# A blue sky with many trees AI-generated content may be incorrect.A logo with white and green circles AI-generated content may be incorrect.emsTradepoint API Guide

emsTradepoint Limited

API User Guide

June 2025

This document is a comprehensive guide to integrating with emsTradepoint's Exchange API for gas and carbon trading.

The emsTradepoint (ETP) API provides programmatic access to New Zealand's leading energy trading platform, enabling automated trading, market data analysis, and portfolio management for natural gas and carbon markets.

For detailed parameter specifications and response schemas, refer to the interactive API documentation.

## HOW THIS DOCUMENT IS STRUCTURED

This guide is organised to take you from initial setup through to production deployment:

**Getting Started**: Authentication methods, base URLs, and your first API calls to establish connectivity.

**Core API Operations**: Comprehensive coverage of all available endpoints including company information, product listings, trading operations (bids, offers, orders), market data access, and trading analytics.

**API Reference:** Technical specifications for request parameters, response formats, error codes, and rate limiting policies.

**Testing and Development**: Detailed guidance on using the interactive Swagger documentation, UAT environment testing strategies, and production access requirements.

**Best Practices**: Production-ready code patterns, error handling strategies, and complete workflow examples.

## Code Disclaimer

All Python code examples in this document are provided for illustration purposes only to demonstrate API concepts and integration patterns. These examples are not intended to be used directly in production environments without proper testing, error handling, security considerations, and customisation for your specific use case. Always thoroughly test and validate any code in the UAT environment before implementing in production systems.

## WHAT YOU CAN DO

* **Trade Programmatically**: Submit bids, offers, and manage orders automatically
* **Monitor Markets**: Access real-time market data, pricing, and trading activity
* **Analyse Data**: Retrieve historical trades, indices, and market statistics
* **Manage Positions**: Track your trading positions and exposure across products
* **Integrate Systems**: Connect your existing trading and risk management systems

## MARKET COVERAGE

The API provides access to:

* **Gas Markets**: On The Day (OTD) and Day Ahead (DA) natural gas trading
* **Carbon Markets**: New Zealand Unit (NZU) carbon credit trading
* **Market Indices**: FRMI/FRQI for gas, ECMI/ECQI for carbon

## WHO SHOULD USE THIS API

* **Energy Traders**: Automate trading strategies and market monitoring
* **Portfolio Managers**: Track positions and analyse market exposure
* **Risk Managers**: Monitor trading limits and compliance requirements
* **System Integrators**: Connect trading platforms with back-office systems
* **Data Analysts**: Access market data for research and reporting

## Prerequisites

Before using the API, ensure you have:

* An active ETP account
* API credentials (client ID and secret) from ETP
* Appropriate trading permissions for target markets
* Understanding of the underlying market products and trading rules

## Important Legal Notice

### ⚠️ Market Rules Compliance

Access to the ETP APIs and all underlying market data is strictly governed by ETP's Market Rules. By using these APIs, you acknowledge and agree that:

* All API usage must comply with the current Market Rules and any amendments
* You are responsible for understanding and adhering to trading regulations
* Market data usage is subject to licensing terms and restrictions
* Trading activities through the API are bound by the same rules as manual trading
* Violations of Market Rules may result in API access suspension or termination

**Before proceeding with API integration, please ensure you have reviewed and understood the Market Rules. Contact emsTradepoint if you have questions about compliance requirements.**

# Quick Start

## Base URLs

* **Production**: https://exchange.emsTradepoint.co.nz/api/v1
* **UAT/Testing**: https://uat-exchange.emsTradepoint.co.nz/api/v1

## Authentication

All API requests require OAuth 2.0 authentication. Access tokens are valid for 2 hours.

# Getting Started

## Authentication Methods

### Server-to-Server (B2B)

For backend integrations, authenticate directly with your credentials.

import requests

import json

def get\_access\_token(client\_id, client\_secret, base\_url):

"""Get OAuth2 access token for B2B authentication"""

url = f"{base\_url}/oauth/token"

payload = {

"client\_id": client\_id,

"client\_secret": client\_secret,

"grant\_type": "client\_credentials"

}

headers = {"Content-Type": "application/json"}

response = requests.post(url, json=payload, headers=headers)

response.raise\_for\_status()

return response.json()["access\_token"]

# Usage

BASE\_URL = "https://exchange.emsTradepoint.co.nz"

token = get\_access\_token("your\_client\_id", "your\_client\_secret", BASE\_URL)

### Third-Party Applications

For user-facing applications, redirect users to:

def get\_authorization\_url(base\_url, client\_id, redirect\_uri):

"""Generate authorization URL for third-party authentication"""

return f"{base\_url}/oauth/authorize?client\_id={client\_id}&redirect\_uri={redirect\_uri}&response\_type=code"

# Usage

auth\_url = get\_authorization\_url(BASE\_URL, "your\_client\_id", "https://yourapp.com/callback")

print(f"Redirect user to: {auth\_url}")

### API-Only Accounts

Request a dedicated read-only account with non-expiring credentials for automated systems.

## Making Authenticated Requests

Include your access token in the Authorization header:

class EmsTradePointAPI:

def \_\_init\_\_(self, base\_url, access\_token):

self.base\_url = base\_url

self.headers = {

"Authorization": f"Bearer {access\_token}",

"Content-Type": "application/json"

}

def \_make\_request(self, method, endpoint, \*\*kwargs):

"""Make authenticated API request with error handling"""

url = f"{self.base\_url}/api/v1{endpoint}"

try:

response = requests.request(method, url, headers=self.headers, \*\*kwargs)

response.raise\_for\_status()

return response.json()

except requests.exceptions.HTTPError as e:

if response.status\_code == 429:

print("Rate limit exceeded. Implement backoff strategy.")

elif response.status\_code == 401:

print("Authentication failed. Token may be expired.")

elif response.status\_code == 422:

print(f"Validation error: {response.json().get('error', 'Unknown error')}")

raise

except requests.exceptions.RequestException as e:

print(f"Request failed: {e}")

raise

# Initialize API client

api = EmsTradePointAPI(BASE\_URL, token)

# API Endpoints

## Company Information

Get your company details and trading permissions.

def get\_company\_info(api):

"""Get company information and trading permissions"""

return api.\_make\_request("GET", "/companies/ours")

# Usage

company\_info = get\_company\_info(api)

print(f"Company ID: {company\_info['company\_id']}")

print(f"User IDs: {company\_info['user\_ids']}")

## Products & Markets

### List Available Products

def get\_products(api, market\_category=None):

"""Get list of available trading products"""

params = {}

if market\_category:

params["market\_category"] = market\_category

return api.\_make\_request("GET", "/products", params=params)

# Get gas products (default)

gas\_products = get\_products(api)

# Get carbon products

carbon\_products = get\_products(api, market\_category="NZU")

# Get both gas and carbon products

all\_products = get\_products(api, market\_category="NZU,NGP")

### Get Product Details

def get\_product\_details(api, product\_id):

"""Get detailed product information including bid/offer stack"""

return api.\_make\_request("GET", f"/products/{product\_id}")

# Usage

product\_details = get\_product\_details(api, "product\_123")

print(f"Product: {product\_details['name']}")

print(f"Current price stack: {product\_details['price\_stack']}")

### Market Status

def get\_market\_status(api):

"""Get current state of all markets (OTD, DA, NZU)"""

return api.\_make\_request("GET", "/markets")

# Usage

markets = get\_market\_status(api)

for market in markets:

print(f"{market['name']}: {market['status']}")

## Trading Operations

### Orders

View and manage your trading orders:

def get\_orders(api, market\_category="gas", filters=None):

"""Get trading orders with optional filters"""

endpoint = "/orders" if market\_category == "gas" else "/orders/nzu"

params = filters or {}

return api.\_make\_request("GET", endpoint, params=params)

def withdraw\_all\_orders(api):

"""Withdraw all live orders (bids and offers)"""

return api.\_make\_request("PATCH", "/withdraw/all")

# Get gas orders

gas\_orders = get\_orders(api, market\_category="gas")

# Get carbon orders with date filter

from datetime import datetime

today = datetime.now().strftime("%Y-%m-%d")

carbon\_orders = get\_orders(api, market\_category="carbon",

filters={"date\_from": today})

# Emergency: withdraw all live orders

# withdraw\_all\_orders(api)

### Bids

def submit\_bid(api, bid\_data):

"""Submit a new bid to the trading stack"""

return api.\_make\_request("POST", "/bids", json=bid\_data)

def get\_bid\_details(api, bid\_id):

"""Get details for a specific bid"""

return api.\_make\_request("GET", f"/bids/{bid\_id}")

def withdraw\_bid(api, bid\_id):

"""Withdraw a specific bid"""

return api.\_make\_request("PATCH", f"/bids/{bid\_id}/withdraw")

# Submit a new bid

bid\_data = {

"product\_id": "NGP\_OTD\_AUCK",

"price": 12.50,

"quantity": 1000,

"delivery\_date": "2025-06-05"

}

try:

new\_bid = submit\_bid(api, bid\_data)

bid\_id = new\_bid["id"]

print(f"Bid submitted successfully. ID: {bid\_id}")

# Get bid details

bid\_details = get\_bid\_details(api, bid\_id)

print(f"Bid status: {bid\_details['status']}")

except requests.exceptions.HTTPError as e:

print(f"Failed to submit bid: {e}")

### Offers

def submit\_offer(api, offer\_data):

"""Submit a new offer to the trading stack"""

return api.\_make\_request("POST", "/offers", json=offer\_data)

def get\_offer\_details(api, offer\_id):

"""Get details for a specific offer"""

return api.\_make\_request("GET", f"/offers/{offer\_id}")

def withdraw\_offer(api, offer\_id):

"""Withdraw a specific offer"""

return api.\_make\_request("PATCH", f"/offers/{offer\_id}/withdraw")

# Submit a new offer

offer\_data = {

"product\_id": "NGP\_OTD\_AUCK",

"price": 13.00,

"quantity": 500,

"delivery\_date": "2025-06-05"

}

try:

new\_offer = submit\_offer(api, offer\_data)

offer\_id = new\_offer["id"]

print(f"Offer submitted successfully. ID: {offer\_id}")

except requests.exceptions.HTTPError as e:

if e.response.status\_code == 422:

error\_detail = e.response.json().get("error", "Unknown validation error")

print(f"Offer validation failed: {error\_detail}")

## Trading Data

### Trades

View completed transactions

def get\_trades(api, market\_category="gas", filters=None):

"""Get completed trades with optional filters"""

endpoint = "/trades" if market\_category == "gas" else "/trades/nzu"

params = filters or {}

return api.\_make\_request("GET", endpoint, params=params)

# Get recent gas trades

gas\_trades = get\_trades(api, market\_category="gas")

# Get carbon trades for specific date range

from datetime import datetime, timedelta

yesterday = (datetime.now() - timedelta(days=1)).strftime("%Y-%m-%d")

today = datetime.now().strftime("%Y-%m-%d")

carbon\_trades = get\_trades(api, market\_category="carbon",

filters={

"date\_from": yesterday,

"date\_to": today

})

for trade in carbon\_trades[:5]: # Show first 5 trades

print(f"Trade: {trade['quantity']} units at ${trade['price']}")

### Market Activity

Get real-time market updates:

def get\_ticker(api, data\_type="all"):

"""Get latest market activity (trades, orders, or both)"""

endpoints = {

"all": "/ticker",

"trades": "/ticker/trades",

"orders": "/ticker/orders"

}

return api.\_make\_request("GET", endpoints[data\_type])

# Get latest 20 transactions of all types

latest\_activity = get\_ticker(api, "all")

# Get only the latest trades

latest\_trades = get\_ticker(api, "trades")

# Get only the latest orders

latest\_orders = get\_ticker(api, "orders")

print("Recent Market Activity:")

for activity in latest\_activity[:5]:

print(f"{activity['type']}: {activity['product']} - ${activity['price']}")

### Market Indices

Access historical pricing indices

def get\_indices(api, market\_category="gas", date\_range=None):

"""Get market indices (FRMI/FRQI for gas, ECMI/ECQI for carbon)"""

endpoint = "/index" if market\_category == "gas" else "/index/nzu"

params = date\_range or {}

return api.\_make\_request("GET", endpoint, params=params)

# Get latest gas indices

gas\_indices = get\_indices(api, market\_category="gas")

# Get carbon indices for specific month

carbon\_indices = get\_indices(api, market\_category="carbon",

date\_range={

"date\_from": "2025-05-01",

"date\_to": "2025-05-31"

})

print("Gas Market Indices:")

for index in gas\_indices:

print(f"{index['period']}: ${index['value']} ({index['type']})")

### Off-Market Trades

Submit trades executed outside the exchange:

def submit\_off\_market\_trade(api, trade\_data):

"""Submit an off-market trade for reporting"""

return api.\_make\_request("POST", "/off\_market\_trades", json=trade\_data)

# Submit off-market trade

off\_market\_trade = {

"product\_id": "NGP\_DA\_AUCK",

"price": 11.75,

"quantity": 2000,

"delivery\_date": "2025-06-06",

"counterparty": "Company XYZ",

"trade\_date": "2025-06-04"

}

try:

result = submit\_off\_market\_trade(api, off\_market\_trade)

print(f"Off-market trade submitted: {result['trade\_id']}")

except requests.exceptions.HTTPError as e:

print(f"Failed to submit off-market trade: {e}")

## Request Parameters and Filters

### Pagination

* Default: 30 records per page
* Maximum: 30 records per page (cannot be increased)
* Use per\_page parameter to request fewer records
* Paginated responses include navigation links in the Link header

### Market Categories

By default, endpoints return gas market data. Use these filters to specify markets:

* market\_category=NZU - Carbon market only
* market\_category=NZU, NGP - Both carbon and gas markets
* No parameter - Gas market only (default)

**Note**: You'll receive a 401 error if you don't have permission to access requested markets.

### Additional Filters

Most endpoints support additional filtering parameters. Check the [live API documentation](https://uat-exchange.emstradepoint.co.nz/api/docs) for endpoint-specific filters.

## Response Format

### Success Responses

**200 OK - Data Retrieved**

{

"data": {

// Your requested data

}

}

**201 Created - Data Submitted**

{

"data": {

// Details of created resource

}

}

## Error Responses

### 400 Bad Request - Invalid Input

{

"error": "Invalid date format provided"

}

**Common causes**: Wrong date format, text where number expected, missing required fields.

### 401 Unauthorized - Authentication Failed

{

"error": "Invalid or expired access token"

}

Your access token is missing, invalid, or expired.

422 Unprocessable Entity - Business Logic Error

{

"error": "Validation failed: Exceeds trading limit"

}

Request is valid but violates business rules (trading limits, insufficient permissions, etc.).

### 429 Too Many Requests - Rate Limited

{

"error": "Rate limit exceeded"

}

You've exceeded the requests-per-minute limit. Implement exponential backoff retry logic.

## Rate Limiting

The API implements rate limiting to ensure fair usage. When you hit the limit:

* You'll receive a 429 status code
* Implement retry logic with exponential backoff
* Monitor your request frequency to stay within limits

# Testing & Documentation

## Interactive API Documentation (Swagger)

The ETP API provides comprehensive interactive documentation powered by Swagger/OpenAPI. This is your primary resource for understanding API specifications, testing endpoints, and validating your integration approach.

## Access the Documentation

* **UAT Environment**: <https://uat-exchange.emstradepoint.co.nz/api/docs>

## How to Use Swagger Documentation

### 1. Explore API Specifications

* **Endpoint Details**: Each endpoint shows HTTP methods, parameters, and response schemas
* **Request Examples**: Copy-paste ready JSON payloads for testing
* **Response Schemas**: Understand exactly what data you'll receive
* **Error Responses**: See all possible error conditions and formats

### 2. Interactive Testing

* Before writing code, test endpoints directly in Swagger:
* Click "Try it out" on any endpoint
* Fill in required parameters
* Execute the request
* Review the response format and data

### 3. Authentication Setup

* Use the "Authorize" button in Swagger to set your bearer token
* Test authenticated endpoints without writing code first
* Verify your token permissions across different market categories

### 4. Parameter Validation

* Test different filter combinations
* Understand pagination behaviour
* Validate date formats and ranges
* Test market category filters (gas vs carbon)

## Swagger Best Practices

### Before Writing Code:

* Test the exact endpoint you plan to use
* Validate request/response formats
* Test error conditions (invalid dates, missing auth, etc.)
* Document expected responses for your integration

*Example: Testing bid submission in Swagger first*

swagger\_bid\_example = {

"product\_id": "NGP\_OTD\_AUCK",

"price": 12.50,

"quantity": 1000,

"delivery\_date": "2025-06-05"

}

Test this payload in Swagger, then use in your Python code:

def submit\_bid\_from\_swagger\_example(api):

return api.\_make\_request("POST", "/bids", json=swagger\_bid\_example)

## UAT Environment Testing

The User Acceptance Testing (UAT) environment provides a safe sandbox for developing and validating your integration before production deployment.

### Getting UAT Access

* **Contact emsTradepoint** to request UAT credentials
* **Provide Integration Details**: Explain your intended use case and trading requirements
* **Receive Test Credentials**: Get temporary client ID and secret for UAT testing
* **Test Account Setup**: Receive demo trading account with test balances

## UAT Testing Strategy

### Phase 1: Basic Integration Testing

def test\_basic\_integration():

"""Test fundamental API operations in UAT"""

*# Test authentication*

token = get\_access\_token("uat\_client\_id", "uat\_secret", UAT\_BASE\_URL)

api = EmsTradePointAPI(UAT\_BASE\_URL, token)

*# Test basic data retrieval*

company\_info = get\_company\_info(api)

products = get\_products(api)

markets = get\_market\_status(api)

print("✓ Basic API integration successful")

return True

def test\_market\_data\_access():

"""Test market data permissions and filtering"""

*# Test gas market access*

gas\_trades = get\_trades(api, market\_category="gas")

*# Test carbon market access (if permitted)*

try:

carbon\_trades = get\_trades(api, market\_category="carbon")

print("✓ Carbon market access confirmed")

except HTTPError as e:

if e.response.status\_code == 401:

print("ℹ Carbon market access not available")

*# Test filtering capabilities*

filtered\_trades = get\_trades(api, filters={"date\_from": "2025-06-01"})

print("✓ Market data access validated")

### Phase 2: Trading Operations Testing

def test\_trading\_operations():

"""Test order management and trading functions"""

*# Test bid submission*

test\_bid = {

"product\_id": "NGP\_OTD\_AUCK",

"price": 10.00, *# Use conservative test price*

"quantity": 100, *# Small test quantity*

"delivery\_date": "2025-06-05"

}

bid\_result = submit\_bid(api, test\_bid)

bid\_id = bid\_result["id"]

*# Test bid retrieval*

bid\_details = get\_bid\_details(api, bid\_id)

*# Test bid withdrawal*

withdraw\_result = withdraw\_bid(api, bid\_id)

print("✓ Trading operations validated")

def test\_error\_handling():

"""Test error conditions and edge cases"""

*# Test invalid product ID*

try:

invalid\_bid = {"product\_id": "INVALID", "price": 10, "quantity": 100}

submit\_bid(api, invalid\_bid)

except HTTPError as e:

assert e.response.status\_code == 422

print("✓ Invalid product ID error handling correct")

*# Test expired token handling*

*# Test rate limiting behavior*

*# Test malformed request handling*

### Phase 3: Production Readiness Testing

def test\_production\_readiness():

"""Comprehensive testing to demonstrate production readiness"""

*# Test complete trading workflow*

automated\_trading\_example()

*# Test error recovery*

test\_error\_handling()

*# Test rate limiting compliance*

test\_rate\_limiting\_behavior()

*# Test token refresh mechanism*

test\_token\_management()

*# Test market hours and edge cases*

test\_market\_conditions()

print("✓ Production readiness validation complete")

## Production Access Requirements

### ⚠️ Important: Production Access Approval Process

Before ETP grants access to the production platform, developers **must demonstrate correct API usage** through comprehensive UAT testing. This requirement ensures market integrity and protects all participants.

## Demonstration Requirements

### 1. Technical Competency

* You must demonstrate:
* Proper authentication handling and token refresh
* Correct error handling for all response codes
* Appropriate rate limiting and retry logic
* Proper parameter validation before submission
* Understanding of market categories and filters

def demonstrate\_technical\_competency():

"""Example of what emsTradepoint expects to see"""

*# Robust authentication*

token\_manager = TokenManager("client\_id", "secret", UAT\_BASE\_URL)

*# Proper error handling*

@handle\_api\_errors

def safe\_trading\_operation():

api = EmsTradePointAPI(UAT\_BASE\_URL, token\_manager.get\_valid\_token())

return submit\_bid(api, validated\_bid\_data)

*# Rate limiting compliance*

result = make\_request\_with\_backoff(safe\_trading\_operation)

return result

### 2. Market Rules Understanding

* Demonstrate understanding of trading limits and exposure rules
* Show proper validation of delivery dates and product specifications
* Prove compliance with market timing and trading windows
* Evidence of proper risk management controls

### 3. Integration Quality

* **Code Quality**: Clean, maintainable, production-ready code
* **Error Handling**: Comprehensive error recovery strategies
* **Security**: Proper credential management and secure practices
* **Monitoring**: Adequate logging and monitoring capabilities

## Production Approval Process

### Step 1: UAT Testing Completion

Complete testing checklist:

test\_checklist = {

"authentication": test\_authentication\_flows(),

"market\_data": test\_market\_data\_access(),

"trading\_ops": test\_trading\_operations(),

"error\_handling": test\_error\_handling(),

"rate\_limiting": test\_rate\_limiting\_compliance(),

"production\_ready": test\_production\_readiness()

}

*# All tests must pass before production request*

assert all(test\_checklist.values()), "Complete all UAT testing first"

### Step 2: Documentation Submission Submit to emsTradepoint:

* **Integration Architecture**: How your system integrates with the API
* **Error Handling Strategy**: Your approach to handling failures and errors
* **Rate Limiting Compliance**: Evidence of proper rate limiting implementation
* **Security Measures**: Credential management and security practices
* **Testing Results**: Evidence of successful UAT testing completion

### Step 3: Code Review (if required) emsTradepoint may request:

* Code samples demonstrating proper API usage
* Evidence of Market Rules compliance in your implementation
* Demonstration of risk management controls
* Proof of proper error handling and recovery

### Step 4: Production Credentials Upon approval:

* Receive production client credentials
* Get access to production API endpoints
* Begin live trading operations

## Testing Resources

### Sample Test Scenarios

def comprehensive\_testing\_suite():

"""Complete testing suite for production readiness"""

test\_scenarios = [

*# Authentication scenarios*

("Token refresh", test\_token\_refresh),

("Expired token handling", test\_expired\_token\_recovery),

*# Market data scenarios*

("Gas market data", lambda: test\_market\_access("gas")),

("Carbon market data", lambda: test\_market\_access("carbon")),

("Historical data queries", test\_historical\_data),

*# Trading scenarios*

("Bid submission", test\_bid\_submission),

("Offer submission", test\_offer\_submission),

("Order withdrawal", test\_order\_withdrawal),

("Invalid order handling", test\_invalid\_orders),

*# Edge cases*

("Market closed handling", test\_market\_closed),

("Rate limiting", test\_rate\_limits),

("Network failures", test\_network\_resilience)

]

results = {}

for name, test\_func in test\_scenarios:

try:

results[name] = test\_func()

print(f"✓ {name}: PASSED")

except Exception as e:

results[name] = False

print(f"✗ {name}: FAILED - {e}")

return results

*# Run comprehensive testing*

test\_results = comprehensive\_testing\_suite()

production\_ready = all(test\_results.values())

print(f"\nProduction Ready: {'YES' if production\_ready else 'NO'}")

## Documentation Templates

When requesting production access, include:

*# Integration Summary Template*

integration\_summary = {

"purpose": "Automated gas trading for portfolio optimisation",

"trading\_volume": "Expected 1000-5000 GJ per day",

"market\_participation": ["NGP\_OTD", "NGP\_DA"],

"risk\_controls": "Position limits, price validation, manual overrides",

"error\_handling": "Exponential backoff, circuit breakers, alerting",

"monitoring": "Real-time dashboards, automated alerts, audit logging"

}

**Remember**: **Production access is a privilege that requires demonstrated competency and compliance.** Take UAT testing seriously and ensure your integration meets professional standards before requesting production credentials.

# Support

Need help? Contact the emsTradepoint Service Desk:

* **Email**: supportdesk@emstradepoint.co.nz
* **Phone**: (04) 590-6692

# Best Practices

## Authentication Management

import time

from datetime import datetime, timedelta

class TokenManager:

def \_\_init\_\_(self, client\_id, client\_secret, base\_url):

self.client\_id = client\_id

self.client\_secret = client\_secret

self.base\_url = base\_url

self.token = None

self.token\_expires\_at = None

def get\_valid\_token(self):

"""Get a valid access token, refreshing if necessary"""

if not self.token or datetime.now() >= self.token\_expires\_at:

self.\_refresh\_token()

return self.token

def \_refresh\_token(self):

"""Refresh the access token"""

self.token = get\_access\_token(

self.client\_id,

self.client\_secret,

self.base\_url

)

# Tokens expire in 2 hours, refresh 5 minutes early

self.token\_expires\_at = datetime.now() + timedelta(hours=1, minutes=55)

# Usage

token\_manager = TokenManager("client\_id", "client\_secret", BASE\_URL)

## Rate Limiting & Retry Logic

import time

import random

from requests.exceptions import HTTPError

def make\_request\_with\_backoff(api\_func, max\_retries=3, base\_delay=1):

"""Make API request with exponential backoff for rate limiting"""

for attempt in range(max\_retries):

try:

return api\_func()

except HTTPError as e:

if e.response.status\_code == 429: # Rate limited

if attempt < max\_retries - 1:

# Exponential backoff with jitter

delay = base\_delay \* (2 \*\* attempt) + random.uniform(0, 1)

print(f"Rate limited. Retrying in {delay:.2f} seconds...")

time.sleep(delay)

continue

raise # Re-raise if not rate limited or max retries exceeded

# Usage

def safe\_get\_trades():

return api.\_make\_request("GET", "/trades")

trades = make\_request\_with\_backoff(safe\_get\_trades)

## Error Handling

def handle\_api\_errors(func):

"""Decorator for comprehensive API error handling"""

def wrapper(\*args, \*\*kwargs):

try:

return func(\*args, \*\*kwargs)

except HTTPError as e:

status\_code = e.response.status\_code

error\_body = e.response.json() if e.response.content else {}

if status\_code == 400:

print(f"Bad Request: {error\_body.get('error', 'Invalid input')}")

elif status\_code == 401:

print("Authentication failed. Please refresh your token.")

elif status\_code == 422:

print(f"Validation Error: {error\_body.get('error', 'Business rule violation')}")

elif status\_code == 429:

print("Rate limit exceeded. Please slow down your requests.")

else:

print(f"API Error {status\_code}: {error\_body}")

raise

except Exception as e:

print(f"Unexpected error: {e}")

raise

return wrapper

@handle\_api\_errors

def submit\_bid\_safely(api, bid\_data):

return api.\_make\_request("POST", "/bids", json=bid\_data)

## Complete Trading Example

def automated\_trading\_example():

"""Example of a complete trading workflow"""

# Initialise API with token management

token\_manager = TokenManager("client\_id", "client\_secret", BASE\_URL)

def get\_api\_client():

token = token\_manager.get\_valid\_token()

return EmsTradePointAPI(BASE\_URL, token)

api = get\_api\_client()

try:

# 1. Check market status

markets = get\_market\_status(api)

if not any(market['status'] == 'open' for market in markets):

print("No markets are currently open")

return

# 2. Get available products

products = get\_products(api)

target\_product = next((p for p in products if p['name'] == 'NGP\_OTD\_AUCK'), None)

if not target\_product:

print("Target product not available")

return

# 3. Check current market activity

latest\_trades = get\_ticker(api, "trades")

if latest\_trades:

last\_price = latest\_trades[0]['price']

print(f"Last traded price: ${last\_price}")

# 4. Submit a competitive bid (slightly below last price)

bid\_price = last\_price - 0.25 if latest\_trades else 12.00

bid\_data = {

"product\_id": target\_product['id'],

"price": bid\_price,

"quantity": 1000,

"delivery\_date": "2025-06-05"

}

new\_bid = make\_request\_with\_backoff(

lambda: submit\_bid(api, bid\_data)

)

print(f"Bid submitted successfully: {new\_bid['id']}")

# 5. Monitor bid status

time.sleep(5) # Wait a moment

bid\_status = get\_bid\_details(api, new\_bid['id'])

print(f"Bid status: {bid\_status['status']}")

except Exception as e:

print(f"Trading workflow failed: {e}")

# Run the example

# automated\_trading\_example()

# Key Recommendations

* **Handle Authentication**: Store and refresh access tokens properly
* **Implement Retry Logic**: Handle rate limits and temporary failures gracefully
* **Validate Inputs**: Check data before sending to avoid 400 errors
* **Monitor Permissions**: Ensure your account has access to requested markets
* **Use Filters Wisely**: Apply appropriate filters to get only the data you need
* **Test Thoroughly**: Use the UAT environment before going live