```
import requests
import pandas as pd
from datetime import datetime
import os
from typing import Dict, Tuple, Any
import logging

# Set up logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# You'll need to set these environment variables with your actual API keys
ALPHA_VANTAGE_API_KEY = os.getenv("ALPHA_VANTAGE_API_KEY")
WORLD_BANK_API_KEY = os.getenv("WORLD_BANK_API_KEY")
FRED_API_KEY = os.getenv("FRED_API_KEY")


def fetch_real_economic_data(
    country: str, start_date: datetime, end_date: datetime
) -> Tuple[str, Dict[str, Any]]:
    data = {}

    def get_alpha_vantage_data(indicator: str) -> pd.Series:
        try:
                                                                    url =
f"https://www.alphavantage.co/query?function={indicator}&interval=monthly&apikey={ALPHA_VANT
```

```
AGE_API_KEY}"
            response = requests.get(url)
            response.raise_for_status()
            df = pd.DataFrame(
                response.json()["Monthly Time Series"]
            ).T
            df.index = pd.to_datetime(df.index)
            df = df.sort_index()
            return df["4. close"].astype(float)
        except Exception as e:
            logger.error(
                f"Error fetching Alpha Vantage data: {str(e)}"
            )
            return pd.Series()


    def get_world_bank_data(indicator: str) -> pd.Series:
        try:
                                                                    url =
f"http://api.worldbank.org/v2/country/{country}/indicator/{indicator}?format=json&date={start_date.ye
ar}:{end_date.year}&per_page=1000"
            response = requests.get(url)
            response.raise_for_status()
            data = response.json()[1]
            df = pd.DataFrame(data)
            df["date"] = pd.to_datetime(df["date"], format="%Y")
            df = df.set_index("date").sort_index()
```

```
            return df["value"].astype(float)
        except Exception as e:
            logger.error(f"Error fetching World Bank data: {str(e)}")
            return pd.Series()

    def get_fred_data(series_id: str) -> pd.Series:
        try:
            url = f"https://api.stlouisfed.org/fred/series/observations?series_id={series_id}&api_key={FRED_API_KEY}&file_type=json"
            response = requests.get(url)
            response.raise_for_status()
            df = pd.DataFrame(response.json()["observations"])
            df["date"] = pd.to_datetime(df["date"])
            df = df.set_index("date").sort_index()
            return df["value"].astype(float)
        except Exception as e:
            logger.error(f"Error fetching FRED data: {str(e)}")
            return pd.Series()

    # Fetch data from different sources
    data["GDP_growth_rate"] = get_world_bank_data("NY.GDP.MKTP.KD.ZG")
    data["unemployment_rate"] = get_world_bank_data("SL.UEM.TOTL.ZS")
    data["inflation_rate"] = get_world_bank_data("FP.CPI.TOTL.ZG")
    data["debt_to_GDP_ratio"] = get_world_bank_data(
        "GC.DOD.TOTL.GD.ZS"
```

```
)
data["current_account_balance"] = get_world_bank_data(
    "BN.CAB.XOKA.CD"
)
data["yield_curve_slope"] = get_fred_data("T10Y2Y")
data["stock_market_index"] = get_alpha_vantage_data(
    "TIME_SERIES_MONTHLY"
)
data["consumer_confidence_index"] = get_fred_data(
    "CSCICP03USM665S"
)
data["business_confidence_index"] = get_fred_data(
    "BSCICP03USM665S"
)

# Combine all data into a single DataFrame
df = pd.DataFrame(data)
df = df.loc[start_date:end_date]

if df.empty:
    logger.warning(
        "No data retrieved for the specified date range and country."
    )
    return "No data available", {
        "country": country,
        "real_time_data": {},
```

```
            "historical_data": {},
        }

    # Prepare the dictionary output
    output_dict = {
        "country": country,
        "real_time_data": df.iloc[-1].to_dict(),
        "historical_data": df.to_dict(),
    }

    # Create summary string
    summary = f"Economic Data Summary for {country} (as of {end_date.strftime('%Y-%m-%d')}):\n"
    for key, value in output_dict["real_time_data"].items():
        if pd.notna(value):
            summary += (
                f"{key.replace('_', ' ').title()}: {value:.2f}\n"
            )
        else:
            summary += f"{key.replace('_', ' ').title()}: Data not available\n"

    return summary, output_dict


# Example usage
if __name__ == "__main__":
    country = "US"  # ISO country code
```

```
start_date = datetime(2020, 1, 1)
end_date = datetime.now()

summary, data = fetch_real_economic_data(
    country, start_date, end_date
)
print(summary)
print("\nOutput Dictionary (truncated):")
print(f"Country: {data['country']}")
print("Real-time data:", data["real_time_data"])
print("Historical data: {First day, Last day}")
if data["historical_data"]:
    first_day = min(
        next(iter(data["historical_data"].values())).keys()
    )
    last_day = max(
        next(iter(data["historical_data"].values())).keys()
    )
    print(
        f"  {first_day}:",
        {
            k: v[first_day] if first_day in v else "N/A"
            for k, v in data["historical_data"].items()
        },
    )
    print(
```

```
        f"  {last_day}:",
        {
            k: v[last_day] if last_day in v else "N/A"
            for k, v in data["historical_data"].items()
        },
    )
else:
    print("  No historical data available.")
```