



SUPER APP

Método

Visão VR
Ambições &
Expectativas

Cenário Atual

Análises

Pontos de
atenção

Arquitetura
sugerida

Síntese e
recomendações



Método de trabalho

MÉTODO

Etapas do trabalho



Como conduzimos os trabalhos

PROCESSO

COMPREENSÃO

VISÃO VR
ESTRATÉGIA &
TECNOLOGIA

IMERSÃO
NO CENÁRIO ATUAL
DE ARQUITETURA

REFINAMENTO
DO CENÁRIO ATUAL
TECNOLÓGICA DA VR

ANÁLISE

RECOMENDAÇÕES

FERRAMENTAS

INCEPTION

ENTREVISTAS

AVALIAÇÃO DE
JORNADAS

ANÁLISE DE
DOCUMENTAÇÃO

INSPEÇÕES
TÉCNICAS

PONTOS DE
CONTATO VR

OBSERVAÇÃO DE
CANAIS

ARQUITETURAS E MODELOS
DE REFERÊNCIAS

- Funcionalidades
- Arquitetura tecnológica
- Implementação
- Infraestrutura
- Segurança
- Time
- Parceiros

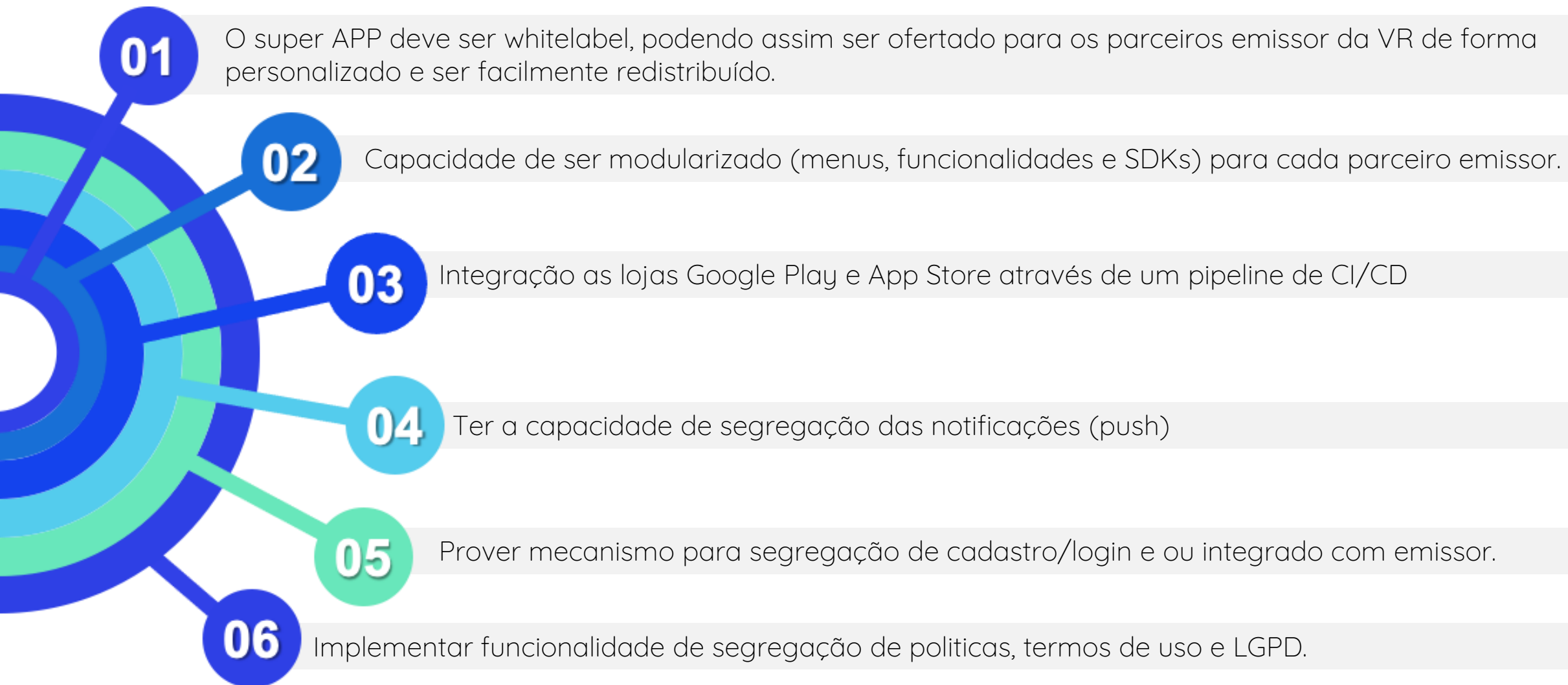
PERFIS

- ANDRÉ ALVES(CONSULTOR SENIOR)
- ANDRE XAVIER(CONSULTOR DEVOPS)
- ALEX GAMAS (ARQUITETO SOLUÇÕES)
- ANDERSON GAMA(ARQUITETO SOLUÇÕES)
- JOSE INACIO FERRARINI (ESPECIALISTA MOBILE)
- THALES SANTOS (ESPECIALISTA MOBILE)

DURAÇÃO: Aprox. 4 semanas

Visão VR

Drivers



Cenário atual

- Funcionalidades atuais
- Arranjo Arquitetural
- Implementação

Carrossel

Exibe todos os cartões vinculados ao CPF da conta do usuário, onde é possível navegar entre os cartões e obter com informações de saldo para cada um dos benefícios.



Saldo / Extrato / Gráfico de Utilização

Disponibiliza o saldo atual de cada cartão e o extrato de todas as transações realizadas. É possível ter detalhe das transações realizadas baseada em filtro de períodos. Também é disponibilizado um gráfico de utilização de cada benefício no período dos 30 dias.



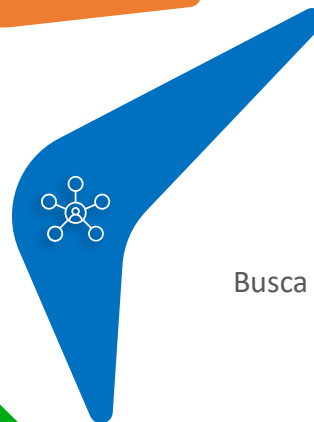
Cadastro do usuário (Onboarding)

Formulário para criação do usuário com preenchimento de dados pessoais, definição da senha e aceite no termo de uso e política de privacidade,. No primeiro acesso é realizada a vinculação dos cartões ao CPF da conta criada.



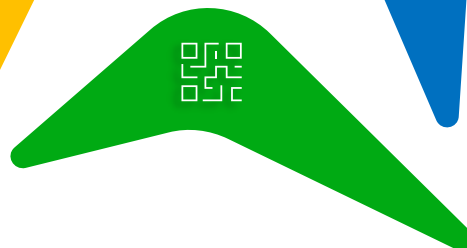
Busca da Rede Credenciada

Busca da rede por endereço, nome, CEP com possibilidade de favoritar no mapa os estabelecimentos de sua preferência.



Cartão Virtual / Pagamento via QR Code

Geração de cartões virtuais e pagamento via QR Code gerado a partir das maquininhas de cartões da Cielo ou via VR Pague.



Arranjo Arquitetural

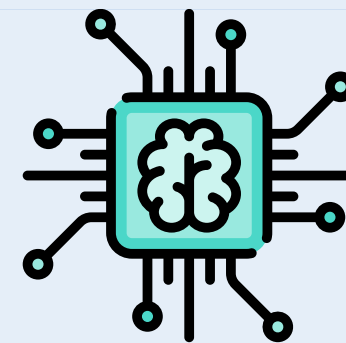
Arquitetura técnica (1/4)

❖ Aspectos técnicos gerais da Arquitetura atual

- ❑ Atualmente o aplicativo mobile VR & VC é desenvolvido em linguagem nativa iOS e Android e é baseado na arquitetura Model-View-Controller.
- ❑ A estrutura de serviços está dividida em dois ambientes,
 - Um ON-PREMISES na TIVIT, que suporta o barramento IBM, 4Time, a aplicação SNCORE e o banco de dados Oracle.
 - E um ambiente na CLOUD AWS, onde encontram-se os serviços de notification center, gestão e promoção, MC busca, gestão de identidade e RHSSO, sendo que esses serviços são orquestrados pelo OPENSIFT.

❖ Integrações

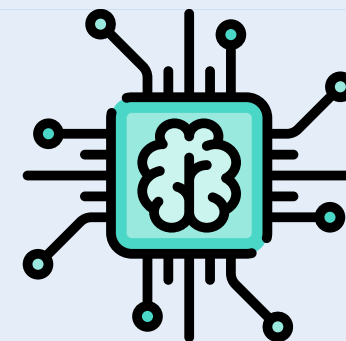
- ❑ A integração com os serviços é feita via API e conta com troca de arquivos JSON. Há uma camada de abstração via SDK para cada parceiro fornecedor integrado à plataforma.
- ❑ A integração com os serviços da VR é feito com três provedores de serviços diferentes, sendo eles o API Gateway da Sensidia, o Barramento IBM Websphere e 4Time WAS.
- ❑ Para comunicação via notificações é utilizado um broker de mensagens, através do Amazon MQ for Rabbit MQ, conectado ao serviço Notification Center responsável pelo envio de mensagens transacionais e relacionais.
- ❑ Integração com Plataforma Google Maps para disponibilizar o recurso de adicionar marcadores, utilizado para facilitar a trajetória para as redes credenciadas.



Arquitetura técnica (2/4)

❖ Emprego das Tecnologias

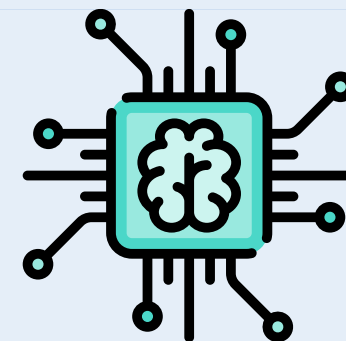
- ❑ Em termos tecnológicos o aplicativo mobile é desenvolvido utilizando-se de Objective C e Java como linguagens principais. Contudo, para a versão iOS, as novas funcionalidades já estão sendo desenvolvidas em Swift.
- ❑ O Aplicativo mobile não adota a estratégia de armazenamento em bancos de dados local, as poucas informações que são salvas no dispositivo, utiliza-se de um arquivo que é persistido em sessões (Preferences).
- ❑ Os Microservices que são consumidos pelo APP são desenvolvidos na linguagem Java com SpringBoot.. Estas aplicações são empacotadas em containers para execução em cluster Kurbenetes (gerido pelo Red Hat OpenShift).
- ❑ Os bancos de dados para os serviços de gestão de identidade e login do usuário utilizados são MYSQL e PostgreSQL que são contratados via serviços (RDS) no ambiente AS.
- ❑ É utilizado o ElasticSearch alocado na AWS como plataforma para armazenamento das *push notifications* geradas pelo Notification Center.



Arquitetura técnica (3/4)

❖ Segurança

- ❑ As informação de senha de acesso dos usuários são armazenadas de forma criptografada através de um algoritmo simétrico.
- ❑ O Aplicativo não utiliza-se de recursos de ofuscação para proporcionar redução do tamanho do APP e ajudar contra ataques de engenharia reversa, principalmente na versão Android.
- ❑ As informações de autenticação são trafegadas para o barramento IBM via Json utilizando-se protocolo https.
- ❑ Não existe atualmente um mecanismo de positivação via SMS ou checagem do CPF a partir de um birô de crédito.
- ❑ Como mecanismo de autenticação do APP é utilizada a solução da Red Hat Single Sign-On (RH-SSO), baseado em nos padrões OAuth 2.0.
- ❑ A segurança da comunicação entre as redes Cloud AWS e TIVIT é garantida por uma *VPN site-to-site*, estabelecida entre os ambientes. Este mecanismo garante que as duas redes possam se comunicar internamente através de acesso irrestrito entre si.



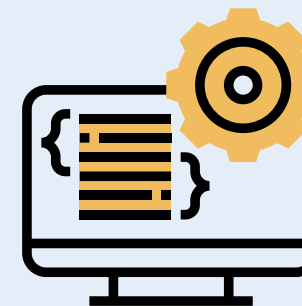
Implementação



Práticas do desenvolvimento

❖ CI / CD / Build & Deploy

- ❑ Há uma iniciativa em fase de testes de pipeline CI/CD tanto para iOS quanto Android utilizando ferramenta SaaS (Bitrise) na nuvem.
- ❑ Em paralelo, também está sendo analisada opção de criação de ambiente na própria empresa para a execução de pipeline CI/CD
- ❑ Apesar dessa iniciativa, atualmente, os processos relacionados a *Integration* e *Delivery* dos aplicativos (tanto iOS quanto Android) estão sendo disparados manualmente nas próprias máquinas dos desenvolvedores.
- ❑ Há um pipeline consolidado e específico para testes de integração. Esse pipeline é executado diariamente em horário específico (ou também sob demanda) e executa uma suíte de testes de sanidade escrita em Ruby.
 - Testes de regressão não fazem parte dessa pipeline executada diariamente, mas podem vir a ser executados sob demanda

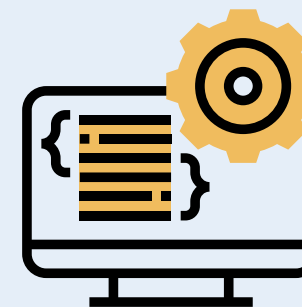




Práticas do desenvolvimento

❖ Testes

- ❑ A cobertura de testes unitários é ainda baixa para os apps (tanto iOS quanto Android).
- ❑ Para os testes de integração, já há um conjunto significativo de testes automatizados de integração, sendo executados diariamente. Os scripts desses testes são codificados em Ruby e estão em um repositório apartado do repositório do código fonte dos apps.
- ❑ Em relação aos testes sistêmicos, os critérios de aceite das histórias são especificados em linguagem Gherkin. Em um exemplo visto, os termos utilizados estão mais voltados para a parte tecnológica do que em relação ao negócio em si.
- ❑ BDD (Behaviour Driven Development) Apesar do uso de especificação dos critérios de aceite e testes de aceitação em Gherkin, as práticas e técnicas relacionadas ao BDD ainda estão sendo consideradas e não há consenso se devem ser adotadas na empresa
- ❑ Os critérios de aceite são a base para os testes funcionais de aceitação, porém a especificação dos cenários (com exemplos concretos de uso) só é feita quando há requisição do time de desenvolvimento.
- ❑ Observamos que há iniciativa de automação de testes sistêmicos funcionais com utilização de stack com Appium mas essa iniciativa está atualmente pausada.
- ❑ Atualmente os dispositivos onde são testados os apps se resumem aos dispositivos físicos do próprio time de testers. Há em curso um estudo de uso de serviços de teste com "device farms" utilizando providers como AWS e Google Firebase.
- ❑ É utilizado o plugin Zephyr Scale no JIRA que gerencia os ciclos e suítes de testes e, para testes automatizados (com execução via Jenkins), atualiza também status dos casos de testes definidos.
- ❑ Performance, para casos específicos (sob demanda), é construída suíte de testes de performance utilizando jMeter executados em ambiente de desenvolvimento e homologação.



Análises

- Tecnologia mobile cross plataforma
- Adequabilidade aos drivers

Tecnologia mobile cross plataforma



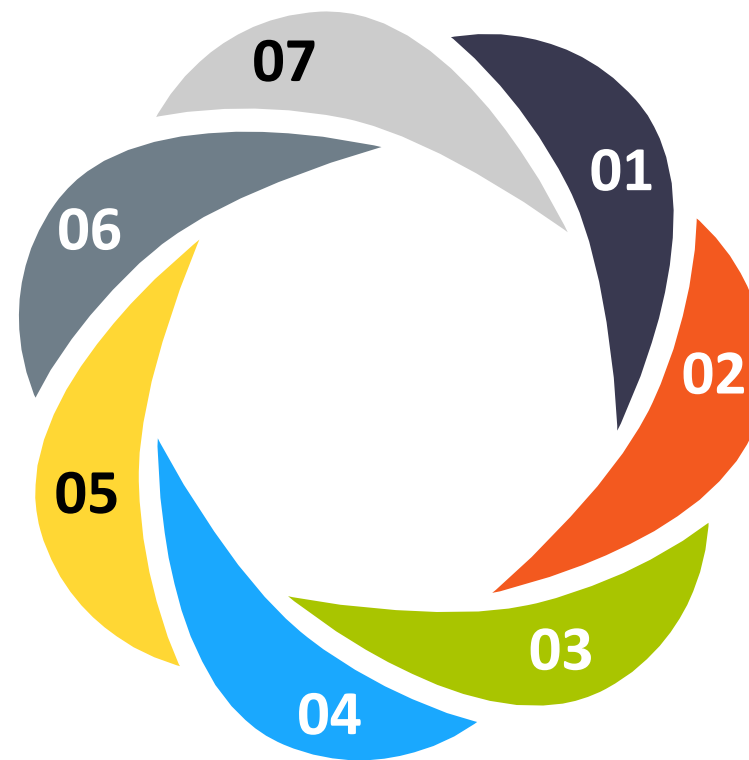
Trazemos a seguir uma análise das plataformas mobile híbridas, conhecidas como *cross-plataform*.

Foram avaliadas as três principais plataformas do mercado: Flutter, React Native e Kotlin Native.

O estudo foi baseado em critérios de avaliação considerados importantes para o cenário da VR e nos drivers que nos foram apresentados.

Em seguida foi feito um comparativo, entre a plataforma cross melhor avaliada com a solução nativa (iOS e Android).

Critérios de avaliação para as Plataformas Cross



Modularização

Capacidade de modularizar o desenvolvimento da plataforma para que seja possível escalar os times de desenvolvimento e facilitar o acoplamento de novas funcionalidades.

Mini Apps

Capacidade de incorporar mini apps, através de webview garantindo segurança, performance e usabilidade.

- Avaliaremos a possibilidade de limitação do controle de navegação e funções nativas do aparelho.

Restrições da loja

Risco da Apple ou Google restringir o aplicativo em de qualquer forma por causa do uso de uma plataforma subjacente.

- Desenvolvimento com Flutter UX pode não corresponder ao HIG da Apple (Diretrizes de Interface Humana).
- A possibilidade de atualizações Over The Air em React Native / Flutter podem ser um fator bloqueador na Apple.

Estabilidade da API/ferramenta

O quanto mudanças e evoluções na plataforma, sua API ou mudanças de ferramentas que exigem alterações no código desenvolvido.

Avaliaremos aqui:

- Mudanças na API base ou de dependência que a solução tornam incompatível com versões anteriores.
- Mudanças nos componentes nativos (OS) que quebram o comportamento interno de uma solução cross-plataforma.

Experiência no desenvolvimento

Fatores que contribuem para permitir que um desenvolvedor entregue de maneira produtiva o aplicativo móvel. Avaliaremos:

- Hot Reload
- Visibilidade do componente
- Ferramentas de Debugger
- Integração IDE
- Ferramentas de teste

Viabilidade de longo prazo

Confiança que o mantenedor da plataforma proverá suporte a longo prazo (cinco anos) e a probabilidade de que a comunidade seja capaz de apoiar o projeto se o mantenedor decidir não continuar.

Avaliaremos aqui:

- Adoção por grandes empresas
- Tamanho da comunidade (número de núcleos contribuidores, contribuidores externos, etc ...)

Especialização na plataforma

Um engenheiro deve ser capaz de escrever código móvel para Android e iOS sem preocupações com as especificidades da plataforma alvo.

O código deve ter a mesma aparência e se comportar no Android e iOS, com baixa ocorrência de falhas ou problemas específicos do sistema operacional.

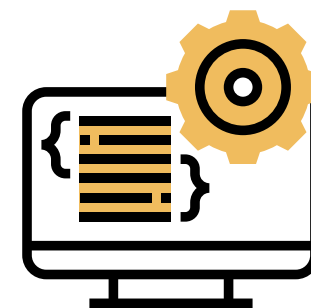
Avaliaremos o grau de abstração provido pela plataforma.

☐ IDE e Hot Reload

- O Flutter tem melhor escalabilidade, pois possui hot reload integrado que se adapta bem à base de código e é mais robusto do que o Kotlin Native.
- Flutter também apresenta ferramentas de depuração de código e IU que funcionam prontamente em ambas as plataformas, integradas ao IDE.
- Em contraste, o React Native requer ferramentas e configurações adicionais (como o uso de um navegador) para fazer a depuração funcionar, e o Kotlin Native requer as ferramentas da **plataforma subjacente**.

☐ Testes

- Flutter possui uma infraestrutura de teste integrada para testes de unidade, integração e ponta a ponta. Isso inclui a capacidade de exercitar telas e fluxos sem a necessidade de renderizar na tela e também um automatizador de teste que simula a entrada de um usuário.
- Por outro lado, o React Native requer dependências de terceiros para que os testes de integração e ponta a ponta funcionem, já o Kotlin Native tem suporte de teste de unidade muito limitado, o que exigiria que escrevêssemos um código personalizado para essas funcionalidades.

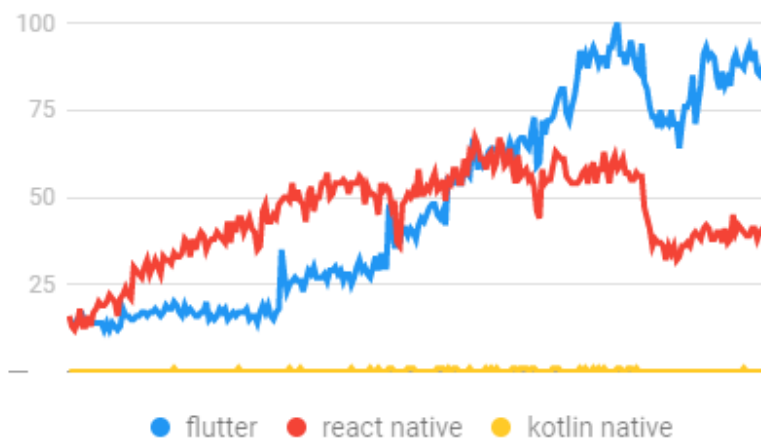


Plataforma	Reload at scala	Estabilidade Teste	Testability	Score
flutter	Médio	Médio - Alto	Alto	Médio - Alto
react native	Baixo	Médio	Alto	Médio
kotlin native	Médio	Baixo	Baixo	Médio - Baixo

❑ Viabilidade a longo prazo (Framework Cross Plataforma)

- A expectativa de vida do framework, e se isso se encaixa em uma visão de longo prazo. Para essa abordagem, estamos considerando 5 anos como longo prazo. Nós estamos também considerando a probabilidade de a comunidade absorver a manutenção custos no caso de o mantenedor original interromper o suporte para o framework.

Popularidade de busca da plataforma

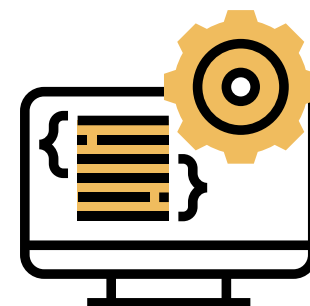


Tamanho da comunidade

Plataforma	# Stars/Forks	StackOverflow questions
flutter	126.000/18.300	97.696
react native	97.300/21.200	104.070
kotlin native	38.400/4.700	61.325

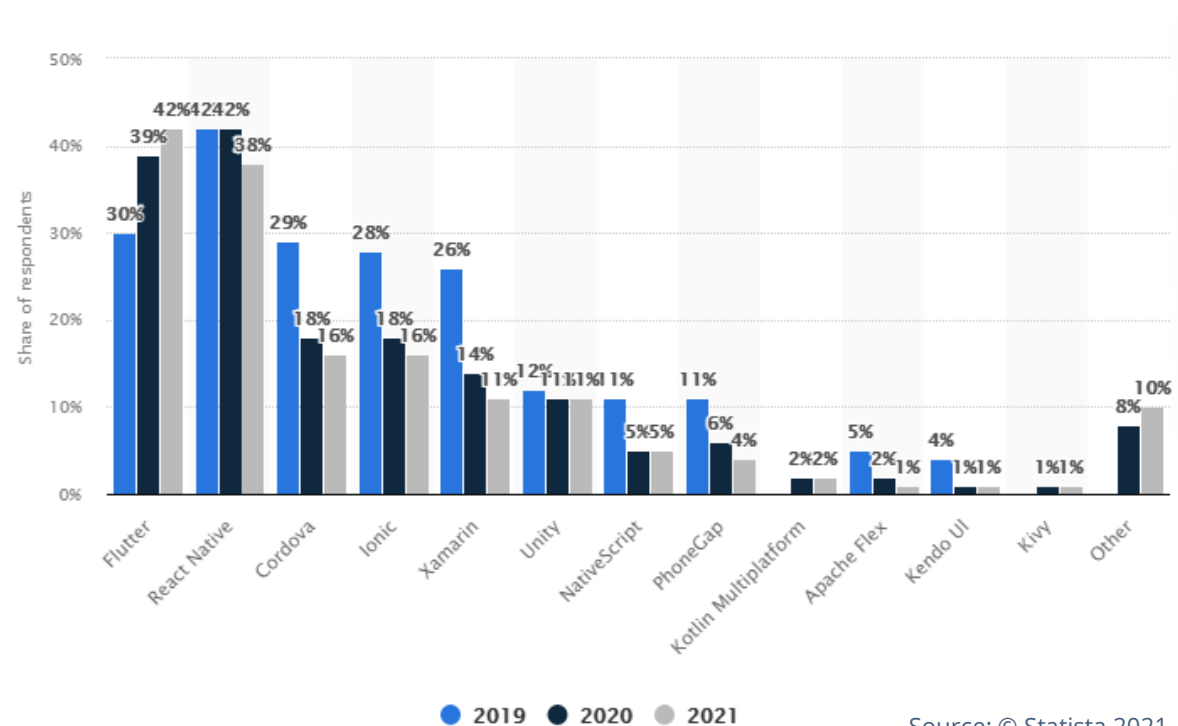
Adoção de grandes empresas

flutter – NuBank, Ifood, Banco Bs2, Globo
react native – Walmart, Airbnb, UbearEats
kotlin native – Netflix (Prodicle), Leroy Merlin

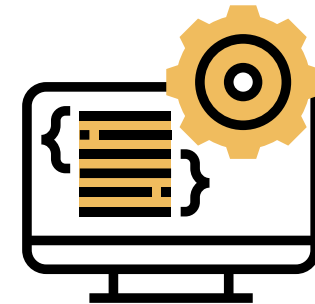


❑ Viabilidade a longo prazo (Framework Cross Plataform)

Flutter é a estrutura móvel multiplataforma mais popular usada por desenvolvedores globais, de acordo com uma pesquisa de desenvolvedores de 2021 pela Statista. De acordo com a pesquisa, 42% dos desenvolvedores de software usaram o Flutter. A segunda estrutura móvel de plataforma cruzada mais popular entre os desenvolvedores foi React Native.



Source: © Statista 2021



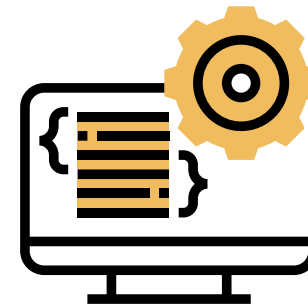
❑ Viabilidade a longo prazo (Framework Cross Plataform)

Stackoverflow 2020 Survey - Categoria: Frameworks, Libraries, and Tools

Em fevereiro de 2020, quase 65.000 desenvolvedores nos contaram como aprendem e sobem de nível, quais ferramentas estão usando e o que desejam.



Source: Stackoverflow



❑ Viabilidade a longo prazo (Framework Cross Plataform)

❑ Conclusão

A comunidade em torno do React Native é grande, madura e existe há mais tempo do que todas as outras alternativas. Embora o Flutter seja uma tecnologia mais recente, tem melhor documentação oficial e suporte do mantenedor do que o React Native e está crescendo em um ritmo acelerado.

Além disso, a comunidade no React Native está mais envolvida e mais propensa a continuar o desenvolvimento se o Facebook decidir retirar o suporte oficial. Com relação ao compromisso do mantenedor, vemos o Flutter e o React Native como tendo a mesma probabilidade de ter suporte contínuo.

O Flutter foi um projeto de vários anos dentro da Google antes do lançamento público inicial e que está sendo usado atualmente em mais de 10 projetos no Google, muitos dos que serão público e se estenderá por milhões e dezenas de milhões de usuários.

Em 2021 nota-se um crescimento ainda maior do Flutter pela comunidade. O SDK Flutter se posiciona como um das mais desejada pelos desenvolvedores para construção de aplicativos mobile cross plataforma.

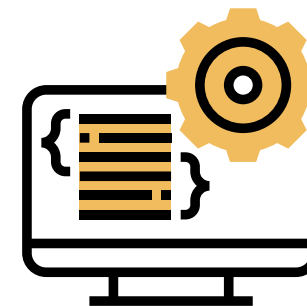


❑ Sem Necessidade de Especialista na Plataforma

O risco de exigir um código específico da plataforma é proporcional à proximidade da plataforma do sistema operacional subjacente. Quanto mais longe, maior a abstração e menos provável que o código de plataforma cross.

O Flutter oferece abstrações mais integradas (como navegação) que, de outra forma, exigiriam dependências externas para serem atendidas. No Kotlin Native, não há abstração de IU fornecida e, como tal, teríamos que escrever nossa própria. Isso aumenta a probabilidade de vazar o funcionamento interno das plataformas nativas

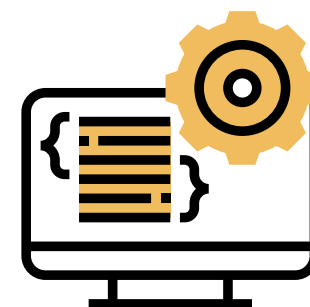
Plataforma	Risco de ter uma única plataforma de código
flutter	baixo
reactive native	médio
kotlin native	alto



❑ Estabilidade da API / Plataforma

Baseado no histórico de grandes mudanças na plataforma, levando em consideração a API, as ferramentas disponíveis e o ambiente de desenvolvimento. Também consideramos as mudanças significativas na principal dependência exigida

O React Native tem dependências em ordem de magnitude a mais do que as outras alternativas e, como tal, é muito mais vulnerável a mudanças significativas nessas áreas. Tanto o Kotlin Native quanto o Flutter têm APIs estáveis, enquanto o Kotlin Native tem uma área de superfície muito menor

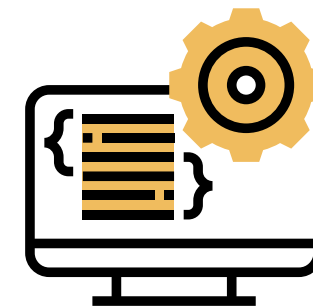


Plataforma	Histórico de mudanças significativas	Quebra com mudanças de dependências	Score
flutter		muito poucas dependências, pois é mais completo em seu core	
reactive native			
kotlin native			

Nota: a pontuação final foi traduzida para uma escala em que quanto maior, melhor.

❑ Restrições da lojas

Medir a estabilidade da plataforma em relação à distribuição da loja e possíveis restrições que podem se aplicar ao aplicativo se uma determinada plataforma for escolhida. Consideramos a presença de atualizações Over the Air (wireless update) um risco potencial, uma vez que a Apple restringiu esse comportamento. Outro fator que levamos em consideração foi a possibilidade de rejeição devido à estrutura de interface do usuário multiplataforma não estar em conformidade com as diretrizes integradas da plataforma. É mais provável que isso aconteça no Flutter, pois não usa os componentes integrados do sistema.



Plataforma	Risco da Tecnologia	Adoção do Mercado	Pontuação
flutter	Baixo (Apple pode julgar alguns restrições de UX)	Médio (Google ADS, Alibaba, Nubank)	Médio-Alto
reactive native	Muito Baixo (possibilidade OTA)	Alto (Facebook, Instagram, Uber)	Alto
kotlin native	Mínimo	Mínimo	Muito-Alto

Nota: a pontuação final foi traduzida para uma escala em que quanto maior, melhor.
Para as colunas intermediárias, quanto menor é melhor

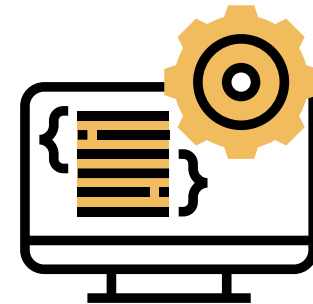
Com relação à possibilidade de ser restrito pelas diretrizes da loja, Kotlin Native é a opção mais segura, pois não tem a possibilidade integrada de executar código que ainda não foi empacotado com o aplicativo e requer a criação de nossa própria IU abstrações usando componentes do sistema.

❑ Restrições da lojas

❑ Conclusão

Em relação às limitações da plataforma, Kotlin Native é a abordagem mais flexível. Ele permite o compartilhamento de código em todos os contextos e não impõe limites aos recursos da plataforma / dispositivo.

Flutter e React Native têm limitações semelhantes principalmente devido à sua abordagem de memória e não podem ser usados fora do processo principal do aplicativo. Essa limitação pode mudar com o tempo, mas consideramos que não é um bloqueador para uso, pois geralmente não dependemos muito do uso do processo auxiliar e ainda é possível escrever código nativo para atender a esses requisitos



❑ Consideramos o uso de WebView

- Flutter
 - Utiliza o componente WebView nativo de cada plataforma (iOS: WKWebView, Android: WebView)
- React native
 - Versão padrão da WebView removida do Core do React Native. Múltiplos substitutos feitos pela comunidade.
- Kotlin Native
 - Não possui componentes (UI), delegando esta funcionalidade para o nativo de cada plataforma. Com isso, além de desenvolvedores Kotlin, é preciso desenvolvedores iOS e Android.

- Flutter
 - Suporte por padrão à modularização e uso de código embarcado.
- React native
 - Suporte por padrão à modularização.
- Kotlin Native
 - Suporte via Gradle à modularização.

Análise das tecnologia mobile cross plataforma

Comparação com Plataformas Nativas

Cross Platform	Nativo
O desenvolvimento tende a ser mais genérico, não direcionado a uma plataforma específica.	O desenvolvimento tende a ser mais direcionado à plataforma
Base de código comum entre as plataformas, ou seja, o código único para rodar em iOS e Android.	Uma base de código entre aplicativos para a mesma plataforma.
Tende a exigir maior esforço de desenvolvimento para não impactar performance	Aplicativo tende a ser mais performático
Exige maior esforço de desenvolvimento para que o comportamento do código (UI) seja mais próximo ao que o usuário da plataforma está acostumado	O App naturalmente tende a se comportar e aparentar o que o usuário da plataforma está acostumado
Custo de desenvolvimento tende a ser menor	Custo de desenvolvimento tende a ser maior
Possível ambiente de desenvolvimento mais barato	Ambiente de desenvolvimento mais caro (iOS)
Equipe de desenvolvimento mais genérico	Equipe de desenvolvimento mais específico, com tendência a ser segregado pela plataforma
Novas features da plataforma dependem de implementação no Flutter	Novas features da plataforma estão naturalmente disponíveis

Adequabilidade aos drivers VR

Driver	Flutter	React native	Kotlin native
Whitelabel			
Modularização menus, features e SDK			
Pipeline CI/CD para a loja			
Segregação de notificações			
Segregação de cadastro e login			
segregação de políticas, termos de uso e LGPD			
Aderência			

Pontos de atenção

- Necessidade de rescrever os legados.
- Fortes diretrizes de integração.

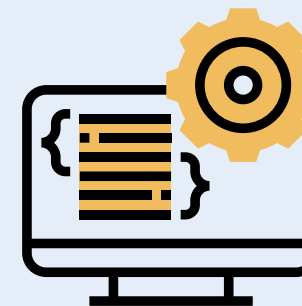
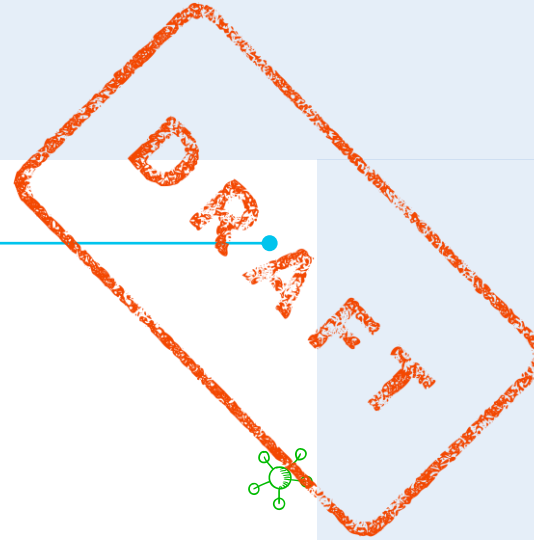
Arquitetura proposta

- Desenho de arquitetura
- Diretrizes de segurança

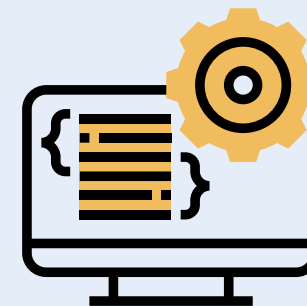
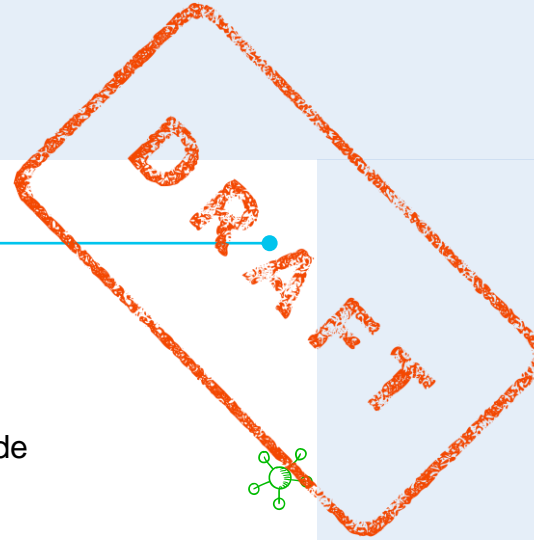
Arquitetura

❖ BRAINSTORM

- ❑ Objetivo:
 - ❑ prevenção de fraudes
 - ❑ Validação de documentos
 - ❑ Face match (reconhecimento facial)
 - ❑ Vantagens:
 - ❑ Valoriza a segurança
 - ❑ Processos de cadastro e login otimizados e com alto grau de confiabilidade
 - ❑ Simplificação da experiência do usuário
 - ❑ Reduz tentativas de fraudes
 - ❑ Otimização de recursos
 - ❑ Sugestão:
 - ❑ Onboarding
 - ❑ Mais uma camada de positivação em transações "fora do comum" possivelmente identificados pelo negócio
-
- ❑ FaceID será uma positivação a mais
 - ❑ soluções para jornada de onboardig e transactions (PREVENÇÃO DE FRAUDES), avaliação dos serviços



ANÁLISE: Autenticação



❖ TaaS (Trust as a Service) - IDWALL



- ☐ Extração de dados de documentos com OCR (RG, CNH e CRLV)
- ☐ Face Match
 - ☐ Captura automática da foto quando atendidos os critérios de qualidade
 - ☐ Comparação de foto do cadastro com foto do documento
 - ☐ Liveness check – garante que foto foi capturada ao vivo

☐ Background check

- ☐ Consulta de dados **de pessoas físicas e jurídicas** em mais de 200 bases de dados públicas e privadas (agilizar processo de onboarding digital)
- ☐ Dashboard centraliza dados como dívidas, processos judiciais, antecedentes criminais, infrações de trânsito e situação de veículos, verificação de CNH, situação cadastral de CPF e CNPJ e etc
- ☐ Análise de risco automatizado e workflow configurável conforme regras do negócio, eliminando necessidade de processos manuais

☐ UX intuitiva que instrui usuário a capturar corretamente a imagem

☐ Documentoscopia – Utiliza técnicas forenses para identificar autenticidade e integridade de documentos (RG, CNH e RNE) acessível através de API

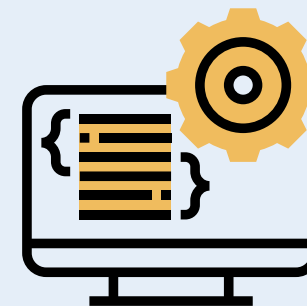
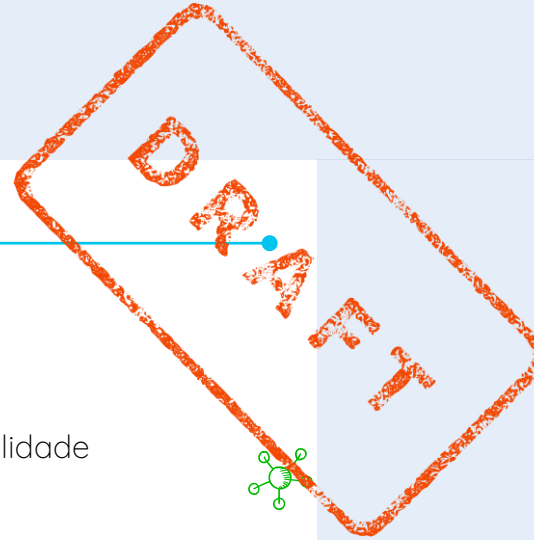
☐ Professional Services – serviço de consultoria em onboarding e validação de usuários composta por time multidisciplinar e certificado

☐ SDK compatível apenas com linguagens nativas (iOS e Android)

☐ Parceiros (instituições financeiras):



ANÁLISE: Autenticação

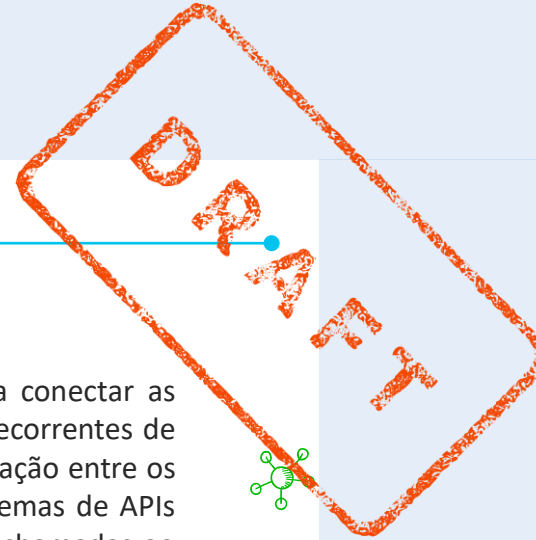


❖ TaaS (Trust as a Service) - UNICO



- ❑ Extração de dados de documentos com OCR (RG, CNH e CPF)
- ❑ Unico Check (face match)
 - ❑ Captura automática da foto quando atendidos os critérios de qualidade
 - ❑ Comparação de foto do cadastro com foto do documento
 - ❑ Liveness check – garante que foto foi capturada ao vivo
- ❑ Confirmação de identificação do usuário em menos de 1 segundo
- ❑ Validação automática se documento enviado realmente é o que foi solicitado
- ❑ Consulta de dados em bases de dados públicas e própria (agilizar processo de onboarding digital)
- ❑ UX intuitiva que instrui usuário a capturar corretamente a imagem (impede envio de fotos embaçadas ou de baixa qualidade)
- ❑ SDK compatível apenas com Android (nativo), iOS (nativo), JavaScript, React Native, Flutter, npm
- ❑ Parceiros (instituições financeiras):



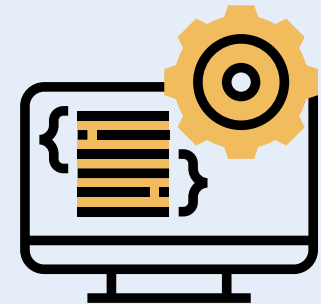


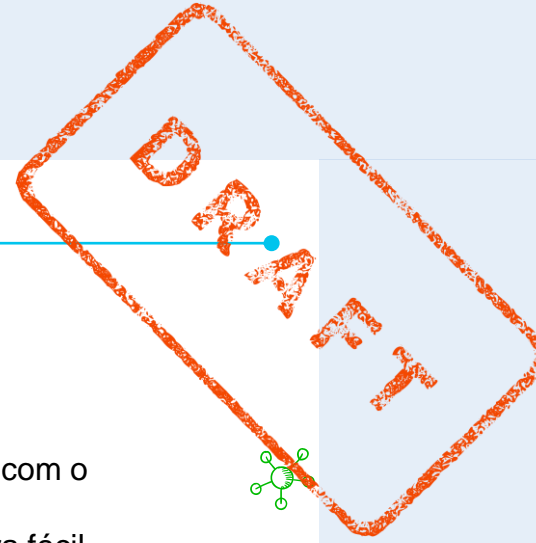
❖ Integrações third-party via Conector de SDKs (Software Development Kit)

- ❑ Atualmente não existe no Aplicativo VR e VC o conceito de uma camada de abstração para conectar as diversas plataformas via SDK. Com o objetivo de diminuir possíveis adaptações no Super APP decorrentes de remoção ou mudanças nas funcionalidades de cada SDK, é recomendado uma camada de abstração entre os SDKs e as funcionalidades do APP.. Essa abstração também deve garantir uso otimizado ao sistemas de APIs dos SDKs, evitar memory leak por parte do SDK e garantir performance na implementação das chamadas ao recursos dos SDKs.

. through scripts, cocoa pods, and providing Xcode template

- ❑ **Slow app builds**
- ❑ **Dependency hell**
- ❑ **Stability and performance issues**
- ❑ **Security risk**
- ❑ **Data quality**
- ❑ **Referências** : <https://www.headspin.io/blog/webinar/missing-the-bloat-improving-mobile-user-experience-through-sdk-abstraction/>





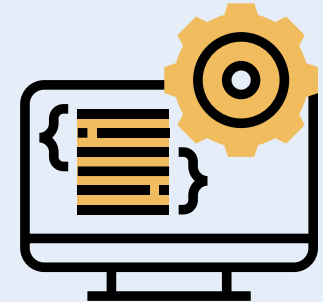
❖ Whitelabel (Thales)

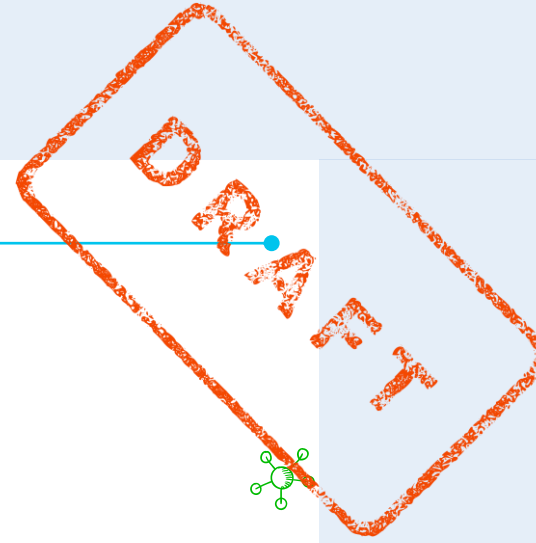
❖ Definição de modulo base

❖ Guia de estilo:

- ❖ Design deve ser pensado já na modularização do que pode ou não ser alterado de acordo com o flavour
- ❖ Definição de paleta de cores com nomes sugestivos estilo cor-texto cor-titulo cor-fundo para fácil mapeamento de estilo para cada flavour
- ❖ Definição de assets por "paleta" também

<https://proandroiddev.com/dynamic-screens-using-server-driven-ui-in-android-262f1e7875c1>





❖ Modularização (Mini APPS) Thales e Ferrarini

- ❖ Com o app todo nativo os mini apps podem ser adicionados como forma de libs

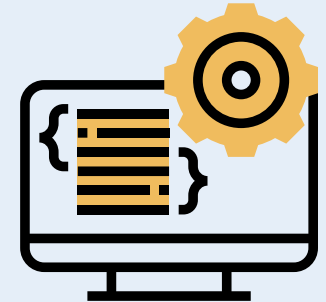
Referências

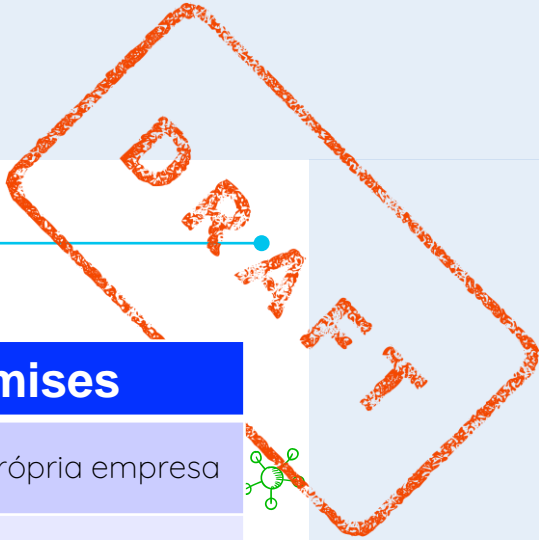
<https://proandroiddev.com/build-a-modular-android-app-architecture-25342d99de82>

<https://medium.com/google-developer-experts/modularizing-android-applications-9e2d18f244a0>

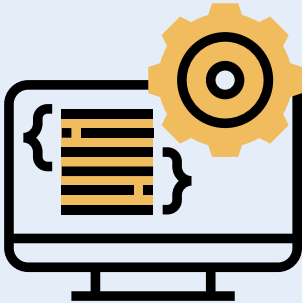
<https://medium.com/@mydogtom/modularization-part-1-application-structure-overview-9e465909a9bc>

- ❑ <https://reactnative.dev/docs/integration-with-existing-apps>



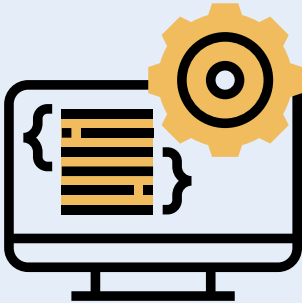


	SaaS	On-premises
Acesso ao código-fonte e dados sensíveis	Provedor tem acesso de leitura & escrita	Acesso restrito à própria empresa
Serviços/Ferramentas CI/CD	Bitrise Codemagic Github Workflows	Fastlane
Custos	Custo de uso do serviço	Máquinas p/ CI/CD Gerenciamento de ferramentas Custo com pessoal
Gerenciamento de infraestrutura	Gerenciamento do provedor	Gerenciamento da própria empresa (ou terceiros)
Restrições	Uso restrito às opções disponibilizadas pelo provedor	Maior liberdade para configuração de pipelines



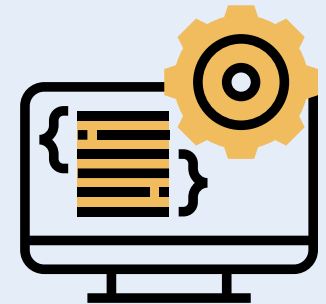
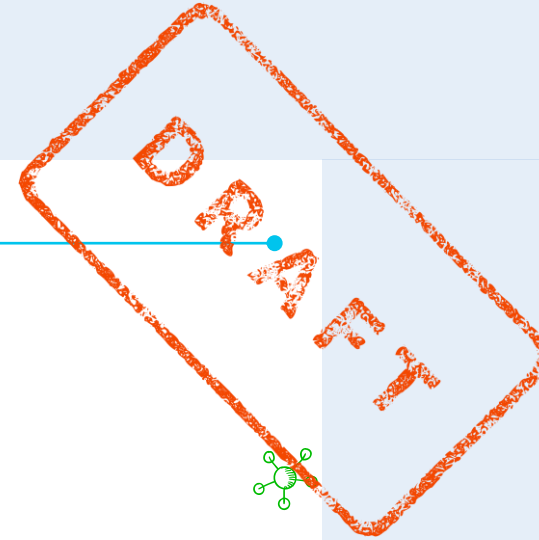


	Bitrise	Codemagic
Preço	A partir de USD35/mês	
Máquinas dedicadas		
Editor de Workflow		
Complexidade configuração via YAML (pipeline as code)	Média	Baixa
Conexão com repositórios	GitHub, GitLab ou Bitbucket	
Single Sign-On	SAML SSO com 2FA	
Integrações		





- ❖ Pipeline CI/CD (Andre Xavier)
- ❖ Testes Automatizados (Andre Xavier)





❖ Segregação de Notificações

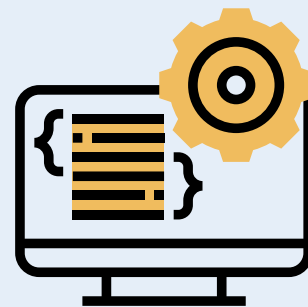
Isolação do sistema de comunicação com o usuário através de uma interface (contrato) que é conhecida, documentada e gerenciável.

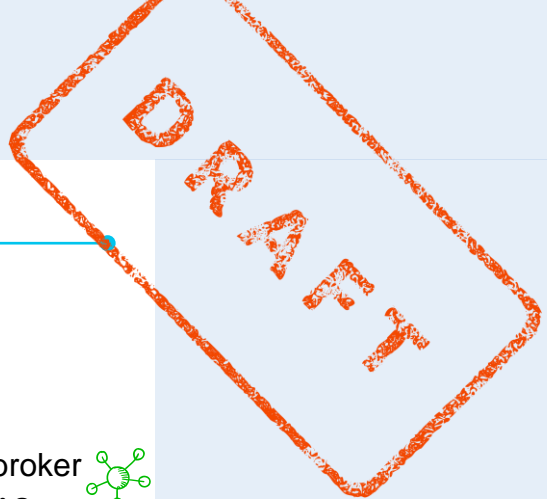


Segregação e controle de utilização através de identificação de aplicação (`client_id`) / serviço (`service_id`) chamador. Esta identificação permite que os assets necessários para a construção da mensagem final sejam selecionados.


Desacoplamento dos canais de comunicação e possibilidade de expansão horizontal. Cada consumidor irá responder a um prefixo `routing_key` específico do canal de comunicação dependendo do tipo da mensagem. Pode-se adicionar mais consumidores para aumentar a capacidade de entrega geral do sistema ou reconfigurar a topologia das filas para beneficiar um `client_id`.

Cada instância que tenha responsabilidade de se comunicar com brokers externos deve ter acesso as credenciais necessárias para a realização deste trabalho. Estas credenciais devem estar disponíveis somente leitura em um serviço de vault.

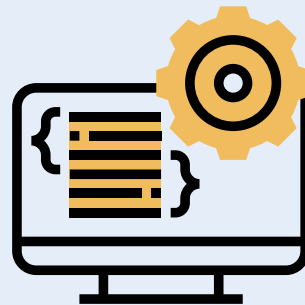




❖ Segregação de Notificações

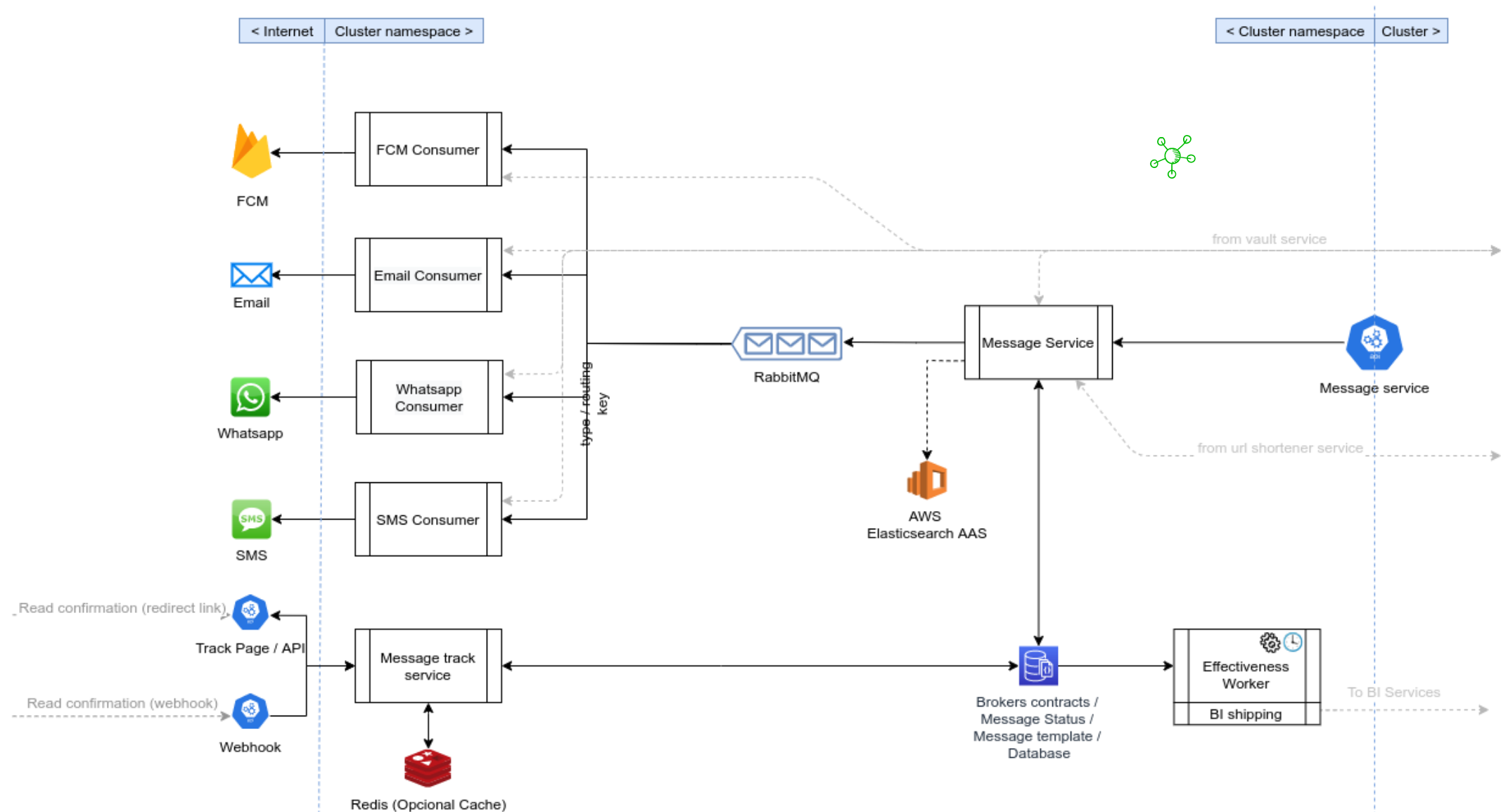
Retorno de confirmação de leitura de mensagem através de webhook, pagina de redirecionamento ou API. O broker  escolhido para lidar com as mensagens deve ser capaz de informar via webhook quando o destinatário efetuar a leitura da mensagem. Quando o canal for Aplicativo móvel o próprio aparelho deve fazer a chamada síncrona ou não, batch ou individual à API de confirmação de leitura. Em se tratando de brokers que não possuam formas de informar a leitura, um link de redirecionamento pode ser embarcado na mensagem.

Contabilização de efetividade do disparo de mensagem. Através de um serviço autônomo acionado com uma base temporal predefinida a contagem das estatísticas acerca das quantidades e tempos decorridos entre os envios e as leituras (caso existam) são feitos e enviados para o sistema de BI correspondente.

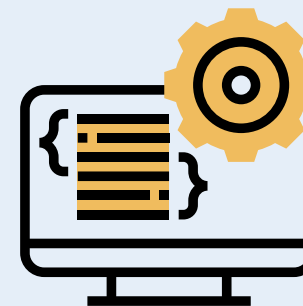
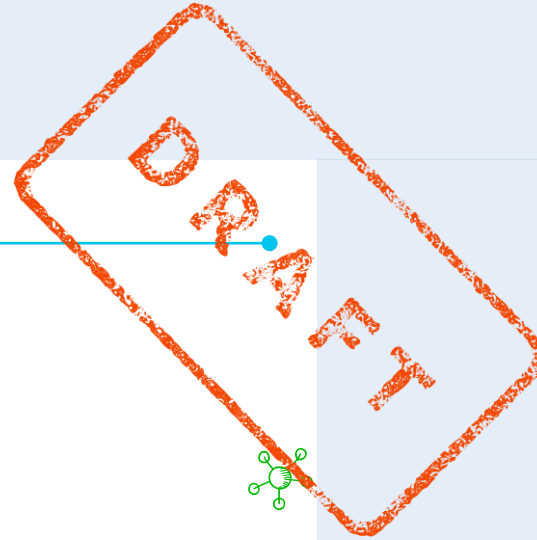




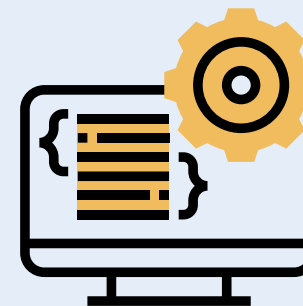
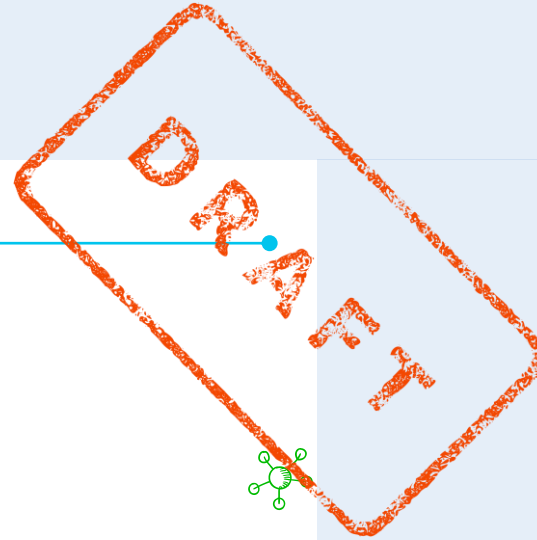
Segregação de Notificações



- ❖ segregação de cadastro/login (Anderson / André)
 - ❖ Da uma verificada em solução de elevação de privilegio



- ❖ segregação de políticas, termos de uso e LGPD. (Andre. Andre Xavier e Alex)



Segurança

Síntese & recomendações





SEE YOU SOON!