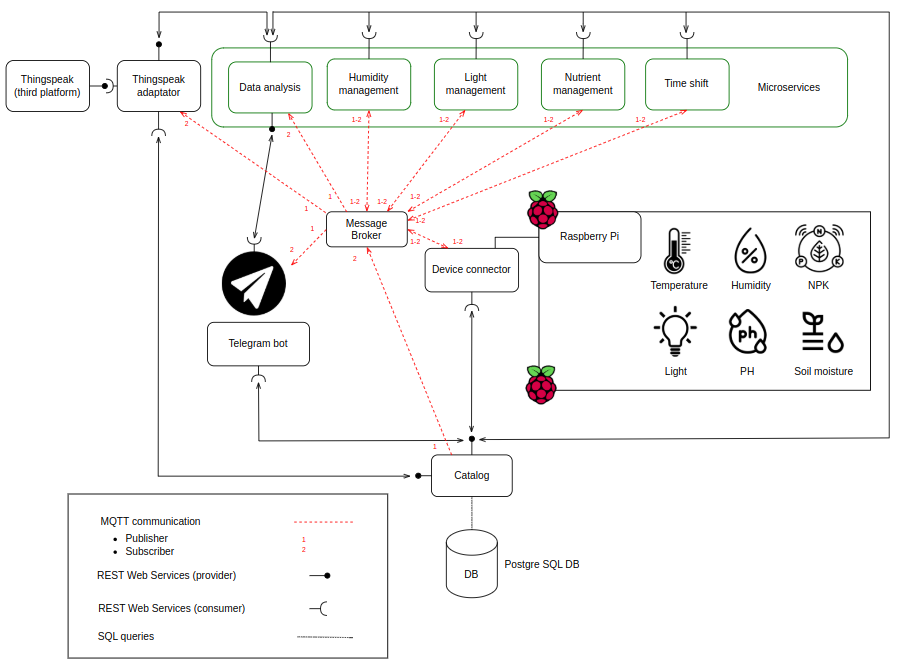
1. **Name of Use Case**

|  |  |
| --- | --- |
| **Name of the Use Case** | **IOT enhanced smart farming for plant monitoring** |
| **Version No.** | V1.0 |
| **Submission Date** | 06 December 2024 |
| **Team Members (with student ids)** | Guillaume CLOUARD s335346  Kevin George KOSMA PATABENDIGE DALPATHADU s338635  Lorenzo CAVALLARO s346742  Negar Abdi s343747 |

1. **Scope and Objectives of Function**

|  |  |
| --- | --- |
| **Scope and Objectives of Use Case** | |
| **Scope** | The IOT-Enhanced smart farming aims to simplify plant care by automating the essential functions such as watering and monitoring soil health that send information to the owner. This process reduced the manual intervention and optimize plant growth. |
| **Objective(s)** | 1. Monitor soil moisture, temperature, light and soil health level  2. Sends status and notifications on plant health via a Web application |
| **Domain(s)** | Smart Gardening, IOT agriculture |
| **Stakeholder(s)** | Home Gardeners, Urban Farming, IOT based greenhouse |
| **Short description** | The IOT based smart pot is equipped with sensors that can monitor the soil moisture and temperature plus the sufficiency of light. Besides that, the processes can be monitored by users via a web application. |

1. **Diagram of Use Case**

****

1. **Complete description of the system**

The proposed IoT platform for Smart Farming is designed using a microservices architecture and utilizes two communication paradigms:

1. **Publish/Subscribe** model based on the **MQTT protocol** to provide asynchronous communication.
2. **Request/Response** model using **REST Web Services** which provides the seamless interaction among system components.

**In this setup, the following key actors are defined:**

* **Postgre SQL DB** is used to store information about the resources and devices of our system. It contains these tables:
  + Sensors, (sensor\_id, *greenhouse\_id*, *plant\_id*, type, thing\_speak\_channel\_id, name, unit)
  + Greenhouses, (greenhouse\_id, *user\_id*, name, location)
  + Users, (user\_id, username, email, password\_hash)
  + Devices, (device\_id, *greenhouse\_id*, name, type, configuration)
  + Plants, (plant\_id, *greenhouse\_id*, name, species)
  + Scheduled events, ()

All the components of our system have to obtain, by passing through the Catalog, the information that they need from the DB.

* **Catalog** is the service that provides a REST API to the other components to permit to them to interact with the DB. All the queries and interactions to the DB are done by this service on request of a component of the system. To do so, catalog has some methods callable through HTTP request to a specific URI. In order to notify the microservices in case of modification of parameters (tresholds, desired intervals, scheduled events) made by the user, it must have an MQTT connection as a publisher to the message broker.
* **Device Connector** is the core of our system and it rappresents the SW in execution on the Raspberry Pi . Firstly, it exploits REST API of the Catalog to read from the DB the information about the system where it is working, to understand which sensors are connected to the system and which greenhouse it is handling. In order to be scalable, we want that each implementation of the system has its own device connector. So at each we give an ‘id’ by using the host name of the socket, this will be used to quary the DB and see if that device is known or not. If it is not, we can register it, if it is we can proceed with the reading and receiving of row data collected by the sensors, and then deploy these to the message broker by using MQTT as a publisher. These data will be sent to the microservices that finally will return the actions that need to be taken in order to keep the system stable. When the results is ready, the message broker will notify the device connector through an MQTT connection where this component is the subscriber.
* **Telegram Bot** is the point of interaction between the system and the final user. The user can identify itself by using some credentials, then it can use a token to connect itself to its system. The credentials and the tokens are stored in the DB, so the bot has to exploit the REST API of the catalog to read these . Once the user logged into the system, it can use some commands to interact and set the parameters as it wants. For example, it can change the desired intervals of values for each parameter that the system is handling, set new or delate scheduled events, see values and graphs on the collected data in a period of time. The previous data used for graphs are obtained from the data analysis microservice through an HTTP request to the REST API of that component.
* **Message Broker** is an external component that we use as a broker in our MQTT connections. Its address is ‘mqtt.eclipseprojects.io’ and the port is ‘1883’. Publishers in our system will send data with senML format to the broker associating them to a specif topic. Subscribers will ‘subscribe’ them self to the topics of their interest, then the broker will notify them when someone publish data of those topics.
* **Thing Speak** is a third part SW that we use as an additional DB to store recurrent data collected during time. We already have a DB based on Postgre in our system, but it is not convenient to use it for recurrent data. Thing Speak is a very useful implementation cause it handles data in JSON format and also permits to track trends of these during time. However, we can’t connect directly our system to a third part SW, cause it is out of our control. So we must use an adapter that beehives as an interpreter between our system and Thing Speak SW. Thing Speak provides a REST API where the adapter should connect to send or receive data.
* **Thing Speak adapter** is a component of our system that handles the interactions with Thing Speak SW. Through MQTT protocol it receives all the data collected by the sensors (it is a subscriber to all the topics of sensors) in order to save them on Thing Speak as a DB. It also provides a REST API to the data analysis microservice to send information about data in a period of time. Also the thing speak adaptor must be registered In the DB, so it has a connection to the REST API of the catalog.
* **Microservices** are the componets that have the aim of handling row data, analyze them and return as a result a consideration, that could be statistics, notifications or actions that need to be taken. They are all connected to the catalog for exploiting the REST API, so they can register them self on the DB and read from it the information about scheduled events, desired intervals of values for the observed parameters. Then they all use MQTT to receive data of their interest and also for sending commands and considerations. Our microservices are:
  + Data Analysis, as a subscriber it receives all the data collected by the sensors via MQTT and keeps statistics on them. With these it can also evaluate predictions and expectations. It exposes a REST API where the Telegram Bot can read these results or can request to see the data referring to a specific period of time, also by using plots made by this microservice. It gets the previous data from Thing Speak by an HTTP request.
  + Humidity Management, as a subscriber it gets via MQTT the row data of the air humidity and soil moisture. Given the decided interval of values for these parameters, it checks if the measured values are acceptable or an action is needed. Before taking a decision it gets by HTTP the likelihood of the observation that is handling, if the value has a low likelihood it waits to receive more values and only then decide if an action is needed. The actions to take and the notifications are sent via MQTT as a subscriber.
  + Nutrient Management, same as the humidity management but it handles Npk level, soil moisture and pH.
  + Light Management, same as the humidity management but it handles light and temperature values.
  + Time Shift, it takes care of notifying as a MQTT subscriber the components of a scheduled events that need to be executed. These notifications are received by the microservices that handles the decision of taking actions (Management components), that should check if the action should be executed or if it doesn’t. The scheduled events are saved on the DB, so this microservice reads them by exploiting the REST API of the catalog. We must handle when the user apply some modifications on these events saved into the DB. For doing so, time shift should be also a MQTT subscriber, so it can be notified from the catalog of some changes.

1. **Desired Hardware components (only among those we can provide)**

|  |  |  |
| --- | --- | --- |
| **Device Name** | **Quantity** | **Needed for…** |
| Temperature sensor | 1+ | Monitoring and regulating temperature level |
| Humidity sensor | 1+ | Monitoring humidity level |
| NPK sensor | 1+ | Monitoring nutrients present in the soil |
| Soil humidity sensor | 1+ | Monitoring soil moisture level |
| pH sensor | 1+ | Monitoring pH of the soil |
| Light sensor | 1+ | Monitor natural or artificial light |
| LEDs | 1+ | Regulate artificial lightning, ensuring optimal light exposure for growth. |