verificador de Lint, refatoração, depuração, entre outras (ANDROID, 2020).

React Native é uma biblioteca JavaScript criada pelo Facebook em 2015. Tendo contribuições de indivíduos e empresas em todo mundo, incluindo Callstack, Expo, Infinite Red, Microsoft e Software Mansion (Facebook, 2021).

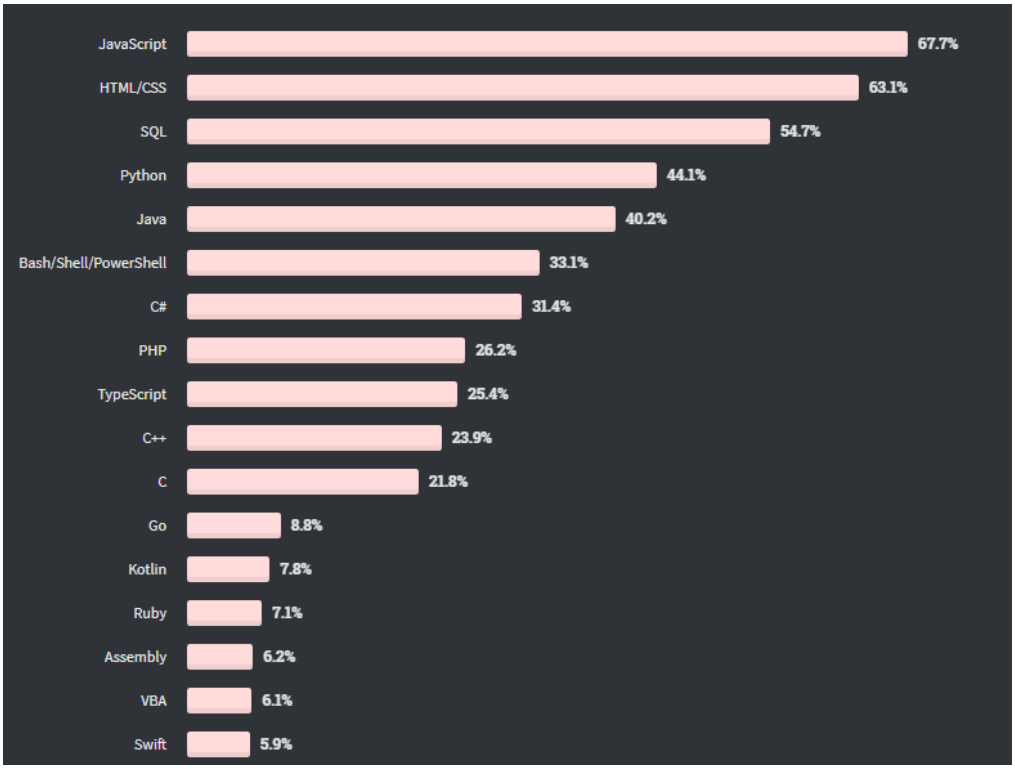
O método para quantificar esforço do desenvolvimento Goal/Question/Metric (Objetivo, Questão, Métrica) conhecido como GQM é utilizado para extrair dados relevantes durante o processo de criação de aplicações. O Dal'Osto (2003) afirma que existe três níveis no GQM. Primeiro Nível Conceitual, uma meta é definida para um objeto de acordo com uma variedade de razões. Segundo Nível Operacional, são utilizadas questões para definir a melhoria e garantir a meta solucionada. Terceiro Nível Quantitativo, é associado um conjunto de dados às questões, afim de respondê-las de forma quantitativa.

Para complementar a estimativa de esforço existe uma formula baseada em pontos de caso de uso para quantificar o tempo gasto no desenvolvimento.

> A técnica de estimativa por Pontos de Caso de Uso foi proposta em 1993 por Gustav Karner, da Objectory (hoje, Rational Software). Ela baseia-se em dois métodos bastante utilizados - o mecanismo de Pontos de Função e uma metodologia conhecida como Mk II, uma adaptação da técnica de PFs, bastante utilizada na Inglaterra. A forma de lançar uma estimativa é o principal diferencial da métrica por Casos de Uso: o método trata de estimar o tamanho de um sistema de acordo com o modo como os usuários o utilizarão, a complexidade de ações requerida por cada tipo de usuário e uma análise em alto nível dos passos necessários para a realização de cada tarefa, em um nível muito mais abstrato que a técnica de Pontos de Função. (FREIRE, 2003).

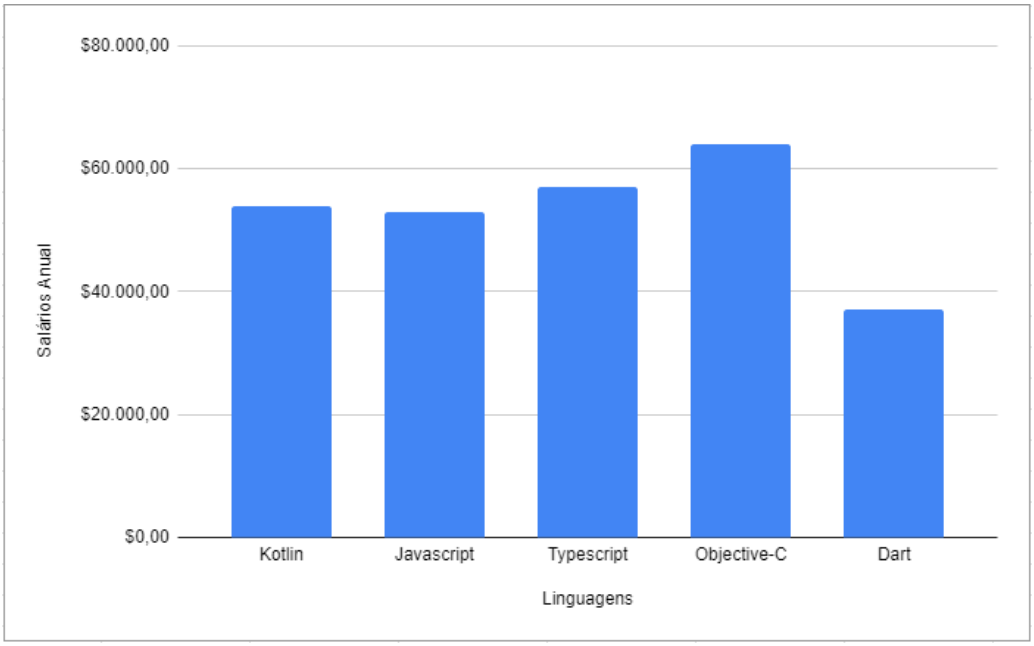
Um dos itens que é importante para analisar as vantagens e desvantagens de uma ferramenta é o custo do desenvolvimento e este valor é bastante variável e existe alguns fatores que influenciam diretamente como oferta de vagas, mão-de-obra, ferramentas e salários. Sobre oferta de vagas houve um aumento na demanda por profissionais na área de tecnologia da informação devido a Covid-19 o qual pode gerar um déficit de profissionais de até 260 mil vagas até 2024 segundo Sena e Granato (2021). E entre as habilidades mais desejadas pelas empresas estão Git, JavaScript e ReactJS (SENA, 2021). Segundo o Stack Overflow (2020) um dos maiores sites de perguntas e respostas sobre programação do mundo colocam o JavaScript como tecnologia mais popular em 2020 e também traz o salário como podemos ver no GRAF. 1 e GRAF. 2.

GRÁFICO 1 – Linguagens de programação, script e marcação



Fonte: Stack Overflow (2020)

GRÁFICO 2 – Linguagens e salários anuais



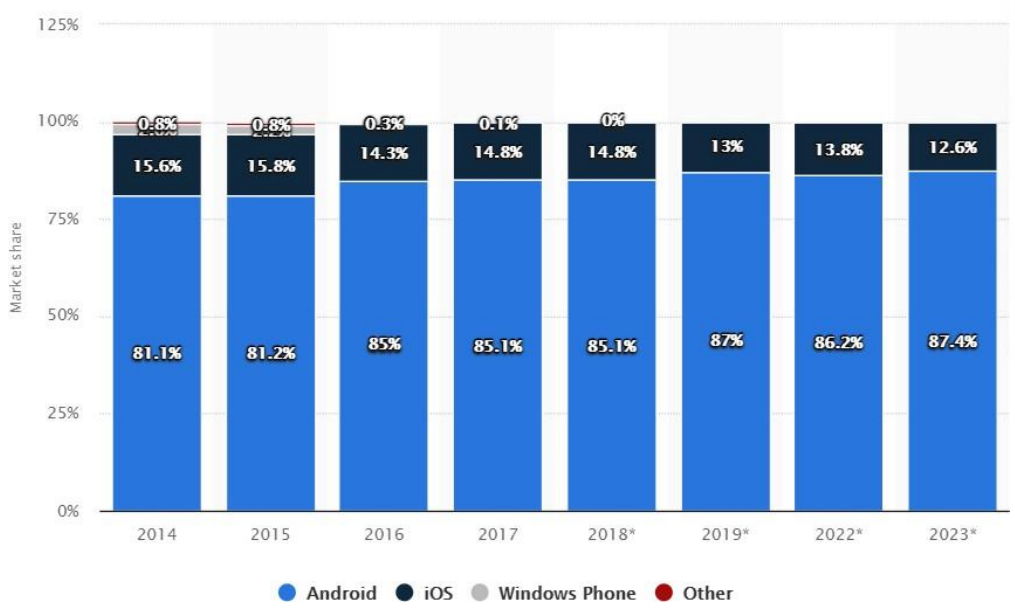
Fonte: Adaptado do Stack Overflow (2020)

E o número de desenvolvedores no mundo chega a 23,9 milhões e aqueles que atuam com JavaScript chega a ser mais da metade o que coloca o em destaque a facilidade para encontrar mão-de-obra para atuar com essa tecnologia. No terceiro trimestre de 2020, de acordo com a Developer Economics, o número de desenvolvedores de software usando JavaScript foi de 12,4 milhões. Isso significa que 53% de todos os desenvolvedores do mundo usaram JavaScript em algum momento (DAXX, 2020).

Outro fator relevante é número de usuários das principais plataformas mobile no mundo tendo hoje um cenário totalmente dominado em quantidade pelo Android que chega a ser 85% entre todos os dispositivos mobile de acordo com o GRAF. 3. O que favorece o uso de Java/Kotlin para atingir o maior número de usuários significativamente.

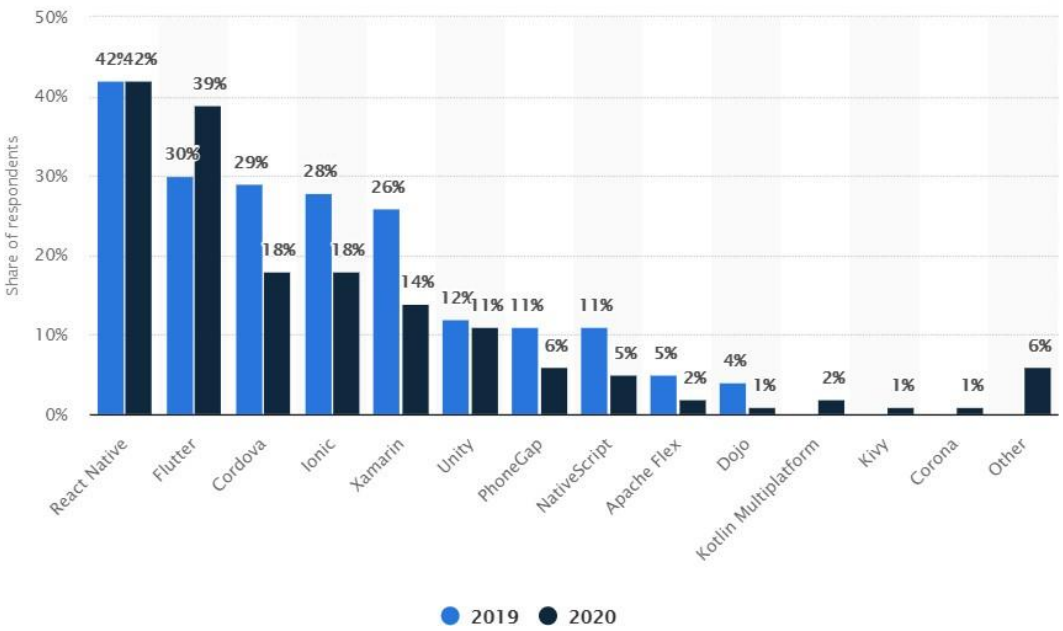
No cenário multiplataforma o React Native vem crescendo consideravelmente nos últimos anos e reforça o cenário das ferramentas que utilizam JavaScript para o desenvolvimento. Como podemos ver no GRAF. 4.

GRÁFICO 3 – Parcela das remessas globais de smartphones por sistema operacional de 2014a 2023.



Fonte: Statista (2021)

GRÁFICO 4 – Frameworks multiplataforma usado por desenvolvedores de software nomundo em 2019 e 2020.



Fonte: Statista (2020)

De acordo com os dados levantados conseguimos mensurar características e analisar quais são as vantagens e desvantagens de React Native e Java/Kotlin. Podemos demonstrar que com as informações obtidas, obtivemos avanços consideráveis em toda a popularidade do Javascript que contribui diretamente para o desenvolvimento de bibliotecas e ferramentas voltadas para essa linguagem, que por sua vez, tem sua base principal o React Native, consequentemente qualificado com salários razoáveis, referenciando o fato de que o desenvolvimento mobile tem maiores chances de terem o custo mais baixo quando comparado ao nativo, porém com sua performance de 6% a 8% menor comparado a aplicativos construídos com Java/Kotlin e Objective-C/Swift. Nesse contexto, sem referenciar que para acessar recursos nativos dos dispositivos o React Native se mantém na dependência de bibliotecas e plugins desenvolvidos por outras empresas e desenvolvedores diferente do Java/Kotlin, que tem funcionalidades desenvolvidas pela própria Google para acessar recursos nativos e que podem ser acessados diretamente na sua documentação oficial.

Já o desenvolvimento nativo usando Java/Kotlin implica em uma performance melhor na maioria das situações, principalmente quando se trata de usar recursos como GPS, câmera, leitor biométrico e entre outros, segundo DORFER; DEMETZ; HUBER, 2020. Valorizado

com bons salários e beneficiado pelo sistema Android ser o mais utilizado no mundo até o momento. Possui uma documentação vasta com muitos exemplos e plugins mantidos pela empresa responsável pelo sistema operacional, neste caso o Google, que traz toda a sua credibilidade pela integridade de sua empresa, porém com uma dependência, já que temos uma escassez de bibliotecas externas fora do ecossistema Android. Observando dessa forma, nos exemplos desenvolvidos citados, pode se notar uma verbosidade muito grande quando comparado ao React Native.

Assim podemos concluir que para escolher entre React Native e Android é necessário analisar o cenário que cerca o projeto como um todo e verificar a viabilidade de cada um para diminuir a possibilidade de fracasso nas dependências alocadas para projeto em questão

2. METODOLOGIA DA PESQUISA

O estudo em questão refere-se a uma pesquisa quantitativa e qualitativa com as informações mais relevantes possíveis para colaborar com as empresas e profissionais que vão escolher com qual tecnologia querem trabalhar. E para evidenciar os aspectos segue a descrição das etapas a serem desenvolvidas.

Para analisar a oferta de profissionais e os custos da construção do software seja multiplataforma ou nativa será analisado dados estatísticos sobre oferta de mão-de-obra e vagas de emprego e a média salarial dos profissionais de acordo com a senioridade para as tecnologias React Native e Java/Kotlin usando os dados disponíveis em algumas plataformas como Stack Overflow, Glassdoor, Linkedin, Statista, repositórios do Github colocando de forma gráfica para fácil interpretação.

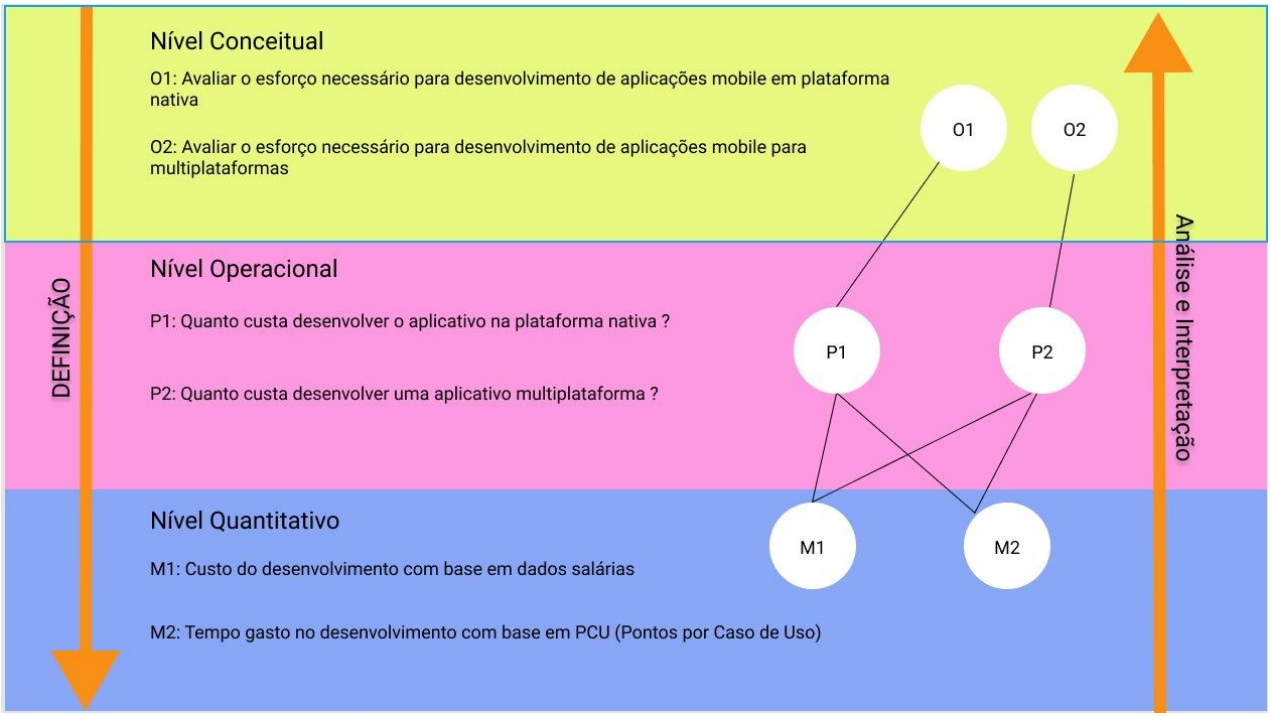
Já as limitações de cada ferramenta será feita uma análise descritiva sobre os recursos disponíveis para acessar as opções que hardware oferta para dispositivos mobile como câmera, sensores, leitor de digital entre outros verificando desempenho e características, usando como fonte de pesquisa as documentações oficiais dos frameworks e APIs disponíveis e a manutenibilidade dos mesmos para auxiliar nessas funcionalidades e os dados levantados serão estruturados em uma tabela seguindo com linhas listando os recursos dos dispositivos mobile e contendo as colunas: Desempenho/Características React Native e Desempenho/Características Java/Kotlin.

O desenvolvimento de funcionalidades para exemplificar pontos relevantes das ferramentas se faz necessário e para fazer a quantificação de esforço será usada o método GQM e estimativa por pontos de caso de uso conhecido como PCU, demonstrados no Apêndice A e B. E assim medir e evidenciar o esforço para o desenvolvimento da mesma função em diferentes plataformas. Nesse caso será desenvolvido um login com acesso a biometria com impressão digital assim podendo transmitir uma visão clara de acesso a um recurso que vem se tornando bastante popular. E assim extrair dados de tempo necessário para construção, documentação e suporte.

2. DESENVOLVIMENTO

Foi usado o método GQM para definição de quais são os objetivos e métricas de como atingir os objetivos definidos usando três camadas: Nível Conceitual, Nível Operacional e Nível Quantitativo, no qual podemos ver na imagem abaixo.

Figura 1 - Definição de Camadas



Fonte os Autores

Para construir as funcionalidades em ambas tecnologias React Native e Java/Kotlin, foi estimado quanto tempo levaria para o desenvolvimento usando PCU. Baseado no modelo PCU, foi criado um documento que contempla as características relevantes para estimativas, que é usado tanto para estimar o desenvolvimento em React Native quanto Java/Kotlin que podemos visualizar no APÊNDICE B.

No desenvolvimento prático, foram criadas duas aplicações para exemplificar o uso dos frameworks React Native e Java/Kotlin, e testadas no mesmo ambiente em um Android Pixel 4 com a versão 11 do seu sistema operacional.

Exemplificando a aplicação do React Native, é necessário a instalação de pacotes externos, neste caso, usamos o *react-native-touch-id*. Na primeira etapa usamos um *Hook* (uma função do ciclo de vida) do React Native chamado *useEffect* que ao renderizar a tela, ele executa e verifica se o dispositivo tem a opção de usar *fingerprint* (impressão digital) como meio de autenticação.

Figura 2 - useEffect

```
useEffect(() => {
  TouchID.isSupported().then(success => {
    setSupported(true)
  }).catch(error => {
    console.log('Error Touch: ' + error)
    alert('Touch ID não suportado/habilitado')
  })
}, [])
```

Fonte: Os autores

Após essa verificação temos a função a ser chamada pelo usuário quando clicar no botão de *login*. Primeiramente é declarado um objeto com as configurações como título do modal, cor quando houver erro e a respectiva mensagem. Na sequência, vem a chamada da função da biblioteca que retorna sucesso ou erro.

Figura 3 - Login

```
const handleLogin = () => {
  const configs = {
    title: 'Autenticação',
    color: '#FF0000',
    sensorErrorDescription: 'Touch ID Inválido'
  }
  TouchID.authenticate("Login", configs).then(success => {
    console.log('Seja bem-vindo')
    setName('Atirson Fabiano')
  }).catch(error => {
    console.log('Falha na Autenticação')
  })
}
```

Fonte: Os autores

E por fim temos a chamada da função que foi atrelada ao botão entrar através do evento *onPress*.

Figura 4 - onPress

```
<TouchableHighlight style={styles.btn} onPress={ handleLogin }>
  <Text style={{ color: '#FFF', fontWeight: 'bold' }}>Entrar</Text>
</TouchableHighlight>
```

Fonte: Os autores

Já no Java/Kotlin precisamos adicionar as dependências a biblioteca *Biometric*. Depois é declarado as variáveis para execução, função de biometria e de customização do modal de mensagem.

Figura 5 - Biometric

```
lateinit var executor: Executor
lateinit var biometricPrompt: androidx.biometric.BiometricPrompt
lateinit var promptInfo: androidx.biometric.BiometricPrompt.PromptInfo
```

Fonte: Os autores

Após isso temos a chamada da função que pode retornar erro, sucesso ou falha. Para executar a função passamos dois parâmetros, a variável declarada com o tipo Executor e a função de *Callback* de autenticação.

Figura 6 - Callback

```
biometricPrompt=androidx.biometric.BiometricPrompt( activity: this@MainActivity,executor,object:
    androidx.biometric.BiometricPrompt.AuthenticationCallback() {
    override fun onAuthenticationError(errorCode: Int, errString: CharSequence) {
        super.onAuthenticationError(errorCode, errString)
        authStatusTv.text = "Error $errString"
    }

    override fun onAuthenticationSucceeded(result: BiometricPrompt.AuthenticationResult) {
        super.onAuthenticationSucceeded(result)

        authStatusTv.text = "Successfully Auth"
    }

    override fun onAuthenticationFailed() {
        super.onAuthenticationFailed()

        authStatusTv.text = "Falied Auth"
    }
})
```

Fonte: Os autores

E por fim temos a customização das mensagens no modal e respostas mediante a erros e a declaração da função que irá ficar monitorando o evento de *click* para executar.

Figura 7 – Monitoramento do evento de *click*

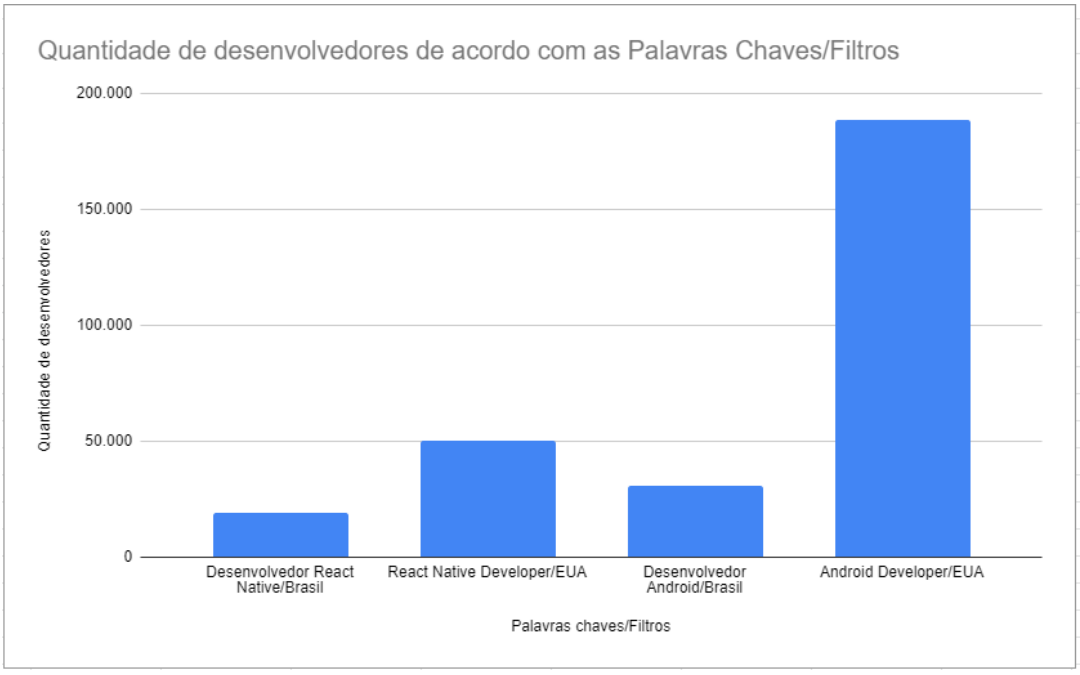
```
promptInfo=androidx.biometric.BiometricPrompt.PromptInfo.Builder()
    .setTitle("Biometric Authentication")
    .setSubtitle("Login using fingerprint or face")
    .setNegativeButtonText("Cancel")
    .build()

authBtn.setOnClickListener { it: View!
    biometricPrompt.authenticate(promptInfo)
}
```

Fonte: Os autores

3. RESULTADOS

Uma pesquisa realizada para consultar a quantidade de desenvolvedores que estão disponíveis para cada tecnologia, sendo que a disponibilidade deles impacta diretamente no custo, construção e manutenibilidade do software foi feita usando o Linkedin, uma rede social de negócios fundada em 2002 que conta com mais de 575 milhões de usuários (OSMAN, 2021). Como podemos ver na Gráfico 5, foram utilizadas as palavras chaves (Desenvolvedor React Native, Desenvolvedor Android) com filtro de localidade no Brasil e (React Native Developer, Android Developer) com filtro de localidade nos Estados Unidos: Pesquisa realizada em novembro de 2021.

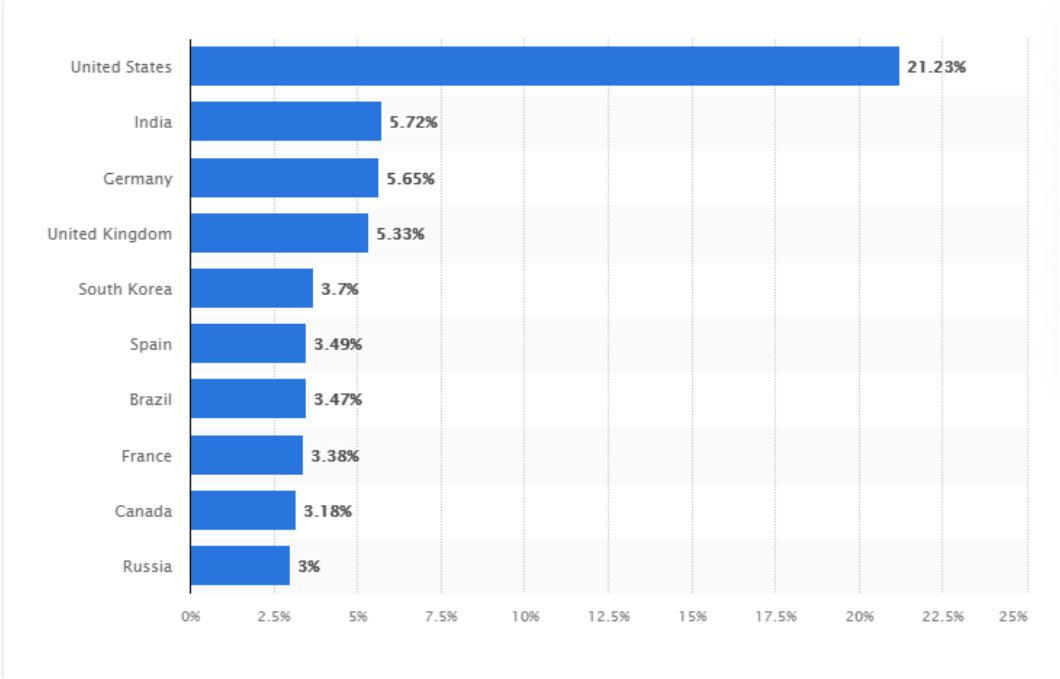
Gráfico 5 – Quantitativo de desenvolvedores



Fonte os Autores

Podemos notar que a quantidade de desenvolvedores Android nos Estados Unidos é bem superior, o que reflete onde se deve procurar quando sua equipe de desenvolvimento for montada, e o que complementa essa visão é o Gráfico 6 a seguir que mostra a distribuição de desenvolvedores Android pelos países mais relevantes no âmbito do desenvolvimento de software, segundo a pesquisa do STATISTA, apenas 3,47% de todos os desenvolvedores Android atuam no Brasil.

Gráfico 6 - Distribution of Android app developers worldwide as of 1st quarter 2018, by country



Fonte adaptado do STATISTA, 2021

Outro ponto crucial para avaliar o desenvolvimento de um software é a manutenibilidade do mesmo, no caso o Android possui uma documentação própria focada em Java/Kotlin e possui um histórico de versões que pode ser visto no Apêndice C. O histórico se encontra na documentação oficial da ferramenta em Android Versions ou em sites como Javapoint Android Versions. Já o React Native possui mais de 800 versões sendo mais de 50 estáveis e todas possuem sua documentação no site oficial da ferramenta em React Native Version, e também podem ser encontradas no gerenciador de pacotes NPM ao pesquisar por React Native e verificar as especificações em Versions.

Para demonstrar os principais fatores que colaboram para determinar o custo do desenvolvimento de aplicativos móveis multiplataforma, segundo Denysov (2020), listamos abaixo juntamente com uma tabela, os valores para construção de aplicativos dentro de um conceito base de MVP (Mínimo do Produto Viável) baseado nos estudos do CEO Vitaly Makhov da Doit Software.

- Desenvolvimento sincronizado: Ao desenvolver aplicações híbridas durante este processo, terá o benefício de ter em uma base de código para duas plataformas funcionando, quase que totalmente finalizado, faltando apenas alguns ajustes e adaptações especificas para plataforma Android e iOS.

- Código reutilizável: Além de ter um desenvolvimento sincronizado, também terá uma base de código que pode ser reutilizada para ambas as plataformas, cortando os gastos quase que pela metade quando comparado ao desenvolvimento nativo.

- Frameworks/Biblioteca existentes: As bibliotecas e frameworks desenvolvidas para estas ferramentas tem a tendência de trazer funcionalidades para ambas as plataformas, ou seja, com uma biblioteca/framework você terá funcionalidade para Android e iOS.

- Equipe reduzida: Para o desenvolvimento multiplataforma você terá provavelmente uma equipe menor do que se tivesse que ter equipe para Android e uma para iOS o que reduz os custos do desenvolvimento drasticamente.

Tabela 1 - Relação de preço por APP (híbrido)

Tipo de App	Preço
Aplicativo básico: aplicativos como calculadoras, relógios e pequenos jogos são os aplicativos mais baratos para desenvolver. Estas são soluções que não requerem uma conexão de rede ou desenvolvimento de back-end e levam cerca de um mês para serem concluídas	**$10,000 – 15,000**
Os aplicativos baseados em dados são, por exemplo, calendários, mapas, soluções meteorológicas, etc., que recebem muitas informações, analisam e compartilham com os usuários. Esses tipos de aplicativos têm perdido seu valor nos últimos anos e são lançados principalmente como produtos.	**$15,000 – $20,000**
Aplicativos com autenticação de usuário e dados personalizados: um exemplo de aplicativo de autenticação é um aplicativo de fidelidade em que os usuários precisam fazer login para obter acesso aos recursos. O aplicativo armazena dados do usuário e compartilha informações entre dispositivos. Este é um projeto mais sofisticado e, portanto, caro, que exigirá gerenciamento de usuários.	**$40,000 – $80,000**
Os aplicativos de redes sociais são autoexplicativos - Facebook, Instagram, LinkedIn, etc. Eles precisam lidar com milhões de interações, chats e permitir que os usuários compartilhem informações. Tudo isso requer uma infraestrutura de back-end sólida e um investimento maior.	**$60,000 – $300,000**
Os aplicativos de comércio eletrônico: incluem Amazon, Alibaba e outros mercados. Eles exigem uma lista extensa de recursos, como registro de usuário, interações sociais, catálogos, páginas e descrições de produtos, opções de pagamento, etc. A estimativa de custo que fornecemos é para um pequeno aplicativo de comércio eletrônico.	**$60,000 – $300,000**
Aplicativos sob demanda como Uber ou Grubhub conectam provedores de serviços com usuários finais e satisfazem as necessidades imediatas das pessoas. Esses tipos de aplicativos combinam os recursos de soluções de redes sociais e aplicativos de comércio eletrônico e exigem um financiamento significativo.	**$80,000 – $150,000**
Aplicativo de marketplace: aplicativos como o TripAdvisor são chamados de plataformas de mercado que incorporam as funcionalidades de eCommerce e aplicativos sob demanda. Ao contrário dos aplicativos de comércio eletrônico, as soluções de mercado oferecem uma variedade de serviços diferentes e fornecem acesso imediato ao provedor de serviços de escolha do usuário.	**$150,000 – $300,000**

Fonte: Adaptado de Makhov (2021)

Neste contexto para atender as duas plataformas de forma nativa estes custos podem sofrer grandes alterações. Porém o desenvolvimento multiplataforma implica em alguns aspectos negativos que devem ser avaliados se fazem diferença no contexto ao qual a empresa e projeto estão inseridos o uso de React Native tem um uso de CPU e memória substancialmente maior quando comparado com o aplicativo nativo e em relação ao uso de bateria devido a esse consumo maior do React Native a elevou o uso de energia em cerca de 6% a 8% (DORFER; DEMETZ; HUBER, 2020)

Abaixo podemos notar uma média de salário em algumas regiões e esta visão colabora para quando for contratar uma empresa, freelancer ou um desenvolvedor interno temos essa relação de valores atualmente o que influenciam diretamente nos custos do desenvolvimento de software.

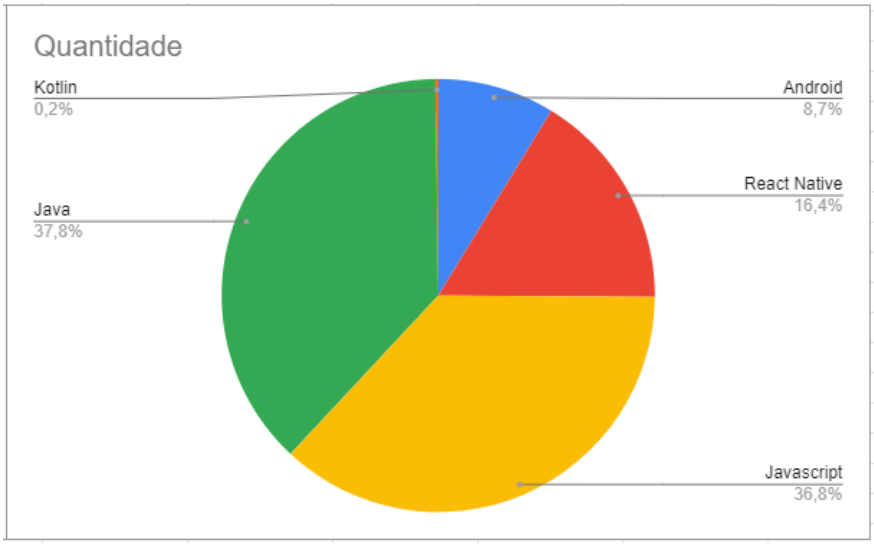
Tabela 2 - Relação Preço por Região do Engenheiro de Software

Região	Preço por Hora
USA	$100-150/hora
EUROPA OCIDENTAL	$65-100/hora
EUROPA ORIENTAL	$50-60/hora
UCRÂNIA	$40-60/hora
ÍNDIA	$20-30/hora
BRASIL	$35-$70/hora

Fonte: Adaptado de Makhov (2021) e Geerligs (2021)

Concluímos que, como fator importante para a escolha dos frameworks, colocamos a disponibilidade de pacotes e plugins para colaborar com o desenvolvimento de funcionalidades. E foi pesquisado no gerenciador de pacotes NPM todos os disponíveis para ambas as ferramentas Java/Kotlin e React Native. É observável que os plugins diretamente focados em React Native são o dobro dos específicos direcionados ao Android e devido ao Kotlin ser uma ferramenta ainda muito nova, vemos uma escassez de pacotes diretamente para esta linguagem, demonstrado assim no gráfico abaixo.

Gráfico 5 – Distribuição de Plugins por Linguagem



Fonte: Adaptado do NPM (2021)

4. CONCLUSÃO E CONSIDERAÇÕES FINAIS

Ao quantificar os dados que são relevantes para demonstrar de forma eficiente as opções benéficas e os contrapontos de implementar o uso de framework híbrido em relação ao nativo, com foco no React Native e Java/Kotlin, demonstramos as potencialidades de ambas e o crescimento de cada uma no mercado de tecnologia.

A pesquisa impacta positivamente no âmbito do desenvolvimento de software e colabora com empresas e profissionais que podem usar esta pesquisa como uma fonte de referência para auxiliar a fazer a melhor escolha de tecnologia para desenvolver seus produtos. Colocando de forma clara as vantagens e desvantagens de cada escolha. E ajudar a diminuir o índice de fracassos das construções de softwares por incompatibilidade de problema de negócio e recursos ofertados pelas ferramentas.

Ampliamos a gama de conhecimentos técnicos de desenvolvimento de software na atualidade relacionados aos métodos nativos e híbridos, com React Native e Java/Kotlin, apresentamos a idealização do desenvolvimento multiplataforma como uma forma de otimização de processos e métodos para uma empresa ou organização.

Demonstramos o crescimento da metodologia híbrida nos últimos anos com o objetivo de apresentar seu impacto positivo na atualidade. E analisamos a sua projeção de crescimento a longo prazo usando os dados coletados. Provendo assim uma visão crítica aos leitores deste estudo para que façam as devidas ressalvas quando estiverem em posições cruciais em relação ao planejamento do desenvolvimento de software.

Reforçando nossos objetivos, podemos aferir que o custo do desenvolvimento multiplataforma pode ser 50% mais barato que o nativo, devido o reaproveitamento de código para dois sistemas operacionais Android e iOS. As limitações identificadas foram principalmente a escassez de bibliotecas para Kotlin, e também uma dependência muito grande dos recursos disponibilizados pela Google que é a própria mantenedora da documentação, e já o React Native possui o contrário, as bibliotecas são inúmeras e são mantidas pela comunidade, tendo como ponto negativo dessas bibliotecas a possibilidade de

serem descontinuada a qualquer momento. Já a oferta de profissionais é limitada para ambas as tecnologias, impulsionando os salários e aumentando as oportunidades para os desenvolvedores que querem ingressar na área. A manutenibilidade das ferramentas é bem clara e pode ser analisada pela documentação oficial o que é perfeito em ambos os casos no qual você sempre terá acesso as informações e especificações na versão em que estiver trabalhando. Já a produtividade não possui um parecer direto, pois depende muito do que podemos afirmar, observando que o desenvolvimento nativo vai gastar mais tempo ou uma quantidade maior de mão de obra, já que o multiplataforma necessita de um desenvolvedor para codificar para as duas.

REFERÊNCIAS

ABEINFO (São Paulo, Brasil). Associação de Empresas e Profissionais da Informação. In: 2021: 5 tendências que vão ditar o sucesso dos apps no Brasil. Barra Funda - SP, 28 jan. 2021. Disponível em: https://abeinfobrasil.com.br/2021-5-tendencias-que-vao-ditar-o-sucesso-dos-appsnobrasil/#:~:text=A%20descoberta%20de%20novos%20aplicativos,demanda%20reprimida%20por%20dispositivos%205G.&text=Nos%20primeiros%20tr%C3%AAs%20meses%20da,h%C3%A1%20um%20ano%20ou%20mais. Acesso em: 3 maio 2021.

ANDROID. Adicionar o Kotlin a um app existente. In: Adicionar o Kotlin a um app existente. [S. l.], 28 dez. 2020. Disponível em: https://developer.android.com/kotlin/add-kotlin#:~:text=Android%20Studio%20provides%20full%20support,refactoring%2C%20debugging%2C%20and%20more. Acesso em: 4 maio 2021.

BOEHM, BARRY. SOFTWARE ENGINEERING ECONOMICS. IEEE Transactions on Software Engineering, [S. l.], p. 200-217, 1984. Disponível em: http://bls.buu.ac.th/~s55103/00CourseResource/Boehm-SE-Economics.pdf. Acesso em: 3 abr. 2021.

DAL 'OSTO, Fábio. Método para avaliação de ambientes de desenvolvimento de software combinando CMM e GQM. Porto Alegre - RS, 2003. Disponível em: https://lume.ufrgs.br/handle/10183/6818#:~:text=O%20modelo%20GQM%20descreve%20uma,e%20quest%C3%B5es%20que%20as%20satisfazem.&text=Esses%20grafos%20permitem%20a%20visualiza%C3%A7%C3%A3o,suas%20metas%2C%20quest%C3%B5es%20e%20m%C3%A9tricas. Acesso em: 10 maio 2021.

DAXX (Estados Unidos). How Many Software Developers Are in the US and the World? [S. l.], 9 fev. 2020. Disponível em: https://www.daxx.com/blog/development- trends/number-software-developers-world. Acesso em: 11 maio 2021.

DORFER, Thomas; DEMETZ, Lukas; HUBER, Stefan. Impact of mobile cross-platform development on CPU, memory and battery of mobile devices when using common mobile app features. Procedia Computer Science, [s. l.], v. 175, n. 2019, p. 189–196, 2020. Disponível em: https://doi.org/10.1016/j.procs.2020.07.029

DENYSOV, Andrew. Native App vs Cross-Platform: Cost Comparison in 2021 [S. l.], 16 jun 2020. Disponível em: https://inveritasoft.com/blog/how-much-cross-platform-mobile-app-development-costs .Acesso em: 29 outubro 2021.

FACEBOOK. React Native. In: React Native. [S. l.], 2021. Disponível em: https://reactnative.dev. Acesso em: 4 maio 2021.

FREIRE, Herval. Calculando Estimativas: o Método de Pontos de Caso de Uso [S. l.] fev. 2003. Disponível em: https://www.cin.ufpe.br/~raa3/projetao/2Iteracao/HelpPCU/HelpPCU/usecasepoints.pdf. Acesso em: 11 maio 2021.

MAKHOV, Vitaly. App development cost: how to manage your budget properly [S. l.], 27 set 2021. Disponível em: https://doit.software/blog/app-development-cost#screen1 .Acesso em: 29 outubro 2021.

NPM. NPM Packages. In: NPM Packages Search [S. l.], 2020. Disponível em: https://www.npmjs.com/search?q=javascript%20java%20react%20native%20kotlin. Acesso em: 29 outubro 2021.

OSMAN, Maddy. Estatísticas e Fatos do LinkedIn (2021). 20 jul. 2021. Disponível em: https://kinsta.com/pt/blog/estatisticas-e-fatos-do-linkedin/

SABINO, Felipe; LEÃO, Pedro. Híbrido vs Nativo. In: Híbrido vs Nativo. [S. l.], 16 out. 2016. Disponível em: https://medium.com/taqtilebr/h%C3%ADbrido-vs-nativo-c8591df0dce6. Acesso em: 3 maio 2021.

SENA, Victor; GRANATO, Luísa. 260.000 vagas sem dono: um raio-x das vagas mais quentes agora (e no futuro). EXAME, [S. l.], p. 1, 18 fev. 2021. Disponível em: https://exame.com/carreira/260-000-vagas-de-trabalho-sem-dono-conheca-o-setor-que-ganhou-forca-com-a-pandemia/. Acesso em: 10 maio 2021.

SENA, Victor. Veja 15 profissões em alta para 2021: saúde e tecnologia saem na frente. Exame, [S. l.], p. 1, 3 fev. 2021. Disponível em: https://exame.com/carreira/veja-15-profissoes-em-alta-para-2021-saude-e-tecnologia-saem-na-frente/. Acesso em: 11 maio 2021.

SINGHA , Tribhuvan; SINGHA, Ranvijay; MISHRAA, Krishn Kumar. Software Cost Estimation Using Environmental Adaptation Method. Procedia Computer Science, [S. l.] 1, 2018. Disponível em: https://www.sciencedirect.com/science/article/pii/S1877050918321008?via%3Dihub Acesso em: 13 abr. 2021.

STACK OVERFLOW. Programming, Scripting, and Markup Languages. In: Programming, Scripting, and Markup Languages. [S. l.], 2020. Disponível em: https://insights.stackoverflow.com/survey/2020/#technology-programming-scripting-and-markup-languages. Acesso em: 4 Maio 2021.

STATISTA. Distribution of Android app developers worldwide as of 1st quarter 2018, by country. 21 OUT. 2021. Disponível em: https://www.statista.com/statistics/271988/android-app-developer-country/. Acesso em: 11 novembro 2021.

STATISTA. Telecommunications. In: Share of global smartphone shipments by operating system from 2014 to 2023. [S. l.], 19 mar. 2021. Disponível em: https://www.statista.com/statistics/272307/market-share-forecast-for-smartphone- operating-systems/#:~:text=Smartphones%20running%20the%20Android%20operating,percent%20share%20of%20the%20market. Acesso em: 4 Maio 2021.

STATISTA. Telecommunications. In: Cross-platform mobile frameworks used by software developers worldwide in 2019 and 2020. [S. l.], 2 jun. 2020. Disponível em:https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/. Acesso em: 4 maio 2021.

APÊNDICE A - Estimativa de Esforço Baseado em Pontos de Caso de Uso - React Native

1 - Mensurando Complexidade dos Atores

Ator	Interface	Peso
Simples	Interface de Programa (API)	1
Médio	Outro sistema interagindo através de um protocolo de comunicação, como TCP/IP ou FTP	2
Complexo	Um usuário interagindo através de uma interface gráfica (stand-alone ou Web)	3

2 - Mensurando Complexidade dos Use Cases (PCUNA - Pontos de Casos de Uso Não Ajustados)

Tipo de Caso de Uso	Número de Trasações	Peso
Simples	Até 3	1
Médio	4 a 7	2
Complexo	7 ou mais	3

3 - Considerando Fatores Técnicos (FCT - Fatores de Complexidade Técnica)

Fator	Descrição	Peso
T1	Sistema distribuído	2
T2	Tempo de Resposta	2
T3	Eficiência	1
T4	Processamento complexo	1
T5	Código reusável	1
T6	Facilidade de instalação	0.5
T7	Facilidade de uso	0.5
T8	Portabilidade	2
T9	Facilidade de mudança	1
T10	Concorrência	1
T11	Recursos de segurança	1
T12	Acessível por terceiros	1
T13	Requer treinamento especial	1

4 - Considerando Fatores Ambientais (FA - Fator Ambiental)

Fator	Descrição	Peso
F1	Familiaridade da equipe com os ambientes e frameworks de Desenvolvimento	1.5
F2	Experiência com a Aplicação em desenvolvimento	0.5
F3	Experiência em Orientação a Objetos	1
F4	Presença de analista experiente	0.5
F5	Motivação	1
F6	Requisitos estáveis	2
F7	Desenvolvedores em meio-expediente	-1
F8	Linguagem de programação difícil	-1

5 - Pontos de Caso de Uso (PCU - Pontos de Caso de Uso)

Valor	Formula	Resultado
PCU	PCUNA * FCT * FA	
Pessoa-hora por unidade de PCU		1
Estimativa em pessoa-hora		1
Tamanho da Equipe		1
Estimativa em horas		

Atores	Peso
0	1
0	2
1	3
Total	3
Fórmula para Cáculo	AP = Atores X Peso (Depois soma todos os valores que foram multiplicados)

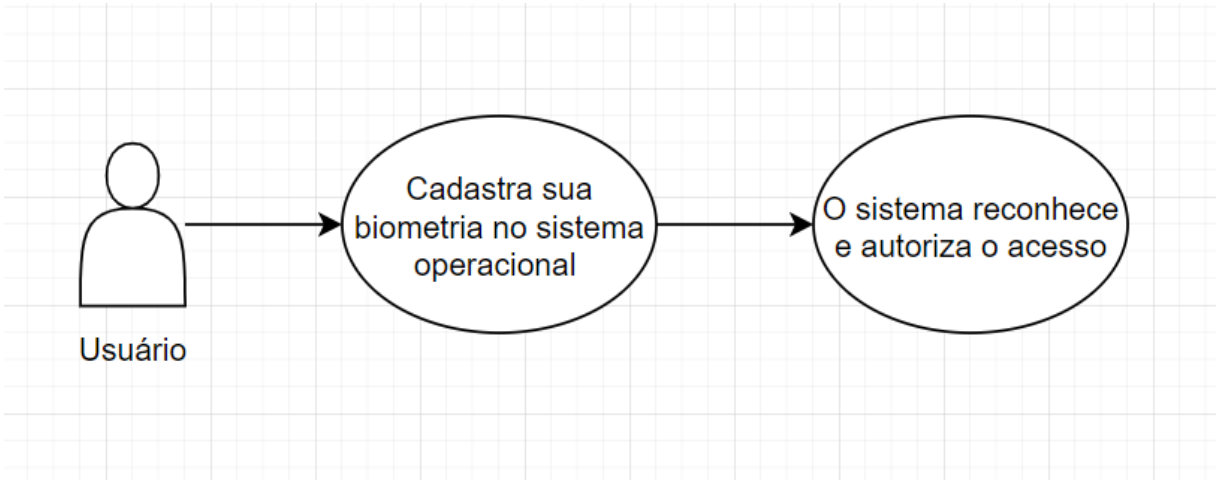
Qtd. de Caso de Uso	Peso
1	1
0	2
0	3
Total	1
Fórmula para Cáculo	QP = Qtd. de Caso de Uso X Peso (Depois soma todos os valores que foram multiplicados)
Pontos de Caso de Uso Não Ajustados	PCUNA = AP + TP (Neste caso é 4)

Atribuído	Peso
0	2
1	2
1	1
0	1
4	1
4	0.5
4	0.5
0	2
1	1
0	1
0	1
0	1
0	1
Total	12
Fórmula para Cáculo	0.6 + (0.01 x Total) (Neste caso o valor é 0.72)

Atribuído	Peso
1	1.5
4	0.5
4	1
0	0.5
4	1
1	2
0	-1
0	-1
Total (Atribuído x Peso)	13.5
Fórmula para Cáculo	1.4 + (-0.03 x Total) (Neste caso o valor é 0.995)

Resultado Final Estimativa	Realidade do tempo gasto
2.8	2.5hrs
1	
2.8	
1	
2.8hrs	

Caso de Uso


Cadastra sua biometria no sistema operacional
O sistema reconhece e autoriza o acesso
Usuário

APÊNDICE B - Estimativa de Esforço Baseado em Pontos de Caso de Uso - Java/Kotlin

1 - Mensurando Complexidade dos Atores

Ator	Interface	Peso
Simples	Interface de Programa (API)	1
Médio	Outro sistema interagindo através de um protocolo de comunicação, como TCP/IP ou FTP	2
Complexo	Um usuário interagindo através de uma interface gráfica (stand-alone ou Web)	3

2 - Mensurando Complexidade dos Use Cases (PCUNA - Pontos de Casos de Uso Não Ajustados)

Tipo de Caso de Uso	Número de Trasações	Peso
Simples	Até 3	1
Médio	4 a 7	2
Complexo	7 ou mais	3

3 - Considerando Fatores Técnicos (FCT - Fatores de Complexidade Técnica)

Fator	Descrição	Peso
T1	Sistema distribuído	2
T2	Tempo de Resposta	2
T3	Eficiência	1
T4	Processamento complexo	1
T5	Código reusável	1
T6	Facilidade de instalação	0.5
T7	Facilidade de uso	0.5
T8	Portabilidade	2
T9	Facilidade de mudança	1
T10	Concorrência	1
T11	Recursos de segurança	1
T12	Acessível por terceiros	1
T13	Requer treinamento especial	1

4 - Considerando Fatores Ambientais (FA - Fator Ambiental)

Fator	Descrição	Peso
F1	Familiaridade da equipe com os ambientes e frameworks de Desenvolvimento	1.5
F2	Experiência com a Aplicação em desenvolvimento	0.5
F3	Experiência em Orientação a Objetos	1
F4	Presença de analista experiente	0.5
F5	Motivação	1
F6	Requisitos estáveis	2
F7	Desenvolvedores em meio-expediente	-1
F8	Linguagem de programação difícil	-1

5 - Pontos de Caso de Uso (PCU - Pontos de Caso de Uso)

Valor	Formula	Resultado
PCU	PCUNA * FCT * FA	
Pessoa-hora por unidade de PCU		1
Estimativa em pessoa-hora		1
Tamanho da Equipe		1
Estimativa em horas		

Atores	Peso
0	1
0	2
1	3
Total	3
Fórmula para Cáculo	AP = Atores X Peso (Depois soma todos os valores que foram multiplicados)

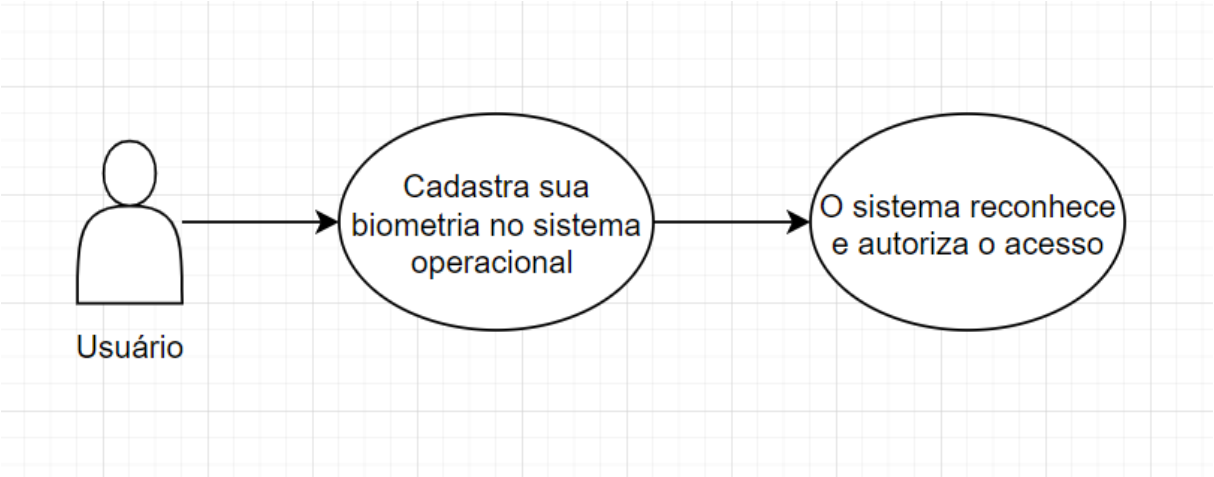
Qtd. de Caso de Uso	Peso
1	1
0	2
0	3
Total	1
Fórmula para Cáculo	QP = Qtd. de Caso de Uso X Peso (Depois soma todos os valores que foram multiplicados)
Pontos de Caso de Uso Não Ajustados	PCUNA = AP + QP (Neste caso é 4)

Atribuído	Peso
0	2
1	2
1	1
0	1
4	1
4	0.5
4	0.5
0	2
1	1
0	1
0	1
0	1
0	1
Total	12
Fórmula para Cáculo	0.6 + (0.01 x Total) (Neste caso o valor é 0.72)

Atribuído	Peso
0	1.5
2	0.5
4	1
0	0.5
4	1
1	2
0	-1
0	-1
Total (Atribuído x Peso)	12.5
Fórmula para Cáculo	1.4 + (-0.03 x Total) (Neste caso o valor é 1.025)

Resultado Final Estimativa	Realidade do tempo gasto
2.95	4.5hrs
1	
2.95	
1	
2.95hrs	

Caso de Uso


Usuário
Cadastra sua biometria no sistema operacional
O sistema reconhece e autoriza o acesso

APENDICE C – Histórico de Manutenibilidade Android

Nome da Versão	Número da Versão	Data de Lançamento
Sem codinome oficial	1.0	23 de setembro de 2008 (13 anos)
	1.1	9 de fevereiro de 2009 (12 anos)
Cupcake	1.5	27 de abril de 2009 (12 anos)
Donut	1.6	15 de setembro de 2009 (12 anos)
Eclair	2.0	27 de outubro de 2009 (12 anos)
	2.0.1	3 de dezembro de 2009 (11 anos)
	2.1	11 de janeiro de 2010 (11 anos)
Froyo	2.2 - 2.2.3	20 de maio de 2010 (11 anos)
Gingerbread	2.3 - 2.3.2	6 de dezembro de 2010 (10 anos)
	2.3.3 - 2.3.7	9 de fevereiro de 2011 (10 anos)
Honeycomb	3.0	22 de fevereiro de 2011 (10 anos)
	3.1	10 de maio de 2011 (10 anos)
	3.2 - 3.2.6	15 de julho de 2011 (10 anos)
Ice Cream Sandwich	4.0 - 4.0.2	18 de outubro de 2011 (10 anos)
	4.0.3 - 4.0.4	16 de dezembro de 2011 (9 anos)
Jelly Bean	4.1 - 4.1.2	9 de julho de 2012 (9 anos)
	4.2 - 4.2.2	13 de novembro de 2012 (9 anos)
	4.3 - 4.3.1	24 de julho de 2013 (8 anos)
KitKat	4.4 - 4.4.4	31 de outubro de 2013 (8 anos)
	4.4W - 4.4W.2	25 de junho de 2014 (7 anos)
Lollipop	5.0 - 5.0.2	4 de novembro de 2014 (7 anos)
	5.1 - 5.1.1	2 de março de 2015 (6 anos)
Marshmallow	6.0 - 6.0.1	2 de outubro de 2015 (6 anos)
Nougat	7.0	22 de agosto de 2016 (5 anos)
	7.1 - 7.1.2	4 de outubro de 2016 (5 anos)
Oreo	8.0	21 de agosto de 2017 (4 anos)
	8.1	5 de dezembro de 2017 (3 anos)
Pie	9	6 de agosto de 2018 (3 anos)
Android 10	10	3 de setembro de 2019 (2 anos)
Android 11	11	8 de setembro de 2020 (1 ano)
Android 12	12	4 de outubro de 2021 (0 anos)