Guia Completo: Sistema de Mercado de Previsões

Índice

O que é Mercado de Previsões

Mercado de previsões é uma plataforma onde pessoas apostam dinheiro em resultados de eventos futuros - políticos, esportivos, econômicos, etc. A ideia é que o "preço" de cada resultado reflita a probabilidade coletiva que as pessoas atribuem a ele acontecer.

Como Funciona
- Você compra "ações" de um resultado (ex: "Candidato X vai ganhar")
- Se acertar, recebe um valor fixo (geralmente R$1 ou US$1 por ação)
- Se errar, perde o que investiu
- Os preços flutuam conforme as apostas, como numa bolsa de valores

---

Como Funciona a Kalshi

A brasileira Luana Lopes Lara, aos 29 anos, se tornou a bilionária "self-made" mais jovem do mundo através da **Kalshi**, empresa que cofundou em 2018 com Tarek Mansour.

Funcionamento da Plataforma

A plataforma permite apostar em eventos reais através de contratos binários:
- Você compra ações de "Sim" ou "Não" para perguntas específicas
- O preço flutua entre 1 centavo e 1 dólar conforme a probabilidade do evento ocorrer
- Se acertar, recebe US$ 1 por contrato
- Se errar, perde o investimento

Tipos de Eventos
- Eleições e política
- Indicadores econômicos (inflação, taxa de juros)
- Esportes (atualmente 90% do volume)
- Clima e temperatura
- Cultura pop (Oscar, lançamentos musicais)

Diferença das Casas de Apostas Tradicionais

As odds não são definidas pela casa e sim por oferta e demanda. A Kalshi funciona como intermediadora, cobrando taxa por transação, enquanto os preços mudam conforme novas informações chegam ao mercado.

O Crescimento Bilionário

A Kalshi levantou US$ 1 bilhão a um valuation de US$ 11 bilhões em dezembro de 2024, na terceira rodada de capital do ano. A empresa multiplicou seu valor por 5x em poucos meses.

Arquitetura Técnica

Stack Tecnológico Recomendado

Backend
- **Node.js + TypeScript** ou **Go**: alta performance para matching de ordens
- **PostgreSQL**: dados transacionais, histórico de trades
- **Redis**: cache, filas de processamento, websockets
- **RabbitMQ/Kafka**: processamento assíncrono de ordens

Frontend
- **React + Next.js**: interface responsiva
- **TailwindCSS**: estilização rápida
- **WebSockets**: atualização em tempo real de preços
- **Chart.js/D3.js**: gráficos de preço

Infraestrutura
- **AWS/Google Cloud**: escalabilidade
- **Docker + Kubernetes**: containers
- **Nginx**: load balancer

Componentes Principais

1. Motor de Matching (Order Book)

```typescript
// Exemplo simplificado de order matching
class OrderBook {
  private buyOrders: Order[] = [];
  private sellOrders: Order[] = [];

  addOrder(order: Order) {
    if (order.type === 'BUY') {
      this.buyOrders.push(order);
      this.buyOrders.sort((a, b) => b.price - a.price); // Maior preço primeiro
    } else {
      this.sellOrders.push(order);
      this.sellOrders.sort((a, b) => a.price - b.price); // Menor preço primeiro
    }

    this.matchOrders();
  }

  private matchOrders() {
    while (this.buyOrders.length > 0 && this.sellOrders.length > 0) {
      const bestBuy = this.buyOrders[0];
      const bestSell = this.sellOrders[0];

      // Se preço de compra >= preço de venda, há match
      if (bestBuy.price >= bestSell.price) {
        const executionPrice = bestSell.price; // Preço do vendedor
```

```
        const quantity = Math.min(bestBuy.quantity, bestSell.quantity);

        this.executeTrade(bestBuy, bestSell, executionPrice, quantity);

        // Atualiza ou remove ordens
        bestBuy.quantity -= quantity;
        bestSell.quantity -= quantity;

        if (bestBuy.quantity === 0) this.buyOrders.shift();
        if (bestSell.quantity === 0) this.sellOrders.shift();
      } else {
        break; // Não há mais matches possíveis
      }
    }
  }
}
```

2. Sistema de Contratos (Mercados)

```typescript
interface Market {
  id: string;
  question: string;
  description: string;
  outcomes: ['SIM', 'NÃO']; // Binário
  endDate: Date;
  resolutionSource: string;
  status: 'OPEN' | 'CLOSED' | 'RESOLVED';
  totalVolume: number;
  currentYesPrice: number; // 0.00 a 1.00
}

class MarketContract {
  async createMarket(data: CreateMarketDTO): Promise<Market> {
    // Validações
    // Criar mercado no banco
    // Inicializar order book
    // Configurar resolução automática
  }

  async resolveMarket(marketId: string, outcome: 'SIM' | 'NÃO') {
    // Pagar vencedores (R$ 1.00 por contrato)
    // Zerar perdedores
    // Atualizar histórico
    // Calcular comissões
  }
}
```

3. Sistema de Carteira e Pagamentos

```typescript
class Wallet {
  async deposit(userId: string, amount: number) {
    // Integração com gateway (Stripe, Pagar.me, PicPay)
    // Atualizar saldo
    // Registrar transação
  }
```

```
  async withdraw(userId: string, amount: number) {
    // Verificar saldo disponível (não em posições abertas)
    // Processar saque via PIX/TED
    // Aplicar taxas
  }

  async getAvailableBalance(userId: string): Promise<number> {
    const total = await this.getTotalBalance(userId);
    const locked = await this.getLockedInPositions(userId);
    return total - locked;
  }
}
```

Mecanismo de Preços

Como o Preço Flutua

NÃO vem de órgão governamental ou API externa. O preço é determinado puramente por **oferta e demanda** entre os usuários da plataforma.

Exemplo Prático Passo a Passo

Mercado: "Flamengo vai ganhar o Brasileirão 2025?"

Momento 1 - Criação do mercado
```
Preço inicial SIM: R$ 0,50 (50% de probabilidade)
Preço inicial NÃO: R$ 0,50 (50% de probabilidade)
```

Momento 2 - Usuários começam a apostar
```
João quer comprar SIM e está disposto a pagar até R$ 0,55
Maria quer vender SIM por no mínimo R$ 0,60

Não há match! (comprador oferece menos que vendedor pede)
```

Order Book neste momento
```
VENDAS (SELL orders - SIM):
R$ 0,60 - 100 contratos (Maria)
R$ 0,62 - 50 contratos (Pedro)

COMPRAS (BUY orders - SIM):
R$ 0,55 - 200 contratos (João)
R$ 0,53 - 150 contratos (Ana)
```

Momento 3 - Match acontece
```
Carlos quer MUITO comprar SIM e aceita pagar R$ 0,60

MATCH! Carlos compra de Maria
Preço executado: R$ 0,60
```

```
Quantidade: 100 contratos

O último preço negociado agora é R$ 0,60 ← Preço atual
```

Momento 4 - Flamengo ganha 3 jogos seguidos
```
Muita gente quer comprar SIM agora!

Novos compradores:
- R$ 0,62 - 200 contratos
- R$ 0,65 - 300 contratos
- R$ 0,70 - 150 contratos

Preço sobe para R$ 0,70 (último trade executado)
```

Código do Motor de Matching

```typescript
class OrderMatchingEngine {
  private buyOrders: Order[] = [];  // Ordenado por preço DESC
  private sellOrders: Order[] = []; // Ordenado por preço ASC

  addOrder(order: Order) {
    // Adiciona ordem ao livro
    if (order.side === 'BUY') {
      this.buyOrders.push(order);
      this.buyOrders.sort((a, b) => b.price - a.price);
    } else {
      this.sellOrders.push(order);
      this.sellOrders.sort((a, b) => a.price - b.price);
    }

    this.matchOrders();
  }

  private matchOrders() {
    while (this.buyOrders.length > 0 && this.sellOrders.length > 0) {
      const bestBuy = this.buyOrders[0];
      const bestSell = this.sellOrders[0];

      if (bestBuy.price >= bestSell.price) {
        const executionPrice = bestSell.timestamp < bestBuy.timestamp
          ? bestSell.price
          : bestBuy.price;

        const quantity = Math.min(bestBuy.quantity, bestSell.quantity);

        this.executeTrade({
          buyerId: bestBuy.userId,
          sellerId: bestSell.userId,
          price: executionPrice,
          quantity: quantity,
          marketId: bestBuy.marketId
        });

        this.updateMarketPrice(bestBuy.marketId, executionPrice);
```

```
        bestBuy.quantity -= quantity;
        bestSell.quantity -= quantity;

        if (bestBuy.quantity === 0) this.buyOrders.shift();
        if (bestSell.quantity === 0) this.sellOrders.shift();

      } else {
        break;
      }
    }
  }

  private executeTrade(trade: TradeExecution) {
    // 1. Transfere dinheiro do comprador para escrow
    this.walletService.lock(trade.buyerId, trade.price * trade.quantity);

    // 2. Cria posição para o comprador
    this.positionService.addPosition({
      userId: trade.buyerId,
      marketId: trade.marketId,
      outcome: 'SIM',
      quantity: trade.quantity,
      avgPrice: trade.price
    });

    // 3. Remove posição do vendedor
    this.positionService.reducePosition({
      userId: trade.sellerId,
      marketId: trade.marketId,
      outcome: 'SIM',
      quantity: trade.quantity,
      salePrice: trade.price
    });

    // 4. Libera dinheiro para o vendedor
    this.walletService.credit(trade.sellerId, trade.price * trade.quantity);

    // 5. Registra o trade
    this.tradeRepository.save(trade);

    // 6. Notifica via WebSocket
    this.websocketService.broadcast({
      type: 'TRADE_EXECUTED',
      marketId: trade.marketId,
      price: trade.price,
      quantity: trade.quantity
    });
  }
}
```

Economia da Plataforma

De Onde Vem o Lucro da Plataforma

1. Taxa de Transação (Principal)

```typescript
// A cada trade executado
const tradeFee = tradeValue * 0.02; // 2% de taxa

// Exemplo:
// Trade de R$ 0,65 × 1000 contratos = R$ 650
// Taxa da plataforma: R$ 650 × 0.02 = R$ 13
```

Você ganha tanto faz quem vence! A taxa é cobrada no momento da transação.

2. Spread (Diferença Bid-Ask)

```
Melhor oferta de COMPRA: R$ 0,63
Melhor oferta de VENDA: R$ 0,67

Spread = R$ 0,04
```

3. Saldo Parado (Float)

Usuários depositam dinheiro na plataforma. Nem todo mundo usa 100% do saldo o tempo todo. Esse dinheiro parado pode render juros enquanto está na conta.

Como Funciona o Balanço (Soma Zero)

A plataforma sempre fica com saldo zero matematicamente. **O lucro de um é a perda do outro.**

Exemplo com Order Book

```
João quer comprar SIM a R$ 0,70
Maria quer vender SIM a R$ 0,70 (= comprar NÃO a R$ 0,30)

MATCH ACONTECE:

João paga: R$ 0,70
Maria paga: R$ 0,30
Total no escrow: R$ 1,00

Quando SIM vence:
- João recebe: R$ 1,00 (lucrou R$ 0,30)
- Maria recebe: R$ 0,00 (perdeu R$ 0,30)

Quando NÃO vence:
- João recebe: R$ 0,00 (perdeu R$ 0,70)
- Maria recebe: R$ 1,00 (lucrou R$ 0,70)
```

Matemática Fundamental

```
preçoSIM + preçoNÃO = R$ 1,00 SEMPRE

Se SIM está a R$ 0,65:
NÃO automaticamente está a R$ 0,35
```

Código de Resolução de Mercado

```typescript
class MarketResolution {
  async resolveMarket(marketId: string, outcome: 'SIM' | 'NÃO') {
    const positions = await this.getPositions(marketId);

    for (const position of positions) {
      if (position.outcome === outcome) {
        // VENCEDOR: recebe R$ 1,00 por contrato
        const payout = position.quantity * 1.00;
        await this.walletService.credit(position.userId, payout);

        const profit = payout - (position.avgPrice * position.quantity);

      } else {
        // PERDEDOR: recebe R$ 0,00
        // O dinheiro já foi travado quando comprou
      }
    }

    // Verificar balanço
    const totalPaid = await this.getTotalPayout(marketId);
    const totalLocked = await this.getTotalLocked(marketId);

    if (totalPaid !== totalLocked) {
      throw new Error('BALANÇO NÃO FECHA! Bug no sistema!');
    }
  }
}
```

Gerenciamento de Risco

```typescript
class RiskManagement {

  // 1. SEMPRE garantir que preçoSIM + preçoNÃO = 1.00
  validatePrices(simPrice: number, naoPrice: number) {
    if (Math.abs((simPrice + naoPrice) - 1.00) > 0.01) {
      throw new Error('Preços não somam R$ 1,00!');
    }
  }

  // 2. SEMPRE travar o dinheiro quando ordem é criada
  async createOrder(order: Order) {
    const requiredBalance = order.price * order.quantity;
    await this.walletService.lock(order.userId, requiredBalance);
    await this.orderBook.addOrder(order);
  }

  // 3. Taxa SEMPRE cobrada antecipadamente
  async chargeFee(trade: Trade) {
    const feeAmount = trade.value * 0.02;
    await this.walletService.deduct(trade.buyerId, feeAmount);
    await this.platformWallet.credit(feeAmount);
  }
}
```

Market Making (Para Garantir Liquidez)

```typescript
class MarketMaker {
  async maintainLiquidity(marketId: string) {
    const currentPrice = await this.getCurrentPrice(marketId);

    // Coloca ordens dos dois lados com pequeno spread
    await this.placeOrder({
      marketId,
      side: 'BUY',
      outcome: 'SIM',
      price: currentPrice - 0.02,
      quantity: 100
    });

    await this.placeOrder({
      marketId,
      side: 'SELL',
      outcome: 'SIM',
      price: currentPrice + 0.02,
      quantity: 100
    });
  }
}
```

Schema de Banco de Dados

```sql
-- Usuários
CREATE TABLE users (
  id UUID PRIMARY KEY,
  email VARCHAR UNIQUE,
  balance DECIMAL(10,2) DEFAULT 0,
  created_at TIMESTAMP
);

-- Mercados
CREATE TABLE markets (
  id UUID PRIMARY KEY,
  question TEXT,
  end_date TIMESTAMP,
  status VARCHAR,
  resolution VARCHAR, -- 'SIM', 'NÃO', NULL
  total_volume DECIMAL(12,2)
);

-- Posições dos usuários
CREATE TABLE positions (
  id UUID PRIMARY KEY,
  user_id UUID REFERENCES users(id),
  market_id UUID REFERENCES markets(id),
  outcome VARCHAR, -- 'SIM' ou 'NÃO'
  quantity INTEGER,
  avg_price DECIMAL(4,2),
```

```
  status VARCHAR -- 'OPEN', 'CLOSED'
);

-- Ordens (order book)
CREATE TABLE orders (
  id UUID PRIMARY KEY,
  user_id UUID,
  market_id UUID,
  type VARCHAR, -- 'BUY', 'SELL'
  outcome VARCHAR,
  price DECIMAL(4,2),
  quantity INTEGER,
  filled_quantity INTEGER DEFAULT 0,
  status VARCHAR, -- 'PENDING', 'FILLED', 'CANCELLED'
  created_at TIMESTAMP
);

-- Trades executados
CREATE TABLE trades (
  id UUID PRIMARY KEY,
  market_id UUID,
  buy_order_id UUID,
  sell_order_id UUID,
  price DECIMAL(4,2),
  quantity INTEGER,
  executed_at TIMESTAMP
);
```

Aspectos Legais

⚠️ CRÍTICO: Aspectos Legais no Brasil

Antes de qualquer linha de código, você precisa entender:

No Brasil, isso é complexo:
- Mercados de previsão podem ser enquadrados como "jogos de azar"
- Lei 14.790/2023 regulamentou apostas esportivas
- Mercados de previsão em eventos políticos/econômicos estão em zona cinzenta
- Necessita licença do Ministério da Fazenda
- Possivelmente autorização da CVM
- Operar sem licença = crimes contra a economia popular

Recomendação: Consulte um advogado especializado em direito digital e regulação financeira ANTES de desenvolver.

Legislação Atual (2025)

```
Lei 14.790/2023:
✓ Regulamenta apostas esportivas
✗ NÃO regulamenta mercados de previsão
✗ NÃO menciona blockchain/cripto

Código Penal (Art. 50):
"Estabelecer ou explorar jogo de azar em lugar público
ou acessível ao público"
```

Pena: 3 meses a 1 ano + multa
```

### Alternativas Legais

#### Opção 1: Play Money
```
Usuários apostam com pontos virtuais
Sem depósito/saque em Real
Apenas prêmios simbólicos
100% legal
```

#### Opção 2: Educacional/Acadêmico
```
Plataforma de "estudo de probabilidades"
Usuários simulam apostas
Dados usados para pesquisa
```

#### Opção 3: Aguardar Regulação
```
Lobby no Congresso
Proposta de lei específica
Obter licença quando sair regulação
Pode demorar 5-10 anos
```

---

## Implementação com Blockchain

### ⚠️ Blockchain NÃO te Deixa "Fora da Lei"

**Mito:** "Se usar blockchain e cripto, o governo não pode me regular"

**Realidade:** Você ainda está sujeito às leis do país onde você mora, seus usuários moram, sua empresa está registrada, e seus servidores estão hospedados.

### Casos Reais: Polymarket

- Em 2022, multada em US$ 1,4 milhão pela CFTC
- Forçada a bloquear usuários americanos
- Bloqueada na Austrália em 2025
- Bloqueada também em França, Bélgica, Polônia, Singapura, Tailândia e Romênia

**Por que blockchain não salvou:** Reguladores disseram "não é uma questão de tecnologia, mas sim de lei"

### Por Que Cripto NÃO te Deixa "Fora da Lei"

#### 1. Governos Bloqueiam IPs e DNS
```
Usuário brasileiro tenta acessar:
↓
ISP (Vivo, Claro) bloqueia DNS
↓
Site não carrega
↓

```
Usuário precisa VPN (maioria desiste)
```

2. Rampa de Entrada/Saída (Fiat)
```
Usuário quer depositar R$ 500
↓
Precisa converter Real → USDC
↓
Exchanges brasileiras são reguladas
↓
BC pode ordenar bloqueio de transações
```

3. Responsabilização Pessoal
```
Governo não pode "prender o smart contract"
MAS pode prender VOCÊ

Você mora no ES? Tem CPF?
↓
Polícia Federal
↓
Processo criminal
```

Smart Contract (Solidity)

```solidity
pragma solidity ^0.8.0;

contract PredictionMarket {
    struct Market {
        string question;
        uint256 endTime;
        uint256 totalYesShares;
        uint256 totalNoShares;
        bool resolved;
        bool outcome;
    }

    mapping(uint256 => Market) public markets;
    mapping(uint256 => mapping(address => uint256)) public yesPositions;
    mapping(uint256 => mapping(address => uint256)) public noPositions;

    IERC20 public usdc;

    function createMarket(string memory _question, uint256 _endTime) external {
        uint256 marketId = block.timestamp;
        markets[marketId] = Market({
            question: _question,
            endTime: _endTime,
            totalYesShares: 0,
            totalNoShares: 0,
            resolved: false,
            outcome: false
        });
    }
```

```
    function buyYes(uint256 marketId, uint256 amount) external {
        require(block.timestamp < markets[marketId].endTime, "Market closed");

        usdc.transferFrom(msg.sender, address(this), amount);

        yesPositions[marketId][msg.sender] += amount;
        markets[marketId].totalYesShares += amount;
    }

    function resolveMarket(uint256 marketId, bool outcome) external onlyOwner {
        require(!markets[marketId].resolved, "Already resolved");

        markets[marketId].resolved = true;
        markets[marketId].outcome = outcome;
    }

    function claim(uint256 marketId) external {
        require(markets[marketId].resolved, "Not resolved");

        uint256 payout = 0;

        if (markets[marketId].outcome) {
            payout = yesPositions[marketId][msg.sender];
            yesPositions[marketId][msg.sender] = 0;
        } else {
            payout = noPositions[marketId][msg.sender];
            noPositions[marketId][msg.sender] = 0;
        }

        usdc.transfer(msg.sender, payout);
    }
}
```

Frontend Web3

```typescript
import { ethers } from 'ethers';

async function connectWallet() {
  if (window.ethereum) {
    const provider = new ethers.providers.Web3Provider(window.ethereum);
    await provider.send("eth_requestAccounts", []);
    const signer = provider.getSigner();
    return signer;
  }
}

async function buyYesPosition(marketId, amount) {
  const signer = await connectWallet();

  // 1. Aprovar USDC
  const usdcContract = new ethers.Contract(USDC_ADDRESS, USDC_ABI, signer);
  await usdcContract.approve(MARKET_ADDRESS, amount);

  // 2. Comprar
  const marketContract = new ethers.Contract(MARKET_ADDRESS, MARKET_ABI, signer);
  await marketContract.buyYes(marketId, amount);
}
```

```

### Vantagens do Blockchain

**1. Transparência Total**
- Todas as transações são públicas
- Qualquer um pode auditar
- Impossível manipular resultados escondido

**2. Não-custodial**
- Usuário mantém fundos na própria carteira
- Você nunca "segura" o dinheiro deles
- Menos risco de hack

**3. Resistência à Censura (teórica)**
- Smart contract fica na blockchain
- Mesmo se site sair do ar, contrato continua
- Outros podem criar interfaces

### Desvantagens do Blockchain

**1. Experiência do Usuário**
```
95% das pessoas não sabem:
- Criar carteira MetaMask
- Comprar USDC
- Pagar gas fees
- Aprovar tokens
```

**2. Custos de Transação**
```
Ethereum: R$ 50-200 por transação (inviável)
Polygon: R$ 0,10-1,00 por transação (ok)
BSC: R$ 0,05-0,50 (ok)
```

**3. Velocidade**
```
Ethereum: 15 segundos por bloco
Polygon: 2 segundos por bloco
Centralizado: instantâneo
```

**4. Irreversibilidade**
```
Usuário mandou dinheiro errado? Perdeu.
Bug no smart contract? Dinheiro travado.
Não há "estorno"
```

### Arquitetura Híbrida (Melhor Opção)

```typescript
// BACKEND CENTRALIZADO (para UX e velocidade):
- Node.js + PostgreSQL
- Transações rápidas
- KYC/AML compliance
- Suporte ao cliente
```
```

```
// BLOCKCHAIN (para transparência):
- Snapshot diário dos saldos
- Merkle tree de posições
- Smart contract para disputas
- Prova de solvência

// Melhor dos dois mundos:
✓ Rápido como web2
✓ Auditável como web3
✓ Pode adicionar KYC
✗ Ainda precisa de regulação
```

Desafios e Custos

Desafios Técnicos Principais

Performance:
- Processar milhares de ordens por segundo
- Atualizar preços em tempo real via WebSocket
- Garantir atomicidade nas transações

Segurança:
- Prevenir front-running
- Evitar manipulação de mercado
- Proteção contra lavagem de dinheiro (KYC/AML)

Liquidez:
- Mercados novos têm poucos traders
- Pode precisar de market makers automatizados
- Spreads grandes no início

Custos e Monetização

Receita
- Taxa por transação (2-5% do volume)
- Spread entre compra/venda
- Saque (taxa fixa)

Custos Iniciais
- **Desenvolvimento**: R$ 150k - 500k (equipe de 3-6 devs por 6-12 meses)
- **Infraestrutura**: R$ 2k - 10k/mês
- **Legal/licenças**: R$ 50k - 200k+
- **Marketing**: variável

Resumo

Pontos Principais

1. **Preço flutua** = última transação executada entre usuários
2. **Plataforma não perde** = cobra taxa de cada transação (2-5%)
3. **Sistema fecha no zero** = perdas de uns pagam ganhos de outros
4. **Lucro da plataforma** = taxas + spread
5. **Código controla tudo** = nenhum dado externo define preço

Tecnologia

- **NÃO usa blockchain** necessariamente
- É um sistema web tradicional com order book
- Blockchain pode adicionar transparência mas não é obrigatório

Legalidade

- **Maior desafio não é técnico, é regulatório**
- Brasil não tem regulação clara para mercados de previsão
- Operar sem licença pode resultar em processo criminal
- Blockchain NÃO te protege da lei

Recomendação Final

Tecnicamente: Totalmente viável com tecnologias web tradicionais

Legalmente: Consulte advogado especializado ANTES de desenvolver

Alternativas seguras:
- Mercado play money (sem dinheiro real)
- Plataforma educacional
- Aguardar regulação específica

Fluxo de Funcionamento Completo

1. Criação de Conta e Depósito

```typescript
// Usuário se registra
class UserOnboarding {
  async register(email: string, password: string) {
    // 1. Criar conta
    const user = await db.users.create({
      email,
      passwordHash: await bcrypt.hash(password, 10),
      balance: 0,
      verified: false
    });

    // 2. Enviar email de verificação
    await emailService.sendVerification(user.email);

    // 3. KYC (se necessário por regulação)
    if (REQUIRES_KYC) {
      await kycService.initiateVerification(user.id);
    }

    return user;
  }

  async deposit(userId: string, amount: number) {
    // 1. Gerar QR Code PIX ou link pagamento
    const payment = await paymentGateway.createPayment({
      amount,
      userId,
```

```
      type: 'deposit'
    });

    // 2. Aguardar confirmação (webhook)
    // Quando pagamento confirmar:
    await db.users.increment('balance', amount, { where: { id: userId }});

    // 3. Notificar usuário
    await notificationService.send(userId, 'Depósito confirmado!');

    return payment;
  }
}
```

2. Navegação e Escolha de Mercado

```typescript
class MarketExplorer {
  async listMarkets(filters?: MarketFilters) {
    const markets = await db.markets.findAll({
      where: {
        status: 'OPEN',
        category: filters?.category,
        endDate: { $gt: new Date() }
      },
      order: [['totalVolume', 'DESC']]
    });

    // Adiciona preços atuais
    for (const market of markets) {
      const orderBook = await orderBookService.get(market.id);
      market.currentYesPrice = orderBook.lastPrice;
      market.spread = orderBook.bestAsk - orderBook.bestBid;
    }

    return markets;
  }

  async getMarketDetails(marketId: string) {
    const market = await db.markets.findOne({ id: marketId });

    // Histórico de preços
    const priceHistory = await db.trades
      .findAll({ marketId })
      .select('price', 'createdAt')
      .orderBy('createdAt', 'ASC');

    // Order book atual
    const orderBook = await orderBookService.get(marketId);

    // Volume nas últimas 24h
    const volume24h = await db.trades
      .where('marketId', marketId)
      .where('createdAt', '>', Date.now() - 86400000)
      .sum('price * quantity');

    return {
      ...market,
```

```
      priceHistory,
      orderBook,
      volume24h
    };
  }
}
```

3. Colocação de Ordem

```typescript
class OrderPlacement {
  async placeOrder(userId: string, orderData: OrderDTO) {
    // 1. Validações
    await this.validateOrder(userId, orderData);

    // 2. Verificar saldo disponível
    const requiredAmount = orderData.price * orderData.quantity;
    const user = await db.users.findOne({ id: userId });

    if (user.balance < requiredAmount) {
      throw new Error('Saldo insuficiente');
    }

    // 3. Travar o dinheiro
    await db.transaction(async (trx) => {
      await trx('users')
        .where({ id: userId })
        .decrement('balance', requiredAmount);

      await trx('locked_funds').insert({
        userId,
        amount: requiredAmount,
        orderId: generateId()
      });
    });

    // 4. Criar ordem
    const order = await db.orders.create({
      userId,
      marketId: orderData.marketId,
      side: orderData.side, // 'BUY' ou 'SELL'
      outcome: orderData.outcome, // 'SIM' ou 'NÃO'
      price: orderData.price,
      quantity: orderData.quantity,
      filledQuantity: 0,
      status: 'PENDING'
    });

    // 5. Adicionar ao order book
    await orderBookService.addOrder(order);

    // 6. Tentar fazer match imediatamente
    const matches = await orderMatchingEngine.processOrder(order);

    // 7. Notificar usuário dos matches
    if (matches.length > 0) {
      await notificationService.send(userId,
        `${matches.length} ordens executadas!`
```

```
      );
    }

    return { order, matches };
  }

  async validateOrder(userId: string, orderData: OrderDTO) {
    // Mercado está aberto?
    const market = await db.markets.findOne({ id: orderData.marketId });
    if (market.status !== 'OPEN') {
      throw new Error('Mercado fechado');
    }

    // Preço válido? (entre R$ 0,01 e R$ 0,99)
    if (orderData.price < 0.01 || orderData.price > 0.99) {
      throw new Error('Preço inválido');
    }

    // Quantidade mínima?
    if (orderData.quantity < 1) {
      throw new Error('Quantidade mínima é 1');
    }

    // Usuário não está banido?
    const user = await db.users.findOne({ id: userId });
    if (user.banned) {
      throw new Error('Usuário banido');
    }
  }
}
```

4. Execução de Trade

```typescript
class TradeExecution {
  async executeTrade(buyOrder: Order, sellOrder: Order) {
    const tradeId = generateId();
    const quantity = Math.min(buyOrder.quantity, sellOrder.quantity);
    const price = sellOrder.price; // Preço do vendedor

    await db.transaction(async (trx) => {
      // 1. Registrar trade
      await trx('trades').insert({
        id: tradeId,
        marketId: buyOrder.marketId,
        buyOrderId: buyOrder.id,
        sellOrderId: sellOrder.id,
        buyerId: buyOrder.userId,
        sellerId: sellOrder.userId,
        price,
        quantity,
        timestamp: new Date()
      });

      // 2. Atualizar ordens
      await trx('orders')
        .where({ id: buyOrder.id })
        .increment('filledQuantity', quantity);
```

```
  await trx('orders')
    .where({ id: sellOrder.id })
    .increment('filledQuantity', quantity);

  // 3. Criar/atualizar posições

  // Comprador ganha posição SIM
  await this.updatePosition(trx, {
    userId: buyOrder.userId,
    marketId: buyOrder.marketId,
    outcome: buyOrder.outcome,
    quantity: quantity,
    price: price
  });

  // Vendedor perde posição SIM (ou ganha NÃO)
  await this.updatePosition(trx, {
    userId: sellOrder.userId,
    marketId: sellOrder.marketId,
    outcome: sellOrder.outcome,
    quantity: -quantity,
    price: price
  });

  // 4. Movimentar fundos
  const totalAmount = price * quantity;

  // Libera fundos travados do comprador
  await trx('locked_funds')
    .where({ userId: buyOrder.userId, orderId: buyOrder.id })
    .decrement('amount', totalAmount);

  // Credita vendedor
  await trx('users')
    .where({ id: sellOrder.userId })
    .increment('balance', totalAmount);

  // 5. Cobrar taxa (2%)
  const fee = totalAmount * 0.02;
  await trx('users')
    .where({ id: buyOrder.userId })
    .decrement('balance', fee);

  await trx('platform_revenue').insert({
    tradeId,
    amount: fee,
    type: 'trading_fee'
  });

  // 6. Atualizar preço do mercado
  await trx('markets')
    .where({ id: buyOrder.marketId })
    .update({
      lastPrice: price,
      lastTradeAt: new Date(),
      totalVolume: db.raw('total_volume + ?', [totalAmount])
    });
});
```

```
    // 7. Broadcast via WebSocket
    await websocketService.broadcastToMarket(buyOrder.marketId, {
      type: 'TRADE',
      price,
      quantity,
      timestamp: new Date()
    });

    // 8. Notificar usuários
    await notificationService.send(buyOrder.userId,
      `Trade executado: ${quantity} contratos a R$ ${price}`
    );
    await notificationService.send(sellOrder.userId,
      `Trade executado: ${quantity} contratos a R$ ${price}`
    );

    return tradeId;
  }

  async updatePosition(trx, positionData) {
    const existing = await trx('positions').findOne({
      userId: positionData.userId,
      marketId: positionData.marketId,
      outcome: positionData.outcome
    });

    if (existing) {
      // Atualiza posição existente (média ponderada)
      const newQuantity = existing.quantity + positionData.quantity;
      const newAvgPrice = (
        (existing.avgPrice * existing.quantity) +
        (positionData.price * positionData.quantity)
      ) / newQuantity;

      await trx('positions')
        .where({ id: existing.id })
        .update({
          quantity: newQuantity,
          avgPrice: newAvgPrice
        });
    } else {
      // Cria nova posição
      await trx('positions').insert({
        userId: positionData.userId,
        marketId: positionData.marketId,
        outcome: positionData.outcome,
        quantity: positionData.quantity,
        avgPrice: positionData.price,
        status: 'OPEN'
      });
    }
  }
}
```

5. Resolução de Mercado

```typescript
```

```
class MarketResolutionService {
  async resolveMarket(marketId: string, outcome: 'SIM' | 'NÃO', source: string) {
    // 1. Verificar permissão (só admin pode resolver)
    // 2. Verificar se mercado já não foi resolvido
    const market = await db.markets.findOne({ id: marketId });

    if (market.resolved) {
      throw new Error('Mercado já resolvido');
    }

    if (market.endDate > new Date()) {
      throw new Error('Mercado ainda não terminou');
    }

    await db.transaction(async (trx) => {
      // 3. Atualizar mercado
      await trx('markets')
        .where({ id: marketId })
        .update({
          resolved: true,
          resolution: outcome,
          resolutionSource: source,
          resolvedAt: new Date()
        });

      // 4. Buscar todas as posições
      const positions = await trx('positions')
        .where({ marketId, status: 'OPEN' });

      let totalPayout = 0;

      // 5. Pagar vencedores
      for (const position of positions) {
        if (position.outcome === outcome) {
          // VENCEDOR: R$ 1,00 por contrato
          const payout = position.quantity * 1.00;
          totalPayout += payout;

          await trx('users')
            .where({ id: position.userId })
            .increment('balance', payout);

          await trx('positions')
            .where({ id: position.id })
            .update({
              status: 'WON',
              payout: payout,
              profit: payout - (position.avgPrice * position.quantity)
            });

          // Notificar
          await notificationService.send(position.userId,
            `🎉 Você ganhou! R$ ${payout.toFixed(2)}`
          );
        } else {
          // PERDEDOR: R$ 0,00
          await trx('positions')
            .where({ id: position.id })
            .update({
```

```
            status: 'LOST',
            payout: 0,
            profit: -(position.avgPrice * position.quantity)
          });

        await notificationService.send(position.userId,
          `😔 Você perdeu neste mercado`
        );
      }
    }

    // 6. Verificar balanço (segurança)
    const lockedFunds = await trx('locked_funds')
      .where({ marketId })
      .sum('amount');

    if (Math.abs(totalPayout - lockedFunds) > 0.01) {
      throw new Error(`ERRO CRÍTICO: Balanço não fecha!
        Payout: ${totalPayout}, Locked: ${lockedFunds}`);
    }

    // 7. Liberar fundos travados
    await trx('locked_funds')
      .where({ marketId })
      .delete();
  });

  // 8. Broadcast resolução
  await websocketService.broadcastToMarket(marketId, {
    type: 'MARKET_RESOLVED',
    outcome,
    resolvedAt: new Date()
  });

  return { success: true, outcome };
 }
}
```

6. Saque de Fundos

```typescript
class WithdrawalService {
 async requestWithdrawal(userId: string, amount: number, pixKey: string) {
  // 1. Validações
  const user = await db.users.findOne({ id: userId });

  if (user.balance < amount) {
    throw new Error('Saldo insuficiente');
  }

  // 2. Verificar se tem posições abertas
  const openPositions = await db.positions
    .where({ userId, status: 'OPEN' })
    .sum('quantity * avgPrice');

  const availableBalance = user.balance - openPositions;

  if (availableBalance < amount) {
```

```
    throw new Error(
      `Você tem R$ ${openPositions.toFixed(2)} em posições abertas.
       Disponível para saque: R$ ${availableBalance.toFixed(2)}`
    );
  }

  // 3. Taxa de saque (ex: R$ 2,00 fixo)
  const withdrawalFee = 2.00;
  const netAmount = amount - withdrawalFee;

  await db.transaction(async (trx) => {
    // 4. Debitar saldo
    await trx('users')
      .where({ id: userId })
      .decrement('balance', amount);

    // 5. Criar registro de saque
    const withdrawal = await trx('withdrawals').insert({
      userId,
      amount: netAmount,
      fee: withdrawalFee,
      pixKey,
      status: 'PENDING',
      requestedAt: new Date()
    });

    // 6. Registrar receita da plataforma (taxa)
    await trx('platform_revenue').insert({
      withdrawalId: withdrawal.id,
      amount: withdrawalFee,
      type: 'withdrawal_fee'
    });
  });

  // 7. Processar pagamento PIX (assíncrono)
  await paymentProcessor.processPix({
    pixKey,
    amount: netAmount,
    userId
  });

  // 8. Notificar
  await notificationService.send(userId,
    `Saque de R$ ${netAmount.toFixed(2)} em processamento`
  );

  return { success: true, netAmount };
}

async processPixCallback(withdrawalId: string, status: string) {
  // Webhook do gateway de pagamento

  await db.withdrawals
    .where({ id: withdrawalId })
    .update({
      status,
      processedAt: new Date()
    });
```

```
    const withdrawal = await db.withdrawals.findOne({ id: withdrawalId });

    if (status === 'SUCCESS') {
      await notificationService.send(withdrawal.userId,
        `✅ Saque concluído! R$ ${withdrawal.amount.toFixed(2)}`
      );
    } else if (status === 'FAILED') {
      // Devolver dinheiro
      await db.users
        .where({ id: withdrawal.userId })
        .increment('balance', withdrawal.amount + withdrawal.fee);

      await notificationService.send(withdrawal.userId,
        `❌ Saque falhou. Dinheiro devolvido à sua conta.`
      );
    }
  }
}
```

Sistema de Notificações em Tempo Real

WebSocket Server

```typescript
import { WebSocketServer } from 'ws';

class RealtimeService {
  private wss: WebSocketServer;
  private connections: Map<string, Set<WebSocket>> = new Map();

  constructor() {
    this.wss = new WebSocketServer({ port: 8080 });

    this.wss.on('connection', (ws, req) => {
      const userId = this.authenticateConnection(req);

      ws.on('message', (message) => {
        const data = JSON.parse(message.toString());
        this.handleMessage(ws, userId, data);
      });

      ws.on('close', () => {
        this.removeConnection(userId, ws);
      });
    });
  }

  handleMessage(ws: WebSocket, userId: string, data: any) {
    switch(data.type) {
      case 'SUBSCRIBE_MARKET':
        this.subscribeToMarket(ws, data.marketId);
        break;
      case 'UNSUBSCRIBE_MARKET':
        this.unsubscribeFromMarket(ws, data.marketId);
        break;
    }
```

```
  }

  subscribeToMarket(ws: WebSocket, marketId: string) {
    if (!this.connections.has(marketId)) {
      this.connections.set(marketId, new Set());
    }
    this.connections.get(marketId).add(ws);
  }

  broadcastToMarket(marketId: string, data: any) {
    const subscribers = this.connections.get(marketId);
    if (!subscribers) return;

    const message = JSON.stringify(data);

    for (const ws of subscribers) {
      if (ws.readyState === WebSocket.OPEN) {
        ws.send(message);
      }
    }
  }

  // Notificar atualização de preço
  async notifyPriceUpdate(marketId: string, price: number) {
    this.broadcastToMarket(marketId, {
      type: 'PRICE_UPDATE',
      marketId,
      price,
      timestamp: Date.now()
    });
  }

  // Notificar novo trade
  async notifyTrade(marketId: string, trade: Trade) {
    this.broadcastToMarket(marketId, {
      type: 'NEW_TRADE',
      marketId,
      price: trade.price,
      quantity: trade.quantity,
      timestamp: trade.timestamp
    });
  }

  // Notificar order book update
  async notifyOrderBookUpdate(marketId: string, orderBook: OrderBook) {
    this.broadcastToMarket(marketId, {
      type: 'ORDERBOOK_UPDATE',
      marketId,
      bids: orderBook.bids.slice(0, 10), // Top 10
      asks: orderBook.asks.slice(0, 10),
      timestamp: Date.now()
    });
  }
}
```

Dashboard e Analytics

Métricas da Plataforma

```typescript
class AnalyticsService {
  // Métricas em tempo real
  async getPlatformMetrics() {
    const [
      totalUsers,
      activeUsers24h,
      totalVolume,
      volume24h,
      totalMarkets,
      activeMarkets,
      totalTrades,
      trades24h
    ] = await Promise.all([
      db.users.count(),
      db.users.where('lastActive', '>', Date.now() - 86400000).count(),
      db.trades.sum('price * quantity'),
      db.trades.where('timestamp', '>', Date.now() - 86400000).sum('price * quantity'),
      db.markets.count(),
      db.markets.where('status', 'OPEN').count(),
      db.trades.count(),
      db.trades.where('timestamp', '>', Date.now() - 86400000).count()
    ]);

    return {
      users: { total: totalUsers, active24h: activeUsers24h },
      volume: { total: totalVolume, last24h: volume24h },
      markets: { total: totalMarkets, active: activeMarkets },
      trades: { total: totalTrades, last24h: trades24h }
    };
  }

  // Top traders
  async getTopTraders(limit = 10) {
    return db.query(`
      SELECT
        u.id,
        u.email,
        SUM(p.profit) as total_profit,
        COUNT(DISTINCT p.market_id) as markets_traded,
        COUNT(*) as total_positions
      FROM users u
      JOIN positions p ON p.user_id = u.id
      WHERE p.status IN ('WON', 'LOST')
      GROUP BY u.id
      ORDER BY total_profit DESC
      LIMIT ${limit}
    `);
  }

  // Mercados mais populares
  async getPopularMarkets(limit = 10) {
    return db.query(`
      SELECT
        m.*,
        COUNT(DISTINCT t.user_id) as unique_traders,
```

```
        SUM(t.price * t.quantity) as total_volume,
        COUNT(t.id) as trade_count
      FROM markets m
      LEFT JOIN trades t ON t.market_id = m.id
      WHERE m.status = 'OPEN'
      GROUP BY m.id
      ORDER BY total_volume DESC
      LIMIT ${limit}
    `);
  }

  // Performance de categorias
  async getCategoryPerformance() {
    return db.query(`
      SELECT
        m.category,
        COUNT(DISTINCT m.id) as market_count,
        SUM(t.price * t.quantity) as total_volume,
        AVG(t.price) as avg_price
      FROM markets m
      LEFT JOIN trades t ON t.market_id = m.id
      GROUP BY m.category
      ORDER BY total_volume DESC
    `);
  }
}
```

Segurança e Prevenção de Fraudes

Sistema Anti-Fraude

```typescript
class FraudDetectionService {
  // Detectar wash trading (usuário negociando consigo mesmo)
  async detectWashTrading(userId: string) {
    const suspiciousTrades = await db.query(`
      SELECT
        t.*,
        bo.user_id as buyer_id,
        so.user_id as seller_id
      FROM trades t
      JOIN orders bo ON t.buy_order_id = bo.id
      JOIN orders so ON t.sell_order_id = so.id
      WHERE bo.user_id = so.user_id
        AND bo.user_id = ?
      LIMIT 100
    `, [userId]);

    if (suspiciousTrades.length > 10) {
      await this.flagUser(userId, 'WASH_TRADING');
    }
  }

  // Detectar insider trading (usuário com informação privilegiada)
  async detectInsiderTrading(marketId: string) {
    // Verifica se alguém fez trades grandes logo antes da resolução
```

```
    const resolution = await db.markets.findOne({ id: marketId });
    const resolvedAt = resolution.resolvedAt;

    const suspiciousTrades = await db.query(`
      SELECT
        t.*,
        o.user_id,
        SUM(t.price * t.quantity) as trade_value
      FROM trades t
      JOIN orders o ON t.buy_order_id = o.id OR t.sell_order_id = o.id
      WHERE t.market_id = ?
        AND t.timestamp > ?
        AND t.timestamp < ?
      GROUP BY o.user_id
      HAVING trade_value > 1000
    `, [marketId, resolvedAt - 3600000, resolvedAt]); // 1h antes

    for (const trade of suspiciousTrades) {
      await this.flagUser(trade.user_id, 'POSSIBLE_INSIDER_TRADING');
    }
  }

  // Rate limiting
  async checkRateLimit(userId: string, action: string) {
    const key = `ratelimit:${userId}:${action}`;
    const count = await redis.incr(key);

    if (count === 1) {
      await redis.expire(key, 60); // 1 minuto
    }

    const limits = {
      'CREATE_ORDER': 100,
      'CANCEL_ORDER': 200,
      'API_CALL': 1000
    };

    if (count > limits[action]) {
      throw new Error('Rate limit exceeded');
    }
  }

  async flagUser(userId: string, reason: string) {
    await db.fraud_flags.insert({
      userId,
      reason,
      timestamp: new Date(),
      status: 'PENDING_REVIEW'
    });

    // Notificar admin
    await adminNotificationService.alert({
      type: 'FRAUD_DETECTION',
      userId,
      reason
    });
  }
}
```

Sistema de KYC (Know Your Customer)

```typescript
class KYCService {
  async verifyUser(userId: string, documents: KYCDocuments) {
    // 1. Validar documentos
    const validation = await documentValidator.validate({
      cpf: documents.cpf,
      rg: documents.rg,
      selfie: documents.selfie,
      proofOfAddress: documents.proofOfAddress
    });

    if (!validation.valid) {
      return { approved: false, reason: validation.errors };
    }

    // 2. Verificar em lista de sanções
    const sanctionCheck = await externalAPI.checkSanctions(documents.cpf);

    if (sanctionCheck.flagged) {
      await db.users.update({ id: userId }, {
        kycStatus: 'REJECTED',
        kycReason: 'SANCTIONS_LIST'
      });
      return { approved: false, reason: 'User in sanctions list' };
    }

    // 3. Verificação facial
    const faceMatch = await faceRecognitionAPI.compare(
      documents.rg_photo,
      documents.selfie
    );

    if (faceMatch.confidence < 0.95) {
      return { approved: false, reason: 'Face verification failed' };
    }

    // 4. Aprovar usuário
    await db.users.update({ id: userId }, {
      kycStatus: 'APPROVED',
      kycApprovedAt: new Date(),
      verified: true
    });

    return { approved: true };
  }
}
```

Monitoramento e Observabilidade

Logs e Alertas

```typescript
class MonitoringService {
```

```
  // Log estruturado
  async logEvent(event: LogEvent) {
    await logger.info({
      timestamp: new Date(),
      eventType: event.type,
      userId: event.userId,
      marketId: event.marketId,
      details: event.details,
      metadata: {
        ip: event.ip,
        userAgent: event.userAgent
      }
    });
  }

  // Métricas de performance
  async trackMetric(metric: Metric) {
    await prometheus.gauge(metric.name).set(metric.value);

    // Alertas automáticos
    if (metric.name === 'order_matching_latency' && metric.value > 1000) {
      await this.alert('HIGH_LATENCY', {
        metric: metric.name,
        value: metric.value,
        threshold: 1000
      });
    }
  }

  // Health check
  async healthCheck() {
    const checks = {
      database: await this.checkDatabase(),
      redis: await this.checkRedis(),
      orderBook: await this.checkOrderBook(),
      websockets: await this.checkWebSockets()
    };

    const healthy = Object.values(checks).every(c => c.status === 'ok');

    return {
      status: healthy ? 'healthy' : 'unhealthy',
      checks,
      timestamp: new Date()
    };
  }

  async checkDatabase() {
    try {
      await db.raw('SELECT 1');
      return { status: 'ok' };
    } catch (error) {
      return { status: 'error', message: error.message };
    }
  }
}
```

Testes Automatizados

Testes Unitários

```typescript
import { describe, it, expect, beforeEach } from 'vitest';

describe('OrderMatchingEngine', () => {
  let engine: OrderMatchingEngine;

  beforeEach(() => {
    engine = new OrderMatchingEngine();
  });

  it('should match buy and sell orders at correct price', async () => {
    const buyOrder = {
      id: '1',
      side: 'BUY',
      price: 0.65,
      quantity: 100,
      userId: 'user1'
    };

    const sellOrder = {
      id: '2',
      side: 'SELL',
      price: 0.60,
      quantity: 100,
      userId: 'user2'
    };

    await engine.addOrder(sellOrder);
    await engine.addOrder(buyOrder);

    const trades = await engine.getTrades();

    expect(trades).toHaveLength(1);
    expect(trades[0].price).toBe(0.60); // Preço do vendedor
    expect(trades[0].quantity).toBe(100);
  });

  it('should not match when bid < ask', async () => {
    const buyOrder = {
      side: 'BUY',
      price: 0.55,
      quantity: 100
    };

    const sellOrder = {
      side: 'SELL',
      price: 0.60,
      quantity: 100
    };

    await engine.addOrder(sellOrder);
    await engine.addOrder(buyOrder);

    const trades = await engine.getTrades();
```

```
    expect(trades).toHaveLength(0);
  });

  it('should handle partial fills', async () => {
    const buyOrder = {
      side: 'BUY',
      price: 0.65,
      quantity: 150
    };

    const sellOrder = {
      side: 'SELL',
      price: 0.60,
      quantity: 100
    };

    await engine.addOrder(sellOrder);
    await engine.addOrder(buyOrder);

    const trades = await engine.getTrades();
    expect(trades[0].quantity).toBe(100);

    const orderBook = await engine.getOrderBook();
    expect(orderBook.bids[0].quantity).toBe(50); // Restante
  });
});

describe('MarketResolution', () => {
  it('should pay winners correctly', async () => {
    const marketId = 'market1';
    const userId = 'user1';

    // Usuário comprou 100 SIM a R$ 0,70
    await db.positions.create({
      userId,
      marketId,
      outcome: 'SIM',
      quantity: 100,
      avgPrice: 0.70
    });

    await db.users.create({
      id: userId,
      balance: 0
    });

    // Resolve market como SIM
    await marketResolution.resolveMarket(marketId, 'SIM');

    const user = await db.users.findOne({ id: userId });
    expect(user.balance).toBe(100); // 100 contratos × R$ 1,00

    const position = await db.positions.findOne({ userId, marketId });
    expect(position.status).toBe('WON');
    expect(position.profit).toBe(30); // R$ 100 - R$ 70
  });

  it('should verify balance closes to zero', async () => {
    const marketId = 'market1';
```

```
    // Usuário 1: 100 SIM a R$ 0,70 = R$ 70
    await createPosition('user1', marketId, 'SIM', 100, 0.70);

    // Usuário 2: 100 NÃO a R$ 0,30 = R$ 30
    await createPosition('user2', marketId, 'NÃO', 100, 0.30);

    // Total locked: R$ 100

    await marketResolution.resolveMarket(marketId, 'SIM');

    // User1 recebe R$ 100
    // User2 recebe R$ 0
    // Total pago: R$ 100 ✓
  });
});
```

Testes de Integração

```typescript
describe('Complete Trading Flow', () => {
  it('should handle full trading lifecycle', async () => {
    // 1. Criar usuários
    const user1 = await createUser('user1@example.com');
    const user2 = await createUser('user2@example.com');

    // 2. Depositar fundos
    await wallet.deposit(user1.id, 100);
    await wallet.deposit(user2.id, 100);

    // 3. Criar mercado
    const market = await marketService.createMarket({
      question: 'Bitcoin vai passar de $100k?',
      endDate: new Date('2025-12-31')
    });

    // 4. User1 coloca ordem de compra
    await orderService.placeOrder(user1.id, {
      marketId: market.id,
      side: 'BUY',
      outcome: 'SIM',
      price: 0.70,
      quantity: 100
    });

    // 5. User2 coloca ordem de venda
    await orderService.placeOrder(user2.id, {
      marketId: market.id,
      side: 'SELL',
      outcome: 'SIM',
      price: 0.70,
      quantity: 100
    });

    // 6. Verificar que trade foi executado
    const trades = await db.trades.findAll({ marketId: market.id });
    expect(trades).toHaveLength(1);
    expect(trades[0].price).toBe(0.70);
```

```
    // 7. Verificar posições
    const pos1 = await db.positions.findOne({
      userId: user1.id,
      marketId: market.id
    });
    expect(pos1.quantity).toBe(100);
    expect(pos1.outcome).toBe('SIM');

    // 8. Verificar saldos
    const balance1 = await wallet.getBalance(user1.id);
    const balance2 = await wallet.getBalance(user2.id);

    expect(balance1).toBe(100 - 70 - 1.4); // depositou - gastou - taxa
    expect(balance2).toBe(100 + 70 - 1.4); // depositou + recebeu - taxa

    // 9. Resolver mercado
    await marketResolution.resolveMarket(market.id, 'SIM');

    // 10. Verificar pagamento final
    const finalBalance1 = await wallet.getBalance(user1.id);
    expect(finalBalance1).toBe(128.6); // recebeu R$ 100

    const finalBalance2 = await wallet.getBalance(user2.id);
    expect(finalBalance2).toBe(98.6); // não recebeu nada
  });
});
```

Otimizações de Performance

Caching Strategy

```typescript
class CacheService {
  private redis: Redis;

  // Cache de order book (atualizado a cada trade)
  async cacheOrderBook(marketId: string, orderBook: OrderBook) {
    await this.redis.setex(
      `orderbook:${marketId}`,
      10, // 10 segundos
      JSON.stringify(orderBook)
    );
  }

  async getOrderBook(marketId: string): Promise<OrderBook | null> {
    const cached = await this.redis.get(`orderbook:${marketId}`);
    return cached ? JSON.parse(cached) : null;
  }

  // Cache de preço atual
  async cachePrice(marketId: string, price: number) {
    await this.redis.setex(
      `price:${marketId}`,
      5, // 5 segundos
      price.toString()
```

```
    );
  }

  // Cache de mercados populares
  async cachePopularMarkets() {
    const markets = await db.markets
      .where('status', 'OPEN')
      .orderBy('totalVolume', 'DESC')
      .limit(20);

    await this.redis.setex(
      'popular_markets',
      300, // 5 minutos
      JSON.stringify(markets)
    );
  }
}
```

Database Indexing

```sql
-- Índices críticos para performance

-- Buscar ordens por mercado e preço
CREATE INDEX idx_orders_market_price
ON orders(market_id, price DESC, created_at);

-- Buscar trades por mercado e tempo
CREATE INDEX idx_trades_market_time
ON trades(market_id, executed_at DESC);

-- Buscar posições por usuário
CREATE INDEX idx_positions_user_status
ON positions(user_id, status);

-- Buscar mercados ativos
CREATE INDEX idx_markets_status_volume
ON markets(status, total_volume DESC);

-- Composite index para matching de ordens
CREATE INDEX idx_orders_matching
ON orders(market_id, outcome, side, price, status, created_at);
```

Query Optimization

```typescript
// BAD: N+1 queries
async function getMarketsWithPrices() {
  const markets = await db.markets.findAll();

  for (const market of markets) {
    market.currentPrice = await getLastPrice(market.id); // N queries!
  }

  return markets;
}
```

```
// GOOD: Single query com JOIN
async function getMarketsWithPricesOptimized() {
  return db.query(`
    SELECT
      m.*,
      (
        SELECT t.price
        FROM trades t
        WHERE t.market_id = m.id
        ORDER BY t.executed_at DESC
        LIMIT 1
      ) as current_price
    FROM markets m
    WHERE m.status = 'OPEN'
  `);
}
```

API Documentation

REST API Endpoints

```typescript
// GET /api/markets - Listar mercados
app.get('/api/markets', async (req, res) => {
  const { category, status, limit = 20, offset = 0 } = req.query;

  const markets = await marketService.listMarkets({
    category,
    status,
    limit: parseInt(limit),
    offset: parseInt(offset)
  });

  res.json({ markets });
});

// GET /api/markets/:id - Detalhes do mercado
app.get('/api/markets/:id', async (req, res) => {
  const market = await marketService.getMarket(req.params.id);

  if (!market) {
    return res.status(404).json({ error: 'Market not found' });
  }

  res.json({ market });
});

// POST /api/orders - Criar ordem
app.post('/api/orders', authenticateUser, async (req, res) => {
  const { marketId, side, outcome, price, quantity } = req.body;

  try {
    const order = await orderService.placeOrder(req.user.id, {
      marketId,
      side,
      outcome,
```

```
      price,
      quantity
    });

    res.json({ order });
  } catch (error) {
    res.status(400).json({ error: error.message });
  }
});

// DELETE /api/orders/:id - Cancelar ordem
app.delete('/api/orders/:id', authenticateUser, async (req, res) => {
  try {
    await orderService.cancelOrder(req.user.id, req.params.id);
    res.json({ success: true });
  } catch (error) {
    res.status(400).json({ error: error.message });
  }
});

// GET /api/positions - Minhas posições
app.get('/api/positions', authenticateUser, async (req, res) => {
  const positions = await positionService.getUserPositions(req.user.id);
  res.json({ positions });
});

// POST /api/wallet/deposit - Depositar
app.post('/api/wallet/deposit', authenticateUser, async (req, res) => {
  const { amount } = req.body;

  const payment = await walletService.createDeposit(req.user.id, amount);
  res.json({ payment });
});

// POST /api/wallet/withdraw - Sacar
app.post('/api/wallet/withdraw', authenticateUser, async (req, res) => {
  const { amount, pixKey } = req.body;

  try {
    const withdrawal = await walletService.requestWithdrawal(
      req.user.id,
      amount,
      pixKey
    );
    res.json({ withdrawal });
  } catch (error) {
    res.status(400).json({ error: error.message });
  }
});
```

Deployment e Infraestrutura

Docker Compose

```yaml
version: '3.8'
```

```
services:
  # Backend API
  api:
    build: ./backend
    ports:
      - "3000:3000"
    environment:
      DATABASE_URL: postgres://user:pass@db:5432/predmarket
      REDIS_URL: redis://redis:6379
      JWT_SECRET: ${JWT_SECRET}
    depends_on:
      - db
      - redis
    restart: unless-stopped

  # WebSocket Server
  websocket:
    build: ./websocket
    ports:
      - "8080:8080"
    environment:
      REDIS_URL: redis://redis:6379
    depends_on:
      - redis
    restart: unless-stopped

  # PostgreSQL Database
  db:
    image: postgres:15
    volumes:
      - postgres_data:/var/lib/postgresql/data
    environment:
      POSTGRES_DB: predmarket
      POSTGRES_USER: user
      POSTGRES_PASSWORD: pass
    restart: unless-stopped

  # Redis Cache
  redis:
    image: redis:7-alpine
    volumes:
      - redis_data:/data
    restart: unless-stopped

  # Frontend
  frontend:
    build: ./frontend
    ports:
      - "80:80"
    depends_on:
      - api
    restart: unless-stopped

volumes:
  postgres_data:
  redis_data:
```

Kubernetes Deployment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: predmarket-api
spec:
  replicas: 3
  selector:
    matchLabels:
      app: predmarket-api
  template:
    metadata:
      labels:
        app: predmarket-api
    spec:
      containers:
      - name: api
        image: predmarket/api:latest
        ports:
        - containerPort: 3000
        env:
        - name: DATABASE_URL
          valueFrom:
            secretKeyRef:
              name: predmarket-secrets
              key: database-url
        resources:
          requests:
            memory: "512Mi"
            cpu: "500m"
          limits:
            memory: "1Gi"
            cpu: "1000m"
        livenessProbe:
          httpGet:
            path: /health
            port: 3000
          initialDelaySeconds: 30
          periodSeconds: 10
---
apiVersion: v1
kind: Service
metadata:
  name: predmarket-api
spec:
  selector:
    app: predmarket-api
  ports:
  - port: 80
    targetPort: 3000
  type: LoadBalancer
```

Considerações Finais

Checklist de Lançamento

- [ ] **Legal**
  - [ ] Consultar advogado especializado
  - [ ] Verificar necessidade de licenças
  - [ ] Implementar termos de uso
  - [ ] Política de privacidade (LGPD)
  - [ ] Sistema de KYC/AML

- [ ] **Segurança**
  - [ ] Penetration testing
  - [ ] Auditoria de código
  - [ ] Rate limiting
  - [ ] Proteção DDoS
  - [ ] Backup automático

- [ ] **Performance**
  - [ ] Load testing (1000+ usuários simultâneos)
  - [ ] Otimização de queries
  - [ ] CDN para assets
  - [ ] Caching estratégico

- [ ] **Monitoramento**
  - [ ] Logs centralizados
  - [ ] Alertas automáticos
  - [ ] Dashboard de métricas
  - [ ] Tracking de erros (Sentry)

- [ ] **UX**
  - [ ] Tutorial interativo
  - [ ] Documentação clara
  - [ ] Suporte ao cliente
  - [ ] Mobile responsive

Roadmap Sugerido

Fase 1 - MVP (3-6 meses)
- Sistema básico de order matching
- Carteira e pagamentos
- 3-5 mercados iniciais
- Interface web simples

Fase 2 - Crescimento (6-12 meses)
- App mobile
- API pública
- Market making automatizado
- Mais categorias de mercados

Fase 3 - Escala (12-18 meses)
- Trading algorítmico
- Liquidez cross-platform
- Parcerias com provedores de dados
- Internacionalização

Recursos Adicionais

Referências Técnicas:
- [Kalshi Documentation](https://docs.kalshi.com)
- [Order Book Implementation](https://github.com/topics/order-book)

- [Financial Exchange Architecture](https://www.youtube.com/watch?v=b1e4t2k2KJY)

Comunidades:
- r/algotrading
- r/predictionmarkets
- Discord: Prediction Markets Developers

Conclusão

Este guia cobre todos os aspectos técnicos de construção de uma plataforma de mercado de previsões:

✅ **Arquitetura completa** - Backend, frontend, banco de dados
✅ **Mecanismo de preços** - Order matching, oferta e demanda
✅ **Economia da plataforma** - Como você lucra sem riscos
✅ **Segurança** - Anti-fraude, KYC, rate limiting
✅ **Performance** - Caching, indexing, otimizações
✅ **Deployment** - Docker, Kubernetes, produção

Lembre-se: O maior desafio NÃO é técnico - é regulatório. Consulte um advogado antes de lançar qualquer plataforma que envolva dinheiro real.

Boa sorte com seu projeto! 🚀