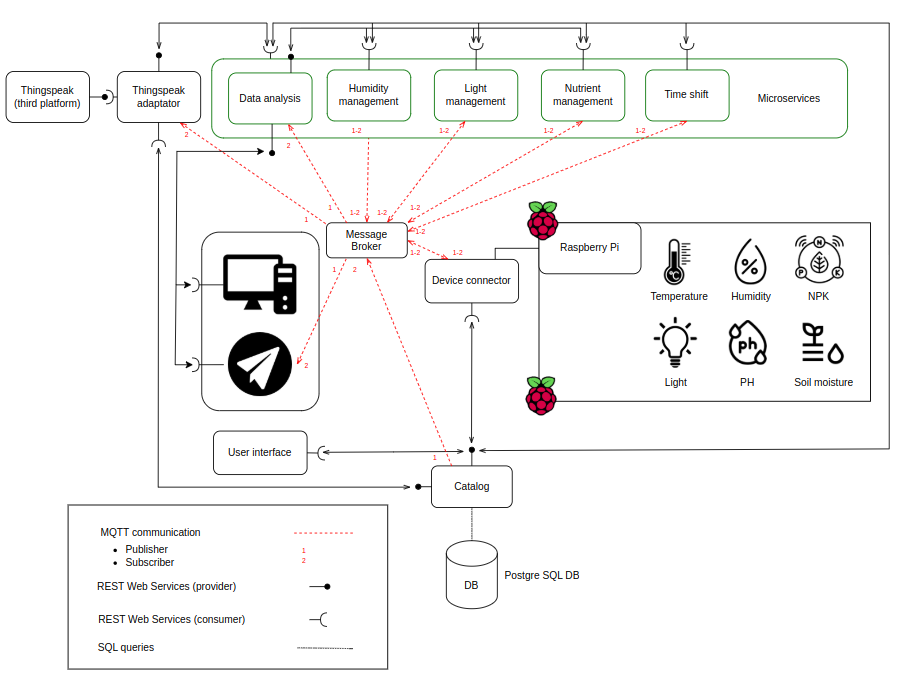
1. **Name of Use Case**

| **Name of the Use Case** | **IOT enhanced smart greenhouse** |
| --- | --- |
| **Version No.** | V1.0 |
| **Submission Date** | 13 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 greenhouse 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 reduces the manual intervention and optimize plant growth. |
| **Objective(s)** | 1. Monitor soil moisture, soil pH, NPK level, light, air temperature and humidity  2. Sends status and notifications on plant health via Telegram  3. User can set scheduled events and system’s parameters through a Web Application |
| **Domain(s)** | Smart Farming, IOT agriculture |
| **Stakeholder(s)** | Home Gardeners, IOT based greenhouse |
| **Short description** | The IOT based smart greenhouse is equipped with sensors that can monitor air temperature, humidity and amount of light inside the greenhouse. Then it also collects data about each plant as NPK level, soil moisture and soil pH. These data are processed in order to automate the system and send notifications and data to the user. |

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, name, unit, treshold\_range, thing\_speak\_field)
  + Greenhouses, (greenhouse\_id, *user\_id*, name, location, thing\_speak\_channel\_read\_api\_key, thing\_speak\_channel\_write\_api\_key, accessToken)
  + Users, (user\_id, username, email, password\_hash)
  + Devices, (device\_id, *greenhouse\_id*, name, type)
  + Plants, (plant\_id, *greenhouse\_id*, name, species, thing\_speak\_channel\_read\_api\_key, thing\_speak\_channel\_write\_api\_key)
  + Scheduled events, (event\_id, *greenhouse\_id*, event\_type, start\_time, end\_time, frequency, status, created\_at)

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 them to register and get configuration. 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 (thresholds, desired intervals, scheduled events) made by the user, it is also an MQTT publisher.
* **Device Connector** is the core of our system and it represents the SW in execution on the Raspberry Pi . Firstly, it exploits the REST API of the Catalog for reading from the DB the information about the system where it is working, so it can 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. Each device is associated with a json configuration file that contains the address of the MQTT broker and the id of the device. This will be used to query the DB and see if that device is known or not. If it is not, we can register it, if it is, it reads the sensors that it must handle and then we can proceed with the reading and receiving of raw 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 plants in the best conditions. These actions will be received by the device connector through an MQTT connection as a subscriber.

* **Telegram Bot** is a 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 exploit some commands to see simple information about previous data and events. The main task of this component is to receive notifications from the system through MQTT connection as a subscriber. In this way the user can be notified of happening events by the management components.
* **Web Application** is used to permit the user to have access to detailed graphs of previous data collected by the system and stored in ThingSpeak. It asks and receives these graphs from Data Analysis microservice by exploiting its REST API. The user can also use the Web Application for setting and changing scheduled events and parameters of the system, as accepted range of values for a specific parameter (humidity, soil moisture, …)
* **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 specific 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 because it handles data in JSON format and also permits to track trends of these during time. However, we can’t directly connect our system to a third part SW, cause it is out of our control. So we must use an adapter that behaves 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 adaptator** 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. 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 adapter must be registered In the DB, so it has a connection to the REST API of the catalog.
* **Microservices** are the components that have the aim of handling raw data, analyze them and return as a result a consideration, that could be evaluations on the state of the system, notifications or actions that need to be taken. They are all connected to the catalog for exploiting the REST API, so they can register themselves on the DB and read from it the information about scheduled events and 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. When it receives a value, it uses this to update the statistics and then it forgets about the received value. So it doesn’t need to store received values, but it evaluates the statistics live. With these it can also evaluate predictions and expectations (For example computing the mean and when a new value comes in verify it doesn’t differ too much from the mean, if not it might be an error). It exposes a REST API where the Telegram Bot and the Web Application 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 raw data of the air humidity and soil moisture. Given the decided interval of values for these parameters, it checks if the measurements are acceptable or an action is needed. Before taking a decision it asks to Data Analysis microservice the deviation of the observation compared to the mean by using an HTTP request. If the value has a high deviation it waits to receive more values and only then decides 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 the system as a MQTT subscriber of 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 at the begin this microservice reads them by exploiting the REST API of the catalog. We must handle when the user applies some modifications on these events saved into the DB. For doing so, time shift should also be a MQTT subscriber, so it can be notified from the catalog of 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. |