alarm-thingy: An IoT Smart Alarm to Help You Wake Up More Gracefully

Internet of Things

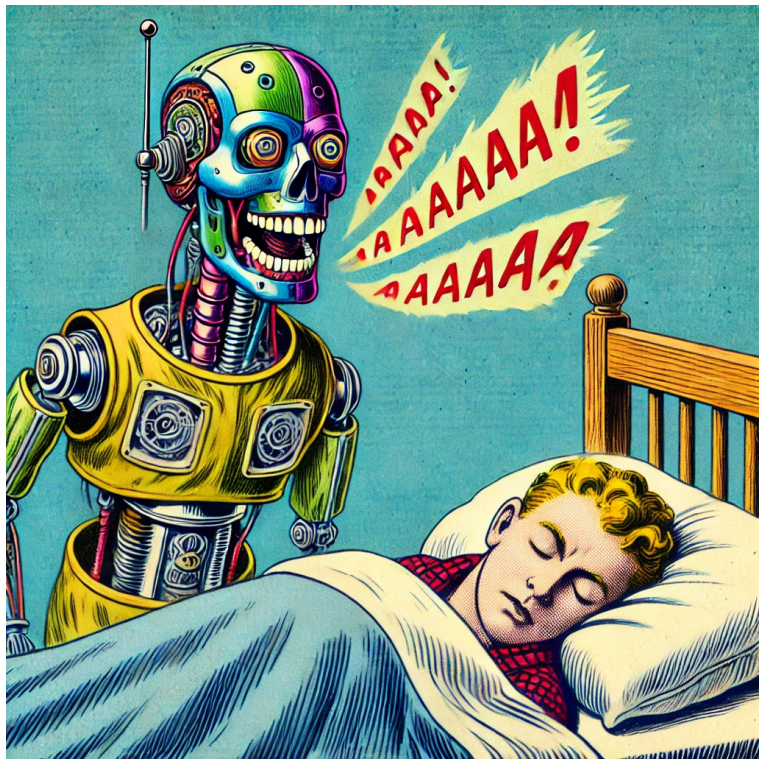
Presented by Angelo Galavotti

Università di Bologna, December 2024

Introduction

Traditional alarm clocks often rely on loud sounds, which can disrupt not only the user but also others nearby.

This conventional approach to waking can lead to stress, irritation, and an unpleasant start to the day.



alarm-thingy

alarm-thingy is a smart alarm developed to ease and **improve** the waking up experience of the user. It works by reading a **pressure mat sensor**, which detects if the user is laying in their bed.

Here's a quick overview of the main features:

- **React-based** UI, with **statistics** display and a settings menu.
- Additional **Telegram-based** interface (implemented via a bot).
- **Grafana dashboards**, to view the latest **state history** of the sensor and the **average sleeping time** for each day.
- Based on the current **weather condition**, the alarm will play a different chime, to notify the user.
- Predicts the **likelihood** of the user being in bed at the current time, through a **Prophet model**.
- and more...

Hardware

alarm-thingy is based on an **ESP32** board, to which these periphals are connected:

- **DFPlayerMini**, which loads the music through an SD card and functions as the DAC of the speaker.
- A simple **speaker**.
- A **pressure mat** sensor, usually used in car seats.
- Blue LED.

Once the schematics were finalized, the components were soldered onto a circuit board and encased in a 3D printed shell.

Software components

The software suite of alarm-thingy includes:

- The **ESP32 firmware**, based on MicroPython.
- A **Data Proxy**, which also functions as the backend of the Web App.
- A **Data Analysis** module, which loads the Prophet model and provides insights on the sleeping duration.
- A **Web-based** user interface..
- A **Telegram bot**.
- **InfluxDB** and **Grafana** instances.
- **MQTT Broker**.

Aside from the firmware and the Data Proxy, the other components run on **Docker containers**, connected together through **Docker compose**.

ESP32 firmware

- Based on **MicroPython**, uses a library for communication with the DFPlayer Mini module.
- At startup, the ESP32 **attempts to connect** to the network and the MQTT Broker.
  - At the same time, it plays **sound effects** to notify the user of the **network status** of the ESP32.
- In addition, the Data Proxy finds the ESP32 through **mDNS**. Afterwards, it sends it the **IP of MQTT Broker** through a **TCP** connection.
- Features a **fallback mechanism** for when connection to the nodes is lost, and quickly recovers its state when the connection is reestablished.

ESP32 firmware

- Afterwards, it reads the sensor data and sends it to the Data Proxy through either **HTTP or MQTT**.
  - MQTT: uses the `iot_alarm/sensor_data` topic.
  - HTTP: sends the data directly to the Data Proxy.
- The data readings from the sensor are filtered using a **moving average**.
- The **blue LED blinks** according to the rate of **successful** data transmissions.

Data Proxy

- Implemented using **Python** and **Flask**.
- Provides the **API endpoints** that are leveraged by the React Web App and the Telegram bot
  - Such endpoints allow for the user to add, update or remove alarms, among other functionalities.
- Receives **data** from the ESP32 and sends it to the **InfluxDB** instance.
- In another thread, it runs the **alarm logic**.
- Before sending the `start_alarm` command to the ESP32, it fetches the **weather data**[1] and sends it to the alarm.
  - The ESP32 uses the data to decide which ringtone to pick.

[1] the weather data and geocoding API is provided by `https://open-meteo.com/`

Data Analysis module

- Implemented using **Python** and **Flask**, again.
- Provides the **API endpoints** used to get information on:
  - The total **sleep duration** of the user.
  - The **mean latency** of HTTP requests sent by the ESP32.
  - The **likelihood** of the user being on the bed at the current time, predicted via a Prophet model.
- The **delay information** is received from the ESP32 via **MQTT**. The module then uses the data to compute the average.
- The Prophet model is trained on synthetic data, however the module features scripts to allow for training using data extracted from InfluxDB.
- The module contains **scripts** to compute the **accuracy** of the alarm in terms of user detection.

Web App

- Implemented using **React**, designed using **Tailwind** and **daisyUI** components.
- Allows the user to **add**, **delete**, **update** the alarms.
- Contains an **'Information' menu**, which shows:
  - The total sleeping time in the last 24 hours, and the **mean HTTP latency**.
  - A **Grafana dashboard** showing the sensor readings and their moving average.
  - The current **weather** (and allows the user to **set a location**).
  - The **likelihood** of the user being in bed at the current time.
- Features a **'Settings' menu**, which allows the user to:
  - Set the **sampling rate** of the alarm.
  - Set the **volume** of the alarm.
  - **Stop the alarm** if its currently playing a ringtone.
  - Switch **from HTTP to MQTT** and vice versa for sending sensor information, **on the fly**.
  - Enable the **'angry mode'**.

Web App

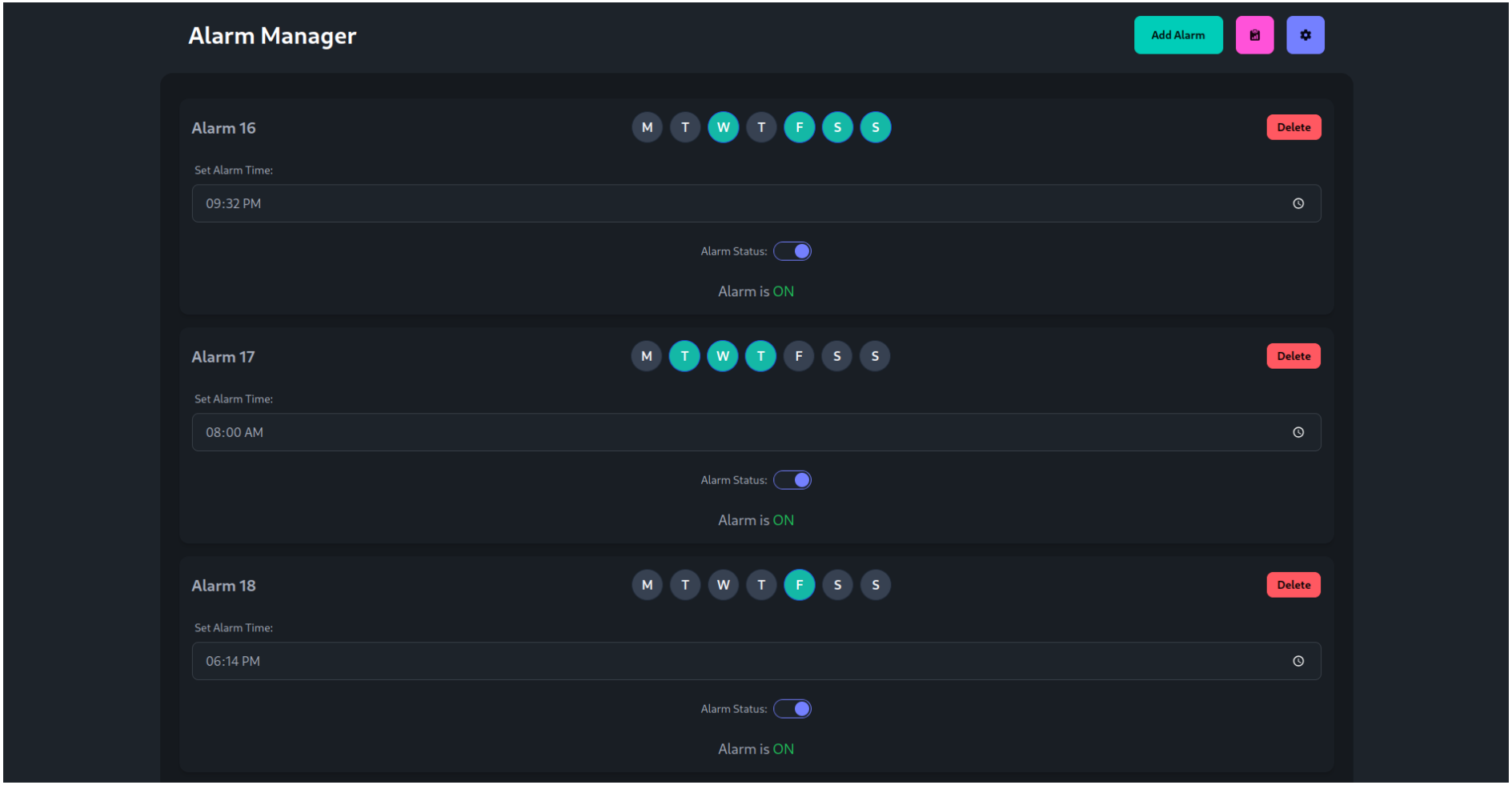


Figure: The landing page of the Web App.

Web App

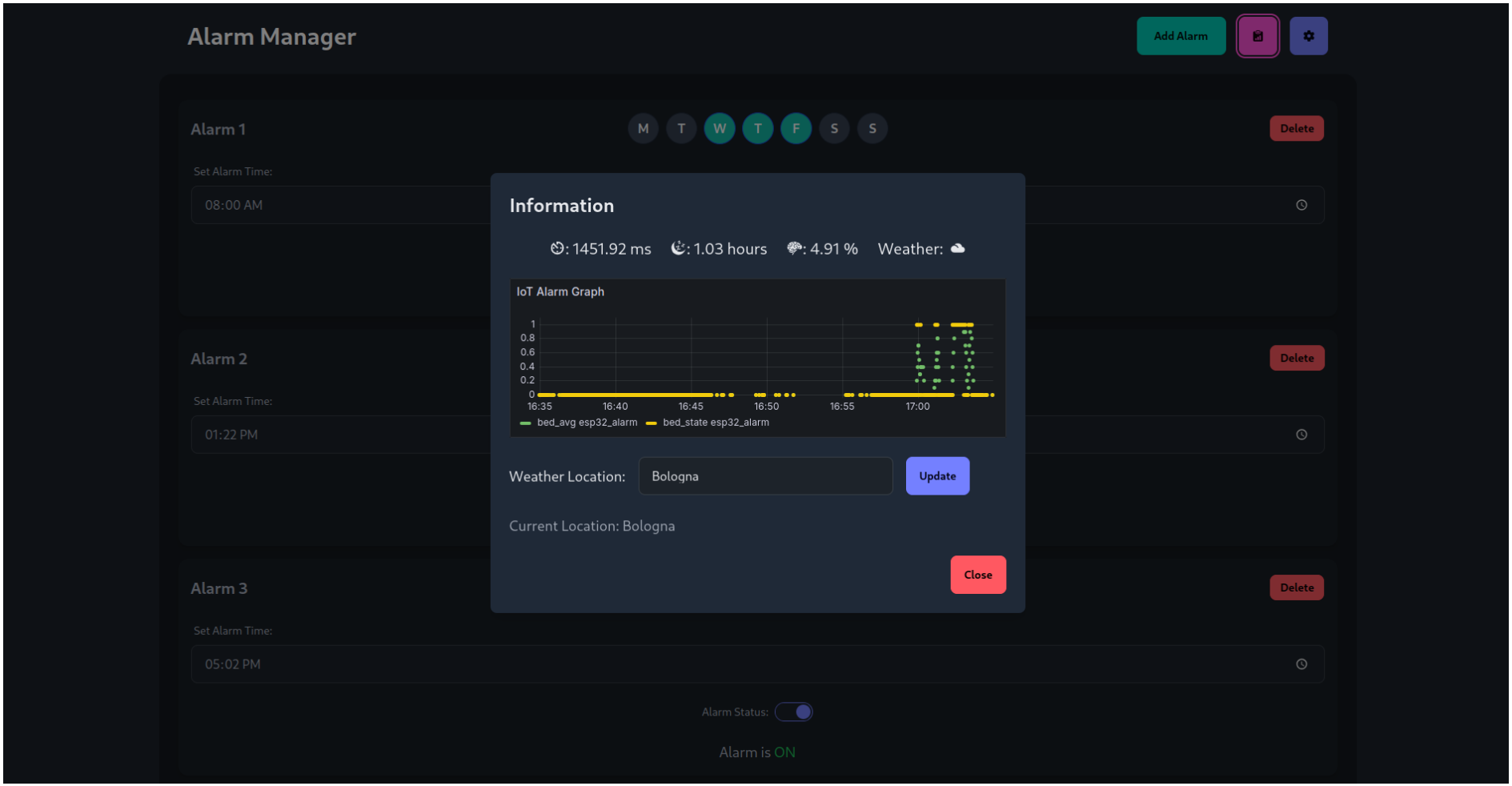


Figure: The 'Information' menu.

Web App

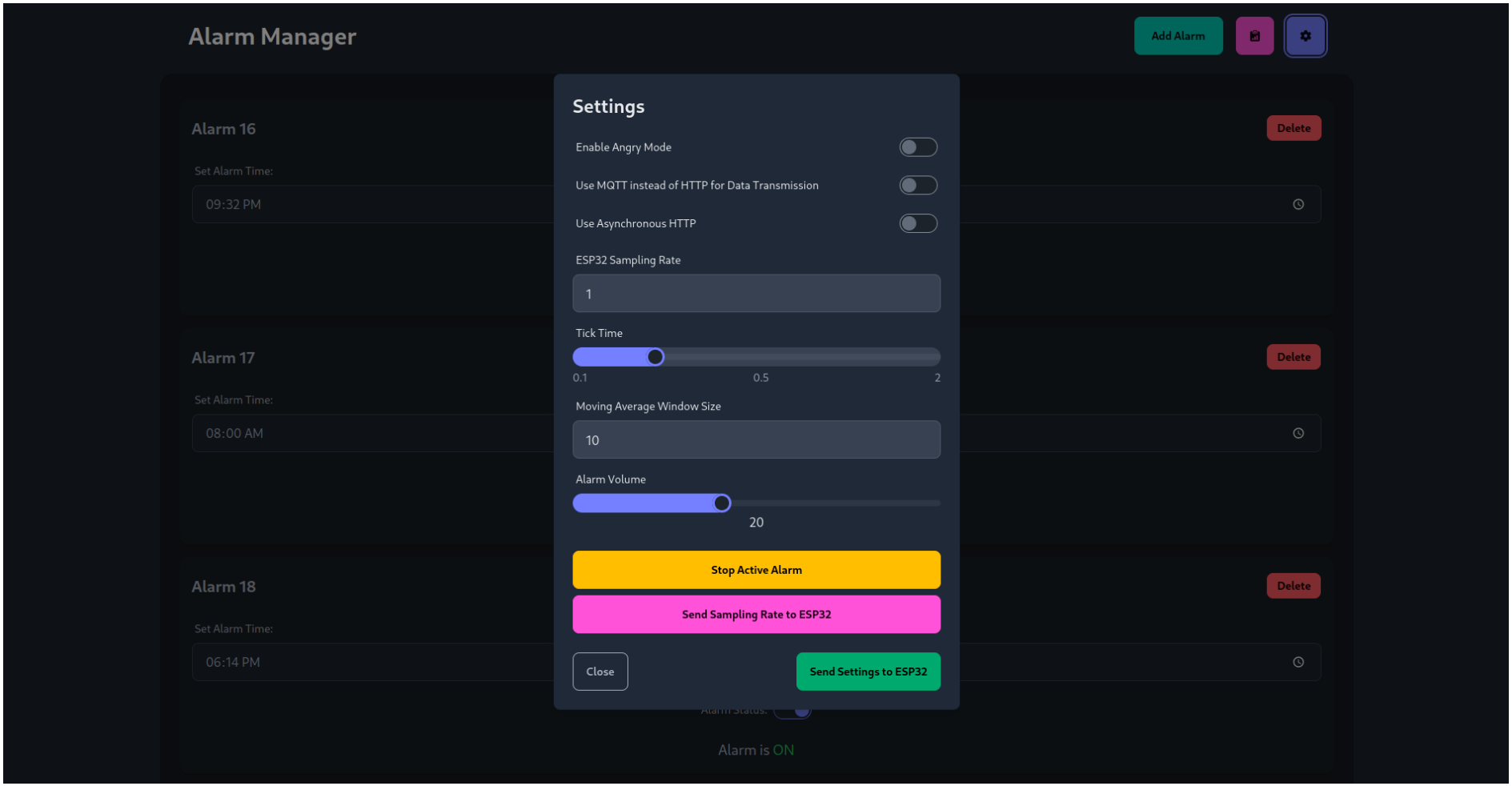


Figure: The 'Settings' menu.

Telegram Bot

- Implemented using **Python** and the `python-telegram-bot` library.
- Retains a lot of the same features as the React-based UI. Some of the available commands are:
  - `/add_alarm`: adds a new alarm.
  - `/delete_alarm`: deletes an alarm from `alarms.json`.
  - `/update_alarm`: updates the properties of alarm.
  - `/stop_alarm`: stops the alarm that is currently running.
- Can be accessed by messaging the bot at `@alarm_thingy_bot`.

Telegram Bot

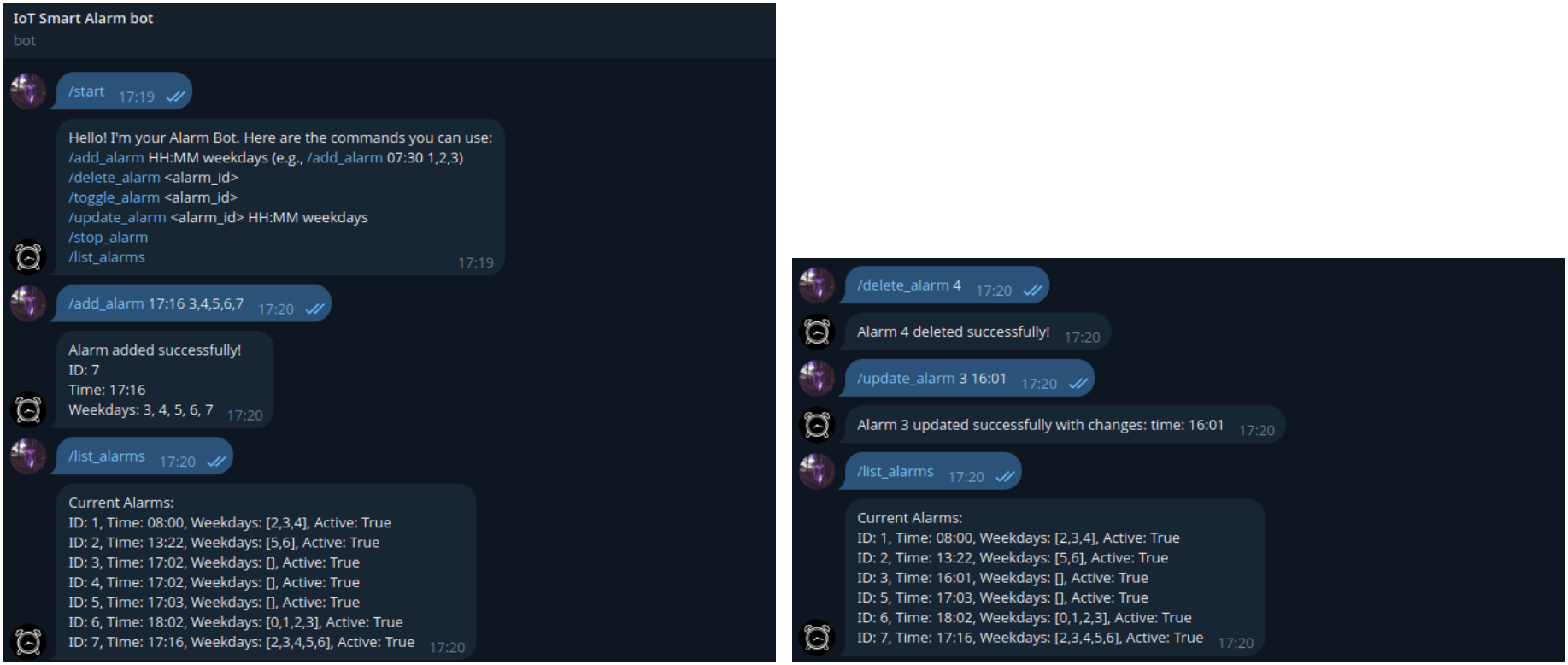


Figure: An example of interaction with the Telegram Bot.

MQTT Broker/Grafana/InfluxDB

- The MQTT Broker, Grafana and InfluxDB instances run on **Docker** containers.
- The MQTT Broker adopted is based on Eclipse's **Mosquitto**.
- The Grafana instance features **two main dashboards**, which display:
  - The **mean sleep duration** on each day.
  - The latest values of the **sensor state**. A variation of it is displayed on the Web App's Information menu.

Results - User detection

The sensor was tested in four trials, each lasting approximately one hour.

Trial	Accuracy
1	94.06%
2	88.21%
3	96.72%
4	92.11%
Mean	92.78%

Table: Measures obtained in the four trials.

Results - Mean Latency

The mean latency hovers around 430 ms. The mean value of the averages is 397.61 ms.

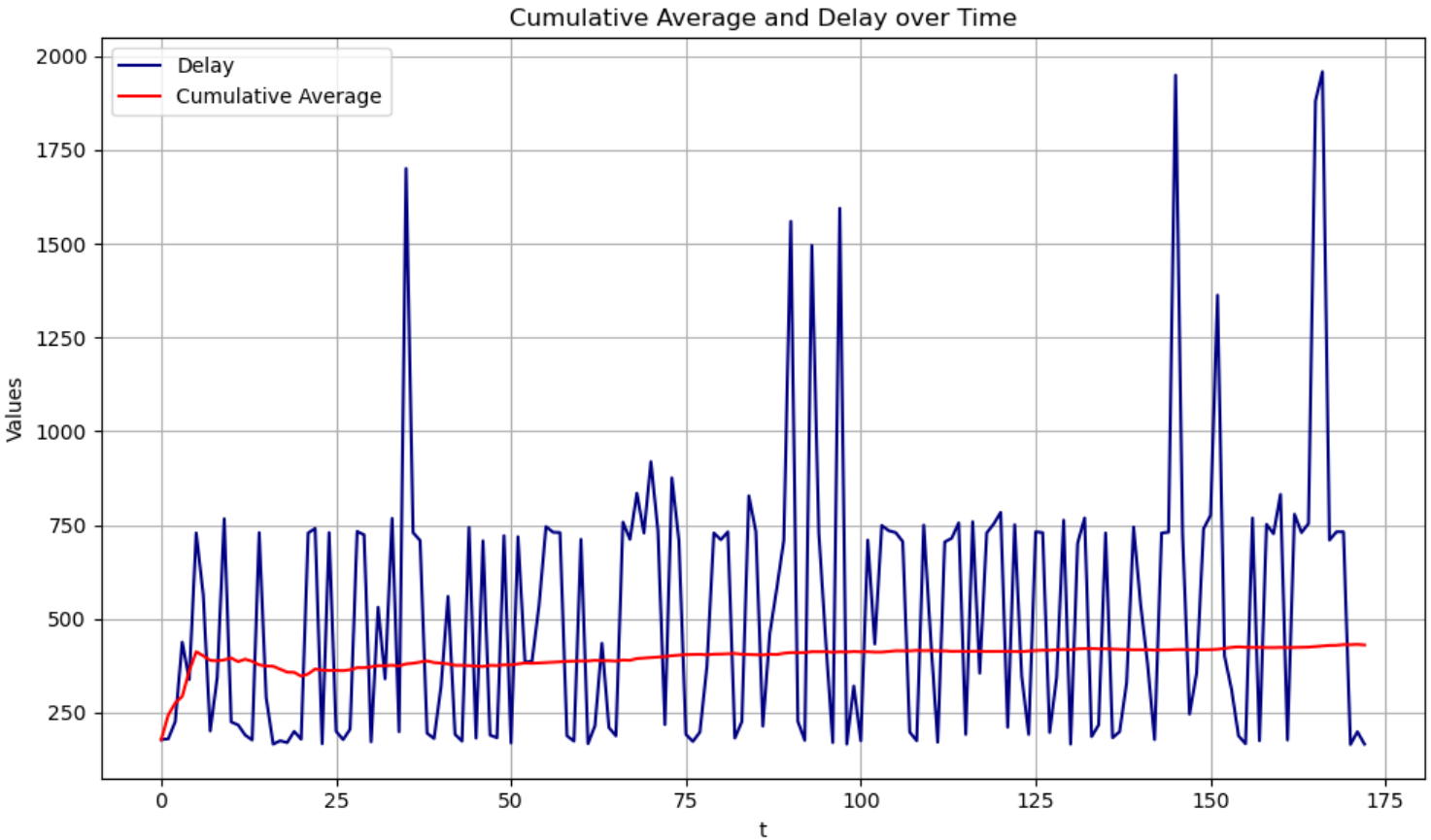


Figure: A graph showing the cumulative average of the delays for HTTP requests.

Video demo

Demo time!

https://youtu.be/afLa9XW69rA

Thank you!