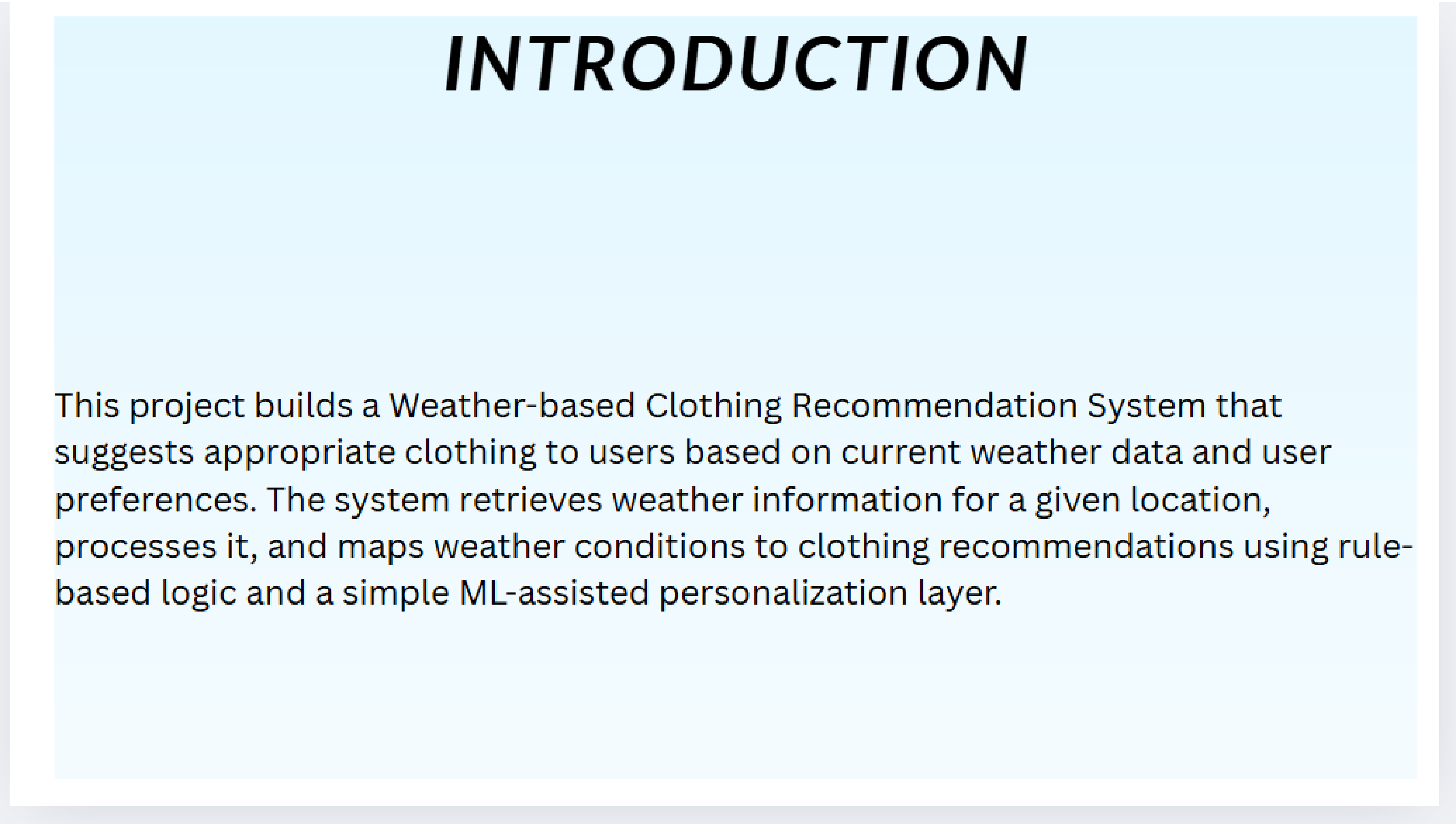
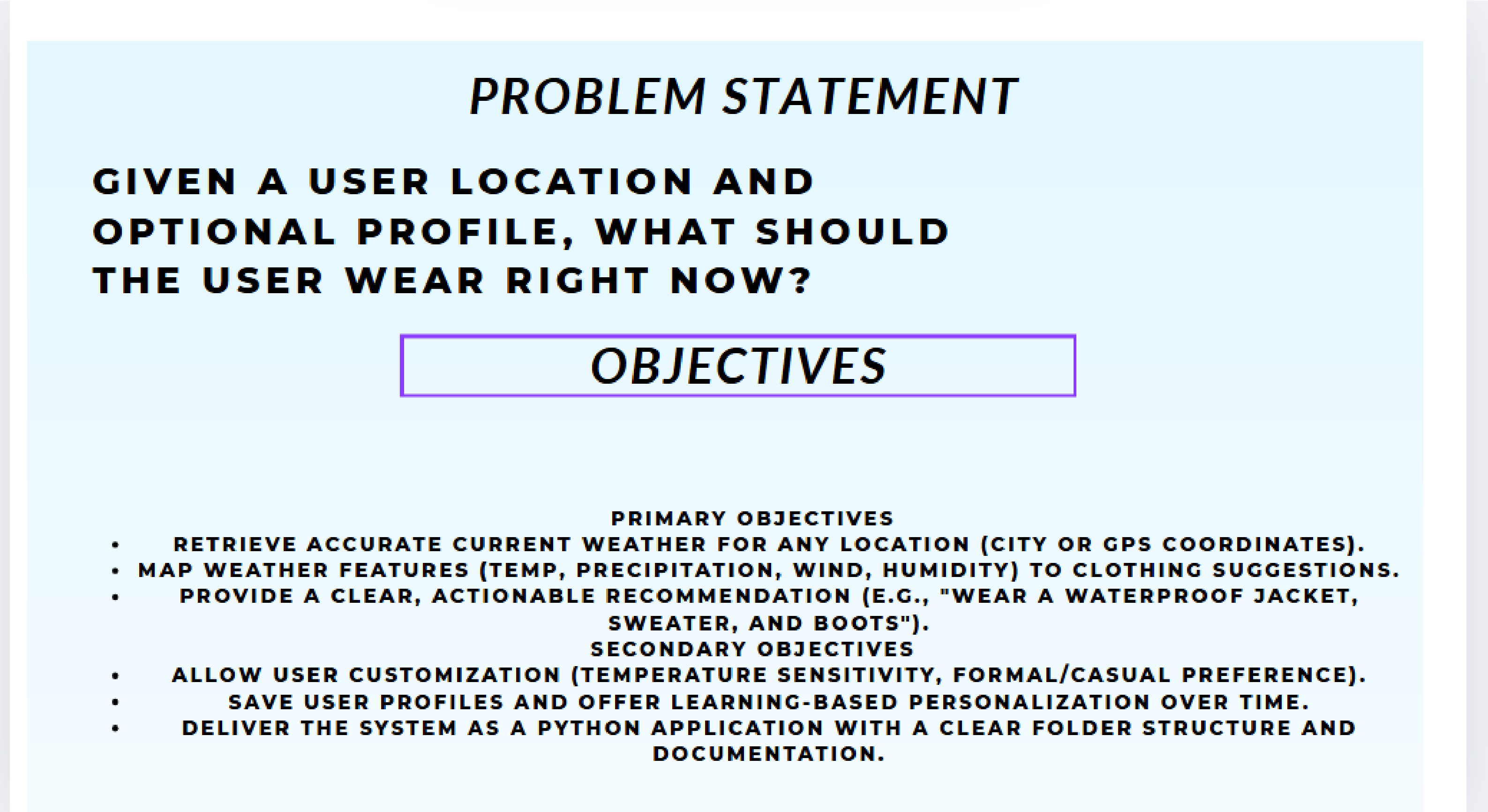
WEATHER AND CLOTH RECOMMENDATION SYSTEM

NAME	KANISHKA
REG. NO.	25BAI10070



INTRODUCTION

This project builds a Weather-based Clothing Recommendation System that suggests appropriate clothing to users based on current weather data and user preferences. The system retrieves weather information for a given location, processes it, and maps weather conditions to clothing recommendations using rule-based logic and a simple ML-assisted personalization layer.



PROBLEM STATEMENT

GIVEN A USER LOCATION AND OPTIONAL PROFILE, WHAT SHOULD THE USER WEAR RIGHT NOW?

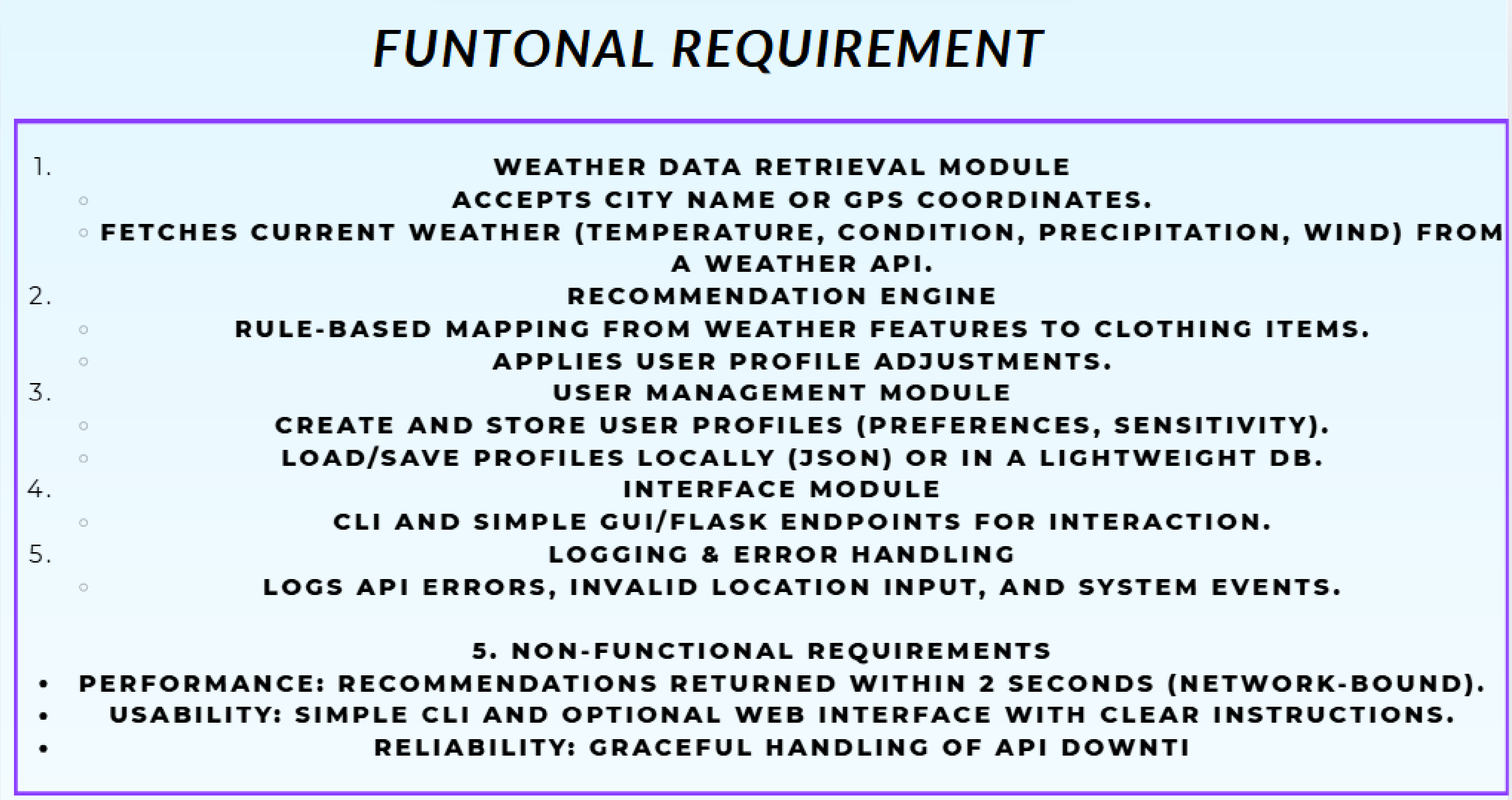
OBJECTIVES

PRIMARY OBJECTIVES

- **RETRIEVE ACCURATE CURRENT WEATHER FOR ANY LOCATION (CITY OR GPS COORDINATES).**
- **MAP WEATHER FEATURES (TEMP, PRECIPITATION, WIND, HUMIDITY) TO CLOTHING SUGGESTIONS.**
- **PROVIDE A CLEAR, ACTIONABLE RECOMMENDATION (E.G., "WEAR A WATERPROOF JACKET, SWEATER, AND BOOTS").**

SECONDARY OBJECTIVES

- **ALLOW USER CUSTOMIZATION (TEMPERATURE SENSITIVITY, FORMAL/CASUAL PREFERENCE).**
- **SAVE USER PROFILES AND OFFER LEARNING-BASED PERSONALIZATION OVER TIME.**
- **DELIVER THE SYSTEM AS A PYTHON APPLICATION WITH A CLEAR FOLDER STRUCTURE AND DOCUMENTATION.**

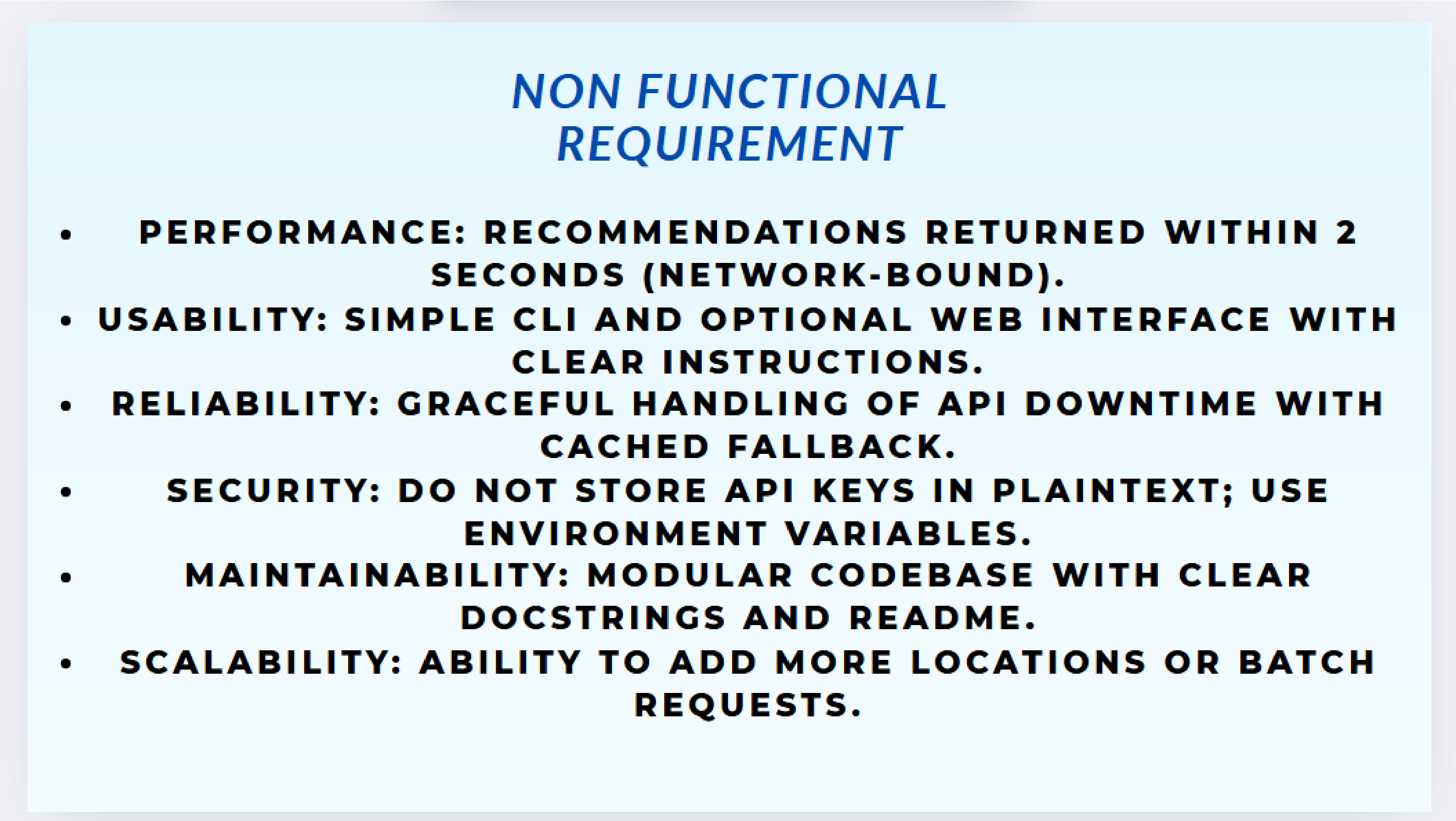


FUNTONAL REQUIREMENT

1. **WEATHER DATA RETRIEVAL MODULE**
    - **ACCEPTS CITY NAME OR GPS COORDINATES.**
    - **FETCHES CURRENT WEATHER (TEMPERATURE, CONDITION, PRECIPITATION, WIND) FROM A WEATHER API.**
2. **RECOMMENDATION ENGINE**
    - **RULE-BASED MAPPING FROM WEATHER FEATURES TO CLOTHING ITEMS.**
    - **APPLIES USER PROFILE ADJUSTMENTS.**
3. **USER MANAGEMENT MODULE**
    - **CREATE AND STORE USER PROFILES (PREFERENCES, SENSITIVITY).**
    - **LOAD/SAVE PROFILES LOCALLY (JSON) OR IN A LIGHTWEIGHT DB.**
4. **INTERFACE MODULE**
    - **CLI AND SIMPLE GUI/FLASK ENDPOINTS FOR INTERACTION.**
5. **LOGGING & ERROR HANDLING**
    - **LOGS API ERRORS, INVALID LOCATION INPUT, AND SYSTEM EVENTS.**

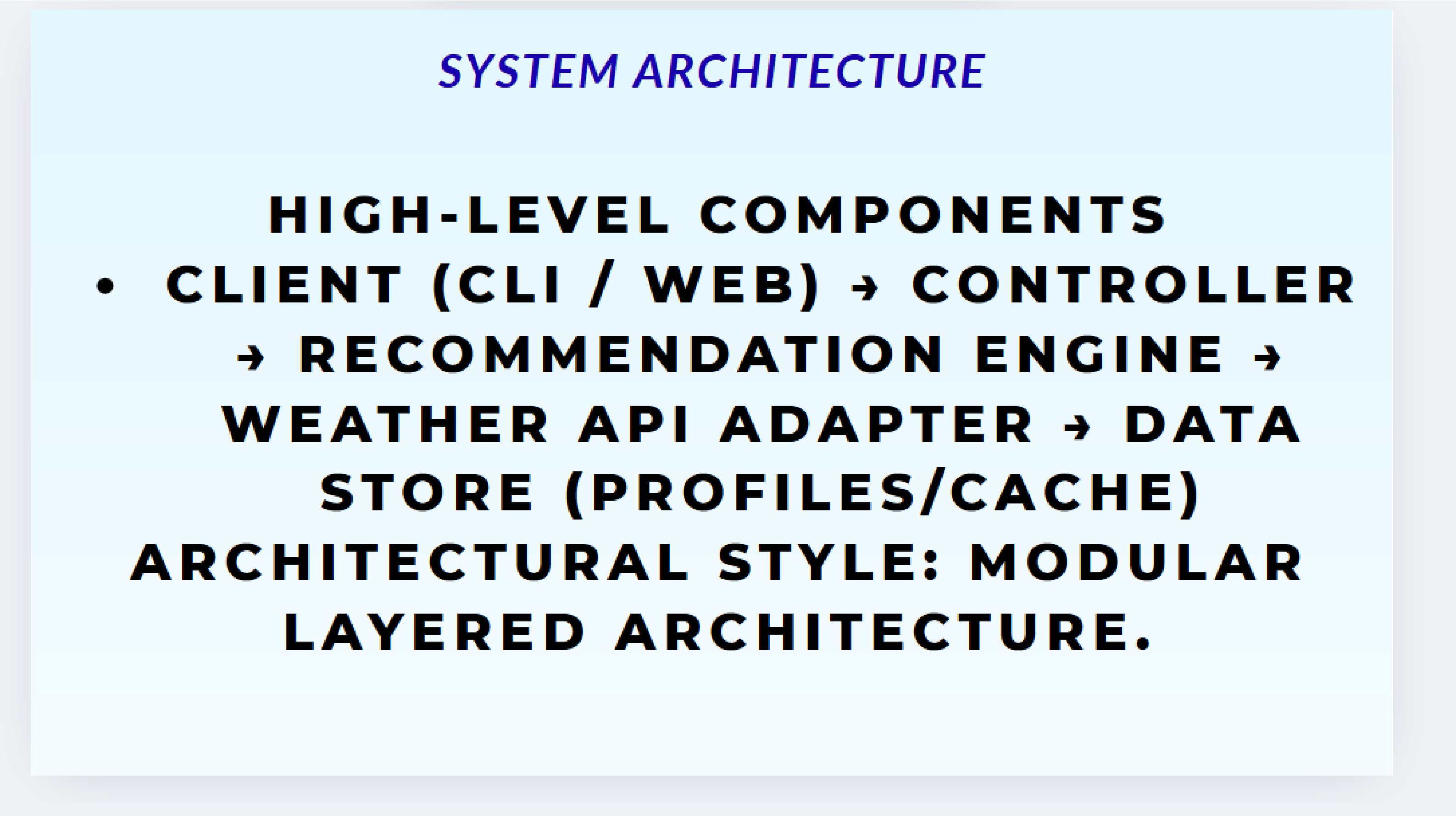
5. NON-FUNCTIONAL REQUIREMENTS

- **PERFORMANCE: RECOMMENDATIONS RETURNED WITHIN 2 SECONDS (NETWORK-BOUND).**
- **USABILITY: SIMPLE CLI AND OPTIONAL WEB INTERFACE WITH CLEAR INSTRUCTIONS.**
- **RELIABILITY: GRACEFUL HANDLING OF API DOWNTI**



NON FUNCTIONAL REQUIREMENT

- **PERFORMANCE: RECOMMENDATIONS RETURNED WITHIN 2 SECONDS (NETWORK-BOUND).**
- **USABILITY: SIMPLE CLI AND OPTIONAL WEB INTERFACE WITH CLEAR INSTRUCTIONS.**
- **RELIABILITY: GRACEFUL HANDLING OF API DOWNTIME WITH CACHED FALLBACK.**
- **SECURITY: DO NOT STORE API KEYS IN PLAINTEXT; USE ENVIRONMENT VARIABLES.**
- **MAINTAINABILITY: MODULAR CODEBASE WITH CLEAR DOCSTRINGS AND README.**
- **SCALABILITY: ABILITY TO ADD MORE LOCATIONS OR BATCH REQUESTS.**

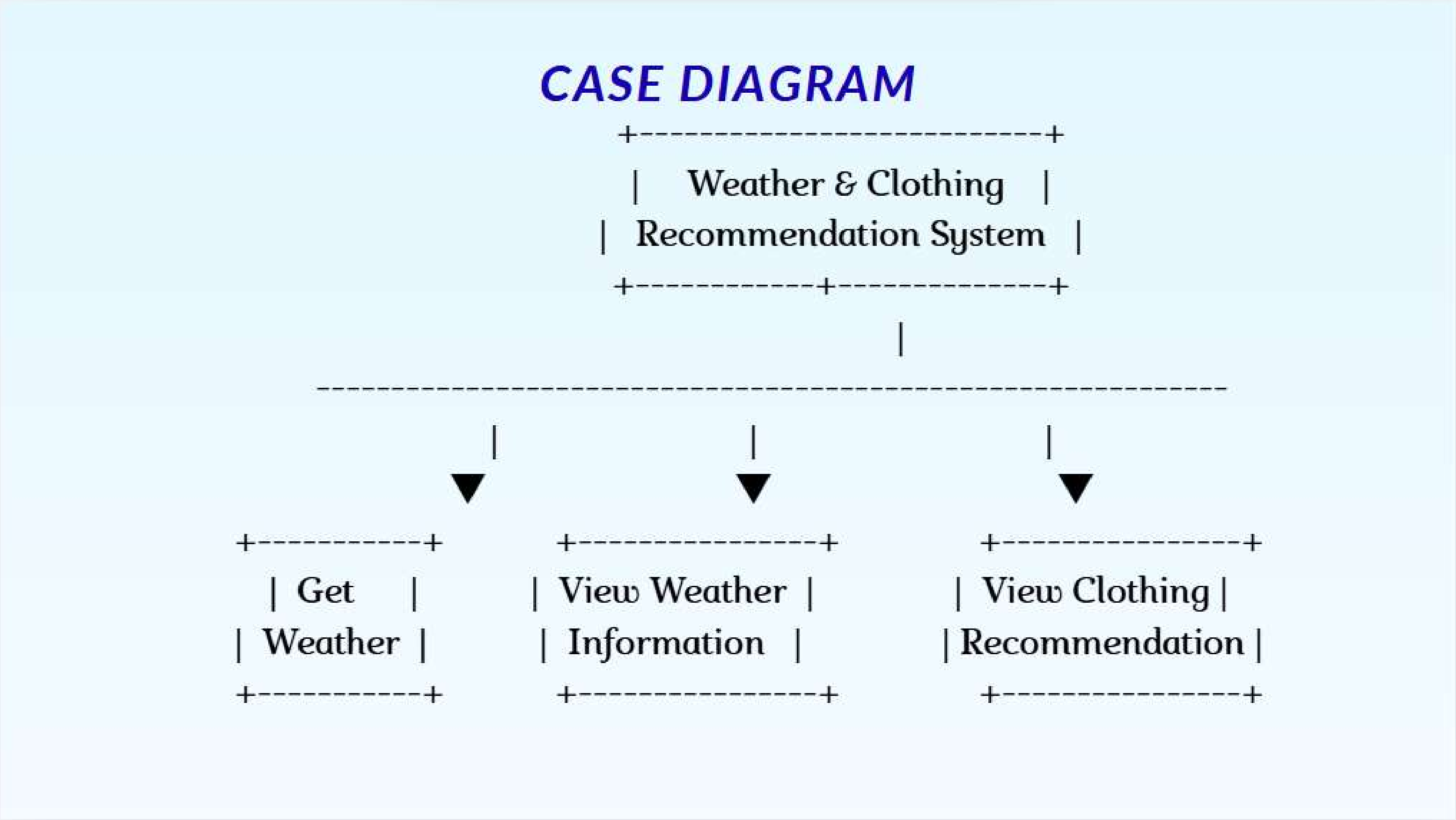


SYSTEM ARCHITECTURE

HIGH-LEVEL COMPONENTS

- **CLIENT (CLI / WEB) → CONTROLLER → RECOMMENDATION ENGINE → WEATHER API ADAPTER → DATA STORE (PROFILES/CACHE)**

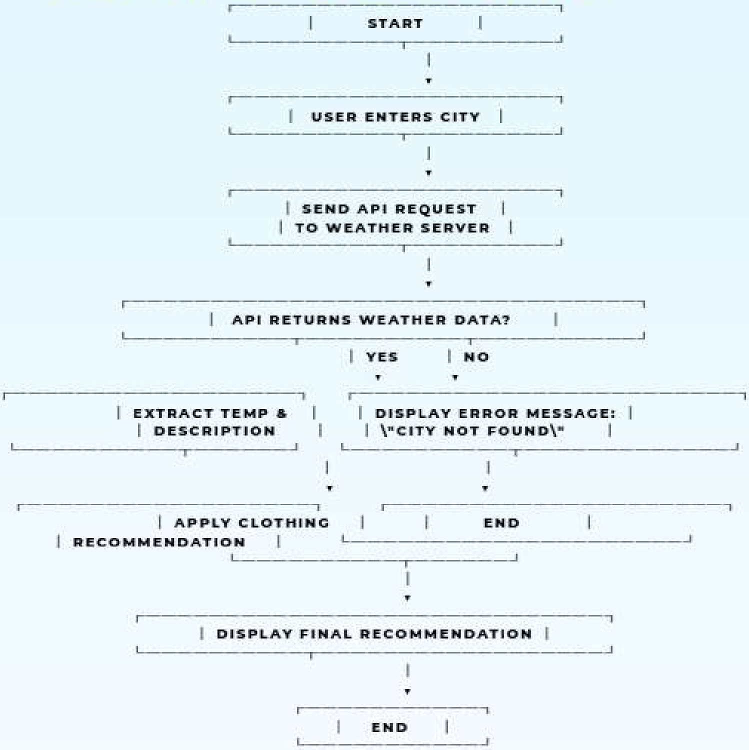
ARCHITECTURAL STYLE: MODULAR LAYERED ARCHITECTURE.



CASE DIAGRAM

```
            +---------------------------+
            |    Weather & Clothing     |
            |  Recommendation System    |
            +------------+--------------+
                         |
-----------------------------------------------------------
        |                   |                   |
        ▼                   ▼                   ▼
+-----------+      +----------------+      +----------------+
|  Get      |      |  View Weather  |      |  View Clothing |
|  Weather  |      |  Information   |      | Recommendation |
+-----------+      +----------------+      +----------------+
```

WORKFLOW DIAGRAM



START

USER ENTERS CITY

SEND API REQUEST TO WEATHER SERVER

API RETURNS WEATHER DATA?

YES

NO

EXTRACT TEMP & DESCRIPTION

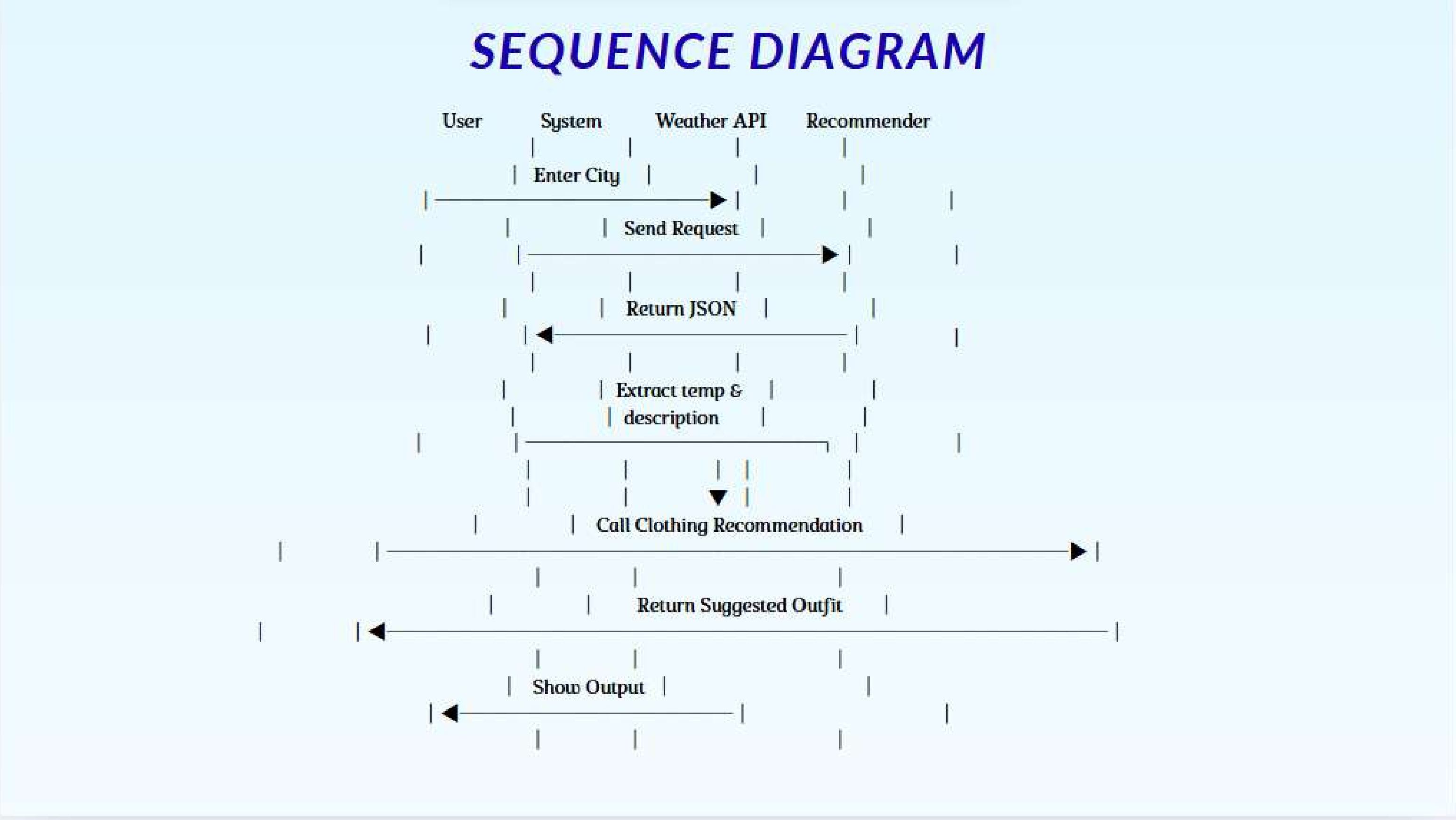
DISPLAY ERROR MESSAGE: \"CITY NOT FOUND\"

APPLY CLOTHING RECOMMENDATION

END

DISPLAY FINAL RECOMMENDATION

END



SEQUENCE DIAGRAM

User System Weather API Recommender

Enter City

Send Request

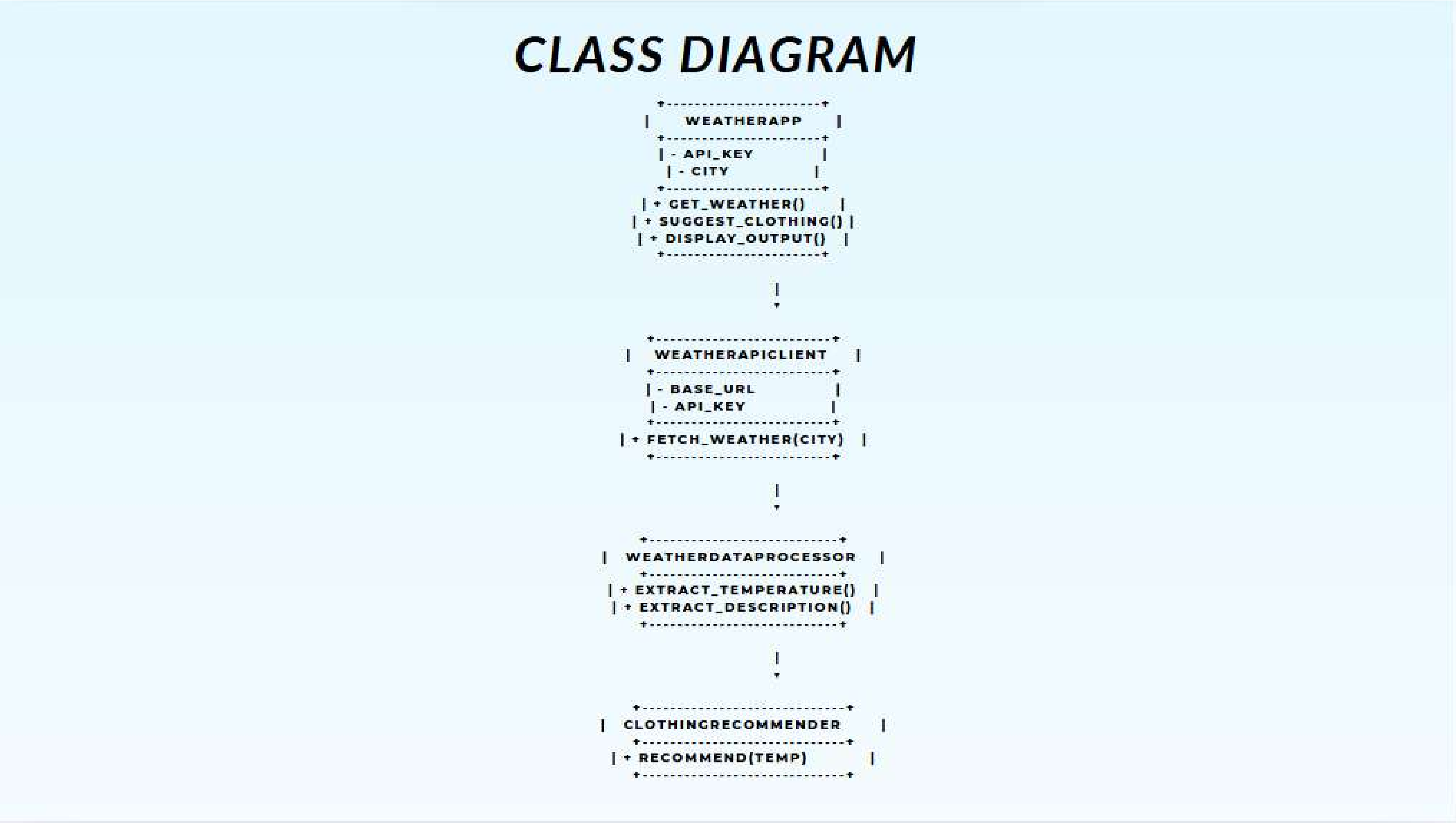
Return JSON

Extract temp & description

Call Clothing Recommendation

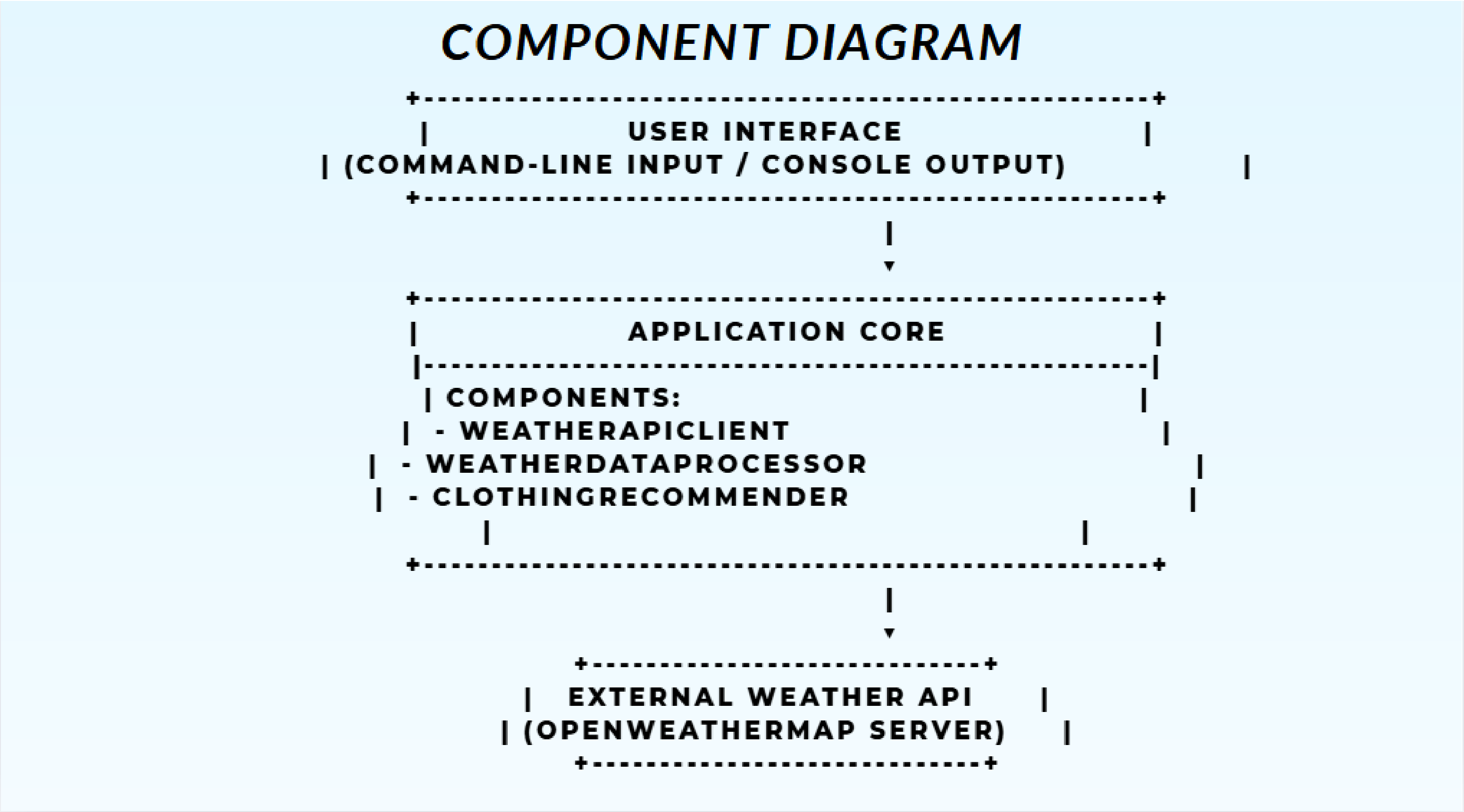
Return Suggested Outfit

Show Output



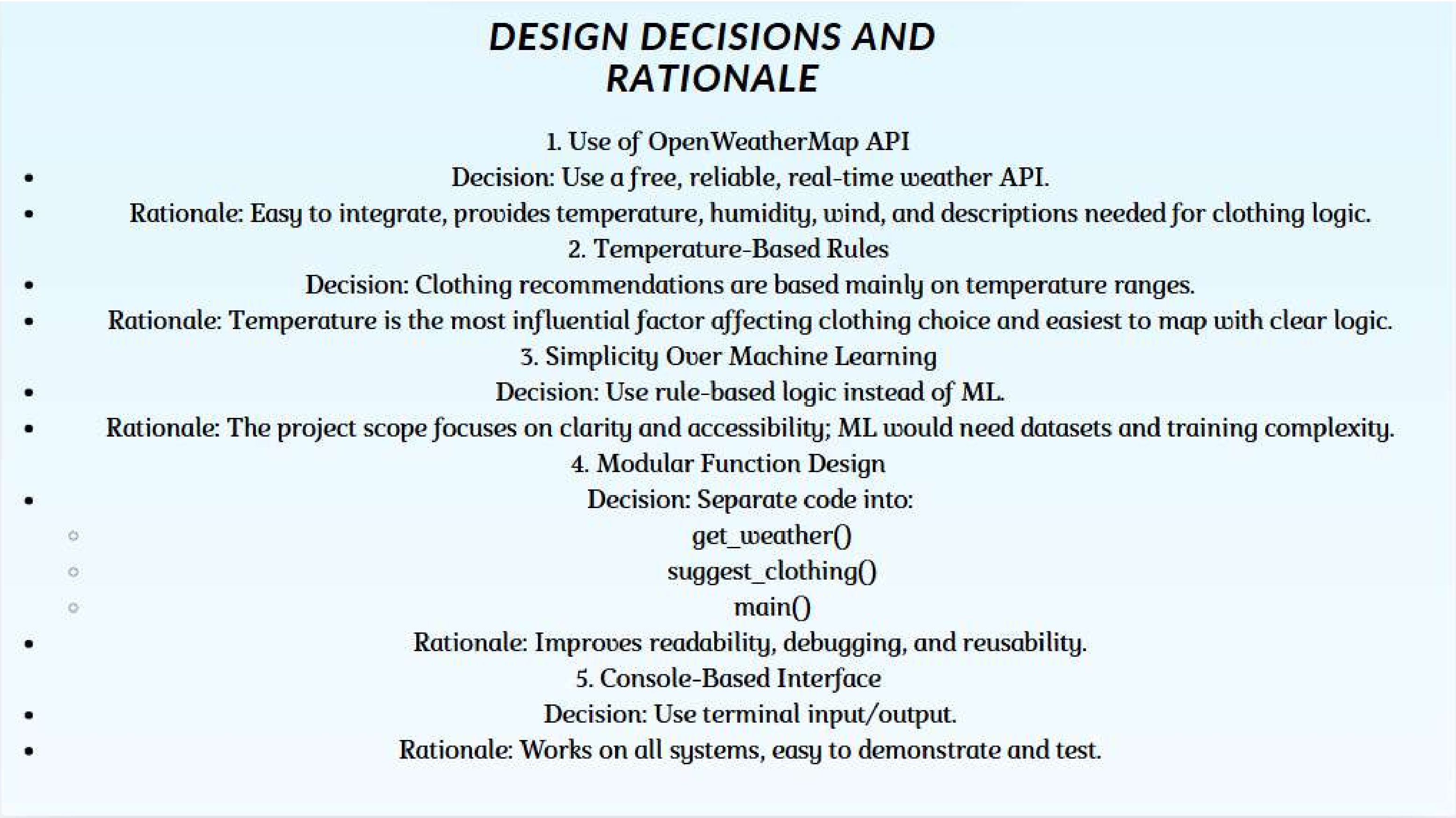
CLASS DIAGRAM

```
     +---------------------+
  |     WEATHERAPP     |
     +---------------------+
     | - API_KEY          |
      | - CITY            |
     +---------------------+
  | + GET_WEATHER()      |
 | + SUGGEST_CLOTHING() |
  | + DISPLAY_OUTPUT()   |
     +---------------------+

               |
               ▼

    +-----------------------+
 |    WEATHERAPICLIENT    |
    +-----------------------+
    | - BASE_URL            |
     | - API_KEY           |
    +-----------------------+
 | + FETCH_WEATHER(CITY)   |
    +-----------------------+

               |
               ▼

   +-------------------------+
|   WEATHERDATAPROCESSOR    |
   +-------------------------+
 | + EXTRACT_TEMPERATURE()   |
  | + EXTRACT_DESCRIPTION()  |
   +-------------------------+

               |
               ▼

  +---------------------------+
|   CLOTHINGRECOMMENDER       |
  +---------------------------+
 | + RECOMMEND(TEMP)          |
  +---------------------------+
```



COMPONENT DIAGRAM

```
      +-----------------------------------------------------+
       |                USER INTERFACE                      |
| (COMMAND-LINE INPUT / CONSOLE OUTPUT)              |
      +-----------------------------------------------------+
                                  |
                                  ▼
      +-----------------------------------------------------+
      |                APPLICATION CORE                     |
      |-----------------------------------------------------|
       | COMPONENTS:                                        |
      |  - WEATHERAPICLIENT                                  |
    |  - WEATHERDATAPROCESSOR                                 |
    |  - CLOTHINGRECOMMENDER                                 |
              |                                         |
      +-----------------------------------------------------+
                                  |
                                  ▼
                +-----------------------------+
            |   EXTERNAL WEATHER API      |
           | (OPENWEATHERMAP SERVER)     |
                +-----------------------------+
```



DESIGN DECISIONS AND RATIONALE

1. Use of OpenWeatherMap API

- Decision: Use a free, reliable, real-time weather API.
- Rationale: Easy to integrate, provides temperature, humidity, wind, and descriptions needed for clothing logic.

2. Temperature-Based Rules

- Decision: Clothing recommendations are based mainly on temperature ranges.
- Rationale: Temperature is the most influential factor affecting clothing choice and easiest to map with clear logic.

3. Simplicity Over Machine Learning

- Decision: Use rule-based logic instead of ML.
- Rationale: The project scope focuses on clarity and accessibility; ML would need datasets and training complexity.

4. Modular Function Design

- Decision: Separate code into:
  - get_weather()
  - suggest_clothing()
  - main()
- Rationale: Improves readability, debugging, and reusability.

5. Console-Based Interface

- Decision: Use terminal input/output.
- Rationale: Works on all systems, easy to demonstrate and test.

IMPLEMENTATION DETAILS

PROGRAMMING LANGUAGE

- **PYTHON 3**

LIBRARIES USED

- **REQUESTS — TO CALL API**
- **JSON — FOR READING WEATHER RESPONSES**

CORE FUNCTIONS

✓ GET_WEATHER(CITY_NAME, API_KEY)

- **SENDS API REQUEST**
- **RETRIEVES JSON DATA**
- **EXTRACTS TEMPERATURE AND WEATHER DESCRIPTION**

✓ SUGGEST_CLOTHING(TEMP)

- **USES TEMPERATURE RANGES TO DECIDE CLOTHING**
- **EXAMPLE:**
  - **>= 30°C: LIGHT T-SHIRT, SUNGLASSES**
  - **< 10°C: COAT, SCARF, GLOVES**

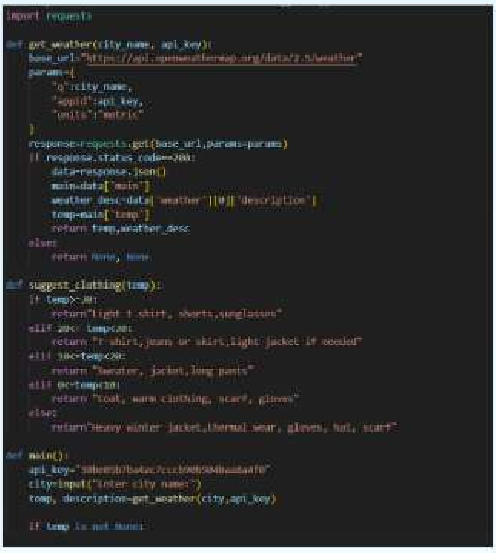
✓ MAIN()

- **ACCEPTS USER INPUT**
- **CALLS WEATHER & RECOMMENDATION FUNCTIONS**
- **DISPLAYS OUTPUT**

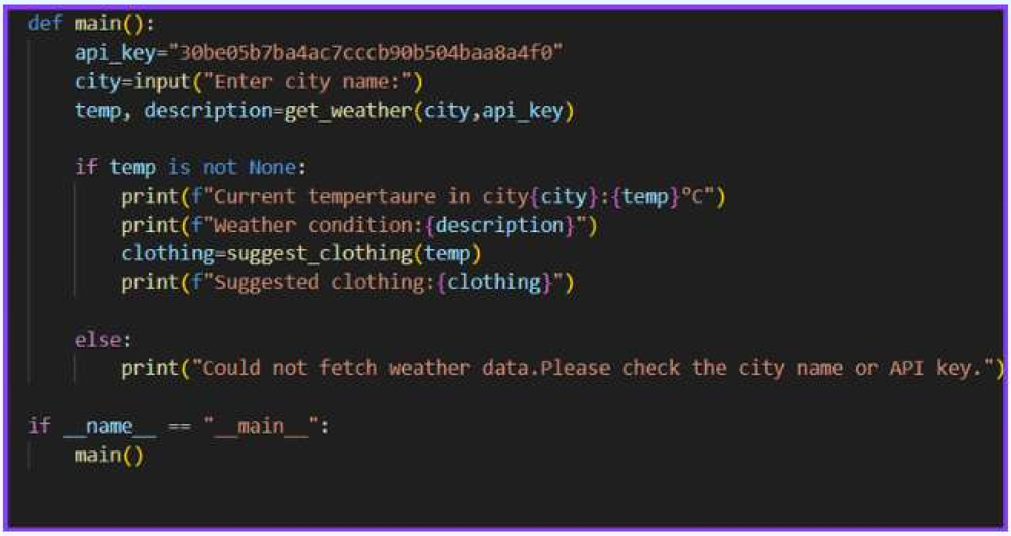
API DETAILS

- **URL: HTTPS://API.OPENWEATHERMAP.ORG/DATA/2.5/WEATHER**
- **PARAMETERS:**
  - **Q → CITY NAME**
  - **APPID → API KEY**
  - **UNITS=METRIC → CELSIUS**

CODE SCREENSHOTS

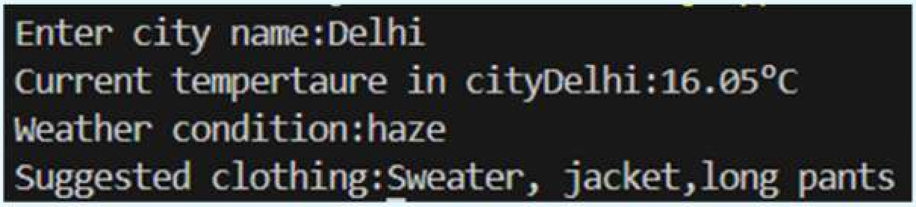


```python
import requests

def get_weather(city_name, api_key):
    base_url="https://api.openweathermap.org/data/2.5/weather"
    params={
        "q":city_name,
        "appid":api_key,
        "units":"metric"
    }
    response=requests.get(base_url,params=params)
    if response.status_code==200:
        data=response.json()
        main=data['main']
        weather_desc=data['weather'][0]['description']
        temp=main['temp']
        return temp,weather_desc
    else:
        return None, None

def suggest_clothing(temp):
    if temp>=30:
        return"Light t-shirt, shorts,sunglasses"
    elif 20<= temp<30:
        return "T-shirt,jeans or skirt,light jacket if needed"
    elif 10<=temp<20:
        return "Sweater, jacket,long pants"
    elif 0<=temp<10:
        return "Coat, warm clothing, scarf, gloves"
    else:
        return"Heavy winter jacket,thermal wear, gloves, hat, scarf"

def main():
    api_key="30be05b7ba4ac7cccb90b504baa8a4f0"
    city=input("Enter city name:")
    temp, description=get_weather(city,api_key)

    if temp is not None:
```

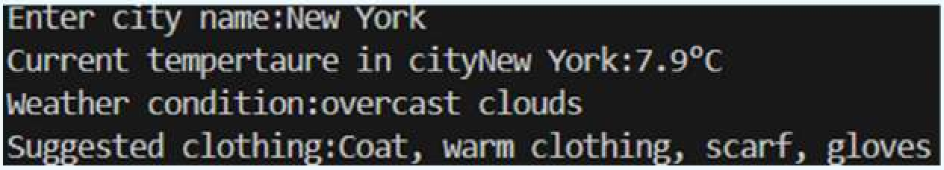


```python
def main():
    api_key="30be05b7ba4ac7cccb90b504baa8a4f0"
    city=input("Enter city name:")
    temp, description=get_weather(city,api_key)

    if temp is not None:
        print(f"Current tempertaure in city{city}:{temp}°C")
        print(f"Weather condition:{description}")
        clothing=suggest_clothing(temp)
        print(f"Suggested clothing:{clothing}")

    else:
        print("Could not fetch weather data.Please check the city name or API key.")

if __name__ == "__main__":
    main()
```

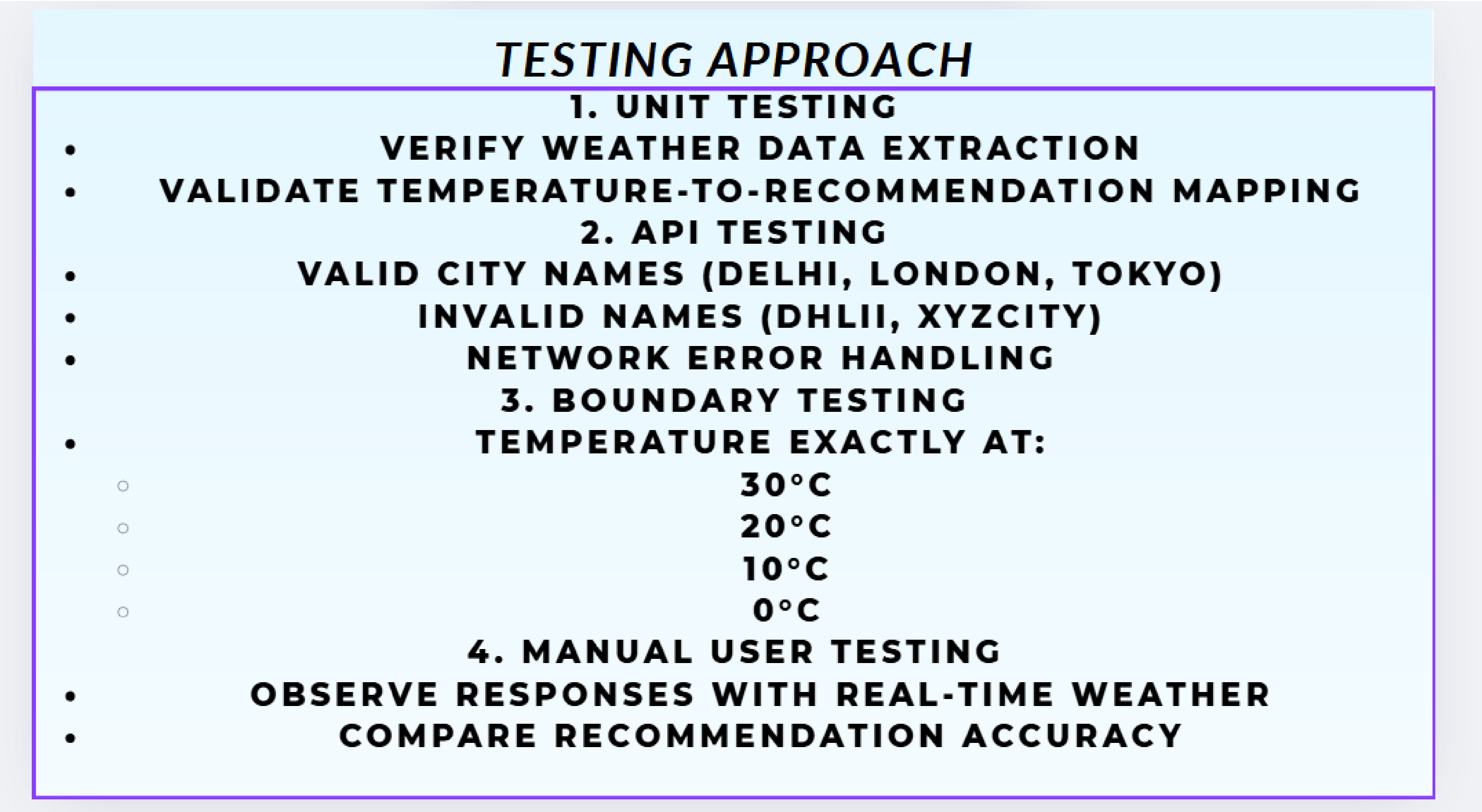
SCREENSHOTS



```
Enter city name:Delhi
Current tempertaure in cityDelhi:16.05°C
Weather condition:haze
Suggested clothing:Sweater, jacket,long pants
```



```
Enter city name:New York
Current tempertaure in cityNew York:7.9°C
Weather condition:overcast clouds
Suggested clothing:Coat, warm clothing, scarf, gloves
```



TESTING APPROACH

1. UNIT TESTING

- VERIFY WEATHER DATA EXTRACTION
- VALIDATE TEMPERATURE-TO-RECOMMENDATION MAPPING

2. API TESTING

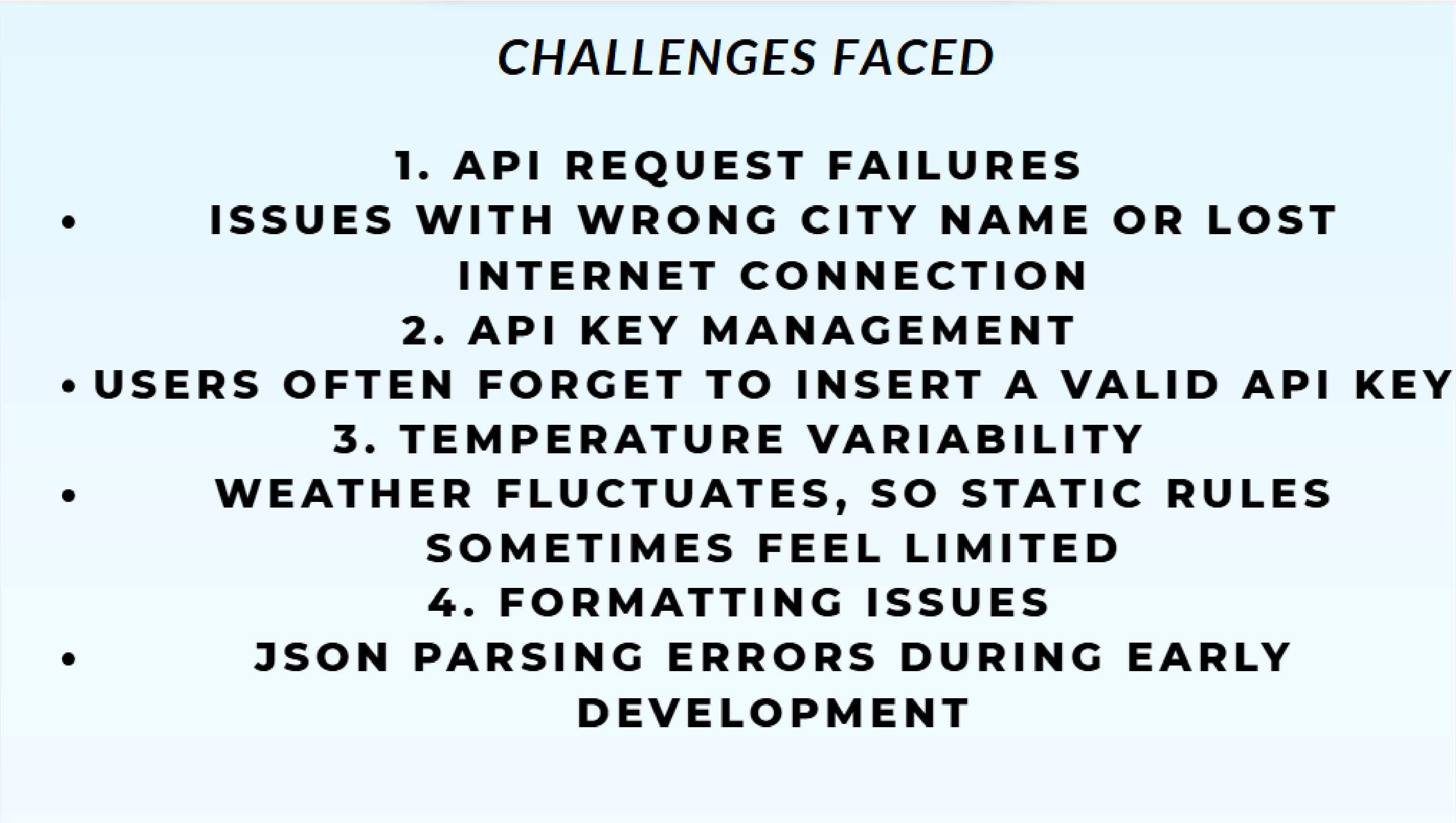
- VALID CITY NAMES (DELHI, LONDON, TOKYO)
- INVALID NAMES (DHLII, XYZCITY)
- NETWORK ERROR HANDLING

3. BOUNDARY TESTING

- TEMPERATURE EXACTLY AT:
  - 30°C
  - 20°C
  - 10°C
  - 0°C

4. MANUAL USER TESTING

- OBSERVE RESPONSES WITH REAL-TIME WEATHER
- COMPARE RECOMMENDATION ACCURACY



CHALLENGES FACED

1. API REQUEST FAILURES

- **ISSUES WITH WRONG CITY NAME OR LOST INTERNET CONNECTION**

2. API KEY MANAGEMENT

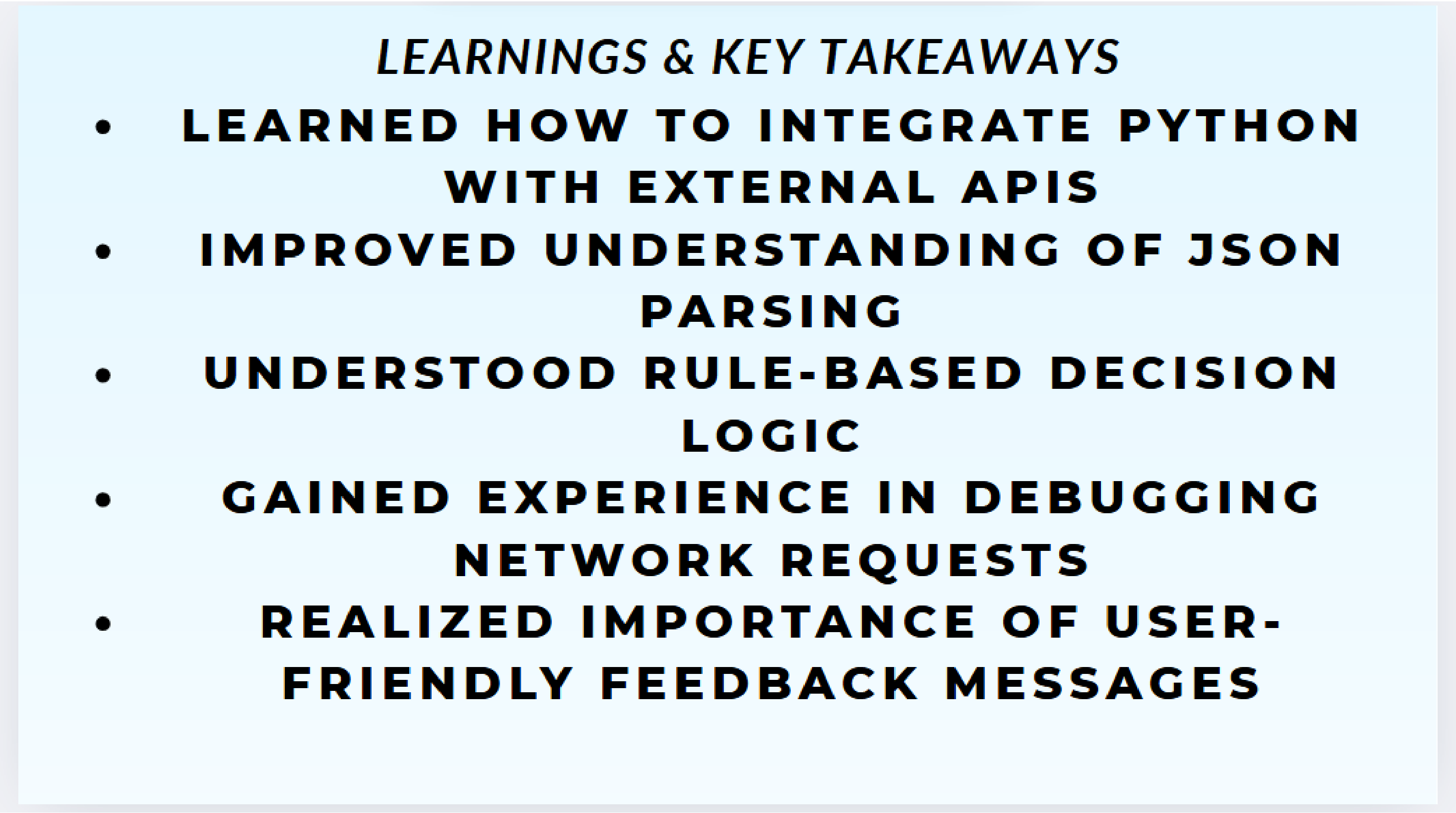
- **USERS OFTEN FORGET TO INSERT A VALID API KEY**

3. TEMPERATURE VARIABILITY

- **WEATHER FLUCTUATES, SO STATIC RULES SOMETIMES FEEL LIMITED**

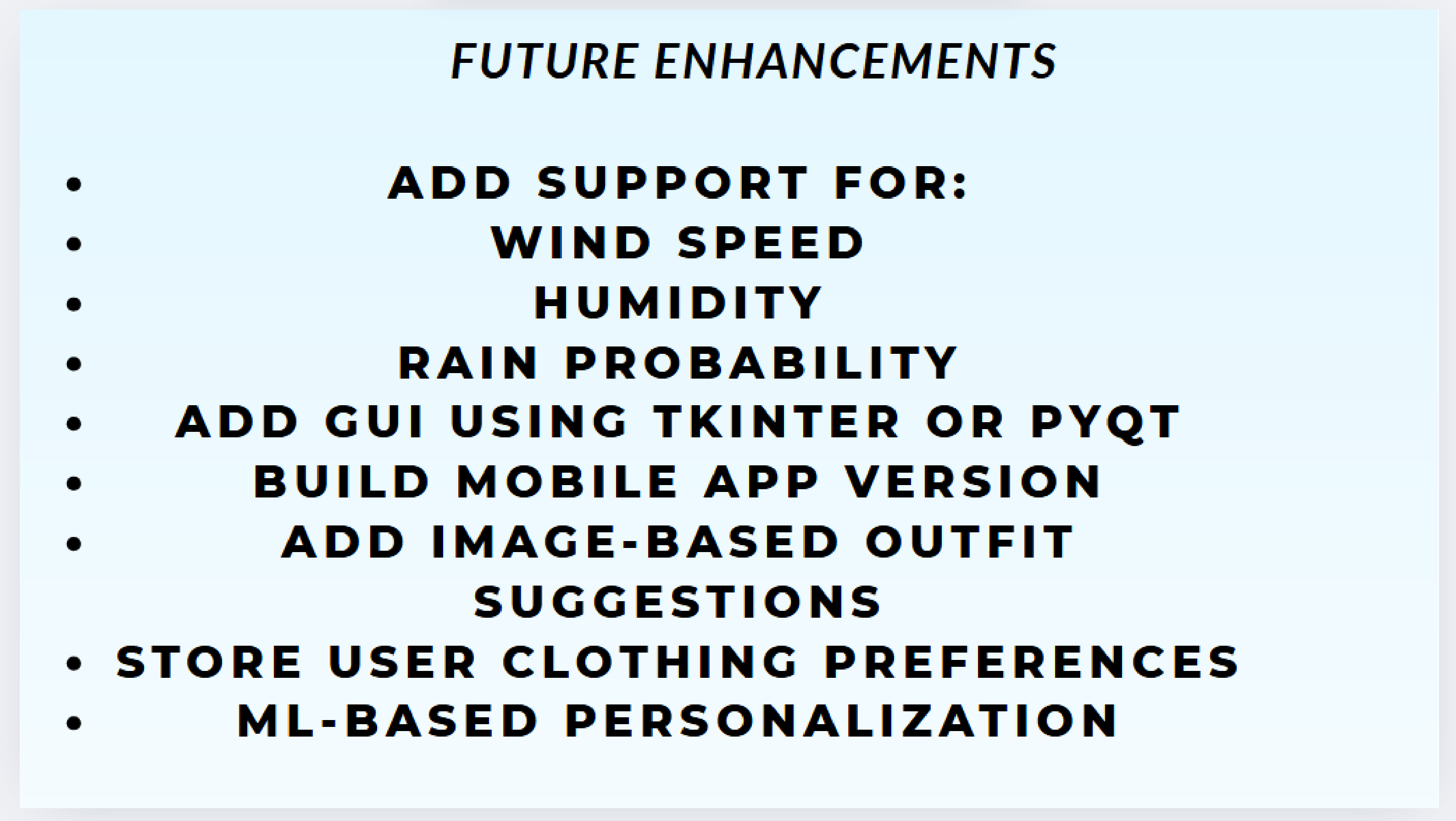
4. FORMATTING ISSUES

- **JSON PARSING ERRORS DURING EARLY DEVELOPMENT**



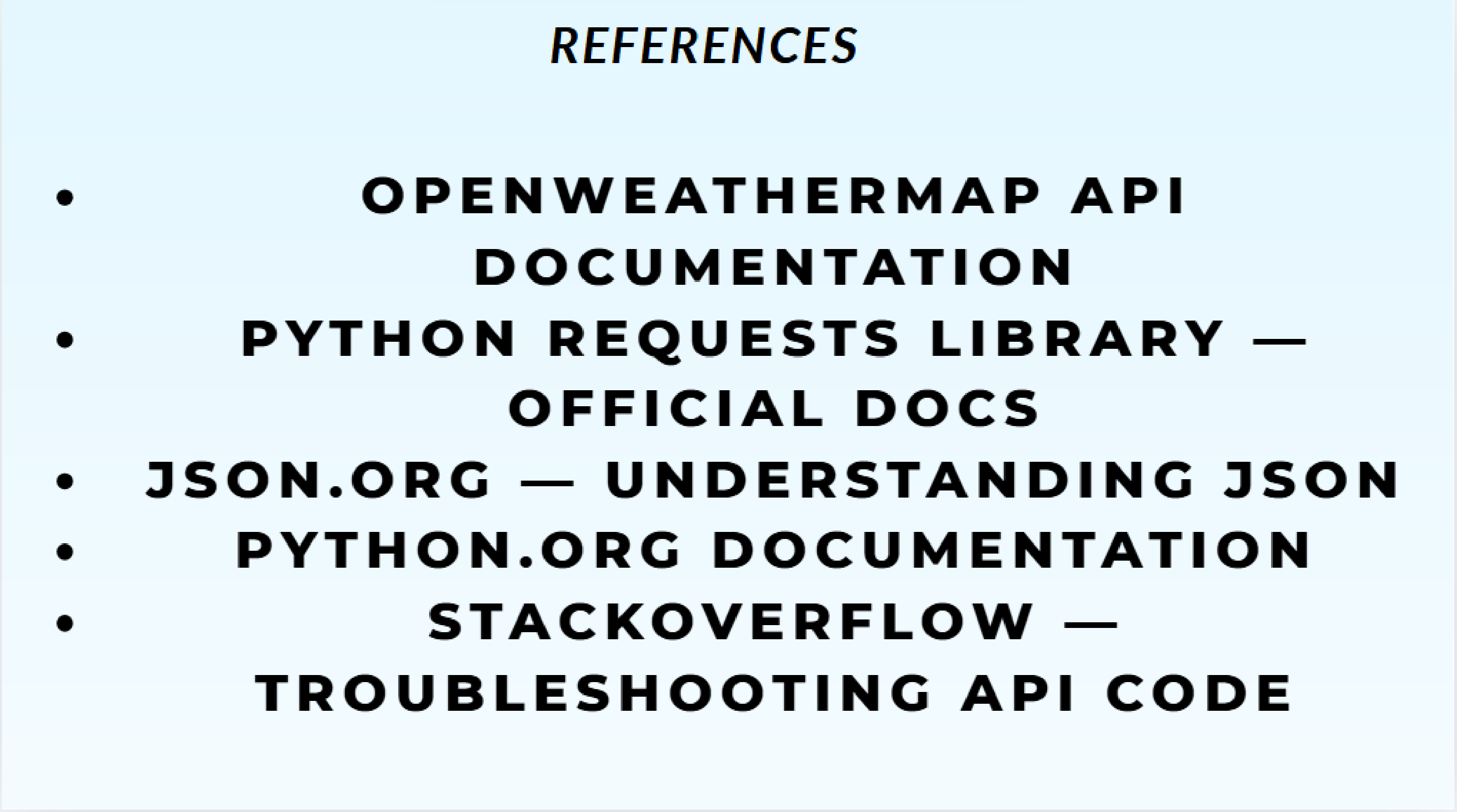
LEARNINGS & KEY TAKEAWAYS

- **LEARNED HOW TO INTEGRATE PYTHON WITH EXTERNAL APIS**
- **IMPROVED UNDERSTANDING OF JSON PARSING**
- **UNDERSTOOD RULE-BASED DECISION LOGIC**
- **GAINED EXPERIENCE IN DEBUGGING NETWORK REQUESTS**
- **REALIZED IMPORTANCE OF USER-FRIENDLY FEEDBACK MESSAGES**



FUTURE ENHANCEMENTS

- ADD SUPPORT FOR:
- WIND SPEED
- HUMIDITY
- RAIN PROBABILITY
- ADD GUI USING TKINTER OR PYQT
- BUILD MOBILE APP VERSION
- ADD IMAGE-BASED OUTFIT SUGGESTIONS
- STORE USER CLOTHING PREFERENCES
- ML-BASED PERSONALIZATION



REFERENCES

- OPENWEATHERMAP API DOCUMENTATION
- PYTHON REQUESTS LIBRARY — OFFICIAL DOCS
- JSON.ORG — UNDERSTANDING JSON
- PYTHON.ORG DOCUMENTATION
- STACKOVERFLOW — TROUBLESHOOTING API CODE