SimuTrador - Complete Trading Simulation Platform

Overview

SimuTrador is a comprehensive trading simulation platform that combines robust historical data management with high-fidelity order execution simulation. The platform consists of two integrated systems:

1. **OHLCV Data Manager** - A sophisticated data pipeline for fetching, storing, and processing historical market data
2. **WebSocket Simulation Engine** - A real-time trading simulator that executes orders against historical data with realistic market conditions

Together, these systems provide developers and quantitative traders with a complete solution for strategy development, backtesting, and validation.

🎯 Core Philosophy

SimuTrador is built on the principle of **privacy-respecting realistic simulation**. Unlike traditional backtesting platforms that require you to upload your strategy code, SimuTrador allows you to:

- **Keep your strategy logic private** - Your proprietary algorithms never leave your environment
- **Test with realistic execution** - Includes slippage, latency, and commissions for accurate results
- **Use your own data** - Client manages market data access while server validates execution
- **Scale from research to production** - Same logic works for backtesting and live trading

🏗️ System Architecture

Data Management Layer (OHLCV Manager)

- **Multi-provider data fetching** from Polygon.io, Financial Modeling Prep, and Tiingo
- **Intelligent storage** using partitioned Parquet files for optimal performance
- **Asset-aware resampling** that matches provider aggregation patterns
- **Automated data validation** with gap detection and filling
- **Nightly update workflows** for maintaining current data

Simulation Layer (WebSocket Engine)

- **Real-time order execution** with tick-by-tick progression
- **Realistic market simulation** including slippage and commission modeling
- **Flow control mechanisms** for managing simulation pace
- **Multi-asset support** with portfolio tracking
- **Interactive controls** for pause/resume and state inspection

Data Separation Model

```
Client Side:                    Server Side:
- Market data access           - Order validation
- Strategy logic               - Execution simulation
- Order generation             - Portfolio tracking
- Simulation control           - Performance calculation
```

🔒 Key Advantages

Privacy & Security

- **No strategy exposure** - Your trading logic remains completely private
- **Data sovereignty** - You control your market data sources
- **Secure authentication** - JWT-based access with API key management
- **Isolated execution** - Each simulation runs in its own secure context

Execution Fidelity

- **Realistic fills** - Advanced slippage and latency modeling
- **Market microstructure** - Bid-ask spread simulation and partial fills
- **Commission accuracy** - Configurable fee structures matching real brokers
- **Market hours simulation** - Pre-market, regular, and after-hours sessions

Developer Experience

- **API-first design** - Integrates seamlessly into CI/CD pipelines
- **Multiple timeframes** - From 1-minute to daily data with automatic resampling
- **Real-time feedback** - Live progress tracking and interactive controls
- **Comprehensive metrics** - Sharpe ratio, drawdown, win rate, and custom analytics

Scalability & Performance

- **Concurrent simulations** - Run multiple strategies simultaneously
- **Efficient data access** - Columnar storage with intelligent caching
- **Streaming execution** - Memory-efficient processing of large datasets
- **Cloud-ready architecture** - Designed for horizontal scaling

🚀 Use Cases

Quantitative Research

- **Strategy development** - Rapid prototyping and testing of trading ideas
- **Parameter optimization** - Systematic testing of strategy parameters
- **Risk analysis** - Stress testing under different market conditions
- **Performance attribution** - Understanding sources of returns

Algorithm Validation

- **Backtesting accuracy** - Ensure historical performance matches live execution
- **Slippage analysis** - Understand real-world execution costs
- **Capacity planning** - Test strategy performance at different scales
- **Regime testing** - Validate across different market environments

Production Preparation

- **Pre-deployment testing** - Final validation before live trading
- **Risk management** - Test stop-loss and position sizing logic
- **Broker integration** - Validate order routing and execution logic
- **Compliance testing** - Ensure adherence to trading rules and limits

💡 Competitive Advantages

vs. Traditional Backtesting Platforms

- **No vendor lock-in** - Use any data source, any programming language
- **No GUI overhead** - Pure API access for programmatic control
- **No forced frameworks** - Integrate with your existing tools and workflows
- **Realistic execution** - More accurate than simple OHLC-based backtests

vs. Professional Platforms

- **Cost-effective** - Fraction of the cost of enterprise solutions
- **Developer-friendly** - Built for programmers, not point-and-click users

- **Transparent pricing** - No hidden fees or usage-based surprises
- **Open architecture** - Extensible and customizable

vs. DIY Solutions

- **Production-ready** - Battle-tested execution engine and data pipeline
- **Maintained infrastructure** - No need to build and maintain complex systems
- **Realistic modeling** - Sophisticated slippage and commission simulation
- **Scalable architecture** - Handles large-scale simulations efficiently

💸 Pricing Model

SimuTrador offers flexible pricing tiers to accommodate different user needs:

Tier	Price	Features
Starter	$29/month	10k ticks/day, 3 simulations/day, basic data access
Pro	$79/month	1M ticks/day, unlimited simulations, multi-provider data
Enterprise	Custom	Unlimited usage, priority support, custom integrations

Pay-per-use options:

- `$0.01 per 1000 ticks simulated` - Perfect for occasional testing
- `$0.05 per session` with detailed performance reports

📈 Roadmap & Future Vision

Phase 1 (Q4 2025) - December 2025

- 🧩 **SDK Development** - Python client library
- **Local server for Stock Trading** - Simulator to support personal strategies
- **Realistic market conditions Fidelity Level 1** - Support realistic market conditions with slippage and spreads, without advanced concepts such as partial fills.
- 📊 **Baktesting Reports Fidelity Level 1**- Generate useful reports with some important metrics but not advanced analytics

Phase 2 (Q1 2025) - March 2026

- 🌐 **SaaS Platform** - Hosted solution with user management
- 📱 **Dashboard Interface** - Web-based monitoring and control panel

- 🤖 **Rate Limits and Subscriptions** - Implement basic rate limits for free and paid plan, stick only to two plans for simplicity

Phase 3 (Q2 2025) - June 2026

- 📈 **Live Trading Bridge** - Seamless transition from simulation to live trading
- **Compare Real Strategy results with IBKR paper Trading** - Validate the SaaS by comparing live trading results of any given strategy with the simulation results

Phase 4 (Q4 2025) - Sep 2026

- ⚡ Go Live

Long-term (2027+)

- 🏢 **Enterprise Features** - Multi-tenant architecture and compliance tools
- 🌍 **Global Markets** - Support for international exchanges and instruments
- 🔬 **Research Platform** - Collaborative environment for strategy development
- ⚡ **Real-time Simulation** - Live market simulation with streaming data

🛠️ Technical Implementation

Data Pipeline Architecture

The OHLCV Manager implements a sophisticated data processing pipeline:

```
Data Sources → Validation → Storage → Resampling → API Access
     ↓              ↓          ↓          ↓           ↓
  Polygon.io    Market Cal.  Parquet   Asset-Aware  REST/WS
  FMP/Tiingo    Gap Detection Files    Algorithms   Endpoints
```

Simulation Engine Design

The WebSocket engine provides real-time execution simulation:

```
Client Strategy → Order Generation → Server Validation → Execution
Simulation
       ↓                ↓                 ↓                  ↓
   Private Logic    Market Orders      Order Validation    Realistic Fills
   Local Data       Limit Orders      Risk Checks         Slippage Model
   Timing Control   Stop Orders       Portfolio Limits    Commission Calc
```

🔧 Getting Started

1. Data Setup

First, configure your data sources and begin collecting historical data:

```
# Start the OHLCV Manager
curl -X POST http://localhost:8002/nightly-update/start \
  -H "Content-Type: application/json" \
  -d '{"symbols": ["AAPL", "GOOGL", "MSFT"]}'
```

2. Authentication

Obtain your API credentials and exchange for a JWT token:

```
# Get JWT token
curl -X POST https://api.simutrador.com/auth/token \
  -H "X-API-Key: sk_live_your_api_key_here"
```

3. Simulation Setup

Connect to the WebSocket API and create your first simulation:

```
const ws = new WebSocket(
  "wss://api.simutrador.com/ws/simulate?token=your_jwt_token"
);

// Create simulation session
ws.send(
  JSON.stringify({
    type: "create_session",
    data: {
      session_id: "my_strategy_test",
      symbols: ["AAPL", "GOOGL"],
      start: "2024-01-01T14:30:00Z",
      end: "2024-01-01T21:00:00Z",
      initial_cash: "100000.00",
    },
  })
);
```

4. Strategy Implementation

Implement your trading logic with full privacy:

```
# Your strategy logic (runs locally)
def my_strategy(timestamp, market_data):
    # Your proprietary algorithm here
    if should_buy(market_data):
        return create_buy_order("AAPL", 100)
    elif should_sell(market_data):
        return create_sell_order("AAPL", 50)
    return None
```

📚 Documentation Structure

This documentation is organized into two main sections:

OHLCV Data Manager

Comprehensive guide to the data management system:

- Data provider integration and configuration
- Storage architecture and optimization
- Validation and quality assurance
- Automated update workflows
- API reference for data access

WebSocket Simulation API

Complete reference for the trading simulation engine:

- Authentication and connection management
- Session lifecycle and configuration
- Order types and execution modeling
- Real-time communication protocols
- Error handling and recovery

🤝 Community & Support

Documentation & Resources

- **API Documentation** - Complete reference with examples
- **SDK Libraries** - Official clients in multiple languages
- **Example Strategies** - Open-source trading algorithm examples

- **Best Practices** - Guidelines for optimal performance

Support Channels

- **GitHub Issues** - Bug reports and feature requests
- **Discord Community** - Real-time chat with other developers
- **Email Support** - Direct access to the development team
- **Enterprise Support** - Dedicated support for business customers

Contributing

SimuTrador is built with the developer community in mind:

- **Open Source Components** - Core libraries available on GitHub
- **API Feedback** - Regular community input on API design
- **Feature Requests** - Community-driven roadmap planning
- **Beta Testing** - Early access to new features

OHLCV Data Manager

> ohlcv_manager
>
> # OHLCV Manager
>
> The OHLCV Manager is SimuTrador's comprehensive data management system that handles the complete lifecycle of trading data from fetching to storage and analysis. It provides a robust, scalable solution for managing historical price data across multiple timeframes and asset types.
>
> ## Overview
>
> The OHLCV Manager orchestrates several key components to provide reliable trading data:
>
> - **Data Fetching**: Efficient 1-minute data retrieval from multiple providers (Polygon.io, Financial Modeling Prep, Tiingo)
> - **Data Storage**: Partitioned Parquet file storage with optimized folder structure
> - **Data Resampling**: Intelligent conversion from 1-minute to various timeframes (5min, 15min, 30min, 1h, 2h, 4h, daily)
> - **Data Validation**: Comprehensive completeness analysis and gap detection

- **Data Analysis**: Quality assessment and reporting tools
- **Automated Updates**: Nightly update workflows for maintaining current data

Architecture

Core Components

1. **Data Providers** (`services/data_providers/`)
   - Vendor-agnostic interface for multiple data sources
   - Polygon.io client with intelligent batching and rate limiting
   - Support for Financial Modeling Prep and Tiingo
   - Automatic retry mechanisms and error handling
2. **Storage Service** (`services/storage/data_storage_service.py`)
   - Partitioned Parquet file storage: `storage/candles/timeframe/symbol/date.parquet`
   - Efficient pagination and data retrieval
   - Automatic deduplication and data merging
   - Optimized last update date detection
3. **Resampling Service** (`services/storage/data_resampling_service.py`)
   - Asset-type-aware resampling strategies
   - Pandas-based OHLCV aggregation
   - Market session alignment for different asset classes
   - Bulk resampling capabilities
4. **Validation Service** (`services/validation/stock_market_validation_service.py`)
   - Trading day validation using market calendars
   - Data completeness analysis
   - Gap detection and reporting
   - Market hours and holiday handling
5. **Workflow Services** (`services/workflows/`)
   - Orchestrated nightly update processes
   - Multi-symbol concurrent processing
   - Progress tracking and status reporting
   - Error handling and recovery

API Endpoints

Trading Data API (`/trading-data`)

Get Trading Data

```
GET /trading-data/data/{symbol}
```

Retrieve stored trading data for a symbol with pagination support.

Parameters:

- `symbol` (path): Trading symbol (e.g., AAPL, MSFT)
- `timeframe` (query): Data timeframe (default: "1min")
  - Supported: `1min` , `5min` , `15min` , `30min` , `1h` , `2h` , `4h` , `daily`
- `start_date` (query): Start date filter (YYYY-MM-DD format)
- `end_date` (query): End date filter (YYYY-MM-DD format)
- `order_by` (query): Sort order - `asc` or `desc` (default: desc)
- `page` (query): Page number (1-based, default: 1)
- `page_size` (query): Items per page (default: 1000, max: 10000)

Response:


```
{
  "symbol": "AAPL",
  "timeframe": "1min",
  "candles": [
    {
      "date": "2024-01-15T20:00:00Z",
      "open": "185.50",
      "high": "186.25",
      "low": "185.30",
      "close": "186.00",
      "volume": "1250000.00000000"
    }
  ],
  "start_date": "2024-01-15T13:30:00Z",
  "end_date": "2024-01-15T20:00:00Z",
  "pagination": {
    "page": 1,
    "page_size": 1000,
    "total_items": 390,
    "total_pages": 1,
    "has_next": false,
    "has_previous": false
```

```
  }
}
```

List Stored Symbols

```
GET /trading-data/symbols
```

List all symbols that have stored data.

Parameters:

- `timeframe` (query): Timeframe to check (default: "1min")

Response:

```
["AAPL", "GOOGL", "MSFT", "TSLA"]
```

Nightly Update API (`/nightly-update`)

Start Nightly Update

```
POST /nightly-update/start
```

Trigger the complete nightly update workflow for stock market data.

Request Body:

```
{
  "symbols": ["AAPL", "GOOGL", "MSFT"], // Optional, uses default if null
  "start_date": "2024-01-01", // Optional override
  "end_date": "2024-01-15", // Optional override
  "force_validation": true, // Default: true
  "enable_resampling": true, // Default: true
  "max_concurrent_symbols": 5 // Default: from settings
}
```

Response:

```
{
  "request_id": "550e8400-e29b-41d4-a716-446655440000",
```

```
  "status": "started",
  "message": "Nightly update started for 3 symbols"
}
```

Get Update Status

```
GET /nightly-update/status/{request_id}
```

Get detailed status and progress information for a nightly update request.

Response:

```
{
  "request_id": "550e8400-e29b-41d4-a716-446655440000",
  "status": "in_progress",
  "started_at": "2024-01-15T02:00:00Z",
  "symbols_count": 3,
  "is_complete": false,
  "progress": {
    "total_symbols": 3,
    "completed_symbols": 1,
    "current_symbol": "GOOGL",
    "current_step": "Downloading 1-minute data",
    "progress_percentage": 33.3,
    "estimated_time_remaining_seconds": 120,
    "symbols_in_progress": ["GOOGL"]
  }
}
```

Get Progress Details

```
GET /nightly-update/status/{request_id}/progress
```

Get detailed progress information for each symbol in the update.

Get Update Results

```
GET /nightly-update/status/{request_id}/details
```

Get complete results of a finished nightly update.

List Active Updates

```
GET /nightly-update/active
```

List all currently running nightly update requests.

Data Analysis API (`/data-analysis`)

Analyze Data Completeness

```
POST /data-analysis/completeness
```

Analyze data completeness for specified symbols and date range.

Request Body:

```
{
  "symbols": ["AAPL", "GOOGL"],
  "start_date": "2024-01-01",
  "end_date": "2024-01-15",
  "include_details": true,
  "auto_fill_gaps": false,
  "max_gap_fill_attempts": 3
}
```

Response:

```
{
  "analysis_period": {
    "start_date": "2024-01-01",
    "end_date": "2024-01-15"
  },
  "symbol_completeness": {
    "AAPL": {
      "total_trading_days": 10,
      "valid_days": 9,
      "invalid_days": 1,
      "completeness_percentage": 98.5,
      "total_expected_candles": 3900,
```

```
      "total_actual_candles": 3841,
      "missing_candles": 59,
      "validation_results": [...]
    }
  },
  "overall_statistics": {
    "total_symbols": 2,
    "total_trading_days": 20,
    "total_valid_days": 18,
    "overall_completeness_percentage": 97.8,
    "total_expected_candles": 7800,
    "total_actual_candles": 7629,
    "total_missing_candles": 171
  },
  "symbols_needing_attention": ["GOOGL"],
  "recommendations": [
    "1 symbols have less than 95% data completeness",
    "Consider running a full data update to improve completeness"
  ]
}
```

Data Models

PriceCandle

The core data model representing OHLCV (Open, High, Low, Close, Volume) data:

```
{
  "date": "2024-01-15T20:00:00Z",      # UTC timestamp
  "open": "185.50",                    # Opening price (Decimal, 2
decimal places)
  "high": "186.25",                    # Highest price (Decimal, 2
decimal places)
  "low": "185.30",                     # Lowest price (Decimal, 2 decimal
places)
  "close": "186.00",                   # Closing price (Decimal, 2
decimal places)
  "volume": "1250000.00000000"         # Trading volume (Decimal, 8
decimal places)
}
```

Supported Timeframes

- `1min` - 1-minute candles (source data)
- `5min` - 5-minute candles
- `15min` - 15-minute candles
- `30min` - 30-minute candles
- `1h` - 1-hour candles
- `2h` - 2-hour candles
- `4h` - 4-hour candles
- `daily` - Daily candles

Data Storage Structure

The system uses a partitioned Parquet file structure for optimal performance:

```
storage/
└── candles/
    ├── 1min/
    │   ├── AAPL/
    │   │   ├── 2024-01-15.parquet
    │   │   ├── 2024-01-16.parquet
    │   │   └── ...
    │   └── GOOGL/
    │       └── ...
    ├── 5min/
    │   └── ...
    ├── daily/
    │   ├── AAPL.parquet
    │   ├── GOOGL.parquet
    │   └── ...
    └── ...
```

Storage Benefits

- **Partitioned by timeframe and symbol**: Enables efficient querying
- **Daily files for intraday data**: Optimizes loading and reduces memory usage
- **Single file for daily data**: Simplifies daily candle management
- **Parquet format**: Provides compression and fast columnar access
- **Automatic deduplication**: Prevents duplicate data on updates

Resampling Strategy

The system uses asset-type-aware resampling to match external provider aggregation patterns:

US Equity (AAPL, MSFT, etc.)

- **Market Hours**: 09:30-16:00 ET (13:30-20:00 UTC)
- **Short Timeframes** (5min, 15min, 30min): `offset='13h30min'` (market session aligned)
- **Daily Boundary**: Market close (20:00 UTC / 16:00 ET)
- **Rationale**: Matches US market session boundaries

Crypto (BTC-USD, ETH-USDT, etc.)

- **Market Hours**: 24/7 continuous trading
- **All Timeframes**: Standard UTC alignment (no offset)
- **Daily Boundary**: UTC midnight (00:00 UTC)
- **Rationale**: Matches provider's UTC-based crypto aggregation

Forex (EURUSD, GBP/USD, etc.)

- **Market Hours**: 24/5 global sessions
- **Short Timeframes**: `offset='8h00min'` (London session aligned)
- **Daily Boundary**: UTC midnight (00:00 UTC)
- **Rationale**: Aligns with major forex trading session

OHLCV Aggregation Rules

- **Open**: First value in the period
- **High**: Maximum value in the period
- **Low**: Minimum value in the period
- **Close**: Last value in the period
- **Volume**: Sum of all volumes in the period

Data Validation

Trading Day Validation

- Uses `pandas_market_calendars` for official NYSE trading days
- Validates market hours: 09:30-16:00 ET (13:30-20:00 UTC)
- Handles market holidays and half-days
- Expected candles per trading day:
  - Full day: 390 candles (6.5 hours × 60 minutes)
  - Half day: 210 candles (3.5 hours × 60 minutes)

Completeness Analysis

- **Per-symbol analysis**: Individual symbol data quality metrics
- **Gap detection**: Identifies missing time periods
- **Completeness percentage**: Actual vs expected candle counts
- **Quality thresholds**: Flags symbols with <95% completeness
- **Recommendations**: Automated suggestions for data improvement

Gap Filling

- **Automatic gap detection**: Identifies missing data periods
- **Polygon URL generation**: Creates API URLs for missing data
- **Retry mechanisms**: Handles temporary API failures
- **Progress tracking**: Reports gap filling success rates

Nightly Update Workflow

The nightly update process ensures data currency through automated workflows:

Process Steps

1. **Validation Phase**
   - Check existing data completeness
   - Identify symbols needing updates
   - Determine date ranges for updates
2. **Data Fetching Phase**
   - Download missing 1-minute data from providers

- Handle rate limiting and retries
- Validate and store new data

3. **Resampling Phase**
   - Generate all target timeframes from 1-minute data
   - Apply asset-type-aware resampling strategies
   - Store resampled data in appropriate partitions
4. **Reporting Phase**
   - Generate update summaries
   - Report success/failure statistics
   - Provide recommendations for data quality

Concurrency and Performance

- **Concurrent symbol processing**: Configurable parallelism
- **Intelligent batching**: Optimizes API usage
- **Progress tracking**: Real-time status updates
- **Error isolation**: Individual symbol failures don't affect others

Configuration

Data Provider Settings

```
# Default provider priority
providers = ["polygon", "financial_modeling_prep", "tiingo"]

# Rate limiting
polygon_requests_per_minute = 5
max_concurrent_symbols = 5
```

Storage Settings

```
# Base storage path
base_path = "storage"
candles_path = "candles"

# Pagination defaults
```

```
default_page_size = 1000
max_page_size = 10000
```

Nightly Update Settings

```
# Update configuration
enable_auto_resampling = true
max_concurrent_symbols = 5
default_symbols = ["AAPL", "GOOGL", "MSFT", "TSLA", "AMZN"]

# Target timeframes for resampling
target_timeframes = ["5min", "15min", "30min", "1h", "2h", "4h", "daily"]
```

Error Handling

Provider Errors

- **Rate limiting**: Automatic backoff and retry
- **Authentication failures**: Clear error messages
- **Data unavailability**: Graceful degradation
- **Network issues**: Exponential backoff retry

Storage Errors

- **Disk space**: Monitoring and alerts
- **File corruption**: Validation and recovery
- **Permission issues**: Clear error reporting
- **Concurrent access**: File locking mechanisms

Validation Errors

- **Missing data**: Gap identification and filling
- **Data quality**: Outlier detection and reporting
- **Format issues**: Data cleaning and normalization
- **Timezone handling**: Consistent UTC conversion

Performance Optimization

Storage Optimizations

- **Columnar storage**: Parquet format for fast queries
- **Partitioning**: Efficient data organization
- **Compression**: Reduced storage footprint
- **Indexing**: Fast symbol and date lookups

Query Optimizations

- **Pagination**: Memory-efficient data retrieval
- **Date filtering**: Optimized range queries
- **Lazy loading**: Load only required data
- **Caching**: Frequently accessed data caching

Processing Optimizations

- **Vectorized operations**: Pandas-based processing
- **Batch processing**: Efficient bulk operations
- **Parallel processing**: Multi-symbol concurrency
- **Memory management**: Streaming data processing

Monitoring and Observability

Logging

- **Structured logging**: JSON format for analysis
- **Log levels**: Configurable verbosity
- **Performance metrics**: Timing and throughput data
- **Error tracking**: Detailed error context

Metrics

- **Data completeness**: Per-symbol quality metrics
- **Update performance**: Processing times and throughput

- **API usage**: Rate limiting and quota tracking
- **Storage utilization**: Disk usage and growth trends

Health Checks

- **Data freshness**: Last update timestamps
- **Service availability**: API endpoint health
- **Storage health**: File system status
- **Provider connectivity**: External API status

WebSocket Simulation API

ws_api_v2

SimuTrade WebSocket API v2.0

High-fidelity trading simulator API for developers and quantitative traders. Test your strategies against historical data with realistic execution modeling while keeping your proprietary logic completely private.

🎯 Core Concepts

Data Separation Model

- **Client responsibility**: Obtain and manage your own market data
- **Server responsibility**: Validate orders against the server's copy of the same data
- **Synchronization**: Both sides use identical data sources, different access methods
- **Privacy**: Server never sees your market data or strategy logic

Simulation Approach

- **Time progression**: Server advances simulation time tick-by-tick
- **Order execution**: Server simulates realistic fills based on historical market conditions
- **Client control**: Client controls simulation pace through tick acknowledgments
- **State management**: Server maintains portfolio state, client maintains strategy state

Client-Server Responsibilities

```
Client Side:                    Server Side:
- Market data access           - Order validation
- Strategy logic               - Execution simulation
- Order generation             - Portfolio tracking
- Simulation control           - Performance calculation
```

📋 API Overview

SimuTrade uses a **WebSocket-only architecture** for the simulation API:

- **WebSocket API**: Complete simulation lifecycle - authentication, session creation, real-time execution, and results
- **Admin Dashboard**: Separate system for account management and API key generation (not part of simulation API)

How It Works

1. **Get API key** from a separate admin dashboard/account management system
2. **Exchange API key for JWT** via REST endpoint - get short-lived access token
3. **Connect WebSocket** with JWT token - server validates and establishes authenticated session
4. **Create session** via WebSocket specifying symbols, date range, and data provider
5. **Start simulation** with flow control settings
6. **Receive ticks** with timestamps only (no market data shared)
7. **Send orders** based on your strategy logic and local data
8. **Get realistic fills** with slippage, commissions, and latency modeling
9. **Control simulation** interactively (pause, resume, query state)
10. **Receive final results** when simulation completes
11. **Connection closes** automatically after simulation or due to timeouts/limits

🚧 Execution Engine *(TBD)*

Details on order execution simulation, slippage modeling, and fill logic to be added.

📊 Market Data Integration *(TBD)*

Supported data providers, symbol coverage, and data quality specifications to be added.

⚡ Performance & Limits *(TBD)*

Throughput specifications, concurrent session limits, and operational constraints to be added.

🔐 Authentication & Authorization

Authentication Flow

SimuTrade uses a two-step JWT-based authentication process:

Step 1: Token Exchange (REST)

```
POST /auth/token
X-API-Key: sk_live_1234567890abcdef

Response: {
  "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9...",
  "expires_in": 3600,
  "token_type": "Bearer",
  "user_id": "user_12345",
  "plan": "professional"
}
```

Step 2: WebSocket Connection

```
ws://api.simutrade.com/ws/simulate?
token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9...
```

Security Benefits

- **Easy key rotation**: Revoke API keys without disrupting active connections
- **Short-lived tokens**: Limit exposure if token is compromised (1 hour expiry)
- **Industry standard**: Aligns with broker and financial API patterns
- **Stateless validation**: Server can validate JWT without database lookups

Authorization Model

- **User Isolation**: API key identifies user and enforces access controls
- **Resource Limits**: Plan-based limits on symbols, duration, concurrent sessions
- **Rate Limiting**: Tiered limits based on user's subscription plan
- **Session Ownership**: Users can only access simulations created with their API key

API Key Management

API keys are obtained from a separate admin dashboard system that handles:

- Account management and billing
- API key generation and rotation
- Usage monitoring and rate limit configuration
- User plan management

🔄 Connection Management & Lifecycle

Connection Duration Limits

WebSocket connections have time-based limits to ensure security and resource management:

```
CONNECTION_LIMITS = {
    "free": {
        "max_connection_duration_sec": 3600,      # 1 hour
        "idle_timeout_sec": 1800,                 # 30 minutes
        "max_simulation_duration_sec": 86400,     # 24 hours
        "concurrent_connections": 1
    },
    "professional": {
        "max_connection_duration_sec": 28800,     # 8 hours
        "idle_timeout_sec": 3600,                 # 1 hour
        "max_simulation_duration_sec": 604800,    # 1 week
        "concurrent_connections": 10
    },
    "enterprise": {
        "max_connection_duration_sec": 86400,     # 24 hours
        "idle_timeout_sec": 7200,                 # 2 hours
        "max_simulation_duration_sec": 31536000,  # 1 year
        "concurrent_connections": 100
```

```
    }
}
```

Connection Closure Scenarios

Automatic Closure:

- **Simulation completion**: Connection closes after `simulation_complete` message
- **Maximum duration reached**: Connection time limit exceeded
- **Idle timeout**: No client activity for specified duration
- **API key revoked**: Key disabled in admin dashboard
- **Rate limits exceeded**: Too many requests or concurrent connections
- **Server maintenance**: Planned system maintenance

Client-Initiated Closure:

- Client application closes WebSocket connection
- Client sends explicit disconnect message

Network-Initiated Closure:

- Network disconnection or instability
- Client application crash
- Firewall or proxy timeouts

Connection Warnings & Grace Periods

Server provides warnings before forced closure:

```
// Warning 5 minutes before timeout
{
  "type": "connection_warning",
  "data": {
    "warning_type": "approaching_timeout",
    "message": "Connection will expire in 5 minutes",
    "expires_at": "2024-01-01T16:00:00Z",
    "action_required": "complete_simulation_or_reconnect"
  }
}

// Final warning 1 minute before closure
```

```
{
  "type": "connection_warning",
  "data": {
    "warning_type": "imminent_closure",
    "message": "Connection closing in 60 seconds",
    "expires_at": "2024-01-01T16:00:00Z",
    "action_required": "save_state_and_reconnect"
  }
}
```

Graceful Connection Closure

```
// Server notifies before closing
{
  "type": "connection_closing",
  "data": {
    "reason": "idle_timeout",
    "message": "Connection idle for 30 minutes",
    "close_code": 4000,
    "reconnect_allowed": true,
    "session_state": "preserved_for_resume"
  }
}
// WebSocket closes with specified code
```

⚡ Rate Limiting

Rate Limit Tiers

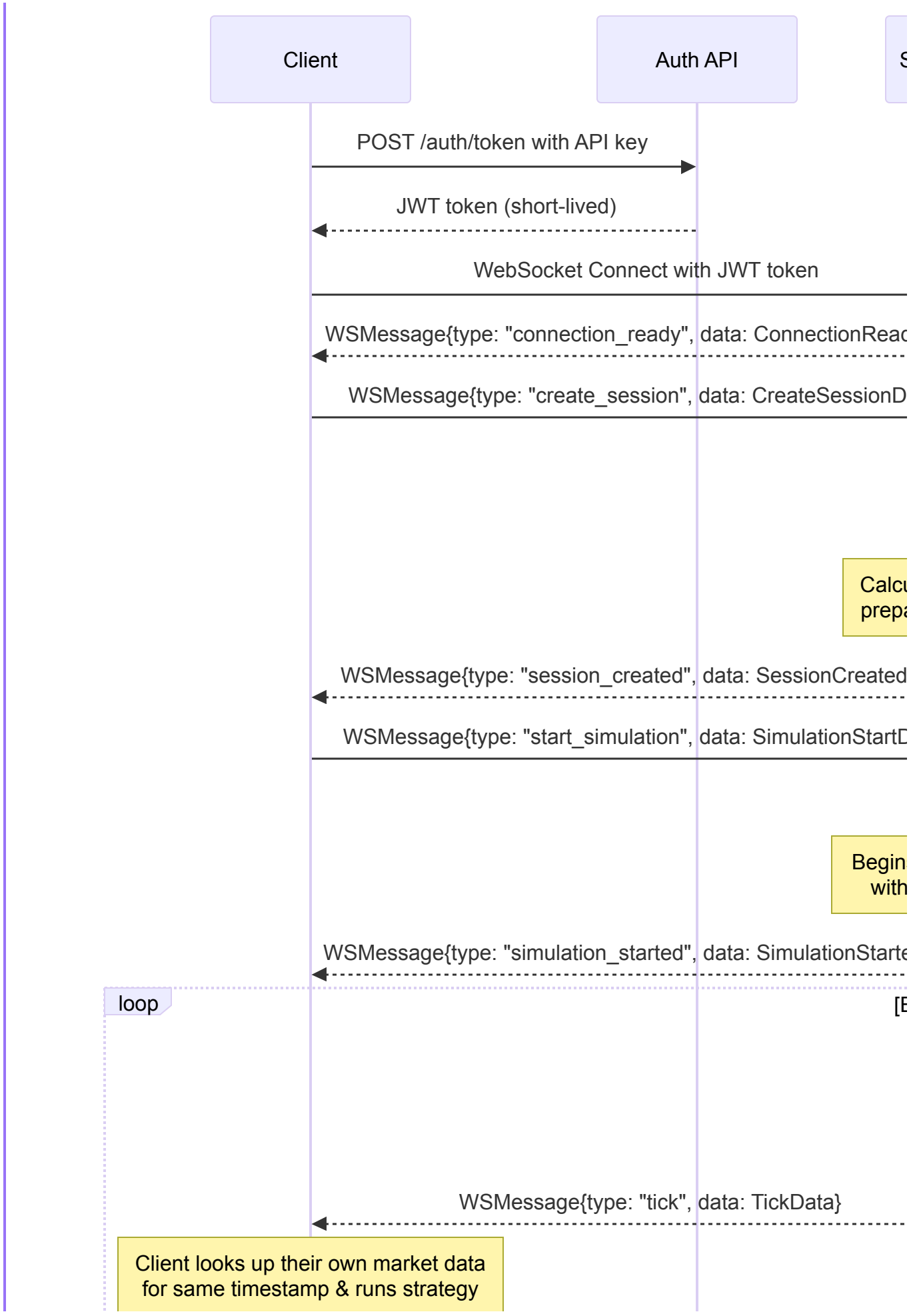
```
class UserPlan(str, Enum):
    free = "free"
    professional = "professional"
    enterprise = "enterprise"

# Rate limits per plan
RATE_LIMITS = {
    "free": {
        "sessions_per_hour": 5,
        "concurrent_sessions": 1,
        "orders_per_minute": 100,
```

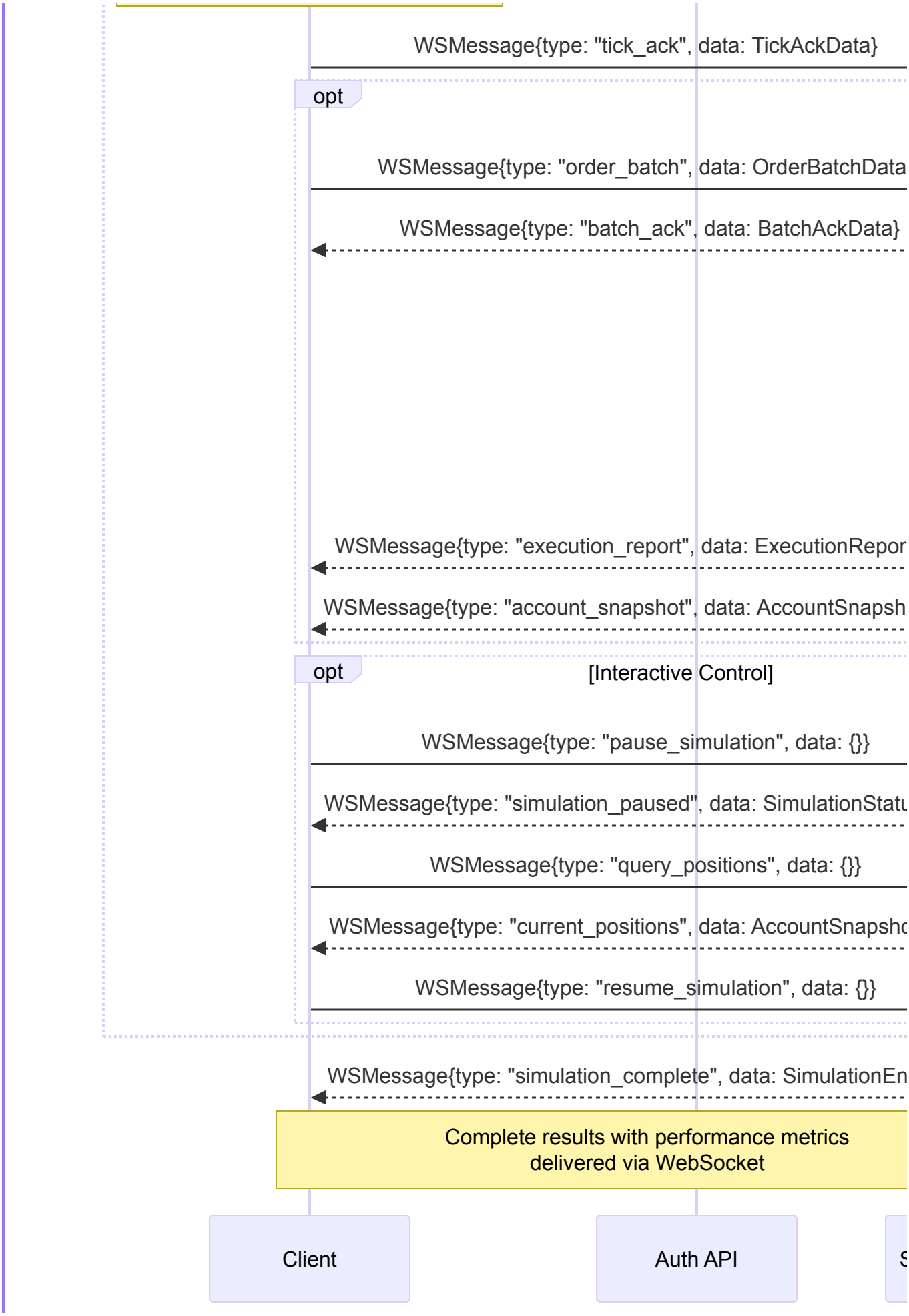
```
        "max_simulation_hours": 24,
        "max_symbols_per_session": 5,
        "websocket_connections": 1
    },
    "professional": {
        "sessions_per_hour": 50,
        "concurrent_sessions": 10,
        "orders_per_minute": 1000,
        "max_simulation_hours": 168,  # 1 week
        "max_symbols_per_session": 50,
        "websocket_connections": 10
    },
    "enterprise": {
        "sessions_per_hour": 500,
        "concurrent_sessions": 100,
        "orders_per_minute": 10000,
        "max_simulation_hours": 8760,  # 1 year
        "max_symbols_per_session": 500,
        "websocket_connections": 100
    }
}
```

Rate Limit Errors

```
{
  "type": "error",
  "data": {
    "error_code": "RATE_LIMIT_EXCEEDED",
    "message": "Too many sessions created",
    "error_type": "rate_limit",
    "severity": "error",
    "recoverable": true,
    "retry_after_ms": 3600000,
    "context": {
      "limit": "5 sessions/hour",
      "current_usage": 6,
      "reset_time": "2024-01-01T16:00:00Z",
      "plan": "free"
    }
  }
}
```

Message Flow Overview


Client
Auth API
POST /auth/token with API key
JWT token (short-lived)
WebSocket Connect with JWT token
WSMessage{type: "connection_ready", data: ConnectionReac
WSMessage{type: "create_session", data: CreateSessionD
Calcu
prepa
WSMessage{type: "session_created", data: SessionCreated
WSMessage{type: "start_simulation", data: SimulationStartD
Begin
with
WSMessage{type: "simulation_started", data: SimulationStarte
loop
[E
WSMessage{type: "tick", data: TickData}
Client looks up their own market data
for same timestamp & runs strategy


WSMessage{type: "tick_ack", data: TickAckData}
opt
WSMessage{type: "order_batch", data: OrderBatchData
WSMessage{type: "batch_ack", data: BatchAckData}
WSMessage{type: "execution_report", data: ExecutionRepor
WSMessage{type: "account_snapshot", data: AccountSnapsh
opt
[Interactive Control]
WSMessage{type: "pause_simulation", data: {}}
WSMessage{type: "simulation_paused", data: SimulationStatu
WSMessage{type: "query_positions", data: {}}
WSMessage{type: "current_positions", data: AccountSnapsho
WSMessage{type: "resume_simulation", data: {}}
WSMessage{type: "simulation_complete", data: SimulationEn
Complete results with performance metrics
delivered via WebSocket
Client
Auth API

Quick Reference

Authentication: `POST /auth/token` (API key) → JWT token → WebSocket connect
Configuration: `create_session` → `session_created` (validates symbols, loads data)
Execution: `start_simulation` → `simulation_started` (begins tick flow)
Simulation: `tick` → `tick_ack` → `order_batch` → `execution_report` → repeat
Completion: `simulation_complete` → connection closes

> **Note:** Session creation and simulation start are separate steps - configure first, then execute. This allows validation of symbols/data availability before committing to simulation execution.

Complete Simulation Lifecycle

```
1. REST: POST /auth/token (API key) → JWT token
2. WebSocket: Connect with JWT token
3. WS: connection_ready ← Server confirms authentication
4. WS: create_session → Configure simulation parameters
5. WS: session_created ← Server validates and confirms
6. WS: start_simulation → Begin execution
7. WS: simulation_started ← Server confirms start
8. WS: tick ← Server advances time (repeated)
9. WS: tick_ack → Client confirms ready (repeated)
10. WS: order_batch → Client sends orders (as needed)
11. WS: execution_report ← Server reports fills (as needed)
12. WS: simulation_complete ← Simulation finished
13. Connection closes automatically
```

Error Recovery & Reconnection

```
1. Connection drops during simulation
2. REST: POST /auth/token → Get new JWT (if expired)
3. WebSocket: Reconnect with JWT token
4. WS: connection_ready ← Server confirms reconnection
5. WS: resume_session → Request session resume
6. WS: state_sync ← Server synchronizes current state
7. Continue simulation from last processed tick
```

📋 Core Message Types

Message Type	Direction	Purpose	Requires Response
`connection_ready`	Server → Client	Connection established with limits	No
`create_session`	Client → Server	Create simulation session	Yes (`session_created`)
`session_created`	Server → Client	Session validated and ready	No
`start_simulation`	Client → Server	Begin simulation execution	Yes (`simulation_started`)
`simulation_started`	Server → Client	Simulation execution began	No
`tick`	Server → Client	Time advancement	Yes (`tick_ack`)
`tick_ack`	Client → Server	Confirm tick receipt	No
`order_batch`	Client → Server	Submit batch of orders	Yes (`batch_ack`)
`batch_ack`	Server → Client	Order batch acceptance	No
`execution_report`	Server → Client	Order fill notification	No
`account_snapshot`	Server → Client	Account state update	No
`simulation_complete`	Server → Client	Simulation finished	No
`connection_warning`	Server → Client	Connection timeout warning	No
`connection_closing`	Server → Client	Connection closure notification	No
`resume_session`	Client → Server	Resume after disconnect	Yes (`state_sync`)
`state_sync`	Server → Client	State synchronization	No
`error`	Server → Client	Error with recovery info	No

Data Models

```python
from pydantic import BaseModel, Field
from typing import Literal, Optional, List, Dict, Any
from datetime import datetime
from enum import Enum
from decimal import Decimal

# ===== CORE MESSAGE ENVELOPE =====

class WSMessage(BaseModel):
    """All WebSocket messages use this envelope."""
    type: str                           # Message type
    data: Dict[str, Any]                # Payload (simple dict for JSON safety)
    request_id: Optional[str] = None    # For request/response correlation
    timestamp: datetime                 # Message timestamp
    sequence_id: Optional[int] = None   # For message ordering
    # Authentication context (populated by server)
    user_id: Optional[str] = None       # For multi-tenant logging and authorization

# ===== ENUMS =====

class DataProvider(str, Enum):
    polygon = "polygon"
    alpaca = "alpaca"
    iex = "iex"

class UserPlan(str, Enum):
    """User subscription plans with different rate limits."""
    free = "free"
    professional = "professional"
    enterprise = "enterprise"

class OrderSide(str, Enum):
    buy = "buy"
    sell = "sell"

class OrderType(str, Enum):
    market = "market"
    limit = "limit"
    stop = "stop"
    stop_limit = "stop_limit"
```

```python
class OrderStatus(str, Enum):
    pending = "pending"
    accepted = "accepted"
    rejected = "rejected"
    partially_filled = "partially_filled"
    filled = "filled"
    cancelled = "cancelled"

class SessionState(str, Enum):
    initializing = "initializing"
    ready = "ready"
    running = "running"
    paused = "paused"
    completed = "completed"
    error = "error"

# ===== AUTHENTICATION =====

class TokenRequest(BaseModel):
    """REST API: Request JWT token (API key sent in header)."""
    # API key sent in X-API-Key header, no body needed
    pass

class TokenResponse(BaseModel):
    """REST API: JWT token response."""
    access_token: str
    expires_in: int                    # Token lifetime in seconds
    token_type: str = "Bearer"
    user_id: str
    plan: UserPlan

class UserLimitsResponse(BaseModel):
    """REST API: Current user rate limits."""
    plan: UserPlan
    limits: Dict[str, int]             # Current limits
    usage: Dict[str, int]              # Current usage
    reset_times: Dict[str, datetime]   # When limits reset

# ===== WEBSOCKET SESSION MANAGEMENT =====

class CreateSessionData(BaseModel):
    """WebSocket: Create new simulation session."""
    session_id: str
    symbols: List[str]                 # Multi-asset support
    start: datetime
```

```
    end: datetime
    data_provider: DataProvider = DataProvider.polygon
    data_version: str = "adjusted"
    initial_cash: Decimal
    commission_per_share: Decimal = Decimal("0.005")
    slippage_bps: int = 5              # Basis points

class SessionCreatedData(BaseModel):
    """WebSocket: Session creation confirmation."""
    session_id: str
    estimated_ticks: int               # Total ticks in simulation
    symbols_loaded: List[str]          # Confirmed symbols with data
available
    data_range_actual: Dict[str, Any]  # Actual data availability per
symbol

# ===== WEBSOCKET MESSAGES =====

class ConnectionReadyData(BaseModel):
    """Server confirms WebSocket connection and authentication."""
    user_id: str
    plan: UserPlan
    server_time: datetime
    connection_expires_at: datetime
    idle_timeout_sec: int
    max_simulation_duration_sec: int
    concurrent_connections_limit: int
    supported_features: List[str]      # ["batch_orders",
"partial_fills", "interactive_control"]

class ConnectionWarningData(BaseModel):
    """Server warns about impending connection closure."""
    warning_type: Literal["approaching_timeout", "imminent_closure",
"rate_limit_warning"]
    message: str
    expires_at: Optional[datetime] = None
    action_required: str
    seconds_remaining: Optional[int] = None

class ConnectionClosingData(BaseModel):
    """Server notifies about connection closure."""
    reason: Literal["idle_timeout", "max_duration", "api_key_revoked",
"rate_limit", "simulation_complete", "server_maintenance"]
    message: str
    close_code: int                    # WebSocket close code
    reconnect_allowed: bool
```

```
    session_state: Optional[str] = None  # "preserved_for_resume",
"lost", "completed"

class SimulationStartData(BaseModel):
    """Client requests simulation start."""
    flow_control: bool = True          # Enable tick acknowledgments
    max_pending_ticks: int = 1         # Backpressure control
    account_update_frequency: Literal["every_fill", "every_tick",
"on_demand"] = "every_fill"

class SimulationStartedData(BaseModel):
    """Server confirms simulation has begun."""
    session_id: str
    started_at: datetime
    estimated_duration_sec: int
    tick_interval_ms: int              # Time between ticks
    flow_control_enabled: bool

class TickData(BaseModel):
    """Server advances simulation time."""
    sim_time: datetime
    sequence_id: int                   # For ordering guarantees
    market_session: Literal["pre_market", "regular", "after_hours"]
    symbols_trading: List[str]         # Which symbols are tradeable now
    is_eod: bool = False

class TickAckData(BaseModel):
    """Client acknowledges tick and signals readiness."""
    sequence_id: int                   # Echo from TickData
    processing_status: Literal["ready", "processing", "need_time"]
    orders_pending: int = 0            # How many orders client will send
    max_wait_ms: int = 1000           # How long client needs

# ===== ORDER MANAGEMENT =====

class OrderData(BaseModel):
    """Individual order within a batch."""
    order_id: str
    symbol: str
    side: OrderSide
    type: OrderType
    quantity: Optional[int] = None
    notional: Optional[Decimal] = None  # Alternative to quantity
    price: Optional[Decimal] = None     # For limit orders
    stop_price: Optional[Decimal] = None
    time_in_force: Literal["day", "gtc", "ioc"] = "day"
```

```python
class OrderBatchData(BaseModel):
    """Client submits batch of orders."""
    batch_id: str
    orders: List[OrderData]
    execution_mode: Literal["atomic", "best_effort"] = "best_effort"
    parent_strategy: Optional[str] = None  # For tracking

class BatchAckData(BaseModel):
    """Server acknowledges order batch."""
    batch_id: str
    accepted_orders: List[str]         # order_ids that were accepted
    rejected_orders: Dict[str, str]    # order_id -> rejection_reason
    estimated_fills: Dict[str, Decimal] # order_id ->
estimated_fill_price

class ExecutionReportData(BaseModel):
    """Server reports order execution."""
    execution_id: str
    order_id: str
    symbol: str
    side: OrderSide
    executed_quantity: int
    executed_price: Decimal
    execution_time: datetime
    commission: Decimal
    slippage_bps: int
    is_partial: bool
    remaining_quantity: int
    fill_reason: str                   # "market_open", "limit_touched",
etc.

# ===== ACCOUNT & PORTFOLIO =====

class PositionData(BaseModel):
    """Current position in a symbol."""
    symbol: str
    quantity: int                      # Positive = long, negative =
short
    avg_cost: Decimal
    market_value: Decimal
    unrealized_pnl: Decimal

class AccountSnapshotData(BaseModel):
    """Current account state."""
    cash: Decimal
```

```
    equity: Decimal
    buying_power: Decimal
    day_pnl: Decimal
    positions: List[PositionData]
    open_orders: int

# ===== ERROR HANDLING =====

class ErrorData(BaseModel):
    """Enhanced error reporting."""
    error_code: str                    # "INVALID_ORDER",
"INSUFFICIENT_FUNDS", etc.
    message: str
    error_type: Literal["validation", "execution", "connection", "data"]
    severity: Literal["warning", "error", "fatal"]
    recoverable: bool
    retry_after_ms: Optional[int] = None
    failed_request_id: Optional[str] = None
    context: Dict[str, Any] = {}

# ===== SESSION RECOVERY =====

class ResumeSessionData(BaseModel):
    """Client requests session resume after disconnect."""
    session_id: str
    last_processed_sequence_id: int
    client_state_hash: Optional[str] = None

class StateSyncData(BaseModel):
    """Server provides current state for recovery."""
    current_sim_time: datetime
    next_sequence_id: int
    account_snapshot: AccountSnapshotData
    pending_orders: List[str]          # order_ids still open
    missed_executions: List[ExecutionReportData]  # Fills during
disconnect

# ===== SIMULATION END =====

class SimulationEndData(BaseModel):
    """Final simulation results."""
    session_id: str
    final_equity: Decimal
    total_return_pct: Decimal
    total_trades: int
    winning_trades: int
```

```
    total_commission: Decimal
    max_drawdown_pct: Decimal
    sharpe_ratio: Optional[Decimal] = None
    sortino_ratio: Optional[Decimal] = None
    calmar_ratio: Optional[Decimal] = None
    simulation_duration_sec: int
```

🔄 Complete Message Flow Examples

Example 1: Complete Simulation Flow

```
// 1. REST: Exchange API key for JWT token
POST /auth/token
X-API-Key: sk_live_1234567890abcdef

Response: {
  "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9...",
  "expires_in": 3600,
  "token_type": "Bearer",
  "user_id": "user_12345",
  "plan": "professional"
}

// 2. WebSocket: Connect with JWT token
ws://api.simutrade.com/ws/simulate?
token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9...

→ {"type": "connection_ready", "data": {
    "user_id": "user_12345",
    "plan": "professional",
    "connection_expires_at": "2024-01-01T22:30:00Z",
    "idle_timeout_sec": 3600,
    "max_simulation_duration_sec": 604800,
    "concurrent_connections_limit": 10
  }}

// 2. WebSocket: Create session
← {"type": "create_session", "data": {
    "session_id": "strategy_test_001",
    "symbols": ["AAPL", "MSFT"],
    "start": "2024-01-01T14:30:00Z",
    "end": "2024-01-01T21:00:00Z",
    "data_provider": "polygon",
```

```
    "initial_cash": "100000.00"
  }, "request_id": "req_001"}

→ {"type": "session_created", "data": {
    "session_id": "strategy_test_001",
    "estimated_ticks": 390,
    "symbols_loaded": ["AAPL", "MSFT"]
  }, "request_id": "req_001"}

// 3. WebSocket: Start simulation
← {"type": "start_simulation", "data": {
    "flow_control": true,
    "max_pending_ticks": 1
  }, "request_id": "req_002"}

→ {"type": "simulation_started", "data": {
    "session_id": "strategy_test_001",
    "started_at": "2024-01-01T14:30:00Z",
    "flow_control_enabled": true
  }, "request_id": "req_002"}

// 4. Simulation loop
→ {"type": "tick", "data": {"sim_time": "2024-01-01T14:30:00Z",
"sequence_id": 1}}
← {"type": "tick_ack", "data": {"sequence_id": 1, "processing_status":
"ready", "orders_pending": 2}}

← {"type": "order_batch", "data": {
    "batch_id": "batch_001",
    "orders": [
      {"order_id": "ord_001", "symbol": "AAPL", "side": "buy", "type":
"market", "quantity": 100},
      {"order_id": "ord_002", "symbol": "MSFT", "side": "buy", "type":
"limit", "quantity": 50, "price": "380.00"}
    ]
  }, "request_id": "req_003"}

→ {"type": "batch_ack", "data": {
    "batch_id": "batch_001",
    "accepted_orders": ["ord_001", "ord_002"],
    "rejected_orders": {},
    "estimated_fills": {"ord_001": "180.25", "ord_002": "380.00"}
  }, "request_id": "req_003"}

→ {"type": "execution_report", "data": {
    "execution_id": "exec_001",
```

```
    "order_id": "ord_001",
    "symbol": "AAPL",
    "executed_quantity": 100,
    "executed_price": "180.27",
    "commission": "0.50",
    "slippage_bps": 1
  }}

→ {"type": "account_snapshot", "data": {
    "cash": "81972.50",
    "equity": "100000.00",
    "positions": [{"symbol": "AAPL", "quantity": 100, "avg_cost":
"180.27"}]
  }}

// 5. Simulation completion
→ {"type": "simulation_complete", "data": {
    "session_id": "strategy_test_001",
    "final_equity": "105000.00",
    "total_trades": 25,
    "sharpe_ratio": "1.85"
  }}

// 6. Connection closes automatically after simulation
```

Example 2: Session Resume After Disconnection

```
// Client reconnects after network issue
// First get new JWT token (if previous one expired)
POST /auth/token
X-API-Key: sk_live_1234567890abcdef

Response: {"access_token": "eyJ...", "expires_in": 3600}

// Then reconnect with JWT token
ws://api.simutrade.com/ws/simulate?
token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9...

→ {"type": "connection_ready", "data": {...}}

// Client requests session resume
← {"type": "resume_session", "data": {
    "session_id": "strategy_test_001",
    "last_processed_sequence_id": 45
```

```
  }, "request_id": "req_resume"}

→ {"type": "state_sync", "data": {
    "current_sim_time": "2024-01-01T15:15:00Z",
    "next_sequence_id": 46,
    "account_snapshot": {...},
    "missed_executions": [
      {"execution_id": "exec_012", "order_id": "ord_008", ...}
    ]
  }, "request_id": "req_resume"}

// Continue normal simulation from sequence_id 46
```

🛡️ Error Handling Strategy

Error Categories

```
# Validation Errors (4xx equivalent)
{
  "type": "error",
  "data": {
    "error_code": "INVALID_ORDER_QUANTITY",
    "message": "Order quantity must be positive",
    "error_type": "validation",
    "severity": "error",
    "recoverable": true,
    "failed_request_id": "req_002",
    "context": {"order_id": "ord_001", "quantity": -100}
  }
}

# Execution Errors (business logic)
{
  "type": "error",
  "data": {
    "error_code": "INSUFFICIENT_BUYING_POWER",
    "message": "Not enough cash to execute order",
    "error_type": "execution",
    "severity": "warning",
    "recoverable": true,
    "context": {"required": "50000.00", "available": "25000.00"}
  }
}
```

```
# System Errors (5xx equivalent)
{
  "type": "error",
  "data": {
    "error_code": "DATA_PROVIDER_UNAVAILABLE",
    "message": "Unable to fetch market data",
    "error_type": "data",
    "severity": "fatal",
    "recoverable": false,
    "retry_after_ms": 30000
  }
}
```