**Forex & Crypto Trading Journal - Technical Documentation**

**Table of Contents**

1. [Project Overview](https://claude.ai/chat/82878c2d-a9e5-47f3-b351-e9c11f9f82ae#project-overview)
2. [System Architecture](https://claude.ai/chat/82878c2d-a9e5-47f3-b351-e9c11f9f82ae#system-architecture)
3. [Backend Implementation (Python)](https://claude.ai/chat/82878c2d-a9e5-47f3-b351-e9c11f9f82ae#backend-implementation-python)
   * [Technology Stack](https://claude.ai/chat/82878c2d-a9e5-47f3-b351-e9c11f9f82ae#backend-technology-stack)
   * [Database Schema](https://claude.ai/chat/82878c2d-a9e5-47f3-b351-e9c11f9f82ae#database-schema)
   * [API Endpoints](https://claude.ai/chat/82878c2d-a9e5-47f3-b351-e9c11f9f82ae#api-endpoints)
   * [Authentication & Authorization](https://claude.ai/chat/82878c2d-a9e5-47f3-b351-e9c11f9f82ae#authentication--authorization)
   * [Data Processing](https://claude.ai/chat/82878c2d-a9e5-47f3-b351-e9c11f9f82ae#data-processing)
   * [AI Analysis Module](https://claude.ai/chat/82878c2d-a9e5-47f3-b351-e9c11f9f82ae#ai-analysis-module)
4. [Frontend Implementation (ReactJS)](https://claude.ai/chat/82878c2d-a9e5-47f3-b351-e9c11f9f82ae#frontend-implementation-reactjs)
   * [Technology Stack](https://claude.ai/chat/82878c2d-a9e5-47f3-b351-e9c11f9f82ae#frontend-technology-stack)
   * [Component Structure](https://claude.ai/chat/82878c2d-a9e5-47f3-b351-e9c11f9f82ae#component-structure)
   * [State Management](https://claude.ai/chat/82878c2d-a9e5-47f3-b351-e9c11f9f82ae#state-management)
   * [UI/UX Design](https://claude.ai/chat/82878c2d-a9e5-47f3-b351-e9c11f9f82ae#uiux-design)
5. [Implementation Plan](https://claude.ai/chat/82878c2d-a9e5-47f3-b351-e9c11f9f82ae#implementation-plan)
6. [Testing Strategy](https://claude.ai/chat/82878c2d-a9e5-47f3-b351-e9c11f9f82ae#testing-strategy)
7. [Deployment](https://claude.ai/chat/82878c2d-a9e5-47f3-b351-e9c11f9f82ae#deployment)
8. [Maintenance & Future Enhancements](https://claude.ai/chat/82878c2d-a9e5-47f3-b351-e9c11f9f82ae#maintenance--future-enhancements)

**Project Overview**

The Forex & Crypto Trading Journal is a comprehensive platform designed to help traders track, analyze, and improve their trading performance. The application allows users to record detailed information about each trade, set goals, track progress, and receive AI-powered insights based on their trading patterns.

**Core Features**

* **Trade Entry & Management**: Complete trade logging with extensive metadata
* **Portfolio Management**: Track invested capital and account balance over time
* **Goal Setting & Tracking**: Set and monitor trading goals across different timeframes
* **Performance Analytics**: Comprehensive statistics and performance metrics
* **AI-Powered Analysis**: Identify patterns in trading behavior and provide actionable insights
* **Visual Documentation**: Upload and store screenshots of trades at different stages

**System Architecture**

The system will follow a client-server architecture with a clear separation between the frontend and backend components.

**High-Level Architecture Diagram**

┌─────────────────┐ ┌───────────────────┐ ┌─────────────────┐

│ │ │ │ │ │

│ React Frontend │◄────►│ Python Backend │◄────►│ Database │

│ │ │ │ │ │

└─────────────────┘ └───────────────────┘ └─────────────────┘

▲ ▲ ▲

│ │ │

▼ ▼ ▼

┌─────────────────┐ ┌───────────────────┐ ┌─────────────────┐

│ │ │ │ │ │

│ User's │ │ Media Storage │ │ AI Analysis │

│ Browser │ │ (Image Files) │ │ Engine │

│ │ │ │ │ │

└─────────────────┘ └───────────────────┘ └─────────────────┘

**Communication Flow**

1. User interacts with the React frontend through their browser
2. Frontend makes API calls to the Python backend
3. Backend processes requests, interacts with the database, and returns responses
4. For AI analysis, the backend communicates with the AI engine
5. Images (screenshots) are stored in a dedicated media storage system

**Backend Implementation (Python)**

**Backend Technology Stack**

* **Framework**: FastAPI
* **Database**: PostgreSQL
* **ORM**: SQLAlchemy
* **Authentication**: JWT (JSON Web Tokens)
* **AI/ML**: scikit-learn, TensorFlow, or PyTorch (for trade analysis)
* **Image Processing**: Pillow/OpenCV
* **Testing**: Pytest

**Database Schema**

sql

*-- Users Table*

CREATE TABLE users (

id SERIAL PRIMARY KEY,

username VARCHAR(50) UNIQUE NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

password\_hash VARCHAR(255) NOT NULL,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,

last\_login TIMESTAMP WITH TIME ZONE

);

*-- Accounts Table*

CREATE TABLE accounts (

id SERIAL PRIMARY KEY,

user\_id INTEGER REFERENCES users(id),

name VARCHAR(100) NOT NULL,

initial\_balance DECIMAL(18,8) NOT NULL,

current\_balance DECIMAL(18,8) NOT NULL,

currency VARCHAR(10) NOT NULL,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP

);

*-- Deposits Table*

CREATE TABLE deposits (

id SERIAL PRIMARY KEY,

account\_id INTEGER REFERENCES accounts(id),

amount DECIMAL(18,8) NOT NULL,

date TIMESTAMP WITH TIME ZONE NOT NULL,

notes TEXT

);

*-- Trades Table*

CREATE TABLE trades (

id SERIAL PRIMARY KEY,

account\_id INTEGER REFERENCES accounts(id),

date\_open TIMESTAMP WITH TIME ZONE NOT NULL,

date\_closed TIMESTAMP WITH TIME ZONE,

currency\_pair VARCHAR(20) NOT NULL,

position\_size DECIMAL(18,8) NOT NULL,

direction VARCHAR(10) NOT NULL CHECK (direction IN ('LONG', 'SHORT')),

entry\_price DECIMAL(18,8) NOT NULL,

stop\_loss DECIMAL(18,8),

take\_profit DECIMAL(18,8),

exit\_price DECIMAL(18,8),

risk\_reward DECIMAL(10,2),

win\_loss VARCHAR(10) CHECK (win\_loss IN ('WIN', 'LOSS', 'OPEN')),

profit\_amount DECIMAL(18,8),

loss\_amount DECIMAL(18,8),

profit\_percentage DECIMAL(10,2),

loss\_percentage DECIMAL(10,2),

balance\_after DECIMAL(18,8),

created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP

);

*-- Trade Details Table*

CREATE TABLE trade\_details (

id SERIAL PRIMARY KEY,

trade\_id INTEGER REFERENCES trades(id),

step\_1\_conditions TEXT,

step\_2\_bias TEXT,

step\_3\_narrative TEXT,

step\_4\_execution TEXT,

comments TEXT

);

*-- Trade Screenshots Table*

CREATE TABLE trade\_screenshots (

id SERIAL PRIMARY KEY,

trade\_id INTEGER REFERENCES trades(id),

screenshot\_type VARCHAR(20) NOT NULL CHECK (screenshot\_type IN ('HTF', 'BEFORE', 'AFTER', 'OTHER')),

file\_path VARCHAR(255) NOT NULL,

uploaded\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP

);

*-- Goals Table*

CREATE TABLE goals (

id SERIAL PRIMARY KEY,

user\_id INTEGER REFERENCES users(id),

period\_type VARCHAR(20) NOT NULL CHECK (period\_type IN ('WEEKLY', 'MONTHLY', 'YEARLY')),

start\_date DATE NOT NULL,

end\_date DATE NOT NULL,

profit\_target DECIMAL(10,2),

trades\_target INTEGER,

win\_rate\_target DECIMAL(5,2),

other\_targets TEXT,

notes TEXT,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP

);

*-- Analysis Results Table*

CREATE TABLE analysis\_results (

id SERIAL PRIMARY KEY,

user\_id INTEGER REFERENCES users(id),

analysis\_type VARCHAR(50) NOT NULL,

result\_data JSONB NOT NULL,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP

);

**API Endpoints**

**Authentication Endpoints**

* POST /api/auth/register - Register a new user
* POST /api/auth/login - Login and get JWT token
* POST /api/auth/refresh - Refresh JWT token
* POST /api/auth/logout - Logout and invalidate token

**User Endpoints**

* GET /api/users/me - Get current user info
* PUT /api/users/me - Update user info
* PATCH /api/users/password - Change password

**Account Endpoints**

* GET /api/accounts - List all user accounts
* POST /api/accounts - Create a new account
* GET /api/accounts/{id} - Get account details
* PUT /api/accounts/{id} - Update account details
* DELETE /api/accounts/{id} - Delete an account

**Deposit Endpoints**

* GET /api/accounts/{account\_id}/deposits - List all deposits for an account
* POST /api/accounts/{account\_id}/deposits - Add a new deposit
* PUT /api/deposits/{id} - Update a deposit
* DELETE /api/deposits/{id} - Delete a deposit

**Trade Endpoints**

* GET /api/accounts/{account\_id}/trades - List all trades for an account
* POST /api/accounts/{account\_id}/trades - Create a new trade
* GET /api/trades/{id} - Get trade details
* PUT /api/trades/{id} - Update trade details
* DELETE /api/trades/{id} - Delete a trade
* PATCH /api/trades/{id}/close - Close an open trade

**Trade Details Endpoints**

* GET /api/trades/{trade\_id}/details - Get trade details
* POST /api/trades/{trade\_id}/details - Create trade details
* PUT /api/trades/{trade\_id}/details - Update trade details

**Screenshot Endpoints**

* GET /api/trades/{trade\_id}/screenshots - List all screenshots for a trade
* POST /api/trades/{trade\_id}/screenshots - Upload a new screenshot
* DELETE /api/screenshots/{id} - Delete a screenshot

**Goals Endpoints**

* GET /api/goals - List all goals
* POST /api/goals - Create a new goal
* GET /api/goals/{id} - Get goal details
* PUT /api/goals/{id} - Update goal
* DELETE /api/goals/{id} - Delete a goal

**Analysis Endpoints**

* GET /api/analysis/overview - Get trading performance overview
* GET /api/analysis/patterns - Get pattern analysis
* GET /api/analysis/recommendations - Get AI recommendations
* POST /api/analysis/screenshot - Analyze a trading screenshot

**Authentication & Authorization**

The system will use JWT (JSON Web Tokens) for authentication:

1. **Registration Process**:
   * User submits username, email, and password
   * Backend validates data and creates a new user with a hashed password
   * Returns success confirmation
2. **Login Process**:
   * User submits email/username and password
   * Backend validates credentials
   * If valid, generates JWT token and returns it to frontend
   * Frontend stores token in localStorage/cookies
3. **Authentication Middleware**:
   * All protected API endpoints check for valid JWT token in Authorization header
   * Middleware validates token and attaches user info to request context
   * Invalid/expired tokens return 401 Unauthorized
4. **Token Refresh**:
   * Short-lived access tokens (24h)
   * Refresh token endpoint to get new access token

**Data Processing**

The backend will handle various data processing tasks:

1. **Trade Calculations**:
   * Automatic calculation of risk/reward ratios
   * Profit/loss calculations in both currency and percentage
   * Running balance updates
   * Win/loss determination
2. **Performance Metrics Aggregation**:
   * Win/loss ratios by currency pair, direction, etc.
   * Average profit/loss metrics
   * Trade frequency analysis
   * Drawdown calculations
3. **Image Handling**:
   * Screenshot resizing and compression
   * Metadata extraction
   * Secure storage with reference links in database

**AI Analysis Module**

The AI analysis module will provide insights based on the trader's historical data:

1. **Pattern Recognition**:
   * Identify successful and unsuccessful trading patterns
   * Analyze correlation between trade parameters and outcomes
   * Time-based analysis (time of day, day of week, etc.)
2. **Screenshot Analysis**:
   * Technical indicator identification in charts
   * Entry/exit point evaluation
   * Pattern recognition in chart formations
3. **Personalized Recommendations**:
   * Suggest improvements based on historical performance
   * Identify optimal trading conditions
   * Alert on deviations from successful patterns

**Implementation Approach**

python

*# Example of AI Analysis Service*

class TradeAnalysisService:

def \_\_init\_\_(self, user\_id):

self.user\_id = user\_id

def get\_performance\_overview(self):

"""Generate overall performance metrics"""

*# Query trades from database*

*# Calculate metrics*

*# Return formatted results*

def analyze\_patterns(self):

"""Identify patterns in trading behavior"""

*# Fetch historical trades*

*# Apply pattern recognition algorithms*

*# Generate insights*

def analyze\_screenshot(self, image\_path, trade\_context):

"""Analyze trading chart screenshots"""

*# Process image with computer vision*

*# Identify chart patterns, indicators*

*# Compare with trade details*

*# Generate feedback*

def generate\_recommendations(self):

"""Create personalized recommendations"""

*# Combine pattern analysis, performance metrics*

*# Generate actionable recommendations*

*# Format response for frontend*

**Frontend Implementation (ReactJS)**

**Frontend Technology Stack**

* **Core Framework**: React 18+
* **State Management**: Redux Toolkit or React Query
* **UI Framework**: Material-UI or Tailwind CSS
* **Form Handling**: React Hook Form
* **Data Visualization**: Recharts, D3.js, or Chart.js
* **API Communication**: Axios
* **Testing**: Jest, React Testing Library

**Component Structure**

src/

├── assets/ # Static assets, images, etc.

├── components/ # Reusable components

│ ├── auth/ # Authentication related components

│ ├── common/ # Common UI elements (buttons, inputs, etc.)

│ ├── dashboard/ # Dashboard specific components

│ ├── trades/ # Trade related components

│ ├── goals/ # Goal related components

│ ├── analysis/ # Analysis related components

│ └── layout/ # Layout components (header, sidebar, etc.)

├── context/ # React context providers

├── hooks/ # Custom React hooks

├── pages/ # Page components

│ ├── Auth/ # Login, Register pages

│ ├── Dashboard/ # Main dashboard page

│ ├── Trades/ # Trade listing and detail pages

│ ├── Goals/ # Goal setting and tracking pages

│ └── Analysis/ # Performance analysis pages

├── services/ # API services

├── store/ # Redux store setup and slices

├── utils/ # Utility functions

├── App.js # Main App component

└── index.js # Application entry point

**Key Pages and Components**

1. **Authentication Pages**
   * Login
   * Registration
   * Password Recovery
2. **Dashboard**
   * Performance Overview
   * Recent Trades
   * Goal Progress
   * Quick Actions
3. **Trade Management**
   * Trade List
   * Trade Entry Form
   * Trade Detail View
   * Trade Editing
4. **Account Management**
   * Account Overview
   * Deposit Management
   * Balance History
5. **Goals**
   * Goal Setting
   * Progress Tracking
   * Period Review
6. **Analysis**
   * Performance Metrics
   * Pattern Analysis
   * AI Recommendations
   * Screenshot Analysis

**State Management**

The application will use Redux Toolkit for global state management:

javascript

*// Example Redux slice for trades management*

import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';

import tradeService from '../services/tradeService';

export const fetchTrades = createAsyncThunk(

'trades/fetchAll',

async (accountId, { rejectWithValue }) => {

try {

const response = await tradeService.getTradesByAccount(accountId);

return response.data;

} catch (error) {

return rejectWithValue(error.response.data);

}

}

);

const tradesSlice = createSlice({

name: 'trades',

initialState: {

trades: [],

activeTrade: null,

loading: false,

error: null,

},

reducers: {

setActiveTrade: (state, action) => {

state.activeTrade = action.payload;

},

clearActiveTrade: (state) => {

state.activeTrade = null;

},

},

extraReducers: (builder) => {

builder

.addCase(fetchTrades.pending, (state) => {

state.loading = true;

})

.addCase(fetchTrades.fulfilled, (state, action) => {

state.trades = action.payload;

state.loading = false;

})

.addCase(fetchTrades.rejected, (state, action) => {

state.loading = false;

state.error = action.payload;

});

},

});

export const { setActiveTrade, clearActiveTrade } = tradesSlice.actions;

export default tradesSlice.reducer;

**UI/UX Design**

The frontend will implement a clean, modern UI with:

1. **Responsive Design**
   * Mobile-first approach
   * Adaptive layouts for all screen sizes
2. **Dashboard Widgets**
   * Performance charts
   * Trade statistics
   * Goal progress indicators
3. **Trade Entry Interface**
   * Step-by-step form with validation
   * Screenshot upload with previews
   * Calculation assistance
4. **Data Visualization**
   * Performance charts
   * Win/loss ratio visualizations
   * Balance history graphs
   * Trade distribution charts

**Example Trade Entry Component**

jsx

import React, { useState } from 'react';

import { useDispatch } from 'react-redux';

import { useForm } from 'react-hook-form';

import { createTrade } from '../store/tradesSlice';

import { TextField, Select, Button, DateTimePicker } from '../components/common';

import { ScreenshotUpload } from '../components/trades';

const TradeEntryForm = ({ accountId }) => {

const dispatch = useDispatch();

const { register, handleSubmit, watch, formState: { errors } } = useForm();

const [screenshots, setScreenshots] = useState({

htf: null,

before: null,

after: null

});

const watchDirection = watch('direction');

const watchEntryPrice = watch('entryPrice');

const watchStopLoss = watch('stopLoss');

const watchTakeProfit = watch('takeProfit');

*// Calculate risk/reward ratio based on form values*

const calculateRiskReward = () => {

if (!watchEntryPrice || !watchStopLoss || !watchTakeProfit) return null;

const entry = parseFloat(watchEntryPrice);

const sl = parseFloat(watchStopLoss);

const tp = parseFloat(watchTakeProfit);

if (watchDirection === 'LONG') {

const risk = entry - sl;

const reward = tp - entry;

return risk > 0 ? (reward / risk).toFixed(2) : null;

} else {

const risk = sl - entry;

const reward = entry - tp;

return risk > 0 ? (reward / risk).toFixed(2) : null;

}

};

const riskRewardRatio = calculateRiskReward();

const onSubmit = (data) => {

const tradeData = {

...data,

accountId,

riskReward: riskRewardRatio,

screenshots

};

dispatch(createTrade(tradeData));

};

const handleScreenshotUpload = (type, file) => {

setScreenshots(prev => ({

...prev,

[type]: file

}));

};

return (

<form onSubmit={handleSubmit(onSubmit)}>

<h2>New Trade Entry</h2>

{*/\* Basic Trade Information \*/*}

<div className="form-section">

<h3>Trade Details</h3>

<DateTimePicker

label="Date Opened"

name="dateOpen"

register={register}

required

error={errors.dateOpen}

/>

<TextField

label="Currency Pair"

name="currencyPair"

register={register}

required

error={errors.currencyPair}

/>

<TextField

label="Position Size"

name="positionSize"

type="number"

step="0.00000001"

register={register}

required

error={errors.positionSize}

/>

<Select

label="Direction"

name="direction"

options={[

{ value: 'LONG', label: 'Long' },

{ value: 'SHORT', label: 'Short' }

]}

register={register}

required

error={errors.direction}

/>

<TextField

label="Entry Price"

name="entryPrice"

type="number"

step="0.00000001"

register={register}

required

error={errors.entryPrice}

/>

<TextField

label="Stop Loss"

name="stopLoss"

type="number"

step="0.00000001"

register={register}

error={errors.stopLoss}

/>

<TextField

label="Take Profit"

name="takeProfit"

type="number"

step="0.00000001"

register={register}

error={errors.takeProfit}

/>

{riskRewardRatio && (

<div className="calculated-field">

<label>Risk/Reward Ratio</label>

<div className="value">1:{riskRewardRatio}</div>

</div>

)}

</div>

{*/\* Trade Strategy \*/*}

<div className="form-section">

<h3>Strategy</h3>

<TextField

label="Step 1 - Conditions"

name="step1Conditions"

multiline

rows={3}

register={register}

error={errors.step1Conditions}

/>

<TextField

label="Step 2 - Bias"

name="step2Bias"

multiline

rows={3}

register={register}

error={errors.step2Bias}

/>

<TextField

label="Step 3 - Narrative"

name="step3Narrative"

multiline

rows={3}

register={register}

error={errors.step3Narrative}

/>

<TextField

label="Step 4 - Execution"

name="step4Execution"

multiline

rows={3}

register={register}

error={errors.step4Execution}

/>

</div>

{*/\* Screenshots \*/*}

<div className="form-section">

<h3>Screenshots</h3>

<ScreenshotUpload

label="Higher Timeframe Chart"

type="htf"

onUpload={(file) => handleScreenshotUpload('htf', file)}

preview={screenshots.htf}

/>

<ScreenshotUpload

label="Before Entry"

type="before"

onUpload={(file) => handleScreenshotUpload('before', file)}

preview={screenshots.before}

/>

<ScreenshotUpload

label="After Exit"

type="after"

onUpload={(file) => handleScreenshotUpload('after', file)}

preview={screenshots.after}

/>

</div>

{*/\* Comments \*/*}

<div className="form-section">

<TextField

label="Comments"

name="comments"

multiline

rows={4}

register={register}

error={errors.comments}

/>

</div>

<Button type="submit" variant="primary">

Save Trade

</Button>

</form>

);

};

export default TradeEntryForm;

**Implementation Plan**

**Phase 1: Core Infrastructure (2-3 weeks)**

* Set up development environment
* Implement database schema
* Create user authentication system
* Develop basic API endpoints
* Set up frontend project structure

**Phase 2: Trade Management Features (3-4 weeks)**

* Implement trade CRUD operations
* Develop trade form with calculations
* Create trade listing and detail views
* Implement screenshot uploads
* Build account balance tracking

**Phase 3: Analytics & Dashboard (2-3 weeks)**

* Develop performance metrics calculations
* Create data visualization components
* Implement dashboard with key metrics
* Build goal tracking system

**Phase 4: AI Analysis Module (3-4 weeks)**

* Develop pattern recognition algorithms
* Implement screenshot analysis
* Create recommendation engine
* Integrate AI analysis with frontend

**Phase 5: Testing & Refinement (2-3 weeks)**

* Conduct comprehensive testing
* Fix bugs and issues
* Optimize performance
* Refine UI/UX

**Phase 6: Deployment & Launch (1-2 weeks)**

* Set up production environment
* Deploy application
* Monitor performance
* Gather initial user feedback

**Testing Strategy**

**Backend Testing**

* **Unit Tests**: Test individual functions and methods
* **Integration Tests**: Test API endpoints and database interactions
* **Load Tests**: Verify system performance under load

**Frontend Testing**

* **Component Tests**: Test individual React components
* **Integration Tests**: Test component interactions
* **E2E Tests**: Test complete user flows

**AI Module Testing**

* **Accuracy Tests**: Verify analysis accuracy
* **Performance Tests**: Test processing speed
* **Edge Case Testing**: Test with unusual data patterns

**Deployment**

**Backend Deployment**

* Set up PostgreSQL database server
* Deploy FastAPI application to a cloud provider (AWS, GCP, or Azure)
* Configure NGINX as a reverse proxy
* Set up SSL certificates

**Frontend Deployment**

* Build optimized React production bundle
* Deploy to a static hosting service (Netlify, Vercel, or AWS S3)
* Configure CDN for improved performance
* Set up proper CORS configurations

**CI/CD Pipeline**

* Implement automated testing
* Set up continuous integration with GitHub Actions or GitLab CI
* Configure automated deployments

**Maintenance & Future Enhancements**

**Regular Maintenance**

* Security updates
* Performance optimization
* Bug fixes
* Database backups

**Potential Future Enhancements**

* Mobile application
* Advanced AI features
* Social features (sharing insights, leaderboards)
* Integration with trading platforms
* Real-time market data integration
* Paper trading simulator
* Strategy backtesting
* Custom indicator creation
* Notifications system
* Export/import functionality