```
import concurrent.futures
import os
from datetime import datetime, timedelta
from typing import Any, Dict, List, Tuple

import requests
import yfinance as yf
from alpha_vantage.cryptocurrencies import CryptoCurrencies
from alpha_vantage.foreignexchange import ForeignExchange
from alpha_vantage.timeseries import TimeSeries
from loguru import logger



def fetch_yahoo_finance_data(tickers: List[str]) -> Dict[str, Any]:
    try:
        yf_data = yf.download(tickers, period="1d")["Close"]
        return {
            "S&P 500": yf_data["^GSPC"].iloc[-1],
            "Dow Jones": yf_data["^DJI"].iloc[-1],
            "NASDAQ": yf_data["^IXIC"].iloc[-1],
            "Gold Price": yf_data["GC=F"].iloc[-1],
            "Oil Price": yf_data["CL=F"].iloc[-1],
            "10-Year Treasury Yield": yf_data["^TNX"].iloc[-1],
        }
    except Exception as e:
        logger.error(f"Error fetching Yahoo Finance data: {str(e)}")
```

```
        return {ticker: "N/A" for ticker in tickers}



def fetch_polygon_ticker_data(
    api_key: str, ticker: str
) -> Dict[str, Any]:
    url = f"https://api.polygon.io/v2/aggs/ticker/{ticker}/prev?apiKey={api_key}"
    try:
        response = requests.get(url)
        response.raise_for_status()
        data = response.json()
        return {ticker: data["results"][0]["c"]}
    except requests.RequestException as e:
        logger.error(
            f"Error fetching Polygon data for {ticker}: {str(e)}"
        )
        return {ticker: None}



def fetch_polygon_forex_data(
    api_key: str, from_currency: str, to_currency: str
) -> Dict[str, Any]:
                                                                    url                         =
f"https://api.polygon.io/v2/aggs/ticker/C:{from_currency}{to_currency}/prev?apiKey={api_key}"
    try:
        response = requests.get(url)
```

```
        response.raise_for_status()
        data = response.json()
        return {
            f"{from_currency} to {to_currency}": data["results"][0][
                "c"
            ]
        }
    except requests.RequestException as e:
        logger.error(
            f"Error fetching Polygon forex data for {from_currency}/{to_currency}: {str(e)}"
        )
        return {f"{from_currency} to {to_currency}": None}


def fetch_polygon_economic_data(
    api_key: str, indicator: str
) -> Dict[str, Any]:
    end_date = datetime.now().strftime("%Y-%m-%d")
    start_date = (datetime.now() - timedelta(days=30)).strftime(
        "%Y-%m-%d"
    )
                                                                    url                    =
f"https://api.polygon.io/v2/aggs/ticker/{indicator}/range/1/day/{start_date}/{end_date}?apiKey={api_k
ey}"
    try:
        response = requests.get(url)
```

```
        response.raise_for_status()
        data = response.json()
        return {indicator: data["results"][-1]["c"]}
    except requests.RequestException as e:
        logger.error(
            f"Error fetching Polygon economic data for {indicator}: {str(e)}"
        )
        return {indicator: None}


def fetch_polygon_data(api_key: str) -> Dict[str, Any]:
    if not api_key:
        logger.warning(
            "Polygon API key not found. Skipping Polygon data."
        )
        return {}

    result_dict = {}

    # Define data to fetch
    stock_tickers = ["SPY", "DIA", "QQQ", "GLD", "USO", "TLT"]
    forex_pairs = [("USD", "EUR"), ("USD", "GBP"), ("USD", "JPY")]
    economic_indicators = {
        "I:CPI": "Consumer Price Index",
        "I:GDPUSD": "US GDP",
        "I:UNRATE": "US Unemployment Rate",
```

```
    "I:INDPRO": "Industrial Production Index",
    "I:HOUST": "Housing Starts",
    "I:RSXFS": "Retail Sales",
    "I:CPIUCSL": "Inflation Rate",
    "I:FEDFUNDS": "Federal Funds Rate",
    "I:GFDEBTN": "US National Debt",
    "I:REALGDP": "Real GDP",
}

# Fetch stock data
for ticker in stock_tickers:
    result_dict.update(fetch_polygon_ticker_data(api_key, ticker))

# Fetch forex data
for from_currency, to_currency in forex_pairs:
    result_dict.update(
        fetch_polygon_forex_data(
            api_key, from_currency, to_currency
        )
    )

# Fetch economic indicator data
for indicator in economic_indicators:
    result_dict.update(
        fetch_polygon_economic_data(api_key, indicator)
    )
```

```
    return result_dict



def fetch_exchange_rates() -> Dict[str, Any]:
    exchange_url = "https://open.er-api.com/v6/latest/USD"
    try:
        response = requests.get(exchange_url)
        response.raise_for_status()
        data = response.json()
        if data.get("rates"):
            return {
                "USD to EUR": data["rates"].get("EUR", "N/A"),
                "USD to GBP": data["rates"].get("GBP", "N/A"),
                "USD to JPY": data["rates"].get("JPY", "N/A"),
            }
        else:
            logger.error("Exchange rate data structure unexpected")
            return {
                "USD to EUR": "N/A",
                "USD to GBP": "N/A",
                "USD to JPY": "N/A",
            }
    except requests.RequestException as e:
        logger.error(f"Error fetching exchange rate data: {str(e)}")
        return {
```

```
            "USD to EUR": "N/A",
            "USD to GBP": "N/A",
            "USD to JPY": "N/A",
        }



def fetch_world_bank_data(
    indicator: Tuple[str, str],
) -> Dict[str, Any]:
    indicator_name, indicator_code = indicator
    wb_url = f"http://api.worldbank.org/v2/indicator/{indicator_code}?date=2021:2022&format=json"
    try:
        response = requests.get(wb_url)
        response.raise_for_status()
        data = response.json()
        if (
            isinstance(data, list)
            and len(data) > 1
            and len(data[1]) > 0
        ):
            return {indicator_name: data[1][0].get("value", "N/A")}
        else:
            logger.error(
                f"Unexpected data structure for {indicator_name}"
            )
            return {indicator_name: "N/A"}
```

```
    except requests.RequestException as e:
        logger.error(
            f"Error fetching {indicator_name} data: {str(e)}"
        )
        return {indicator_name: "N/A"}



def fetch_alpha_vantage_data(api_key: str) -> Dict[str, Any]:
    if not api_key:
        logger.warning(
            "Alpha Vantage API key not found. Skipping Alpha Vantage data."
        )
        return {}

    ts = TimeSeries(key=api_key, output_format="json")
    fx = ForeignExchange(key=api_key)
    cc = CryptoCurrencies(key=api_key)

    result = {}
    try:
        data, _ = ts.get_daily("MSFT")
        result["MSFT Daily Close"] = data["4. close"]

        data, _ = fx.get_currency_exchange_rate(
            from_currency="USD", to_currency="EUR"
        )
```

```
        result["USD to EUR (Alpha Vantage)"] = data[
            "5. Exchange Rate"
        ]

        data, _ = cc.get_digital_currency_daily(
            symbol="BTC", market="USD"
        )
        result["BTC to USD"] = data["4b. close (USD)"]
    except Exception as e:
        logger.error(f"Error fetching Alpha Vantage data: {str(e)}")

    return result


def fetch_macro_economic_data() -> Tuple[str, Dict[str, Any]]:
    """
    Fetches comprehensive macro-economic data from various APIs using multithreading.

    Returns:
        Tuple[str, Dict[str, Any]]: A tuple containing:
            - A formatted string with the macro-economic data
            - A dictionary with the raw macro-economic data
    """
    logger.info("Starting to fetch comprehensive macro-economic data")

    result_dict: Dict[str, Any] = {}
```

```
# Define data fetching tasks
tasks = [
    (
        fetch_yahoo_finance_data,
        (["^GSPC", "^DJI", "^IXIC", "GC=F", "CL=F", "^TNX"],),
    ),
    (fetch_polygon_data, (os.environ.get("POLYGON_API_KEY"),)),
    (fetch_exchange_rates, ()),
    (
        fetch_alpha_vantage_data,
        (os.environ.get("ALPHA_VANTAGE_API_KEY"),),
    ),
]

# Execute tasks concurrently
with concurrent.futures.ThreadPoolExecutor(
    max_workers=20
) as executor:
    future_to_task = {
        executor.submit(task, *args): task.__name__
        for task, args in tasks
    }
    for future in concurrent.futures.as_completed(future_to_task):
        task_name = future_to_task[future]
        try:
```

```
            data = future.result()
            result_dict.update(data)
            logger.success(
                f"Successfully fetched data from {task_name}"
            )
        except Exception as e:
            logger.error(
                f"{task_name} generated an exception: {str(e)}"
            )

# Create the formatted string output

# Update the output_string in fetch_macro_economic_data function
output_string = f"""
Macro-economic Data (as of {datetime.now().strftime('%Y-%m-%d %H:%M:%S')})
------------------------------------------------------------
Stock Market Indices:
S&P 500 (SPY): ${result_dict.get('SPY')}
Dow Jones (DIA): ${result_dict.get('DIA')}
NASDAQ (QQQ): ${result_dict.get('QQQ')}

Commodities:
Gold (GLD): ${result_dict.get('GLD')}
Oil (USO): ${result_dict.get('USO')}

Bonds:
```

20+ Year Treasury Bond (TLT): ${result_dict.get('TLT')}

Forex:

USD to EUR: {result_dict.get('USD to EUR')}

USD to GBP: {result_dict.get('USD to GBP')}

USD to JPY: {result_dict.get('USD to JPY')}

Economic Indicators:

Consumer Price Index: {result_dict.get('I:CPI')}

US GDP: ${result_dict.get('I:GDPUSD')} billion

US Unemployment Rate: {result_dict.get('I:UNRATE')}%

Industrial Production Index: {result_dict.get('I:INDPRO')}

Housing Starts: {result_dict.get('I:HOUST')} thousand

Retail Sales: ${result_dict.get('I:RSXFS')} billion

Inflation Rate: {result_dict.get('I:CPIUCSL')}%

Federal Funds Rate: {result_dict.get('I:FEDFUNDS')}%

US National Debt: ${result_dict.get('I:GFDEBTN')} billion

Real GDP: ${result_dict.get('I:REALGDP')} billion

Other Market Data:

S&P 500 (Yahoo): {result_dict.get('S&P 500', 'N/A')}

Dow Jones (Yahoo): {result_dict.get('Dow Jones', 'N/A')}

NASDAQ (Yahoo): {result_dict.get('NASDAQ', 'N/A')}

Gold Price (Yahoo): ${result_dict.get('Gold Price', 'N/A')}

Oil Price (Yahoo): ${result_dict.get('Oil Price', 'N/A')}

10-Year Treasury Yield (Yahoo): {result_dict.get('10-Year Treasury Yield', 'N/A')}%

```
    MSFT Daily Close: {result_dict.get('MSFT Daily Close', 'N/A')}
    BTC to USD: {result_dict.get('BTC to USD', 'N/A')}

    Exchange Rates (Other Sources):
    USD to EUR (Open Exchange Rates): {result_dict.get('USD to EUR', 'N/A')}
    USD to GBP (Open Exchange Rates): {result_dict.get('USD to GBP', 'N/A')}
    USD to JPY (Open Exchange Rates): {result_dict.get('USD to JPY', 'N/A')}
    USD to EUR (Alpha Vantage): {result_dict.get('USD to EUR (Alpha Vantage)', 'N/A')}
    """

    logger.info("Finished fetching comprehensive macro-economic data")
    return output_string, result_dict


# Example usage
if __name__ == "__main__":
    logger.add("macro_economic_data.log", rotation="500 MB")

    try:
        output_str, output_dict = fetch_macro_economic_data()
        print(output_str)
        print("Dictionary output:", output_dict)
    except Exception as e:
        logger.exception(f"An error occurred: {str(e)}")
```