emsTradepoint Limited

API User Guide

June 2025

EMS TRADEPOINT

EMSTRADEPOINT API GUIDE

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 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