

# LAB ASSIGNMENT-18.2

NAME : Hansini Reddy

ROLL NO:2403A510D5

COURSE:AI ASSISTED CODING

BATCH :01(AIML)

## **Task 1 – Movie Database API**

**Task:** Connect to a Movie Database API (e.g., OMDb or TMDB) to fetch details of a movie.

**Instructions:**

Prompt AI to generate Python code to query the API by movie title.

Handle errors like invalid movie name, missing/expired API key, and timeout.

Display title, release year, genre, IMDb rating, and director.

**Expected Output:**

A Python script that retrieves and displays movie details in a clean format.

## **Task 2 – Public Transport API**

**Task:** Use a Public Transport API (e.g., city bus/train API or mock data) to fetch live arrival times.

**Instructions:**

- Fetch the next 5 arrivals for a given stop/station ID.
- Handle invalid station codes, unavailable service, and malformed responses.
- Display results in a readable table with route number, destination, and arrival time.

**Expected Output:**

- A script that provides real-time public transport information with robust error handling.

---

#### **Task 4 – Real-Time Application: Translation API**

**Scenario:** Build a translator using a free Translation API (e.g., Libre Translate, Google Translate).

**Requirements:**

Accept input text and target language from the user.

Handle invalid language codes, API quota exceeded, and empty text input.

Display original and translated text clearly.

Implement a retry mechanism if the API fails on the first attempt.

**Expected Output:**

A script that translates text to the specified language with strong error handling.

---

#### **TASK\_1**

```
import requests
```



```
api_key = "64537a40" # Replace with your actual API key
base_url = "http://www.omdbapi.com/"

# Construct the full API request URL
url = f"{base_url}?apikey={api_key}&t={movie_title}"

# Make the GET request and handle potential errors
try:
    response = requests.get(url, timeout=10) # Added timeout for robustness
    response.raise_for_status() # Raise an HTTPError for bad responses (4xx)
    print("API request successful.")

# Handle API response
try:
    movie_data = response.json()

    if movie_data.get("Response") == "True":
        title = movie_data.get("Title", "N/A")
        year = movie_data.get("Year", "N/A")
        genre = movie_data.get("Genre", "N/A")
        imdb_rating = movie_data.get("imdbRating", "N/A")
        director = movie_data.get("Director", "N/A")

        print("\nMovie Details:")
        print(f"Title: {title}")
        print(f"Release Year: {year}")
        print(f"Genre: {genre}")
```



↑ ↓ ⌛ 🗑️ :

```
    print(f"Title: {title}")
    print(f"Release Year: {year}")
    print(f"Genre: {genre}")
    print(f"IMDb Rating: {imdb_rating}")
    print(f"Director: {director}")
else:
    # Handle API-specific errors (e.g., movie not found, invalid API
    print(f"Error from API: {movie_data.get('Error', 'Unknown API er"
except requests.exceptions.JSONDecodeError:
    print("Error: Could not parse JSON response from API.")

except requests.exceptions.Timeout:
    print("Error: The request to the API timed out.")
except requests.exceptions.ConnectionError:
    print("Error: Could not connect to the API. Please check your internet c"
except requests.exceptions.HTTPError as e:
    print(f"HTTP error occurred: {e}")
except requests.exceptions.RequestException as e:
    print(f"An unexpected error occurred during the API request: {e}")

... API request successful.
```

Movie Details:  
Title: Kushi  
Release Year: 2023  
Genre: Comedy, Drama, Romance  
IMDb Rating: 5.4  
Director: Shiva Nirvana

## TASK-2

```
▶ import requests
    import json

# 1. Define Mock Data
mock_arrival_data = {
    "arrivals": [
        {
            "route": "101",
            "destination": "Downtown",
            "arrival_time": "5 minutes"
        },
        {
            "route": "205",
            "destination": "Uptown",
            "arrival_time": "12 minutes"
        },
        {
            "route": "50",
            "destination": "Airport",
            "arrival_time": "20 minutes"
        },
        {
            "route": "303",
            "destination": "Suburbia",
            "arrival_time": "25 minutes"
        },
        {
            "route": "417",
            "destination": "City Center",
            "arrival_time": "30 minutes"
        }
    ]
}
```



```
station_id = input("Please enter the station ID: ") [↑ ↓ ⌂ ⌄ ⌅ :]

# 3. Fetch Data from API or Use Mock Data
api_key = "64537a40" # Replace with your actual API key
base_url = "http://api.publictransport.io/v1/" # Replace with the actual API

# Construct the full API request URL
# This URL format is an example, replace it with the actual API endpoint and
url = f"{base_url}arrivals?station_id={station_id}&apikey={api_key}"

api_available = False
fetched_data = None

if api_key != "YOUR_API_KEY":
    try:
        response = requests.get(url, timeout=10) # Added timeout for robustness
        response.raise_for_status() # Raise an HTTPError for bad responses
        fetched_data = response.json()
        api_available = True
        print("API request successful.")

    except requests.exceptions.Timeout:
        print("Error: The request to the API timed out. Using mock data.")
    except requests.exceptions.ConnectionError:
        print("Error: Could not connect to the API. Please check your internet connection and try again later.")
    except requests.exceptions.HTTPError as e:
        print(f"HTTP error occurred: {e}. Using mock data.")
    except requests.exceptions.RequestException as e:
        print(f"An unexpected error occurred during the API request: {e}. Us")


```

```
    else:
        print("API key not provided or is placeholder. Using mock data if API request failed")
else:
    data_to_process = fetched_data

# 4. Handle API Response and Errors
if data_to_process and isinstance(data_to_process.get("arrivals"), list):
    print("Arrivals data found and is in the correct format.")

# 5. Extract and Format Arrival Times
arrivals_list = data_to_process.get("arrivals", [])
extracted_arrivals = []

for i, arrival in enumerate(arrivals_list[:5]):
    if isinstance(arrival, dict):
        route = arrival.get("route", "N/A")
        destination = arrival.get("destination", "N/A")
        arrival_time = arrival.get("arrival_time", "N/A")
        extracted_arrivals.append({
            "Route Number": route,
            "Destination": destination,
            "Arrival Time": arrival_time
        })
    else:
        print(f"Warning: Skipping invalid arrival entry at index {i}.")
```



↑ ↓ ↻ ⟲ ⟳

```
# 6. Display Results
if extracted_arrivals:
    print("\n--- Next 5 Arrivals ---")
    # Print header
    header = "| {:<15} | {:<20} | {:<15} | ".format("Route Number", "Dest
    print("-" * len(header))
    print(header)
    print("-" * len(header))

    # Print rows
    for arr in extracted_arrivals:
        row = "| {:<15} | {:<20} | {:<15} | ".format(
            arr["Route Number"],
            arr["Destination"],
            arr["Arrival Time"]
        )
        print(row)

        print("-" * len(header))
    else:
        print("No arrival data to display.")

else:
    print("Error: 'arrivals' key not found in data or data is not a list. Ca

# Print the data used (either fetched or mock)
print("\n--- Data Used (JSON) ---")
print(json.dumps(data_to_process, indent=4))
```

Please enter the station ID: 101



API key not provided or is placeholder. Using mock data.

\*\*\* Arrivals data found and is in the correct format.

--- Next 5 Arrivals ---

Route Number	Destination	Arrival Time
101	Downtown	5 minutes
205	Uptown	12 minutes
50	Airport	20 minutes
303	Suburbia	25 minutes
417	City Center	33 minutes

--- Data Used (JSON) ---

```
{  
  "arrivals": [  
    {  
      "route": "101",  
      "destination": "Downtown",  
      "arrival_time": "5 minutes"  
    },  
    {  
      "route": "205",  
      "destination": "Uptown",  
      "arrival_time": "12 minutes"  
    },  
    {  
      "route": "50",  
      "destination": "Airport",  
      "arrival time": "20 minutes"  
    }  
  ]  
}
```

### Task-3

```
▶ import yfinance as yf
import pandas as pd
import io

def get_stock_data(ticker):
    """
    Fetches the latest daily stock data for a given ticker symbol.
    Uses yfinance as a fallback if no dedicated API key is available.
    Handles invalid ticker symbols and null/empty responses.
    """

    stock_data = None
    api_key = "64537a40" # Replace with your Alpha Vantage API key

    # --- Attempt to use a dedicated API (Alpha Vantage example) ---
    if api_key != "YOUR_ALPHA_VANTAGE_API_KEY":
        base_url = "https://www.alphavantage.co/query"
        params = {
            "function": "TIME_SERIES_DAILY_ADJUSTED",
            "symbol": ticker,
            "apikey": api_key,
            "outputsize": "compact" # or "full" for more data
        }
        try:
            print(f"Attempting to fetch data for {ticker} from Alpha Vantage")
            response = requests.get(base_url, params=params, timeout=10)
            response.raise_for_status()
            data = response.json()
        except requests.RequestException as e:
            print(f"Error fetching data from Alpha Vantage: {e}")
            stock_data = None
    else:
        stock_data = yf.download(ticker)

    return stock_data
```



```
if "Time Series (Daily)" in data:  
    # Get the latest day's data  
    latest_day_key = list(data["Time Series (Daily)"].keys())[0]  
    daily_data = data["Time Series (Daily)"][latest_day_key]  
    stock_data = {  
        "Open": daily_data["1. open"],  
        "Close": daily_data["4. close"],  
        "High": daily_data["2. high"],  
        "Low": daily_data["3. low"],  
        "Volume": daily_data["6. volume"]  
    }  
    print("Successfully fetched data from Alpha Vantage.")  
elif "Error Message" in data:  
    print(f"Error from Alpha Vantage API: {data['Error Message']}")  
else:  
    print("Unexpected response format from Alpha Vantage API.")  
  
except requests.exceptions.Timeout:  
    print("Alpha Vantage API request timed out. Falling back to yfin")  
except requests.exceptions.ConnectionError:  
    print("Could not connect to Alpha Vantage API. Falling back to yfin")  
except requests.exceptions.HTTPError as e:  
    print(f"HTTP error from Alpha Vantage API: {e}. Falling back to yfin")  
except requests.exceptions.RequestException as e:  
    print(f"An unexpected error occurred with Alpha Vantage API request: {e}")  
except requests.exceptions.JSONDecodeError:  
    print("Error: Could not parse JSON response from Alpha Vantage")
```



```
● if stock_data is None:
    print(f"Fetching data for {ticker} using yfinance")
    try:
        ticker_data = yf.Ticker(ticker)
        # Get the latest day's data
        hist = ticker_data.history(period="1d")

        if not hist.empty:
            latest_day_data = hist.iloc[0]
            stock_data = {
                "Open": latest_day_data["Open"],
                "Close": latest_day_data["Close"],
                "High": latest_day_data["High"],
                "Low": latest_day_data["Low"],
                "Volume": latest_day_data["Volume"]
            }
            print("Successfully fetched data from yfinance.")
    except Exception as e:
        print(f"An error occurred while fetching data with yfinance: {e}")

    return stock_data

# --- Main part of the script ---
if __name__ == "__main__":
    ticker_symbol = input("Please enter the stock ticker symbol (e.g., AAPL)
```



```
return stock_data

# --- Main part of the script ---
if __name__ == "__main__":
    ticker_symbol = input("Please enter the stock ticker symbol (e.g., AAPL)

    daily_data = get_stock_data(ticker_symbol)

    # --- Display Results ---
    if daily_data:
        print("\n--- Latest Daily Stock Data ---")
        # Create a pandas DataFrame for formatted display
        df = pd.DataFrame([daily_data])

        # Format numerical columns for better readability
        for col in ["Open", "Close", "High", "Low"]:
            df[col] = df[col].apply(lambda x: f"{x:.2f}")
        df["Volume"] = df["Volume"].apply(lambda x: f"{x:,}")

        # Use display() for a nicely formatted table in Colab
        display(df)

    else:
        print(f"Could not retrieve stock data for {ticker_symbol}.")
```

```
# --- Sample Run and Output ---
print("\n--- Sample Run ---")
```

... Please enter the stock ticker symbol (e.g., AAPL): AAPL  
Fetching data for AAPL using yfinance...  
Successfully fetched data from yfinance.

--- Latest Daily Stock Data ---

	Open	Close	High	Low	Volume
0	268.96	269.43	273.73	267.46	41,282,100.0



## TASK\_4

```
import requests
import time

def translate_text(text, target_lang):
    """
    Translates text to a target language using the LibreTranslate API.
    Includes a retry mechanism for API calls.
    """
    api_url = "https://libretranslate.com/translate"
    # LibreTranslate might require an API key for some instances

    # Replace with your actual API key if required by the LibreTranslate instance
    api_key = "64537a40"

    payload = {
        "q": text,
        "source": "auto", # Automatically detect source language
        "target": target_lang,
        "api_key": api_key # Include API key in the payload
    }

    retries = 2
    for attempt in range(retries + 1):
        try:
            print(f"Attempt {attempt}/{retries + 1} to translate...")
            response = requests.post(api_url, json=payload, timeout=10)
            response.raise_for_status() # Raise HTTPError for bad responses
            translated_data = response.json()
```





↑ ↓ ⌂ ⌂ :

```
if "translatedText" in translated_data:
    print("Translation successful.")
    return translated_data["translatedText"]
elif "error" in translated_data:
    print(f"API error: {translated_data['error']}")  
    return f"Error: {translated_data['error']}"
else:
    print("Unexpected API response format.")
    return "Error: Unexpected API response format."  
  
except requests.exceptions.Timeout:
    print("Request timed out.")
except requests.exceptions.ConnectionError:
    print("Could not connect to the API.")
except requests.exceptions.HTTPError as e:
    print(f"HTTP error occurred: {e}")
    if response.status_code == 400: # Bad Request, often due to invalid input
        error_message = response.json().get("error", "Bad request")
        return f"Error: {error_message}"
    elif response.status_code == 429: # Too Many Requests (Quota Exceeded)
        print("API quota exceeded. Please try again later.")
        if attempt < retries:
            time.sleep(2 * (attempt + 1)) # Wait before retrying
            continue
        return "Error: API quota exceeded after multiple retries."
    elif response.status_code == 503: # Service Unavailable
        print("API service unavailable.")
        if attempt < retries:
            time.sleep(2 * (attempt + 1)) # Wait before retrying
```

◀ ▶ ⌂

```
# --- Main part of the script ---
if __name__ == "__main__":
    # Get user input
    input_text = input("Enter the text to translate: ")
    if not input_text:
        print("Error: Input text cannot be empty.")
    else:
        target_language = input("Enter the target language code (e.g.
        if not target_language:
            print("Error: Target language code cannot be empty.")
        else:
            # Perform translation
            translated_text = translate_text(input_text, target_lang

            # Display results
            print("\n--- Translation Result ---")
            print(f"Original Text: {input_text}")
            print(f"Translated Text: {translated_text}")

# --- Sample Input and Output ---
print("\n--- Sample Run ---")
sample_text = "Hello, how are you?"
sample_lang = "fr"
print(f"Sample Input Text: {sample_text}")
print(f"Sample Target Language: {sample_lang}")
sample_translated_text = translate_text(sample_text, sample_lang
print(f"Sample Translated Text: {sample_translated_text}")
print("\n--- End of Sample Run ---")
```

```
    print(f"Sample Translated Text: {sample_translated_text}\n")

# --- Sample Input and Output ---
print("\n--- Sample Run ---")
sample_text = "Hello, how are you?"
sample_lang = "fr"
print(f"Sample Input Text: {sample_text}")
print(f"Sample Target Language: {sample_lang}")
sample_translated_text = translate_text(sample_text, sample_lang)
print(f"Sample Translated Text: {sample_translated_text}")
print("\n--- End of Sample Run ---")

...
o translate: Hello,How are you?
language code (e.g., es for Spanish, fr for French, de for German): es
translate...
red: 403 Client Error: Forbidden for url: https://libretranslate.com/translate
translate...
red: 403 Client Error: Forbidden for url: https://libretranslate.com/translate
translate...
red: 403 Client Error: Forbidden for url: https://libretranslate.com/translate

Result ---
Hello,How are you?
```