WEATHER APPLICATION

A MINI PROJECT REPORT

Submitted By

JOHN PRATHAP SINGH S 220701112
GOUTHAM A K 220701077
GIRIDHAR M 220701074

In partial fulfillment for the award of the degree of

BACHELOR OF

ENGINEERING

IN

COMPUTER SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE

(AUTONOMOUS)THANDALAM

CHENNAI-602105

2024 - 25

BONAFIDE CERTIFICATE

Certified that this project report "**WEATHER APPLICATION**" is the bonafide work of **"JOHN PRATHAP SINGH (220701112), A K GOUTHAM (220701077), GIRIDHAR M (220701074)"** who carried out the project work under my supervision.

Submitted for the Practical Examination held on

SIGNATURE	**SIGNATURE**
Dr.R.SABITHA	**Dr.G.Dharani Devi**
ACADEMIC HEAD	**ASSOCIATE PROFESSOR**
Dept. of Computer Science and Engg,	Dept. of Computer Science and Engg,
Rajalakshmi Engineering College	Rajalakshmi Engineering College
Thandalam,Chennai-602105	Thandalam,Chennai-602105

INTERNAL EXAMINER	**EXTERNAL EXAMINER**

ABSTRACT

This project is a Weather Application that fetches and displays current and historical weather data for a specified city using the OpenWeatherMap API.

The application is built using Python and Tkinter for the user interface, and it stores weather data in a MongoDB database. Users can enter a city name to retrieve the current weather conditions, including temperature, humidity, wind speed, and a weather description.

The application also displays up to 11 recent historical weather records for the same city, providing a time-based overview of past conditions.

Error handling is incorporated to manage invalid city names, ensuring users are prompted to enter valid inputs.

This application demonstrates the integration of real-time data retrieval, database management, and user-friendly interface design.

TABLE OF CONTENTS

1.1 INTRODUCTION

This project is a Weather Application designed to provide real-time and historical weather data for any specified city. Built with Python and Tkinter for the user interface, and MongoDB for data storage, the application allows users to enter a city name and retrieve the latest weather information. This includes details such as temperature, humidity, wind speed, and a weather description. Additionally, the application displays up to 11 recent weather records for the city, giving users an overview of weather changes over time.

The application also includes error handling to ensure users are informed if they enter an invalid city name.

This project combines real-time data retrieval, database interaction, and a user-friendly interface to deliver a comprehensive weather tracking tool.

1.2 OBJECTIVES

Real-Time Weather Data Retrieval:

- ❖ Implement a reliable system to fetch current weather data for specified cities from the OpenWeatherMap API.

Persistent Data Storage:

- ❖ Store weather data in MongoDB to allow for efficient retrieval and long-term trend analysis.

Historical Data Access:

- ❖ Provide endpoints to retrieve historical weather data, enabling users to access and analyze past weather information.

Resilient API Requests:

- ❖ Use the tenacity library to ensure robust API requests with retry mechanisms to handle temporary failures and network issues.

Concurrent client Handling:

- ❖ Develop a multithreaded socket server to manage multiple client connections concurrently, ensuring responsiveness and efficient communication.

Scalability and Performance:

- ❖ Design the application architecture to handle high traffic and large volumes of data efficiently, ensuring scalability and performance.

1.3 MODULES

Weather Data Fetching Module:

- **Description:** This module is responsible for fetching current weather data from the OpenWeatherMap API.
- **Components:**
    - API request handling
    - Data parsing and validation
    - Error handling and retries using the tenacity library

Tkinter GUI Module

- **Description:** This module facilitates the graphical user interface (GUI) for the Weather Application. It provides the necessary components for user interaction and data display.
- **Components:**
    - Main Window Management
    - Input Frame
    - Data Display Label
    - Event Handling
    - Integration with Application Logic

MongoDB Integration Module:

- **Description:** This module handles the interaction with the MongoDB database for storing and retrieving weather data.
- **Components:**
    - Database connection setup
    - Data insertion and querying functions
    - Indexing and optimization for efficient data retrieval

Historical Data Retrieval Module:

- **Description:** This module provides functionality to query and return historical weather data from MongoDB.
- **Components:**
    - Query construction based on city and timestamp
    - Data formatting and JSON response generation

Concurrency and Synchronization Module:

- **Description:** This module ensures the smooth operation of concurrent tasks within the application.
- **Components:**
    - Thread management for the socket server
    - Event handling for graceful shutdown
    - Synchronization mechanisms to prevent race conditions

Configuration and Environment Module:

- **Description:** This module manages the configuration settings and environment variables required for the application.

SURVEY OF TECHNOLOGIES

2.1 SOFTWARE DESCRIPTION

The weather application provides real-time and historical weather data using a combination of Flask, MongoDB, and a multithreaded socket server. Below is a concise description of its components and features:

CORE COMPONENTS :

Weather Data Fetching:

- **Functionality:** Fetches weather data from the OpenWeatherMap API using the requests library and ensures reliability with tenacity for retries.

MongoDB Integration:

- **Purpose:** Stores weather data including city name, weather conditions, temperature, humidity, wind speed, and timestamps for historical access.

Historical Data Retrieval:

- **Functionality:** Provides access to historical weather data stored in MongoDB.

Key Features

- **Real-Time Data:** Fetches current weather information for any city.
- **Historical Data Access:** Provides access to past weather data.
- **Resilient API Communication:** Ensures reliable data retrieval with automatic retries.

- **Concurrent Client Handling:** Supports multiple simultaneous client connections.
- **Scalability and Efficiency:** Handles high traffic and large data volumes.

Technical Specifications

- **Programming Language:** Python 3.7+
- **Frameworks and Libraries:** Tkinter, Pymongo, Requests, Tenacity
- **Database:** MongoDB 4.0+
- **APIs:** OpenWeatherMap API
- **Deployment:** Cloud platforms (AWS, Azure, Google Cloud)

2.2 LANGUAGES

Python:

- **Purpose:** Backend development.
- **Frameworks and Libraries:** Flask, Pymongo, Requests, Tenacity.
- **Version:** Python 3.7 or higher.

MongoDB:

- **Purpose:** Provides the database management for the Weather Application, storing and retrieving current and historical weather data.
- **Frameworks and Libraries:** MongoDB, Pymongo
- **Version:** MongoDB 4.0 or higher, Pymongo 3.11 or higher

REQUIREMENTS AND ANALYSIS

3.1 REQUIREMENTS SPECIFICATION

Functional Requirements

Real-Time Data Retrieval:

- Fetch current weather data from the OpenWeatherMap API based on user-provided city names.

Data Storage:

- Store weather data in MongoDB, including city name, weather conditions, temperature, humidity, wind speed, and timestamps.

Historical Data Access:

- Provide an endpoint to retrieve the last 12 historical weather entries for a specified city.

Resilient API Requests:

- Implement automatic retry mechanisms for API requests to handle transient errors.

API Endpoints:

- Offer /get_weather to fetch current weather data.
- Provide /get_historical_weather to retrieve historical weather data.

Non-Functional Requirements

Performance:

- The response time for fetching and displaying current weather data should not exceed 2 seconds.

Scalability:

- Design for horizontal scaling to manage increased load and data volume.

Reliability:

- Implement retry mechanisms with exponential backoff for API requests.

Security:

- Secure API keys and sensitive information.
- Implement basic input validation to prevent injection attacks.

Maintainability:

- Maintain modular, well-documented codebase.
- Include logging and monitoring for debugging and health tracking.

Usability:

- Ensure intuitive API endpoints with comprehensive documentation..

System Architecture

User Interface: Tkinter Module

- A GUI application built using Tkinter for user interaction.
- Provides input fields for the city name and displays current and historical weather data.

Database: MongoDB

- Stores weather data efficiently.
- Keeps historical weather data for querying and analysis.
- A WeatherData collection within a WeatherApplication database stores documents with weather information.

API Integration: OpenWeatherMap API

- Provides current weather data for specified cities.
- The Tkinter application sends requests to this API to fetch weather information.

Retry Mechanism: Tenacity Library

- Handles retries for API requests in case of temporary failures.
- Implements exponential backoff strategy to avoid overwhelming the API with repeated requests in case of failures.

3.2 HARDWARE AND SOFTWARE REQUIREMENTS

Software Requirements :

- **Python 3.7+:** Required for backend development and running the Tkinter , MongoDB integration, API requests.
- **Pymongo:** Python driver for MongoDB, necessary for interacting with the MongoDB database.
- **Requests:** Python library for making HTTP requests to the OpenWeatherMap API to fetch weather data.
- **Tenacity:** Library used for implementing resilient API request retry mechanisms.
- **MongoDB 4.0+:** NoSQL database required for storing weather data, including city names, weather conditions, temperature, humidity, wind speed, and timestamps.

Hardware Requirements :

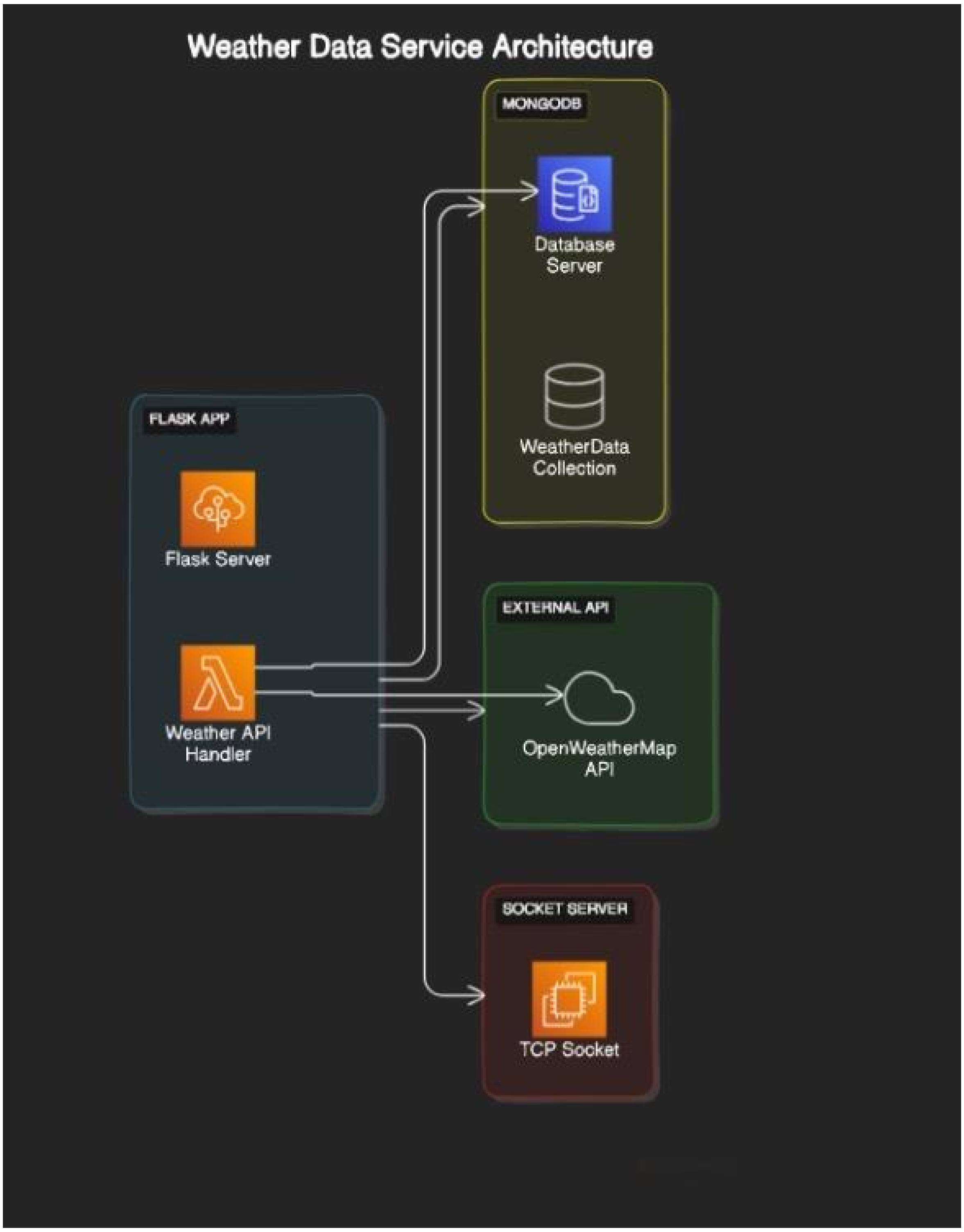
- **Processor:** Intel Core i3 or equivalent, 2.4 GHz or faster recommended for optimal performance.

- **Memory (RAM):** Minimum 4 GB RAM; 8 GB or more recommended for handling concurrent client connections and large data volumes.

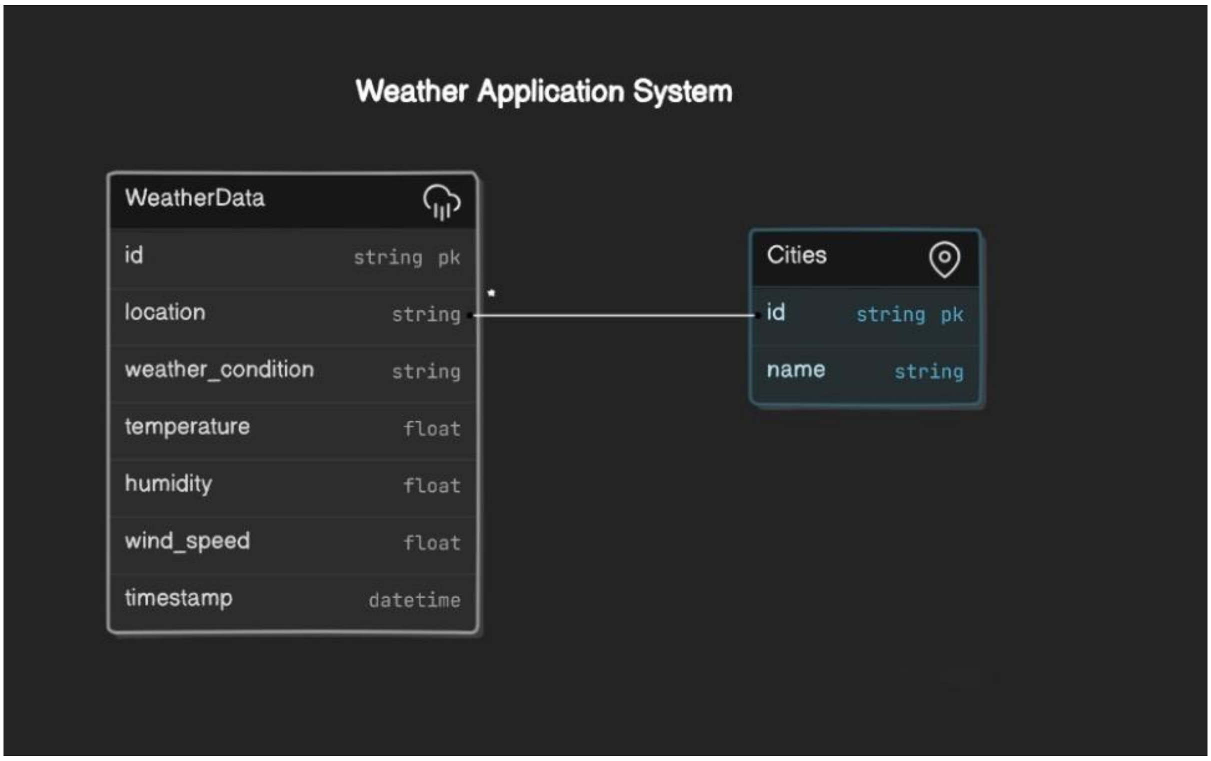
- **Storage:** At least 20 GB of available disk space for storing weather data and application files.

- **Network:** Stable internet connection required for accessing external APIs (e.g., OpenWeatherMap) and deploying the application for client-server communication.

3.3 ARCHITECTURE DIAGRAM

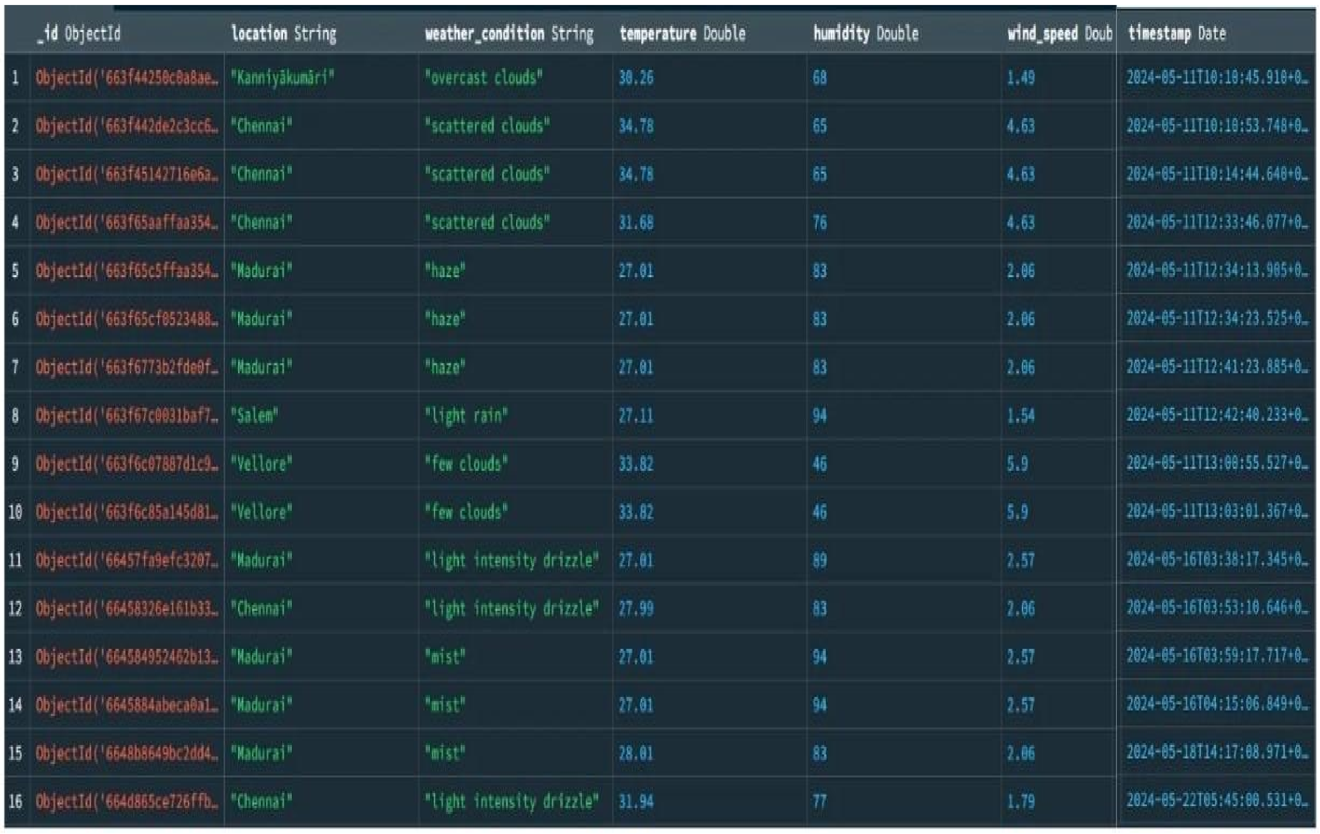


3.4 ER DIAGRAM



3.5 NORMALISATION

TABLE :



	_id ObjectId	location String	weather_condition String	temperature Double	humidity Double	wind_speed Doub	timestamp Date
1	ObjectId('663f44250c0a8ae…	"Kanniyākumāri"	"overcast clouds"	30.26	68	1.49	2024-05-11T10:10:45.910+0…
2	ObjectId('663f442de2c3cc6…	"Chennai"	"scattered clouds"	34.78	65	4.63	2024-05-11T10:10:53.748+0…
3	ObjectId('663f45142716e6a…	"Chennai"	"scattered clouds"	34.78	65	4.63	2024-05-11T10:14:44.640+0…
4	ObjectId('663f65aaffaa354…	"Chennai"	"scattered clouds"	31.68	76	4.63	2024-05-11T12:33:46.077+0…
5	ObjectId('663f65c5ffaa354…	"Madurai"	"haze"	27.01	83	2.06	2024-05-11T12:34:13.905+0…
6	ObjectId('663f65cf0523488…	"Madurai"	"haze"	27.01	83	2.06	2024-05-11T12:34:23.525+0…
7	ObjectId('663f6773b2fde0f…	"Madurai"	"haze"	27.01	83	2.06	2024-05-11T12:41:23.885+0…
8	ObjectId('663f67c0031baf7…	"Salem"	"light rain"	27.11	94	1.54	2024-05-11T12:42:40.233+0…
9	ObjectId('663f6c07887d1c9…	"Vellore"	"few clouds"	33.82	46	5.9	2024-05-11T13:00:55.527+0…
10	ObjectId('663f6c85a145d81…	"Vellore"	"few clouds"	33.82	46	5.9	2024-05-11T13:03:01.367+0…
11	ObjectId('66457fa9efc3207…	"Madurai"	"light intensity drizzle"	27.01	89	2.57	2024-05-16T03:38:17.345+0…
12	ObjectId('66458326e161b33…	"Chennai"	"light intensity drizzle"	27.99	83	2.06	2024-05-16T03:53:10.646+0…
13	ObjectId('664584952462b13…	"Madurai"	"mist"	27.01	94	2.57	2024-05-16T03:59:17.717+0…
14	ObjectId('6645884abeca0a1…	"Madurai"	"mist"	27.01	94	2.57	2024-05-16T04:15:06.849+0…
15	ObjectId('6648b8649bc2dd4…	"Madurai"	"mist"	28.01	83	2.06	2024-05-18T14:17:08.971+0…
16	ObjectId('664d865ce726ffb…	"Chennai"	"light intensity drizzle"	31.94	77	1.79	2024-05-22T05:45:00.531+0…

1NF :

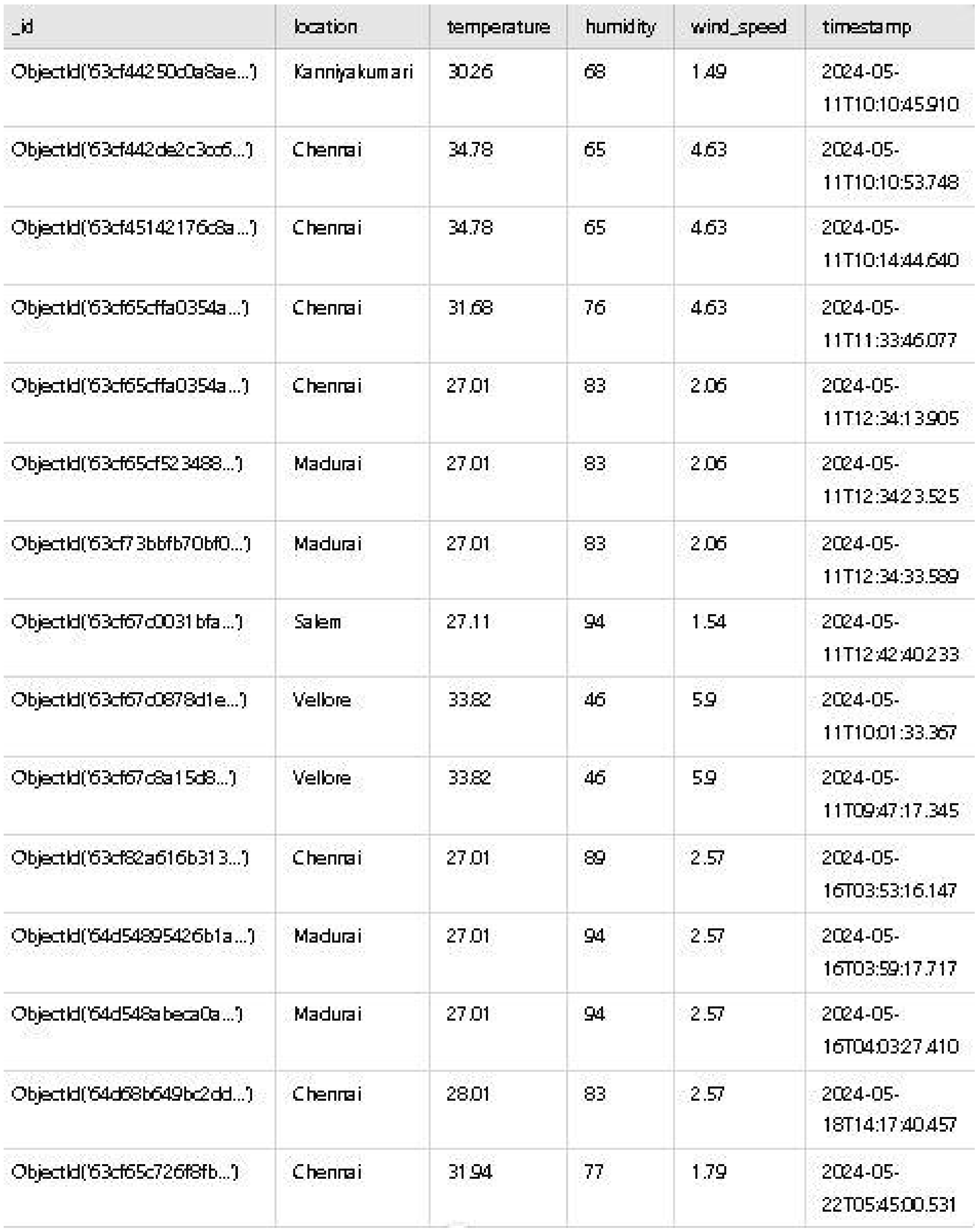
- The Original Table is in 1 NF

2NF :

- The Original Table is also in 2 NF

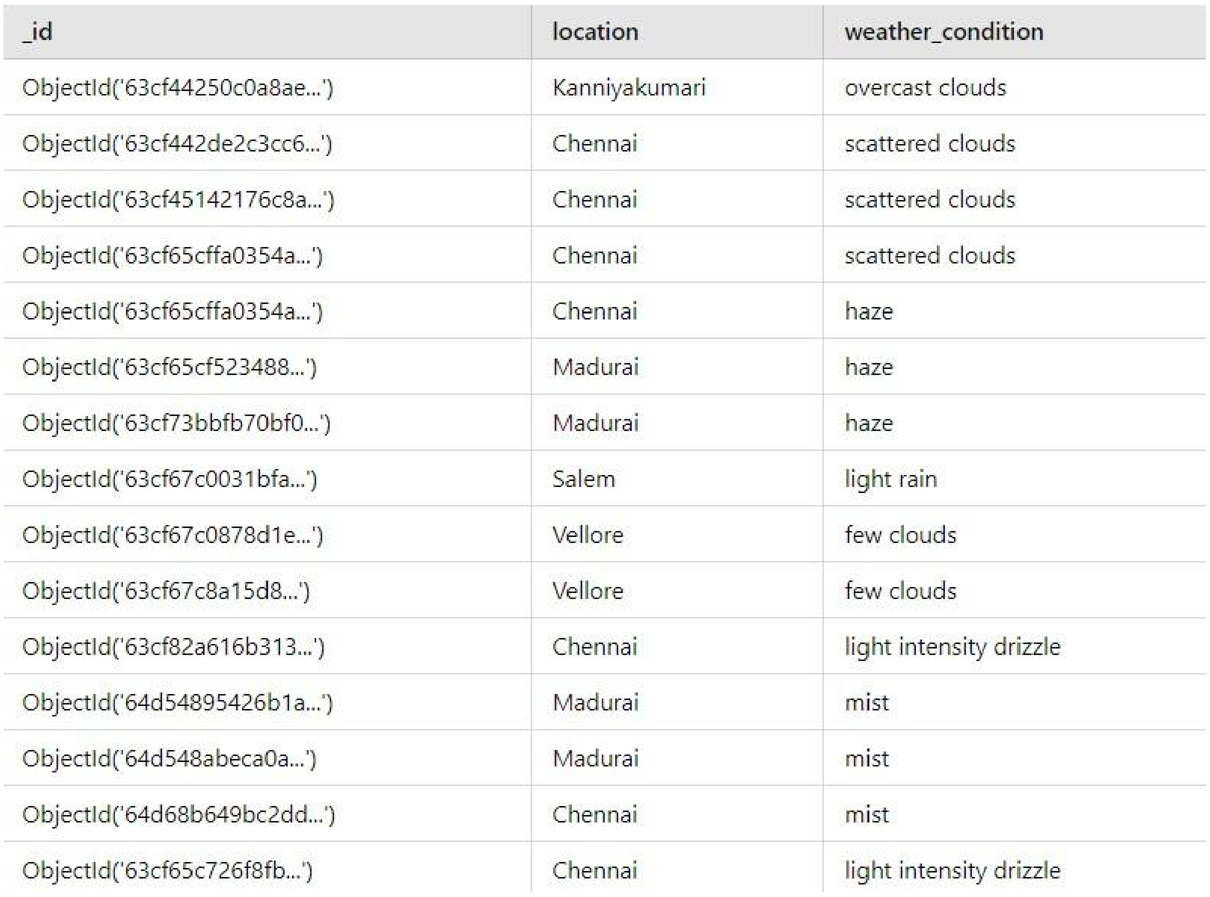
3NF :

WEATHER DATA TABLE :



_id	location	temperature	humidity	wind_speed	timestamp
ObjectId('63cf44250d0a8ae...')	Kanniyakumari	30.26	68	1.49	2024-05-11T10:10:45.910
ObjectId('63cf442de2c3cc6...')	Chennai	34.78	65	4.63	2024-05-11T10:10:53.748
ObjectId('63cf45142176c8a...')	Chennai	34.78	65	4.63	2024-05-11T10:14:44.640
ObjectId('63cf65cffa0354a...')	Chennai	31.68	76	4.63	2024-05-11T11:33:46.077
ObjectId('63cf65cffa0354a...')	Chennai	27.01	83	2.06	2024-05-11T12:34:13.905
ObjectId('63cf65cf523488...')	Madurai	27.01	83	2.06	2024-05-11T12:34:23.525
ObjectId('63cf73bbfb70bf0...')	Madurai	27.01	83	2.06	2024-05-11T12:34:33.589
ObjectId('63cf67c0031bfa...')	Salem	27.11	94	1.54	2024-05-11T12:42:40.233
ObjectId('63cf67c0878d1e...')	Vellore	33.82	46	5.9	2024-05-11T10:01:33.367
ObjectId('63cf67c8a15d8...')	Vellore	33.82	46	5.9	2024-05-11T09:47:17.345
ObjectId('63cf82a616b313...')	Chennai	27.01	89	2.57	2024-05-16T03:53:16.147
ObjectId('64d5489542 6b1a...')	Madurai	27.01	94	2.57	2024-05-16T03:59:17.717
ObjectId('64d548abeca0a...')	Madurai	27.01	94	2.57	2024-05-16T04:03:27.410
ObjectId('64d68b649bc2dd...')	Chennai	28.01	83	2.57	2024-05-18T14:17:40.457
ObjectId('63cf65c726f8fb...')	Chennai	31.94	77	1.79	2024-05-22T05:45:00.531

WEATHER CONDITIONS TABLE :



_id	location	weather_condition
ObjectId('63cf44250c0a8ae...')	Kanniyakumari	overcast clouds
ObjectId('63cf442de2c3cc6...')	Chennai	scattered clouds
ObjectId('63cf45142176c8a...')	Chennai	scattered clouds
ObjectId('63cf65cffa0354a...')	Chennai	scattered clouds
ObjectId('63cf65cffa0354a...')	Chennai	haze
ObjectId('63cf65cf523488...')	Madurai	haze
ObjectId('63cf73bbfb70bf0...')	Madurai	haze
ObjectId('63cf67c0031bfa...')	Salem	light rain
ObjectId('63cf67c0878d1e...')	Vellore	few clouds
ObjectId('63cf67c8a15d8...')	Vellore	few clouds
ObjectId('63cf82a616b313...')	Chennai	light intensity drizzle
ObjectId('64d54895426b1a...')	Madurai	mist
ObjectId('64d548abeca0a...')	Madurai	mist
ObjectId('64d68b649bc2dd...')	Chennai	mist
ObjectId('63cf65c726f8fb...')	Chennai	light intensity drizzle

PROGRAM CODE

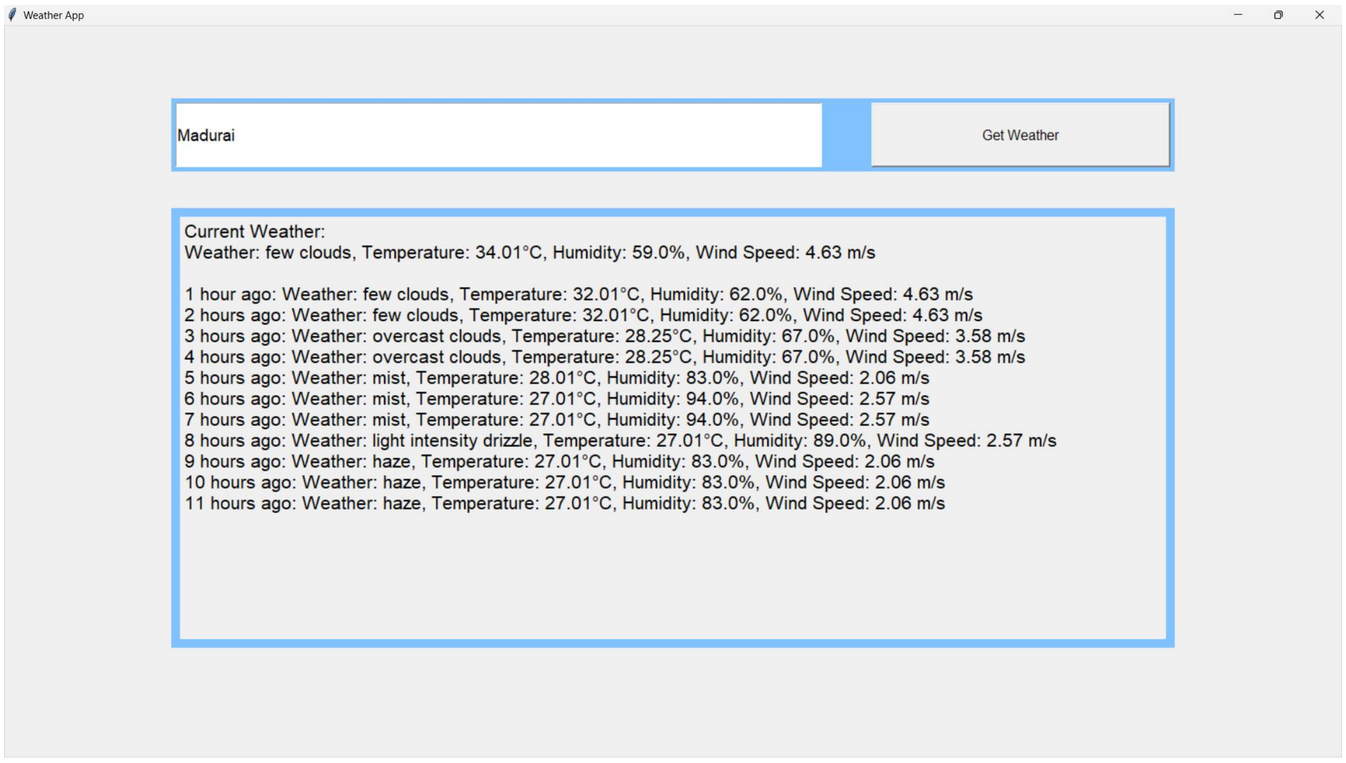
```
import tkinter as tk
import requests
from pymongo import MongoClient
import pymongo
from datetime import datetime, timezone
from tenacity import retry, wait_exponential, stop_after_attempt

# Set up MongoDB connection
client = MongoClient("mongodb://localhost:27017/")
db = client["WeatherApplication"]
collection = db["WeatherData"]

def get_weather(city_name):
  api_key = "e67d26ea926e3c5313b292170b4574a8"
  url=
      f"https://api.openweathermap.org/data/2.5/weather?q={city_name}&app
      id={api_key}&units=metric"
  try:
    response = requests.get(url)
    response.raise_for_status()
    weather_data = response.json()
    if "name" in weather_data:
      city_name = weather_data["name"]
      weather_condition = weather_data["weather"][0]["description"]
      temperature = weather_data["main"]["temp"]
      humidity = float(weather_data["main"]["humidity"])
```

```
            wind_speed = float(weather_data["wind"]["speed"])
            current_timestamp = datetime.now(timezone.utc)
            weather_info = {
                "location": city_name,
                "weather_condition": weather_condition,
                "temperature": temperature,
                "humidity": humidity,
                "wind_speed": wind_speed,
                "timestamp": current_timestamp
            }
            return weather_info
        else:
            return None
    except requests.exceptions.HTTPError as e:
        if e.response.status_code == 404:
            return None
        else:
            raise

def store_weather_data(weather_info):
    if weather_info:
        collection.insert_one(weather_info)
        print("Weather data stored successfully.")
    else:
        print("No weather data to store.")

def get_historical_weather(city):
    query = {"location": city}
    historical_weather_data = collection.find(query).sort("timestamp",
                                pymongo.DESCENDING).skip(1).limit(11)
    historical_weather_list = list(historical_weather_data)
    return historical_weather_list

def display_weather():
    city = entry.get()
    current_weather_info = get_weather(city)
    if current_weather_info:
        store_weather_data(current_weather_info)
        historical_weather = get_historical_weather(city)
        weather_info="Current Weather:\n"
weather_info+=(f"Weather:{current_weather_info['weather_conditon']},"
                    f"Temperature: {current_weather_info['temperature']}°C, "
                   f"Humidity:{current_weather_info['humidity']}%,"
                 f"WindSpeed: {current_weather_info['wind_speed']} m/s\n\n")
         for idx, data in enumerate(historical_weather):
            time_label = f"{idx + 1} hour{'s' if idx + 1 > 1 else ''} ago"
            weather_info += (f"{time_label}: Weather:
                               {data['weather_condition']}, "
                             f"Temperature: {data['temperature']}°C, "
                             f"Humidity: {data['humidity']}%, "
                             f"Wind Speed: {data['wind_speed']} m/s\n")

        label.config(text=weather_info)
    else:
        label.config(text="City not found. Please enter a valid city name.")

app = tk.Tk()
app.title("Weather App")
```

```
canvas = tk.Canvas(app, height=400, width=300)
canvas.pack()

frame = tk.Frame(app, bg="#80c1ff", bd=5)
frame.place(relx=0.5, rely=0.1, relwidth=0.75, relheight=0.1, anchor="n")

entry = tk.Entry(frame, font=("Arial", 14))
entry.place(relwidth=0.65, relheight=1)

button = tk.Button(frame, text="Get Weather", font=("Arial", 12),
command=display_weather)
button.place(relx=0.7, relheight=1, relwidth=0.3)

lower_frame = tk.Frame(app, bg="#80c1ff", bd=10)
lower_frame.place(relx=0.5, rely=0.25, relwidth=0.75, relheight=0.6,
anchor="n")
label = tk.Label(lower_frame, font=("Arial", 16), anchor="nw", justify="left",
bd=4)
label.place(relwidth=1, relheight=1)
app.mainloop()
```

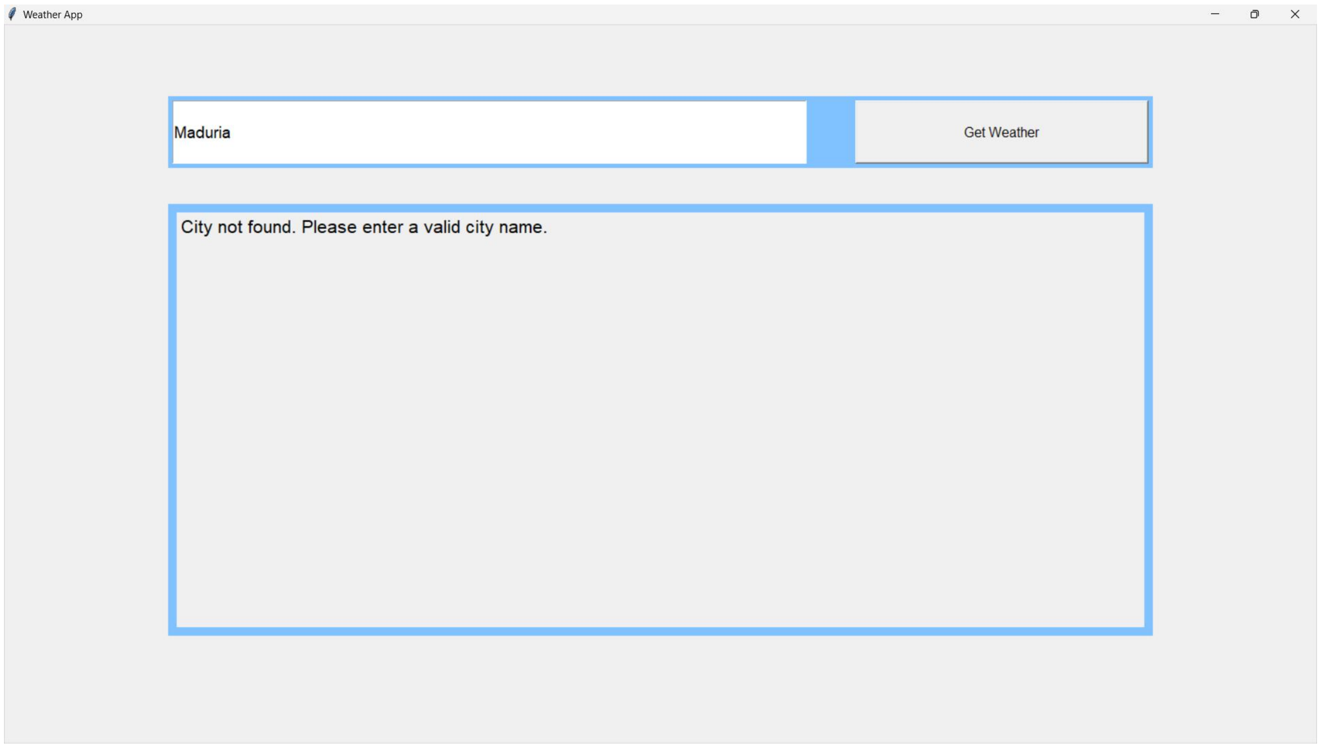
OUTPUT

When City Name is valid



```
Current Weather:
Weather: few clouds, Temperature: 34.01°C, Humidity: 59.0%, Wind Speed: 4.63 m/s

1 hour ago: Weather: few clouds, Temperature: 32.01°C, Humidity: 62.0%, Wind Speed: 4.63 m/s
2 hours ago: Weather: few clouds, Temperature: 32.01°C, Humidity: 62.0%, Wind Speed: 4.63 m/s
3 hours ago: Weather: overcast clouds, Temperature: 28.25°C, Humidity: 67.0%, Wind Speed: 3.58 m/s
4 hours ago: Weather: overcast clouds, Temperature: 28.25°C, Humidity: 67.0%, Wind Speed: 3.58 m/s
5 hours ago: Weather: mist, Temperature: 28.01°C, Humidity: 83.0%, Wind Speed: 2.06 m/s
6 hours ago: Weather: mist, Temperature: 27.01°C, Humidity: 94.0%, Wind Speed: 2.57 m/s
7 hours ago: Weather: mist, Temperature: 27.01°C, Humidity: 94.0%, Wind Speed: 2.57 m/s
8 hours ago: Weather: light intensity drizzle, Temperature: 27.01°C, Humidity: 89.0%, Wind Speed: 2.57 m/s
9 hours ago: Weather: haze, Temperature: 27.01°C, Humidity: 83.0%, Wind Speed: 2.06 m/s
10 hours ago: Weather: haze, Temperature: 27.01°C, Humidity: 83.0%, Wind Speed: 2.06 m/s
11 hours ago: Weather: haze, Temperature: 27.01°C, Humidity: 83.0%, Wind Speed: 2.06 m/s
```

When City Name is invalid



```
City not found. Please enter a valid city name.
```