

1)App.py:

```
from flask import Flask, request, jsonify, send_from_directory
import numpy as np
import tensorflow as tf
import joblib

app = Flask(__name__, static_url_path='', static_folder='./frontend')

model = tf.keras.models.load_model('models/lstm_weather_model.keras')
scaler = joblib.load('models/scaler.save')
WINDOW = 14
FEATURES = ['temp','humidity','windspeed','sealevelpressure']

@app.route('/')
def home():
    return send_from_directory('../frontend', 'index.html')

@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json()
    last_days = data['last_days']

    if len(last_days) != WINDOW:
        return jsonify({"error": f"Provide exactly {WINDOW} days of data"})


```

```

X_input = np.array([[day[feature] for feature in FEATURES] for day in last_days])

X_scaled = scaler.transform(X_input)

X_scaled = X_scaled.reshape(1, WINDOW, len(FEATURES))

pred_scaled = model.predict(X_scaled)

temp_min = scaler.data_min_[0]

temp_max = scaler.data_max_[0]

pred_temp = pred_scaled[0][0] * (temp_max - temp_min) + temp_min

return jsonify({"predicted_temperature": round(float(pred_temp),2)})}

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=5000)

```

2)data_prep.py:

```

import pandas as pd

import numpy as np

from sklearn.preprocessing import MinMaxScaler

import joblib

import os

def load_data(csv_path):

    df = pd.read_csv(csv_path, parse_dates=['DATE'])

    # Select features for model

    df = df[['DATE','temp','humidity','windspeed','sealevelpressure']]

```

```

df = df.sort_values('DATE').reset_index(drop=True)

return df


def create_sequences(values, window):
    X, y = [], []
    for i in range(len(values) - window):
        X.append(values[i:i+window])
        y.append(values[i+window, 0]) # first column is target (temp)
    return np.array(X), np.array(y)


def preprocess(csv_path='data/delhi_weather.csv',
              window=14,
              scaler_path='models/scaler.save',
              output_dir='models'):

    os.makedirs(output_dir, exist_ok=True)
    df = load_data(csv_path)

    # Convert dataframe to numpy array
    series = df[['temp','humidity','windspeed','sealevelpressure']].values

    scaler = MinMaxScaler()
    scaled = scaler.fit_transform(series)

    X, y = create_sequences(scaled, window)
    X = X.reshape((X.shape[0], window, X.shape[2]))

```

```
split = int(0.8 * len(X))

X_train, X_test = X[:split], X[split:]

y_train, y_test = y[:split], y[split:]

joblib.dump(scaler, scaler_path)

print(f"Preprocessing done. scaler saved to {scaler_path}")

return X_train, X_test, y_train, y_test, scaler
```

3)train.py:

```
import tensorflow as tf

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from data_prep import preprocess
import os

def build_model(input_shape):
    model = Sequential()
    model.add(LSTM(64, input_shape=input_shape, return_sequences=False))
    model.add(Dense(1)) # predict temp only
    model.compile(optimizer='adam', loss='mse')
    return model

def main():
    window = 14
```

```
# Preprocess data

X_train, X_test, y_train, y_test, scaler = preprocess(window=window)

print("Training data shape:", X_train.shape, y_train.shape)
print("Validation data shape:", X_test.shape, y_test.shape)

# Build model

model = build_model((X_train.shape[1], X_train.shape[2]))

# Train model

model.fit(X_train, y_train, epochs=50, batch_size=16, validation_data=(X_test, y_test))

# Evaluate

loss = model.evaluate(X_test, y_test)

print(f"Test loss: {loss}")

# Save model

os.makedirs('models', exist_ok=True)

model.save('models/lstm_weather_model.keras')

print("Model saved to models/lstm_weather_model.keras")

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

3)Index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Weather Prediction</title>
<link rel="stylesheet" href="style.css">
</head>
<body>
<div class="container">
    <h1>Daily Weather Predictor</h1>
    <p>Enter last 14 days weather data (JSON format):</p>
    <textarea id="dataInput" placeholder='Paste last 14 days JSON here'></textarea>
    <button id="predictBtn">Predict Temperature</button>
    <div id="loader" class="hidden"></div>
    <h2 id="result"></h2>
</div>

<script>
const predictBtn = document.getElementById("predictBtn");
const resultDiv = document.getElementById("result");
const loader = document.getElementById("loader");

```

```
predictBtn.addEventListener("click", async () => {

  try {

    const data = document.getElementById("dataInput").value;
    if(!data) return alert("Please enter last 14 days data!");

    loader.classList.remove("hidden");
    resultDiv.innerText = "";

    const response = await fetch('/predict', {
      method: 'POST',
      headers: {'Content-Type': 'application/json'},
      body: JSON.stringify({last_days: JSON.parse(data)})
    });

    const result = await response.json();
    loader.classList.add("hidden");

    if(result.error) {
      resultDiv.innerText = "Error: " + result.error;
      resultDiv.classList.add("error");
    } else {
      resultDiv.innerText = `🌡️ Predicted Temperature: ${result.predicted_temperature} °C`;
      resultDiv.classList.remove("error");
      resultDiv.classList.add("success");
    }
  }
})
```

```
    } catch (err) {  
        loader.classList.add("hidden");  
        resultDiv.innerText = "✖ Invalid JSON or server error!";  
        resultDiv.classList.add("error");  
    }  
});  
</script>  
</body>  
</html>
```

5)style.css:

```
body {  
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;  
    background: linear-gradient(to right, #4facfe, #00f2fe);  
    display: flex;  
    justify-content: center;  
    align-items: center;  
    height: 100vh;  
    margin: 0;  
    color: #333;  
}
```

```
.container {
```

```
background: #fff;  
padding: 30px 40px;  
border-radius: 20px;  
box-shadow: 0 10px 25px rgba(0,0,0,0.2);  
width: 500px;  
max-width: 90%;  
text-align: center;  
animation: fadeIn 1s ease-in-out;  
}  
  
h1 {
```

```
margin-bottom: 20px;  
color: #0077ff;  
}
```

```
textarea {  
width: 100%;  
height: 180px;  
padding: 15px;  
border-radius: 10px;  
border: 1px solid #ccc;  
font-family: monospace;  
font-size: 14px;  
resize: none;  
}
```

```
button {  
    margin-top: 20px;  
    padding: 12px 25px;  
    border: none;  
    border-radius: 10px;  
    background: linear-gradient(to right, #0077ff, #00c6ff);  
    color: #fff;  
    font-size: 16px;  
    cursor: pointer;  
    transition: all 0.3s ease;  
}  
  
button:hover {
```

```
    transform: scale(1.05);  
    box-shadow: 0 5px 15px rgba(0,0,0,0.3);  
}
```

```
#loader {  
    border: 5px solid #f3f3f3;  
    border-top: 5px solid #0077ff;  
    border-radius: 50%;  
    width: 40px;  
    height: 40px;  
    margin: 20px auto;  
    animation: spin 1s linear infinite;  
}
```

```
.hidden { display: none; }

.success { color: green; font-weight: bold; animation: fadeIn 0.8s; }

.error { color: red; font-weight: bold; animation: fadeIn 0.8s; }

@keyframes spin {
    0% { transform: rotate(0deg); }
    100% { transform: rotate(360deg); }
}

@keyframes fadeIn {
    from { opacity: 0; }
    to { opacity: 1; }
}
```

Output:

