StratForge: AI-Powered No-Code Trading Platform

Design Document v1.0

---

1. Executive Summary

Project Name: StratForge

Purpose: A paper trading platform that combines traditional brokerage functionality with AI-powered market analysis and natural language-based, no-code trading strategy creation. Users can query AI about market movements and create sophisticated automated trading strategies without writing code.

Target Users: Finance-savvy individuals with semi-technical competence who want the analytical power of Interactive Brokers with the accessibility of Fidelity/Schwab.

Core Value Proposition: "Interactive Brokers for everyone" - institutional-grade strategy automation through natural language, powered by AI insights.

---

2. Product Overview

2.1 Problem Statement

Current trading platforms fall into two categories:

1. **Beginner-friendly (Fidelity, Schwab, Webull):** Easy to use but lack sophisticated strategy automation
2. **Advanced (Interactive Brokers):** Powerful automation but require programming knowledge

Gap: Finance-savvy traders who understand markets but don't want to code are underserved.

2.2 Solution

A paper trading platform that:

1. Provides traditional brokerage UI (charts, order entry, portfolio tracking)
2. Integrates AI assistant for market analysis and insights
3. Enables no-code strategy creation through natural language
4. Automatically executes strategies when conditions are met
5. Offers institutional-grade capabilities with consumer-grade UX

---

3. Core Features

3.1 Traditional Brokerage Interface

Portfolio Dashboard:

- Account balance and buying power

- Holdings with real-time P&L
- Open orders and order history
- Performance charts and statistics

Stock Detail View:

- Price charts (1D, 1W, 1M, 3M, 1Y, All)
- Technical indicators (customizable)
- Order entry (Market, Limit, Stop-Loss, Stop-Limit)
- Company fundamentals and news

Market Data:

- Watchlists (custom and curated)
- Market movers (gainers/losers)
- Sector performance
- 1-minute delayed data refresh

3.2 AI Market Assistant

Query Capabilities: Users can ask natural language questions like:

- "Why did Tesla stock go up 5% today?"
- "Give me a synopsis of market sentiment today"
- "What are analysts saying about the tech sector?"
- "Compare AAPL and MSFT performance this month"
- "Explain the recent Fed announcement's impact on my portfolio"

Response Format:

- **Analysis:** AI-generated interpretation of market data
- **Sources:** Citations to news articles, SEC filings, analyst reports
- **Data:** Relevant charts, price movements, volume data
- **Context:** Historical comparisons and related events

Data Sources:

1. **Gemini's Built-in Knowledge:** Up to yesterday's data (primary)
2. **RAG System (Optional):** Last 7 days of financial news via news API
3. **Real-time Market Data:** Current prices, volume, technical indicators from Alpaca
4. **Technical Analysis:** Calculated indicators (RSI, MACD, MA, etc.)

Technical Implementation:

- **Primary LLM:** Gemini (for up-to-date financial knowledge)
- **Market Data Cache:** 1-minute refresh cycle
- **News RAG:** Weekly vector DB update for recent news
- **Indicator Calculation:** Real-time via Python scripts (not AI-generated)

3.3 No-Code Strategy Builder

User Experience: Users describe strategies in natural language:

- "Set a stop loss when RSI drops below 30 and MACD shows bearish crossover"
- "Take profit if price rises 5% and volume is above average for 3 consecutive minutes"
- "Buy AAPL when it dips below 50-day MA and sentiment is positive"
- "Sell 50% position when 3 out of 5 indicators signal overbought"

LLM Processing:

1. User inputs natural language strategy description
2. Gemini parses requirements and generates Python script
3. Script includes:
    - Data fetching from Alpaca API
    - Indicator calculations using TA-Lib or pandas-ta
    - Conditional logic for entry/exit signals
    - Webhook call to StratForge API when triggered
4. User reviews generated script (shown in readable format)
5. User approves and activates strategy

Script Execution:

- Scripts run on **StratForge servers** (not user's computer)
- Continuous monitoring during market hours (9:30 AM - 4:00 PM ET)
- When conditions met → webhook to StratForge API → trade executed in user's Alpaca account
- Users can modify strategy (LLM regenerates script)
- Version history stored for each strategy

Supported Indicators: All common technical indicators via Python libraries:

- Trend: SMA, EMA, MACD, ADX
- Momentum: RSI, Stochastic, CCI, ROC
- Volatility: Bollinger Bands, ATR, Keltner Channels
- Volume: OBV, Volume MA, VWAP
- Custom: Any indicator calculable in Python

Multi-Strategy Support:

- Users can run multiple strategies simultaneously
- Each strategy can target different stocks
- Portfolio-level risk management rules
- Strategy priority/conflict resolution

3.4 Strategy Monitoring Dashboard

Active Strategies View:

- List of all active strategies
- Current status: Monitoring, Triggered, Paused, Error
- Last condition check timestamp
- Next evaluation time

- Quick actions: Pause, Edit, Delete

Performance Analytics:

- Strategy-specific P&L
- Win rate and average return
- Number of trades executed
- Sharpe ratio and max drawdown
- Comparison against buy-and-hold

Execution Logs: Detailed logs showing:

- Timestamp of each condition check
- Indicator values at check time
- Why trade was/wasn't triggered
- Execution details (price, quantity, slippage)
- Errors or warnings

Backtesting (Future Phase):

- Test strategy against historical data
- Hypothetical performance metrics
- Parameter optimization suggestions

---

4. Technical Architecture

4.1 System Architecture

```
┌─────────────────────────────────────────────────────────────────┐
│                        Frontend (React)                          │
│  ┌──────────────┐  ┌──────────────┐  ┌─────────────────────┐   │
│  │  Portfolio   │  │   AI Chat    │  │  Strategy Builder   │   │
│  │  Dashboard   │  │  Interface   │  │    Dashboard        │   │
│  └──────────────┘  └──────────────┘  └─────────────────────┘   │
└─────────────────────────────────────────────────────────────────┘
                              │
                              │ HTTPS/WebSocket
                              ▼
┌─────────────────────────────────────────────────────────────────┐
│                StratForge Backend (Node.js/Python)               │
│  ┌───────────────────────────────────────────────────────────┐  │
│  │                  API Gateway Layer                         │  │
│  │  • Authentication & Authorization                          │  │
│  │  • Rate Limiting & Request Validation                      │  │
│  │  • WebSocket Manager (live data streams)                   │  │
│  └───────────────────────────────────────────────────────────┘  │
│                                                                  │
│  ┌───────────────────────────────────────────────────────────┐  │
│  │                  Application Services                      │  │
│  │  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐    │  │
│  │  │   Trading    │  │ AI Assistant │  │   Strategy   │   │  │
│  │  │   Service    │  │   Service    │  │   Executor   │   │  │
│  │  └──────────────┘  └──────────────┘  └──────────────┘   │  │
│  └─────────────────────────────────────────────────────────┘  │
│                                                                │
│  ┌─────────────────────────────────────────────────────────┐  │
│  │                      Data Layer                           │  │
│  │  • PostgreSQL (users, strategies, trades, logs)           │  │
│  │  • Redis (caching, real-time data, job queues)            │  │
│  │  • Vector DB - Pinecone/Weaviate (optional news RAG)      │  │
│  └─────────────────────────────────────────────────────────┘  │
└───────────────────────────────────────────────────────────────┘
             │                    │                    │
             │                    │                    │
             ▼                    ▼                    ▼
    ┌──────────────────┐  ┌──────────────┐  ┌──────────────────┐
    │   Alpaca API     │  │  Gemini API  │  │    News APIs     │
    │ (Paper Trading)  │  │(AI Analysis) │  │ (Financial News) │
    └──────────────────┘  └──────────────┘  └──────────────────┘
```

4.2 Technology Stack

Frontend:

- **Framework:** React 18+ with TypeScript
- **State Management:** Redux Toolkit or Zustand
- **Charts:** TradingView Lightweight Charts or Recharts
- **Real-time Updates:** WebSocket connection to backend
- **Styling:** Tailwind CSS

Backend:

- **API Server:** Node.js with Express (REST) + Socket.io (WebSocket)
- **Strategy Executor:** Python (for script execution and TA calculations)
- **Task Queue:** Bull (Redis-based) for background jobs
- **Scheduler:** Node-cron for periodic tasks

Database:

- **Primary:** PostgreSQL 14+
  - Users, authentication
  - Strategies (metadata, code, status)
  - Trade history and portfolio data
  - Execution logs
- **Cache:** Redis 7+
  - Market data cache (1-min TTL)
  - User sessions
  - Job queue for strategy execution
- **Vector DB (Optional):** Pinecone or Weaviate
  - Financial news embeddings for RAG

External APIs:

- **Trading:** Alpaca API (paper trading)
- **AI:** Google Gemini API
- **News (Optional):** Alpha Vantage, Polygon.io, or NewsAPI
- **Market Data:** Alpaca (real-time for paper accounts)

Infrastructure:

- **Hosting:** AWS, Google Cloud, or DigitalOcean
- **Container Orchestration:** Docker + Docker Compose (POC) → Kubernetes (Production)
- **CI/CD:** GitHub Actions
- **Monitoring:** Sentry (errors), Grafana + Prometheus (metrics)

4.3 Data Flow

AI Query Flow:

```
1. User types: "Why did TSLA go up today?"
2. Frontend → Backend API: /api/ai/query
3. Backend fetches TSLA current data from cache/Alpaca
4. Backend constructs prompt with:
   - User question
   - Current TSLA price, volume, % change
   - Technical indicators
   - Instruction to cite sources
5. Backend → Gemini API: Structured prompt
6. Gemini responds with analysis + sources
7. Backend optionally enriches with RAG news data
8. Backend → Frontend: Formatted response with citations
9. Frontend renders in chat interface
```

Strategy Creation Flow:

```
1. User inputs: "Stop loss when RSI < 30 and MACD bearish"
2. Frontend → Backend: /api/strategies/create
3. Backend → Gemini API:
   - Prompt: "Generate Python script for this strategy..."
   - Context: Available indicators, Alpaca API format, webhook endpoint
4. Gemini returns Python script
5. Backend validates script (syntax check, security scan)
6. Backend stores script in database (inactive)
7. Backend → Frontend: Script preview + explanation
8. User reviews and clicks "Activate"
9. Frontend → Backend: /api/strategies/:id/activate
10. Backend adds script to execution queue (Redis)
11. Python worker picks up script, starts monitoring
```

Strategy Execution Flow:

```
1. Python worker runs user's strategy script every 1 minute
2. Script fetches latest data from Alpaca API
3. Script calculates indicators using TA-Lib
4. Script evaluates conditions
5. If conditions met:
   a. Script calls StratForge webhook: POST /api/webhooks/trade-signal
   b. Payload: {user_id, strategy_id, symbol, action, quantity}
6. Backend receives webhook
7. Backend validates signal (user still owns strategy, within limits)
8. Backend → Alpaca API: Place order
9. Backend logs execution in database
10. Backend → Frontend (WebSocket): Real-time notification
11. Frontend updates UI: Order placed, strategy triggered
```

4.4 Market Data Architecture

Caching Strategy: Since market data updates every minute:

- **Redis Cache:** Store last 1 minute of OHLCV data per symbol
- **TTL:** 60 seconds
- **Update Trigger:** Background job polls Alpaca every 60 seconds
- **On Query:** Check cache first, fetch from Alpaca if miss

Why Not Vector DB for Prices?

- Prices change too frequently for vector DB (designed for static embeddings)
- Redis is faster for time-series data with TTL
- Vector DB reserved for news articles (update weekly)

Data Refresh Process:

```
Every 60 seconds during market hours (9:30 AM - 4:00 PM ET):
1. Cron job triggers market data refresh
2. Fetch latest bars for all active watchlist symbols
3. Update Redis cache with new OHLCV data
4. Broadcast updates to connected clients via WebSocket
5. Strategy scripts automatically use latest data on next check
```

---

5. User Experience Design

5.1 User Onboarding

Step 1: Account Creation

- Email/password registration
- Email verification
- Quick profile setup (experience level, interests)

Step 2: Alpaca Connection

- "Connect your Alpaca paper trading account"
- Instructions: "Create free Alpaca paper account at alpaca.markets"
- Input: Alpaca API Key + Secret
- Validation + test connection
- Initial portfolio sync

Step 3: Feature Tour

- Interactive walkthrough of:
    1. Portfolio dashboard
    2. AI assistant chat
    3. Strategy builder
    4. Example queries and strategies
- "Start Trading" CTA

5.2 Main Interface Layout

Top Navigation Bar:

- Logo + App name
- Search bar (stock symbols)
- Watchlist dropdown
- Account menu (balance, settings, logout)

Left Sidebar:

- Dashboard
- Markets (movers, sectors)
- Portfolio
- Strategies (active/paused)
- AI Assistant (chat icon)
- Settings

Main Content Area: Dynamically changes based on selection:

- **Dashboard:** Portfolio summary + market overview + recent activity
- **Stock Detail:** Charts + order entry + AI insights
- **Strategy Builder:** Natural language input + script preview + management
- **AI Chat:** Conversational interface with market queries

Right Sidebar (Collapsible):

- Watchlist
- Open orders
- Recent alerts/notifications

5.3 AI Assistant Interface

Chat Window:

- Clean, modern chat UI (similar to ChatGPT/Claude)
- User messages on right, AI on left
- Support for:
  - Text queries
  - Charts embedded in responses
  - Source citations (clickable links)
  - Follow-up questions
- Suggested queries: "Try asking: Why did the market drop today?"

Query Examples Display: Pre-populated suggestions:

- "What's driving the tech sector today?"
- "Compare AMD vs NVDA this week"
- "Explain the latest Fed decision"
- "Should I be worried about my AAPL position?"

Response Format:

```
AI: Based on today's trading, Tesla (TSLA) rose 5.2% primarily due to:

📈 Key Factors:
1. Better-than-expected Q4 delivery numbers announced this morning
2. Analyst upgrade from Morgan Stanley (price target $350 → $400)
3. Broader EV sector strength (+3.1% average)

📊 Technical View:
- Broke resistance at $245 with strong volume (2.1M above average)
- RSI now at 68 (approaching overbought but not extreme)

📰 Sources:
- Reuters: "Tesla Q4 deliveries beat estimates" [link]
- Morgan Stanley Research Note (Jan 15) [link]

Current price: $252.30 (+$12.45, +5.2%)
```

5.4 Strategy Builder Interface

Input Section:

- Large text area: "Describe your trading strategy in plain English"
- Placeholder examples shown
- Character limit: 500 words
- "Generate Strategy" button

Generated Script Preview:

- Split view:
  - **Left:** Natural language explanation of what script does
  - **Right:** Actual Python code (collapsible, syntax highlighted)

- Confidence indicator: "High Confidence" / "Review Recommended"
- Edit button: Modify description → regenerate

Configuration Options:

- Symbol(s) to apply strategy
- Position size ($ amount or % of portfolio)
- Max trades per day
- Active hours (market hours or after-hours)

Activation:

- "Test Strategy" (dry run, shows what would have triggered)
- "Activate Strategy" (starts live monitoring)
- Warning: "This strategy will execute trades automatically"

Strategy Card (in dashboard):

```
┌─────────────────────────────────────────┐
│ 🎯 RSI + MACD Mean Reversion             │
│ Symbol: AAPL | Status: ● Active          │
│ ─────────────────────────────────────    │
│ Last checked: 2 mins ago                 │
│ Trades executed: 3 | P&L: +$142.50 (+1.2%) │
│ Next check: 58 seconds                   │
│ ─────────────────────────────────────    │
│ [View Logs] [Pause] [Edit] [Delete]     │
└─────────────────────────────────────────┘
```

---

6. Security & Compliance

6.1 Authentication & Authorization

- **User Auth:** JWT tokens with refresh mechanism
- **Alpaca API Keys:** Encrypted at rest (AES-256), stored per user
- **API Rate Limiting:** Per user and global limits
- **Role-based Access:** User, Admin roles for future expansion

6.2 Strategy Script Security

Sandboxing:

- Python scripts run in isolated environments (Docker containers)
- Resource limits: CPU (1 core), Memory (512MB), execution time (30s per run)
- Network restrictions: Only allow Alpaca API and StratForge webhook

Code Validation:

- Static analysis for dangerous imports (os, subprocess, eval, exec)

- Blacklist forbidden modules
- Syntax validation before storage
- Whitelist allowed libraries: pandas, numpy, ta-lib, requests (for webhook)

Execution Monitoring:

- Log all script executions
- Alert on repeated failures or suspicious behavior
- Kill switch for runaway scripts

6.3 Financial Safeguards

Paper Trading Only (POC):

- No real money at risk
- Clearly labeled as "Paper Trading" throughout UI
- Disclaimer on signup

Risk Limits (Future):

- Max position size per strategy
- Portfolio-level exposure limits
- Daily loss limits with auto-pause

Data Privacy:

- Users' strategies and trades are private (not shared)
- No selling of trading data
- Transparent data usage policy

6.4 Compliance Considerations

Disclaimers:

- "StratForge is for educational and simulation purposes"
- "Not financial advice - consult a licensed advisor"
- "Past performance does not guarantee future results"

Future Regulatory Prep:

- If moving to real trading: SEC registration, broker-dealer license
- Terms of Service reviewed by legal counsel
- Data retention policies for audit trails

---

7. Implementation Roadmap

Phase 1: Core Platform (8-10 weeks)

Weeks 1-2: Infrastructure Setup

- Project scaffolding (React frontend, Node.js backend, Python executor)

- Database schema design and setup (PostgreSQL)
- Docker development environment
- Basic authentication system

Weeks 3-4: Alpaca Integration + Portfolio

- Alpaca API client implementation
- User API key storage and management
- Portfolio dashboard (balances, holdings, P&L)
- Basic order entry (market orders only)

Weeks 5-6: Market Data & Charts

- Market data caching system (Redis)
- Background job for 1-min data refresh
- Stock detail page with charts (TradingView Lightweight Charts)
- Watchlist functionality

Weeks 7-8: AI Assistant (Basic)

- Gemini API integration
- Chat interface implementation
- Simple queries: "Why did [symbol] move today?"
- Market data context injection into prompts

Weeks 9-10: Testing & Polish

- End-to-end testing of core flows
- UI/UX refinements
- Performance optimization
- Documentation

Phase 2: Strategy Builder (6-8 weeks)

Weeks 1-2: Natural Language Processing

- Strategy description input UI
- Gemini prompt engineering for script generation
- Python script validation logic
- Script preview interface

Weeks 3-4: Strategy Execution Engine

- Python worker for script execution
- Redis job queue implementation
- Indicator calculation (TA-Lib integration)
- Webhook receiver for trade signals

Weeks 5-6: Strategy Management

- Activate/pause/delete strategies
- Strategy dashboard and cards

- Execution logs and history
- Error handling and notifications

Weeks 7-8: Multi-Strategy Support

- Run multiple strategies concurrently
- Conflict resolution logic
- Portfolio-level risk checks
- Performance testing (handle 100+ active strategies)

Phase 3: Advanced Features (4-6 weeks)

Weeks 1-2: Enhanced AI

- RAG system for financial news (optional)
- Source citations in AI responses
- More complex queries (multi-stock comparisons, sentiment analysis)
- Chat history and conversation threads

Weeks 3-4: Strategy Analytics

- Performance metrics per strategy
- Backtesting framework (basic)
- Strategy comparison tools
- Export reports (PDF/CSV)

Weeks 5-6: UX Enhancements

- Real-time WebSocket updates for trades
- Mobile-responsive design
- Dark mode
- Onboarding tutorial improvements

Phase 4: Beta Testing & Refinement (4 weeks)

- Private beta with 50-100 users
- Collect feedback and analytics
- Bug fixes and stability improvements
- Load testing and scaling prep

Total Timeline: 22-28 weeks (5.5-7 months) for production-ready POC

---

8. Success Metrics

Primary KPIs

- **User Activation Rate:** % of signups who connect Alpaca and place first trade (target: 60%+)
- **Strategy Creation Rate:** % of users who create at least one strategy (target: 40%+)
- **AI Query Engagement:** Average queries per user per session (target: 3+)
- **Strategy Success Rate:** % of strategies that execute trades as intended (target: 95%+)

Secondary KPIs

- Daily active users (DAU) and monthly active users (MAU)
- Average strategies per active user (target: 2-3)
- Time spent in app per session (target: 15+ minutes)
- User retention (D7, D30, D90)

Operational Metrics

- API response times (p95 < 200ms)
- Strategy execution latency (< 10 seconds from signal to order)
- System uptime (target: 99.5%+)
- Error rates (< 0.5% of requests)

Qualitative Feedback

- User satisfaction surveys (NPS score)
- Feature requests and pain points
- Comparison to competitors (usability testing)

---

9. Risk Analysis

Risk	Impact	Likelihood	Mitigation
Alpaca API rate limits exceeded	High	Medium	Implement request caching, rate limiting, backoff strategies
Gemini API costs spiral	Medium	Medium	Set monthly budget caps, optimize prompts, cache common queries
User-generated scripts cause server issues	High	Low	Strict sandboxing, resource limits, code validation
Strategy execution delays (>1 min)	Medium	Medium	Optimize Python execution, scale workers horizontally
Users expect real trading immediately	Medium	High	Clear "paper trading only" messaging, manage expectations
Security breach of Alpaca API keys	Critical	Low	Encryption at rest/transit, regular security audits, key rotation
Low user adoption due to complexity	High	Medium	Simplify onboarding, better tutorials, responsive support
Competitor launches similar product	Medium	Medium	Rapid iteration, unique AI features, community building

---

10. Open Questions & Future Considerations

Technical Decisions

1. **Vector DB for News RAG:** Implement now or defer?
    - **Recommendation:** Defer to Phase 3, use Gemini's built-in knowledge first
2. **Backtesting Framework:** Build custom or integrate library (Backtrader)?
    - **Recommendation:** Start with Backtrader integration (faster)
3. **Mobile Apps:** Native (iOS/Android) or responsive web?
    - **Recommendation:** Responsive web first, native apps in later phase
4. **Multi-Account Support:** Allow users to manage multiple Alpaca accounts?
    - **Use case:** Users testing different strategies in separate accounts
    - **Recommendation:** Single account for POC, add later if requested

Business Questions

5. **Monetization Strategy:**
    - Options: Subscription tiers, pay-per-strategy, freemium with limits
    - **Recommendation:** Decide after beta feedback, start free for POC
6. **Real Money Trading:**
    - Regulatory requirements (broker-dealer license, capital requirements)
    - Timeline: 12-18 months post-POC if pursuing
    - **Recommendation:** Focus on paper trading excellence first
7. **Educational Content:**
    - Integrate tutorials, courses, or market education?
    - **Recommendation:** Yes, differentiates from pure brokerages
8. **Social Features:**
    - Share strategies, leaderboards, follow top performers?
    - **Recommendation:** Explore in Phase 4+ after core product solid

---

11. Appendix

A. User Personas

Primary Persona: Alex (Finance Enthusiast)

- Age: 28-35
- Background: Works in tech/finance, reads WSJ/Bloomberg
- Tech comfort: High (uses Python occasionally, understands APIs)
- Trading experience: 2-3 years, mostly manual trading
- Pain point: Spends too much time watching charts, wants automation
- Goal: Create strategies without becoming a full-time algo trader

Secondary Persona: Jamie (Career Switcher)

- Age: 40-50
- Background: Mid-career professional exploring finance
- Tech comfort: Medium (Excel power user, no coding)
- Trading experience: 6 months, uses Robinhood/Webull
- Pain point: Overwhelmed by complex platforms like Interactive Brokers
- Goal: Learn algorithmic trading concepts without coding

B. Competitive Analysis

Feature	StratForge	Interactive Brokers	Webull	TradingView
No-code strategies	✅ Natural language	❌ Requires coding	⚠ Limited	⚠ Pine Script
AI market analysis	✅ Integrated	❌ None	❌ None	❌ None
Paper trading	✅ Free unlimited	✅ Yes	✅ Yes	✅ Yes
Technical indicators	✅ All common	✅ Extensive	⚠ Basic	✅ Extensive
Learning curve	🟢 Low	🔴 High	🟡 Medium	🟡 Medium
Real trading	❌ POC only	✅ Yes	✅ Yes	⚠ Via brokers

Key Differentiator: Only platform combining natural language strategy creation with AI-powered market insights.

C. Technical Stack Justification

Why React?

- Large ecosystem, excellent for financial UIs (recharts, TradingView widgets)
- Strong community support and hiring pool
- TypeScript support for type safety

Why Node.js + Python hybrid?

- Node.js: Fast API responses, WebSocket support, JavaScript ecosystem
- Python: Superior for financial calculations (pandas, TA-Lib, NumPy), LLM integrations

Why PostgreSQL?

- ACID compliance for financial data
- Complex queries for analytics
- JSON support for flexible strategy storage

Why Redis?

- Sub-millisecond caching for real-time data
- Built-in pub/sub for WebSocket broadcasts
- Job queue for background tasks (Bull)

Why Alpaca?

- Free paper trading with real-time data
- Well-documented REST + WebSocket APIs
- Commission-free trading (if moving to real)
- Fractional shares support

D. Example Strategy Scripts

Example 1: Simple RSI Mean Reversion

```
import alpaca_trade_api as tradeapi
import pandas_ta as ta
import requests

def check_strategy():
    # Fetch data
    api = tradeapi.REST(user_api_key, user_api_secret, base_url='paper')
    bars = api.get_bars('AAPL', '1Min', limit=30).df

    # Calculate RSI
    bars['rsi'] = ta.rsi(bars['close'], length=14)
    current_rsi = bars['rsi'].iloc[-1]

    # Check condition
    if current_rsi < 30:
        # Signal BUY
        requests.post('https://stratforge.com/api/webhooks/trade-signal',
json={
            'user_id': 'user_123',
            'strategy_id': 'strat_456',
            'symbol': 'AAPL',
            'action': 'buy',
            'quantity': 10,
            'reason': f'RSI oversold at {current_rsi:.2f}'
        })

check_strategy()
```

Example 2: Multi-Indicator Strategy

```
def check_complex_strategy():
    # Fetch and prepare data
    bars = get_latest_bars('TSLA', 60)  # 60 minutes

    # Calculate indicators
    rsi = ta.rsi(bars['close'], 14)
    macd = ta.macd(bars['close'])
    bb = ta.bbands(bars['close'], 20)
    volume_ma = bars['volume'].rolling(20).mean()
    # Check 3 out of 5 conditions
    conditions_met = 0

    if rsi.iloc[-1] < 30:
        conditions_met += 1
    if macd['MACD_12_26_9'].iloc[-1] < macd['MACDs_12_26_9'].iloc[-1]:
        conditions_met += 1
    if bars['close'].iloc[-1] < bb['BBL_20_2.0'].iloc[-1]:
        conditions_met += 1
    if bars['volume'].iloc[-1] > volume_ma.iloc[-1] * 1.5:
        conditions_met += 1
    if bars['close'].iloc[-1] < bars['close'].iloc[-2]:
        conditions_met += 1

    if conditions_met >= 3:
        send_trade_signal('buy', 'TSLA', 5)
```

E. API Endpoints Summary

Authentication:

- `POST /api/auth/register` - Create account
- `POST /api/auth/login` - Login
- `POST /api/auth/refresh` - Refresh JWT token

Alpaca Integration:

- `POST /api/alpaca/connect` - Save API keys
- `GET /api/alpaca/account` - Get account info
- `GET /api/portfolio` - Get portfolio summary

Market Data:

- `GET /api/market/quote/:symbol` - Get current price
- `GET /api/market/bars/:symbol` - Get historical bars
- `GET /api/market/movers` - Get top gainers/losers

AI Assistant:

- `POST /api/ai/query` - Send query to AI
- `GET /api/ai/history` - Get chat history

Strategies:

- `POST /api/strategies` - Create new strategy
- `GET /api/strategies` - List user's strategies
- `PUT /api/strategies/:id` - Update strategy
- `POST /api/strategies/:id/activate` - Activate strategy
- `POST /api/strategies/:id/pause` - Pause strategy
- `DELETE /api/strategies/:id` - Delete strategy

- `GET /api/strategies/:id/logs` - Get execution logs

Trading:

- `POST /api/orders` - Place order
- `GET /api/orders` - List orders
- `DELETE /api/orders/:id` - Cancel order

Webhooks:

- `POST /api/webhooks/trade-signal` - Receive trade signal from strategy script

Document Control

Version: 1.0
Last Updated: January 2026
Owner: [Project Team]
Status: Ready for Review
Next Review: After Phase 1 completion