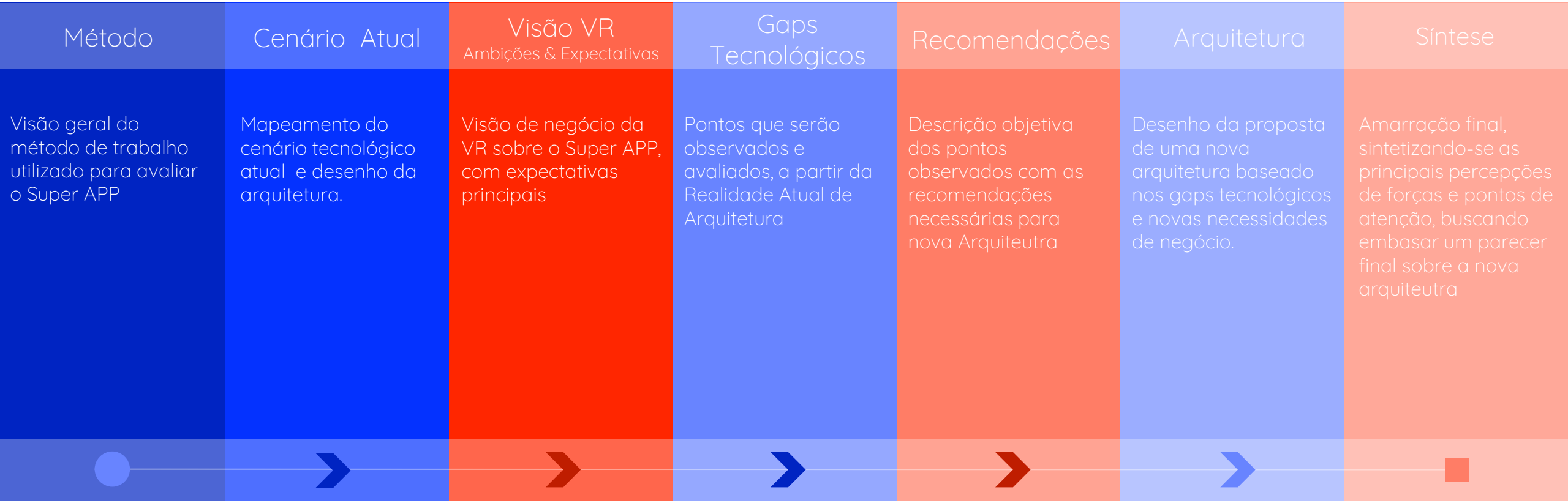
SUPER APP

ÍNDICE

MÉTODO

MÉTODO

As etapas do trabalho:



MÉTODO

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

CENÁRIO ATUAL: Funcionalidades

Sob um ponto de vista funcional do Aplicativo Mobile VR & VC, identificamos 05 principais grupos de funcionalidades:

- Cadastro do usuário (Onboarding)
- Carrossel
- Saldo / Extrato / Gráfico de Utilização
- Cartão Virtual/ Pagamento via QR Code
- Busca da Rede Credenciada

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

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 beneficio no período dos 30 dias.

Cadastro do usuário

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

Cartão Virtual/Pagamento via QR Code

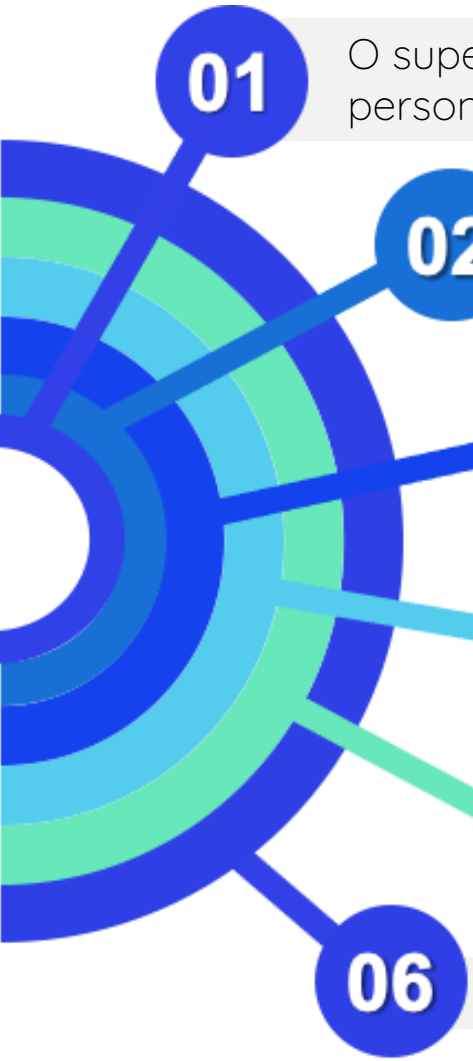
Geração de cartões virtuais s e pagamento via QR Code gerado a partir das maquininhas de cartões da Cielo ou via VR Pague .

Busca da Rede Credenciada

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

CENÁRIO ATUAL: Requisitos

Do ponto de vista estratégico, destacamos os requisitos que devem ser suportados pela nova arquitetura do aplicativo mobile VR & VC



01 O super APP deve ser whitelabel, podendo assim ser ofertado para os parceiros emissor da VR de forma personalizado e ser facilmente redistribuído.

02 Capacidade de ser modularizado (menus, funcionalidades e SDKs) para cada parceiro emissor.

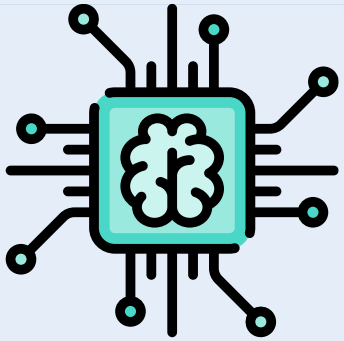
03 Integração as lojas Google Play e App Store através de um pipeline de CI/CD

04 Ter a capacidade de segregação das notificações (push)

05 Prover mecanismo para segregação de cadastro/login e ou integrado com emissor.

06 Implementar funcionalidade de segregação de politicas, termos de uso e LGPD.

Arquitetura técnica (1/4)



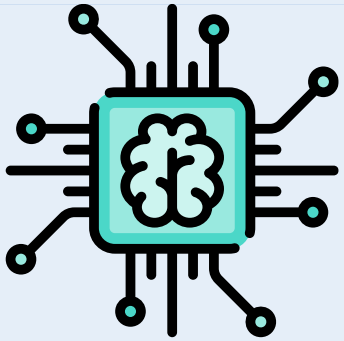
❖ Aspectos técnicas gerais da Arquitetura atual

- ❑ 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, ON-PREMISES na TIVIT, que suporta o barramento IBM, 4Time, aplicação SNCORE e banco de dados Oracle. E um ambiente na CLOUD AWS onde encontra-se os serviços de notification center, gestão e promoção, MC busca, gestão de identidade e RHSSO, sendo que esses serviços são orquestrados pelo OPENSHIFT.

❖ 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 disponibilizado 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.

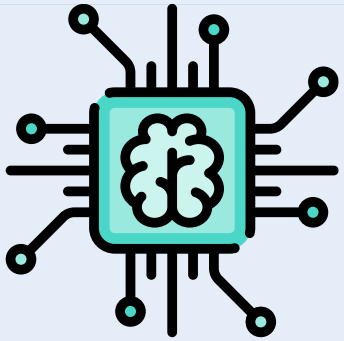
- ❑ O Aplicativo mobile não adota a estratégia de armazenamento em bancos de dados local, as poucas informações que são salvas no dispositivo, utiliza-se de um arquivo que é persistido em sessões (Preferencies),

- ❑ 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 banco 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)

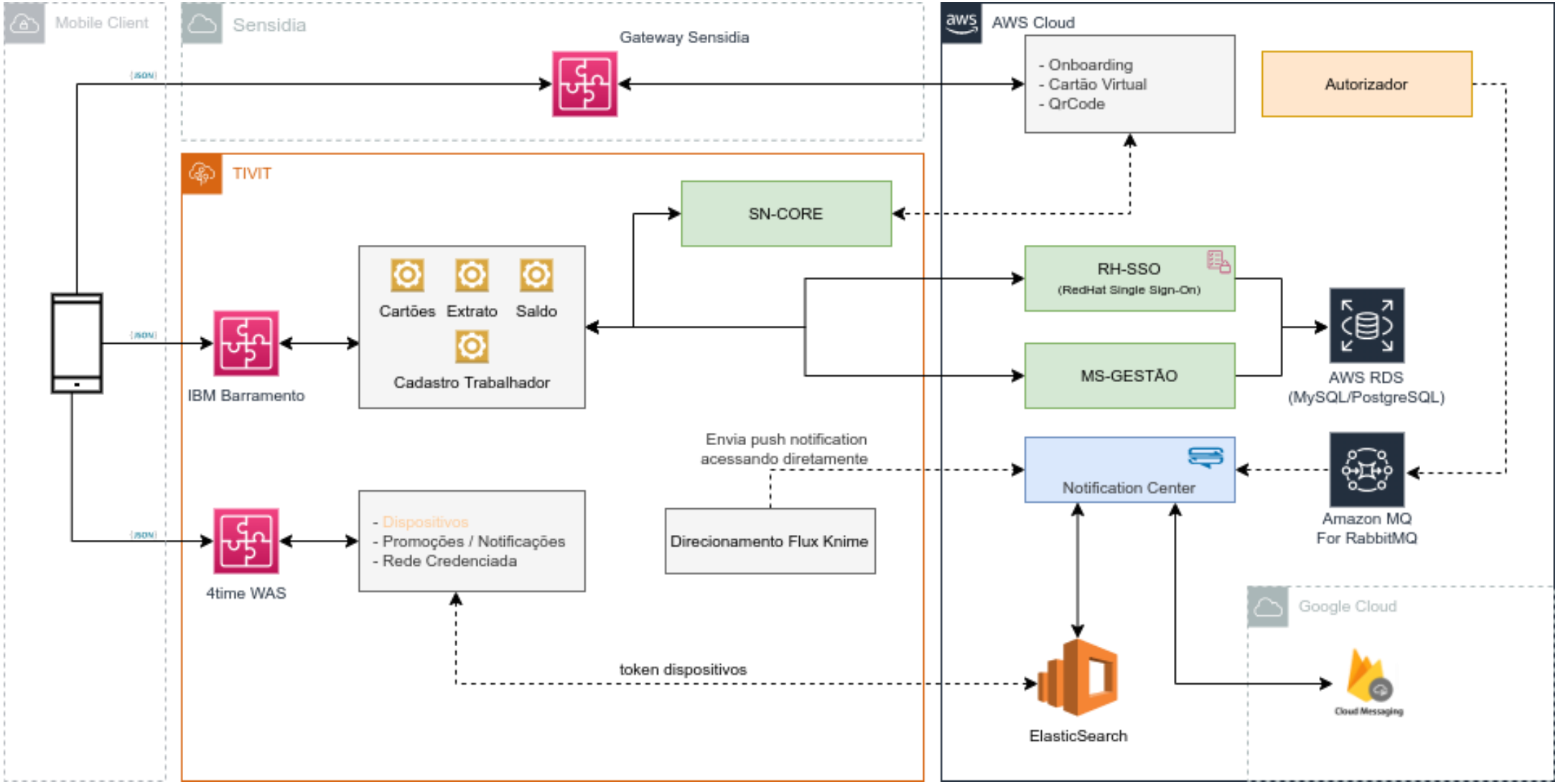


❖ Segurança

- ❑ 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.

Arquitetura técnica (4/4)

❖ Blue Print



Práticas do desenvolvimento

- ❖ CI / CD / Build & Deploy

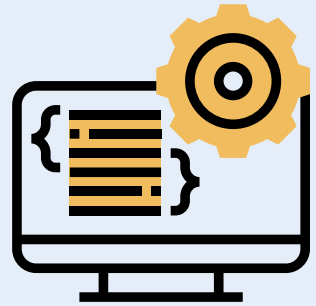
- ❑ Há uma iniciativa em fase de testes de pipeline CI/CD tanto para iOS quanto Android utilizando ferramenta SaaS (Bitrise) na nuvem.

- ❑ 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 CI/CD dos aplicativos (tanto iOS quanto Android) estão sendo disparados manualmente nas próprias máquinas dos desenvolvedores

- ❑ Há consolidado um pipeline 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.

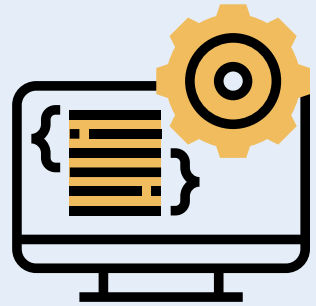
    - ❑ Os testes de regressão não fazem parte dessa pipeline executada diariamente, mas podem vir a ser executados sob demanda



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.
- ❑ 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
- ❑ 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
- ❑ Performance, para casos específicos (sob demanda), é construída suíte de testes de performance utilizando jMeter executados em ambiente de desenvolvimento e homologação



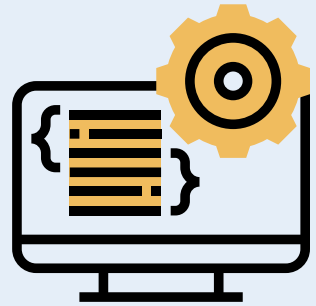
ANÁLISE

ANÁLISE: Plataforma

❖ Critérios de Avaliação da Plataforma

❑ Para avaliação da plataforma de desenvolvimento da VR foram considerado os seguintes critérios de avaliação

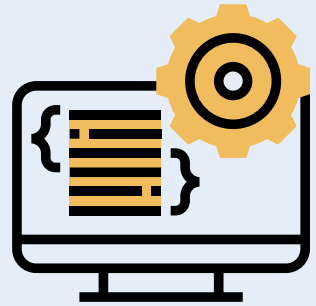
- ❑ Experiência de Desenvolvimento - Fatores que contribuem para permitir que um desenvolvedor entregar e ser produtivo no aplicativo móvel.
  - ❑ Hot Reload
  - ❑ Visibilidade do componente
  - ❑ Ferramentas de Debugger
  - ❑ Integração IDE
  - ❑ Ferramentas de teste
- ❑ Viabilidade de Longo Prazo - Confiança que o mantenedor da plataforma manterá 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
  - ❑ Adoção por grandes empresas
  - ❑ Tamanho da comunidade (número de núcleos contribuidores, contribuidores externos, etc ...)
- ❑ Sem especialização de plataforma - Um engenheiro deve ser capaz de escrever código móvel para o produto sem diferenciar entre Android e iOS. O código deve ter a mesma aparência e se comportar no Android e iOS, com baixa ocorrência de falhas / problemas específicos do sistema operacional.



ANÁLISE: Plataforma

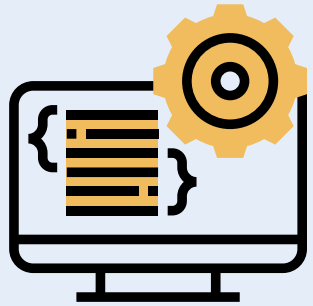
❖ Critérios de Avaliação da Plataforma

- ❑ API / Estabilidade da Ferramenta - plataforma api ou mudanças de ferramentas que exigem a alteração do código interno.
  - ❑ Mudanças nas APIs bases ou mudanças de dependência que o tornam incompatível com versões anteriores.
  - ❑ Mudanças nos componentes nativos (OS) que quebrar o comportamento interno de uma solução cross-plataforma.
- ❑ Restrições das Lojas - Risco de Apple ou Google restringir o aplicativo em de qualquer forma por causa do uso de uma plataforma subjacente.
  - ❑ Flutter UX não corresponde ao HIG da Apple (Diretrizes de Interface Humana).
  - ❑ A possibilidade de atualizações Over The Air em React Native / Flutter tornando-se um bloqueador para a Apple.
- ❑ MINI APPS – Capacidade de incorporar mini apps, através de webview garantindo segurança, performance e usabilidade. Como pré-requisito a limitação do controle de navegação e funções nativas do aparelho.
- ❑ Modularização – Capacidade de modularizar o desenvolvimento da plataforma para que seja possível escalar os times de desenvolvimento e facilitar o acoplamento de novas funcionalidades.



ANÁLISE: Plataforma

- ❖ Critérios de Avaliação da Plataforma
  - ❑ Experiência de Desenvolvimento



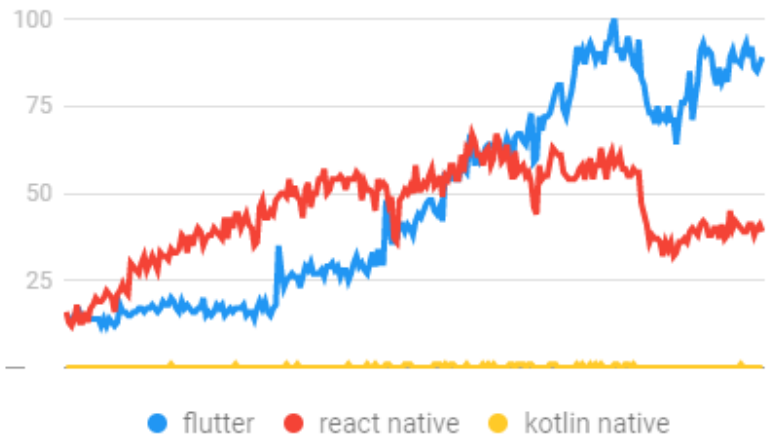
ANÁLISE: Plataforma

❖ Critérios de Avaliação da Plataforma

❑ Viabilidade a longo prazo (Framework Cross Plataform)

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

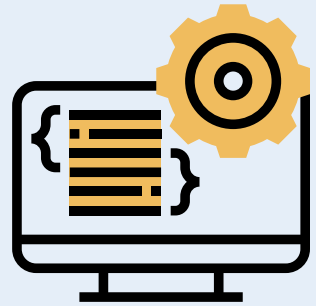


Tamanho da comunidade

Plataforma	# Stars/Forks	StackOverflow questions
flutter	126.000/18.300	97.696
reactive 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
reactive native – Walmart, Airbnb, UbearEats
kotlin native – Netflix (Prodicle), Leroy Merlin

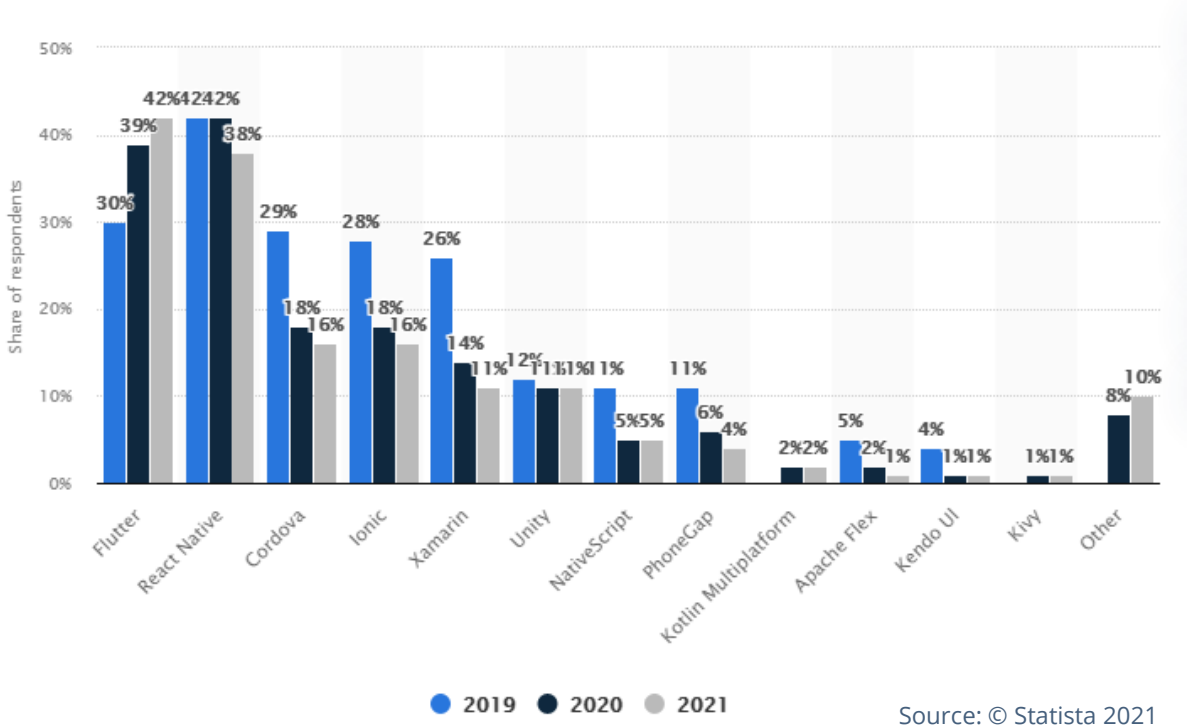


ANÁLISE: Plataforma

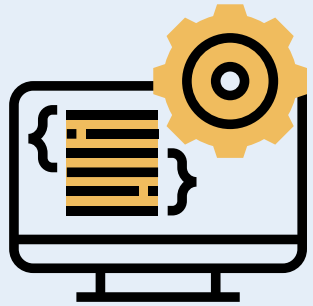
❖ Critérios de Avaliação da Plataforma

❑ 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 Satista. 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



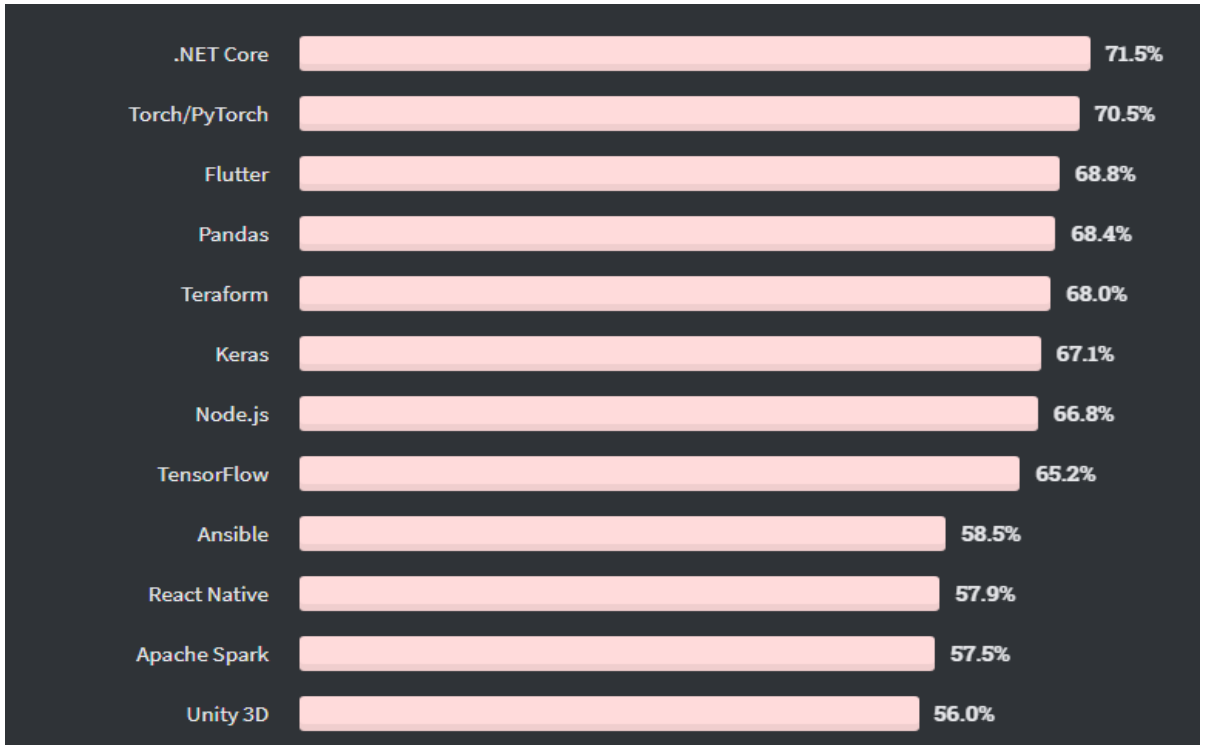
ANÁLISE: Plataforma

❖ Critérios de Avaliação da Plataforma

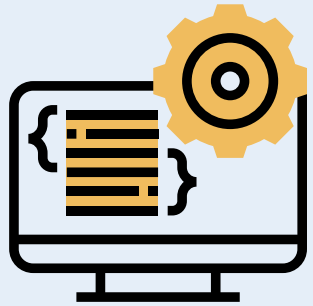
❑ Viabilidade a longo prazo (Framework Cross Plataform)

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



ANÁLISE: Plataforma

❖ Critérios de Avaliação da Plataforma

❑ Viabilidade a longo prazo (Framework Cross Plataform)

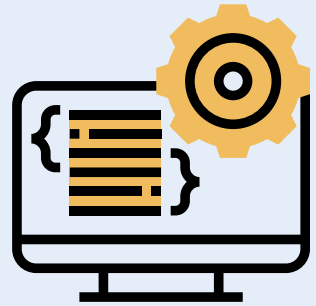
❑ Conclusão

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.



ANÁLISE: Plataforma

❖ Critérios de Avaliação da Plataforma

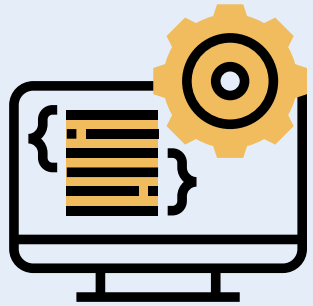
❑ Restrições da lojas

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 o 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.

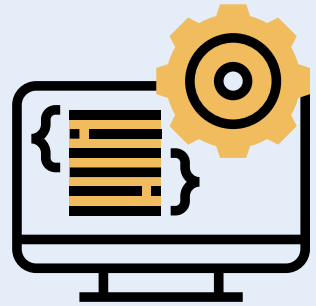


ANÁLISE: Plataforma

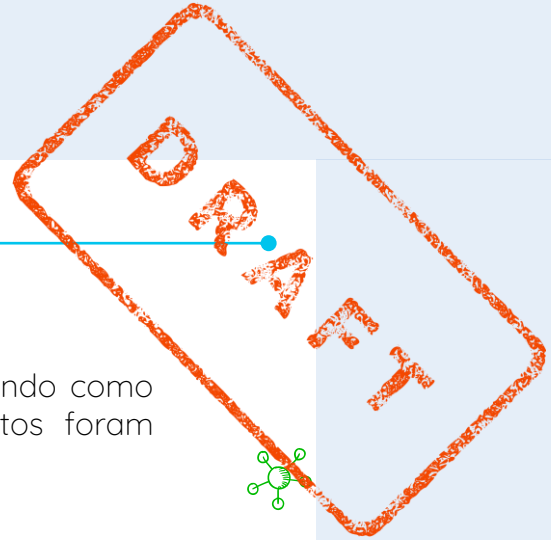
- ❖ Critérios de Avaliação da Plataforma
  - ❑ Restrições da lojas
    - ❑ Conclusão

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

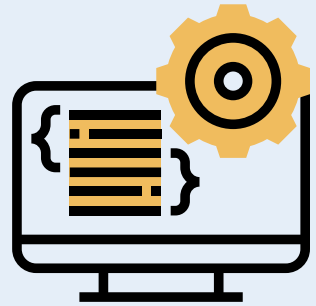


ANÁLISE: Arquitetura

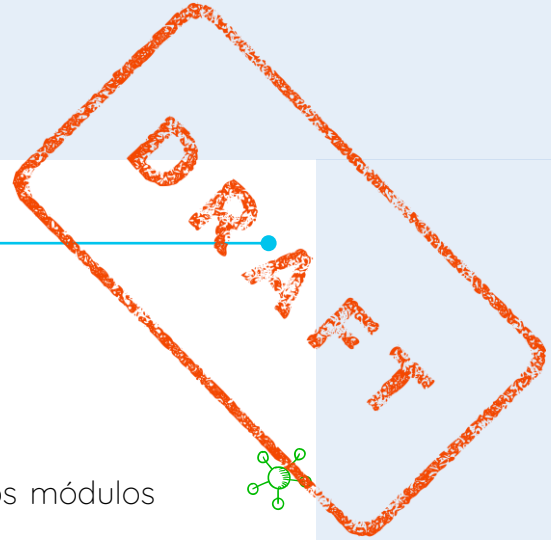


❖ Visão Geral

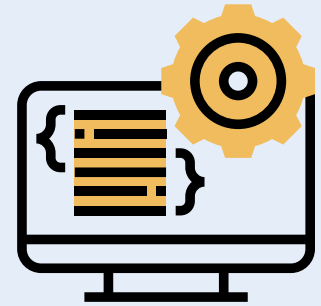
- ❑ Com uma expectativa futura de trazer novos negócios para o ecossistema da VR, tendo como canal principal com o usuário final o aplicativo mobile VR e VC, algumas pontos foram observados para suportar o novo Super App.
  - ❑ Maior tempo de build, devido a adição de novas funcionalidades no aplicativo.
  - ❑ Mais complexidade para testes independentes das novas funcionalidades
  - ❑ Interferência e dependências ao escalar equipes de desenvolvimento.
- ❑ Para mitigar esses problemas, sugerimos a adoção de estratégias de modularização do APP (Multi-Module Architecture), nesse modelo a orquestração dos módulos será de responsabilidade da aplicação core (engine) responsável por carregar os módulos a partir de mecanismos de configurações e prover toda infraestrutura para acoplamentos desses módulos.
  - ❑ Nesse tipo de abordagem os módulos não precisam conhecer ou ter dependências de outro módulos, o conceito de módulo é de ser baseado em feature module (ex.: mini app). Os módulos precisam se conectar a engine através de um contrato, ou seja, dessa forma, a engine poderá garantir que métodos como ini, por exemplo, existem nos módulos acoplados e serão sempre executados. As dependências necessárias pelos módulos deverá ter sua gestão pela aplicação core e a comunicação entre mini app e aplicação core via message bridge.
  - ❑ No processo de desenvolvimento, a equipe responsável pelo módulo, precisa apenas ter o aplicativo core para conectar o seu módulo e poder realizar o Building, Testing, Publication independente de outros módulos.
  - ❑ Referência: https://www.w3.org/TR/mini-app-white-paper/

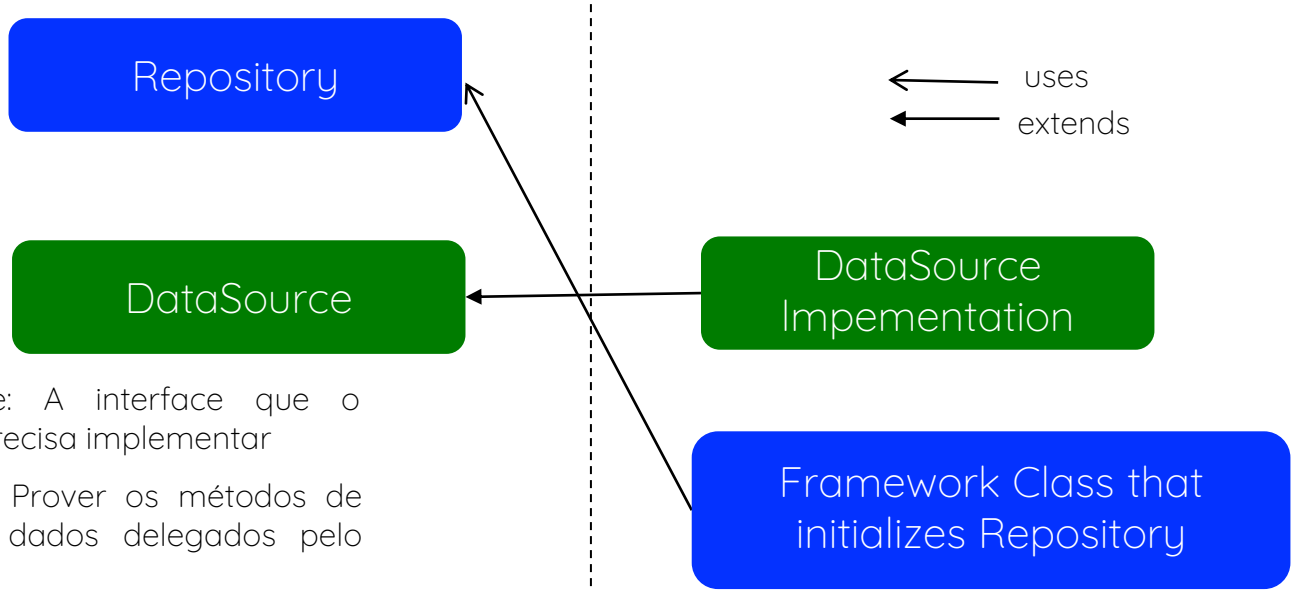


ANÁLISE: Arquitetura



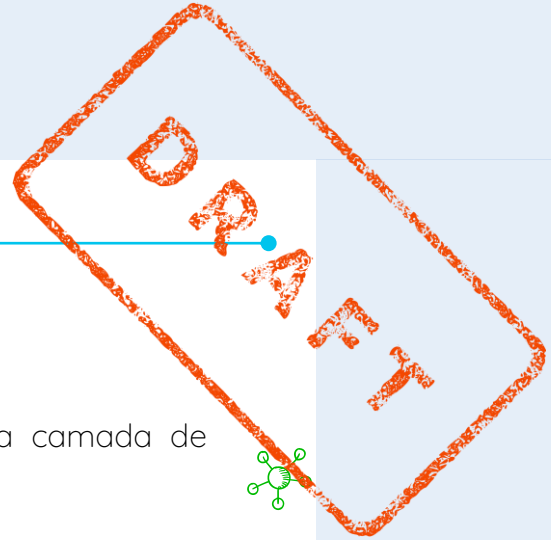
- Solução Nativa ou Hibrido
- Padrão Sugerido
  - A proposta de abordagem é utilização do MVVM pattern e Clean Architecture para os módulos (mini apps) e aplicativo core.
    - Camada de Dados
      - O acesso a fonte de dados e serviços(http) será baseado no padrão de projeto repository pattner, com a criação de uma classe e interface para cada modelo, e aproveitando o principio de inversão de dependência para abstrair a fonte de dados.



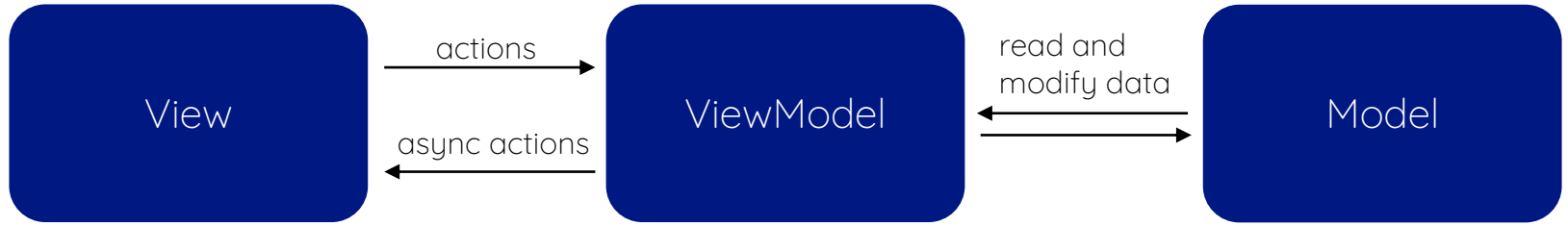


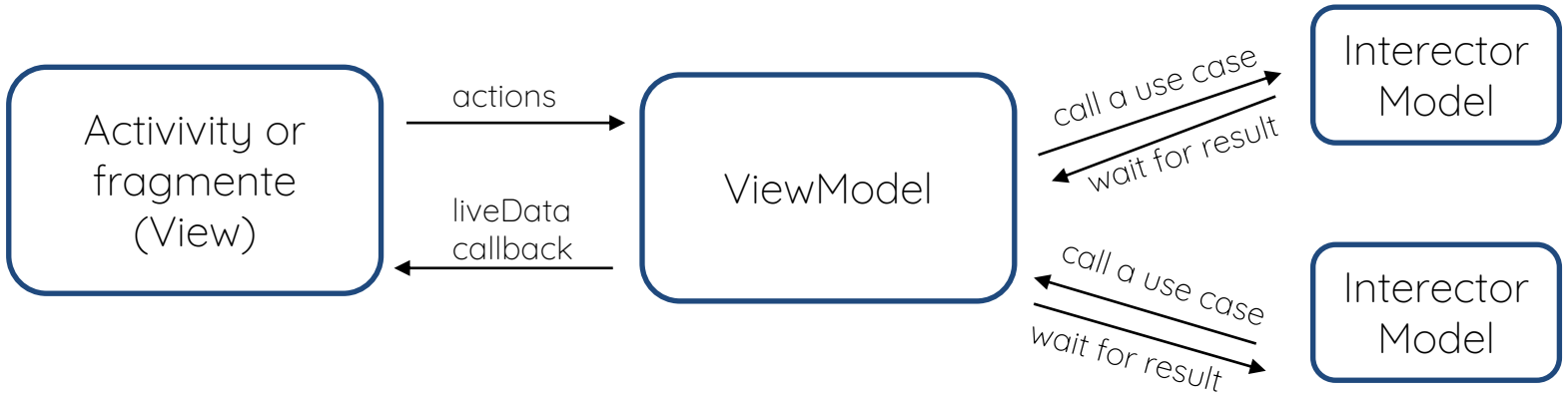
- DataSource: A interface que o framework precisa implementar
- Repository:: Prover os métodos de acesso aos dados delegados pelo DataSource.

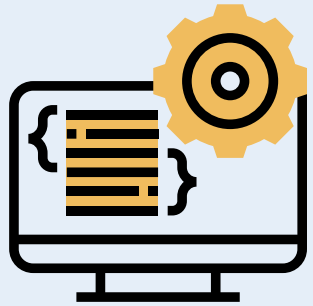
ANÁLISE: Arquitetura



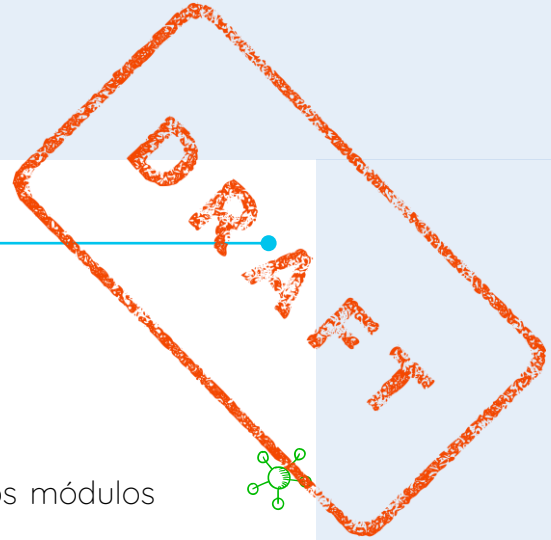
- Padrão Sugerido
  - Camada Framework
    - A camada de framework contém as implementações de interfaces definidas na camada de dados,
  - Camada de Apresentação
    - A campada de apresentação contém todo código relacionado a interface com o usuário. Esta camada está no mesmo nível da camada de framework. Para versão Android o padrão utilizado nessa camada será o MVVM suportado pelo Android JetPack.



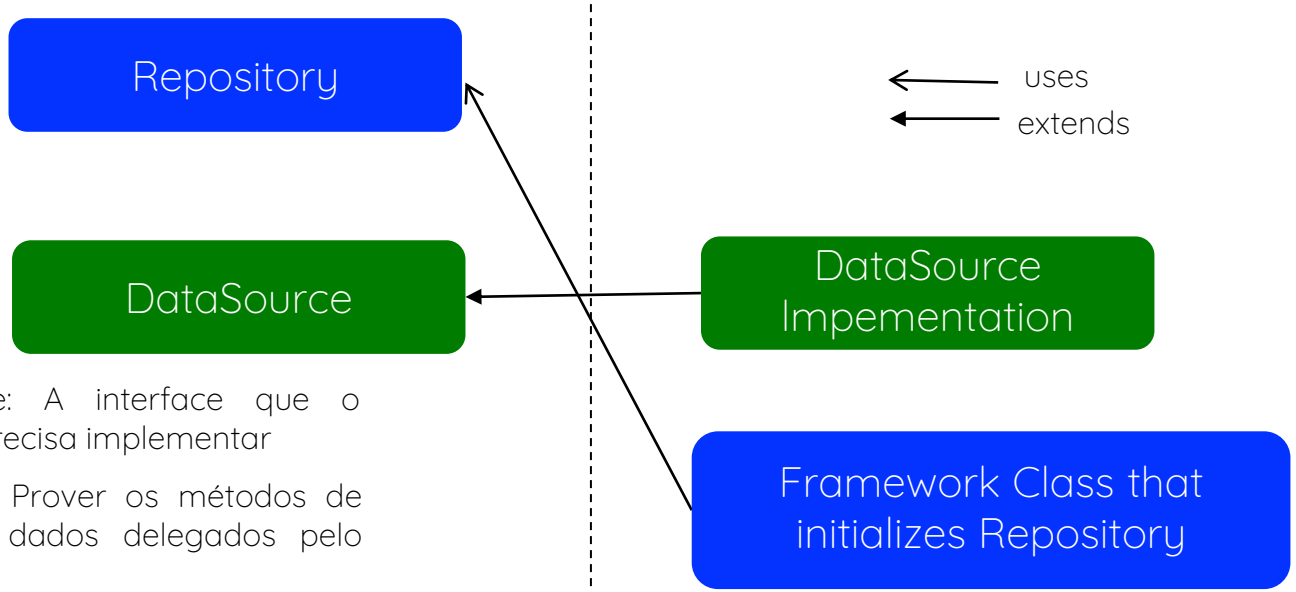




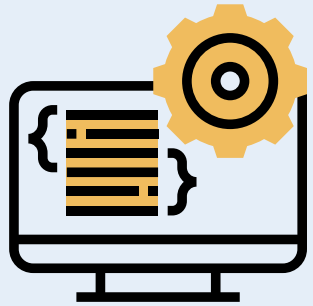
ANÁLISE: Arquitetura



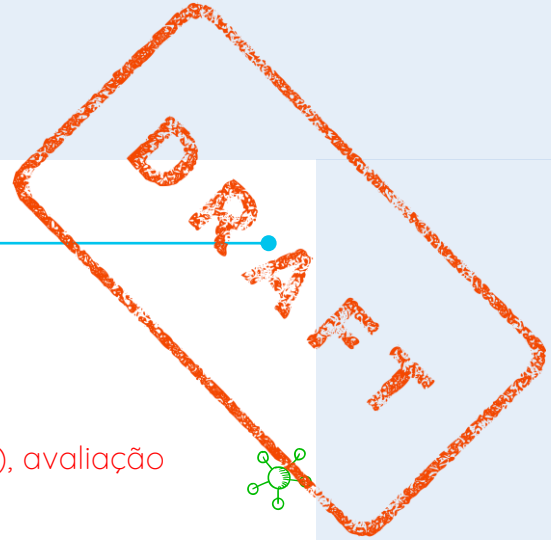
- Solução Nativa ou Hibrido
- Padrão Adotado
  - A proposta de abordagem é utilização do MVVM pattern e Clean Architecture para os módulos (mini apps) e aplicativo core.
    - Camada de Dados
      - O acesso a fonte de dados e serviços(http via retrofit) será baseado no padrão de projeto repository pattner, com a criação de uma classe e interface para cada modelo, e aproveitando o principio de inversão de dependência para abstrair as fontes de dados.



- DataSource: A interface que o framework precisa implementar
- Repository:: Prover os métodos de acesso aos dados delegados pelo DataSource.

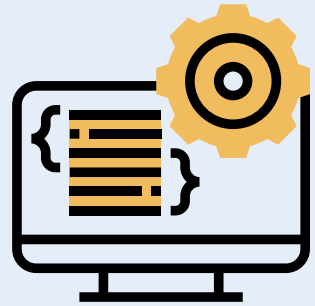


ANÁLISE: Autenticação

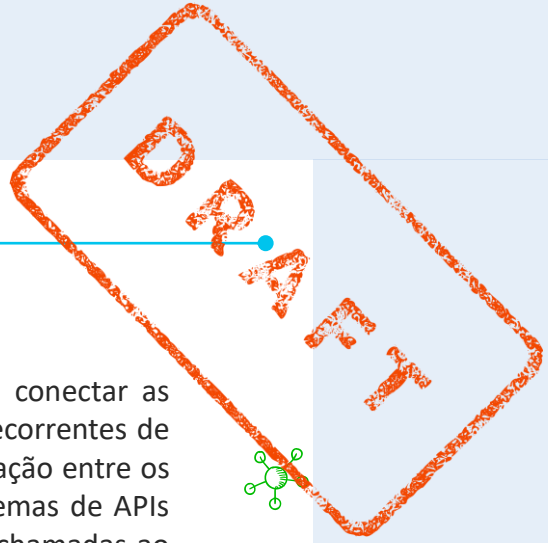


❖ BRAINSTORM

- ❑ FaceID será uma positivação a mais
  - ❑ soluções para jornada de onboardig e transactions (PREVENÇÃO DE FRAUDES), avaliação dos serviços
    - ❑ Único
    - ❑ IDWALL
    - ❑ ZAIG

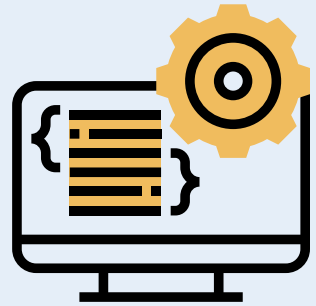


ANÁLISE: SDKs



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

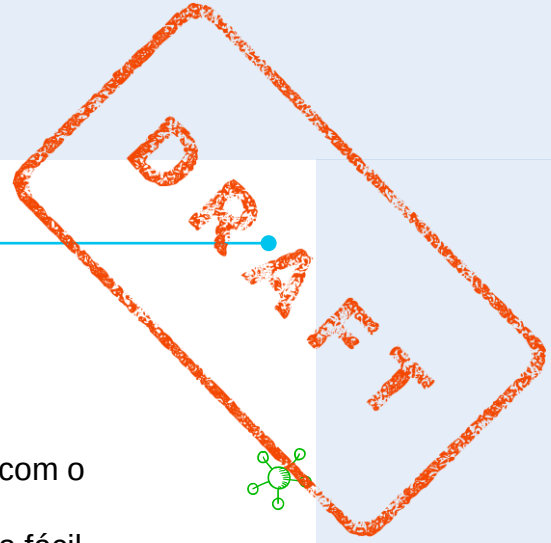
- ❑ Atualmente não existe no Aplicativo VR e VC o conceito de uma camada de abstração para conectar as diversas plataformas via SDK. Com o objetivo de diminuir possíveis adaptações no Super APP decorrentes de remoção ou mudanças nas funcionalidades de cada SDK, é recomendado uma camada de abastração 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/**

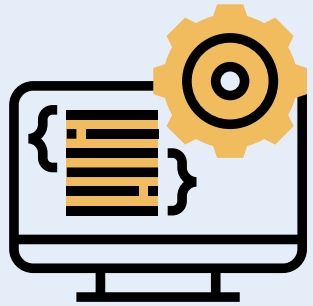
ANÁLISE: WhiteLabel



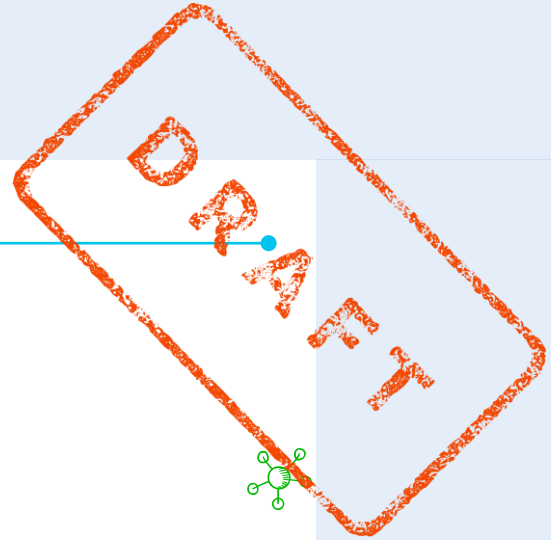
- Whitelabel (Thales)

- 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



ANÁLISE: Modularização

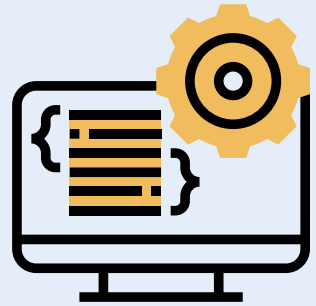


❖ Modularização (Mini APPS) Thales e Ferrarini

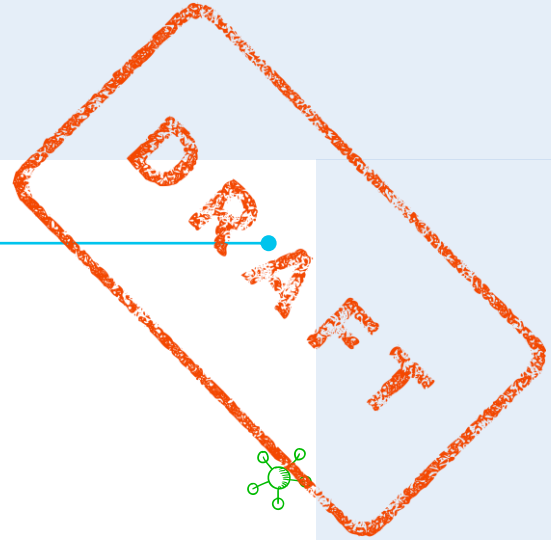
Referências

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

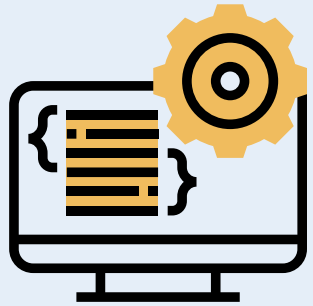
❑ https://reactnative.dev/docs/integration-with-existing-apps



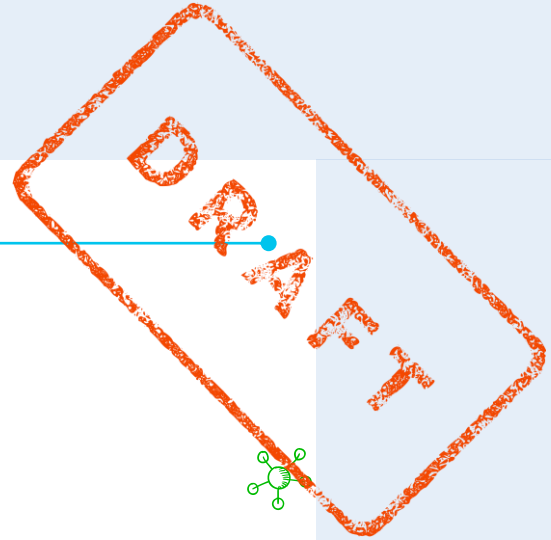
ANÁLISE: SDKs



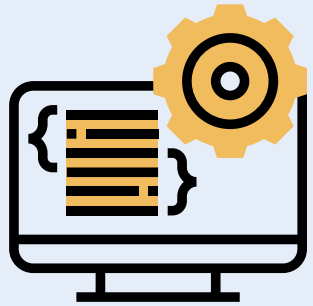
- Pipeline CI/CD (Andre Xavier)
- Testes Automatizados (Andre Xavier)



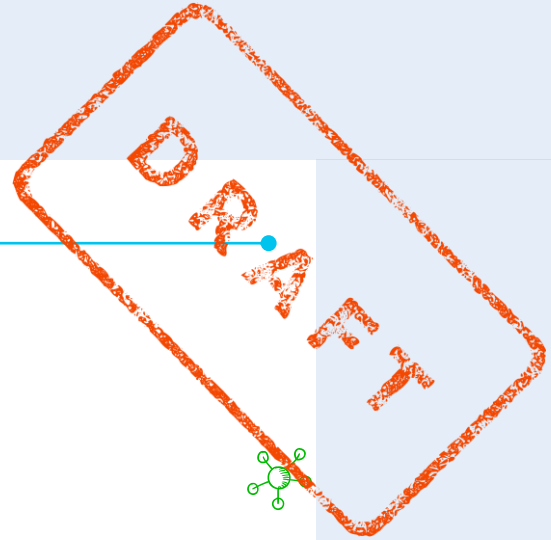
ANÁLISE: SDKs



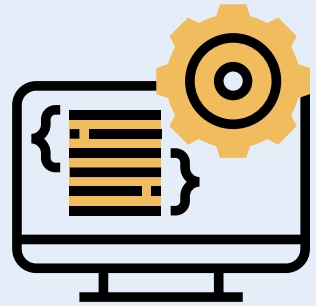
- Segregação de Notificações (Alex)
  - Segregação de bloqueio (case ifood)



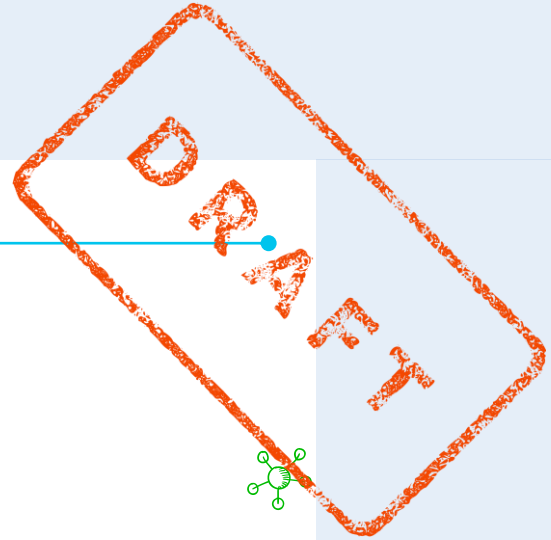
ANÁLISE: SDKs



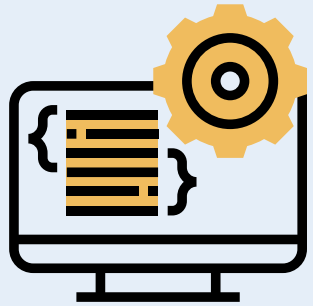
- segregação de cadastro/login (Anderson / André)
  - Da uma verficada em solução de elevação de privilegio



ANÁLISE: Segregação



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



ARQUITETURA PROPOSTA

SÍNTESE & RECOMENDAÇÕES

SEE YOU SOON!