```
from typing import Dict, List, Optional, Union, Any
from datetime import datetime
import json
import requests
from loguru import logger
from dataclasses import dataclass
from datetime import datetime, timezone
import time
import random

# Configure loguru logger
logger.add(
    "solana_transactions.log",
    rotation="500 MB",
    retention="10 days",
    level="INFO",
    format="{time} {level} {message}"
)

# Most reliable RPC endpoints
RPC_ENDPOINTS = [
    "https://api.mainnet-beta.solana.com",
    "https://rpc.ankr.com/solana",
    "https://solana.getblock.io/mainnet"
]
```

```
@dataclass
class TransactionError:
    """Data class to represent transaction errors"""
    error_type: str
    message: str
    timestamp: str = datetime.now(timezone.utc).isoformat()

class SolanaAPIException(Exception):
    """Custom exception for Solana API related errors"""
    pass

class RPCEndpointManager:
    """Manages RPC endpoints and handles switching between them"""

    def __init__(self, endpoints: List[str]):
        self.endpoints = endpoints.copy()
        self.current_endpoint = self.endpoints[0]
        self.last_request_time = 0
        self.min_request_interval = 0.2  # Increased minimum interval
        self.total_requests = 0
        self.max_requests_per_endpoint = 3

    def get_endpoint(self) -> str:
        """Get current endpoint with rate limiting"""
        now = time.time()
        time_since_last = now - self.last_request_time
```

```
        if time_since_last < self.min_request_interval:
            time.sleep(self.min_request_interval - time_since_last)

        self.total_requests += 1
        if self.total_requests >= self.max_requests_per_endpoint:
            self.switch_endpoint()
            self.total_requests = 0

        self.last_request_time = time.time()
        return self.current_endpoint

    def switch_endpoint(self) -> str:
        """Switch to next available endpoint"""
        current = self.current_endpoint
        available_endpoints = [ep for ep in self.endpoints if ep != current]

        if not available_endpoints:
            raise SolanaAPIException("All endpoints exhausted")

        self.current_endpoint = random.choice(available_endpoints)
        logger.info(f"Switched to endpoint: {self.current_endpoint}")
        return self.current_endpoint

def make_request(endpoint_manager: RPCEndpointManager, payload: dict, retry_count: int = 3) ->
dict:
    """
```

```
Makes a request with automatic endpoint switching and error handling.
"""
last_error = None

for attempt in range(retry_count):
    try:
        endpoint = endpoint_manager.get_endpoint()

        response = requests.post(
            endpoint,
            json=payload,
            timeout=10,
            headers={"Content-Type": "application/json"},
            verify=True  # Ensure SSL verification
        )

        if response.status_code != 200:
            raise SolanaAPIException(f"HTTP {response.status_code}: {response.text}")

        data = response.json()

        if "error" in data:
            error_code = data["error"].get("code")
            if error_code == 429:  # Rate limit
                logger.warning(f"Rate limit hit, switching endpoint...")
                endpoint_manager.switch_endpoint()
```

```
                time.sleep(2 ** attempt)  # Exponential backoff
                continue

            if "message" in data["error"]:
                raise SolanaAPIException(f"RPC error: {data['error']['message']}")

        return data

    except (requests.exceptions.SSLError, requests.exceptions.ConnectionError) as e:
        logger.warning(f"Connection error with {endpoint}: {str(e)}")
        endpoint_manager.switch_endpoint()
        continue

    except Exception as e:
        last_error = e
        logger.warning(f"Request failed: {str(e)}")
        endpoint_manager.switch_endpoint()
        time.sleep(1)
        continue

raise SolanaAPIException(f"All retry attempts failed. Last error: {str(last_error)}")

def fetch_wallet_transactions(wallet_address: str, max_transactions: int = 10) -> str:
    """
    Fetches recent transactions for a given Solana wallet address.
```

```
Args:
    wallet_address (str): The Solana wallet address to fetch transactions for
    max_transactions (int, optional): Maximum number of transactions to fetch. Defaults to 10.

Returns:
    str: JSON string containing transaction details
"""
try:
    if not isinstance(wallet_address, str) or len(wallet_address) != 44:
        raise ValueError(f"Invalid Solana wallet address format: {wallet_address}")

    if not isinstance(max_transactions, int) or max_transactions < 1:
        raise ValueError("max_transactions must be a positive integer")

    logger.info(f"Fetching up to {max_transactions} transactions for wallet: {wallet_address}")

    endpoint_manager = RPCEndpointManager(RPC_ENDPOINTS)

    # Get transaction signatures
    signatures_payload = {
        "jsonrpc": "2.0",
        "id": str(random.randint(1, 1000)),
        "method": "getSignaturesForAddress",
        "params": [
            wallet_address,
            {"limit": max_transactions}
```

```
    ]
}

signatures_data = make_request(endpoint_manager, signatures_payload)

transactions = signatures_data.get("result", [])
if not transactions:
    logger.info("No transactions found for this wallet")
    return json.dumps({
        "success": True,
        "transactions": [],
        "error": None,
        "transaction_count": 0
    }, indent=2)

logger.info(f"Found {len(transactions)} transactions")

# Process transactions
enriched_transactions = []
for tx in transactions:
    try:
        tx_payload = {
            "jsonrpc": "2.0",
            "id": str(random.randint(1, 1000)),
            "method": "getTransaction",
            "params": [
```

```
                tx["signature"],
                {"encoding": "json", "maxSupportedTransactionVersion": 0}
            ]
        }

        tx_data = make_request(endpoint_manager, tx_payload)

        if "result" in tx_data and tx_data["result"]:
            result = tx_data["result"]
            enriched_tx = {
                "signature": tx["signature"],
                "slot": tx["slot"],
                "timestamp": tx.get("blockTime"),
                "success": not tx.get("err"),
            }

            if "meta" in result:
                enriched_tx["fee"] = result["meta"].get("fee")
                if "preBalances" in result["meta"] and "postBalances" in result["meta"]:
                    enriched_tx["balance_change"] = sum(result["meta"]["postBalances"]) -
sum(result["meta"]["preBalances"])

            enriched_transactions.append(enriched_tx)
            logger.info(f"Processed transaction {tx['signature'][:8]}...")

    except Exception as e:
```

```
                logger.warning(f"Failed to process transaction {tx['signature']}: {str(e)}")
                continue

        logger.info(f"Successfully processed {len(enriched_transactions)} transactions")

        return json.dumps({
            "success": True,
            "transactions": enriched_transactions,
            "error": None,
            "transaction_count": len(enriched_transactions)
        }, indent=2)

    except Exception as e:
        error = TransactionError(
            error_type="API_ERROR",
            message=str(e)
        )
        logger.error(f"Error: {error.message}")
        return json.dumps({
            "success": False,
            "transactions": [],
            "error": error.__dict__,
            "transaction_count": 0
        }, indent=2)

if __name__ == "__main__":
```

```
# Example wallet address
wallet = "CtBLg4AX6LQfKVtPPUWqJyQ5cRfHydUwuZZ87rmojA1P"


try:
    result = fetch_wallet_transactions(wallet)
    print(result)
except Exception as e:
    logger.error(f"Failed to fetch transactions: {str(e)}")
```