



System Design — Desafio Técnico

Pessoa Desenvolvedora Android Sênior | GOK Inovação Digital

O Cenário

Você acaba de assumir como Android Sênior em um projeto da GOK para um cliente do segmento financeiro: uma fintech de médio porte que processa pagamentos via Pix, boleto e cartão.

O app atual tem **~800 mil usuários ativos mensais**, foi construído há 4 anos com arquitetura monolítica, navegação baseada em Activities, callbacks aninhados e sem separação de camadas. O time anterior saiu. Você precisa **propor a arquitetura do app reescrito** — com uma restrição crítica:

Atenção: O app não pode parar. Releases continuam acontecendo a cada duas semanas enquanto a refatoração ocorre.

O backend já foi modernizado: existe um **BFF (Backend for Frontend)** dedicado ao mobile, um **API Gateway** com rate limiting e circuit breaker, e microserviços independentes para pagamentos, autenticação e notificações.

O entrevistador abre com a seguinte pergunta:

"Me descreva como você arquitetaria esse app do zero — considerando que ele precisa continuar evoluindo em paralelo com a refatoração, que opera em ambiente financeiro crítico e que o time é pequeno."

Requisitos do Sistema

Funcionais

- Autenticação com OAuth2, biometria e suporte a múltiplos fatores (MFA)
- Fluxo completo de pagamento Pix — chave, QR Code estático e dinâmico, copia-e-cola
- Histórico de transações com paginação e filtros
- Notificações push em tempo real para confirmação de transações
- **Modo offline parcial:** visualização de saldo e histórico recente sem conexão
- Deep links para abertura direta em fluxos específicos (ex: pagamento via link)
- Feature flags para rollout gradual de novas funcionalidades

Não Funcionais

Dimensão	Requisito
Disponibilidade	Tolerância zero a crash em fluxos de pagamento
Performance	Telas de pagamento devem renderizar em menos de 300ms após navegação
Segurança	Tokens OAuth2 não podem ser armazenados em SharedPreferences plano; proteção contra repackaging e hooking
Escalabilidade	Arquitetura que permita múltiplos devs trabalhando em features paralelas sem conflito
Testabilidade	Cobertura mínima de 70% nas camadas de domínio e dados
Observabilidade	Crashes e erros de transação rastreáveis com contexto suficiente para reprodução

Os Desafios Ocultos

Estes são os pontos que o entrevistador **não menciona explicitamente**, mas espera que um candidato sênior levante por conta própria. Não espere ser perguntado — antecipe.

1. Migração incremental sem feature freeze

A reescrita não pode ser do tipo *big bang*. É necessário propor uma estratégia de convivência entre código legado e código novo.

A abordagem recomendada é o padrão **Strangler Fig** aplicado ao mobile: módulos novos convivem com telas antigas até que a migração esteja completa, feature por feature.

Dica estratégica: Se você não levantar esse ponto, o entrevistador vai perguntar diretamente: *“Como você garante que o app funcione durante a transição?”* Sair na frente demonstra senioridade.

2. Consistência de estado em fluxos de pagamento

O QR Code dinâmico do Pix tem janela de expiração. O estado da transação no cliente precisa ser gerenciado em pelo menos três cenários críticos:

- O usuário vai para o background **durante** o fluxo de pagamento
- A rede cai **após o envio**, mas antes da confirmação chegar
- O backend retorna timeout, mas a transação **pode ter sido processada**

Erro comum: Tratar o timeout como falha definitiva. Em ambiente financeiro, um timeout ambíguo exige estado pendente, não erro.

3. Segurança do token e superfície de ataque

`EncryptedSharedPreferences` é a camada mínima aceitável — não o teto. Em dispositivos roteados, essa proteção é insuficiente.

A abordagem correta envolve:

- **Keystore** para derivação de chave
- **EncryptedSharedPreferences** para armazenamento
- **Certificate Pinning** para comunicação com o BFF

- **Detecção de root e hooking** como camadas adicionais de defesa

Dica estratégica: Segurança mobile é feita em camadas. Não existe proteção absoluta no dispositivo — o objetivo é elevar o custo do ataque.

4. Custo real da modularização em time pequeno

Modularização não é bala de prata. Ela traz overhead concreto: tempo de build, complexidade de navegação entre módulos e convenções que precisam ser mantidas.

Em um time pequeno com pressão de entrega, defender modularização completa desde o primeiro dia sem reconhecer esse custo é um sinal de imaturidade técnica.

Dica estratégica: Proponha modularização incremental — comece com 2 a 3 módulos core e expanda conforme o time e o produto amadurecem.

5. Modo offline e consistência de dados

“Modo offline parcial” parece simples, mas esconde uma decisão arquitetural relevante: **quando o cache local e o servidor divergem após a reconexão, qual é a fonte da verdade?**

A abordagem correta para dados financeiros:

- **Leitura:** offline permitido, com indicador visual explícito de dados desatualizados (*staleness*)
- **Escrita:** online obrigatório — saldo e transações nunca devem ser enviados com base em cache

6. Observabilidade além do Crashlytics

O Crashlytics captura crashes. Mas o que captura os **erros silenciosos**?

- Uma transação que falhou sem lançar exception

- Um timeout tratado incorretamente como sucesso
- Uma resposta mal-formada do BFF que passou pela validação

Dica estratégica: Pense em **logging estruturado de eventos de negócio**, não apenas em crash analytics. Em ambiente financeiro, o que não foi logado pode não ser auditável.

As Perguntas do Entrevistador

As perguntas seguem uma progressão intencional: começam amplas, afinam em trade-offs e terminam em cenários de pressão.

Abertura

"Por onde você começaria? Me descreve a arquitetura em alto nível antes de entrar nos detalhes."

O que está sendo avaliado: Capacidade de estruturar o raciocínio antes de codar e clareza de comunicação técnica.

Aprofundamento arquitetural

"Você escolheu Clean Architecture com MVI. Por que MVI e não MVVM nesse contexto específico? Quais são os trade-offs reais dessa decisão para o time?"

O que se espera: Ir além de *"MVI é unidirecional"*. Mencionar previsibilidade de estado em fluxos de pagamento, rastreabilidade de UI states para debug — e reconhecer o custo adicional de boilerplate em features simples.

Modularização

"Como você estruturaria os módulos? Como gerenciaria dependências compartilhadas — autenticação, tema, componentes de UI?"

O que se espera: Distinção entre *feature modules*, *core modules* e *app module*.
Estratégia de navegação via contratos de interface ou navigation graph compartilhado.
Reconhecimento do custo de configuração Gradle.

Cenário de falha

"O usuário confirmou um Pix. O app enviou a requisição para o BFF. A rede caiu antes da resposta chegar. O usuário reabre o app 30 segundos depois. O que acontece?"

O que se espera: Não existe resposta única correta — o entrevistador quer ver o raciocínio. Espera-se menção a:

- ☐ Estado pendente persistido localmente
 - ☐ Polling ou push notification para confirmação assíncrona
 - ☐ UI de estado ambíguo: *"Verificando pagamento..."*
 - ☐ Estratégia de idempotência acordada com o backend
-

Segurança

"Onde você armazenaria o access token e o refresh token? E se o device estiver roteado?"

O que se espera: Keystore para derivação de chave, EncryptedSharedPreferences para armazenamento, detecção de root como camada adicional — e a clareza de que segurança mobile é um conjunto de camadas, não uma solução absoluta.

Migração incremental

"O app atual está em produção com 800 mil usuários. Como você refatora sem parar os releases?"

O que se espera: Strangler Fig pattern, feature branches por módulo, testes como rede de segurança para a migração e feature flags para ativação gradual das novas implementações.

Testabilidade

"Como você garante testabilidade em uma Clean Architecture com MVI? Me dá um exemplo concreto de como testaria o fluxo de confirmação de pagamento."

O que se espera:

- Teste unitário do Use Case com repositório mockado (MockK)
 - Teste do ViewModel verificando a sequência de `UiStates` emitidos
 - Teste de integração do repositório contra servidor fake ou interceptor de rede
-

Pressão e ambiguidade

"O PM quer lançar o novo fluxo de Pix em 3 semanas. A arquitetura nova ainda não está pronta. O que você faz?"

O que se espera: Maturidade, não perfeição. O candidato não cede à pressão sem análise. Propõe alternativas concretas — feature flag no legado, escopo reduzido, débito técnico explícito — e comunica riscos com clareza para o PM.

Dica estratégica: Nessa pergunta, demonstrar que você *não* vai simplesmente dizer sim vale mais do que qualquer resposta técnica elaborada.

Trade-offs que Você Deve Levantar por Iniciativa Própria

Atenção: Um candidato sênior não espera ser perguntado sobre trade-offs. Ele os articula espontaneamente como parte do raciocínio arquitetural.


Decisão	Trade-off a articular
Clean Architecture	Mais testabilidade e flexibilidade, mas mais arquivos, mais boilerplate e curva de onboarding maior. Em times pequenos, pode gerar overhead real.
MVI vs MVVM	MVI oferece rastreabilidade de estado superior para fluxos complexos, mas tem mais boilerplate em telas simples. Para um app misto, pode ser válido usar MVI seletivamente nos fluxos financeiros e MVVM nas demais telas.
Modularização completa vs incremental	Modularização total reduz acoplamento e paraleliza o desenvolvimento, mas tem custo alto de configuração Gradle. Em time pequeno com pressão de entrega, começar com 2 a 3 módulos core é mais realista.


Decisão	Trade-off a articular
Offline-first vs online-first	Offline-first melhora resiliência, mas para dados financeiros pode comunicar informações desatualizadas como verdade — um risco de UX e confiança. A abordagem correta: offline para leitura com indicador de <i>staleness</i> , online obrigatório para escrita.
Feature modules vs módulos por camada	Feature modules isolam contextos de negócio. Módulos por camada são mais simples, mas criam acoplamento entre features. Para a GOK com time pequeno, feature modules com core compartilhado é o equilíbrio mais defensável.
EncryptedSharedPreferences vs Keystore direto	EncryptedSharedPreferences é uma abstração conveniente, mas insuficiente em dispositivos comprometidos. A chave deve ser derivada pelo Keystore, com proteções adicionais como camadas complementares.
StateFlow vs LiveData	StateFlow não está vinculado ao ciclo de vida do Android nativamente — exige <code>repeatOnLifecycle</code> para coleta segura. LiveData é mais simples, mas menos expressivo para estados complexos e não funciona bem com Compose. A escolha por StateFlow é a correta, mas o custo de gerenciamento de ciclo de vida precisa ser compreendido.
Strangler Fig vs Big Bang rewrite	Big Bang garante consistência arquitetural, mas congela features por meses — inaceitável para 800 mil usuários em produção. Strangler Fig tem complexidade temporária e risco de inconsistência, mas é o único caminho viável. O candidato deve definir também o critério de <i>“módulo pronto para migrar”</i> .

Sinal de maturidade técnica: O candidato que espontaneamente diz *“essa abordagem tem custo X , e em um time pequeno isso importa”* demonstra que pensa em engenharia de produto — não apenas em engenharia de software. Esse é exatamente o perfil que separa um sênior executor de um sênior de referência.

Career Agent PRO

A preparação perfeita para cada vaga de emprego. Inteligência aplicada ao seu contexto e à empresa que você quer entrar.

 www.career-agent-pro.com

 support@main.career-agent-pro.com