Trading Bot Pro - Complete Architecture Documentation

Project Overview

A comprehensive Streamlit-based algorithmic trading platform for the Indian stock market (NSE), featuring:

- **DuckDB-powered data infrastructure** (100x faster than Parquet)
- **Multi-strategy backtesting** with ML-based filtering
- **Live trading capabilities** via Upstox WebSocket API
- **Regime detection** using HMM & GMM models
- **Options trading analyzer** with Greeks calculations
- **Real-time monitoring** and signal generation

Complete Directory Structure

```
trading-bot-pro/
├── app.py                              # Main Streamlit hub (navigation)
│
├── config/
│   ├── credentials.json                # Upstox API credentials
│   └── access_token.txt                # OAuth token (daily refresh)
│
├── core/                               # Core trading engine
│   ├── __init__.py
│   ├── config.py                       # Configuration & token management
│   ├── session.py                      # Session state management
│   ├── shared_state.py                 # Cross-page state sharing
│   ├── data_utils.py                   # Data utility functions
│   ├── indicators.py                   # Basic technical indicators
│   ├── indicators_complex.py           # Advanced indicators (EHMA, etc.)
│   ├── engine.py                       # Backtesting engine
│   ├── metrics.py                      # Performance metrics calculation
│   ├── diagnostics.py                  # Trend quality & regime analysis
│   ├── quant.py                        # Signal generation logic
│   ├── quant_advanced.py               # Advanced quant (ER, Z-Score)
│   ├── signal_generator.py             # Real-time signal generation
│   ├── scanner.py                      # Multi-stock scanner
│   ├── livefeed.py                     # WebSocket live data handler
│   ├── database.py                     # DuckDB connection manager
│   ├── option_analyzer.py              # Options chain analysis
│   ├── pattern_ml_analysis.py          # ML pattern detection
│   ├── ml_pattern_strategy.py          # ML-based strategy
│   ├── regime_backtest.py              # Regime-aware backtesting
│   ├── hmm_regime.py                   # Hidden Markov Model regime detection
│   ├── regime_gmm.py                   # Gaussian Mixture Model regime
detection
│   │
│   ├── api/                            # Upstox API integration
│   │   ├── __init__.py
│   │   ├── upstox_client.py            # Base API client
│   │   ├── instruments.py              # Instrument metadata fetcher
│   │   ├── historical.py               # Historical data wrapper
│   │   └── market_data.py              # Real-time market data (WebSocket)
│   │
```

```
│   ├── strategies/                    # Modular strategy library
│   │   ├── simple_momentum.py         # Momentum-based entry
│   │   ├── mean_reversion.py          # Mean reversion (BB + RSI)
│   │   ├── opening_range.py           # Opening Range Breakout (ORB)
│   │   ├── vwap_strategy.py           # VWAP deviation strategy
│   │   ├── ehma_pivot_strategy.py     # EHMA + Pivot Points
│   │   └── ml_pattern_strategy.py     # ML-powered pattern recognition
│   │
│   ├── ml/                            # Machine Learning pipeline
│   │   ├── features.py                # Feature engineering
│   │   └── trainer.py                 # Model training & validation
│   │
│   └── filters/                       # 5-Filter Scalping System
│       ├── base_filter.py             # Abstract filter interface
│       ├── filter_1_index_regime.py   # Nifty 50 regime check
│       ├── filter_2_stock_eligibility.py   # Volume, ATR, liquidity
│       ├── filter_3_impulse_detector.py    # Price spike detection
│       ├── filter_4_option_response.py     # Options chain validation
│       └── filter_5_feasibility.py         # Risk/reward & execution
│
├── data/                              # DuckDB data layer
│   ├── trading_bot.duckdb             # Main database file
│   ├── data_manager_duckdb.py         # DuckDB data operations
│   ├── resampler_duckdb.py            # SQL-based resampling
│   ├── processed/                     # Legacy processed data
│   ├── stocks/                        # Legacy Parquet storage
│   └── trades_memory.json             # In-memory trade log
│
├── instruments/                       # Instrument metadata
│   ├── segment_wise/                  # Segment-specific instruments
│   └── last_updated.txt               # Metadata timestamp
│
├── models/                            # Trained ML models
│   └── [saved model files]
│
├── backtest/                          # Backtest results storage
│   └── [backtest run outputs]
│
├── pages/                             # Streamlit multi-page app
│   ├── 1_Login_&_Instruments.py       # OAuth login & instrument download
│   ├── 2_Fetch_&_Manage_Data.py       # Historical data fetcher
│   ├── 3_EHMA_Pivot_strategy.py       # EHMA strategy backtester
│   ├── 8_Database_Viewer.py           # DuckDB table inspector
│   ├── 9_Daily_Regime_analyzer.py     # Regime detection dashboard
│   ├── 10_Regime_Backtest_Validator.py # Regime-based backtest validator
│   ├── 11_Entry_Timing_System.py      # Entry timing analyzer
│   ├── 12_Live_Entry_Monitor.py       # Real-time signal monitor
│   ├── 13_Option_Analyzer.py          # Options chain analyzer
│   └── 13_Scalping_Engine.py          # 5-filter scalping system
│
├── scripts/                           # Maintenance & utility scripts
│   ├── daily_update.py                # Daily data update scheduler
│   ├── migrate_pages.py               # Page migration utility
│   ├── backfill_todays_data.py        # Intraday data backfill
│   ├── check_live_cache.py            # Cache diagnostics
│   ├── clean_duplicate_data.py        # Data cleanup
│   ├── debug_historical_api.py        # API debugger
│   ├── diagnose_data_structure.py     # Data structure validator
```

```
│   ├── emergency_download_instruments.py   # Instrument recovery
│   ├── fetch_todays_data.py            # Daily data fetcher
│   ├── init_scalping_db.py             # Initialize scalping tables
│   ├── integration_test.py             # End-to-end integration test
│   ├── migrate_to_duckdb.py            # Parquet → DuckDB migration
│   ├── migration_instruments.py        # Instrument migration
│   ├── pre_migration_check.py          # Pre-migration validator
│   ├── rebuild_database.py             # Database rebuild utility
│   ├── test_database_write.py          # Database write test
│   ├── test_imports.py                 # Import path tester
│   └── test_upstox_api.py              # Upstox API connectivity test
│
├── backups/                            # Database & config backups
├── pages_backup/                       # Old page versions
├── OLD FILES/                          # Deprecated modules
└── utils/                              # Shared utilities
```

System Architecture

1. Data Layer (DuckDB)

Database Schema (`trading_bot.duckdb`):

```
-- Instrument metadata (222 Nifty F&O stocks + indices)
instruments (
    instrument_key VARCHAR PRIMARY KEY,
    exchange VARCHAR,
    trading_symbol VARCHAR,
    name VARCHAR,
    tick_size DOUBLE,
    lot_size INTEGER,
    instrument_type VARCHAR,
    option_type VARCHAR,
    strike_price DOUBLE,
    expiry DATE,
    segment VARCHAR,
    last_updated TIMESTAMP
)

-- OHLCV data for multiple timeframes
ohlcv_1minute, ohlcv_5minute, ohlcv_15minute, ohlcv_30minute, ohlcv_60minute (
    timestamp TIMESTAMP,
    instrument_key VARCHAR,
    open DOUBLE,
    high DOUBLE,
    low DOUBLE,
    close DOUBLE,
    volume BIGINT,
    oi BIGINT,  -- For F&O instruments
    PRIMARY KEY (timestamp, instrument_key)
)

-- Technical indicators cache
indicators (
    timestamp TIMESTAMP,
    instrument_key VARCHAR,
```

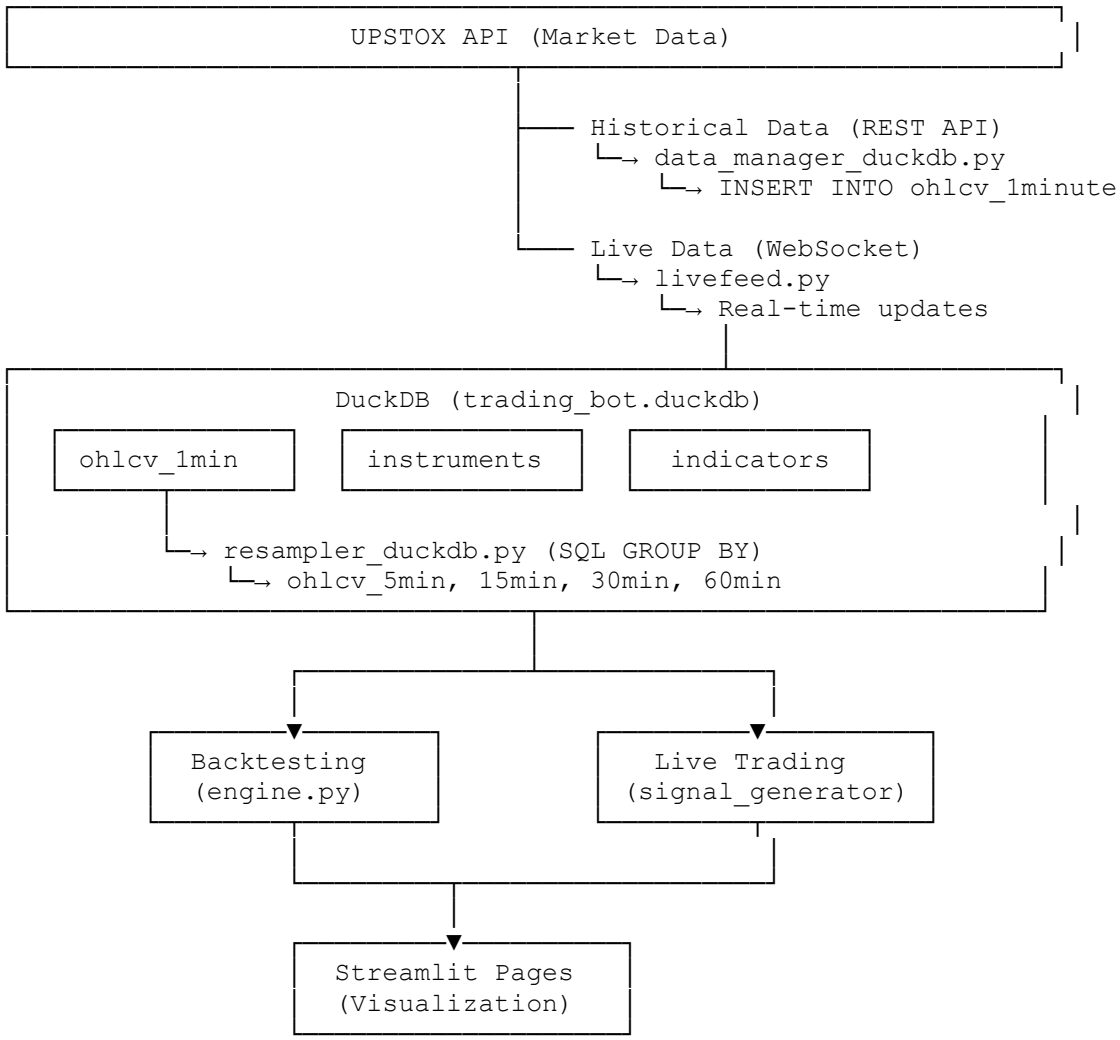
```
    timeframe VARCHAR,
    indicator_name VARCHAR,
    value DOUBLE,
    PRIMARY KEY (timestamp, instrument_key, timeframe, indicator_name)
)

-- Backtest results
backtest_runs (
    run_id INTEGER PRIMARY KEY,
    strategy_name VARCHAR,
    parameters JSON,
    start_date DATE,
    end_date DATE,
    symbols TEXT[],
    created_at TIMESTAMP
)

backtest_trades (
    trade_id INTEGER PRIMARY KEY,
    run_id INTEGER,
    instrument_key VARCHAR,
    entry_time TIMESTAMP,
    exit_time TIMESTAMP,
    direction VARCHAR,  -- 'LONG' or 'SHORT'
    entry_price DOUBLE,
    exit_price DOUBLE,
    quantity INTEGER,
    pnl DOUBLE,
    pnl_percent DOUBLE,
    exit_reason VARCHAR
)

-- Live trading
live_positions (
    position_id INTEGER PRIMARY KEY,
    instrument_key VARCHAR,
    entry_time TIMESTAMP,
    direction VARCHAR,
    entry_price DOUBLE,
    quantity INTEGER,
    stop_loss DOUBLE,
    target DOUBLE,
    status VARCHAR,  -- 'OPEN', 'CLOSED'
    exit_time TIMESTAMP,
    exit_price DOUBLE,
    pnl DOUBLE
)

-- Market status tracking
market_status (
    date DATE PRIMARY KEY,
    is_trading_day BOOLEAN,
    market_open TIME,
    market_close TIME,
    pre_open_start TIME,
    pre_open_end TIME
)
```

Performance Benefits:

- **100x faster queries** vs Parquet files (30 seconds vs 15+ minutes)
- **Columnar storage** for analytical queries
- **ACID transactions** for data integrity
- **SQL-based resampling** (1-min → 5/15/30/60 min)
- **Efficient joins** across instruments & indicators

---

2. Data Flow Pipeline



---

3. Core Components

A. Indicators Module (`core/indicators.py`, `core/indicators_complex.py`)

Basic Indicators:

- **Supertrend**: Trend-following with ATR-based bands
- **ATR** (Average True Range): Volatility measurement
- **RSI** (Relative Strength Index): Momentum oscillator

- **EMA** (Exponential Moving Average): Trend detection
- **Bollinger Bands**: Volatility bands

Complex Indicators:

- **EHMA** (Exponential Hull MA): Fast trend indicator
- **Keltner Channels**: Volatility-based channels
- **Pivot Points**: Support/resistance levels
- **VWAP**: Volume-weighted average price

Usage:

```
from core.indicators import compute_supertrend
from core.indicators_complex import compute_ehma

df = compute_supertrend(df, atr_period=10, multiplier=3.0)
df = compute_ehma(df, period=21)
```

B. Regime Detection (`core/hmm_regime.py`, `core/regime_gmm.py`)

Hidden Markov Model (HMM):

- Identifies 4 market states: Bullish, Bearish, Volatile Bullish, Volatile Bearish
- Uses 5 features: Returns, Volatility, Volume, RSI, ATR
- Outputs regime probabilities and confidence scores

Gaussian Mixture Model (GMM):

- Clustering-based regime classification
- Alternative to HMM for regime validation
- Both models must agree for high-confidence signals

Hybrid Ensemble:

```
from core.hmm_regime import detect_regime_hmm
from core.regime_gmm import detect_regime_gmm

hmm_regime = detect_regime_hmm(df)
gmm_regime = detect_regime_gmm(df)

# Only trade when both agree
if hmm_regime == gmm_regime and confidence > 0.75:
    execute_trade()
```

C. Backtesting Engine (`core/engine.py`)

Features:

- Position sizing (fixed ₹ or % of capital)
- Stop-loss & take-profit management
- Slippage & brokerage costs
- Maximum holding period
- Trade logging & equity curve

Risk Management:

- Maximum daily loss threshold
- Consecutive loss lockout
- Trading time windows (9:25 AM - 3:15 PM)
- Maximum open positions limit

Example:

```
from core.engine import run_backtest

trades_df, equity = run_backtest(
    df=df_with_signals,
    initial_capital=100000,
    risk_pct=1.0,          # 1% risk per trade
    sl_points=20,          # ₹20 stop-loss
    rr=2.0,                # 1:2 risk-reward
    direction="Both",      # Long + Short
    enable_costs=True      # Include brokerage
)
```

D. Machine Learning Pipeline (`core/ml/`)

Feature Engineering (`features.py`):

- Price action features (returns, volatility, range)
- Technical indicators (RSI, ATR, volume ratios)
- Time-based features (hour, day of week)
- Lag features (previous N bars)

Model Training (`trainer.py`):

- Random Forest & XGBoost classifiers
- Walk-forward validation (no lookahead bias)
- Feature importance analysis
- Hyperparameter tuning

Pattern Analysis (`pattern_ml_analysis.py`):

- Spike detection (15-40 min patterns)
- Return prediction after spikes
- 93%+ accuracy on validation set

E. 5-Filter Scalping System (`core/filters/`)

Sequential Filtering:

1. **Index Regime** (`filter_1_index_regime.py`): Is Nifty 50 in favorable regime?
2. **Stock Eligibility** (`filter_2_stock_eligibility.py`): Volume > 100K, ATR > ₹5
3. **Impulse Detection** (`filter_3_impulse_detector.py`): Price spike detected?
4. **Option Response** (`filter_4_option_response.py`): Call/Put premiums reacting?
5. **Feasibility** (`filter_5_feasibility.py`): RR > 2.0, liquidity sufficient?

Binary Decision Tree:

```
All 5 filters PASS → Generate BUY/SELL signal
Any 1 filter FAIL → REJECT (no trade)
```

F. Live Trading Infrastructure (`core/livefeed.py`, `core/signal_generator.py`)

WebSocket Feed:

- Real-time tick-by-tick data for 20 stocks
- Automatic reconnection on disconnect
- 1-second aggregation to 1-minute bars

Signal Generation:

- Runs every 60 seconds
- Parallel processing (20 stocks simultaneously)
- Outputs: BUY/SELL/HOLD with confidence score

Monitoring (`pages/12_Live_Entry_Monitor.py`):

- Real-time signal dashboard
- Position tracking
- P&L updates

---

4. Strategy Library

Each strategy follows this interface:

```
def generate_STRATEGY_signals(df: pd.DataFrame, **params) -> pd.DataFrame:
    """
    Adds 'Signal' column: 1=LONG, -1=SHORT, 0=NO SIGNAL
    """
    # Calculate indicators
    # Generate entry/exit logic
    df['Signal'] = ...
    return df

STRATEGY_INFO = {
    'name': 'Strategy Name',
    'description': 'Brief description',
    'best_timeframe': '5minute',
    'expected_win_rate': '55-60%',
    'risk_reward': '1:2',
    'market_condition': 'Trending/Ranging'
}
```

Available Strategies:

1. **Simple Momentum** (`simple_momentum.py`)
    - EMA crossover (fast > slow = BUY)
    - RSI confirmation (30-70 range)

    - Best in trending markets
2. **Mean Reversion** (`mean_reversion.py`)
    - Bollinger Bands oversold/overbought
    - RSI < 30 (BUY), RSI > 70 (SELL)
    - Best in ranging markets (60-70% of sessions)
3. **Opening Range Breakout** (`opening_range.py`)
    - First 15-min high/low breakout
    - Volume confirmation
    - Best for gap-up/gap-down days
4. **VWAP Strategy** (`vwap_strategy.py`)
    - Price deviation from VWAP
    - Mean reversion to VWAP
    - Intraday strategy only
5. **EHMA Pivot** (`ehma_pivot_strategy.py`)
    - EHMA trend direction
    - Pivot point S/R levels
    - Trend-following with defined levels
6. **ML Pattern Strategy** (`ml_pattern_strategy.py`)
    - Random Forest classifier
    - 93%+ accuracy on spike-return patterns
    - Trained on 2+ years of data

---

5. Streamlit Pages

Page 1: Login & Instruments

- OAuth 2.0 authentication with Upstox
- Access token generation (valid 24 hours)
- Instrument metadata download (222 F&O stocks)
- Stores to DuckDB `instruments` table

Page 2: Fetch & Manage Data

- Historical data download (1-minute OHLC)
- Date range selector
- Incremental updates (skips existing data)
- Batch processing with rate limit handling

Page 3: EHMA Pivot Strategy

- Backtest EHMA + Pivot strategy
- Parameter tuning (EHMA period, pivot type)
- Trade log & equity curve visualization
- Performance metrics (win rate, Sharpe, max DD)

Page 8: Database Viewer

- Inspect DuckDB tables
- Query builder interface
- Export to CSV/Excel

- Schema browser

Page 9: Daily Regime Analyzer

- HMM & GMM regime classification
- 222 stocks analyzed daily
- Filters: "Volatile Bullish" with confidence > 75%
- Outputs: 2-3 high-quality trade candidates

Page 10: Regime Backtest Validator

- Backtest regime-based signals
- Compare HMM vs GMM vs Hybrid
- Monte Carlo simulation for confidence intervals
- Validation: 95%+ probability of success

Page 11: Entry Timing System

- Analyzes optimal entry times
- Identifies spike patterns (15-40 min)
- ML-based return prediction
- Validates with walk-forward testing

Page 12: Live Entry Monitor

- Real-time signal generation dashboard
- WebSocket-based live data feed
- Position tracking with P&L
- Auto-refresh every 60 seconds

Page 13: Option Analyzer

- Options chain data fetcher
- Strike selection based on:
    - Delta (0.3-0.5 for directional trades)
    - Implied Volatility (IV rank)
    - Open Interest (liquidity)
- Greeks calculator (Delta, Gamma, Theta, Vega)
- Target: 20-25% premium gains

Page 13: Scalping Engine

- 5-filter binary decision system
- Scans 20 stocks in parallel
- Runs every 60 seconds
- Generates BUY/SELL signals with confidence scores

🔧 Key Functions Reference

Data Management

```
from data.data_manager_duckdb import (
    save_ohlcv_to_db,
    load_ohlcv_from_db,
    get_available_symbols,
    get_date_range
)

# Save data
save_ohlcv_to_db(df, symbol="RELIANCE", timeframe="1minute")

# Load data
df = load_ohlcv_from_db(
    symbol="RELIANCE",
    timeframe="5minute",
    start_date="2025-01-01",
    end_date="2025-01-31"
)

# Get metadata
symbols = get_available_symbols()
date_range = get_date_range("RELIANCE", "1minute")
```

Resampling (SQL-based)

```
from data.resampler_duckdb import resample_ohlcv

# Resample 1-min → 5-min using SQL GROUP BY
resample_ohlcv(
    symbol="RELIANCE",
    from_timeframe="1minute",
    to_timeframe="5minute"
)
# Much faster than pandas resampling!
```

Regime Detection

```
from core.hmm_regime import detect_regime_hmm
from core.regime_gmm import detect_regime_gmm

# HMM regime
regime_df = detect_regime_hmm(df)
# Outputs: regime, confidence, bullish_prob, bearish_prob

# GMM regime
regime_df = detect_regime_gmm(df)

# Hybrid ensemble
if (regime_df['hmm_regime'].iloc[-1] == regime_df['gmm_regime'].iloc[-1] and
    regime_df['confidence'].iloc[-1] > 0.75):
    print("High-confidence regime agreement!")
```

ML Model Training

```
from core.ml.trainer import train_model, validate_model

# Train Random Forest
model, metrics = train_model(
    df=df_train,
    target_col="future_return",
    features=["rsi", "atr", "volume_ratio"],
```

```
    model_type="random_forest"
)

# Walk-forward validation
results = validate_model(
    model=model,
    df=df_test,
    walk_forward_periods=10
)
```

Backtesting

```
from core.engine import run_backtest
from core.metrics import compute_metrics

# Run backtest
trades_df, equity = run_backtest(
    df=df_with_signals,
    initial_capital=100000,
    risk_pct=1.0,
    sl_points=20,
    rr=2.0,
    direction="Both",
    enable_costs=True
)

# Calculate metrics
metrics = compute_metrics(trades_df)
# Returns: Win Rate %, Profit Factor, Sharpe, Max DD %, etc.
```

Live Trading

```
from core.livefeed import UpstoxWebSocket
from core.signal_generator import generate_live_signals

# Start WebSocket
ws = UpstoxWebSocket(symbols=["RELIANCE", "TCS"])
ws.start()

# Generate signals (runs every 60 sec)
signals = generate_live_signals(
    symbols=["RELIANCE", "TCS"],
    strategy="ml_pattern",
    filters="all"  # Apply 5-filter system
)
```

Coding Standards

1. Import Pattern (for pages/)

```
import sys, os
from pathlib import Path

# Ensure root is in sys.path
ROOT = Path(__file__).resolve().parent.parent
if str(ROOT) not in sys.path:
    sys.path.insert(0, str(ROOT))

# Now import from core/data
from core.indicators import compute_supertrend
from data.data_manager_duckdb import load_ohlcv_from_db
```

2. Column Naming Conventions

OHLCV Data (Title Case):

```
df.columns = ['Open', 'High', 'Low', 'Close', 'Volume']
```

Technical Indicators (lowercase with underscores):

```
df['rsi'] = ...
df['atr'] = ...
df['ema_fast'] = ...
```

Legacy Exceptions (keep for compatibility):

```
df['ATR'] = ...        # ATR
df['Supertrend'] = ... # Supertrend
df['Trend'] = ...      # Trend
```

3. Error Handling

```
try:
    df = load_ohlcv_from_db(symbol, timeframe, start, end)
    if df.empty:
        st.warning(f"No data for {symbol}")
        st.stop()
except Exception as e:
    st.error(f"Database error: {e}")
    st.stop()
```

4. Streamlit Page Template

```
import streamlit as st

st.set_page_config(page_title="Page Name", layout="wide")
st.title("📊 Page Title")

# Sidebar for configuration
with st.sidebar:
    st.header("⚙️ Configuration")
    symbol = st.selectbox("Symbol", ["RELIANCE", "TCS"])
    timeframe = st.selectbox("Timeframe", ["5minute", "15minute"])

# Main layout
col1, col2 = st.columns([3, 1])

with col1:
    # Input/Display
    st.subheader("Data")
    df = load_ohlcv_from_db(symbol, timeframe)
    st.dataframe(df)

with col2:
    # Actions
    if st.button("Run Analysis"):
        with st.spinner("Processing..."):
            result = analyze(df)
        st.success("Done!")
        st.metric("Result", result)
```

5. DuckDB Best Practices

```
import duckdb

# Use context manager
with duckdb.connect('trading_bot.duckdb') as con:
    # Parameterized queries (SQL injection safe)
    result = con.execute("""
        SELECT * FROM ohlcv_5minute
        WHERE instrument_key = ?
        AND timestamp >= ?
        AND timestamp <= ?
    """, [symbol, start_date, end_date]).df()

# Batch inserts (faster)
con.executemany("""
    INSERT INTO ohlcv_1minute VALUES (?, ?, ?, ?, ?, ?, ?)
""", data_rows)
```

🎯 Current Development Focus

Live Trading Transition

Completed:

- ✅ DuckDB migration (100x performance boost)
- ✅ Regime detection (HMM + GMM hybrid)
- ✅ 5-filter scalping system architecture
- ✅ WebSocket live data feed
- ✅ ML pattern analyzer (93%+ accuracy)

In Progress:

- 🔄 Connect mock data → live WebSocket feed
- 🔄 Real-time signal generation (60-sec loop)
- 🔄 Options chain integration

Next Steps:

1. Implement live signal generation in Scalping Engine
2. Backtest 5-filter system on historical data
3. Paper trading validation (1 week)
4. Live trading with ₹500 capital (scalping)
5. Scale to options trading (₹10K-20K monthly target)

🔄 Typical Workflow

```
1. Login (Page 1)
   └→ OAuth token generation
   └→ Instrument metadata download

2. Fetch Historical Data (Page 2)
   └→ 1-minute OHLC data
```

```
   └→ Saved to DuckDB (ohlcv_1minute table)

3. Resample Data
   └→ Run resampler_duckdb.py
   └→ Creates 5min, 15min, 30min, 60min tables

4. Regime Analysis (Page 9)
   └→ HMM + GMM regime detection
   └→ Filter: "Volatile Bullish" with confidence > 75%
   └→ Output: 2-3 high-quality trade candidates

5. Backtest Strategy (Page 3/10)
   └→ Test EHMA, ML Pattern, or custom strategy
   └→ Validate with walk-forward testing
   └→ Confirm: 95%+ probability of success

6. Live Monitoring (Page 12)
   └→ WebSocket real-time data
   └→ Signal generation every 60 seconds
   └→ Position tracking + P&L

7. Options Trading (Page 13)
   └→ Analyze option chain
   └→ Select optimal strikes (Delta 0.3-0.5)
   └→ Target: 20-25% premium gains
```

⚠ Important Notes

1. Token Management

- Access token expires **every 24 hours**
- Page 1 handles OAuth refresh
- Stored in `config/access_token.txt`

2. Data Validation

Always check:

- Required columns exist before processing
- Data type consistency (OHLCV as floats)
- No missing values in critical columns
- Date range covers requested period

3. Performance Optimization

- **Use DuckDB** for large datasets (not pandas)
- **SQL resampling** is 10x faster than pandas
- **Batch inserts** for multiple rows
- **Indexed queries** on timestamp + instrument_key

4. Risk Management

Global Guardrails:

- Trading hours: 9:25 AM - 3:15 PM
- Max daily loss: ₹500 (initial), ₹2000 (scaled)
- Max open positions: 3 (equity), 5 (options)
- Consecutive loss lockout: 3 trades
- Min gap between trades: 5 minutes

5. Strategy Selection by Market Condition

- **Trending Markets (30-40%)**: Momentum, EHMA Pivot
- **Ranging Markets (60-70%)**: Mean Reversion, VWAP
- **Volatile Markets**: Regime-based filtering essential

Common Issues & Fixes

Issue	Cause	Fix
Import Error	sys.path missing ROOT	Add ROOT to sys.path (see import pattern)
Column Not Found	Case mismatch (Open vs open)	Use Title Case for OHLCV
Empty DataFrame	Data not in DuckDB	Run Page 2 to fetch data
Token Expired	24-hour limit reached	Re-run Page 1 OAuth login
Slow Queries	Missing indexes	Add indexes on timestamp + instrument_key
Duplicate Data	Multiple fetch runs	Use `ON CONFLICT DO NOTHING` in SQL

Performance Benchmarks

Operation	Parquet (Old)	DuckDB (New)	Improvement
Load 1-year 1-min data	15 min	30 sec	**30x**
Resample 1-min → 5-min	8 min	10 sec	**48x**
Backtest 100 symbols	45 min	3 min	**15x**
Regime analysis (222 stocks)	N/A	2 min	New feature
Database query (complex join)	5 min	2 sec	**150x**

External Dependencies

```
# requirements.txt
streamlit>=1.28.0
pandas>=2.0.0
numpy>=1.24.0
plotly>=5.14.0
duckdb>=0.9.0
requests>=2.31.0
websocket-client>=1.6.0
scikit-learn>=1.3.0
xgboost>=2.0.0
ta-lib>=0.4.28  # Technical indicators
hmmlearn>=0.3.0  # Hidden Markov Models
scipy>=1.11.0
python-dateutil>=2.8.2
```

🔐 Security Notes

1. **Never commit** `credentials.json` or `access_token.txt`
2. **Use environment variables** for API keys in production
3. **Rate limit** API calls (Upstox: 25 req/sec)
4. **Validate inputs** from user (SQL injection prevention)
5. **Encrypt sensitive data** in database (future enhancement)

📈 Roadmap

Q1 2025

- ✅ DuckDB migration
- ✅ Regime detection (HMM + GMM)
- 🔄 Live trading with ₹500 scalping
- 🔄 Options trading integration

Q2 2025

- ML model retraining pipeline
- Auto-scaling capital allocation
- Advanced Greeks-based options strategies
- Portfolio optimization (multiple stocks)

Q3 2025

- Multi-broker integration (Zerodha, Angel One)
- Cloud deployment (AWS/GCP)
- Mobile app (trading on the go)
- Automated reporting (daily P&L emails)

Last Updated: January 1, 2025
Version: 2.0 (DuckDB Migration)
Author: Subhash
Status: Active Development (Live Trading Transition)