A Smart Early Detection and Alert System for Severe Weather Events

Group 30: Chandan Girish, Tobias Senger, and Ruben Verma

Service Computing Department, IAAS, University of Stuttgart
st175114@stud.uni-stuttgart.de
st161427@stud.uni-stuttgart.de
st151078@stud.uni-stuttgart.de

Abstract. Due to global warming and extensive human activities [1,2], there has been a critical impact on environment conditions in recent years which led to a number of extreme weather events with serious consequences for the environment and humans. For this reason, this work describes building a smart IoT system for early detection, alerting and tackling of harsh weather events with the aim of increasing the safety and health of the people.

Keywords: early detection and alert system · severe weather events · weather disasters · global warming · climate change · smart · IoT.

1 System Introduction

1.1 Goal and Benefits

Severe weather events have become more frequent in the last years and decades, also as a result of human-made climate change [1,2]. These extreme weather events often have serious consequences for the environment and the people. It was only last year that a flood disaster in Germany caused many people to loose their homes or even their lives. Therefore our goal was to detect severe weather events efficiently so that immediate actions can be taken to prevent or at least decrease the consequences of such events. To achieve this goal, this project report describes our smart IoT system for early detection and alerting of severe weather events with the goal of increasing the safety and health of the people. To implement this system, several sensors were used to detect emerging severe weather events (i.e. temperature sensor, water level sensor, wind sensor etc.). Some of these sensors are physical sensors, other sensors are simulated. Based on the sensed data, our system detects current events and automatically assigns a risk level to them. Depending on the risk of a severe weather event, our system uses multiple actuators to alert local residents and authorities (i.e. by alarm tone, alarm sound, notification to authorities etc.) and to run immediate countermeasures (i.e. drive up a protection wall when water level is high). In addition, an application with a user interface was built for monitoring the status of current and possible future events as well as examining the actions taken so

far for these events. The application also allows the users to customize properties of current and upcoming events (e.g. increase or decrease the risk level of the event) and to manually take actions to tackle current events.

2 System Analysis

In this section, we describe the user requirements of our system. The requirements are specified as either functional (e.g. **FR01**) or non-functional (e.g. **NFR01**) and are prioritised to be either mandatory or desirable.

2.1 Mandatory Requirements

FR01
As a local resident, I want to be alerted about current or possible future events having a high-risk so that I can plan preliminary personal actions to take.

FR02
As a local authority, I want to be alerted about current or possible future events having a high-risk so that I can plan public actions to take.

FR03
As a local fire department or emergency service, I want to be alerted about current or possible future events having a high-risk so that I can evaluate and plan rescue actions to take.

FR04
As a safety manager, I want to view the status of current and possible future events in a visual user interface so that I can monitor the weather and consider countermeasures to take.

FR05
As a safety manager, I want to see which alerting actions the system performed automatically to alert local residents, authorities or emergency services about current events so that I can consider which additional manual actions still have to be made.

FR06
As a safety manager, I want a protection wall to drive up automatically in front of the water in case of a high sea level so that flooding of the surrounded area is prevented.

NFR01
As a safety manager, I want the user interface to be simple and intuitive so that I can quickly monitor events and plan actions.

NFR02
As a local resident/ authority/ fire department/ ermergency service, I want the notification about current and possible future events to be clearly perceptible and meaningful so that I can quickly sense the important information.

2.2 Desirable Requirements

FR07
As a safety manager, I want to be able to see the sensed values of all sensors over the last few hours/ days/ weeks so that I can manually review the current developments.

FR08
As a safety manager, I want the system to also sense current weather events based on posts on social media so that human-based IoT data is also considered next to the sensory data.

FR09
As a safety manager, I want the system to also sense data about current and future weather events based on a public weather forecast API so that software-based IoT data is also considered next to the sensory data and human-based data.

FR10
As a safety manager, I want to be able to increase or decrease the risk level of an event so that the automatic risk classification made by the system can be further improved manually.

FR11
As a safety manager, I want to be able to manually drive the flooding protection wall up or down so that I can take precautionary actions even if there is no immediate risk of flooding.

3 System Architecture Design

In Fig. 1, we show the design of the architecture of our system. We took the slide 12 and 21 from the *System Design* slides as template for our architecture design.

4 System Implementation

4.1 Source Code

The software side implementation of our final system can be found under the following URL: `https://github.com/GirishSengerVerma/SCIoT`

4.2 Parts of our System

Our final system consists of multiple parts that we will describe in detail in the following sections:

- **Physical and Simulated Locations**
- **MQTT Broker**
- **PostgreSQL Database**
- **Sensor Simulator**
- **Dashboard Web App**
- **AI Planner**
- **Authorities Notification and Response Bot**

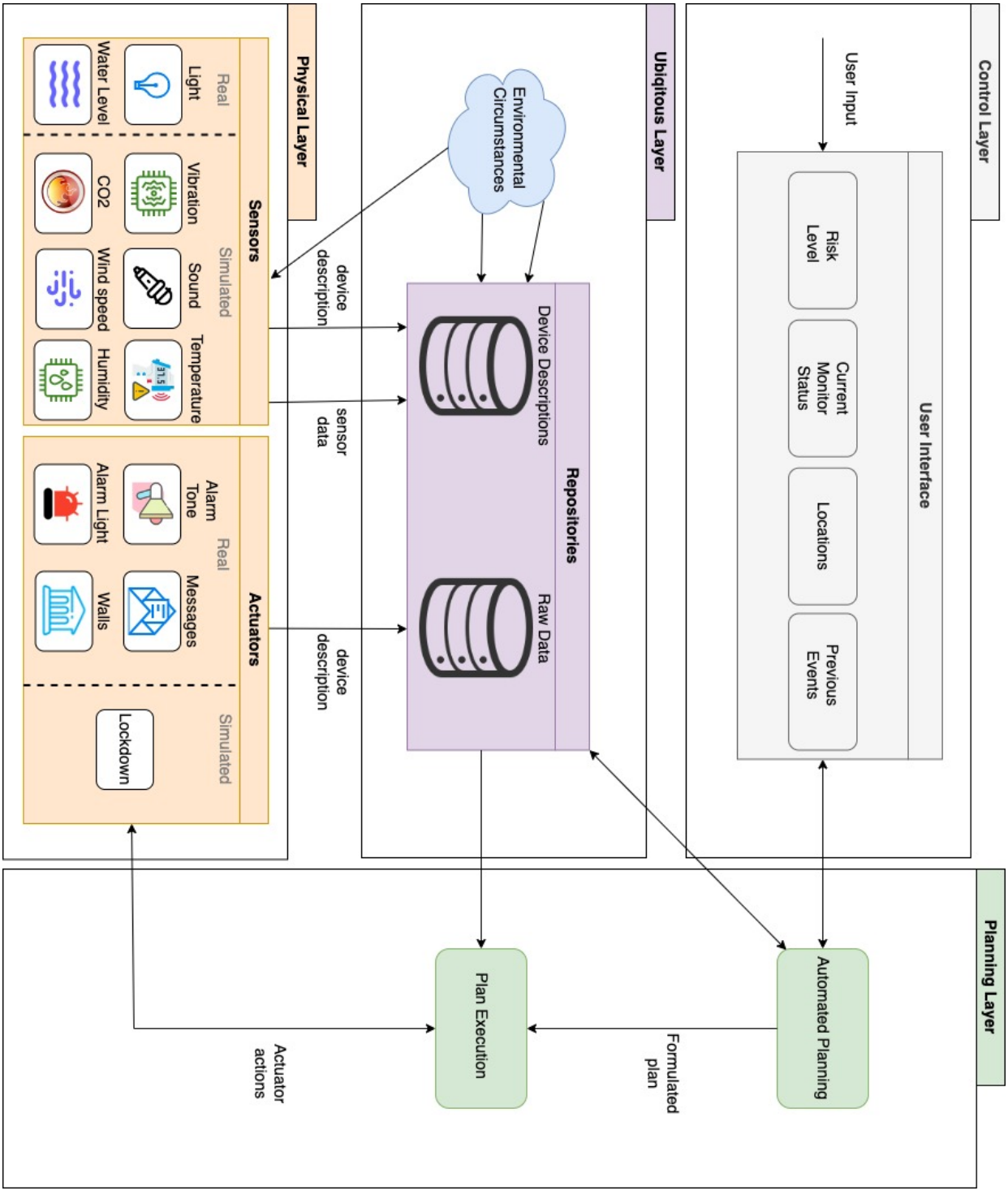


Fig. 1: System Architecture Design Diagram

4.3 Physical and Simulated Locations

In our system we included three different locations that each represent different types of nonresidential buildings in a smart city where different kinds of weather events can occur. The locations represent actual locations in Stuttgart. Namely, we decided on the three locations **Stuttgart Killesbergpark**, **Stuttgart Vaihingen Office** and **Stuttgart Max Eyth See** which we will describe in the following. In addition to these locations where weather events can occur, we added another location called **Authorities Hub** where police, fire trucks or ambulances can refill and reequip when they are not currently on duty.

Stuttgart Killesbergpark represents a publicly accessible forest and park area where lots of people come to in order to take a walk or just to relax. In such a location, the biggest immediate risks for severe weather events are due to wild fires and thunder storms which both represent heavy dangers to the safety and health of the people nearby. For detecting such events, we mostly use simulated sensors (as we will explain later on) together with a physical light sensor to detect thunder. Besides, alerting visitors at such a location about current events is hard as trees block sight all the time. Thus, we decided to not only include a physical alarm light actuator, but also a physical alarm sound actuator as an effective way to alert locals nearby. In addition to these physical sensors and actuators, we also added a simulated actuator that indicates a lockdown of the location. In case of a lockdown, no people should be allowed to enter the location anymore to lower the risk of health and safety violations.

Stuttgart Max Eyth See represents a publicly accessible lake which is also commonly used for hanging out and relaxing. Here, the biggest risk for a severe weather event is due to the lake. In case of a rising water level, floods can occur which can not only destroy the surrounding landscape, farms and villages, but also pose an immediate risk for people visiting the lake. For measuring the water level at the location, we used a physical water level sensor. Further, we decided to use an alarm light as an actuator to notify people nearby about high risks for severe weather events. Besides that, we have another actuator representing a movable water protection wall. In case of a risk for a flooding event, our system drives up this protection wall to decrease the risk for an actual flood. When the danger is gone, the wall moves down again. Again, we also added a simulated lockdown actuator.

Stuttgart Vaihingen Office represents a fully simulated office building location. In this location we only have two simulated actuators, namely a simulated lockdown actuator and a simulated light actuator. At such a location, we have high risks for bad air due to high CO and CO2 concentrations which can pose a heavy risk to the health and safety of the people working in the office building.

Authorities Hub represents a fully simulated location in which units of the police, fire department or hospital can stay to refill and reequip when there are no current weather events where they have to be on duty.

Implementation of the Physical Locations In order to implement the hardware side of the project all we had was the Raspberry Pi with the GrovePi kit. However, because of the requirement mentioned above, the ESP8266 is far more suitable for the scope of our project. One reason for that is, that the ESP8266 inherently have Wifi on board. With that, we can easily connect the hardware with the MQTT Broker with the metadata.

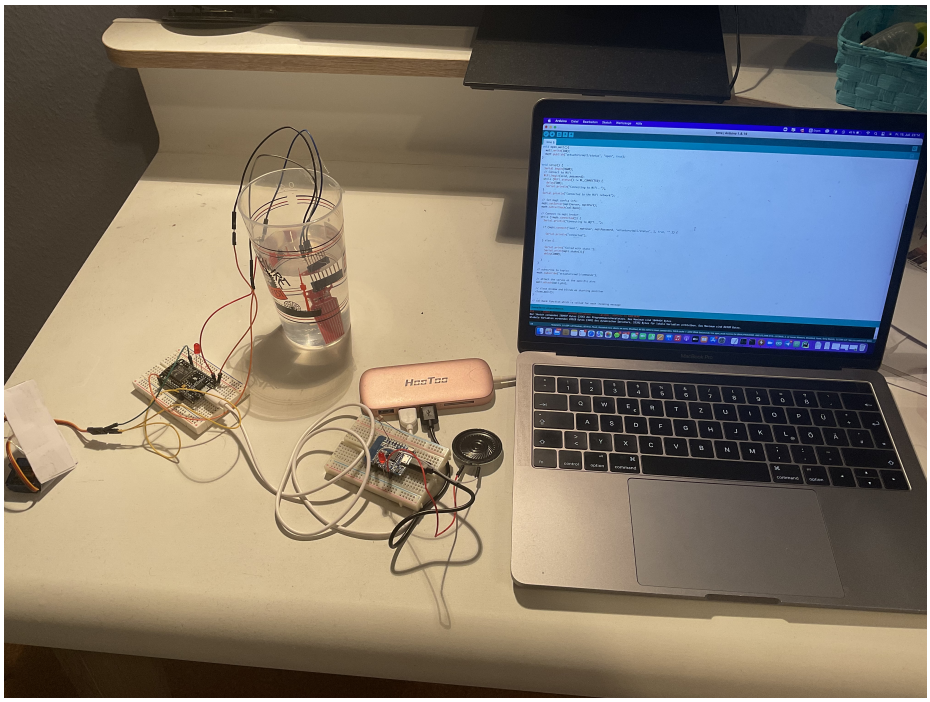


Fig. 2: Physical Side of the project without Casing

Figure 2 shows how we implemented the locations for our project. The location *Stuttgart Killesberg* and *Max Eyth See* are simulated with one ESP8266 per location. For the *Water Level* sensor we used the *DollaTek Water Level Sensor* which provides analogous reading when the sensor touches water. The *Water Level*- and *Light Sensor* constantly publishes their sensor value via MQTT which will be explained in the following section.

4.4 MQTT Broker

One requirement for the project was to use an async messaging communication. We decided to use MQTT together with the Eclipse Mosquitto MQTT broker. In order to be able to communicate with the broker easily from both our software

applications as well as the physical Arduino devices, we deployed this server on a private cloud server of ours. The task of this broker is to collect sent MQTT messages and to distribute these messages to the different clients that described on the topic of this message. Therefore the broker represents a central part of our application by making it possible for the different parts of our application to communicate with each other asynchronously (e.g. without the need for a synchronous connection between all parties).

4.5 PostgreSQL Database

In order to be able to view historic data in our web application, we deployed a PostgreSQL database on a private cloud server of ours. Whenever our application receives data about sensors, actuators, weather events (+ risks and actions) or units, it persists the data in this database. This makes it possible for the safety manager to use the Dashboard web app to get an insight into the history of sensor values, actuator status, past weather events, their risks, actions taken and authorities units movements. This allows the safety manager to efficiently analyze current weather occurrences and to plan adequate actions.

4.6 Sensor Simulator

For simulating the values of different types of sensors at different locations, we have implemented a standalone Python program. This program is able to simulate the values for the following types of IoT sensors:

- **Temperature** in °C
- **Wind Speed** in km/h
- **Humidity** in %
- **Air Pressure** in hPa
- **Vibrations** in Richter Magnitude Scale
- **CO** concentration in ppm
- **CO2** concentration in ppm

On startup, the sensor simulator program announces the metadata for all simulated sensors by sending our a MQTT message for each sensor. Additionally, periodically, the program generates a new value for each sensor and then for each sensors sends a MQTT message with this sensor telemetry data. The current values of all simulated sensors can also be observed in the CLI of the program (see **Fig.** 4). For generating the values, different sensor simulation modes and behaviours can be specified using the simple and intuitive CLI provided by the program (see **Fig.** 5). The different sensor simulation modes are:

- **Extremely Low**
- **Low**
- **Medium**
- **High**
- **Extreme**

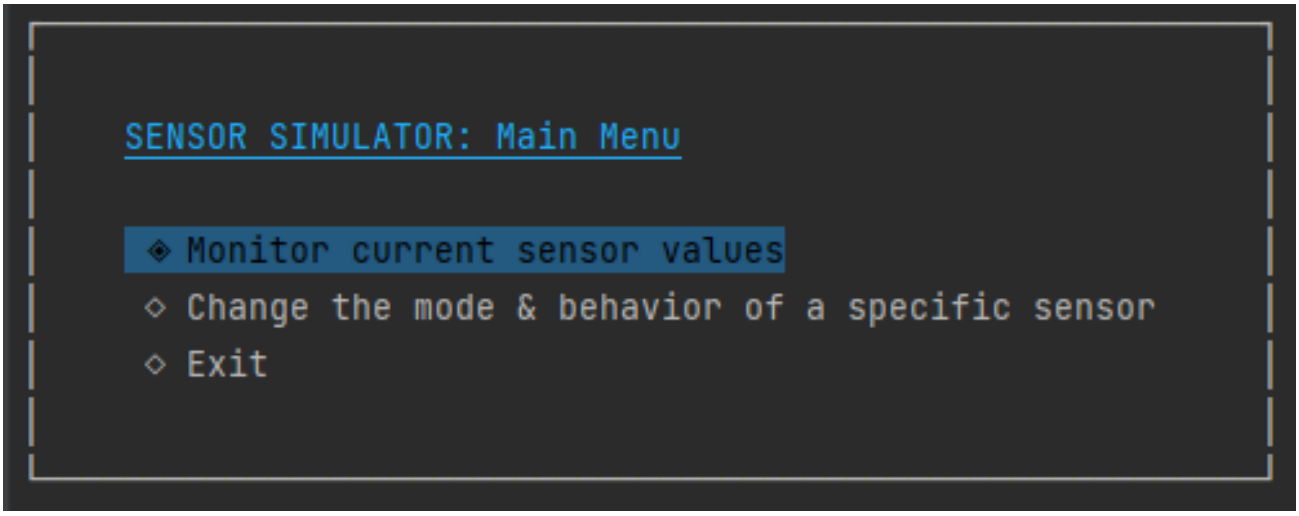


Fig. 3: Sensor Simulator CLI Main Menu

Fig. 4: Sensor Simulator Monitor Page

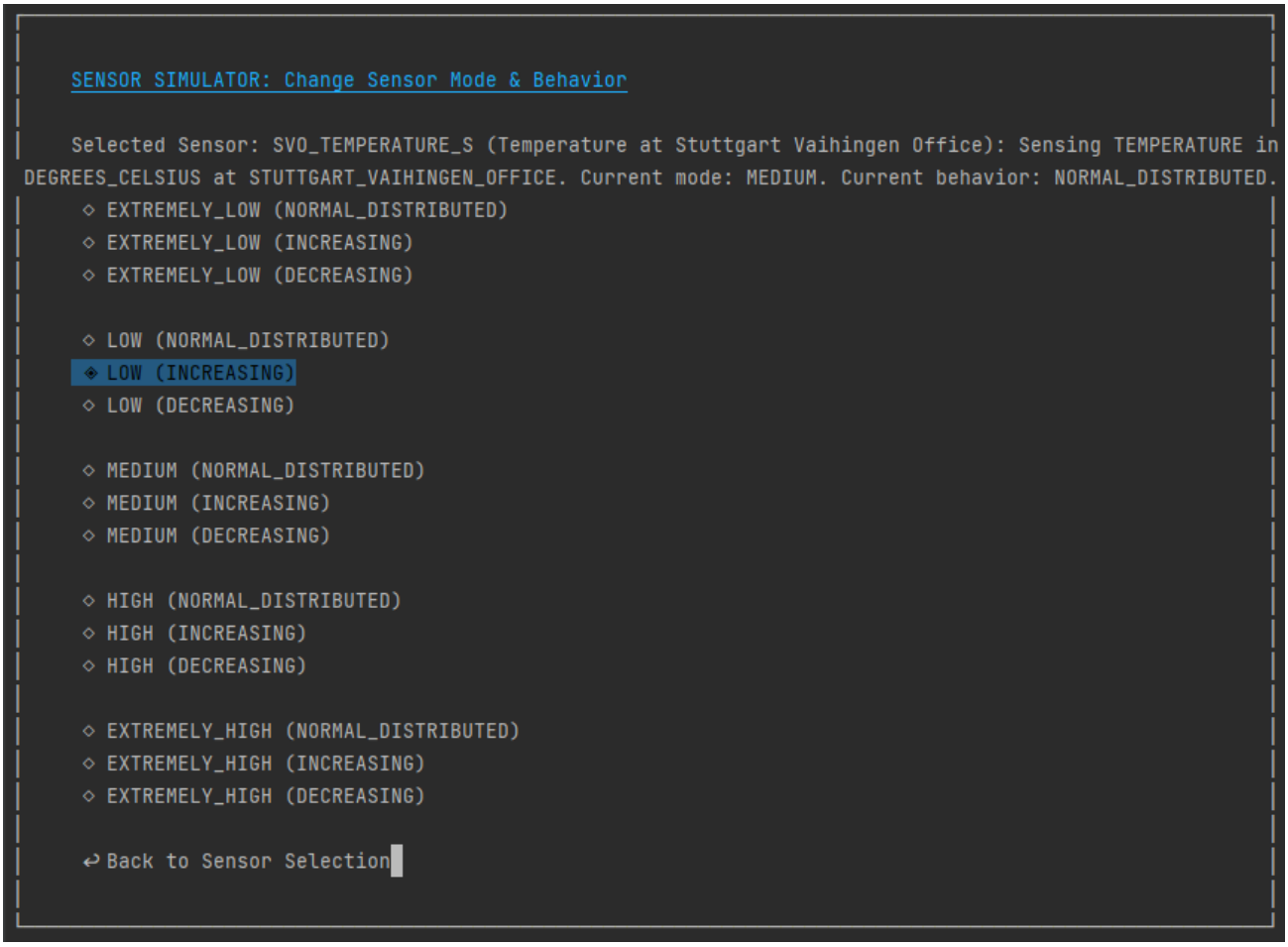


Fig. 5: Sensor Simulator Change Mode and Behaviour Page

and each represent a different range of values to generate from. These ranges differ depending on the type of sensor and are based on real world values. In example, a temperature sensor operating at 'Extremely Low' will produce temperatures under zero degrees Celsius to indicate snowy or icy weather. In addition to the simulation mode, the user of the program can also specify a simulation behaviour for each sensor. Namely, these behaviors are

- **Normal Distributed** to represent a normal distribution of values within the current range of values specified by the simulation mode.
- **Increasing** to represent an increase of the value over time from the lower end to the upper end of the the current range of values specified by the simulation mode.
- **Decreasing** to represent a decrease of the value over time from the upper end to the lower end of the the current range of values specified by the simulation mode.

4.7 Dashboard Web App

The central part of our system from the perspective of the user (i.e. the safety manager) is our Dashboard web application. Before we explain the features of the user interface, we first describe what the application does behind the scenes in order that allows us to monitor and plan in the first place.

Backend In the backend, the application listens to all incoming MQTT messages (on all topics), parses the contained data and persists the data in our PostgreSQL database. Furthermore, the backend holds websocket connections with all connected clients on the frontend in order to send and retrieve data between the client and the server backend. This allows us to update data in the frontend in real time and to process historic data requests coming from a client. Further, based on the current data, the backend first computes all current weather events and their risks in a preprocessing step and then uses this preprocessed information to generate the PDDL problem file for our AI planner, running the planner and then executing the actions proposed by the output plan. Thus, the Dashboard web application does not only provide a rich and extensive frontend for the safety manager, but also serves as an orchestrator who reads and persists data and then detects weather events and plans countermeasures for these events.

Frontend All in all, our goal for the frontend was to provide a simple and intuitive interface for the safety manager for monitoring and planning all kinds of data in our system. Therefore, we decided on a lean minimalist and consistent UI design that allows for an efficient interaction with the application. In addition to this, the UI is fully responsive so that the safety manager can not only use the application on a desktop computer, but can also quickly do monitoring and planning on mobile devices. Inside this application, the safety manager can monitor

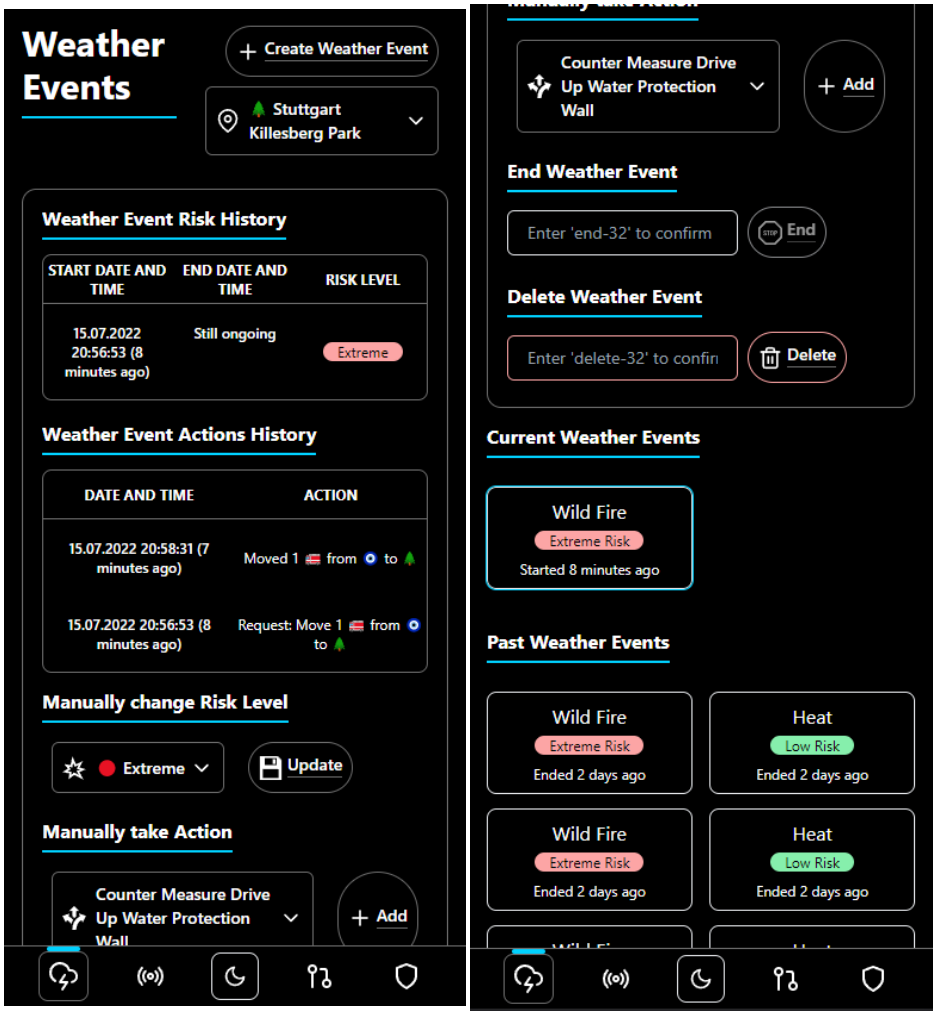
- current and past weather events at all locations
- current and historic sensor values at all locations
- current and historic actuator status at all locations
- current authorities unit positionings and history of movements

Weather Events In detail, on the weather events page, for each weather event, the safety manager gains insight into the type and current risk of the events. In addition to this, the safety manager can view the history of changes regarding the risk level of the weather event as well as see an overview of all actions taken so far (automatically or manually) to tackle the event. Furthermore, the safety manager can also manually change the risk level of an event, take actions (i.e. alert locals by light or sound or drive up or down the water protection wall) or end the weather event. Besides that, the safety manager can manually add weather events at a location and can delete old weather events that should not be shown anymore. Screenshot of the events overview page and create modal are shown in **Fig.** 6, and **Fig.** 7.

Sensors On the sensors page, the safety manager can monitor the current values of all sensors at all locations. By selecting a sensor, the safety manager can also see a live data chart for the development of the sensor values. In addition to the live data, the safety manager can also see historic sensor data (i.e. for the last hour, day, week or month). For the historic data, the safety manager sees the trend for the selected time period (i.e. sensor value decreased by 20% in the last hour) and can also see and navigate through the data development chart. As with the weather events, the safety manager can also manually create additional sensors and delete sensors that are not needed anymore or do not exist anymore. Screenshot of the sensors overview page and create modal are shown in **Fig.** 8 and **Fig.** 9.

Actuators On the actuators page, the safety manager can monitor the current status of all actuators at all locations. After selecting a specific actuator, the safety manager can then see the history of status changes for the actuators. Furthermore, he can manually change the actuator status and can optionally link this status change with an action for a specific weather event. Furthermore, the safety manager can again create additional actuators or delete actuators that are not needed anymore. Screenshot of the actuators overview page and create modal are shown in **Fig.** 10.

Authorities On the authorities page, the safety manager can view the current positioning status for all units (police cars, fire trucks and ambulances) at all locations. In addition to the amount of units of each type currently positioned at the selected location, the safety manager can also view the history of unit movements for the selected type at the selected location. Furthermore, the safety manager can manually move units from one location to another and can optionally link the movement with an action for a current weather event. A screenshot of the authorities overview page is shown in **Fig.** 11.



(a) Overview 1

(b) Overview 2

Fig. 6: Dashboard Web App - Events Overview Mobile

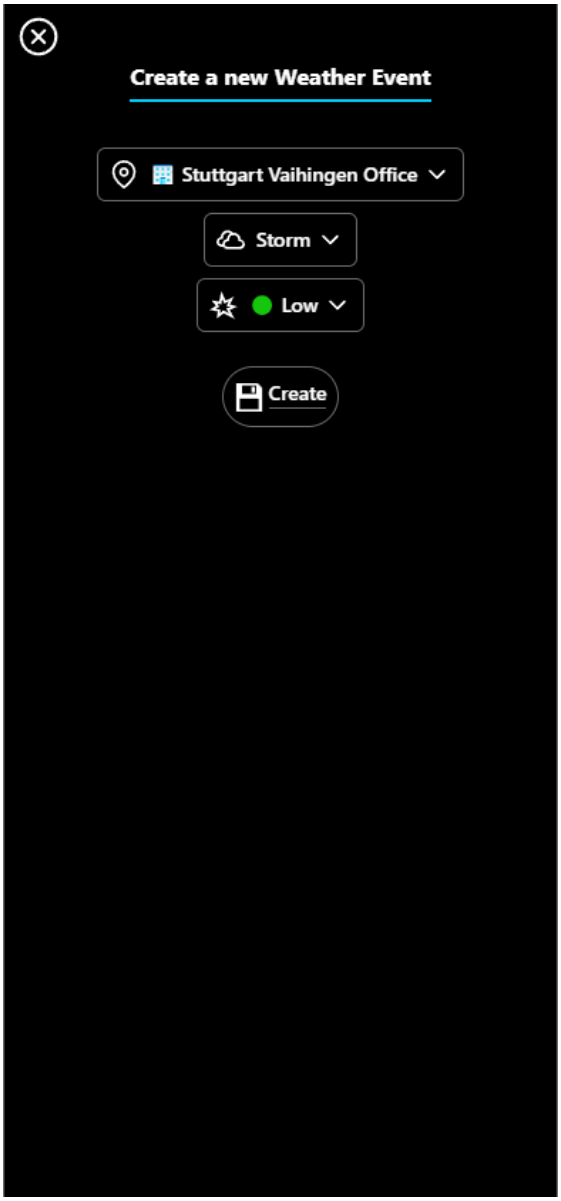


Fig. 7: Dashboard Web App - Events Create Modal Mobile

(a) Live Data (b) Historic Data

Fig. 8: Dashboard Web App - Sensors Overview Page Mobile

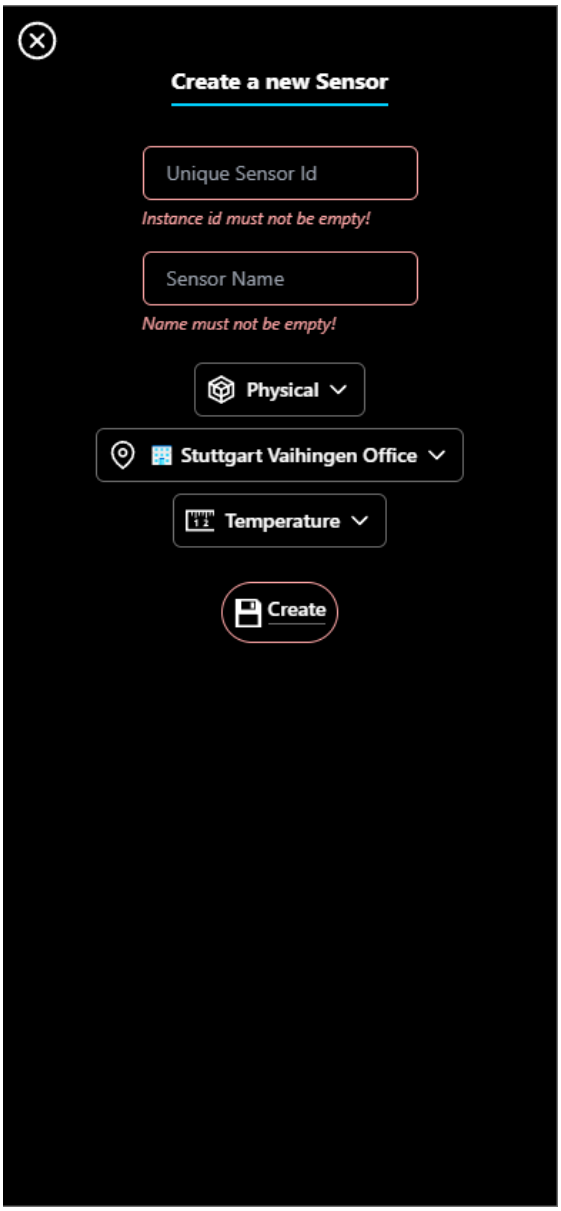
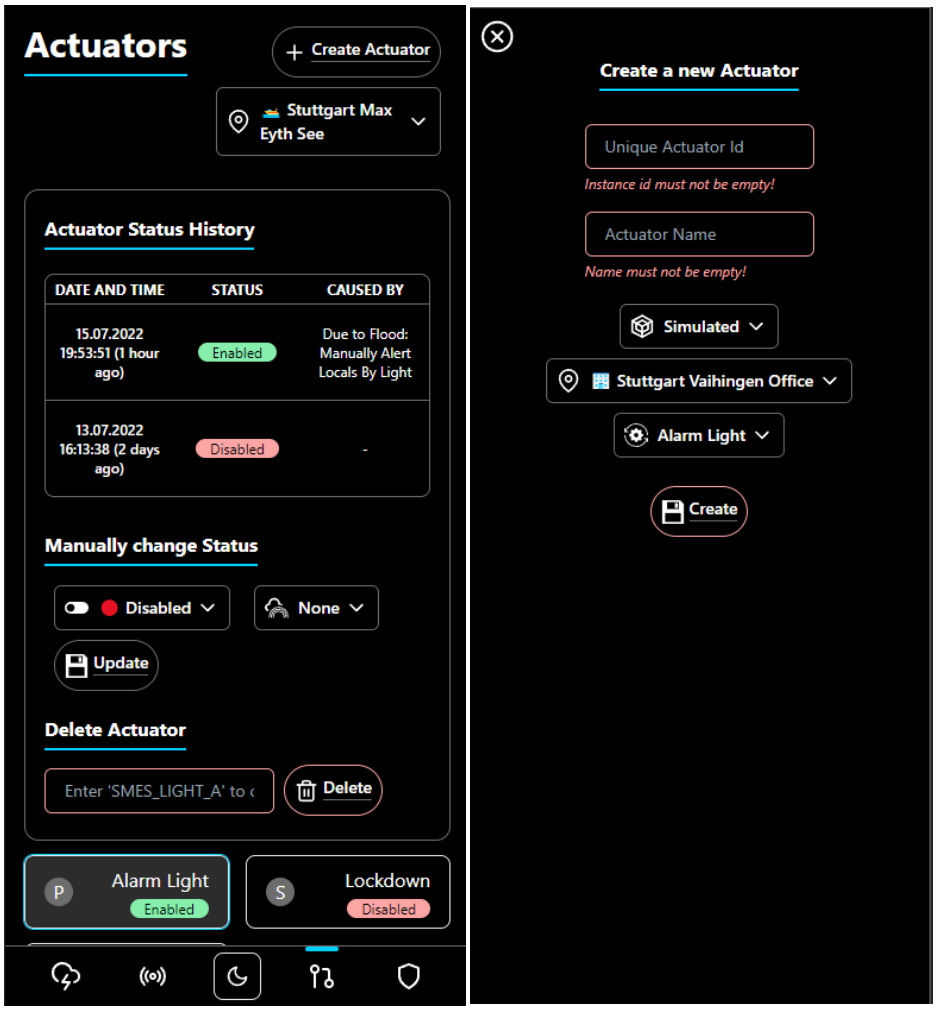


Fig. 9: Dashboard Web App - Sensors Create Modal Mobile



(a) Overview (b) Create Modal

Fig. 10: Dashboard Web App - Actuators Overview Page Mobile

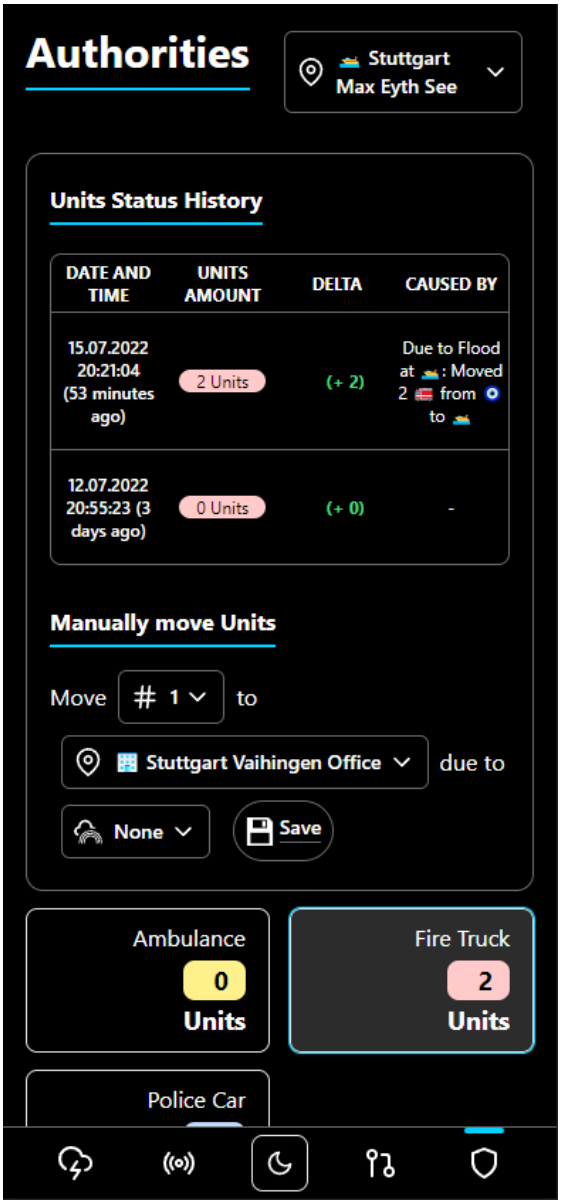


Fig. 11: Dashboard Web App - Authorities Overview Page Mobile

4.8 AI Planner

In a preprocessing step, the dashboard web application periodically determines current weather events at all locations and assigns them their current risk levels based on the values of our sensors and the status of our actuators. This preprocessed data together with other metadata of our system is then used to generate the problem file for our AI planner. The task of the AI planner then is to determine, which types of units and how many of each type should be positioned at each location in order to tackle all current weather events in the most efficient way. This is done using a metric that represents the sum of all current risks for all weather events. Each action performed by an authority unit at a weather event location decreaes this risk. The goal then is to minimize this metric. Furthermore, when a unit is positioned at a location where all weather events have ended, the unit should be rewarded for moving back to the authorities hub (i.e. for refueling and reequipping).

4.9 Authorities Notification and Response Bot

Whenever the AI planner has decided that a certain number of authorities units should be moved from one location to another, our dashboard backend sends a move units request MQTT message to our Authorities Telegram Bot. This bot displays the request to move the units inside the chat and allows the authorities to answer back whenever a certain number of units was indeed moved. The answer is again sent using a MQTT message. The bot therefore allows for quick and easy interaction between our system and local authorities by alerting them about current weather events and requesting them directly where to move units to. Thus, the authorities themselves do not need to manually plan where to move a certain number of units anymore. Instead, this work was already done by our AI planner. This time-saving effect can be crucial in the real-world for ensuring the health and safety of people near severe weather events where every second matters and therefore describes an efficient modern solution to an old problem. Screenshots of the interaction with the Telegram Bot are shown in **Fig.** 12, **Fig.** 13, **Fig.** 14 and **Fig.** 15.

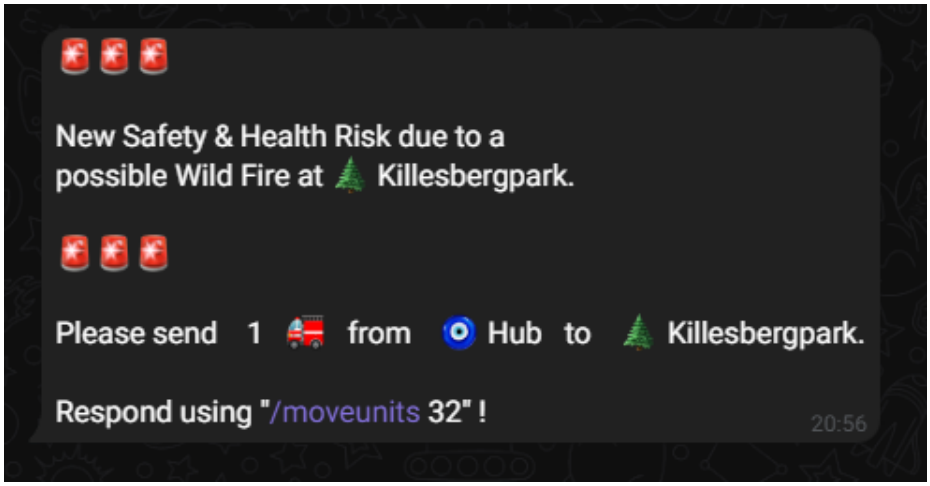


Fig. 12: Authorities Bot - Move Units Request

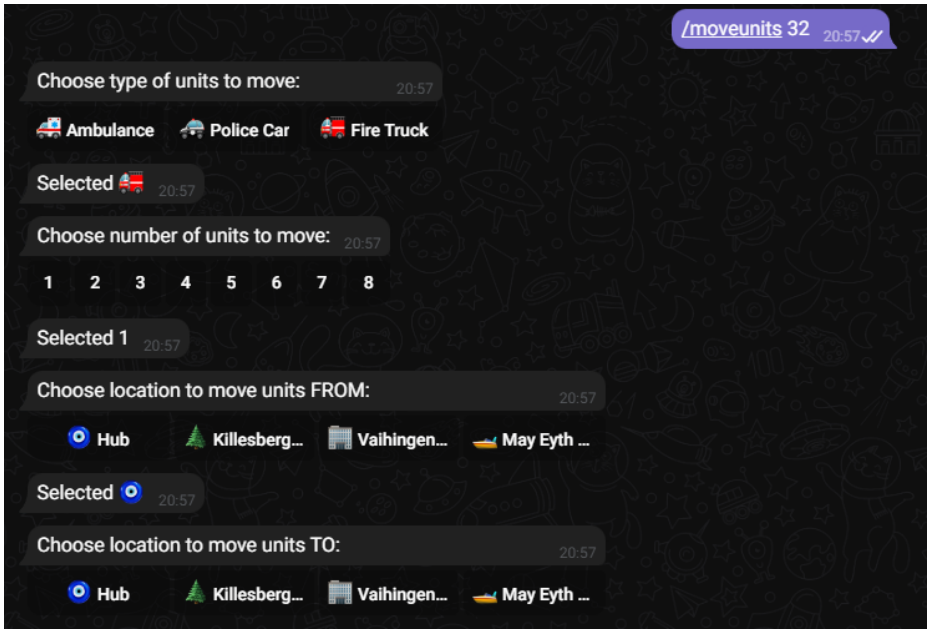


Fig. 13: Authorities Bot - Move Units Response

Fig. 14: Authorities Bot - Move Units Confirmation

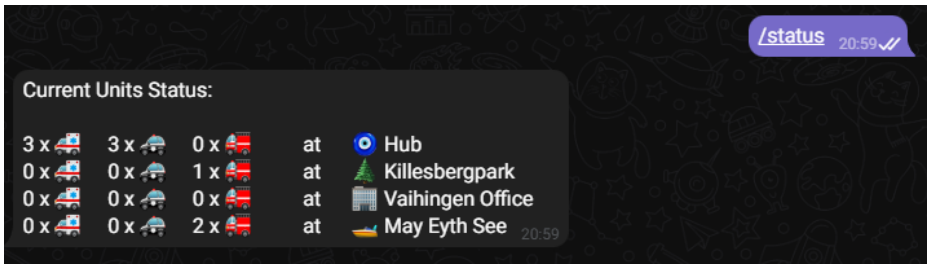


Fig. 15: Authorities Bot - Status

5 Discussion and Conclusion

All in all, this work describes a feature-extensive and very powerful system for efficiently detecting different types of weather events at different non-residential buildings and allowing to tackle these events using automatic and manual actions. Using this system helps the safety manager significantly to remain aware of current weather events, their risks and actions taken so far. This allows the safety manager to focus on the important manual actions needed to ensure the safety and health of all people living in our smart city.

Although our final system already is very extensive and powerful for detecting and tackling several kinds of severe weather events, there are some components included in the desired requirements for which we decided not to implement them due to the limited time we had for this project.

For instance, we did not include a simulated social media environment as we originally planned as a softgoal. In this environment, residents would be able to post about current weather events at their location and our system could then use this human based IoT input data for an even better detection of current weather events and their risks. Implementing this component into our system is a topic of future work.

Furthermore, our system allows to detect current weather events and to gain insights about previous weather events that have already ended. However, another idea was to use data from a public weather API in order to predict future weather events. We did not focus on this part of the system as it was not the main focus of this class. However, it would certainly be interesting to include even more real world IoT data into our system and to use this for monitoring and securing locations beforehand.

In addition to this, another topic of future work would be to extend the AI planner so that it performs planning on even more sub problems related to our system. One idea here would be to measure the number of people that are currently located at a given location and then provide this as additional input data to the AI planner. The AI planner could then use this additional data for a better distribution of available units to the different locations having weather events. This would illustrate an even more realistic model of weather events and tackling those.

References

1. Bronstert, A.: Floods and climate change: interactions and impacts. Risk Analysis: An International Journal **23**(3), 545–557 (2003)
2. Stott, P.: How climate change affects extreme weather events. Science **352**(6293), 1517–1518 (2016). https://doi.org/10.1126/science.aaf7271, `https://www.science.org/doi/abs/10.1126/science.aaf7271`

All links were last followed on May 2, 2022.