```python
from datetime import datetime

import json

import requests

from loguru import logger

from dataclasses import dataclass

from datetime import timezone

import time

from requests.adapters import HTTPAdapter

from urllib3.util.retry import Retry


# Configure loguru logger

logger.add(

    "solana_transactions.log",

    rotation="500 MB",

    retention="10 days",

    level="INFO",

    format="{time} {level} {message}",

)


# Reliable public RPC endpoints

RPC_ENDPOINTS = [

    "https://api.mainnet-beta.solana.com",

    "https://solana.public-rpc.com",

    "https://rpc.ankr.com/solana",

]
```

```
@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


def create_http_session() -> requests.Session:
    """
    Creates a requests session with retry logic and timeouts
    """
    session = requests.Session()

    # Configure retry strategy
    retry_strategy = Retry(
        total=3,
        backoff_factor=0.5,
        status_forcelist=[429, 500, 502, 503, 504],
    )

    adapter = HTTPAdapter(max_retries=retry_strategy)
    session.mount("http://", adapter)
    session.mount("https://", adapter)

    return session


def get_working_endpoint(session: requests.Session) -> str:
    """
    Tests endpoints and returns the first working one.

    Args:
        session: requests.Session object with retry logic

    Returns:
        str: Working RPC endpoint URL

    Raises:
        SolanaAPIException: If no working endpoint is found
    """
    for endpoint in RPC_ENDPOINTS:
        try:
            payload = {
```

```
            "jsonrpc": "2.0",
            "id": 1,
            "method": "getHealth",
        }
        response = session.post(endpoint, json=payload, timeout=5)
        if response.status_code == 200:
            logger.info(f"Using RPC endpoint: {endpoint}")
            return endpoint
    except Exception as e:
        logger.warning(
            f"Endpoint {endpoint} failed health check: {str(e)}"
        )
        continue

    raise SolanaAPIException("No working RPC endpoints found")


def fetch_wallet_transactions(wallet_address: str) -> str:
    """
    Fetches all transactions for a given Solana wallet address using public RPC endpoints.

    Args:
        wallet_address (str): The Solana wallet address to fetch transactions for
            Example: "CtBLg4AX6LQfKVtPPUWqJyQ5cRfHydUwuZZ87rmojA1P"

    Returns:
    str: JSON string containing the list of transactions and their details
        Format: {
            "success": bool,
            "transactions": List[Dict],
            "error": Optional[Dict]
        }
    """
    try:
    # Validate wallet address format (basic check)
    if (
        not isinstance(wallet_address, str)
        or len(wallet_address) != 44
    ):
        raise ValueError(
            f"Invalid Solana wallet address format: {wallet_address}"
        )

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

    # Create session with retry logic
    session = create_http_session()

    # Get working endpoint
    api_endpoint = get_working_endpoint(session)
```

```
# Initialize variables for pagination
all_transactions = []
before_signature = None
limit = 25  # Smaller batch size to be more conservative

while True:
    try:
        # Prepare request payload
        payload = {
            "jsonrpc": "2.0",
            "id": "1",
            "method": "getSignaturesForAddress",
            "params": [
                wallet_address,
                {"limit": limit, "before": before_signature},
            ],
        }

        # Make API request
        response = session.post(
            api_endpoint, json=payload, timeout=10
        )

        data = response.json()
```

```
if "error" in data:
    error_code = data.get("error", {}).get("code")
    if error_code == 429:  # Rate limit
        time.sleep(1)  # Wait before trying again
        continue

    raise SolanaAPIException(
        f"API error: {data['error']}"
    )

# Extract transactions from response
transactions = data.get("result", [])

if not transactions:
    break

# Add transactions to our list
all_transactions.extend(transactions)

# Update pagination cursor
before_signature = transactions[-1]["signature"]

logger.info(
    f"Fetched {len(transactions)} transactions. Total: {len(all_transactions)}"
)
```

```
            # Break if we received fewer transactions than the limit
            if len(transactions) < limit:
                break

            # Add small delay between batches
            time.sleep(0.2)

        except Exception as e:
            logger.error(
                f"Error during transaction fetch: {str(e)}"
            )
            # Try to get a new endpoint if the current one fails
            api_endpoint = get_working_endpoint(session)
            continue

    # Enrich transaction data with additional details
    enriched_transactions = []
    for tx in all_transactions:
        try:
            tx_payload = {
                "jsonrpc": "2.0",
                "id": "1",
                "method": "getTransaction",
                "params": [
                    tx["signature"],
                    {
```

```
            "encoding": "json",
            "maxSupportedTransactionVersion": 0,
        },
    ],
}

response = session.post(
    api_endpoint, json=tx_payload, timeout=10
)
tx_data = response.json()

if "result" in tx_data and tx_data["result"]:
    enriched_transactions.append(
        {
            "signature": tx["signature"],
            "slot": tx["slot"],
            "timestamp": tx["blockTime"],
            "status": (
                "success"
                if not tx.get("err")
                else "error"
            ),
            "details": tx_data["result"],
        }
    )
```

```
            # Small delay between transaction fetches
            time.sleep(0.1)

            # print(tx)
            logger.info(f"Enriched transaction: {tx}")

        except Exception as e:
            logger.warning(
                f"Failed to fetch details for transaction {tx['signature']}: {str(e)}"
            )
            continue

    logger.info(
        f"Successfully fetched and enriched {len(enriched_transactions)} transactions"
    )

    return json.dumps(
        {
            "success": True,
            "transactions": enriched_transactions,
            "error": None,
        }
    )

except SolanaAPIException as e:
    error = TransactionError(
        error_type="API_ERROR", message=str(e)
    )
    logger.error(f"API error: {error.message}")
    return json.dumps(
        {
            "success": False,
            "transactions": [],
            "error": error.__dict__,
        }
    )

except Exception as e:
    error = TransactionError(
        error_type="UNKNOWN_ERROR",
        message=f"An unexpected error occurred: {str(e)}",
    )
    logger.error(f"Unexpected error: {error.message}")
    return json.dumps(
        {
            "success": False,
            "transactions": [],
            "error": error.__dict__,
        }
    )
```

```
# Example usage

if __name__ == "__main__":

    wallet = "CtBLg4AX6LQfKVtPPUWqJyQ5cRfHydUwuZZ87rmojA1P"



    try:

        result = fetch_wallet_transactions(wallet)

        print(json.dumps(json.loads(result), indent=2))

    except Exception as e:

        logger.error(f"Failed to fetch transactions: {str(e)}")
```