# Architecture Decision Record (ADR): Plataforma de Trading

## Contexto

Estamos desenvolvendo uma plataforma de trading desktop que deve oferecer uma interface moderna e ágil para operações de mercado, incluindo day-trading que exige alta performance. Este documento registra as decisões arquiteturais tomadas para guiar o desenvolvimento contínuo da plataforma.

## Data da Decisão

13 de abril de 2025

## Status

Aceito

## Decisões

### 1. Stack Tecnológica

**Decisão:** Utilizar Electron como framework para aplicação desktop, React com TypeScript para a interface de usuário, e Webpack para bundling.

**Justificativa:**

- Electron permite desenvolvimento cross-platform com uma única base de código
- React oferece componentização e reatividade ideal para dados de mercado em tempo real
- TypeScript adiciona segurança de tipos, essencial para sistemas financeiros
- Webpack permite otimizações avançadas de bundling

**Consequências:**

- Tamanho maior da aplicação devido ao runtime do Electron
- Necessidade de otimizações específicas para garantir performance adequada

### 2. Arquitetura da Aplicação

**Decisão:** Arquitetura baseada em dois processos Electron (main e renderer) com frontend React organizado por componentes e páginas.

**Estrutura de diretórios:**

```
src/
├── main/              # Processo principal Electron
├── renderer/          # Processo de renderização (React)
│   ├── components/    # Componentes reutilizáveis
│   │   ├── Chart/     # Componentes relacionados a gráficos
│   │   └── Trading/   # Componentes relacionados a trading
│   ├── pages/         # Páginas da aplicação
│   └── services/      # (A implementar) Serviços de comunicação com APIs
└── shared/            # Código compartilhado entre processos
```

**Justificativa:**

- Separação clara de responsabilidades
- Facilidade de manutenção e escalabilidade
- Organização por domínio/funcionalidade

## 3. Renderização de Gráficos

**Decisão:** Utilizar lightweight-charts para renderização de gráficos financeiros.

**Justificativa:**

- Biblioteca especializada em gráficos financeiros
- Boa performance mesmo com grandes conjuntos de dados
- API compatível com React

**Considerações futuras:**

- Monitorar desempenho com volumes maiores de dados
- Possível migração para soluções baseadas em WebGL para casos de uso mais intensivos

## 4. Estilização e UI/UX

**Decisão atual:** CSS-in-JS via objetos de estilo inline com tema escuro.

**Decisão futura:** Migrar para uma abordagem mais estruturada de CSS, potencialmente com CSS Modules ou Styled Components.

**Justificativa:**

- UI/UX moderno e consistente é crucial para plataformas de trading
- Tema escuro reduz fadiga visual em uso prolongado
- Migração futura para sistemas de estilo mais escaláveis

## 5. Gerenciamento de Estado

**Decisão atual:** Estado local via React Hooks (useState, useEffect).

**Decisão futura:** Implementar um sistema de gerenciamento de estado global mais eficiente para dados de mercado e posições de usuário.

**Opções consideradas:**

- Zustand para estado global leve e eficiente
- Jotai/Recoil para estado atômico
- Context API + useReducer para casos simples

**Justificativa:**

- Day-trading exige atualizações rápidas e consistentes da UI
- Necessidade de comunicação eficiente entre componentes distantes na árvore
- Estado global deve ser otimizado para evitar renderizações desnecessárias

## 6. Otimizações de Desempenho

**Decisões para implementação:**

1. **Renderização eficiente de listas:**
   - Implementar virtualização para listas longas (watchlists, posições)
   - Utilizar `React.memo` e técnicas de memoização para componentes caros
2. **Processamento de dados:**
   - Mover cálculos pesados para Web Workers
   - Implementar throttling e buffering para atualizações de dados em tempo real
3. **Monitoramento de performance:**
   - Adicionar métricas de desempenho
   - Estabelecer benchmarks para operações críticas

**Justificativa:**

- Operações de day-trading são sensíveis à latência
- Grandes volumes de dados de mercado podem comprometer a responsividade da UI

## 7. Comunicação com APIs de Mercado

**Decisão futura:** Implementar uma camada de serviços para abstrair a comunicação com APIs de mercado.

**Considerações:**

- WebSockets para dados em tempo real

- Estratégias de cache para dados estáticos ou que mudam pouco
- Mecanismos de retry e fallback para resiliência

## Monitoramento e Revisão

Este ADR será revisitado regularmente durante o desenvolvimento para avaliar se as decisões tomadas continuam alinhadas com os objetivos do projeto e para incorporar aprendizados.

## Referências


- Electron Documentation
- React Performance Optimization
- TypeScript Documentation
- Lightweight Charts Documentation