‏13/6/2025

Developer Guide Package

Team Snake

**Index**

1. **Introduction  
   • Overview  
   • Purpose**
2. **Core Files Overview  
   • team\_snake\_dashboard\_main.ipynb  
   • mqtt\_sensor\_outdoor.ipynb  
   • mqtt\_sensor\_indoor.ipynb**
3. **Installation & Libraries Used**

**• Required Installations  
• Python Libraries Used**

1. **various design patterns and code templates**

**• Template-Based Layout Pattern (HTML + CSS)**

**• Use of ipywidgets**

**• Observer Pattern (Reactive UI with ipywidgets Observers)**

**• Composite Pattern (Flexible Hierarchy for Dashboard Panels)**

1. **Microservices**

**• IndexService  
• QueryService  
• ResultService**

1. **Firebase – Data Insertion Functions**

**•** **upload\_sensor\_thresholds**()  
**•** **upload\_sensor\_scores**()  
**•** **upload\_failure\_paths ()**  
**•** **upload\_engineer\_performance ()**  
**•** **pload\_daily\_teams\_tasks ()**  
**• upload\_fake\_indoor\_data\_hour ()**  
**• upload\_fake\_indoor\_data\_week ()**

**•** **upload\_fake\_outdoor\_data\_hour ()**  
**• upload\_fake\_outdoor\_data\_week ()**

1. **Firebase – Data Retrieval Functions**

**•** **fetch\_last\_hour\_data ()**  
**•** **fetch\_historical\_data ()**  
**•** **get\_thresholds ()**  
**•** **get\_reliability\_scores ()**  
**•** **get\_failure\_paths ()**  
**• get\_meetings ()**  
**• fetch\_engineer\_data ()**

**• fetch\_team\_names ()**

1. **URLs – External Content Sources for Search Engine**

**• Overview**  
**• Purpose**

**• Indexed URLs**  
**• Features Enabled**

1. **Key Functions**  
   **•** **create\_DashBoard\_page()**  
   **•** **create\_search\_page()**  
   **•** **create\_Rates\_page()**  
   **•** **create\_team\_overview\_page()**  
   **•** **create\_statistics\_page()**  
   **• chatbot\_page()**  
   **• create\_Meetings\_page()**
2. **Firebase Structure  
   • /DailyTeamsTasks  
   • /EngineerPerformance  
   • /FailurePaths  
   • /Index  
   • /LiveSensors  
   • /LiveSensorsOUTDOOR  
   • /Meetings  
   • /SensorScores  
   • /SensorThresholds**

***Introduction***

**Overview  
This project presents a system for real time monitoring and engineer task management based on data from indoor and outdoor sensors. The system includes an interactive dashboard interface, performance tracking, daily task monitoring, and continuous data insertion into a Firebase database. It also features automatic score computation for each engineer, anomaly detection based on sensor thresholds, and a chatbot interface for user assistance. The project integrates visualization tools, team management modules, and historical data analysis functions and also includes a search engine along with many additional features . Together, these components provide a centralized and intuitive environment for both operational control and simulation, designed to be scalable and adaptable to future industrial needs.**

**Purpose  
The goal of the project is to develop a central, interactive, and accessible tool that allows effective supervision of sensor activity, management of engineer tasks, and continuous assessment of system status. The tool serves as a development and simulation environment, while its flexible structure is also prepared for future expansion into an operational real time system.**

***Core Project Files***

**my\_new\_proj\_anan\_nahla.ipynb:  
This file represents the complete application and serves as the central script of the entire project. It brings together all components into one cohesive system, managing both the logic and the user interface. All interactions, data operations, and visual elements are handled within this file, making it the main entry point for running and demonstrating the full functionality of the project in one place. Its structure allows for clear user interaction and system control through a single, unified environment.**

**mqtt\_sensor\_indoor.ipynb**

**This Python script functions as a dedicated MQTT client to receive real-time data from indoor sensors.  
It runs separately from the main dashboard notebook because it uses an infinite loop (loop\_forever()), which cannot be embedded inside a colab interface. By operating independently, it allows simultaneous and continuous data streaming without blocking the user interface.**

**Responsibilities of this script:**

**Connects to the public MQTT broker test.mosquitto.org and subscribes to the topic braude/D106/indoor.**

**Parses each incoming message in JSON format.**

**Extracts key indoor sensor values:  
Temperature, Humidity, Pressure, and Distance.**

**Adds a current timestamp (in Israel timezone) to each record.**

**Uploads the data to Firebase under the path /LiveSensors, using the timestamp as a unique ID.**

**Prints status updates to the console and handles errors gracefully during transmission.**

**mqtt\_sensor\_outdoor.ipynb**

**This Python file serves as a dedicated MQTT client for receiving real-time data from outdoor sensors.  
It operates independently from the main dashboard notebook, as it uses an infinite loop (loop\_forever()), which cannot run within the same notebook. This separate execution allows the system to continuously collect live data without interrupting the graphical interface.**

**Responsibilities of the script:**

**Connects to the MQTT broker (test.mosquitto.org) and subscribes to the topic braude/D106/outdoor.**

**Parses each incoming JSON message received from the sensors.**

**Extracts key sensor values: Temperature, Humidity, and Light Intensity (DLIGHT).**

**Formats the data with a timestamp and uploads it to Firebase under the path /LiveSensorsOUTDOOR.**

***Libraries and Installations Overview***

**Required Installations**

!pip install requests beautifulsoup4

 → For sending HTTP requests and scraping external web pages for content indexing.

!pip install requests beautifulsoup4 nltk

 → Adds NLTK (Natural Language Toolkit) for basic text processing and language tasks.

!pip install firebase

 → Enables interaction with Firebase Realtime Database using the firebase module.

!pip install ipywidgets --quiet

 → Installs ipywidgets for building interactive UI elements like dropdowns and buttons in notebooks.

!pip install nltk

 → Ensures full installation of Natural Language Toolkit for search and AI support .

!pip install seaborn

 → Visualization library based on matplotlib, used for creating attractive statistical plots.

!pip install pytz

 → Adds time zone support, required for accurate date/time handling in meetings and trends.

!pip install ipywidgets

 → Repeated to ensure full widget functionality in COLAB environments.

!pip install plotly

 → Interactive charting library for time series, sensor graphs, and comparative analysis.

!pip install pandas matplotlib

 → Core libraries for data manipulation (pandas) and plotting (matplotlib).

!pip install paho-mqtt

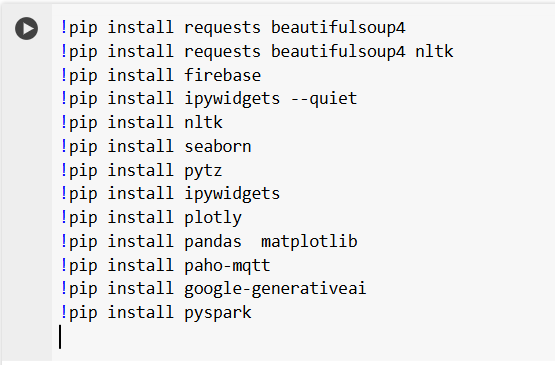
 → Installs the MQTT client library used for real-time sensor communication.

!pip install google-generativeai

 → Enables integration with Gemini AI (Google Generative AI) for chatbot interactions.

!pip install pyspark  
 → Installs PySpark, enabling distributed data processing and big data analytics within the project, useful for handling large datasets efficiently.

Code example:



**Python Libraries Used**

import pandas as pd

 → For handling tabular data, sensor logs, and performance metrics.

import numpy as np

 → For numerical operations, statistical calculations, and array handling.

import datetime

 → For working with dates and times, especially in scheduling and time filtering.

from matplotlib import pyplot as plt

 → For creating charts such as line graphs, bar charts, and time trends.

import seaborn as sns

 → High-level interface for drawing attractive statistical plots.

import plotly.graph\_objects as go

 → For building interactive charts like donut plots, boxplots, and comparison graphs.

import ipywidgets as widgets

 → Used to create interactive GUI elements like dropdowns, buttons, sliders, etc.

from firebase import firebase

 → For reading from and writing to Firebase Realtime Database.

import requests

 → For sending HTTP requests to fetch or post external content (used in indexing, external sources).

import json

 → To parse and manipulate JSON data from Firebase or APIs.

import re

 → Regular expressions for validating input formats (e.g., time in HH:MM format).

import pytz

 → Time zone support for accurate time and date calculations in meetings and logs.

import nltk

 → Toolkit for text processing, tokenization, and enabling AI text analysis.

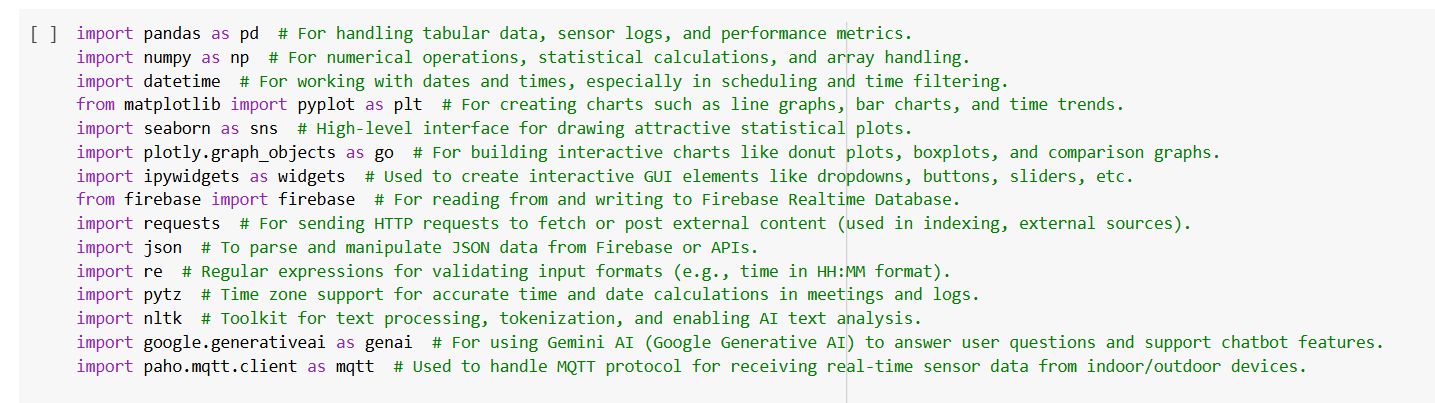
import google.generativeai as genai

 → For using Gemini AI (Google Generative AI) to answer user questions and support chatbot features.

import paho.mqtt.client as mqtt

 → Used to handle MQTT protocol for receiving real-time sensor data from indoor/outdoor devices.

Code example:



***various design patterns and code templates***

**Template-Based Layout Pattern (HTML + CSS)**

**Description:**  
This pattern involves separating the visual layout (HTML structure) and style (CSS) from the logic of the Python code. By embedding predefined HTML and CSS templates in widgets (especially using widgets.HTML), the application ensures a consistent and customizable user interface. It allows for reusability of design elements such as menus, headers, and containers, without duplicating style code.

**Example:**



**Use of ipywidgets** **-Use of ipywidgets for Interactive UI Components**

**Description:  
The project makes extensive use of ipywidgets to build interactive dashboards and user interfaces in Python. This pattern involves using pre-built widget classes such as Dropdown, Button, VBox, HBox, and Output to create reactive elements. These widgets respond to user input in real-time, enabling dynamic updates of charts, data displays, and system simulations without reloading the interface.**

**Example:**

****

**Observer Pattern (Reactive UI with ipywidgets Observers)**

Description:  
The observer pattern is implemented using ipywidgets’ observe() method. This allows components (like dropdowns or sliders) to react to user input in real time, updating charts or views automatically. It decouples the data source from the UI reaction, enabling modular and dynamic behavior.

**Example:**

****

**Composite Pattern (Flexible Hierarchy for Dashboard Panels)**

Description:  
The composite pattern is used to build complex UIs from simple, reusable components. Each dashboard is composed of nested widgets (e.g., VBox, HBox, Output, HTML), allowing you to treat both individual elements and groups of elements uniformly. This flexible hierarchy makes it easy to rearrange, update, or reuse UI sections like headers, panels, and graphs.

Example:

**תמונה שמכילה טקסט, צילום מסך, גופן

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.**

***Microservices***

**In our project, three microservices were adopted, each responsible for a specific domain, ensuring a clear separation of data processing stages:**

1. **Document Management & Indexing (IndexService)  
   This service is responsible for receiving documents, storing them, and building an inverted index. Every word found in a document is stored with information about which documents it appears in and how many times. The purpose of the service is to enable fast and accurate document searches.**
2. **Query Creation (QueryService)  
   This service receives search terms from the user and queries the built index. It identifies the relevant documents for each term, computes their intersection (if multiple terms are provided), and records the frequency of each term in each found document.**
3. **Result Processing & Presentation (ResultService)  
   This service is responsible for processing and organizing the search results. It adds additional information to each result, such as the number of occurrences of terms, occurrences of full phrases, and a direct link to the relevant document.**

**This modular approach simplifies maintenance, supports future scalability, and aligns with the principle of separation of concerns.**

1. IndexService – Document Management & Inverted Indexing

****

1. QueryService – Search Query Handling

תמונה שמכילה טקסט, צילום מסך, גופן, מסמך

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

1. ResultService – Formatting and Presenting Results

תמונה שמכילה טקסט, צילום מסך, מסמך, גופן

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

***Firebase - Data Insertion Functions: Full Function Descriptions***

**Function: upload\_sensor\_thresholds**

**Purpose  
Uploads threshold values (lower and upper bounds) for each sensor type to the /SensorThresholds path in the Firebase database.**

**Functionality**

* **Defines a dictionary with low and high threshold values for each sensor (Temperature, Humidity, Pressure, Distance, etc.).**
* **Uses FBconn.put to upload the data to /SensorThresholds.**
* **Prints a success message after the upload.**

**Function: upload\_sensor\_scores**

**Purpose  
Uploads performance scores for sensors (Accuracy, Stability, Response Time) to the /SensorScores path in Firebase.**

**Functionality**

* **Defines a dictionary where each sensor has three metrics: Accuracy, Stability, and Response Time.**
* **Uses FBconn.put to upload to /SensorScores.**
* **Prints a success message showing the number of sensors uploaded.**

**Function: upload\_failure\_paths**

**Purpose  
Uploads failure paths and possible reasons for anomalies in various metrics to the /FailurePaths path.**

**Functionality**

* **Defines a list of possible causes and their probabilities for each type of anomaly (e.g., “High Temperature”).**
* **Uses FBconn.put to upload the data to Firebase.**
* **Prints a success message if the upload is completed successfully.**

**Function: upload\_engineer\_performance**

**Purpose  
Uploads daily performance data for engineers to the /EngineerPerformance path.**

**Functionality**

* **Defines a list of engineers, each with two daily tasks including progress and score.**
* **Uploads the list to Firebase using FBconn.put.**
* **Prints a success message.**

**Function: upload\_daily\_teams\_tasks**

**Purpose  
Uploads tasks for various teams to the /DailyTeamsTasks path in the database.**

**Functionality**

* **Defines a list of teams, each with a name, a general task, and a detailed task list with completion status .**
* **Uploads the list to Firebase using FBconn.put.**
* **Prints a success or error message.**

**Function: upload\_fake\_indoor\_data\_hour**

**Purpose  
Uploads one hour’s worth of fake indoor sensor data (every 5 minutes) to the /LiveSensors path.**

**Functionality**

* **Calculates a one-hour range from the current time.**
* **Generates a sensor record every 5 minutes, with a chance of an error.**
* **Uploads the data under keys like indoor\_YYYYMMDDHHMMSS in /LiveSensors.**

**Function: upload\_fake\_indoor\_data\_week**

**Purpose  
Uploads one week’s worth of fake indoor sensor data (every 30 minutes) to the /LiveSensors path.**

**Functionality**

* **Calculates the most recent Friday and goes back one week.**
* **For each day: creates 20 records every 30 minutes, with a 25% chance of an error.**
* **Uploads the data under keys like indoor\_YYYYMMDDHHMMSS in /LiveSensors.**

**Function: upload\_fake\_outdoor\_data\_hour**

**Purpose  
Uploads one hour’s worth of outdoor sensor data (every 5 minutes), including Temperature, Humidity, and DLIGHT, to /LiveSensorsOUTDOOR.**

**Functionality**

* **Generates random values, with or without anomalies.**
* **Uploads data under keys like outdoor\_YYYYMMDDHHMMSS.**
* **Repeats the process across the hour range.**

**Function: upload\_fake\_outdoor\_data\_week**

**Purpose  
Uploads one full week of outdoor sensor data (every 30 minutes × 7 days) to the /LiveSensorsOUTDOOR path.**

**Functionality**

* **Calculates the most recent Friday and goes back one week.**
* **For every half-hour slot in the week: generates valid or anomalous data values.**
* **Uploads the data under keys like outdoor\_YYYYMMDDHHMMSS.**

***Firebase – Data Retrieval Functions: Overview and Usage***

**Function: fetch\_last\_hour\_data**

**Purpose  
Fetches indoor and outdoor sensor data from the last hour from /LiveSensors and /LiveSensorsOUTDOOR.**

**Functionality**

* **Calculates a one-hour time window from the current time.**
* **Retrieves data from Firebase and converts the 'Time' column to datetime.**
* **Filters rows that fall within the selected hour.**
* **Returns two DataFrames: one for indoor, one for outdoor sensors.**

**Function: fetch\_historical\_data**

**Purpose  
Fetches historical sensor data (indoor and outdoor) for analysis.**

**Functionality**

* **Retrieves data from /LiveSensors and /LiveSensorsOUTDOOR.**
* **Converts 'Time' to datetime format and adds a weekday column 'Day'.**
* **Merges indoor and outdoor data, sorts by time, and returns a combined DataFrame.**

**Function: get\_thresholds**

**Purpose  
Fetches sensor threshold values from the /SensorThresholds path.**

**Functionality**

* **Returns a dictionary of min/max thresholds for each sensor type.**
* **Returns an empty dictionary if no data found.**

**Function: get\_reliability\_scores**

**Purpose  
Fetches sensor reliability scores from the /SensorScores path.**

**Functionality**

* **Retrieves accuracy, stability, and response time scores for each sensor.**
* **Returns an empty dictionary if no data found.**

**Function: get\_failure\_paths**

**Purpose  
Fetches predefined failure reasons and probabilities from the /FailurePaths path.**

**Functionality**

* **Returns a dictionary where each failure type maps to its root causes and likelihoods.**
* **Used for root cause analysis when anomalies are detected.**

**Function: get\_meetings**

**Purpose  
Fetches all scheduled meetings from the /Meetings path in Firebase.**

**Functionality**

* **Retrieves all meeting records as a dictionary or list.**
* **Returns the raw structure for further UI display or filtering.**

**Function: fetch\_engineer\_data**

**Purpose  
Fetches engineer performance and daily tasks from the /EngineerPerformance path.**

**Functionality**

* **Returns a list of engineers and their performance data.**
* **Handles both dict and list formats to ensure consistency.**

**Function: fetch\_team\_names**

**Purpose  
Fetches the names of all teams from the /DailyTeamsTasks path.**

**Functionality**

* **Returns a list of team names for dropdown menus.**

***URLs – External Content Sources for the Search Engine***

**Overview**:  
This section presents the list of external web pages that were crawled and indexed as part of our search engine. The content of these URLs was processed using Python tools such as BeautifulSoup, requests, and nltk, and stored in Firebase under a structured inverted index format.

**Purpose**:  
To simulate a realistic search experience based on real world technical content, we selected authoritative pages related to MQTT protocol. These were parsed, cleaned, stemmed, and indexed to enable full-text search with query-based matching and relevance analysis.

**Indexed URLs**:

* <https://mqtt.org/>
* <https://mqtt.org/getting-started/>
* <https://mqtt.org/mqtt-specification/>
* <https://mqtt.org/software/>
* <https://mqtt.org/use-cases/>
* <https://mqtt.org/faq/>
* <https://mqtt.org/legal/>

**Features Enabled**:

* Building of an inverted index (word → document IDs, frequency).
* Full phrase and keyword based search.
* Integration with QueryService for creating customized search queries.
* Integration with ResultService for formatting and presenting search results clearly.
* Firebase storage of index and query data for persistence.

***Dashboard Page – Function Overview and Responsibilities***

**Function:** create\_DashBoard\_page **Purpose**This function builds the main dashboard page of the system for the manager, including a personal greeting, live metrics from Firebase, alerts, recommendations, a weekly fault summary, and daily task management for teams and engineers.

**Functionality**

1. Displays a loading spinner while data is being fetched.
2. Defines the last-hour time range and connects to Firebase.
3. Loads Indoor and Outdoor sensor data using fetch\_last\_hour\_data.
4. Displays the manager greeting card.
5. Builds the Indoor and Outdoor metrics sections.
6. Shows a weekly system faults summary with a button to view the full report.
7. Displays a recommendations card based on station performance.
8. Shows a critical alerts card based on sensor anomalies.
9. Presents daily task management for teams, including:  
   • Add team  
   • Add subtasks  
   • Update existing tasks
10. Displays engineer management tools, including:  
    • Add engineer  
    • Remove engineer  
    • Assign new task to engineer
11. Shows a daily technical summary.
12. Returns all components in a single VBox for the dashboard page.

**Function:** create\_manager\_greeting\_section(df, df\_outdoor)

**Purpose**  
Displays a greeting card and overall system status report for the manager, including system status, number of sensors, teams, and engineers.

**Functionality**

1. Calculates daily averages for each sensor (indoor and outdoor).
2. Checks whether all values are within the safe range – determines status: Normal / Warning / Critical.
3. Retrieves data from Firebase: number of teams and engineers.
4. Builds a styled HTML box displaying:
   * Overall system status with color and icon (green, orange, red).
   * Total number of sensors.
   * Number of active teams.
   * Number of engineers.

**Returns**  
A widgets.HTML component displaying the manager greeting and summary report.

**Function:** create\_metrics\_section\_INDOOR  
**Purpose**  
This function displays statistics for indoor station metrics, including averages, the number of threshold violations, and an interactive fault table.

**Functionality**

1. Displays a styled header for the section "Indoor Sensor Summary".
2. Calculates averages for temperature, humidity, air pressure, and distance.
3. Checks for anomalies in each metric based on the following conditions:
   * Temperature outside the range 22–30
   * Humidity outside the range 30–60
   * Air pressure outside the range 950–1050
   * Distance less than 100
4. Creates a new DataFrame faults\_df with all rows where anomalies were detected.
5. Builds a button showing the number of existing faults and enables displaying a styled table upon click.
6. Defines the on\_click function that displays a table highlighting cells where a problem was found.
7. Builds five graphical cards displaying:
   * Average for each metric (temperature, humidity, pressure, distance)
   * Total number of anomalies
8. Returns the display as a VBox containing the header, cards, button, and the fault table output area.

**Function:** create\_metrics\_section\_OUTDOOR  
**Purpose**  
This function presents a statistical overview of the outdoor environmental sensors, including metric averages, the number of alerts for anomalies, and an interactive fault detail table.

**Functionality**

1. Displays a styled external header for the "Outdoor Sensor Summary" section.
2. Calculates averages for three key metrics: temperature, humidity, and DLIGHT.
3. Checks for anomalies in the metrics based on defined conditions:
   * Temperature outside the range 22–30
   * Humidity outside the range 30–60
   * DLIGHT above 15000
4. Creates faults\_df – a sub-table with all rows where anomalies were detected.
5. Builds a button displaying the number of outdoor faults and allows showing a detailed table.
6. Defines an on\_click function that displays a styled table highlighting the problematic fields.
7. Builds four summary cards with average values for each metric, including a card showing the total number of anomalies.
8. Returns all components as a VBox, including the header, cards, display button, and a hidden fault table that expands on demand.

**Function:** create\_station\_recommendations\_section  
**Purpose**  
This function generates a smart recommendations area for the system manager, based on anomalies in physical measurements detected at Indoor and Outdoor stations, to identify issues and improve station conditions.

**Functionality**

1. Defines normal range values for each metric: temperature, humidity, air pressure, distance, and DLIGHT.
2. Iterates over each data row in the Indoor stations:
   * Checks whether any metric is outside the normal range.
   * If so, adds a warning message to the list of recommendations.
3. Performs a similar process for the Outdoor stations, including an additional check for DLIGHT values.
4. Removes duplicate messages using a seen list.
5. If no anomalies are found – displays a green message indicating all is normal.
6. If recommendations are found – builds a styled display (HTML card) showing all suggestions, color-coded by alert type.
7. Returns a widgets.HTML component for display on the dashboard.

**Function:** create\_critical\_alerts\_section  
**Purpose**  
This function displays all critical alerts detected in the indoor and outdoor environmental metrics, including classification by sensor type and data source, with the ability to filter the display by category.

**Functionality**

1. Converts the Time column to datetime format for accurate display of when the anomaly occurred.
2. Detects metric anomalies based on defined conditions:  
   • Temperature outside the 22–30°C range  
   • Humidity outside the 30–60% range  
   • Air pressure outside the 950–1050 hPa range (Indoor only)  
   • Distance less than 100 cm (Indoor only)  
   • DLIGHT greater than 15,000 (Outdoor only)
3. Builds a list of alerts including metric type, timestamp, and environment (Indoor/Outdoor).
4. Removes duplicates to ensure only one alert per metric type and source is shown.
5. Groups alerts by metric type and environment, displaying them in separate tabs.
6. Creates an interactive HTML interface with:  
   • "All" tab to show all alerts  
   • Separate tabs for each metric type (Temperature, Humidity, DLIGHT, etc.)
7. Defines a JavaScript function to switch tab content on click.  
   Returns an HTML component with the complete alert interface.

**Function: create\_engineer\_tasks**

**Purpose**  
This function creates an interactive interface for managing engineers — including adding, removing, task assignment, and progress updates — using Firebase to store and sync data.

**Functionality**

1. Defines custom CSS styling for input fields, buttons, and components.
2. Displays management forms including:  
   • Add engineer (on\_add\_engineer)  
   • Remove engineer (on\_remove\_engineer)  
   • Assign task (on\_assign\_clicked)  
   • Update progress (create\_progress\_update\_box)
3. Loads the list of engineers and refreshes it after every change.
4. Returns all UI components together as a VBox.

**Sub-function: on\_add\_engineer**

**Purpose**  
To add a new engineer with an initial task and a predefined score.

**Functionality**

1. Retrieves values from fields: engineer name, task description, score.
2. Checks whether the name already exists in the database.
3. If not:  
   • Creates a new engineer with one task.  
   • Updates /EngineerPerformance in Firebase.
4. Displays a success/error message and clears the input fields.

**Sub-function: on\_remove\_engineer**

**Purpose**  
To allow the removal of an existing engineer from the system.

**Functionality**

1. The user selects an engineer from a dropdown.
2. The function looks for the engineer by name in the database.
3. If found deletes the engineer from /EngineerPerformance.
4. Refreshes the engineer list and shows a relevant message.

**Sub-function: on\_assign\_clicked**

**Purpose**  
To assign a new task to an existing engineer, including a description and score.

**Functionality**

1. Retrieves the task description and score from input fields.
2. Validates that an engineer is selected and the description is not empty.
3. Adds the task to the engineer’s tasks\_today list.
4. Updates the Firebase database.
5. Provides user feedback for success or failure.

**Function: create\_progress\_update\_box**

**Purpose**  
This function creates an interactive UI for updating the progress percentage of engineer tasks using dropdowns and a text input. The updates are stored in Firebase under /EngineerPerformance.

**Functionality**

1. Loads the list of engineers from the Firebase database.
2. Allows the user to select an engineer from a dropdown menu.
3. Once an engineer is selected — loads their daily tasks into another dropdown.
4. Enables the user to enter a new progress percentage for the selected task (e.g., 75% or 75).
5. Includes a styled “Update Progress” button to perform the update.
6. On button click:  
   • Validates that the input is a number between 0 and 100.  
   • If valid updates the progress field of the task in Firebase.  
   • Displays a success or error message accordingly.
7. Returns a VBox containing all UI components: dropdowns, input field, button, and a message output area.

**Function: create\_daily\_tasks\_section**

**Purpose**  
This function enables daily task management **for teams**. The user can add new teams, add sub-tasks to each team, and update the completion status of each sub-task.

**Functionality**

1. Loads data from Firebase under the path /DailyTeamsTasks.
2. Checks the type of the returned data (list or dictionary) and converts it into a list of teams.
3. Prepares input fields for: new team name, main task, sub-task for existing team, and dropdown for selecting existing teams.
4. Includes three buttons with separate functionality:  
   • Create a new team with a main task  
   • Add a sub-task to a selected team  
   • Mark a sub-task as completed or reopen it
5. Updates Firebase whenever a change occurs (addition or update).
6. Displays success or failure feedback via an output box.
7. Uses tabs (widgets.Tab()) to separate different actions: add team, add sub-task, update task status.
8. Styles the interface using consistent colors and custom CSS.
9. Returns the entire interface as a single VBox containing the title, tabs, and output box.

**Function: update\_summary\_section**

**Purpose**  
This function displays a styled card with a button that leads to the weekly summary report of faults and performance.

**Functionality**

1. Creates a styled heading with a gradient background and the text "Weekly Fault Analysis".
2. Adds a short description explaining the summary's purpose: "View comprehensive fault reports and performance trends".
3. Displays a fixed-size button with a custom style; clicking it triggers the create\_weekly\_summary\_page function.
4. The button is styled with matching colors and smooth hover transitions.
5. Creates an output area to show the full report when the button is clicked.
6. Returns all elements together in a VBox to be displayed on the main page.

**Function: create\_weekly\_summary\_page**

**Purpose**  
This function presents an interactive weekly report on anomalies in indoor and outdoor sensor metrics, including averages, charts, tables, and smart recommendations.

**Functionality**

1. Fetches sensor data from Firebase for INDOOR and OUTDOOR.
2. Filters the data for the past week.
3. Adds the weekday name (Sunday–Saturday) to each measurement.
4. Allows selection of a sensor from a dropdown (Temperature, Humidity, Pressure, Distance, DLIGHT).
5. Checks for fault conditions based on the selected sensor type.
6. Displays a weekly summary with the average, normal range, and a bar chart of daily fault counts.
7. Displays a detailed table with all abnormal readings for the selected sensor.
8. Calculates the fault percentage and shows smart system recommendations based on sensor condition and weekly performance.
9. Includes a "Close" button to exit the summary report.

***Search Page – Function Overview and Responsibilities***

**Class: IndexService**

**Purpose**  
This class is designed to build an **inverted index engine** for fetched webpages and manage document storage and search result handling.

**Functionality**

1. \_\_init\_\_: Initializes two dictionaries — one for storing documents and another for the inverted index.
2. add\_document(doc\_data):
   * Generates a unique ID for each document.
   * Adds all words from the document into the inverted index, storing the documents each word appears in, total counts, and per-document frequency.
3. get\_document(doc\_id): Returns the document content by its ID.
4. search\_word\_docIds(word): Returns a list of document IDs where the word appears (converted to lowercase).

**Function: fetch\_page**

**Purpose**  
Fetches a webpage from a given URL and parses its content using BeautifulSoup for text analysis.

**Functionality**

1. Accepts a URL as input.
2. Sends a GET request and retrieves page content.
3. If the request is successful (status 200), returns a BeautifulSoup-parsed HTML object.
4. If it fails, returns None.

**Function: crawl\_and\_index**

**Purpose**  
Scans a list of websites, processes their content, builds an inverted index, and uploads it to Firebase.

**Functionality**

1. Iterates over each URL in the list.
2. For each URL:
   * Fetches the page using fetch\_page.
   * Extracts visible text from HTML.
   * Cleans words: lowers case, removes stopwords, performs stemming.
3. Creates a doc\_data object (title, content, source) and adds it via IndexService.
4. Converts the index to a JSON-compatible format (sets → lists).
5. Uploads the full index to Firebase under the "Index" key.

**Class: QueryService**

**Purpose**  
Handles search queries based on the inverted index, locates matching documents, and stores query metadata.

**Functionality**

1. \_\_init\_\_: Receives an IndexService instance and initializes a queries dictionary.
2. create\_query(query\_data):
   * Accepts search terms and generates a unique query ID.
   * Locates documents containing all search terms.
   * Calculates word frequency per document.
   * Builds a complete query result structure and returns it or an error.

**Class: ResultService**

**Purpose**  
Processes query results: receives raw results from QueryService, formats them, and stores them by result ID.

**Functionality**

1. \_\_init\_\_: Accepts IndexService and QueryService instances; initializes a results dictionary.
2. format\_results(query\_i):
   * Accepts a query ID.
   * Locates documents returned by the query.
   * Calculates word and phrase occurrences per document.
   * Builds a formatted list (title, URL, term frequencies, phrase matches).
   * Stores and returns the structured result and document count.

**Function: search\_my\_index**

**Purpose**  
Runs the entire search pipeline: receives user query, processes it, and returns formatted results.

**Functionality**

1. Tokenizes, stems, and filters stopwords from user input.
2. If no relevant words remain returns an empty list.
3. Creates a new query via QueryService.
4. If query fails, returns nothing.
5. Passes the query result to ResultService for formatting.
6. Returns the formatted results and the number of matched documents.

**Function: display\_results**

**Purpose**  
Displays search results to the user in an interactive and readable UI.

**Functionality**

1. Clears the output area before displaying new results.
2. If no results shows a red "No results" message.
3. If results exist:
   * Displays the number of results found.
   * Checks if a phrase was searched (more than one word).
   * For each result:
     + Displays the title as a link.
     + Shows term/phrase frequencies.
     + Provides a direct link to the page URL.
4. Constructs HTML content and renders it using display(HTML(...)).

**Function: on\_search**

**Purpose**  
Triggered when the user clicks the search button. It runs the full search and displays results.

**Functionality**

1. Reads user input from the search\_box, trimming whitespace.
2. If empty — shows a red error message and stops.
3. If valid input:
   * Temporarily shows "Searching...".
   * Calls search\_my\_index to process the query.
   * Displays results using display\_results.

**Function: create\_logo\_widget**

**Purpose**  
Creates and returns an Output widget that displays a styled logo image at the top of the interface.

**Functionality**

1. Accepts:
   * width: width of the image.
   * url: image source (default to project logo).
2. Creates an Output widget.
3. Displays the image using IPython.display.
4. Returns the widget for use in UI layouts.

**Function: create\_search\_page**

**Purpose**  
Builds the user interface for the **search page**, including a text box, styled button, results area, and algorithm transparency feature.

**Functionality**

1. Adds a logo at the top using create\_logo\_widget.
2. Includes a text input for search terms/phrases.
3. Adds a styled search button with a magnifying glass icon.
4. Applies custom CSS to the search box and button.
5. Creates an output area for displaying results.
6. Binds the search button to on\_search to process user input.
7. Adds a button "How it works?" to explain the algorithm behind the search.
8. Combines all UI elements into a VBox.
9. Defines the show\_transparency function to show/hide the explanation.
10. Binds the transparency button to toggle the explanation area.
11. Adds this button and output to the main layout interface\_box.
12. Returns the complete search page UI.

***Engineers Rates – Function Overview and Responsibilities***

Function: create\_Rates\_page  
Purpose  
  This function builds the engineers' rating page, including a performance table and daily statistical metrics.  
Functionality  
  • Fetches engineer data from the database using fetch\_engineer\_data.  
  • Builds the rating table using build\_engineer\_race\_table.  
  • Wraps the table in a graphical interface using VBox.  
  • Returns the entire page as a complete display interface.

Function: build\_engineer\_race\_table  
Purpose  
  This function builds a graphical rating table for all engineers, including the display of daily scores, daily tasks, graphs, and general statistics.  
Functionality  
  • Calculates the scores for each engineer using compute\_engineer\_scores.  
  • Sorts the engineers by daily score in descending order.  
  • Calculates the total number of engineers, the average score, and the name of the daily leader.  
  • Creates a top statistics box that displays the general information.  
  • Defines an inner function render\_all\_engineers that renders the engineer cards.  
  • Calls the inner function to display the initial content.  
  • Returns a VBox that contains the statistics and engineer cards.

Function: render\_all\_engineers  
Purpose  
  This function displays the list of all engineers, including a personal info card for each engineer with daily score, performance chart, and task details. It is executed every time the engineer rating page is refreshed.  
Functionality  
  • Recalculates the daily score for each engineer using compute\_engineer\_scores.  
  • Sorts the list of engineers by daily score from highest to lowest.  
  • For each engineer:  
    o Creates a name box with background color based on their ranking.  
    o Displays a donut chart representing the daily score percentage.  
    o Displays a label with the daily point count.  
    o Builds their daily task list, including task names, point values, and a progress bar for each task.  
  • Displays all engineer cards in a VBox widget within a single display area.

Function: create\_task\_progress\_bar  
Purpose Creates a styled progress bar for each engineer's task, with colors varying based on the level of progress.  
Functionality  
  1. Determines the color style of the progress bar based on the percentage:  
    o Green (success) if progress is 100%  
    o Blue (info) if progress is between 75–99%  
    o Orange (warning) if progress is between 50–74%  
    o Red (danger) if progress is less than 50%  
  2. Creates a widgets.FloatProgress object that represents the task’s progress.  
  3. Sets the height and width of the bar for consistent display in the dashboard.  
  4. Returns the progress bar for use in the graphical component.

Function: create\_score\_donut  
Purpose  
Creates a styled Donut chart that visually displays an engineer’s daily score, using colors that match performance levels.  
Functionality  
  1. Creates a pie chart with a custom size (default: (1.3, 1.3)) and a background matching the dashboard.  
  2. Chooses a color based on the score range:  
    o Dark green for high scores (90 and above)  
    o Light green for mid-high scores (70–89)  
    o Yellow for medium scores (50–69)  
    o Orange for low scores (below 50)  
  3. Displays the central score as a percentage inside the donut.  
  4. Renders the pie segments so that the missing portion (100 - score) is colored as the background.  
  5. Returns a widgets.Output containing the chart for display in the dashboard.

Function: compute\_engineer\_scores  
Purpose This function calculates the daily score (daily\_score) and completion rate (completion\_rate) for each engineer, based on the progress of their assigned daily tasks.  
Functionality  
  1. Iterates through each engineer in the engineer\_data list.  
  2. For each engineer, calculates the score contribution of each task:  
    (progress / 100) \* points  
  3. Accumulates these contributions into the engineer's daily\_score.  
  4. Sums the total possible points to perform (total\_possible).  
  5. Adds the daily\_score field to the engineer with a rounded integer value.  
  6. Calculates the completion\_rate as the ratio of daily score to total possible points, expressed as a percentage string (e.g., "75%").  
  7. If there are no possible points, sets the completion rate to "0%".  
  8. Returns the list of engineers with the two new fields: daily\_score and completion\_rate.

***Team overview– Function Overview and Responsibilities***

**Function: create\_team\_overview\_page**  
**Purpose**  
This function builds the **Team Overview** page, displaying progress cards for each team, including task summaries and subtask completion percentages.

**Functionality**

• Fetches team task data from the Firebase database using update\_cards.  
• Calculates the completion percentage of each team's subtasks.  
• Creates styled HTML cards for each team, including the main task, subtasks, and a visual progress bar.  
• Applies conditional color coding (green, yellow, gray) based on progress level.  
• Generates a page header with a styled title.  
• Returns the full page as a VBox containing the header and all team cards.

***Statistics Page– Function Overview and Responsibilities***

**Function: create\_statistics\_page**  
**Purpose**  
  Builds the main statistics page in the system – for presenting advanced analysis of sensor and team data in a comprehensive and visual way.  
**Functionality**  
  1. Displays a temporary loading animation using HTML and CSS while fetching data.  
  2. Fetches relevant data:  
    o fetch\_historical\_data() – historical sensor data  
    o fetch\_engineer\_data() – engineer and task data  
  3. Builds interactive components:  
    o create\_failure\_simulator – failure prediction simulator  
    o create\_reliability\_index – general reliability index  
    o create\_sensor\_trend\_section – real-time trends  
    o create\_root\_cause\_tree – root cause tree analysis  
    o create\_sensor\_trend\_analysis – sensor-specific graphs  
    o create\_comparative\_stats – indoor/outdoor sensor comparison  
    o create\_team\_impact\_timeline – team performance vs. failure analysis  
  4. Applies consistent professional styling to all charts and titles.  
  5. Returns a complete VBox UI with all components in clear order.  
  6. In case of error – displays an appropriate message.

**Function: create\_team\_impact\_timeline  
Purpose**Displays a comparison between detected failures and team actions throughout the week – highlighting effectiveness. **Functionality**  1. Creates a day-by-day table with fields: Day, Number of Failures, Number of Tasks, Effectiveness.  
  2. Computes relative percentages for each column based on the day with the highest value.  
  3. Displays colored bars (Red/Green) to visually compare number of failures vs. actions.  
  4. Determines effectiveness (High or Low) based on whether the team acted more than the number of failures.  
  5. Builds a styled UI with customized CSS, colors, spacing, and title.

**Function: create\_failure\_simulator**Purpose  
  An interactive tool that simulates failure scenarios based on sensor values and computes operational risk. **Functionality**  1. Creates sliders for 5 sensors: temperature, humidity, pressure, distance, and light intensity.  
  2. Any change in the sliders triggers a Risk Score calculation based on deviations.  
  3. Displays the result visually using color (Green / Orange / Red) according to risk level:  
    o  Low risk (0–0.4)  
    o  Medium risk (0.4–0.7)  
    o High risk (0.7–1.0)  
  4. The UI is displayed in a VBox with adapted styling, clear text, and colored emphasis by risk.

**Function: create\_comparative\_stats  
Purpose**This function displays a statistical comparison of Indoor and Outdoor sensors using an interactive BoxPlot chart to illustrate distribution, average, and threshold deviations.  
**Functionality  
  1.**Creates a dropdown with all available sensors (including units and Indoor/Outdoor distinction).  
  2. Allows the user to select a sensor and display its value distribution via a BoxPlot.  
  3. Filters data by source (Indoor/Outdoor) and sensor type.  
  4. Adds graphical lines for upper and lower thresholds if available.  
  5. Styles the view with clear colors, title, help text, and real-time interactivity.  
  6. Returns a full VBox UI with selection menu and dynamic chart.

**Function: create\_sensor\_trend\_analysis  
Purpose**This function displays an interactive graph of sensor trends (indoor and outdoor) by source, sensor type, and time – including average, standard deviation, and alert thresholds. **Functionality  
  1.**Accepts a unified dataset for Indoor and Outdoor sensors with Time and Source columns.  
  2. Creates a selection menu for the source (Indoor/Outdoor) and relevant sensors.  
  3. Filters data based on selection and rounds time to 5-minute intervals.  
  4. Calculates average, standard deviation, and alert thresholds (if available), and plots them on the chart.  
  5. Styles the graph with colors, lines, and a shaded area around the mean.  
  6. Allows full interaction – changing menus automatically updates the graph.  
  7. Returns a VBox view with title, menus, and an interactive chart.

**Function: create\_root\_cause\_tree  
Purpose**This function builds an interactive tool for root cause analysis of failures based on cause probabilities retrieved from Firebase.  
**Functionality  
  1.**Fetches data from /FailurePaths in the database.  
  2. Creates a dropdown menu with all identified anomaly types.  
  3. For each selected anomaly, displays possible causes and a probability bar for each cause.  
  4. Styles the interface with custom CSS – including width, colors, and fonts.  
  5. Uses simple graphical elements to clearly present statistical information.  
  6. Returns a VBox view with a title, selection menu, and a chart of causes by probability.

**Function:** create\_sensor\_trend\_section **Purpose**This function displays interactive graphs of indoor and outdoor sensor metrics from the last hour, allowing the user to select the desired sensor via a dropdown menu.  
**Functionality  
  1.**Fetches sensor data from the last hour (Indoor and Outdoor) using fetch\_last\_hour\_data.  
  2. Defines a dictionary with display options per sensor: display name, data column, title, unit, and relevant database.  
  3. Creates a dropdown menu to select the sensor to display.  
  4. Listens for selection changes and displays the corresponding chart using create\_time\_line\_chart.  
  5. Loads the default chart on initial startup.  
  6. Styles the graphical components with HTML and CSS for neat display of the sensor name, selection menu, and chart.

**Function: create\_time\_line\_chart**Purpose  
  This function displays a line chart showing the trend of a sensor’s metric over time (e.g., temperature or humidity), averaged in 5-minute segments and checking for threshold violations. **Functionality**1. Sorts the data by time and adds a rounded timestamp column (Time\_5min) every 5 minutes.  
  2. Calculates the average of the metric (column) for each 5-minute window.  
  3. Computes the elapsed time since the beginning in minutes (MinutesFromStart) for the X-axis.  
  4. Checks for value violations against min/max thresholds using the thresholds table.  
  5. Creates a line chart of the averaged values, with red lines indicating threshold breaches.  
  6. Displays general statistics below the chart – min, max, and average values.  
  7. Shows a warning message if any threshold was breached.

**Function: create\_reliability\_index  
Purpose**To create a visual display of each sensor’s reliability index in the system, based on accuracy, stability, and response time data pulled from Firebase.  
Functionality  
**1.**Checks if the data is empty (df.empty) – if so, displays “No data available.”  
  2. Fetches reliability scores from all sensors using get\_reliability\_scores().  
  3. For each sensor:  
    o Calculates the average of the three metrics (accuracy, stability, response\_time) and displays a colored overall progress bar.  
    o Creates an HTML card with individual bars for each sub-metric.  
  4. Returns a VBox component containing all the cards in a horizontal display (HBox) with auto-wrapping (flex\_wrap).

***Chatbot Page– Function Overview and Responsibilities***

**Function: chatbot\_page**  
**Purpose:**  
  Creates an interactive AI chatbot page (Gemini) that can read all information from Firebase and respond to questions about the system – such as sensors, teams, failures, and more.

**Functionality:**  
  1. Displays a loading animation until the chat is ready.  
  2. Fetches data from Firebase:  
    o Loads the entire JSON database using FBconn.get('/', None).  
  3. Defines Google’s Gemini model:  
    o Configured with an API Key.  
    o Creates a chat object with an empty history.  
    o Sends an initial message with all data loaded from Firebase.  
  4. Builds the chat interface:  
    o Message area (messages\_box) + input field and send button.  
    o Styled message bubbles – each user/bot message appears in separate colorful design.  
  5. Upon sending:  
    o Captures the user’s input.  
    o Adds the user’s message to the bubbles.  
    o Adds a “Typing…” bubble for the bot.  
    o Sends the user’s message to Gemini and receives a reply.  
  6. Replaces the “Typing…” bubble with Gemini’s actual response after processing.  
  7. Automatically scrolls to the bottom of the page after each reply.

***Meating Managment Page– Function Overview and Responsibilities***

**Function: create\_Meetings\_page**

**Purpose**  
  Builds the full meeting scheduling page, including creating new meetings, viewing scheduled ones, filtering by date, and editing/deleting meetings.  
**Functionality**  
  • Defines internal functions to load styles, validate time format, fetch teams, and render widgets.  
  • Builds the interface with two main sides: left (form) and right (meeting list).  
  • Handles scheduling, saving, and displaying meetings from Firebase.  
  • Wraps all widgets into an interactive HBox layout.

**Function: is\_valid\_time\_format(value)**

**Purpose**  
  Validates that a given time string is in the correct HH:MM format.  
**Functionality**  
  • Uses regex to check if input matches 24-hour format.

**Function: load\_custom\_styles()**

**Purpose**  
  Applies consistent CSS styling to all form widgets and buttons.  
**Functionality**  
  • Injects CSS using HTML() and display() to style dropdowns, inputs, toggle buttons, and meeting buttons.

**Function: fetch\_team\_names()**

**Purpose**  
  Fetches a list of unique team names from /DailyTeamsTasks in Firebase.  
**Functionality**  
  • Retrieves and parses Firebase data.  
  • Handles both dict and list structures.  
  • Returns fallback values if the fetch fails.

**Function: saveMeeting(team, date, time, location, notes, output\_area)**

**Purpose**  
  Saves a new meeting record to Firebase.  
**Functionality**  
  • Prepares a meeting dictionary and sends it to /Meetings.  
  • Displays a success or error message.  
  • Triggers UI refresh.

**Function: schedule\_Meeting(b)**

**Purpose**  
  Triggered when the "Schedule Meeting" button is clicked.  
**Functionality**  
  • Validates all fields (team, date, time, location).  
  • Prevents saving invalid or incomplete data.  
  • Calls saveMeeting() if input is valid and clears the form afterward.

**Function: render\_right\_side()**

**Purpose**  
  Rebuilds the right-side meeting list and reloads meetings for the selected date.  
**Functionality**  
  • Calls on\_date\_change() with current date value.

**Function: show\_meeting\_popup(key, meeting)**

**Purpose**  
  Displays a detailed popup for editing or deleting a specific meeting.  
**Functionality**  
  • Populates input fields with current meeting data.  
  • Allows user to update or delete the meeting.  
  • Includes a close button and styling.

**Function: update\_meeting(b) (internal)**

**Purpose**  
  Handles updating an existing meeting when the "Update" button is clicked.  
**Functionality**  
  • Validates fields again.  
  • Updates Firebase using patch.  
  • Displays a confirmation and refreshes the list.

**Function: delete\_meeting(b) (internal)**

**Purpose**  
  Deletes the meeting from Firebase when "Delete" is clicked.  
**Functionality**  
  • Deletes meeting from /Meetings/{key}.  
  • Displays confirmation and refreshes view.

**Function: close\_popup(b) (internal)**

**Purpose**  
  Closes the popup and reloads the regular meeting list.  
**Functionality**  
  • Calls render\_right\_side().

**Function: get\_meetings()**

**Purpose**  
  Retrieves all meetings from Firebase.  
**Functionality**  
  • Fetches and returns data from /Meetings.

**Function: display\_results(meetings)**

**Purpose**  
  Displays a list of meetings visually on the right panel.  
**Functionality**  
  • Handles empty results.  
  • Creates formatted cards with team, date, time, location, and notes.  
  • Each card includes a button to open show\_meeting\_popup().

**Function: on\_date\_change(change)**

**Purpose**  
  Handles filtering meetings based on selected date.  
**Functionality**  
  • Filters meetings to only those matching the selected date.  
  • Displays filtered list using display\_results().

***Firebase Structure***

* **/DailyTeamsTasks**  
   Stores daily tasks assigned to each team, including the team name, task description, and completion status (Done: true/false).
* **/EngineerPerformance** Stores daily performance data for each engineer. Each entry includes:  
    • name: the engineer's name  
    • tasks\_today: a list of tasks completed today, where each task includes:  
      - task: task description  
      - points: assigned points for the task  
      - progress: completion percentage (0–100)
* **/FailurePaths**  
   Contains root cause analysis data for failures, mapping each anomaly to potential causes with probabilities.
* **/Index**  
   Holds the inverted index .
* **/LiveSensors**  
   Stores real-time data from indoor sensors, including temperature, humidity, pressure, and light values.
* **/LiveSensorsOUTDOOR**  
   Similar to /LiveSensors, but for outdoor sensor data collected in real-time.
* **/Meetings**  
   Contains all scheduled team meetings, including team name, date, time, location, and notes.
* **/SensorScores**  
   Stores reliability scores for each sensor, based on metrics like accuracy, stability, and response time.
* **/SensorThresholds**  
   Defines the minimum and maximum acceptable values for each sensor type, used for alert and trend analysis.