

**Projeto:** AchaPro

**Equipe:** Filipe B; Lucas M; Joao C; Cleiton V; Daniela L.

## 1. Documento de Arquitetura de Software (DAS)

### 1.1. Visão Geral e Stack Tecnológica

A solução adota uma arquitetura Serverless e Modular focada na rapidez de desenvolvimento e escalabilidade horizontal. A aplicação será desenvolvida utilizando Next.js (React) como framework principal, cobrindo tanto o Frontend quanto a camada de API (Backend-for-Frontend). A infraestrutura de dados e serviços backend será provida pelo Supabase (BaaS - Backend as a Service), e a gestão de identidade será delegada ao Clerk.

Esta escolha técnica responde diretamente ao NFR07 (Manutenibilidade e Modularidade), reduzindo a complexidade de gerenciar servidores dedicados e permitindo que a equipe foque nas regras de negócio e na usabilidade exigida pelo NFR03.

### 1.2. Componentes da Arquitetura

#### Frontend e Camada de Aplicação (Next.js)

O cliente será uma aplicação web progressiva (PWA) ou responsiva construída com Next.js.

- **Renderização:** Utilização de *Server-Side Rendering (SSR)* e *React Server Components* para garantir que o tempo de carregamento inicial (First Contentful Paint) respeite o limite de 2 segundos estabelecido no NFR01.
- **Roteamento e Proteção:** O Middleware do Clerk será utilizado para proteger rotas privadas (ex: /propostas, /chat), garantindo que apenas usuários autenticados (Clientes ou Prestadores) acessem funcionalidades restritas.

#### Gerenciamento de Identidade e Segurança (Clerk)

Para atender aos RF01 (Cadastro de Usuários) e NFR05 (Criptografia de Senhas):

- O Clerk gerenciará todo o fluxo de autenticação, incluindo sessões, hashing de senhas e verificação de e-mail.
- A aplicação não armazenará senhas no banco de dados principal. O vínculo entre o usuário logado e seus dados de negócio será feito através do User ID fornecido pelo Clerk, que servirá como chave estrangeira nas tabelas do Supabase.

#### Backend e Persistência (Supabase)

O Supabase atuará como o núcleo de dados, substituindo um backend tradicional monolítico.

- **Banco de Dados (PostgreSQL):** Armazenará os perfis (RF02), tarefas (RF03), propostas (RF05) e avaliações (RF09). A integridade referencial garantirá a consistência entre "Clientes" e "Prestadores".
- **Armazenamento de Arquivos (Storage):** As fotos de perfil (RF02) e imagens das tarefas (RF03) serão armazenadas em *buckets* privados ou públicos no Supabase Storage. O banco de dados guardará apenas as URLs de referência.
- **Tempo Real (Realtime):** Para viabilizar o chat entre Cliente e Prestador (RF07), utilizaremos o recurso de *Realtime* do PostgreSQL. O frontend "escutará" inserções na tabela de mensagens, atualizando a interface instantaneamente sem necessidade de *polling*, otimizando o tráfego.

### 1.3. Fluxos Críticos de Dados

**Publicação de Tarefa (Cliente):** O usuário envia os dados do formulário e imagens. O Next.js processa o upload para o Supabase Storage, recebe a URL e insere o registro da tarefa na tabela *tasks* do banco de dados via cliente Supabase autenticado.

**Contratação e Chat:** Ao aceitar uma proposta (RF06) o sistema cria uma "sala" (registro na tabela *matches*). A partir desse momento, as regras de segurança (Row Level Security - RLS) do Supabase permitem que ambos os usuários subscrevam às mensagens vinculadas àquela sala específica.

## 2. Relatório da Revisão de Design

### 2.1. Avaliação de Conformidade com Requisitos (GQM)

A revisão técnica analisou se as decisões arquiteturais suportam as metas de qualidade definidas:

- **Desempenho (Meta 2 / NFR01 e NFR02):** A escolha do Next.js com SSR é adequada para manter o carregamento abaixo de 2 segundos. No entanto, a dependência de serviços externos (Supabase e Clerk) introduz latência de rede. Veredito: O uso de Edge Functions para lógicas críticas é recomendado para garantir que a resposta da API fique abaixo de 500ms (NFR02).
- **Segurança (NFR05 e NFR06):** A delegação da autenticação para o Clerk elimina o risco de implementação incorreta de criptografia de senhas (NFR05). Toda a comunicação com Clerk e Supabase é forçada via HTTPS/TLS, atendendo integralmente ao NFR06.
- **Manutenibilidade (NFR07):** A separação entre frontend (Next.js) e dados (Supabase) cria um desacoplamento claro. O uso de TypeScript (padrão no Next.js) reforça a qualidade do código e a detecção precoce de erros, alinhando-se à Métrica 2.1.1 (ausência de code smells graves).

## 2.2. Riscos Identificados e Mitigações

Durante a análise dos Requisitos Funcionais em relação à Stack, foram levantados os seguintes pontos de atenção:

1. **Complexidade do Chat (RF07):** Embora o Supabase Realtime facilite a implementação, a lógica de "quem pode falar com quem" deve ser rigorosamente controlada via *Row Level Security (RLS)* no banco de dados. Se as políticas de segurança falharem, um usuário poderá ler mensagens de outras negociações. *Ação:* Priorizar testes de segurança (Penetration Testing) especificamente nas tabelas de mensagens.
2. **Consistência de Dados entre Clerk e Supabase:** Existe o risco de um usuário ser criado no Clerk, mas a criação do perfil correspondente no Supabase falhar (ex: erro de rede). Isso quebraria o RF01 e RF02. *Ação:* Implementar Webhooks do Clerk que acionem uma Edge Function no Supabase para criar o registro do usuário no banco de dados de forma assíncrona e garantida, assegurando a integridade dos dados.
3. **Escalabilidade de Uploads:** O upload de múltiplas fotos de alta resolução para as tarefas (RF03) pode degradar a experiência do usuário móvel. *Ação:* Implementar redimensionamento de imagem no lado do cliente (browser) antes do envio para o Storage, economizando banda e armazenamento.

## 2.3. Parecer Final

A arquitetura proposta baseada na stack Next.js/Supabase/Clerk é Aprovada, pois oferece as ferramentas necessárias para cumprir o cronograma agressivo do MVP (Meta 3) e atende aos requisitos de qualidade. A equipe deve proceder com a configuração do ambiente e a implementação do "Módulo de Usuários" (Épico 2).