Documentação Técnica - TaskMaster

---

1. Arquitetura Utilizada

Padrão Modular por Features:

- Divisão clara em `auth` e `tasks` como funcionalidades independentes
- Estrutura por feature:
    - `context`: Lógica de estado (Context API + useReducer)
    - `screens`: Componentes de tela
    - `components`: Componentes reutilizáveis
    - `navigation`: Configuração de navegação
    - `types`: Definições TypeScript

Camadas Principais:

- **Presentation:** Componentes UI (React Native Paper + Gesture Handler)
- **Application Logic:** Contextos e serviços
- **Data:** API Client com camada de mock
- **Infrastructure:** Configurações de tema e ambiente

---

2. Decisões Técnicas Relevantes

a. Gerenciamento de Estado:

- Context API + useReducer para fluxos complexos (autenticação e tarefas)
- Separação clara entre:
    - Estado de autenticação (usuário, tokens)
    - Estado das tarefas (CRUD + arraste no Kanban)

b. Navegação:

- React Navigation Stack para fluxos autenticados/não autenticados
- Tipagem forte de rotas com TypeScript

c. UI/UX:

- React Native Paper para componentes consistentes
- Sistema de tema unificado (cores, espaçamentos, tipografia)

- Feedback visual integrado (Snackbars + Loaders)

d. Testabilidade:

- Mock API configurável via environment flag
- Separação clara entre serviços e componentes

---

3. Preparação para Integração com Back-End

a. Camada de Serviço:

- `apiClient` configurado com:
  - Interceptores para autenticação (JWT)
  - Timeout global (10s)
  - Headers padronizados

b. Estratégia de Mock:

- Implementação completa com axios-mock-adapter
- Armazenamento local via AsyncStorage
- Estrutura de dados compatível com endpoints reais

c. Tipagem Forte:

- Interfaces `User` e `Task` compartilhadas
- Validação de erros de API via type guards

d. Ambiente Configurável:

- Arquivo `environment.ts` para:
  - URL base da API
  - Flag de mock ativável

e. Autenticação Pronta:

- Fluxo completo de login/registro
- Armazenamento seguro de tokens
- Validação automática via interceptors

---

4. Desafios e Soluções

a. Drag-and-Drop no Kanban

- **Desafio:** Implementação fluída com gestos complexos
- **Solução:** Combinação de:
    - react-native-gesture-handler
    - react-native-reanimated
    - Cálculo de posicionamento relativo

b. Sincronização de Estado Offline

- **Desafio:** Manter consistência com mock API
- **Solução:** Estratégia de storage por usuário:

```
typescript
Copy

Download
AsyncStorage.setItem(`tasks_${userId}`, ...)
```

c. Feedback Visual Integrado

- **Desafio:** Gerência de múltiplos estados (loading/erro/sucesso)
- **Solução:** Componente `Feedback` unificado com:
    - Snackbars posicionados absolutamente
    - Sistema de priorização de mensagens

d. Navegação Segura

- **Desafio:** Proteção de rotas autenticadas
- **Solução:** Estrutura de navegação condicional:

```
tsx
Copy

Download
{state.user ? <MainNavigator/> : <AuthNavigator/>}
```

e. Tipagem de Erros de API

- **Desafio:** Garantir tratamento seguro de erros
- **Solução:** Type guard customizado:

```
typescript
Copy

Download
const isApiError = (error: unknown): error is APIError => {...}
```

---

5. Melhorias Futuras

1. Implementação de testes E2E com Detox
2. Adição de refresh token automático
3. Sistema de notificações push
4. Otimização de performance para listas grandes
5. Internacionalização (i18n)

Esta arquitetura proporciona escalabilidade, mantendo a simplicidade para projetos de médio porte, com todas as bases técnicas necessárias para evolução contínua.