FACULTY OF ELECTRICAL AND ELECTRONICS

ENGINEERING TECHNOLOGY

BVI 3114

TECHNOLOGY SYSTEM OPTIMIZATION II

PROGRAM-BASED LEARNING ASSESSMENT

IOT SENSOR DATA FORECASTING SYSTEM

NO	STUDENT ID	STUDENT NAME
1	VC22015	ISYRAF HILMI BIN IMRAN
2	VC22030	RAZMIN BIN ABDUL RAHIM

DATE OF REPORT SUBMISSION	17.7.2025

TABLE OF CONTENT

1. INTRODUCTION

In today's smart lab environments, even minor environmental changes-like a spike in CO_2 or a shift in lighting-can compromise experiments, waste energy, or create safety risks. Traditionally, monitoring these issues has been reactive: something goes wrong, and we try to fix it. But what if we could predict and prevent those problems before they happen?

That's exactly what our PBL project aimed to achieve.

Our goal was to create a fully functional IoT-powered system that doesn't just monitor environmental data-but also predicts it and responds in real time. Using simple, affordable components and cloud services, we built a smart data forecasting and automation system that brings labs one step closer to full autonomy.

2. PROJECT OVERVIEW

Here's how we turned that vision into a working system:

- Hardware Setup: ESP32 + Sensors
- Sensor: Ultrasonic Distance Sensor
- Controller: ESP32 Microcontroller for real-time data collection
- Cloud Integration:
    - Sensor data is sent every 10 seconds to Google Sheets via Wi-Fi and a Google Apps Script Webhook.
    - This removes the need for local servers and ensures data is always accessible.
- Data Visualization: Google Looker Studio (formerly Data Studio) turns our raw sensor readings into real-time dashboards accessible from any browser or mobile device.
- Forecasting Intelligence:
    - We implemented short-term forecasting using:
        - EMA (Exponential Moving Average)
        - SMA (Simple Moving Average)
    - Forecasts predict the next 24 hours for temperature, light, CO_2 levels, and distance.
- Automation and Smart Responses
    - Based on predictions, the system can:
        - Trigger ventilation early if CO_2 is expected to rise.
        - Warn or reroute moving robots if obstacles are forecasted.
        - This helps shift lab operations from reactive to proactive.

3. CORE CODE AND IMPLEMENTATION

- Arduino (ESP32) Code Highlight
  - Reads ultrasonic sensor every 10 seconds.
  - Sends data to Google Sheets as JSON.
  - Uses HTTPClient and ArduinoJson libraries.
  - Connects via Wi-Fi with hardcoded credentials.
- GitHub Repo
  - https://github.com/RAZSTAR01/PBL
- Google Apps Script Backend
  - Parses incoming JSON data.
  - Logs values into Google Sheets.
  - Forecast script (Simple Moving Average and Exponential Moving Average) predicts 24-hour trends.
  - Menu items added for manual controls and clearing old data

4. VISUALIZATION & APP DEVELOPMENT

- Google Sheet (Live Data)
  - Our Google Sheet acts as the live database and is constantly updated with new readings.
- Looker Studio Dashboard
  - A clean, mobile-friendly dashboard shows all environmental metrics in real-time.
- Live Dashboard
- Android App (Made with Android Studio)
  - Uses a WebView to embed the dashboard.
  - Includes a Refresh button.
  - Features zoom and scroll support.
  - Designed with clean UI for usability.

5. SYSTEM HIGHLIGHTS & MODIFICATION

- We tweaked the ESP32 to also include a ultrasonic distance sensor.
- Data interval was changed from 10 minutes to 10 seconds for real-time behaviour.
- Mobile app UI was enhanced for a better user experience.
- Dashboard layout was optimized for small screen displays.

6. CHALLENGES WE FACED

CHALLENGE	SOLUTION
Data delay from ESP32	Reduced interval to 10 seconds
Unstable Wi-Fi	Added retry logic and feedback for reconnection
Limited forecasting tools	Used forecasting method before upgrading
Google Scripts limits	Optimized locking and payload handling

7. CONCLUSION

This PBL journey helped us explore the complete lifecycle of an IoT data system from sensing to cloud logging, forecasting, automation, and app integration. The most exciting takeaway? We built a system that doesn’t just react it thinks ahead.

The skills gained include:

- IoT hardware integration.
- Cloud APIs (Google Apps Script)
- App development with Android Studio.
- Dashboard design with Looker Studio.

With just an ESP32 and a sensor, we made a lab smarter. And if we can do it here imagine the potential for smart classrooms, greenhouses, or even urban environments.

8. REFERENCES

- Arduino Code

```
#include <WiFi.h>
#include
<HTTPClient.h>
#include
<ArduinoJson.h>
#include <Ultrasonic.h>
// WiFi credentials
const char* ssid = "vivo V23 5G";
const char* password =
"123456789";
// Google Script ID - deploy as web app and get the
URL const char* scriptURL =
"https://script.google.com/macros/s/AKfycbz8tL6KiAsTixmGAz9g8FlHeyxLuQrTqSXmqcvMtZ
V
-P2PpRFp7_V4ZFJyI7zOALtkV/exec";
// Ultrasonic sensor configuration
const int ultrasonicPin = 13; // Digital pin connected to ultrasonic sensor
Ultrasonic ultrasonic(ultrasonicPin);
// Data sending interval (in milliseconds)
const unsigned long sendInterval = 10000; // 10
seconds unsigned long previousMillis = 0;
void setup() {
 // Initialize serial
 communication
 Serial.begin(115200);
 delay(1000);
 Serial.println("ESP32 Ultrasonic Ranger Data Logger");

 // Connect to WiFi
 WiFi.begin(ssid, password);
 Serial.print("Connecting to
 WiFi");
 while (WiFi.status() != WL_CONNECTED) {
 delay(500);
 Serial.print(".");
 }
 Serial.println();
 Serial.print("Connected to WiFi with IP: ");
 Serial.println(WiFi.localIP());
}
```

```
void loop() {
 unsigned long currentMillis = millis();

 // Check if it's time to send data
 if (currentMillis - previousMillis >= sendInterval) {
 previousMillis = currentMillis;

 // Read distance from ultrasonic
 sensor (in cm) long distance =
 ultrasonic.read();

 Serial.print("Distan
 ce: ");
 Serial.print(distanc
 e); Serial.println("
 cm");

 // Send data to Google Sheets
 sendDataToGoogleSheets(distance);
 }
}
void sendDataToGoogleSheets(long distance) {
 // Check WiFi connection
 if (WiFi.status() !=
 WL_CONNECTED) {
 Serial.println("WiFi not
 connected"); return;
 }

 HTTPClient http;
 http.begin(scriptUR
 L);
 http.addHeader("Content-Type", "application/json");

 // Create JSON
 data
 StaticJsonDocumen
 t<200> doc;
 doc["distance"] =
 distance;

String
 jsonString;
 serializeJson(do
 c, jsonString);
```

```
// Send HTTP POST request
int httpResponseCode = http.POST(jsonString);

if
(httpResponseCode
> 0) { String
response =
http.getString();
Serial.println("HTTP Response code: " +
String(httpResponseCode)); Serial.println("Response: "
+ response);
} else {
Serial.print("Error on sending POST: ");
Serial.println(httpResponseCode);
}

http.end();
```

- Google Apps Script Code

```
// Script to receive sensor data from ESP32 and log it to
Google Sheets function doGet(e) {
 return handleResponse(e);
}
function
 doPost(e) {
 return
 handleRespo
 nse(e);
}
function handleResponse(e) {
 // Process the incoming request
 var lock = LockService.getScriptLock();
 lock.tryLock(5000); // Wait 10 seconds for other processes to complete

 try {
 // Get the active sheet
 var spreadsheet =
SpreadsheetApp.openByUrl("https://docs.google.com/spreadsheets/d/1m3oH-
Ay2DI6iSvqG7pEf70r8MebjTJhXH4hbwpEDcE/edit");
 var sheet = spreadsheet.getSheetByName("Sheet1");


 // Parse the
 incoming data
 var payload;
 if (e.postData &&
 e.postData.contents) { payload
 =
 JSON.parse(e.postData.content
 s);
```

```
} else if
(e.parameter)
{ payload =
e.parameter;
} else {
return
ContentService.createTextOutput(JSON.stringi
fy({ 'status': 'error',
'message': 'No data received'
})).setMimeType(ContentService.MimeType.JSON);
}

// Prepare data array
for the sheet var
timestamp = new
Date();
var data = [timestamp];

// Get sensor data based on what's available in the payload
// Add appropriate sensor values to
the data array if (payload.temperature
!== undefined)
data.push(parseFloat(payload.tempera
ture));
if (payload.humidity !==
undefined)
data.push(parseFloat(payload
.humidity)); if
(payload.moisture !==
undefined)
data.push(parseFloat(payload
.moisture));
if (payload.light !== undefined)
data.push(parseFloat(payload.light)); if (payload.motion
!== undefined) data.push(payload.motion);
if (payload.distance !== undefined)
data.push(parseFloat(payload.distance));

// Insert data into the next row
sheet.appendRow(data);

// Return success response
return ContentService.createTextOutput(JSON.stringify({

'status': 'success',
'timestamp': timestamp.toString()
})).setMimeType(ContentService.MimeType.JSON);

} catch (error) {
// Return error response
return
ContentService.createTextOutput(JSON.stringi
fy({ 'status': 'error',
'message': error.toString()
})).setMimeType(ContentService.MimeType.JSON);
```

```
 } finally {
 lock.releaseLo
 ck();
 }
}
// Add menu to
sheet function
onOpen() {
 var ui =
 SpreadsheetApp.getUi();
 ui.createMenu('Sensor Data')
 .addItem('Clear All Data', 'clearData')
 .addToUi();
}
// Function to clear all data
except headers function
clearData() {
 var sheet =
 SpreadsheetApp.getActiveSpreadsheet().getActiveSheet(
 ); var lastRow = sheet.getLastRow();

 if (lastRow > 1) {
 sheet.deleteRows(2,
 lastRow - 1);
 }

 SpreadsheetApp.getUi().alert('All sensor data has been cleared!');
}

// Add this function to your existing Google Apps Script

// Function to generate forecasts (runs on time trigger or manual
execution) function generateForecasts() {
  var sheet = SpreadsheetApp.getActiveSpreadsheet().getSheetByName("Sheet1");
  var forecastSheet = SpreadsheetApp.getActiveSpreadsheet().getSheetByName("Forecasts");

  // If forecast sheet doesn't
  exist, create it if
  (!forecastSheet) {
    forecastSheet = SpreadsheetApp.getActiveSpreadsheet().insertSheet("Forecasts");
    // Add headers based on your sensor type
    forecastSheet.appendRow(["Timestamp", "Forecasted Value", "Upper Bound",
"Lower Bound"]);
  }

  // Get historical data (last 24 hours or maximum available)
  var dataRange = sheet.getRange(2, 1, sheet.getLastRow()-1,
  sheet.getLastColumn()); var values = dataRange.getValues();

// Extract timestamps and sensor values


  var
```

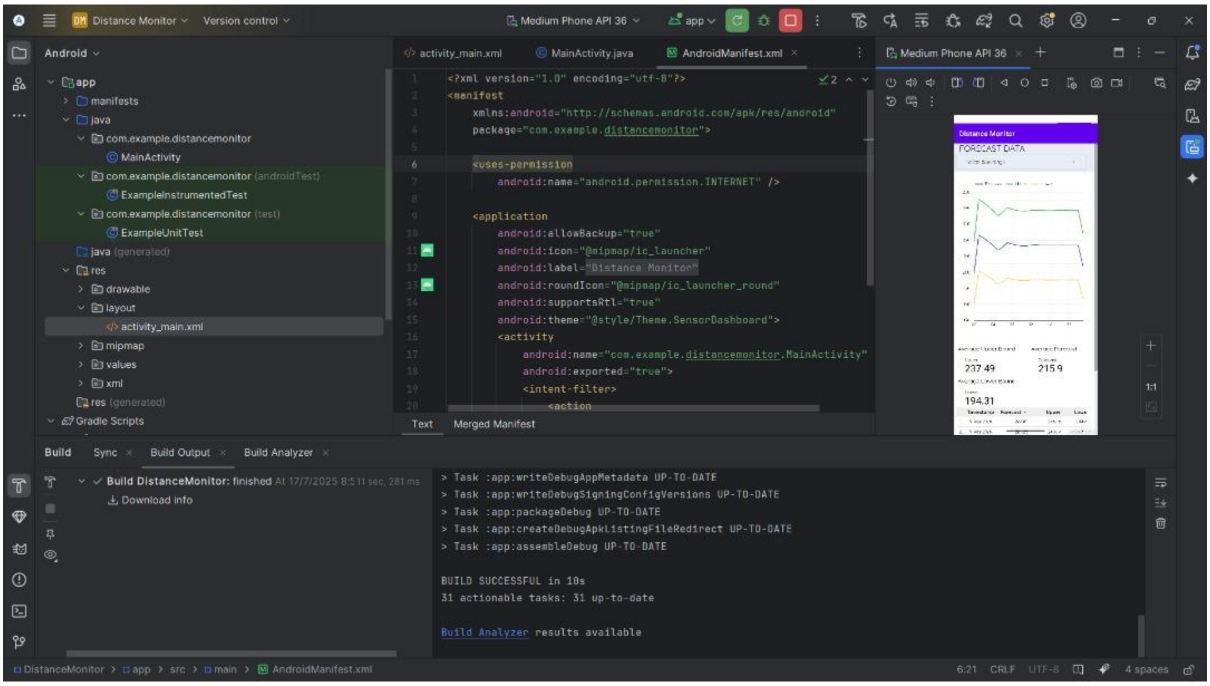
```
timestam
ps = [];
var
sensorVa
lues = [];

for (var i = 0; i < values.length; i++) {
  timestamps.push(values[i][0]); // Assuming timestamp is in column
  A sensorValues.push(values[i][1]); // Assuming sensor value is in
  column B
}

// Calculate forecasts using your chosen algorithm
var forecasts = calculateForecasts(timestamps, sensorValues);

// Clear previous forecasts
if (forecastSheet.getLastRow() > 1) {
  forecastSheet.getRange(2, 1, forecastSheet.getLastRow()-1, 4).clear();
}

// Add new forecasts
for (var i = 0; i <
  forecasts.length; i++) {
  forecastSheet.appendRow([
  forecasts[i].timestamp,
  forecasts[i].forecastValue,
  forecasts[i].upperBound,
  forecasts[i].lowerBound
  ]);
 }
}

// Implement your chosen forecasting
algorithm function
calculateForecasts(timestamps,
values) { var forecasts = [];

  // EXAMPLE: Simple Moving Average implementation
  // Replace with your chosen algorithm
  var windowSize = 6; // For 6-hour moving average

  // Generate forecasts for next 24 hours (at 1-hour intervals)
  var lastTimestamp = new Date(timestamps[timestamps.length - 1]);

for (var i = 1; i <= 24; i++) {
  var nextTimestamp = new Date(lastTimestamp.getTime() + (i * 60 * 60 * 1000));

  // Calculate forecast using rolling window
  var forecastValue = calculateSMA(values, windowSize);

  // Round to 2 decimal places
  forecastValue = Math.round(forecastValue * 100) / 100;

  // Add forecasted value to the values array for rolling updates
  values.push(forecastValue);
```

```
    // Add forecast with
    bounds forecasts.push({
      timestamp:
      nextTimestamp,
      forecastValue:
      forecastValue,
      upperBound: Math.round(forecastValue * 1.1 * 100) / 100,
    lowerBound: Math.round(forecastValue * 0.9 * 100) / 100
  });
}


  return forecasts;
}
// Example: Simple Moving Average implementation
function calculateSMA(values, windowSize) {
  if (values.length < windowSize) {
    windowSize = values.length; // Use all available data if not enough
  }

  var sum = 0;
  for (var i = values.length - windowSize; i < values.length; i++) {
    sum += values[i];
  }
  return sum / windowSize;
}

// Add button to sheet menu to generate forecasts manually
function onOpen() {
  var ui = SpreadsheetApp.getUi();
  ui.createMenu('Sensor Data')
    .addItem('Generate Forecasts', 'generateForecasts')
    .addItem('Clear All Data', 'clearData')
    .addToUi();
}
```

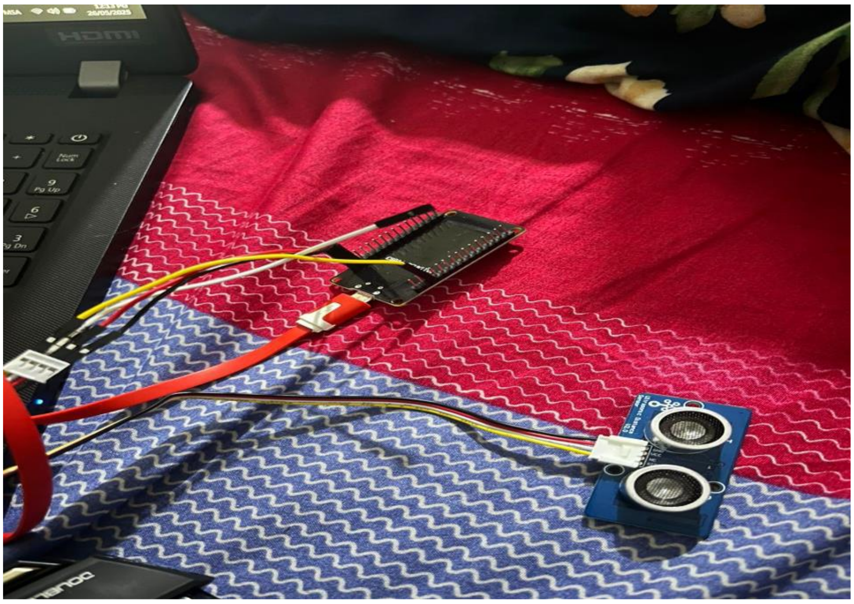
- Google Sheets Link
  - https://docs.google.com/spreadsheets/d/1i_UG_e6P4jl7CRTIaKHLuHTGTJW9Z1IRs1d-MNkviP4/edit?gid=0#gid=0

- Looker Studio Dashboard Link
  - https://lookerstudio.google.com/u/0/reporting/f33214b6-2a00-4b58-99f8-512dea7a9f58/page/2EZRF

- Android Studio
    - As part of the project, we developed a mobile application using Android Studio. The app integrates real-time data by embedding an iframe from Looker Studio, displaying live readings from the distance sensors in a clean, user-friendly interface.



- Sensor Setup



Ultrasonic Distance Sensor:

  - Connect VCC to 5V on ESP32
  - Connect GND to GND on ESP32
  - Connect SIG to a digital GPIO

- Google Sheets

1	Timestamp	Distance
2	15/05/2025 21:52:02	235.263
3	15/05/2025 21:52:12	5.899
4	15/05/2025 21:52:21	235.229
5	15/05/2025 21:52:31	235.297
6	15/05/2025 21:52:41	235.705
7	15/05/2025 21:52:51	235.348
8	15/05/2025 21:53:01	235.314
9	15/05/2025 21:53:11	235.263
10	15/05/2025 21:53:21	5.508
11	15/05/2025 21:53:31	516.732
12	16/05/2025 21:53:41	516.477
13	16/05/2025 21:53:52	31.45
14	16/05/2025 21:54:01	516.511
15	16/05/2025 21:54:12	516.545
16	16/05/2025 21:54:22	516.562
17	16/05/2025 21:54:32	516.443
18	16/05/2025 21:54:42	35.071
19	16/05/2025 21:54:59	0.612
20	16/05/2025 22:01:09	233.988
21	16/05/2025 22:01:18	234.362
22	17/05/2025 22:01:28	234.277
23	17/05/2025 22:01:40	3.621
24	17/05/2025 22:01:49	9.35