Advanced Task: Broker Wrappers - Building a Trading Interface with Requests Library

Objective: To develop a Python script that acts as a wrapper for interacting with a broker's API using the requests library. This task will require understanding RESTful APIs, managing authentication, sending trade orders, and retrieving account and market data. The goal is to build a flexible and reusable broker wrapper that can be integrated into an algorithmic trading system.

Task Description:

1. **Broker API Documentation:**
   - Choose a broker with a well-documented API (e.g., Alpaca, Interactive Brokers, or any other broker with a REST API).
   - Study the API documentation to understand the required endpoints, authentication mechanisms, and data structures.
2. **Environment Setup:**
   - Install necessary Python packages (requests, json, etc.).
   - Set up a .env file or use environment variables to securely store API keys and other sensitive information.
3. **Authentication:**
   - Implement a function to handle API authentication. Depending on the broker, this may involve:
     - API Key authentication: Include the API key in the header of each request.
     - OAuth2 authentication: Implement token-based authentication if required by the broker.
   - Ensure that your wrapper handles token expiration and renewal if using OAuth2.
4. **Account Information Retrieval:**
   - Create functions to retrieve account information such as balance, open positions, and order history.
   - Example:
     - get_account_balance(): Fetches and returns the current account balance.
     - get_open_positions(): Retrieves a list of all open positions in the account.
5. **Market Data Retrieval:**
   - Implement functions to fetch real-time and historical market data for selected assets.
   - Example:
     - get_market_data(symbol, start_date, end_date): Fetches historical price data for a given symbol.

        - get_realtime_quote(symbol): Retrieves the latest price and volume information for a given symbol.
    - Ensure that the data is returned in a format compatible with Pandas DataFrames for easy analysis.
6. **Order Placement:**
    - Develop functions to place, modify, and cancel trade orders.
    - Example:
        - place_order(symbol, quantity, order_type, side): Places a buy/sell order for a specified symbol.
        - modify_order(order_id, new_quantity): Modifies an existing order.
        - cancel_order(order_id): Cancels an open order.
    - Implement error handling to manage unsuccessful order placements and API rate limits.
7. **Advanced Functionality:**
    - Implement support for advanced order types, such as limit orders, stop-loss orders, and trailing stops.
    - Example:
        - place_limit_order(symbol, quantity, price, side): Places a limit order at the specified price.
        - place_stop_loss_order(symbol, quantity, stop_price): Places a stop-loss order.
    - Add a function to retrieve the status of an order and check if it was filled, partially filled, or canceled.
8. **Logging and Error Handling:**
    - Implement a logging mechanism to track all API requests and responses. Include timestamps and status codes.
    - Develop comprehensive error handling for common API issues such as connectivity problems, rate limits, and invalid inputs.
    - Example:
        - log_api_call(endpoint, status_code, response_time): Logs each API call with its outcome.
        - handle_api_error(response): Checks the response for errors and raises appropriate exceptions.
9. **Unit Testing:**
    - Create unit tests for all the functions using a testing framework like unittest or pytest.

    - Use mock objects to simulate API responses without making real API calls during testing.
    - Example:
        - Test cases for successful and failed order placements.
        - Test cases for correct data retrieval and processing.

10. **Deployment and Integration (Optional but Recommended):**
    - Package your broker wrapper as a Python module that can be easily integrated into other projects.
    - Provide documentation or a README file explaining how to use the wrapper, including examples of common tasks like placing an order and fetching market data.
    - Optionally, set up continuous integration (CI) using GitHub Actions or another CI service to automatically run tests on each commit.

Deliverables:

- A well-documented Python script or module that includes all the functions for interacting with the broker's API.
- Unit tests for all the functions with clear coverage of edge cases and potential errors.
- A README file or notebook that explains how to use the broker wrapper with example code snippets.
- Logs of API interactions showing successful and failed requests with appropriate error handling.