Engineer Guide- Pheonix Project

Table of Contents

Project Overview

Project Objective:
The system was developed for the machine lab at ORT Braude College in order to streamline work processes, centralize all necessary information in one system—such as sensor data and tasks to be completed—and increase productivity by awarding points (which can be used to purchase prizes) to employees who complete tasks.

Languages and Technologies:

Programming Language: Python

User Interface: Gradio

Database: Firebase

Code Management: Google Colab + GitHub

External Libraries:

Library	Description
gradio	Used to create an interactive user interface
plotly	Used to display sensor data in dynamic graphs
paho-mqtt	Used to receive data from MQTT-based sensors
nlkt	Used for natural language processing (stop-word removal, stemming)
importnb	Allows importing .ipynb notebook files as Python modules
re	Used for regular expression processing during indexing and search

Python Standard Libraries:

Library	Description
os/sys	Used for file management and dynamic module loading
json	Used to parse and manage JSON data from sensors
datetime	Used to process and format time and date values

Project Structure and Main Components

Logic:
This folder contains the business logic of the system, including interaction with Firebase.

File	Description
Admin	Manages administrator-level features. Allows an admin to: • Get the top 10 most searched terms • Record search terms • View current index status • Get, add, and delete tasks
ChatbotLogic	Handles natural language conversations between users and the system. Integrates a Gemini-based model to process user input and generate responses while maintaining conversation history.
CloudDB	Handles all interactions with the Firebase Realtime Database: • Saves and loads search index data and statistics • Manages sensor data from devices • Creates, updates, and deletes tasks • Handles user data (accounts, coins, existence checks) • Supports general-purpose data retrieval
Indexmgtt	Responsible for crawling and indexing websites: Crawls the MQTT site and fetches all sub-URLs Extracts and stems words from pages Removes stop words Saves the final index and stats to CloudDB
SearchService	Handles internal search functionality: • Loads search index from CloudDB • Processes user queries (with stemming and stop-word removal) • Ranks and returns relevant results • Records search term frequency in the database
SensorDataProcessor	Processes real-time sensor data via MQTT: • Connects to public MQTT broker (test.mosquitto.org) • Subscribes to indoor/outdoor sensor topics • Receives and decodes JSON sensor data • Uploads data to Firebase via CloudDB • Monitors for inactivity and auto-disconnects after timeout
SensorVisualLogic	Visualizes sensor data from Firebase: • Fetches indoor/outdoor sensor readings • Displays time-based graphs using Plotly • Computes average sensor values for selected hour • Dynamically updates sensor and time dropdowns • Supports Distance, Temperature, Humidity, Pressure, and DLIGHT sensors
TaskLogic	Manages task operations in Firebase: • Retrieves all tasks • Adds new tasks • Gets a task by name • Removes tasks by name and Firebase key

UserManager	Manages user authentication and account data: • Handles login and admin access checks • Registers new users with validation • Retrieves all user accounts • Updates coin balance for a user

UI:

One central notebook manages all UI components using Gradio.

File	Description
UI	Contains the following Gradio interfaces: • Admin Panel UI • Admin Tasks Dashboard • User Tasks Dashboard • Search Engine UI • Sensor UI • Shop UI • Chatbot UI • Unified UI (final integration of all pages)

Key Functions and Methods

The system includes several core methods that handle data processing, support the user interface, and integrate with external services such as Firebase and MQTT.

search_word(query)

Module: SearchService

Purpose: Processes a user's search query and returns the most relevant results based on word stemming and stop-word filtering.

Key Features:

-Loads the latest search index from Firebase.

-Preprocesses the query to normalize and filter terms.

-Scores and ranks pages based on word relevance.

-Returns the top-matching results to the UI.

insert_to_db_index(results, page_count)

Module: CloudDB

Purpose: Stores indexed search terms and metadata to Firebase.

Key Features:

- Overwrites current index with the latest results.

- Automatically updates metadata.

- Returns the timestamp of the last update.

insert_to_db_sensor(location, data)

Module: CloudDB

Purpose: Saves real-time sensor data for a specific location.

Key Features:

- Used by MQTT listeners and sensor monitoring modules.

insert_to_db_stats(stats)

Module: CloudDB

Purpose: Updates or initializes the full index statistics record.

Key Features:

- Used in bulk updates when recalculating or resetting statistics.

insert_to_db_task(task)

Module: CloudDB

Purpose: Replaces the full task list with a new structure.

Key Features:

- Not to be confused with `add_task`, which appends a new task.

insert_to_db_user(username, user_data)

Module: CloudDB

Purpose: Adds or updates a user profile in the users collection.

Key Features:

- Ensures user data is structured and easily retrievable

change_coins_for_user(username, coins)

Module: CloudDB

Purpose: Updates the coin balance of a specific user by adding (or subtracting) a specified amount.

Key Features:

- Fetches existing user data and creates a copy to avoid overwriting unrelated fields.
- Adds the given value to the current coin balance (defaulting to 0 if undefined).
- Saves the full updated user object back to the database.
- Returns the new coin total if the user exists, or False if not found.

record_search_term(term)

Module: Admin

Purpose: Logs user search terms to Firebase for statistical analysis and admin monitoring.

Key Features:

- Updates term frequency count.

- Supports admin tools such as top-10 search queries and keyword tracking.

mqtt_handler(self)

Module: SensorDataProcessor

Purpose: To initiate and establish a connection between the MQTT client and a broker, register necessary callbacks, and handle the initial connection process including waiting for successful connection confirmation.

Key Features:

- Registers the MQTT callbacks for connect, disconnect, and message handling.

- Attempts to connect to the public MQTT broker (test.mosquitto.org) on port 1883.

- Starts the MQTT network loop in the background using loop_start().

- Waits up to 10 seconds for a successful connection before returning a result.

- Returns True if connected successfully, otherwise returns False.

- Handles connection failures gracefully with error output.

Interesting Code Snippets

Unified UI:

The create_main_interface() function serves as a Unified UI layer, bringing together all user-facing components-login, user features and admin features-into a single, cohesive interface. Instead of scattering interfaces across multiple entry points, this unified approach centralizes the user experience and dynamically reveals relevant sections based on authentication status and role (admin vs. regular user). Through smart visibility toggles (gr.update(visible=...)), the UI efficiently switches between guest, user, and admin modes while maintaining a consistent structure and theme. This design not only improves maintainability but also ensures a seamless and intuitive interaction flow for all user types.

Search Service:

This code powers a smart search system for MQTT-related content. It connects to a pre-built term index stored in a cloud database and uses stemming to match search queries not just by exact words, but also by root forms (e.g., "publish", "publishing"). It tracks exact and stemmed appearances across documents, ranks results by relevance score**,** and fetches live content previews from the matched URLs.

The score reflects how relevant a document is to the query - based on the number of matches (exact and stemmed) found in each document. The more relevant terms it finds, the higher the score

Sensor Data Processor:

This code sets up a robust MQTT data processing microservice. It connects to a public MQTT broker, subscribes to indoor and outdoor sensor topics, and stores incoming data to a cloud database.

Key features include:

- Auto-reconnect logic for unexpected disconnections.

- Timeout handling: stops the client if no messages are received for 10 seconds.

- JSON parsing + dynamic DB writes.

Running The System

To run the system on Google Colab, simply open the main UI.ipynb file and execute all cells.
The code:

1. Clones the GitHub repository if not already cloned.
2. Switches to the Dev branch.
3. Loads logic and UI modules.
4. Launches the integrated Gradio interface.

There is no need for any special installation or manual imports-all necessary modules and files are already built into the code.