Muhammad Attan
2022610013

Ecosense

"Smart System for Monitoring and Predicting Temperature Using IoT"

Cara Kerja Sistem

1. Sensor DHT11 untuk mengumpulkan data suhu dan kelembaban.
2. ESP8266 mengirim data ke server via protokol HTTP menggunakan FastAPI.
3. Data disimpan dalam database MySQL untuk keperluan training model dan monitoring.
4. Model Machine Learning dilatih menggunakan arsitektur DNN untuk prediksi suhu berdasarkan data historis.
5. Streamlit untuk menampilkan dashboard berbasis web untuk monitoring dan melihat prediksi suhu.

Alat dan Bahan

Alat

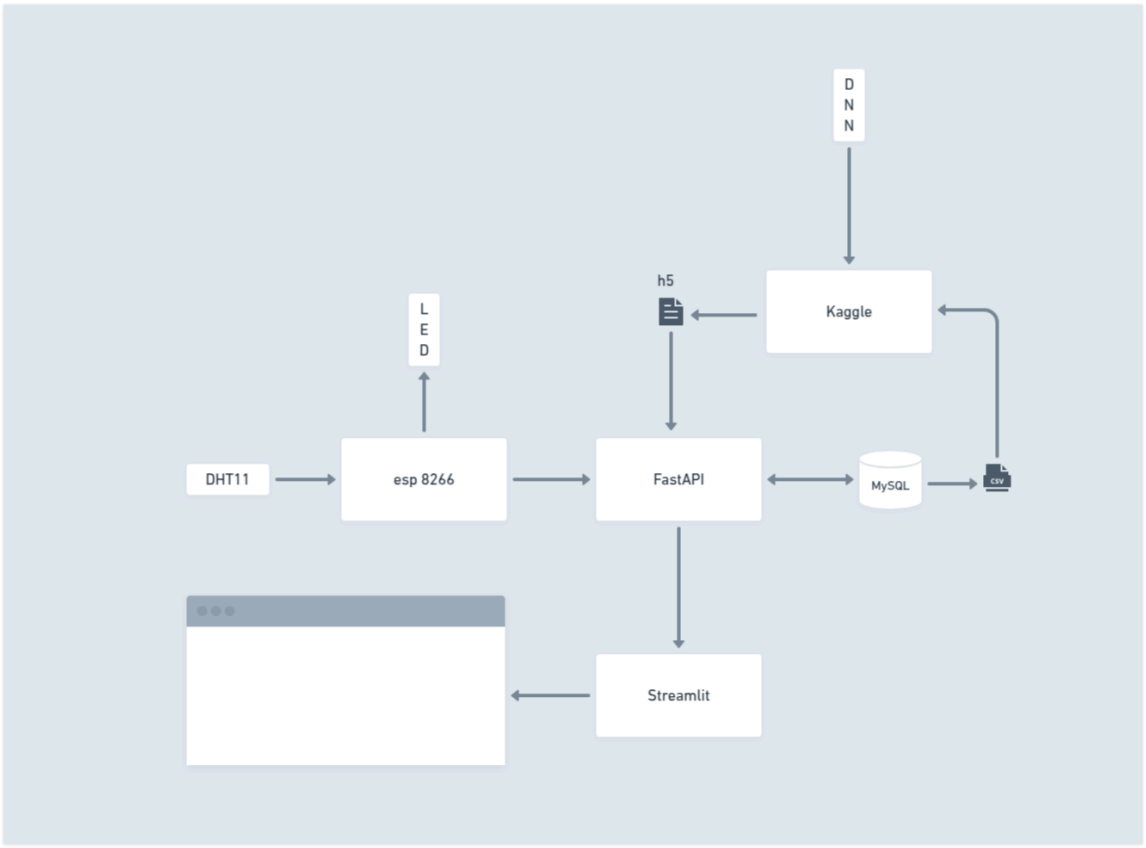
1. ESP8266
2. DHT11
3. Traffic LED
4. Laptop

Bahan

1. Arduino IDE
2. Python

3. FastAPI
4. Streamlit
5. Mysql
6. Kaggle
7. Tensorflow

Blok Diagram



Made with Whimsical

Script Program

Arduino

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
```

```
#include <ESP8266HTTPClient.h>
#include <DHT.h>

#define REDPIN D5
#define YELLOWPIN D6
#define GREENPIN D7

// Replace with your WiFi SSID and Password
const char* ssid = "rosify";
const char* password = "12345678";

// Server URL (the IP address of the Python server)
const String serverName = "http://192.168.36.170:8000/insert_data";

// Create a WiFiClient object
WiFiClient client;

// DHT Sensor setup
#define DHTPIN D2          // Pin where the DHT11 data pin is connected
#define DHTTYPE DHT11      // Defining the sensor type as DHT11
DHT dht(DHTPIN, DHTTYPE);  // Initialize the DHT sensor

// Function to connect to WiFi
void connectToWiFi() {
 WiFi.begin(ssid, password);

 // Wait until connected to WiFi
 while (WiFi.status() != WL_CONNECTED) {
   delay(1000);
   Serial.println("Connecting to WiFi...");
 }

 Serial.println("Connected to WiFi");
}

// Function to read temperature and humidity from the DHT11 sensor
void readDHT11Sensor(float &temperature, float &humidity) {
 humidity = dht.readHumidity();
 temperature = dht.readTemperature();  // Default is Celsius
```

```
 // Check if the readings failed and handle it
 if (isnan(humidity) || isnan(temperature)) {
   Serial.println("Failed to read from DHT sensor!");
   return;
 }
}

// Function to send data to the server
void sendDataToServer(float temperature, float humidity) {
 if (WiFi.status() == WL_CONNECTED) {
   HTTPClient http;
   http.begin(client, serverName); // Initialize the connection to the
server
   http.addHeader("Content-Type", "application/x-www-form-urlencoded");

   // Prepare the POST data
   String postData = "temperature=" + String(temperature, 2) +
"&humidity=" + String(humidity, 2);  // Send data with 2 decimal places

   // Send the POST request
   int httpResponseCode = http.POST(postData);

   // Check the server response
   if (httpResponseCode > 0) {
     Serial.println("Data sent successfully, HTTP Response Code: " +
String(httpResponseCode));
   } else {
     Serial.println("Error sending data, HTTP Response Code: " +
String(httpResponseCode));
   }

   http.end(); // Close the HTTP connection
 } else {
   Serial.println("Error in WiFi connection");
 }
}

void setup() {
 Serial.begin(9600);
  // Connect to WiFi
```

```
 connectToWiFi();

 // LED
 pinMode(REDPIN, OUTPUT);
 pinMode(YELLOWPIN, OUTPUT);
 pinMode(GREENPIN, OUTPUT);

 // Initialize the DHT sensor
 dht.begin();
}

void loop() {
 // Read temperature and humidity from the DHT11 sensor
 float temperature = 0;
 float humidity = 0;
 readDHT11Sensor(temperature, humidity);

 // Output the temperature and humidity values
 Serial.print("Temperature: ");
 Serial.print(temperature, 2);  // Print temperature with 2 decimal places
 Serial.print(" °C, Humidity: ");
 Serial.print(humidity, 2);  // Print humidity with 2 decimal places
 Serial.println(" %");

 if (temperature > 50) {
   digitalWrite(REDPIN, HIGH);
   digitalWrite(YELLOWPIN, LOW);
   digitalWrite(GREENPIN, LOW);
 } else if (temperature > 30 && temperature < 50) {
   digitalWrite(YELLOWPIN, HIGH);
   digitalWrite(REDPIN, LOW);
   digitalWrite(GREENPIN, LOW);
 } else if (temperature < 30) {
   digitalWrite(GREENPIN, HIGH);
   digitalWrite(YELLOWPIN, LOW);
   digitalWrite(REDPIN, LOW);
 } else {
   digitalWrite(GREENPIN, LOW);
   digitalWrite(YELLOWPIN, LOW);
   digitalWrite(REDPIN, LOW);
```

```
 }

 // Send data to the server
 sendDataToServer(temperature, humidity);

 // Wait for 5 seconds before sending data again
 delay(5000);
 digitalWrite(REDPIN, LOW);
}
```

Backend

```
from fastapi import FastAPI, Form, HTTPException
import aiomysql
import uvicorn
from tensorflow.keras.models import load_model
from sklearn.preprocessing import StandardScaler
import pandas as pd
from datetime import datetime, timedelta
from dotenv import load_dotenv
import os
import numpy as np

# Initialize the FastAPI app
app = FastAPI()

# Load the trained model and scaler
trained_model = load_model('./model/dnn.h5')
scaler = StandardScaler()

# Load environment variables from .env file
load_dotenv()

# Database connection helper function
async def get_db_connection():
    """Establish a connection to the MySQL database."""
    return await aiomysql.connect(
        host=os.getenv("DB_HOST", "localhost"),
```

```
        user=os.getenv("DB_USER", "root"),
        password=os.getenv("DB_PASSWORD", ""),
        db=os.getenv("DB_NAME", "ecosense")
    )

# Function to insert DHT11 data into the database
async def insert_ldr_data_to_db(temperature: float, humidity: float):
    """Insert the DHT11 data value into the database."""
    try:
        conn = await get_db_connection()
        async with conn.cursor() as cursor:
            await cursor.execute("INSERT INTO dht11 (temperature, humidity)
VALUES (%s, %s)", (temperature, humidity))
            await conn.commit()
    except aiomysql.MySQLError as err:
        raise HTTPException(status_code=500, detail=f"MySQL error: {err}")
    finally:
        conn.close()

# Function to get the latest temperature data from the database
async def get_latest_temperature_data():
    """Fetch the latest temperature data from the database."""
    try:
        conn = await get_db_connection()
        async with conn.cursor(aiomysql.DictCursor) as cursor:
            await cursor.execute("SELECT temperature, timestamp FROM dht11
ORDER BY timestamp DESC LIMIT 1")
            result = await cursor.fetchone()
        conn.close()
        return result
    except aiomysql.MySQLError as err:
        raise HTTPException(status_code=500, detail=f"MySQL error: {err}")

# Function to make a prediction for the next hour
def make_prediction(temperature: float, timestamp: str):
    """Make a prediction for the next hour based on the latest data."""

    timestamp_dt = timestamp
    new_data = pd.DataFrame([{
        'timestamp': timestamp_dt.timestamp(),
```

```
        'temperature': temperature
    }])
    X_new = new_data[['timestamp', 'temperature']].values
    scaler.fit(X_new)
    X_new_scaled = scaler.transform(X_new)
    predicted_temp = trained_model.predict(X_new_scaled)
    predicted_temp = np.round(predicted_temp, 2)
    new_timestamp = timestamp_dt + timedelta(hours=1)
    return {
        "predicted_temperature": float(predicted_temp[0]),
        "timestamp": new_timestamp.strftime('%Y-%m-%d %H:%M:%S')
    }

async def get_all_data():
    """Fetch all temperature and humidity data from the database."""
    try:
        conn = await get_db_connection()
        async with conn.cursor(aiomysql.DictCursor) as cursor:
            await cursor.execute("SELECT temperature, humidity, timestamp
FROM dht11")
            result = await cursor.fetchall()
        conn.close()
        return result
    except aiomysql.MySQLError as err:
        raise HTTPException(status_code=500, detail=f"MySQL error: {err}")

async def get_latest_data():
    """Fetch the latest temperature and humidity data from the database."""
    try:
        conn = await get_db_connection()
        async with conn.cursor(aiomysql.DictCursor) as cursor:
            await cursor.execute("SELECT temperature, humidity, timestamp
FROM dht11 ORDER BY timestamp DESC LIMIT 1")
            result = await cursor.fetchone()
        conn.close()
        return result
    except aiomysql.MySQLError as err:
        raise HTTPException(status_code=500, detail=f"MySQL error: {err}")

# Endpoint to get all temperature & humidity data
```

```python
@app.get("/all_data")
async def fetch_all_data():
    """Fetch all temperature and humidity data from the database."""
    all_data = await get_all_data()
    if not all_data:
        raise HTTPException(status_code=404, detail="No temperature data found in the database.")
    return {"status": "success", "data": all_data}

# Endpoint to get latest temperature & humidity data
@app.get("/latest_data")
async def fetch_latest_data():
    """Fetch the latest temperature and humidity data from the database."""
    latest_data = await get_latest_data()
    if not latest_data:
        raise HTTPException(status_code=404, detail="No temperature data found in the database.")
    return {"status": "success", **latest_data}

# Endpoint to insert LDR data
@app.post("/insert_data")
async def insert_data(temperature: float = Form(...), humidity: float = Form(...)):
    """Insert LDR data into the database and return a success message."""
    try:
        await insert_ldr_data_to_db(temperature, humidity)
        return {"message": "Data inserted successfully"}
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Error inserting data: {e}")

# Endpoint to run prediction for the next hour
@app.get("/predict_next_hour")
async def predict_next_hour():
    """Predict the temperature for the next hour based on the latest data."""
    latest_data = await get_latest_temperature_data()
    if not latest_data:
        raise HTTPException(status_code=404, detail="No temperature data found in the database.")
```

```
    temperature = latest_data['temperature']
    timestamp = latest_data['timestamp']
    prediction = make_prediction(temperature, timestamp)
    return {"status": "success", **prediction}

# Main execution (run the server)
if __name__ == "__main__":
    uvicorn.run(app, host="192.168.36.170", port=8000, reload=True)
```

Dashboard

```
import streamlit as st
import streamlit_shadcn_ui as ui
import requests
import pandas as pd
import plotly.express as px

# Constants
API_BASE_URL = "http://127.0.0.1:8000"
ROWS_PER_PAGE = 5

# Set page configuration
st.set_page_config(layout="wide", page_title="Ecosense", page_icon="🌿")
st.title("Ecosense Dashboard")

# Utility function to fetch data from the API
def fetch_data(api_url):
    """Fetch data from the API."""
    try:
        response = requests.get(api_url)
        response.raise_for_status()
        return response.json()
    except requests.exceptions.RequestException as e:
        st.error(f"Error fetching data: {e}")
        return None

def process_data(data):
```

```
    """Process the raw API data into a DataFrame."""
    if data and "data" in data:
        df = pd.DataFrame(data["data"])
        if {"temperature", "humidity", "timestamp"}.issubset(df.columns):
            df["Timestamp"] = pd.to_datetime(df["timestamp"])
            df = df[["Timestamp", "temperature",
"humidity"]].rename(columns={"temperature": "Temperature", "humidity":
"Humidity"})
            return df
    return None

def plot_interactive_chart(df):
    """Plot an interactive line chart for temperature and humidity."""
    fig = px.line(df, x='Timestamp', y=['Temperature', 'Humidity'],
                  labels={'value': 'Measurement', 'variable': 'Parameter'},
                  title='Temperature and Humidity Over Time')
    st.plotly_chart(fig)

def display_latest_data():
    """Fetch and display the latest data."""
    latest_data = fetch_data(f"{API_BASE_URL}/latest_data")
    if latest_data:
        cols = st.columns(3)
        with cols[0]:
            ui.metric_card(title="Latest Humidity",
content=f"{latest_data['humidity']}%", description="Latest humidity
level", key="humidity_card")
        with cols[1]:
            ui.metric_card(title="Latest Temperature",
content=f"{latest_data['temperature']}°C", description="Latest temperature
level", key="temperature_card")
        with cols[2]:
            predicted_temp =
fetch_data(f"{API_BASE_URL}/predict_next_hour")
            if predicted_temp:
                ui.metric_card(title="Predicted Temperature",
content=f"{predicted_temp['predicted_temperature']:.2f}°C",
description="Predicted temperature for the next hour",
key="predicted_temp_card")
    else:
```

```
        st.error("Failed to fetch the latest data.")

def display_all_data():
    """Fetch, process, and display all data."""
    data = fetch_data(f"{API_BASE_URL}/all_data")
    df = process_data(data)
    if df is not None:
        st.subheader("All Temperature and Humidity Data")
        total_rows = len(df)
        total_pages = (total_rows + ROWS_PER_PAGE - 1) // ROWS_PER_PAGE

        if 'page' not in st.session_state:
            st.session_state.page = 1

        df = df.sort_values(by="Timestamp", ascending=False)
        start_idx = (st.session_state.page - 1) * ROWS_PER_PAGE
        end_idx = start_idx + ROWS_PER_PAGE
        df_display = df.iloc[start_idx:end_idx].copy()
        df_display["Timestamp"] =
df_display["Timestamp"].dt.strftime('%Y-%m-%d %H:%M:%S')
        ui.table(data=df_display, maxHeight=300)

        cols = st.columns([1, 2, 1])
        with cols[0]:
            if st.button("Previous"):
                if st.session_state.page > 1:
                    st.session_state.page -= 1
        with cols[2]:
            if st.button("Next"):
                if st.session_state.page < total_pages:
                    st.session_state.page += 1

        st.subheader("Interactive Chart")
        plot_interactive_chart(df)
    else:
        st.error("No data available to display.")


def main():
    display_latest_data()
```

```
    display_all_data()

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

Machine Learning

```
import pandas as pd
import numpy as np
from datetime import datetime
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

class DNNModel:
    def __init__(self, data_path, model_path, plot_path):
        self.data_path = data_path
        self.model_path = model_path
        self.plot_path = plot_path
        self.model = None
        self.scaler = StandardScaler()

    def load_data(self):
        data = pd.read_csv(self.data_path)
        data['timestamp'] = data['timestamp'].apply(lambda x:
datetime.strptime(x, "%Y-%m-%d %H:%M:%S").timestamp())
        data['target'] = data['temperature'].shift(-1)
        data = data.dropna()
        self.X = data[['timestamp', 'temperature']].values
        self.y = data['target'].values

    def preprocess_data(self):
        self.X_train, self.X_test, self.y_train, self.y_test =
train_test_split(self.X, self.y, test_size=0.2, random_state=42)
        self.X_train = self.scaler.fit_transform(self.X_train)
        self.X_test = self.scaler.transform(self.X_test)

    def build_model(self):
```

```
        self.model = Sequential([
            Dense(units=64, input_dim=2, activation='relu'),
            Dense(units=32, activation='relu'),
            Dense(units=16, activation='relu'),
            Dense(units=1)
        ])
        self.model.compile(optimizer=Adam(), loss='mean_squared_error')

    def train_model(self, epochs=150, batch_size=16):
        self.history = self.model.fit(self.X_train, self.y_train,
epochs=epochs, batch_size=batch_size, validation_data=(self.X_test,
self.y_test))

    def save_model(self):
        self.model.save(self.model_path)

    def plot_training_loss(self):
        plt.figure(figsize=(10, 6))
        plt.plot(self.history.history['loss'], label='Training Loss')
        plt.plot(self.history.history['val_loss'], label='Validation Loss')
        plt.title('DNN Training and Validation Loss')
        plt.xlabel('Epochs')
        plt.ylabel('Loss')
        plt.legend()
        plt.savefig(self.plot_path)

    def predict(self):
        predictions = self.model.predict(self.X_test)
        return np.round(predictions, 2)

    def run(self):
        self.load_data()
        self.preprocess_data()
        self.build_model()
        self.train_model()
        self.save_model()
        self.plot_training_loss()
        predictions = self.predict()
        print(f'DNN Predictions (rounded): {predictions[:5]}')
        print("Model training complete and saved.")
```

```
if __name__ == "__main__":
    dnn_model = DNNModel(data_path='../dataset/dht11.csv',
model_path='../model/dnn.h5', plot_path='../training/dnn_loss.png')
    dnn_model.run()
```

Full Source in https://github.com/attmhd/ecosense.

Flowchart