Stock Flow AI API Documentation

# Authentication APIs

## 1. Register User

**Endpoint:** POST /api/auth/register

**Input:**

* UserCreate schema (email, username, password)

**Output:**

* UserOut schema (id, email, username, role, approval\_status)

**Description:** Registers a new client user with pending approval status

## 2. Login

**Endpoint:** POST /api/auth/login

**Input:**

* OAuth2 form data (username, password)

**Output:**

* Token schema (access\_token, token\_type, user\_id, role)

**Description:** Authenticates a user and returns a JWT token

# Client APIs

## 1. Get Stocks by User ID

**Endpoint:** GET /api/client/${userId}?stocks

**Input:**

* Path parameter: userId

**Output:**

* List of OwnedStockResponse

**Description:** Gets all stocks owned by a specific user

## 2. Get My Stocks

**Endpoint:** GET /api/client/my-stocks

**Input:**

* Requires client authentication

**Output:**

* List of OwnedStockResponse

**Description:** Gets stocks owned by the currently authenticated client

# Stock Query APIs

## 1. Query Stock

**Endpoint:** POST /api/stock/query

**Input:**

* StockQuery schema (query text, query type)

**Output:**

* StockResponse (AI analysis, stock data)

**Description:** Processes natural language queries about stocks and returns AI-powered analysis

# PDF Document APIs

## 1. Upload PDF

**Endpoint:** POST /api/pdf/upload

**Input:**

* File: PDF document
* Form: document\_name (optional)

**Output:**

* DocumentUploadResponse (success status, document info)

**Description:** Uploads and processes a PDF document for future querying

## 2. Query PDF

**Endpoint:** POST /api/pdf/query

**Input:**

* DocumentQuery schema (query text)

**Output:**

* DocumentQueryResponse (AI-generated answer)

**Description:** Queries uploaded PDF documents using natural language

# Cart APIs

## 1. Add to Cart

**Endpoint:** POST /api/cart/

**Input:**

* StockCartCreate schema (symbol, name, quantity, price, trade\_type)

**Output:**

* StockCartOut (cart item details with ID)

**Description:** Adds a stock to the user's cart

## 2. Get Cart

**Endpoint:** GET /api/cart/

**Input:**

* Requires client authentication

**Output:**

* List of StockCartOut (all items in cart)

**Description:** Retrieves all items in the user's cart

## 3. Remove from Cart

**Endpoint:** DELETE /api/cart/{cart\_id}

**Input:**

* Path parameter: cart\_id

**Output:**

* JSON message confirming removal

**Description:** Removes a specific item from the cart

## 4. Place Orders from Cart

**Endpoint:** POST /api/cart/place-orders

**Input:**

* Optional body parameter: cart\_ids (list of specific cart IDs to process)

**Output:**

* List of TradeRequestOut (created trade requests)

**Description:** Converts cart items into trade requests and removes them from cart

# Trade APIs

## 1. Create Trade Request

**Endpoint:** POST /api/trade/

**Input:**

* TradeRequestCreate schema (symbol, quantity, price, trade\_type)

**Output:**

* TradeRequestOut (created trade request with ID)

**Description:** Creates a new trade request directly (bypassing cart)

## 2. Get Trade Requests

**Endpoint:** GET /api/trade/

**Input:**

* Requires client authentication

**Output:**

* List of TradeRequestOut (all trade requests for the current user)

**Description:** Retrieves all trade requests for the current user

# Admin APIs

## 1. Get Clients

**Endpoint:** GET /api/admin/clients

**Input:**

* Requires admin authentication

**Output:**

* List of UserOut (all client users)

**Description:** Gets all client users in the system

## 2. Get Pending Registrations

**Endpoint:** GET /api/admin/pending-registrations

**Input:**

* Requires admin authentication

**Output:**

* List of UserOut (clients pending approval)

**Description:** Gets all client registrations pending approval

## 3. Approve Client

**Endpoint:** POST /api/admin/approve-client/{user\_id}

**Input:**

* Path parameter: user\_id

**Output:**

* UserOut (updated user with approved status)

**Description:** Approves a pending client registration

## 4. Reject Client

**Endpoint:** POST /api/admin/reject-client/{user\_id}

**Input:**

* Path parameter: user\_id

**Output:**

* UserOut (updated user with rejected status)

**Description:** Rejects a pending client registration

## 5. Create Client

**Endpoint:** POST /api/admin/clients

**Input:**

* UserCreate schema (email, username, password)

**Output:**

* UserOut (created client user)

**Description:** Admin creates a new client user (auto-approved)

## 6. Delete Client

**Endpoint:** DELETE /api/admin/clients/{user\_id}

**Input:**

* Path parameter: user\_id

**Output:**

* JSON message confirming deletion

**Description:** Deletes a client user from the system

## 7. Get All Trade Requests

**Endpoint:** GET /api/admin/trade-requests

**Input:**

* Requires admin authentication

**Output:**

* List of TradeRequestOut (all trade requests in system)

**Description:** Retrieves all trade requests across all users

## 8. Update Trade Status

**Endpoint:** PUT /api/admin/trade-requests/{trade\_id}/status

**Input:**

* Path parameter: trade\_id
* Query parameter: status (new trade status)

**Output:**

* TradeRequestOut (updated trade request)

**Description:** Updates the status of a trade request

## 9. Get Activity Logs

**Endpoint:** GET /api/admin/activity-logs

**Input:**

* Requires admin authentication

**Output:**

* List of activity logs

**Description:** Retrieves system activity logs

# WebSocket APIs

## 1. Chat WebSocket

**Endpoint:** WebSocket /ws/chat

**Input:**

* Authentication via token in query parameter
* JSON messages for communication

**Output:**

* Real-time messages via WebSocket

**Description:** Real-time chat between clients and admins

## 2. Debug Token

**Endpoint:** GET /api/ws/debug/token

**Input:**

* Requires authentication

**Output:**

* JSON with auth status and user info

**Description:** For debugging authentication issues

## 3. Get Chat Partners

**Endpoint:** GET /api/ws/chat/partners

**Input:**

* Requires authentication

**Output:**

* List of available chat partners

**Description:** Gets available chat partners for the current user

# Guidelines for API Usage

## 1. Authentication

- All endpoints (except registration and login) require a valid JWT token

- Provide token in the Authorization header as: Bearer <token>

- Client endpoints require client role, admin endpoints require admin role

## 2. Error Handling

- APIs return standard HTTP status codes (200, 400, 401, 403, 404, 500)

- Error responses include a detail message explaining the issue

## 3. Rate Limiting

- Be mindful of rate limits for external API services like AlphaVantage and Gemini

## 4. Data Validation

- All inputs are validated according to their Pydantic schemas

- Follow the exact schema requirements to avoid validation errors

## 5. WebSocket Communication

- For real-time features, establish a WebSocket connection

- Handle connection, disconnection, and message events appropriately