Cenário dos aplicativos móveis na Senior Sistemas e análise para adoção de uma tecnologia

Tiago Boeing[1]

[1]Senior Sistemas S/A
aplicativos-arquitetura@senior.com.br – Blumenau – SC – Brazil

tiago.boeing@senior.com.br

Novembro de 2020

Resumo. *Este é um relatório produzido pelo departamento de Pesquisa e Arquitetura da Senior Sistemas para análise quanto a adoção de alternativas tecnológicas para o desenvolvimento móvel, considerando o cenário técnico atual das aplicações existentes e fundamentando a decisão com base nos resultados obtidos.*

1 Introdução

As tecnologias para desenvolvimento de aplicativos móveis têm sido um importante marco para a transformação digital nas últimas décadas. Embora em 1983 Steve Jobs tenha previsto um mercado de aplicativos (1983 to today: a history of mobile apps, 2015) e em 1997 o Nokia 6110 embutiu um jogo arcade chamado de "Snake", sendo o que muitos consideram como o primeiro aplicativo móvel (The History of Mobile Apps, 2019), apenas em 2008, um ano após o lançamento do iPhone que a App Store foi apresentada ao mundo e revolucionou a forma de consumo e distribuição de aplicativos.

Por não existir uma cultura *mobile first* os aplicativos normalmente replicam as funcionalidades dos produtos já existentes, não necessariamente com uma experiência específica para cada contexto. A situação atual das aplicações tende a exigir inúmeras horas com manutenção de código legado. Atualmente existem aplicativos escritos em diferentes tecnologias e versões, sendo completamente diferentes umas das outras, este cenário tende a impossibilitar a criação de uma ampla base de conhecimento. Dentre as tecnologias, temos Firemonkey (Delphi), Ionic (1, 3 e 4), Corona, Flutter, Xamarin, Android (Java) e React Native.

Estamos considerando tecnologias *cross-platform* e deixando de lado o desenvolvimento integralmente nativo por se contrapor a alguns de nossos princípios, como *onboarding* de novos desenvolvedores e sendo a produtividade o principal deles. Trazemos Kotlin a princípio de comparação. No nativo todas implementações possuem seu custo multiplicado pela quantidade de plataformas que o aplicativo será lançado, há uma clara barreira para que desenvolvedores ingressem no desenvolvimento móvel por ser necessário aprender linguagens diferentes e específicas de cada uma das plataformas, alto custo com hardware, mão de obra significativamente menor se comparada às tecnologias *cross-platform* e principalmente a impossibilidade de compartilhamento de conhecimento entre as equipes, independentemente da plataforma operada. Esta pesquisa faz parte de uma força-tarefa da empresa relacionada aos aplicativos móveis e tem por

objetivo realizar a coleta de dados e documentar as descobertas, estudar alternativas para desenvolvimento móvel e decidir quanto a adoção de determinada tecnologia.

2 Metodologia

A metodologia se baseia na análise dos aplicativos da Senior, sendo estes distribuídos em vários *frameworks*/tecnologias utilizadas e times que não necessariamente possuem desenvolvedores que atuam ou possuem vivência em ambiente de aplicativos móveis. A exemplo das tecnologias, muitas como Firemonkey/Delphi e Corona se encontram defasadas para o cenário tecnológico atual.

O conjunto de fatores: mão de obra não necessariamente especialista, tecnologias diversas e em alguns casos defasadas contribuem para a depreciação das soluções, além de a longo prazo necessitar de maior investimento da empresa na necessidade de manutenção de código legado, sendo estes muitas vezes relacionados às dependências da comunidade que precisam ser corrigidas pelos próprios desenvolvedores. Tais situações contribuem para um cenário onde realizar a estimativa de tempo e complexidade da correção torna-se um cenário volátil, sendo necessário um profissional alocado para longas e indefinidas jornadas.

Percebendo do problema foram realizadas conversas com os desenvolvedores responsáveis dos times que lidam com aplicativos móveis, buscamos entender quais as tecnologias cada solução utiliza, recursos nativos, objetivos e perspectivas futuras, além da experiência de desenvolvimento. Nós do departamento de arquitetura elencamos alguns critérios considerados importantes ao decidir quanto a adoção de determinada tecnologia. Entendemos que em alguns casos o desenvolvimento integralmente continuará fazendo sentido, mas nesses cenários uma avaliação externa a este estudo deve ser realizada.

2.1 Cálculo

Criamos categorias que possuem diversos critérios, onde cada critério apresenta status e pesos individuais.

- Categoria: Produtividade, Resiliência e outras;
- Critério: *Cross-platform*, Baixo à médio esforço para criar CI/CD e outros.

2.2 Anatomia

2.2.1 Categoria

Uma categoria pode conter n critérios e tem como objetivo subdividi-los baseado nos princípios elencados. O exemplo abaixo não leva em consideração os valores reais, servindo apenas para demonstrar o cálculo.

Tabela 1 Anatomia de uma categoria com critérios

Longo prazo	Status	Peso do critério	Nota atingida
Tamanho da comunidade	Atende/Alto (100%)	0.5	0.5
Comunidade ativa (*Stack Overflow*, etc)	Atende parcialmente (50%)	1	0.5

Pontuação máxima da categoria	1.5
Nota atingida	**1 (66.66%)**

A pontuação máxima é obtida através do cálculo do melhor cenário, isto é o máximo de pontos possíveis na categoria.

$$Pontuação\ máx.\ da\ categoria = \sum pesos\ dos\ critérios$$

Para obtermos a fatia que um critério ocupa dentre todos os outros basta somarmos a pontuação máxima de todos eles. Com essas informações é possível plotar um gráfico para facilitar a visualização.

$$fatia\ do\ critério = \frac{pontuação\ máx.\ da\ categoria\ avaliada}{\sum pontuações\ máx.\ das\ categorias}$$

2.2.2 Critério

Um critério pertence a uma categoria e é o ponto que desejamos avaliar cada tecnologia. Critérios possuem:

- Status;
- Peso/Multiplicador: valor decimal que representa a importância deste critério considerando nosso cenário atual.

$$Nota\ atingida = status * peso$$

Tabela 2 Anatomia de um critério

Critério	Status	Peso/Multiplicador	Nota atingida
Curva de aprendizado	Atende parcialmente/Médio (0.5 ou 50%)	0.5	0.25 (50%)

2.2.3 Status

Utilizamos o status para informar se determinada tecnologia atende ao critério estabelecido. Cada status possui um multiplicador. A exemplo de uma tecnologia que em determinado critério atende parcialmente a nota atingida valerá apenas metade de seu peso/multiplicador.

Tabela 3 Status de um critério

Status	Multiplicador
Não atende/baixo	0 (0%)
Atende parcialmente/médio	0.5 (50%)
Atende/alto	1 (100%)

3 Cenário atual na Senior Sistemas

Em um estudo prévio buscando mapear os frameworks/tecnologias utilizadas para desenvolvimento, lojas/plataformas de publicação e recursos nativos utilizados foram identificados 35 aplicativos, estes incluem todas as unidades conforme tabela abaixo:

Tabela 4 Aplicativos da Senior Sistemas

Aplicativo	**Tecnologia**	**Plataforma (s) publicada (s)**	**Recurso (s) nativo (s) utilizado (s)**
AgroInfo		Android e iOS	
AgroPlus Agrodanieli			
Approvo	Flutter	Android e iOS	Câmera e notificações push
CHK (Check-List de Viagem) ***Atualmente sem atualizações. A equipe está estudando a possibilidade de manutenção.**	Firemonkey/Delphi	Android	Câmera, galeria, armazenamento e geolocalização
Cotei	Flutter	Android e iOS	
CRM (Relacionamento com Cliente)	Firemonkey/Delphi	Android	Armazenamento
Croqui (GIDION) ***Segundo informações não se encontra mais na loja.**	Ionic 3	Android	
Decision Center	Ionic 4	Android e iOS	Armazenamento, notificações push, acesso à rede e execução em segundo plano.
GCE (Coletas e Entregas) ***Sendo reescrito para React Native.**	Ionic 3	Android	Câmera, galeria, geolocalização, execução em segundo plano e armazenamento.
GED (TMS)	Ionic 4	Android	Galeria, armazenamento, microfone e acesso à rede.
Gestão de incidentes Mobile	Ionic 3	Android e iOS	Câmera e armazenamento.
Gestão de Pessoas | HCM	Ionic 1	Android e iOS	
Gestão de Relacionamento/CRM ***Sendo reescrito para React Native.**	Corona	Android e iOS	

Gestão Empresarial PME/Go Up	Ionic 3	Android e iOS	
Gestão Empresarial ERP	Ionic 3	Android e iOS	
Homepay	Flutter	Android e iOS	
JMT (Jornada de Motorista)	Firemonkey/Delphi	Android	
Marcação de Ponto 2.0 | HCM ***Estudos para utilizar reconhecimento facial, touch ID e atualizar do Ionic 3 para o 5 ou codificar nativo em Kotlin.**	Ionic 3	Android e iOS	Geolocalização, armazenamento, acesso à rede, device ID, bluetooth e NFC.
Marcação de Ponto REP | HCM	Ionic 3	Android e iOS	Geolocalização
Marcação de Ponto | HCM	Ionic 3	Android e iOS	Geolocalização, câmera e galeria.
MCE (Coletas e Entregas)	Firemonkey/Delphi	Android	
MCF (Dmari)	Ionic 4	Android	
MCF (Ouro Negro)	Ionic 4	Android	
MCF (Risso)	Firemonkey/Delphi	Android	
MCF (Trânsito Brasil)	Firemonkey/Delphi	Android e iOS	
MCF (Transville)	Ionic 3	Android	
MobAgro ***Não é publicado em lojas, a APK é instalada diretamente.**	Firemonkey/Delphi	Android	
MobServ ***Não é publicado em lojas, a APK é instalada diretamente.**	Xamarin	Android	
Requisita	Flutter	Android e iOS	Microfone e câmera
Ronda de vigilância ***Estudo de melhoria do app em andamento e análise para criar versão para iOS.**	Android/Java	Android	Geolocalização, telefone, armazenamento, câmera, microfone, acesso à rede, device ID, gerenciar contas e definir senhas.
Samcard	Ionic 3	Android e iOS	Geolocalização, bluetooth, acesso à rede, prevenir bloqueio do

			dispositivo e rodar ao inicializar.
SIG (Informações Gerenciais)	Firemonkey/Delphi	Android e iOS	Gráficos
Venda+	Flutter	Android e iOS	
Vimob - Vendas imobiliárias	Flutter	Android e iOS	
Voice picking	Android/Java	Android	

Além de uma quantidade significativa de aplicativos, eles estão distribuídos em aproximadamente 13 diferentes áreas, fortalecendo ainda mais a importância da padronização da tecnologia utilizada nestas soluções, possibilitando gerar uma base de conhecimento unificada que facilite o compartilhamento entre equipes e *onboarding* de novos desenvolvedores.

É importante considerar que atualmente existe uma clara diferenciação das aplicações móveis, web e desktop na Senior. As soluções são construídas sob medida para cada contexto (plataforma) específico, embora não necessariamente a experiência seja, desejamos oferecer funcionalidades diferenciadas para nossos usuários e não simplesmente replicar o que já possuímos na web para mobile por exemplo, nestes casos um *webview* atenderia, embora estaríamos em desconformidade com os termos da App Store (APPLE).

3.1 Mega

A unidade Mega se encontra avançada no cenário de mobilidade, chegando inclusive a realizar POCs e estudos das tecnologias há dois anos. A decisão foi de apostar em *Flutter*, à época ainda considerado um risco, já que não era considerado pronto para produção, o que aconteceu na versão 1.0. Os testes contemplaram projetos isolados, onde desenvolvedores sem experiência no quesito mobile, munidos apenas da documentação oficial e materiais na internet apresentaram alta taxa de aprendizado e produtividade, sendo considerado um sucesso pela empresa. Atualmente a Mega possui todos os aplicativos em *Flutter*.

Um dos apps (Approvo) chegou a ser desenvolvido inicialmente em *React Native* e mantido por cerca de seis meses na tecnologia. Como *Flutter* apresentou-se uma solução produtiva, de fácil aprendizado e estável acabou sendo convertido mais tarde também para *Flutter*, padronizando as soluções e mantendo apenas um *codebase*.

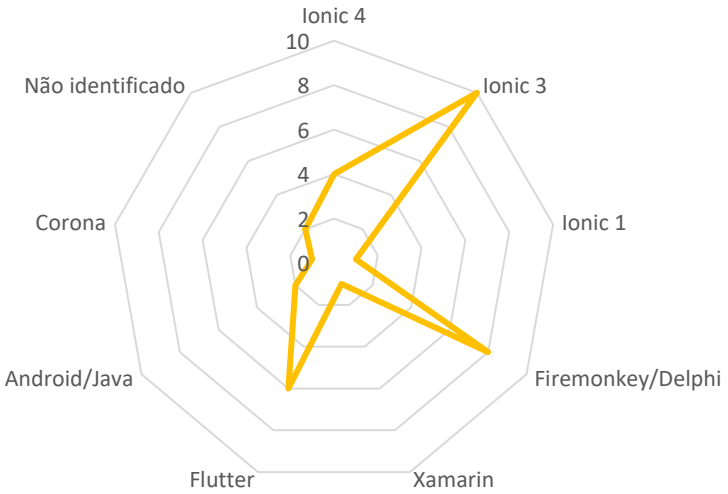
3.2 Aplicativos por tecnologia

Os times atualmente já estão habituados a lidar com tecnologias *cross-platform,* o cenário evidencia que possuímos oito diferentes *frameworks*/tecnologias sendo utilizadas, sendo sete *cross-platform*. É perceptível que todas as equipes lidam com problemas semelhantes, buscando alternativas para reescrita e/ou atualização dos aplicativos mantidos.

A preocupação apresentada pelos times está relacionada a garantia de estabilidade dos aplicativos. É comum localizar repositórios abandonados e ser necessário recorrer a

projetos mantidos pela comunidade ou terceiros. Como alguns frameworks não possuem soluções previamente implementadas para lidar com recursos nativos, dependendo da complexidade da aplicação é imensurável a quantidade de dependências não oficiais necessárias. Conforme ampliamos a quantidade de dependências maximizamos de forma significativa a exposição à possíveis *bugs* causados por implementações externas.

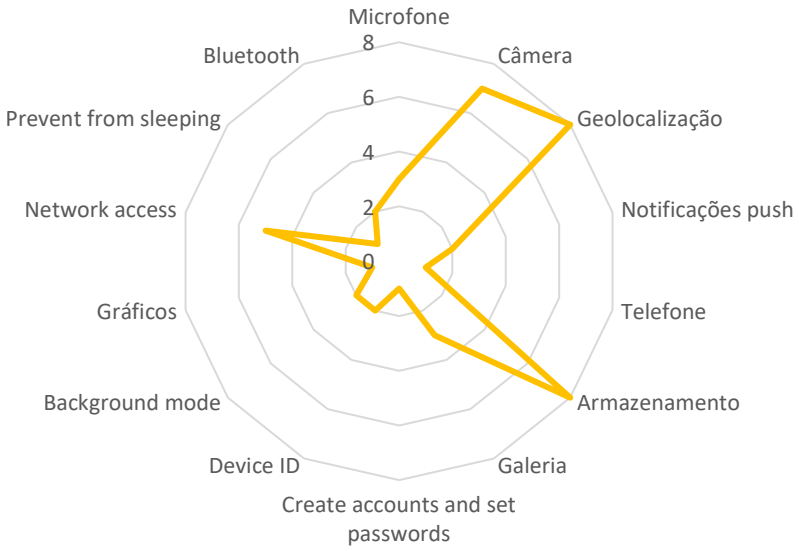
Figura 1 Aplicativos por framework utilizado



3.3 Recursos nativos utilizados

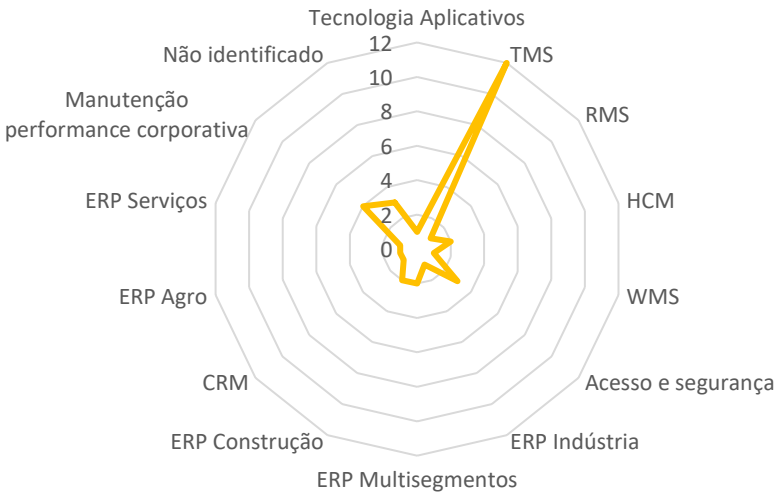
Evidenciando a importância da preocupação com estabilidade, conforme mencionado anteriormente, ao ampliarmos a complexidade dos aplicativos e a necessidade de interação com recursos nativos, dependendo do *framework* escolhido podemos estar maximizando a exposição a instabilidades, atualmente o cenário demonstra que 14 aplicativos, dos 35 possuem a necessidade de acessar recursos nativos.

Figura 2 Recursos nativos utilizados pelos aplicativos



3.4 Aplicativos por área/time

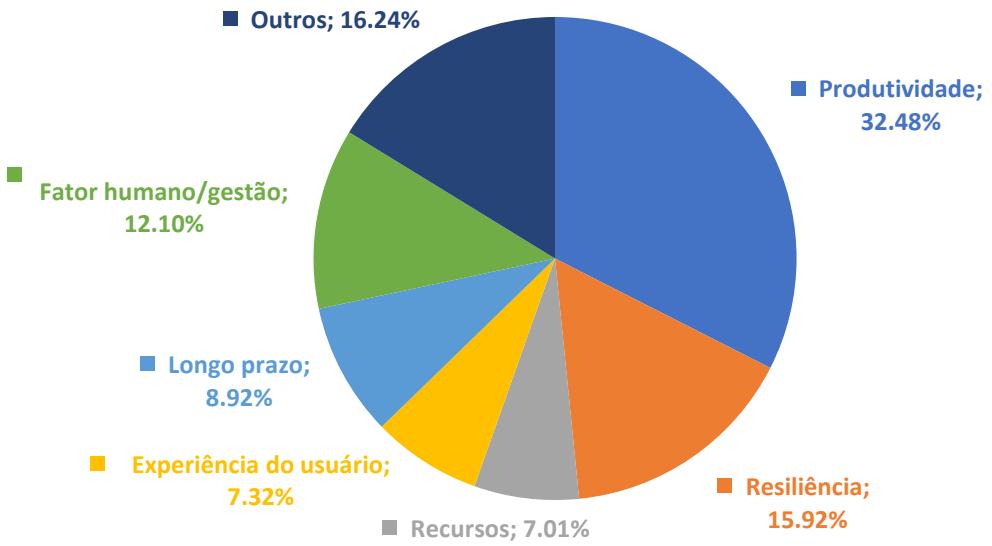
Figura 3 Aplicativos mantidos por cada área/time



4 Categorias e critérios estabelecidos

A fatia ocupada por cada uma das categorias não indica necessariamente seu nível de importância dentro da organização, mas sim técnica. Por exemplo, claro que nos preocupamos com a experiência do usuário, mas dentre as tecnologias analisadas temos avaliações semelhantes e conseguimos atingir uma boa experiência para a complexidade apresentada atualmente. Buscamos definir categorias que são capazes de confrontar as tecnologias de formas mais distintas e que tragam prós e contras, possibilitando identificar critérios "mais compatíveis" com nossos princípios, sempre de forma realista e considerando o cenário atual da empresa.

Figura 4 Distribuição dos critérios estabelecidos



4.1 Produtividade

A primeira categoria e que possui o maior peso é a produtividade, considerada indispensável por nós e que servirá como indicador para mensurarmos o atingimento dos objetivos. Nesta categoria listamos características dos *frameworks* que auxiliam no fluxo de trabalho.

Tabela 5 Critérios da categoria produtividade

Produtividade (pontuação máx.: 5.1 ou 32.48%)	**Peso/Multiplicador**
Cross-platform (Android e iOS)	2
CI/CD de baixo à médio esforço	1
Multiplataforma (mobile/web/desktop)	0.15
Hot reload	0.75
Ferramentas para debug	0.6
Integração com IDEs	0.6

4.1.1 Cross-platform (Android e iOS)

Conforme o estudo, os times já estão habituados a trabalhar com *frameworks* híbridos (*cross-platform*). Implementar determinada funcionalidade apenas uma vez e tê-la em ambas as plataformas é indispensável, o histórico de decisões tecnológicas permite tal conclusão, onde das oito tecnologias utilizadas, sete são *cross-platform.*

Embora existam problemas com as soluções móveis da Senior atualmente, isto não é um indicador de que *cross-platform* seja ineficiente. Em sua maioria estes problemas estão ligados à falta de priorização de tarefas de manutenção nos aplicativos móveis ao longo dos anos, resultando em seu abandono total ou parcial, ou uma base de código de pouca qualidade. A adoção e constante investimento de grandes empresas em *cross-platform* já provou que quando bem aplicada resulta em produtividade, redução de custos e gera bons resultados.

4.1.2 Implementação de baixo à médio esforço de CI/CD

Nossa perspectiva é de ter um fluxo de integração contínua durante o processo de *build* e liberação de *releases* nas lojas de aplicativos, o mesmo processo que possuímos atualmente para liberações de versões nos serviços da Plataforma Senior X. Dessa forma este critério é indispensável.

Esperamos abandonar de vez os fluxos de liberações que exigem incontáveis horas de trabalho totalmente manual do desenvolvedor. É comum que a implementação das funcionalidades tenha tempo inferior à etapa de liberação, definitivamente isto é o que desejamos evitar. A equipe de *DevOps* acompanha de forma próxima este critério e com base na tecnologia adotada será capaz de realizar estudos mais elaborados.

Avaliaremos a complexidade para implementação de um fluxo de CI/CD em cada uma das tecnologias analisadas, esperamos que o fluxo tenha entre baixa à média complexidade. A possibilidade de utilização com *GitLab Self-Hosted* é interessante, visto que atualmente este é nosso provedor *Git* oficial.

4.1.3 Multiplataforma (mobile/web/desktop)

Embora nosso desejo seja de continuar a construir experiências individuais para cada contexto e atualmente há essa diferenciação entre web e móvel dentro dos times, consideramos como um "ponto extra" para o *framework*/tecnologia que é capaz de oferecer esta possibilidade. Diferente do *cross-platform* que visa avaliar a possibilidade de utilizar a tecnologia em diferentes sistemas operacionais móveis (Android e iOS), o critério multiplataforma considera qualquer dispositivo, seja móvel ou não.

4.1.4 Hot reload

Durante o fluxo de desenvolvimento, este é um dos recursos mais vitais para auxiliar o desenvolvedor, sendo indiscutível sua importância. Realizar modificações no código-fonte e visualizar quase que de forma instantânea o resultado poupa tempo de desenvolvimento, diagnóstico de problemas e melhora a experiência com a tecnologia.

Neste critério esperamos que a tecnologia analisada seja capaz de oferecer o *hot reload* e não apenas *live reload*. A diferença entre ambos é que, *hot reload* mantém o estado (*state)* da aplicação e altera apenas as partes modificadas, já *live reload* força que toda a aplicação seja recarregada, eliminando os estados salvos. Toda aplicação com *hot reload* implicitamente contém *live reload*, já o contrário não é verdade.

4.1.5 Ferramentas para debug/depuração

Ferramentas para depuração de código são valorizadas sempre, esperamos que a tecnologia seja altamente depurável durante o fluxo de desenvolvimento. Isto poupa tempo, permite analisar nos mínimos detalhes as chamadas realizadas e auxilia na resolução de problemas com maior facilidade.

4.1.6 Integração com IDEs

Neste critério a tecnologia que permite ser desenvolvida pelo *Visual Studio Code* ou que fornece editores específicos, mas *open source* serão mais bem avaliadas.

O *Visual Studio Code* é *open source* sendo também o editor de código mais popular da atualidade (KEETON, 2019) e a ferramenta oficial para desenvolvimento web/mobile atualmente na empresa, ao qual possuímos um *style guide* específico, naturalmente esperamos uma tecnologia seja compatível.

4.2 Resiliência

Ter uma aplicação que lide bem com as situações mais diversas e "assuma" a responsabilidade de manter tudo bem, mesmo quando houver quebras de compatibilidade nos sistemas operacionais minimiza nossos esforços com manutenção de código e exposição a cenários não previstos, isto gera economias significativas se considerarmos que atualmente desprendemos incontáveis horas para essas atividades, como a correção de bibliotecas ou adequação do *build* para as mudanças recentes das plataformas/lojas.

Tabela 6 Critérios da categoria resiliência

Resiliência (pontuação máx.: 2.5 ou 15.92%)	**Peso/Multiplicador**
Poucas quebras de compatibilidades (*breaking changes*)	0.75
Poucas dependências de produção	1

Capacidade de teste	0.75

4.2.1 Pouca quebra de compatibilidade

Quebras de compatibilidades são normais em qualquer tecnologia, porém uma quantidade significativa de quebras em um curto período tende a exigir atenção. Alterações das APIs costumam gerar impactos diversos, onde em sua maioria será necessária uma ação de manutenção por parte dos usuários (desenvolvedores). Além disso as quebras podem de fato causar transtornos incalculáveis para nossas aplicações, muitas das vezes fazendo com que deixem de funcionar, logo buscamos uma tecnologia que siga boas práticas e mantenha ao máximo a compatibilidade, minimizando os efeitos colaterais aos seus usuários. Este é apenas um dos fatores que contribui para aplicações mais resilientes.

Considerando o cenário atual, onde há inúmeros aplicativos e equipes, não existe uma área voltada exclusivamente ao desenvolvimento móvel e devido às demandas cotidianas nem sempre é possível priorizar a manutenção nos aplicativos, uma quantidade significativa de quebras que exigem ações por parte dos times tendem a gerar ciclos frequentes de manutenção e pouca evolução das soluções.

4.2.2 Poucas dependências de produção

A quantidade de dependências de produção das tecnologias é algo que de certa forma está ligado às quebras de compatibilidades, na maioria dos casos, quanto mais dependências necessárias mais vulneráveis ficamos às alterações externas.

4.2.3 Capacidade de teste

Testes de unidades auxiliam a cobrir o maior número de cenários possíveis, assegurar a qualidade das soluções e são uma barreira, tanto para diagnóstico de possíveis problemas durante o fluxo de desenvolvimento, quanto para assegurar a regra de negócio durante tarefas de manutenção, visto que qualquer quebra não intencional fará com que os testes falhem e forçará o desenvolvedor a reavaliar o cenário. Neste quesito a vasta maioria das tecnologias oferecem suporte por meio da instalação de uma simples dependência que adiciona as ferramentas de testes, tanto de unidade, quanto integração.

4.3 Recursos

Como recursos, listamos características que são importantes quando consideramos nossas perspectivas futuras para os aplicativos móveis e aonde queremos chegar.

Tabela 7 Critérios da categoria recursos

Recursos (pontuação máx.: 1.1 ou 7.01%)	Peso/Multiplicador
Animações fluídas	0.5
Lida bem com processos pesados	0.3
Economia de bateria	0.3

4.3.1 Animações fluídas

Animações fazem toda a diferença para uma interface e experiência do usuário de qualidade, avaliamos aqui não a possibilidade de realizar e/ou incluir animações, isto é algo que qualquer tecnologia de UI oferece, mas sim tê-las fluídas juntamente com toda

a experiência de uso do aplicativo, como a simples ação de realizar scroll em uma lista sem passar a sensação de travamento ao usuário. Poder contar com animações no estilo *Lottie* sem dúvidas é um diferencial.

4.3.2 Lidar bem com cálculos/processos que exigem alto processamento

Considerando o cenário atual e necessidades da maioria de nossos aplicativos isto não é algo fundamental, dessa forma atribuímos um peso inferior, embora consideremos como interessante caso a tecnologia ofereça pois se necessário não precisamos abandoná-la e migrar para uma abordagem integralmente nativa, fugindo do princípio *cross-platform*.

4.3.3 Economia de bateria

Se pudermos economizar energia do dispositivo de nossos usuários, faremos. Indiretamente isto contribui para a experiência e faz com que nossos aplicativos sejam utilizados sem preocupações por mais tempo.

4.4 Experiência do usuário

Nesta categoria listamos os critérios técnicos que possibilitam ao desenvolvedor criar ótima usabilidade para as interfaces.

Tabela 8 Critérios da categoria experiência do usuário

Experiência do usuário (pontuação máx.: 1.15 ou 7,32%)	Peso/Multiplicador
Componentes com interface nativa	0.4
Altamente customizável	0.75

4.4.1 Componentes com interface nativa

Temos perspectivas de um *design system* exclusivo, eliminando um layout baseado integralmente nas interfaces nativas, porém a possibilidade de utilizar determinado componente com o mesmo comportamento e interface do nativo é algo interessante para questões de produtividade, principalmente quando o *design system* não contemplar uma situação específica.

4.4.2 Altamente customizável

Com base em nossas perspectivas futuras de arquitetura e tecnologia, considerando também os esforços de nosso time de UX, a capacidade de customização da tecnologia é de extrema importância para possibilitar a criação de componentes e/ou temas baseados no *design system*, trazendo uma identidade única proprietária aos aplicativos.

4.5 Longo prazo

Se analisada isoladamente, sem as demais categorias esta acaba por induzir as escolhas com base apenas aos interesses da comunidade, o que por si só não necessariamente representa qualidade ou adequação ao nosso cenário, porém não podemos negligenciar a análise deste fator, contribuindo para uma visão mais realista de cada tecnologia. Uma comunidade expressiva e grandes empresas utilizando normalmente fornecem dados quanto à sua expectativa de vida, evolução e taxa de resolução de problemas. Estes dados auxiliam no entendimento e identificação de tendências a longo prazo.

Considerando nosso cenário tecnológico de *front-ends* onde possuímos ainda aplicações *AngularJS* com 3 ~ 5 anos e que começaram a ser reescritas para *Angular 2+* após aproximadamente três anos, ocorrendo por volta de nov/2017 *à* jan/2018 (dados dos repositórios *angular-components* e *primeng-theme*), vamos considerar de 3 ~ 5 anos como um período ideal para longo prazo.

Tabela 9 Critérios da categoria longo prazo

Longo prazo (pontuação máx.: 1.4 ou 8,92%)	**Peso/Multiplicador**
Tamanho da comunidade	0.5
Comunidade ativa (*Stack Overflow*, etc)	0.4
Adoção por grandes empresas	0.5

4.5.1 Tamanho da comunidade

Avaliamos a quantidade de *stars*, *forums* e outros materiais na internet sobre as tecnologias. Como neste caso todas são *open source* e estão hospedadas no *GitHub*, a forma mais tradicional de medir a comunidade é baseada em *stars* e *forks* para a quantidade de colaborações, embora *forks* nem sempre significam que as alterações foram devolvidas ao repositório original por meio de *pull request*.

4.5.2 Comunidade ativa

A comunidade ativa pode ser medida com base em eventos sendo realizados, *lives*, tutoriais, movimentações em tópicos do *Stack Overflow* e outros. Com estes dados conseguimos identificar tendências da comunidade em auxiliar na resolução de problemas e compartilhar conhecimento.

4.5.3 Adoção por grandes empresas

Podemos pontuar também a confiança que grandes empresas possuem na tecnologia em questão, visto que desenvolveram suas soluções com a mesma. Outro fator é que essas empresas tendem a dedicar recursos humanos para trabalhar na evolução da tecnologia ou financeiros por meio de patrocínio às iniciativas *open source*, tal ação as beneficia diretamente visto que conforme a comunidade e a adoção crescem os riscos diminuem e a tecnologia evolui juntamente com sua expectativa de vida.

4.6 Fator humano/gestão

Fatores relacionados à gestão dos recursos humanos e à experiência do desenvolvedor com a tecnologia são mensurados nesta categoria.

Tabela 10 Critérios da categoria fator humano/gestão

Fator humano/gestão (pontuação máx.: 1.9 ou 12,10%)	**Peso/Multiplicador**
Curva de aprendizado	0.5
Design pattern nativo	0.6
Mão de obra	0.8

4.6.1 Curva de aprendizado

Considerando que possuímos desenvolvedores incluídos em diversos contextos e níveis na carreira, uma tecnologia com baixa curva de aprendizado auxiliará para que eles consigam se contextualizar rapidamente, associando as práticas e paradigmas já conhecidos para apresentar resultados em menor tempo.

4.6.2 Design pattern nativo

Buscamos uma tecnologia que possua nativamente um *design pattern*, já que tende a forçar as implementações a seguirem boas práticas. Entendemos que isto pode acabar por tornar o fluxo de desenvolvimento mais restritivo onde as etapas do *framework* devem ser respeitadas, mas em contrapartida o custo inicial das abstrações necessárias diminui e a produtividade tende a aumentar. Os projetos com a mesma tecnologia passam a ter códigos mais homogêneos e a contextualização de um novo desenvolvedor com conhecimento do *framework,* independente de time e aplicativo que atua passa a exigir menos tempo.

4.6.3 Mão de obra

Este critério é um dos que contribuiu para que nossa análise descarte tecnologias integralmente nativas e nos leva a considerar que *cross-platform* é mais interessante para o cenário atual. O mercado de trabalho de tecnologias nativas é extremamente restrito, exige alto investimento e considerando que na maioria dos casos será necessário um desenvolvedor para cada plataforma (iOS e Android), o investimento simplesmente dobra.

Com uma curva de aprendizado menor os nossos desenvolvedores conseguirão absorver as características da tecnologia rapidamente. Valorizamos nossa mão de obra atual e buscamos qualificá-la para operar a tecnologia escolhida, seja através de eventos, como *hands-on* por exemplo ou cursos. Para mensurar a necessidade de contratações buscamos entender as tendências do mercado de trabalho.

4.7 Outros

Aqui elencamos todos os critérios que possuem alta importância e impactam de forma direta ou não em outros categorias. Por exemplo, uma ótima documentação oficial contribui para o fator humano, diminuindo a curva de aprendizado.

Tabela 11 Critérios da categoria outros

Outros (pontuação máx.: 2.55 ou 16.24%)	**Peso/Multiplicador**
Ótima documentação oficial	0.8
Fontes de conhecimentos (documentações, vídeos e cursos)	0.75
Baixo risco de bloqueio por parte da Apple ou Google	0.5
Maturidade da solução	0.5

4.7.1 Ótima documentação oficial

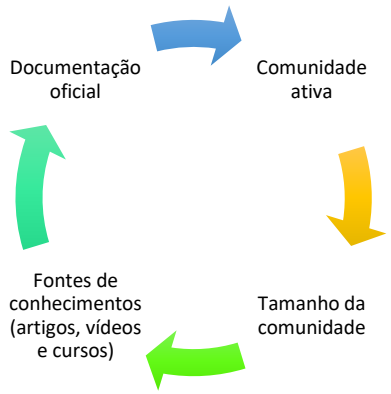
Uma ótima documentação na página oficial é fundamental para qualquer tecnologia. Valorizamos a organização, informações atualizadas, quantidade de exemplos em códigos, vídeos e ilustrações detalhadas sobre a arquitetura adotada. A documentação oficial deve ser o guia para qualquer aprendizado, partindo do absoluto zero às questões mais complexas e auxiliando na resolução de problemas conhecidos.

4.7.2 Fontes de conhecimento (artigos, vídeos e cursos)

As fontes não necessariamente oficiais que contribuem para disseminação de conhecimento e construção de comunidades, seja através da produção de artigos, vídeos tutoriais ou cursos oferecidos em plataformas online e/ou instituições de ensino são importantes para redução da curva de aprendizado.

Neste critério elencamos a seguinte questão: “sempre que necessário, é possível encontrar conteúdo didático de forma fácil?”

Figura 5 Critérios interligados



4.7.3 Baixo risco de bloqueio nas lojas

Consideramos neste critério o risco de bloqueio nas lojas de aplicativos para cada tecnologia analisada levando em consideração problemas comuns relatados pela comunidade e enfrentados durante o gerenciamento de nossos aplicativos móveis ao longo do tempo.

Os bloqueios que costumamos enfrentar estão ligados muitas vezes às dependências não oficiais que acabam sendo utilizadas para suprir algumas deficiências da tecnologia, que não as implementa em seu *core*. Situações de bloqueios devido a implementações internas dessas dependências, como SDKs para exibição de anúncios ou desatualizados são comuns.

4.7.4 Maturidade da solução

Para maturidade da solução consideramos alguns critérios da categoria 4.5 e outras particularidades que podem guiar para melhor entendimento, como quantidade de abstrações já implementadas, ferramentas disponíveis, recursos adicionais da tecnologia e se possui um status considerado pronto para produção.

5 Tecnologias analisadas

Como citado anteriormente, estão sendo analisadas as tecnologias *cross-platform open source*, sendo *React Native*, *Ionic framework*, *Flutter* e *Kotlin* em seu recurso *Kotlin Multiplatform Mobile*. Elencamos critérios técnicos, sendo que estes levam em consideração inúmeros fatores, como a maturidade da mobilidade na Senior e nossas perspectivas como departamento de arquitetura para prover soluções aos times de desenvolvimento e ferramentas de produtividade.

Em uma análise prévia a esta chegamos a realizar a filtragem das tecnologias que seriam avaliadas, incluindo *Xamarin*, que acabou não fazendo tanto sentido para nós devido a distância com nossa *stack* tecnológica atual. *Xamarin* baseia seu desenvolvimento em .NET e C#. Entendemos ainda que desejamos uma abordagem arquitetural mais próxima do nativo e naturalmente buscamos tecnologias capazes de ofertar isto.

5.1 Flutter

O *Flutter* foi lançado em maio de 2017, teve sua versão 1.0 considerada pronta para produção em 04 de dezembro de 2018 e atualmente se encontra na versão 1.22.4. A tecnologia permite o desenvolvimento de interfaces, foi criada pelo *Google* e oferece suporte para Android, iOS, Windows, Mac, Linux, Google Fuchsia e web. *Flutter* está baseado em *Dart*, após o processo de *build* todo o código é compilado diretamente para nativo.

5.2 Ionic Framework 5

Ionic Framework é extremamente conhecido, sendo lançado em 2013, originalmente com *AngularJS*, se encontra atualmente na versão 5.5.1. *Ionic* permite a utilização de *Angular* (2+), *React* ou *VueJS.* Na versão 5.0 lançada 11 de fevereiro de 2020 o *Ionic* citou a capacidade de entregar 60 FPS e introduziu um a tecnologia *capacitor* para interagir com recursos nativos, uma "substituição moderna" ao *Cordova.* Atualmente é possível criar aplicativos para iOS, Android, PWAs e Electron (desktop).

5.3 Kotlin Multiplatform Mobile

Estamos avaliando a possibilidade *cross-platform* e não o *Kotlin* em si. *Kotlin Multiplatform Mobile* utiliza as capacidades multiplataformas do *Kotlin* e inclui várias ferramentas e recursos projetados para tornar a experiência de ponta a ponta de construir aplicativos multiplataforma móvel o mais agradável e eficiente possível. Entrou em estágio *alpha* há poucos meses, não sendo ainda considerado pronto para produção. (JETBRAINS, 2020)

5.4 React Native

O *React Native* é uma biblioteca *Javascript* para desenvolver aplicativos para Android e iOS de forma nativa. Foi criado pelo *Facebook* e lançado em 26 de março de 2015, atualmente se encontra na versão 0.63.3. *React Native* se baseia integralmente em *Javascript* e é possível utilizar *Typescript* para o desenvolvimento. No dispositivo é executado um processo em segundo plano que interpreta o *Javascript* escrito e se comunica com a plataforma nativa através de uma *bridge*.

6 Resultados obtidos

Os resultados obtidos foram segmentados por características para melhorar a comparação.

6.1 Por critério

6.1.1 Produtividade

Tabela 12 Resultados da categoria produtividade

Critério	Flutter	Ionic	Kotlin MM	React Native
Cross-platform	Atende/Alto	Atende/Alto	Atende parcialmente/ Médio	Atende/alto
CI/CD de baixo à médio esforço	Atende/Alto	Atende/Alto	Atende/Alto	Atende/Alto
Multiplataforma (mobile/web/desktop)	Atende parcialmente/ Médio	Atende parcialmente/ Médio	Atende parcialmente/ Médio	Atende/Alto
Hot reload	Atende/Alto	Atende/Alto	Atende/Alto	Atende/Alto
Ferramentas para debug	Atende/Alto	Atende/Alto	Atende/Alto	Atende/Alto
Integração com IDEs	Atende/Alto	Não atende/Baixo	Atende/Alto	Atende/Alto
Total no critério	**98.53%**	**86.76%**	**78.92%**	**100%**

Cross-platform: Flutter, Ionic e React Native satisfazem o critério integralmente. Kotlin MM por ser uma solução recente e ainda em *alpha* (JETBRAINS, 2020) acabou sendo pontuado como parcial, pois embora atenda o princípio *cross-platform* não é maduro o suficiente para medir a eficácia em produção.

CI/CD de baixo à médio esforço: Ionic e React Native são Javascript e possuem maior facilidade para integração ao CI. Flutter e Kotlin MM ficam semelhantes com um custo mediano.

Multiplataforma (mobile/web/desktop): Flutter e Kotlin MM foram marcados como parciais devido a estes recursos estarem em estágio de desenvolvimento.

Os demais critérios, todas as tecnologias possuem, sendo *hot reload*, ferramentas para *debug* e integração com IDE. No caso de Kotlin MM o suporte se baseia no Android Studio. Embora desejemos contar com o Visual Studio Code, atribuímos a nota alta pelo fato de conter integração completa com uma IDE.

6.1.2 Resiliência

Tabela 13 Resultados da categoria resiliência

Critério	Flutter	Ionic	Kotlin MM	React Native
Poucas quebras de compatibilidade	Atende/Alto	Atende/Alto	Atende/Alto	Atende parcialmente/Médio

Poucas dependências de produção	Atende/Alto	Atende parcialmente/Médio	Atende/Alto	Não atende/Baixo
Capacidade de testes	Atende/Alto	Atende/Alto	Atende/Alto	Atende/Alto
Total no critério	**100%**	**80%**	**100%**	**45%**

Poucas quebras de compatibilidades: Flutter apresenta três quebras desde a v1.0.0 e se trata de alterações menores sem muito impacto às aplicações. Ionic 5 é relativamente recente e ainda não possui quebras, visto que a mais recente se trata da mudança da versão 4 para a 5. A partir da quinta versão por enquanto recebeu apenas correções de bugs. Kotlin MM é bem recente e aparentemente tem recebido apenas correções, sem quebras. React Native recebeu nota parcial, pois conforme seu *changelog* quebras na API são constantes e cada inclui grande quantidade de alterações. RN tem seguido o padrão de incluir quebras em *minors* desde a v0.49.0 de 03 de outubro de 2017.

Poucas dependências de produção: Flutter contempla vários recursos em seu *core* e devido a isto necessita de menos dependências, sendo elas dedicadas a funcionalidades e/ou estilos extras. Ionic 5 em um aplicativo de abas (*tabs*) possui 14 dependências, sendo avaliado como parcialmente pois se encontra em meio as três do Flutter e 29 do React Native. Por padrão são injetadas três dependências, o SDK, *cupertino_icons* e *english_words*. Ionic Kotlin MM injeta nove dependências, sendo três para testes. O React Native possui 29 dependências de produção, embora em um projeto gerado injete apenas *react* e *react-native*, porém é necessário considerar as dependências associadas a cada um desses projetos.

6.1.3 Recursos

Tabela 14 Resultados obtidos na categoria recursos

Critério	Flutter	Ionic	Kotlin MM	React Native
Animações fluídas	Atende/Alto	Atende/Alto	Atende/Alto	Atende parcialmente/Médio
Lida bem com processos pesados	Atende/Alto	Não atende/Baixo	Atende/Alto	Não atende/Baixo
Economia de bateria	Atende parcialmente/Médio	Atende parcialmente/Médio	Atende/Alto	Atende parcialmente/Médio
Total no critério	**86.36%**	**59.09%**	**100%**	**36.36%**

Com poucas semanas para produção do relatório, não foi possível realizar um benchmark completo de cada uma das tecnologias relacionados às animações fluídas e uso intensivo de processador, porém é conhecido que Ionic e React Native possuem deficiências nos quesitos de lidar com tarefas que exigem alto poder de processamento, animações fluídas e consequentemente realizar melhor gerenciamento de energia. O desenvolvimento nativo sempre se sairá melhor nesta categoria, Kotlin simplesmente brilha em todos os critérios. Flutter entrega 60FPS ou até mais (de acordo com a capacidade do dispositivo) ao usuário,

tem bom gerenciamento de CPU e memória sendo ideal para o cenário atual. Ionic 5 em sua página oficial promete entregar 60 FPS para animações mais tradicionais, porém cita que para algo mais elaborado Flutter se sai melhor, conforme o próprio comparativo realizado pela equipe do Ionic (IONIC FRAMEWORK, 2020). A página cita ainda que a versão 5 recebeu melhorias que a tornam duas vezes mais rápida que a anterior.

6.1.4 Experiência do usuário

Tabela 15 Resultados da categoria experiência do usuário

Critério	Flutter	Ionic	Kotlin MM	React Native
Componentes com interface nativa	Atende/Alto	Atende/Alto	Atende/Alto	Atende/Alto
Altamente customizável	Atende/Alto	Atende/Alto	Atende/Alto	Atende/Alto
Total no critério	**100%**	**100%**	**100%**	**100%**

6.1.5 Longo prazo

Tabela 16 Resultados da categoria longo prazo

Critério	Flutter	Ionic	Kotlin MM	React Native
Tamanho da comunidade	Atende/Alto	Atende parcialmente/Médio	Não atende/Baixo	Atende/Alto
Comunidade ativa (Stack Overflow, etc)	Atende parcialmente/Médio	Atende parcialmente/Médio	Atende parcialmente/Médio	Atende/Alto
Adoção por grandes empresas	Atende/Alto	Atende parcialmente/Médio	Atende/Alto	Atende/Alto
Total no critério	**85.71%**	**50%**	**50%**	**100%**

Tamanho da comunidade: Kotlin MM é relativamente novo para uma comunidade expressiva, considerando apenas o Kotlin em si o tamanho da comunidade não é expressivo se comparado ao React Native. Ionic tem 42.2k *stars* no GitHub e 13.3k *forks.* O repositório *kotlin-native* possui 6.9k *stars* no GitHub. Este não é um fator que representa a qualidade da tecnologia, mas sim a capacidade de obter suporte com facilidade através de pesquisas ou *fórums*. React Native possui 91.4k *stars* e 20.2k *forks* no GitHub, Flutter fica com 107k *stars* e 15k *forks*. A comunidade do Flutter tem crescido de forma rápida com o lançamento da versão 1.0.0.

Comunidade ativa: se tratando de *fórums*, produção de materiais ou tópicos no Stack Overflow (site em inglês), o React Native sagra-se vencedor, existem inúmeros materiais na internet, ensinando sobre. Flutter atualmente não fica muito atrás, visto que é fácil encontrar respostas, existem também diversos tópicos no Stack Overflow e materiais na internet. Neste quesito a similaridade do React Native com React é um fator que auxilia

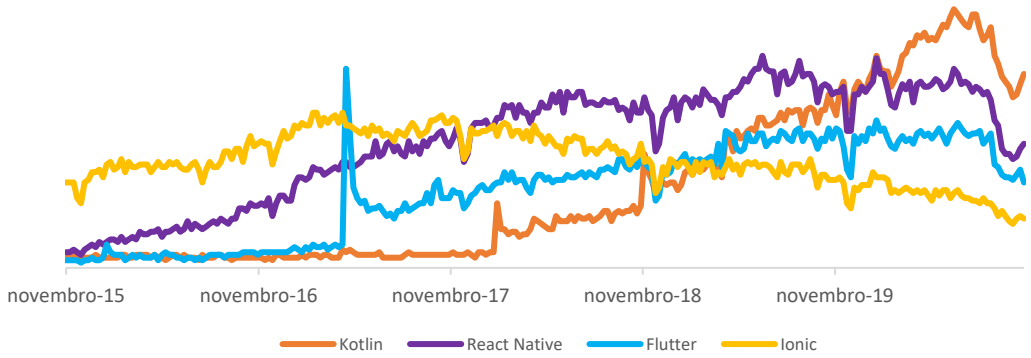
muito, visto que as regras de negócio são implementadas da mesma maneira e qualquer desenvolver com a base (React) é capaz de fornecer auxílio.

Ionic tem uma particularidade devido suas versões major ter diferenças entre si, isto quer dizer que o Ionic 1 é totalmente diferente do 5. Embora seja possível encontrar tópicos sobre o assunto, a frequência é baixa e em muitos casos se referem as versões anteriores, como a 2.

Kotlin possui discussões sobre o assunto no Stack Overflow, porém muitos tópicos tendem a não receber o devido engajamento.

Adoção por grandes empresas: Flutter, Kotlin e React Native têm *showcases* significativos. React Native é utilizado nos apps Facebook, Instagram, Skype, Discord, Tesla, Salesforce, Pinterest e outros. Flutter tem Google Assistant, Google Ads, Alibaba, Ebay, BMW, NuBank, Philips Hue, MGM Resorts e outros. Kotlin conta com Autodesk, VMWare e outros. Ionic possui um showcase mais expressivo se tratando de sua versão empresarial, como estamos avaliando a *open source* acabamos por lhe atribuir uma nota mediana.

Figura 6 Tendências de pesquisas globais no Google nos últimos cinco anos (GOOGLE, 2020)



6.1.6 Fator humano/Gestão

Tabela 17 Resultados da categoria fator humano/gestão

Critério	Flutter	Ionic	Kotlin MM	React Native
Curva de aprendizado	Atende/Alto	Atende/Alto	Atende parcialmente/Médio	Atende/Alto
Design pattern nativo	Atende/Alto	Atende/Alto	Atende/Alto	Atende parcialmente/Médio
Mão de obra	Atende parcialmente/Médio	Atende/Alto	Atende parcialmente/Médio	Atende/Alto
Total no critério	**78.95%**	**100%**	**65.79%**	**84.21%**

Curva de aprendizado: Flutter utiliza Dart como linguagem de programação e apresenta uma curva de aprendizado relativamente baixa, tanto para desenvolvedores com base Java, quanto para quem tem vem da web. Dart tem similaridades com Javascript e Java, outro fator que contribui para uma curva menor é a ótima documentação.

Ionic apresenta ótimos resultados na categoria por ter baixa curva de aprendizado, visto que um projeto Ionic pode ser gerado escolhendo entre *Angular*, *React* ou *VueJS*, dessa forma existem mais chances de utilizar uma sintaxe familiar, em nosso caso a consideração seria por Angular.

Kotlin MM tem semelhanças com Java e possui boa documentação, a curva de aprendizado tende a ser maior se comparada ao Flutter. React Native possui baixa curva de aprendizado, considerando os fundamentos básicos de Javascript ou uma base sólida em React, a curva é quase nula.

***Design pattern* nativo:** Flutter possui um desenvolvimento "guiado" no sentido de que inúmeros *designs patterns* já estão incluídos na linguagem. Isto diminui as possibilidades de que um desenvolvedor com pouco conhecimento realize implementações que dificultam a manutenibilidade do código. O custo inicial para construção de abstrações também é relativamente baixo, pois a linguagem por si só já as contempla. Em Dart é como se tivéssemos apenas uma única maneira correta de criar as coisas.

Ionic, considerando o core em Angular também possui um *design pattern* nativo, sendo utilizada por padrão a arquitetura MVC e há definições específicas como módulos e serviços.

React Native tende a ser mais liberal neste sentido, cada aplicação pode ser construída de inúmeras formas de acordo com cada desenvolvedor, isto sem dúvidas é bom em muitos sentidos, porém visando uma padronização do *codebase* teríamos um investimento inicial na criação de abstrações relativamente alto.

Mão de obra: este critério é importante por considerar a capacidade de mão de obra para a tecnologia específica. Analisando sites como *Glassdoor* e buscando profissionais no *Linkedin*, React Native apresenta a maior capacidade de suprir às demandas. Ionic por sua versatilidade de poder ser utilizado com uma das três tecnologias (*Angular*, *React* ou *VueJS*) e da contratação de um desenvolvedor com conhecimento web apresenta ótimos resultados. Flutter contém um número expressivo de desenvolvedores com a competência cadastrada e existem mais empresas procurando por desenvolvedores Kotlin.

6.1.7 Outros

Tabela 18 Resultados da categoria outros

Critério	Flutter	Ionic	Kotlin MM	React Native
Ótima documentação oficial	Atende/Alto	Atende/Alto	Atende/Alto	Atende parcialmente/Médio
Fontes de conhecimentos (artigos, vídeos e cursos)	Atende/Alto	Atende parcialmente/Médio	Não atende/Baixo	Atende/Alto

Baixo risco de bloqueio por parte da Apple ou Google	Atende/Alto	Atende parcialmente/Médio	Atende/Alto	Atende/Alto
Maturidade da solução	Atende/Alto	Atende/Alto	Não atende/Baixo	Atende/Alto
Total no critério	**100%**	**75.49%**	**50.98%**	**84.31%**

Ótima documentação oficial: Flutter assim como todos os produtos do Google possui uma documentação impecável, extremamente organizada e detalhada com códigos, vídeos produzidos pela própria equipe de desenvolvimento, *playground* e detalhes das APIs. Ionic de forma semelhante ao Flutter apresenta ótima documentação com exemplos em código e demonstração no próprio site. Kotlin possui uma ótima documentação também com inúmeros exemplos em código. React Native embora possua uma documentação relativamente boa e com exemplos, é desorganizada, sendo fácil se perder entre as seções e em algumas ocasiões perceber ausência de mais detalhes.

Fontes de conhecimentos (artigos, vídeos e cursos): neste critério Flutter inicialmente havia sido classificado como parcial, porém com mais pesquisas, tanto em plataformas de cursos, quanto artigos e vídeos, existe uma vasta diversidade de conteúdo sendo produzida com uma frequência absurda. É possível localizar guias extremamente completos para diversas situações, reclassificamos para nota alta. Embora seja comum encontrar cursos e tutoriais para Ionic, a grande maioria se refere as versões mais antigas, sendo estas diferentes da mais recente (versão 5). Kotlin possui menos recursos para aprendizado disponíveis, necessitando se basear mais na documentação oficial, tratando ainda do *cross-platform* do Kotlin por ser muito jovem os materiais diminuem drasticamente.

Baixo risco de bloqueio por parte da Apple ou Google: Flutter e React Native possuem riscos muito baixos de ser restritos pelas lojas de aplicativos, o React Native especialmente possui o recurso de atualização *on the air* para distribuir atualizações sem passar pelas lojas de aplicativos, isto pode ser considerado abusivo pelas plataformas e acabar por bloquear o aplicativo, porém é uma possibilidade bem remota. Ionic apresenta um risco considerável, especificamente na App Store, devido a políticas da Apple (item 4.2 das *guidelines* da loja) não ser permitido utilização de *webviews*, o que dependendo da forma de construção do aplicativo pode acabar fazendo com que seja rejeitado (APPLE). Kotlin por ser nativo apresenta risco muito baixo.

Maturidade da solução: Flutter existe há três anos, é mantido pelo Google e Dart é utilizado em diversos projetos dentro da empresa, se encontra bem maduro, tem evoluído muito desde o lançamento de sua versão 1.0 e existem grandes aplicativos o utilizando. Considerando o tempo em que a versão 4 do Ionic se encontra no mercado e sendo esta semelhante com a 5 o *framework* apresenta boa maturidade. React Native é a tecnologia mais madura de todas, sendo utilizada em diversos aplicativos do Facebook e por várias gigantes, tendo cinco anos. Kotlin é relativamente novo no cenário *cross-platform* não sendo maduro o suficiente neste quesito. Importante ressaltar que estamos avaliando seu recurso *cross-platform* e não Kotlin diretamente.

6.2 Comparativo

Calculando os pesos e percentuais para as categorias obtemos dados interessantes quanto à adequação das tecnologias aos critérios propostos.

Tabela 19 Atendimento aos critérios

	Flutter	Ionic	Kotlin	React Native
Atende aos critérios em	**94.75%**	**81.21%**	**76.59%**	**82.32%**

Figura 7 Gráfico radar comparando as tecnologias nas categorias

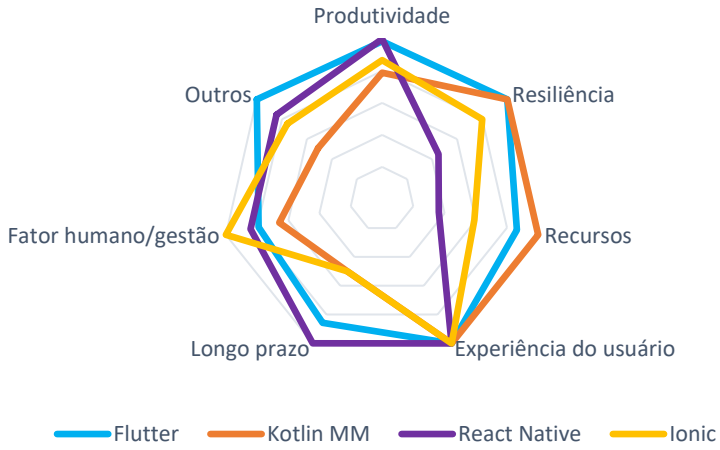
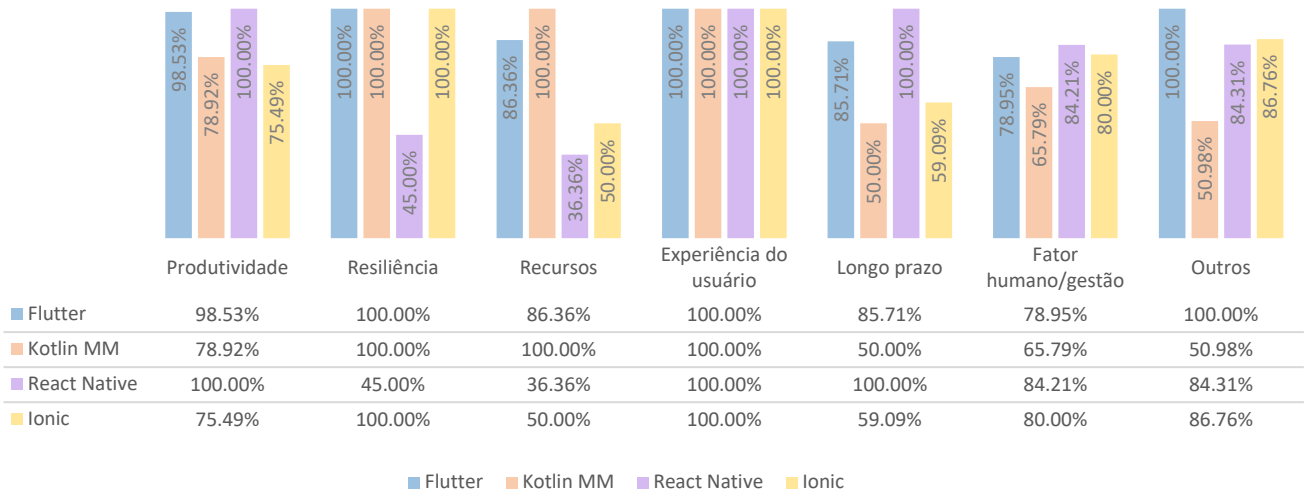


Figura 8 Gráfico de barras comparativo das tecnologias



	Produtividade	Resiliência	Recursos	Experiência do usuário	Longo prazo	Fator humano/gestão	Outros
Flutter	98.53%	100.00%	86.36%	100.00%	85.71%	78.95%	100.00%
Kotlin MM	78.92%	100.00%	100.00%	100.00%	50.00%	65.79%	50.98%
React Native	100.00%	45.00%	36.36%	100.00%	100.00%	84.21%	84.31%
Ionic	75.49%	100.00%	50.00%	100.00%	59.09%	80.00%	86.76%

7 Conclusão

Foi desafiador elaborar uma análise tão extensa, importante e complexa de tecnologias extremamente diferentes em poucas semanas. Consideramos inúmeros fatores, mas

sempre relacionando com questões como: “o que precisamos?”, “o que temos no momento?”, “aonde queremos chegar?”, “quais riscos são toleráveis?” e “o que estamos dispostos a abrir mão?”. Embora não haja tempo suficiente para uma produção ainda mais detalhada e com uma exploração maior das tecnologias, o departamento de Pesquisa e Arquitetura **recomenda Flutter como framework cross-platform oficial. Acreditamos que a tecnologia tem potencial de se adequar às necessidades atuais de nossos aplicativos móveis e suprir perspectivas futuras.** Analisando os resultados temos: Flutter em primeiro lugar com 94.75% de atendimento aos critérios propostos, seguido por React Native com 82.32%, Ionic com 81.21% e Kotlin com 76.59%. A expectativa é de que possamos ter nossas soluções mais recentes caminhando para o Flutter e em algum tempo contar com um rico *codebase* na tecnologia.

Considerando dados relacionados à comunidade do Flutter, temos o potencial de desenvolvê-la através de nosso engajamento como empresa, recomendamos fortemente o compartilhamento público das descobertas ao longo da jornada de implantação da tecnologia na empresa, visando fomentar a comunidade e contribuir ativamente a estas iniciativas *open source*. Todos os desenvolvedores possuem papel fundamental como evangelizadores dentro dos times.

Como na Senior atualmente há diversos aplicativos com especificidades próprias e não uma única aplicação que contempla todos os produtos, estamos cientes que podem existir cenários em que o desenvolvimento nativo seja uma necessidade, como quando a criação de widgets para iOS é um dos requisitos, Flutter e React Native ainda não oferecem suporte, porém na maioria dos cenários e ponderando os requisitos gerais, Flutter sem dúvidas é uma ótima escolha.

> “Para aplicativos de negócios habituais com pequenas animações e aparência brilhante, a tecnologia não importa em nada.” (INVERITA, 2020)

A escolha da tecnologia é apenas uma fração para que possamos alcançar o patamar desejado em aplicativos móveis. Este é um trabalho a ser realizado de forma conjunta com diversas áreas da empresa, além de que investimentos no desenvolvimento de abstrações e ferramentas de produtividade devem ser realizados.

8 Referências

1983 to today: a history of mobile apps. **The Guardian**, 2015. Disponivel em: <https://www.theguardian.com/media-network/2015/feb/13/history-mobile-apps-future-interactive-timeline>. Acesso em: 10 nov. 2020.

APPLE. App Store Review Guidelines. **App Store - Apple**. Disponivel em: <https://developer.apple.com/app-store/review/guidelines/#design>. Acesso em: 26 nov. 2020.

GOOGLE. Google Trends - Flutter x Ionic x Kotlin x React Native. **Google Trends**, 2020. Disponivel em: <https://trends.google.com.br/trends/explore?geo=BR&q=Flutter,React%20Native,Kotlin,Ionic>. Acesso em: 26 nov. 2020.

INVERITA. Flutter vs React Native vs Native: Deep Performance Comparison. **Medium**, 2020. Disponivel em: <https://medium.com/swlh/flutter-vs-react-native-vs-native-deep-performance-comparison-990b90c11433>. Acesso em: 21 nov. 2020.

IONIC FRAMEWORK. Ionic vs Flutter. **Ionic Framework**, 2020. Disponivel em: <https://ionicframework.com/resources/articles/ionic-vs-flutter-comparison-guide>. Acesso em: 26 nov. 2020.

JETBRAINS. Kotlin Multiplatform Mobile Goes Alpha. **JetBrains Blog**, 2020. Disponivel em: <https://blog.jetbrains.com/kotlin/2020/08/kotlin-multiplatform-mobile-goes-alpha/>. Acesso em: 21 nov. 2020.

KEETON, B. J. The 11 Best Code Editors for 2019. **Elegant Themes**, 2019. Disponivel em: <https://www.elegantthemes.com/blog/resources/best-code-editors>. Acesso em: 19 nov. 2020.

THE History of Mobile Apps. **InventionLand**, 2019. Disponivel em: <https://inventionland.com/inventing/the-history-of-mobile-apps/>. Acesso em: 16 nov. 2020.

Agradecimentos do autor

Aos integrantes dos times de desenvolvimento que forneceram dados relacionados aos aplicativos mantidos, realizaram demonstrações e auxiliaram de incontáveis formas, meu gerente Carlos Pereira por confiar este desafio de realizar a análise técnica e sugerir abordagens, a meu coordenador Ivan Alves por atender prontamente sempre que solicitado e auxiliar com dicas para a produção do material, meu time por lidar com as demandas em meio às minhas atividades relacionadas ao relatório, Adriner Andrade sugestões de melhorias na aplicação dos critérios, Kaianne Premoli pela revisão e sugestões e Ricardo Futata coordenador da equipe de arquitetura da Mega pelas informações relacionadas ao levantamento realizado anos antes na unidade.