FAKULTI TEKNOLOGI KEJURUTERAAN ELEKTRIK DAN ELEKTRONIK

PROJECT:

AI-INTEGRATION SENSOR FORECASTING AND SUMMARIZATION

BVI3114 TECHNOLOGY SYSTEM OPTIMIZATION II

BIL	NO ID	NAME
1	VC22017	FARIS AZRI BIN FAISAL RIDZA
2	VC22013	MUHAMMAD AKMAL HAIKAL BIN AZMI
3	VC22029	MUHAMMAD FAUZAN SYAFIQ BIN ROSLAN

Table of Contents

1. Executive Summary

This project demonstrates a real-time sensor data logging and forecasting system integrated with AI-based summarization using Google Apps Script. Light intensity data is collected and logged into Google Sheets. Forecasting is performed using Exponential and Simple Moving Averages (EMA & SMA), while Gemini AI generates automatic summaries to provide meaningful insights.

2. System Architecture Diagram & Workflow

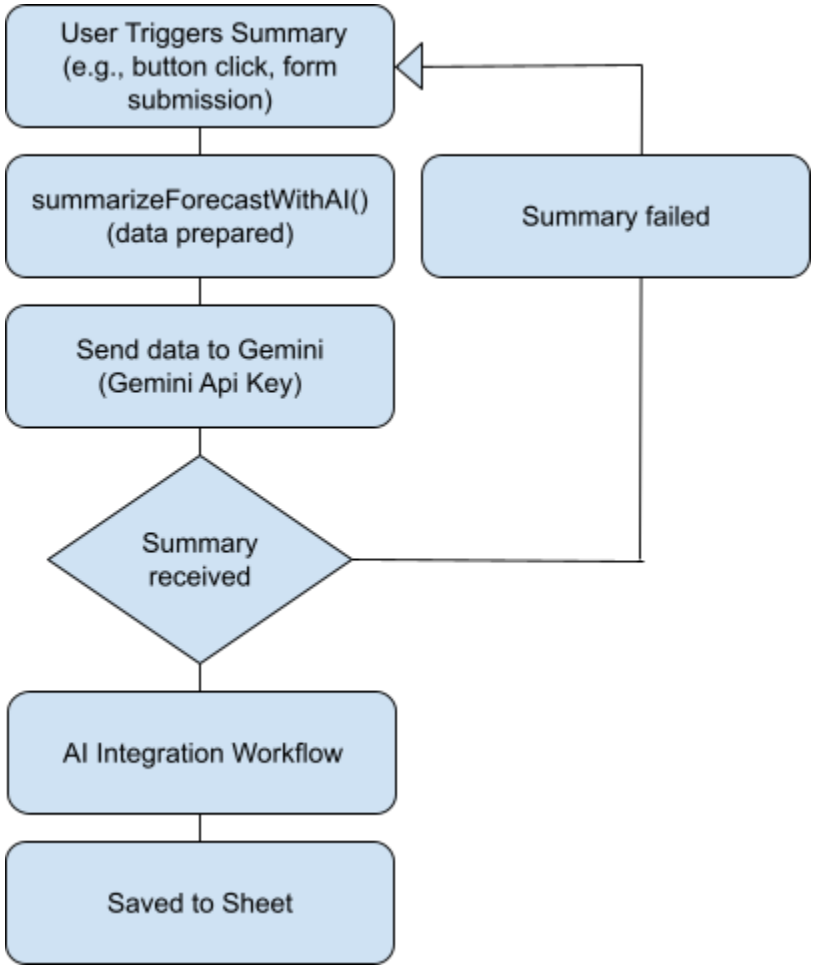


Figure 1: System Architecture Diagram

Here's the sequence of operations for the "*System Architecture Diagram* " in a bulleted list:

- **User Starts Summary**
  User clicks a button or submits a form to generate a summary.

- **Text is Prepared**
  The system gets the text from a spreadsheet, cleans it, and breaks it into parts if it's too long.

- **Send to Gemini AI**
  The cleaned text is sent to the Gemini API with instructions to summarize.

- **Summary is Received**
  Gemini AI returns a short summary based on the input text.

- **Save to Spreadsheet**
  The summary is saved back to the spreadsheet along with the original text.

2.1 **User Starts Summary**

```
35
36  // Get summaries
37  const realtimeData = realtimeSheet.getDataRange().getValues();
38  const forecastData = forecastSheet.getDataRange().getValues();
39
40  const realtimePrompt = "This is real-time sensor data:\n" + convertToCSV(realtimeData) + "\nSummarize the trends.";
41  const forecastPrompt = "This is forecast sensor data:\n" + convertToCSV(forecastData) + "\nSummarize expected trends.";
42
43  const realtimeSummary = getGeminiSummary(realtimePrompt);
44  const forecastSummary = getGeminiSummary(forecastPrompt);
45
46  // Output to Summary sheet
47  summarySheet.clear();
48  summarySheet.getRange("A1").setValue("Real-Time Data Summary:");
49  summarySheet.getRange("A2").setValue(realtimeSummary);
50  summarySheet.getRange("A4").setValue("Forecast Data Summary:");
51  summarySheet.getRange("A5").setValue(forecastSummary);
52  summarySheet.getRange("A7").setValue("Last Updated: " + new Date().toLocaleString());
53
54  Logger.log("Real-Time Summary:\n" + realtimeSummary);
55  Logger.log("Forecast Summary:\n" + forecastSummary);
56  }
```

- This action starts the process to generate a summary from existing text data.
- The text to be summarized is typically stored in Google Sheets

2.2 **Text is Prepared**

	A	B
1	Timestamp	Distance(cm
2	4/12/2025 2:46:55	127.00
3	4/12/2025 2:47:05	100.00
4	4/12/2025 2:47:15	122.00
5	4/12/2025 2:47:24	128.00
6	4/12/2025 2:47:35	130.00
7	4/12/2025 2:47:45	125.00
8	4/12/2025 2:47:58	90.00
9	4/12/2025 2:48:08	17.00
10	4/12/2025 2:48:15	130.00
11	4/12/2025 2:48:24	122.00
12	4/12/2025 2:48:32	135.00
13	4/12/2025 2:48:45	112.00
14	4/12/2025 2:48:55	11.00
15	4/13/2025 2:49:04	122.00
16	4/13/2025 2:49:16	123.00
17	4/13/2025 2:49:25	102.00
18	4/13/2025 2:49:35	108.00
19	4/13/2025 2:49:44	109.00
20	4/13/2025 2:49:55	112.00
21	4/13/2025 2:50:05	59.00
22	4/13/2025 2:50:15	12.00
23	4/13/2025 2:50:25	123.00
24	4/13/2025 2:50:34	124.00
25	4/13/2025 2:50:45	180.00
26	4/13/2025 2:50:55	120.00
27	4/13/2025 2:51:05	227.00

	A	B	C	D
1	timestampe	forecast	upper	lower
2	4/16/2025 3:58:14	120	132	108
3	4/16/2025 4:58:14	128.33	141.16	115.5
4	4/16/2025 5:58:14	140.56	154.62	126.5
5	4/16/2025 6:58:14	143.98	158.38	129.58
6	4/16/2025 7:58:14	129.65	142.62	116.69
7	4/16/2025 8:58:14	113.42	124.76	102.08
8	4/16/2025 9:58:14	129.32	142.25	116.39
9	4/16/2025 10:58:14	130.88	143.97	117.79
10	4/16/2025 11:58:14	131.3	144.43	118.17
11	4/16/2025 12:58:14	129.76	142.74	116.78
12	4/16/2025 13:58:14	127.39	140.13	114.65
13	4/16/2025 14:58:14	127.01	139.71	114.31
14	4/16/2025 15:58:14	129.28	142.21	116.35
15	4/16/2025 16:58:14	129.27	142.2	116.34
16	4/16/2025 17:58:14	129	141.9	116.1
17	4/16/2025 18:58:14	128.62	141.48	115.76
18	4/16/2025 19:58:14	128.43	141.27	115.59
19	4/16/2025 20:58:14	128.6	141.46	115.74
20	4/16/2025 21:58:14	128.87	141.76	115.98
21	4/16/2025 22:58:14	128.8	141.68	115.92
22	4/16/2025 23:58:14	128.72	141.59	115.85
23	4/17/2025 0:58:14	128.67	141.54	115.8
24	4/17/2025 1:58:14	128.68	141.55	115.81
25	4/17/2025 2:58:14	128.72	141.59	115.85

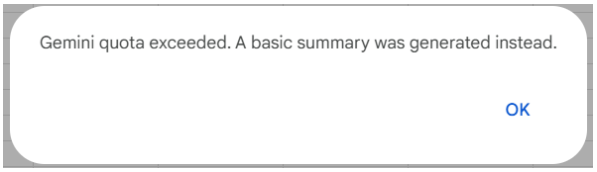
- The system retrieves the original text (e.g., sensor logs, notes, or descriptions) from a specific sheet.
- Basic cleaning and formatting is done to ensure the text is suitable for AI processing

2.3 **Send to Gemini AI**

```
function getGeminiSummary(prompt) {
  const payload = {
    contents: [{ parts: [{ text: prompt }] }]
  };
  const options = {
    method: 'POST',
    contentType: 'application/json',
    payload: JSON.stringify(payload),
    muteHttpExceptions: true
  };
  const url = 'https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent?key=' + apiKey;
  const response = UrlFetchApp.fetch(url, options);
  const result = JSON.parse(response.getContentText());
  return result.candidates?.[0]?.content?.parts?.[0]?.text || "No summary returned.";
}

// Get summaries
const realtimeData = realtimeSheet.getDataRange().getValues();
const forecastData = forecastSheet.getDataRange().getValues();

const realtimePrompt = "This is real-time sensor data:\n" + convertToCSV(realtimeData) + "\nSummarize the trends.";
const forecastPrompt = "This is forecast sensor data:\n" + convertToCSV(forecastData) + "\nSummarize expected trends.";

const realtimeSummary = getGeminiSummary(realtimePrompt);
const forecastSummary = getGeminiSummary(forecastPrompt);
```

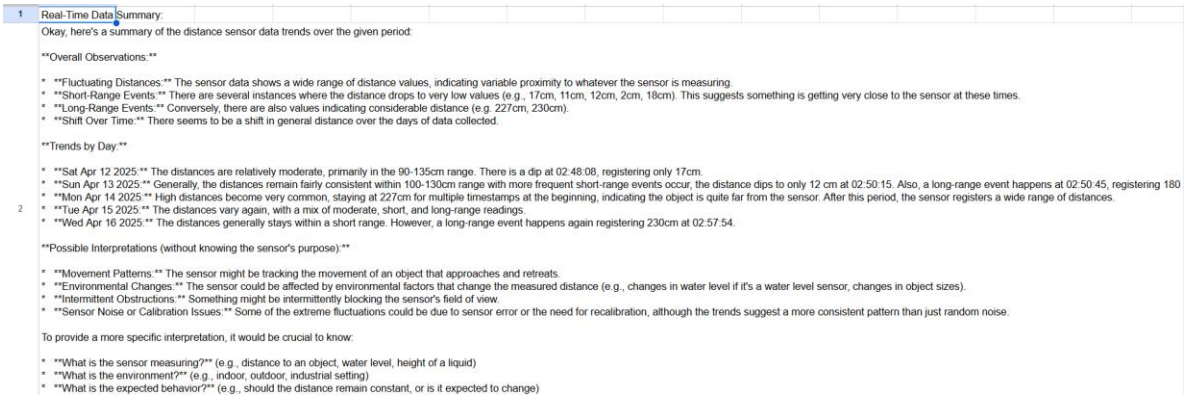
- This step includes authentication using an API key and model selection (e.g., `gemini-1.5-pro`)
- The cleaned text is sent over the internet to the **Gemini AI API**

2.4 **Summary is received**



- Gemini processes the prompt and generates a summarized version of the original text.
- This response is returned as part of the API call output.

2.5 **Save to spreadsheet**



- The returned summary is saved into the Google Sheet, alongside the original text.
- A new row is added for each processed entry with timestamps

3. Google Apps Script Backend

The backend is fully implemented in **Google Apps Script** within a single `.gs` file. It integrates sensor data logging, forecasting algorithms, AI summarization, and spreadsheet-driven UI interaction

- **Data Logging**
  Accepts POST requests via `doPost(e)` to log light sensor data with timestamps into `Sheet1`.
- **AI Summarization**
  The `summarizeForecastWithAI()` function sends forecast trends to **Gemini API** for natural-language summaries.
- **Dynamic Charting**
  Forecast charts are updated directly in Google Sheets using `SpreadsheetApp.getActiveSpreadsheet().getSheetByName(...)`.
- **Forecast Reset**
  `clearForecasts()` clears previous forecast results to avoid data overlap or confusion.

```
function doGet(e) {
 return handleResponse(e);
}
function doPost(e) {
 return handleResponse(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/1m3oHAy2DI6iSvqG7p
Ef70r8MebjJTJhXH4hbwpEDcE/edit");
var sheet = spreadsheet.getSheetByName("Sheet1"); // Parse the incoming data
var payload; if (e.postData && e.postData.contents) {
payload = JSON.parse(e.postData.contents);
}
else if (e.parameter) {
payload = e.parameter;
}
else {
return ContentService.createTextOutput(JSON.stringify({
'status': 'error',
'message': 'No data received'
}
)
)
.setMimeType(ContentService.MimeType.JSON); }
```

4. Sensor Data Acquisition

- **Receives** a POST request from a device (e.g., ESP32). // Get the Sheet1
- **Parses** the JSON body to extract the `distance` value. // Extract the 'distance' field
- **Appends** the timestamp and light value as a new row in **Sheet1**. // Add a new row with timestamp and distance value
- **Returns** a JSON response indicating success or failure. // If something goes wrong, return error response

5. Forecasting Algorithm Selection **(EMA & SMA)**

This project uses two time-series forecasting methods: Exponential Moving Average (EMA) and Simple Moving Average (SMA). These were chosen because they are both accurate and efficient for handling real-time sensor data.

5.1 Exponential Moving Average (EMA)

EMA is a forecasting technique that gives more weight to recent data points, making it sensitive to short-term changes while smoothing out random noise.

- Formula: `EMA[i] = alpha * value[i] + (1 - alpha) * EMA[i-1]`
- Captures short-term trends

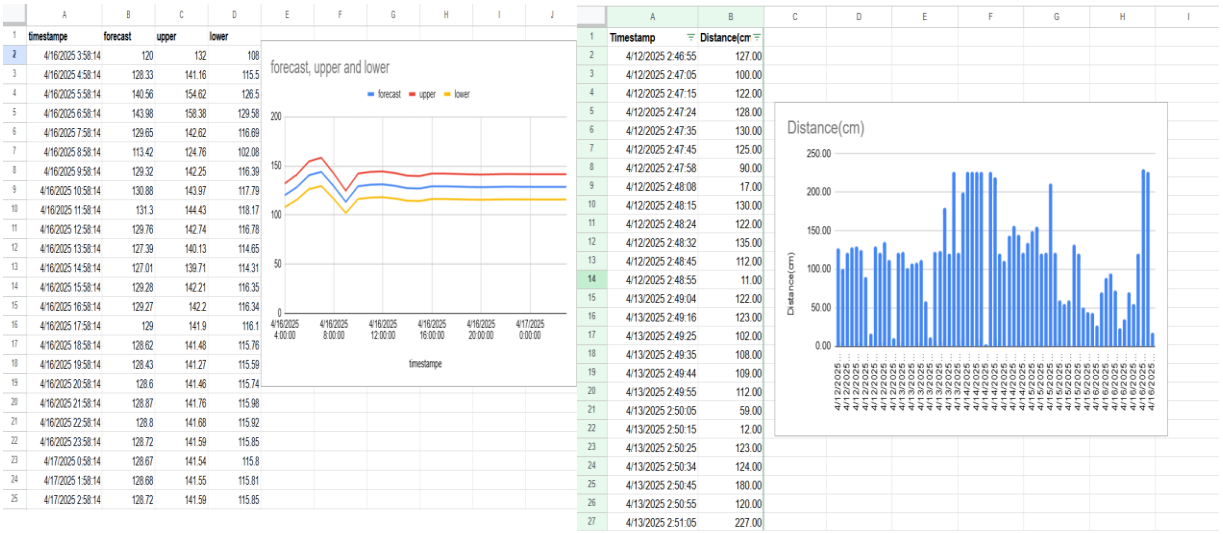
5.2 Justification for Choice of Algorithms

Algorithm	Reason for use
EMA (Exponential Moving Average)	Fast, simple and efficient at identifying short-term patterns and noise reduction

Table 1: *EMA Data for Upper And Lower*

	A	B	C	D
1	**timestampe**	**forecast**	**upper**	**lower**
2	4/16/2025 3:58:14	120	132	108
3	4/16/2025 4:58:14	128.33	141.16	115.5
4	4/16/2025 5:58:14	140.56	154.62	126.5
5	4/16/2025 6:58:14	143.98	158.38	129.58
6	4/16/2025 7:58:14	129.65	142.62	116.69
7	4/16/2025 8:58:14	113.42	124.76	102.08
8	4/16/2025 9:58:14	129.32	142.25	116.39
9	4/16/2025 10:58:14	130.88	143.97	117.79
10	4/16/2025 11:58:14	131.3	144.43	118.17
11	4/16/2025 12:58:14	129.76	142.74	116.78
12	4/16/2025 13:58:14	127.39	140.13	114.65
13	4/16/2025 14:58:14	127.01	139.71	114.31
14	4/16/2025 15:58:14	129.28	142.21	116.35
15	4/16/2025 16:58:14	129.27	142.2	116.34
16	4/16/2025 17:58:14	129	141.9	116.1
17	4/16/2025 18:58:14	128.62	141.48	115.76
18	4/16/2025 19:58:14	128.43	141.27	115.59
19	4/16/2025 20:58:14	128.6	141.46	115.74
20	4/16/2025 21:58:14	128.87	141.76	115.98
21	4/16/2025 22:58:14	128.8	141.68	115.92
22	4/16/2025 23:58:14	128.72	141.59	115.85
23	4/17/2025 0:58:14	128.67	141.54	115.8
24	4/17/2025 1:58:14	128.68	141.55	115.81
25	4/17/2025 2:58:14	128.72	141.59	115.85

6. Charting & Visualization



The chart is generated dynamically within the `Forecasts and Real-Time` sheet. Line charts show:

- Actual values
- Upper/lower bounds for future estimates

7. Implementation challenges and solution

- **Real-Time Data Latency**
  Challenge: Delays in logging or updating live data affected real-time monitoring.
  Solution: Implemented buffered logging and minimized processing delay on the client side using asynchronous operations.
- **Data Storage Limitations**
  Challenge: Google Sheets has a limited number of rows/cells which can get exhausted over time.
  Solution: Added automatic data archiving and row deletion after a threshold is reached.

8. Challenging

- **Sensor Data Missing**
  - Sometimes no value was recorded due to sensor glitch.
  - Fixed by adding checks and using last known value as backup

- **Internet Connection Drop**
  - Data upload failed when Wi-Fi was unstable.
  - Added retry system to resend when connection returns

- **Sheet Data Overload**
  - Too much data slowed down Google Sheets.
  - Limited sheet size and added auto-delete for old rows.

- **Mobile Display Issues**
  - Layout was messy on smaller screens.
  - Improved UI using responsive columns and simpler components.

- **Slow Dashboard Load Time**
  - Too much info caused Android Studio to lag.
  - Reduced chart size and used tabs to split views.

9. Future improvements

- Integrate email/PDF export of summaries
- Add ML model comparison (e.g., LSTM)
- Schedule daily summary generation
- Add authentication/token rotation for secure access

10. Conclusion

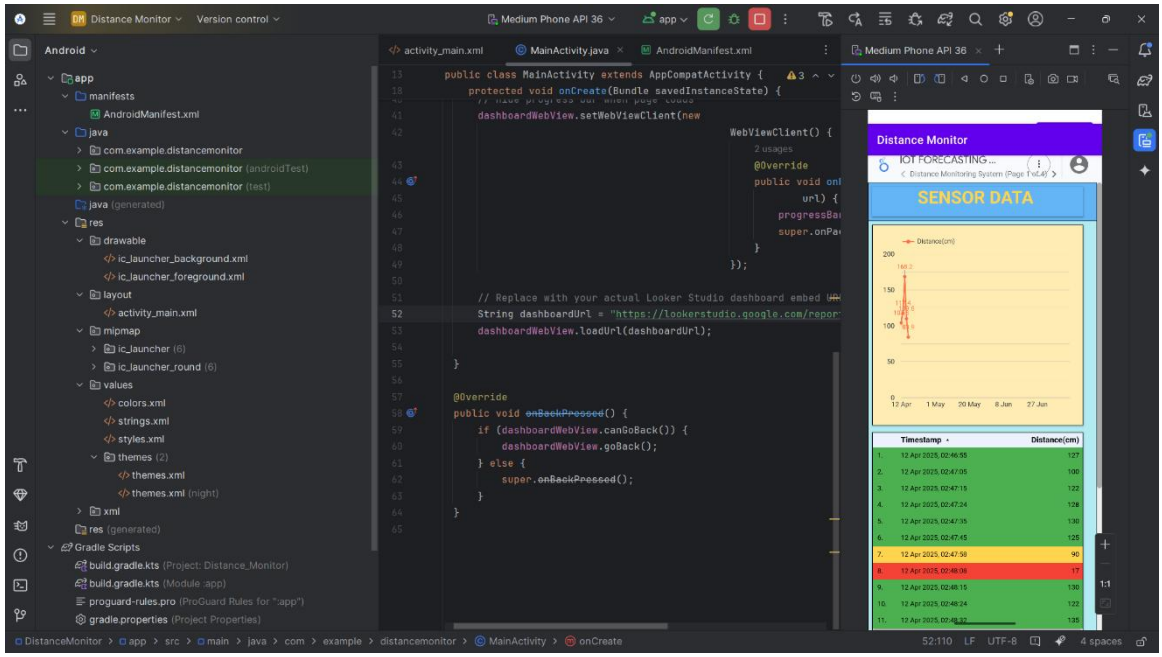
This project successfully demonstrates the integration of real-time sensor monitoring, data forecasting, and intelligent summarization using lightweight algorithms like EMA and SMA. Through the use of APIs and visualization tools, it provides users with meaningful insights into sensor behavior while addressing common challenges such as data accuracy, API limitations, and display clarity.

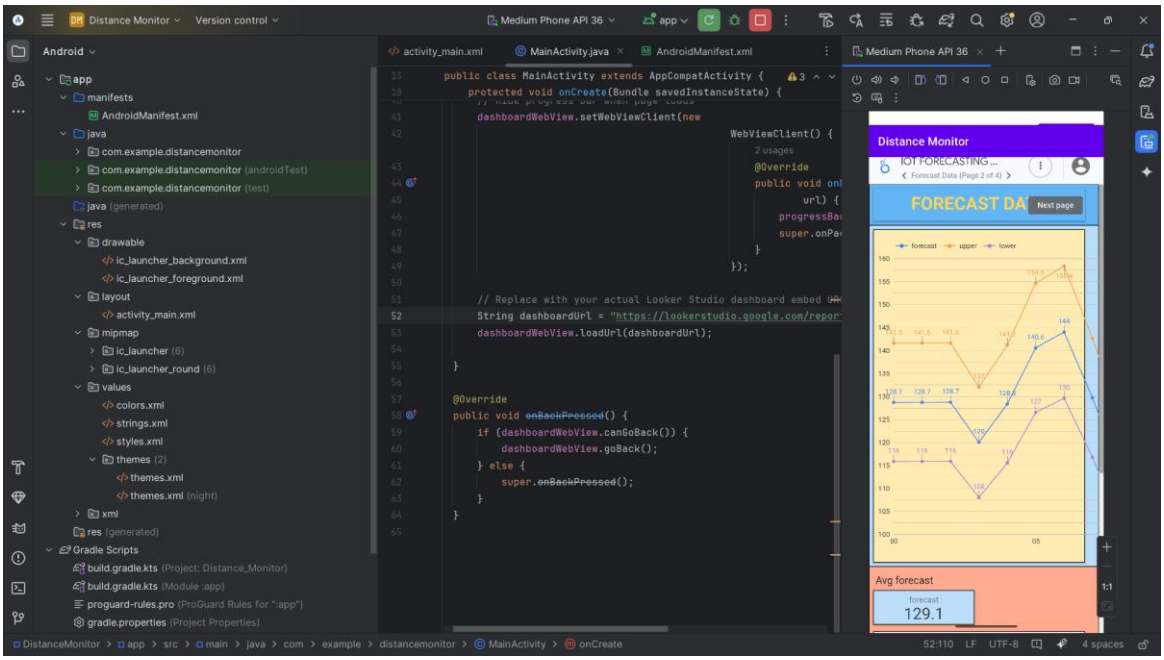
Despite various implementation hurdles, effective solutions were applied to improve system performance, accuracy, and user experience. By combining data science principles with practical engineering, the system lays a strong foundation for further development.

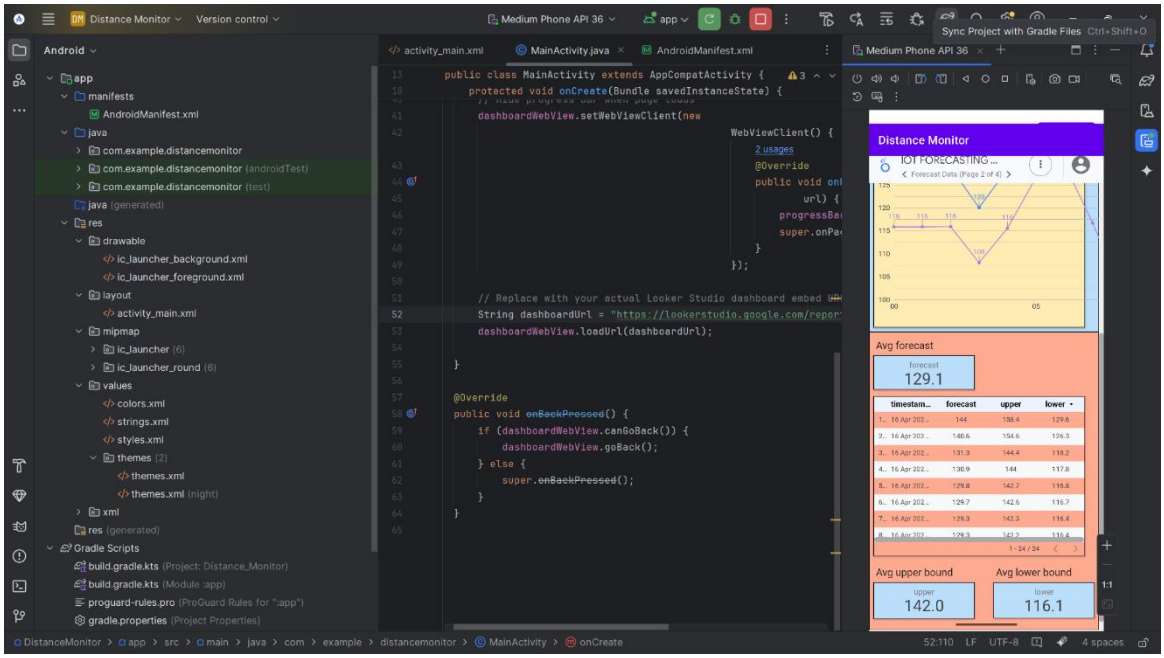
Looking ahead, enhancements such as advanced machine learning models, automated reporting, and security features will make the system even more robust, intelligent, and user-friendly. This project not only meets its current objectives but also opens the door for scalable, real-world applications in smart monitoring and forecasting systems.

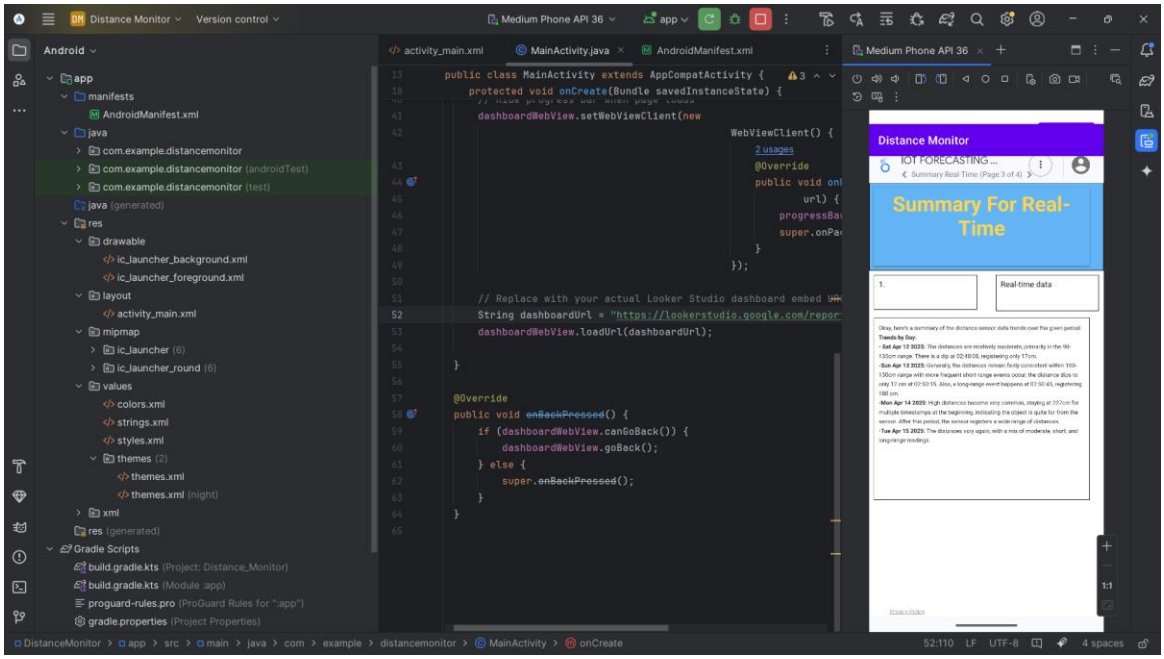
11. References

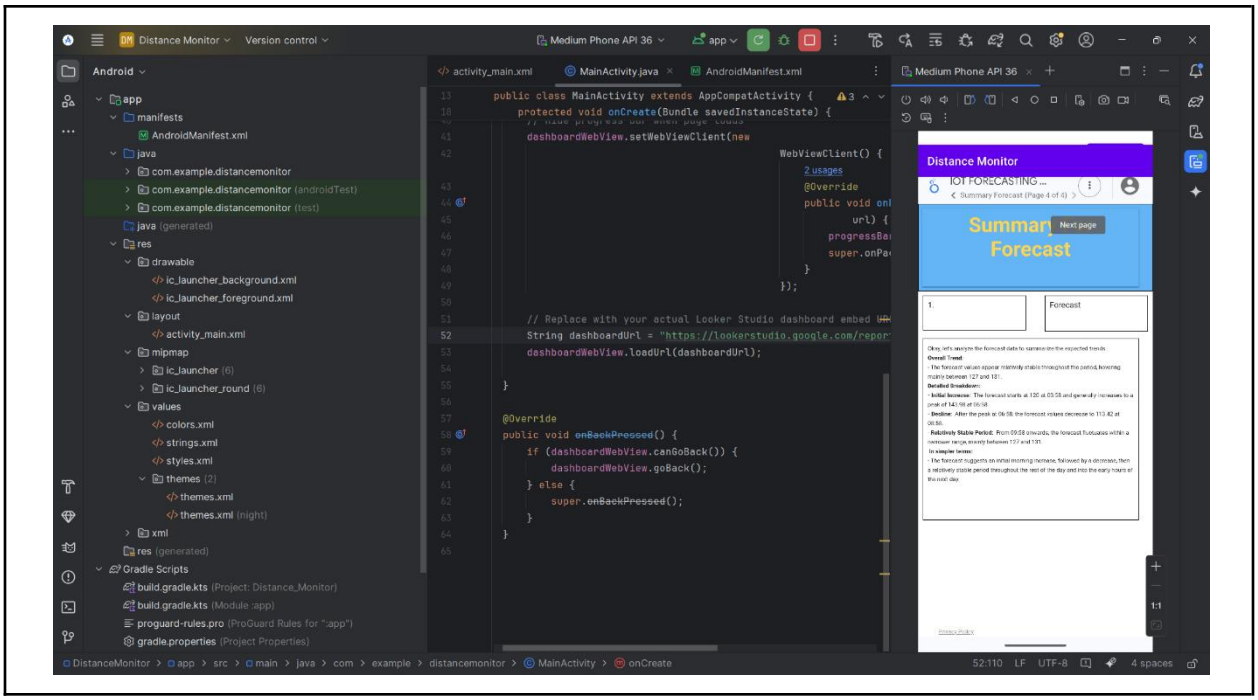
Images of Android Studio App






Distance Monitor
Version control
Medium Phone API 36
app
Sync Project with Gradle Files Ctrl+Shift+O
Android
activity_main.xml
MainActivity.java
AndroidManifest.xml
app
manifests
AndroidManifest.xml
java
com.example.distancemonitor
com.example.distancemonitor (androidTest)
com.example.distancemonitor (test)
java (generated)
res
drawable
ic_launcher_background.xml
ic_launcher_foreground.xml
layout
activity_main.xml
mipmap
ic_launcher (6)
ic_launcher_round (6)
values
colors.xml
strings.xml
styles.xml
themes (2)
themes.xml
themes.xml (night)
xml
res (generated)
Gradle Scripts
build.gradle.kts (Project: Distance_Monitor)
build.gradle.kts (Module :app)
proguard-rules.pro (ProGuard Rules for ":app")
gradle.properties (Project Properties)
public class MainActivity extends AppCompatActivity {
protected void onCreate(Bundle savedInstanceState) {
dashboardWebView.setWebViewClient(new
WebViewClient() {
2 usages
@Override
public void onl
url) {
progressBa
super.onPa
});
// Replace with your actual Looker Studio dashboard embed
String dashboardUrl = "https://lookerstudio.google.com/repor
dashboardWebView.loadUrl(dashboardUrl);
@Override
public void onBackPressed() {
if (dashboardWebView.canGoBack()) {
dashboardWebView.goBack();
} else {
super.onBackPressed();
Distance Monitor
IOT FORECASTING ...
Forecast Data (Page 2 of 4)
Avg forecast
forecast
129.1
timestam...
forecast
upper
lower
1 - 24 / 24
Avg upper bound
upper
142.0
Avg lower bound
lower
116.1
DistanceMonitor
app
src
main
java
com
example
distancemonitor
MainActivity
onCreate
52:110
LF
UTF-8
4 spaces


Distance Monitor
Version control
Medium Phone API 36
app
Android
activity_main.xml
MainActivity.java
AndroidManifest.xml
app
manifests
AndroidManifest.xml
java
com.example.distancemonitor
com.example.distancemonitor (androidTest)
com.example.distancemonitor (test)
java (generated)
res
drawable
ic_launcher_background.xml
ic_launcher_foreground.xml
layout
activity_main.xml
mipmap
ic_launcher (6)
ic_launcher_round (6)
values
colors.xml
strings.xml
styles.xml
themes (2)
themes.xml
themes.xml (night)
xml
res (generated)
Gradle Scripts
build.gradle.kts (Project: Distance_Monitor)
build.gradle.kts (Module :app)
proguard-rules.pro (ProGuard Rules for ":app")
gradle.properties (Project Properties)
public class MainActivity extends AppCompatActivity {
protected void onCreate(Bundle savedInstanceState) {
dashboardWebView.setWebViewClient(new
WebViewClient() {
2 usages
@Override
public void onl
url) {
progressBa
super.onPa
});
// Replace with your actual Looker Studio dashboard embed
String dashboardUrl = "https://lookerstudio.google.com/repor
dashboardWebView.loadUrl(dashboardUrl);
@Override
public void onBackPressed() {
if (dashboardWebView.canGoBack()) {
dashboardWebView.goBack();
} else {
super.onBackPressed();
Distance Monitor
IOT FORECASTING ...
Summary Real-Time (Page 3 of 4)
Summary For Real-Time
1.
Real-time data
DistanceMonitor
app
src
main
java
com
example
distancemonitor
MainActivity
onCreate
52:110
LF
UTF-8
4 spaces



Android Studio provides an integrated environment where you can write code, design layouts, and run your app on an emulator to test and debug in real time.

Images of Android App

Link of Google Sheet

ESP32 Sensor Data Longer

Link of Looker Studio Dashboard

Looker Studio Dashboard

Full code snippet (with commented):

ESP32-Light-Monitoring/Google_apps_script