SmartPlant: an automatic plant care-taking service

January 6, 2020

Ella Velner (11360666) Victor Koppen (12382825) Erik Dolstra (11927429)

Jordy Tjon Sjoe Sjoe (12399868) Lennart de Vink (12365548)

ABSTRACT

The care taking and maintenance of a plant is often considered as a task that consumes considerable time. However, the advancements in the area of Internet of Things (IoT) and agriculture may help reduce the investment of time needed for agricultural care. This paper presents "SmartPlant", which assists users in taking care of their plants through accessible insights on a web application and automates several tasks, e.g. watering of plants. In the development process of SmartPlant, three other services have been used, which include: sensor data, a hosting provider and a plant database called Trefle.io, which is accessible through an API. A variation of sensors have been employed to retrieve data such as soil moisture, temperature and water levels. We delivered a prototype and provided a description for the remaining necessary technology needed to create a scalable solution that allows novice users to practice indoor gardening with no prior knowledge in botany.

1 INTRODUCTION AND MOTIVATION

Advancements in IoT devices for agriculture do not only make the industrial and agriculture area more efficient, but also the homes of people [16]. The maintenance and care taking of plants is often a time consuming job, but has been shown to relieve stress and other mental health problems [21]. Knowledge about ensuring the right growth, nutrients and the right amount of moisture is often absent by people [14]. The goal of this paper is to develop a self-sustaining smart plant which is accessible through a web application, called "SmartPlant". SmartPlant supports individuals in maintaining their indoor plants, which is especially interesting for novice users in the gardening community, such as children and people with little time. SmartPlant helps one to keep track of soil humidity, air humidity, temperature, brightness, and the water level of the water reservoir. This brings valuable data, collected by the various sensors, to the user which assists them in the care taking of their plant.

This paper is written from an information science perspective. All knowledge about botany is gathered from external sources. This allows for a paper focused on creating an IoT service for plants. Online there are many guides on how to gather sensor data from one plant, such as [7] [1]. However, the knowledge on how to create a network of plants or create a sustainable service for automated gardening seems to be missing. Thus, to fill this gap we detail the infrastructure of an automated indoor amateur gardening service. To design a plant care-taking service that can be used by anyone, we created the following research question:

How can we design a service using technologies from information science that allows users to do indoor gardening with little to no prior knowledge in botany?

As a proof of concept we have built a service that can accept SmartPlant data and view this for a user in a web application that is hosted on the server of an external party. The rest of this paper details the steps made to built this basic version of SmartPlant, and discusses how this basic version can scale to a commercial service.

SmartPlant works as a service, where the user can buy a plant care-taking program that expires after it's used. Today's services are often built upon other services [5]. SmartPlant is no exception to this. For SmartPlant three other services are used: sensor data, a hosting provider and a plant database which is accessible through an API. These services are implemented in different ways. The implementation of sensor data is described in 4.2: sensors. The implementation and selecting a hosting provider is described in 4.3: servers. Lastly, the implementation of a plant database is described in 4.4: Data gathering.

2 RELATED WORK

Plants that send an e-mail when the humidity reaches below a certain level have been developed by [14]. The researchers connected a sensor that measures the humidity of one plant and programmed the software to mail one or more users when the humidity was low. The application sends one or various notifications, telling the user what the plant needs. According to the researchers the user feels more connected to the plant and creates emotional bonds by interacting with its plant. They also mention how they could measure the happiness, irritation or sadness of the plant by measuring the soil.

Researchers [18] developed an online garden community where people could get advice and information about gardening. They developed an interactive online platform that provides community groups and smart gardening advice where people could connect and interact with each other. One could model their water tank and monitor its behavior under different circumstances. For example, simulate the most efficient way of determining the size, area, soil and watering for the plant owned by the user. However, they do explicitly mention that technology is the last thing people think of when gardening. Thus, gardeners could propose to be a challenging group to reach.

The advancements made in the area of augmented reality do not go unnoticed in the context of smart gardening. Okayami and Miyawaki [16] developed a smart garden through the use of augmented reality (AR). They aimed to guide home gardeners with gardening activities and at the same time measure the positions and viewing point of the user. By applying AR the gardener could be guided with determining the location where to plant the seed or plant. The gardener also got information about the distance between each plant and would be informed about how large the plant will eventually grown, because people that are new to gardening do often not know how certain plants would grow.

To make smart gardening a working product, sensors are necessary to be implemented to track things like humidity, amount of sunlight, and water needs. For example, [2] looked into a smart irrigation system to automatically water plants, using a Wireless Sensor Network (WSN). They used soil humidity sensors and electrovalves to track when the plant needed water and to give it water when needed. These were programmed with TinyOS and were then sent to a Java application with the use of a MySQL database. The system was connected to a PC via USB. However, to make it fully wireless, without the USB connection to the PC, either WiFi or Bluetooth is necessary. Because of its broader bandwidth and flexibility WiFi is the better option. Another option is WiMAX, but this is much less compatible and not very often used in devices for consumers [8]. The sensors will then not be connected to a PC, but to another microcontroller or microprocessor, like an Arduino UNO, Raspberry Pi, or a NodeMCU. Advantages of the NodeMCU are that it has a built-in WiFi module, unlike the Arduino UNO, and it is cheaper than the Raspberry Pi. The NodeMCU is widely used in IoT services with sensors [20]. For example, the ESP8266 (the WiFi module on the NodeMCU) has been used before to make an irrigation system. Singh & Saikia [22] created this system with multiple sensors, like a water flow sensor and soil moisture sensor, to automatically water multiple crops for farming.

According to Arcuri [3], building exploitable online services is often done using a Micro-Services architecture. Hyper text Transfer Protocol (HTTP) is a protocol that can be used to have Micro-Services communicate over a network. HTTP uses a set of commands read, write and update values over the web. Microsoft [15] provides a guidelines to create a RESTful Application Programmable Interface (API), that uses these HTTP commands to create an easy to use API.

2.1 Competition in the market

Some of the major players in the smart gardening market are Plantui6, AeroGarden, and Click & Grow. According to Oprea and Alexandra [17], the products and services of these companies have the following characteristics.

Platui6 uses hydroponic growing (growing plants in water instead of soil), uses "combined white and red LED-light for greater efficiency" (p. 14) and does not come with an application. AeroGarden has the same characteristics. The only difference between AeroGarden and Plantui6 is that Aerogarden does have a mobile application, called AeroGarden Wi-Fi. Last, Click & Grow uses soil to grow the plants. The company promises an optimal soil solution. Their plants are bought as cups. However, Click & Grow doesn't come with an LED lamp or a mobile application.

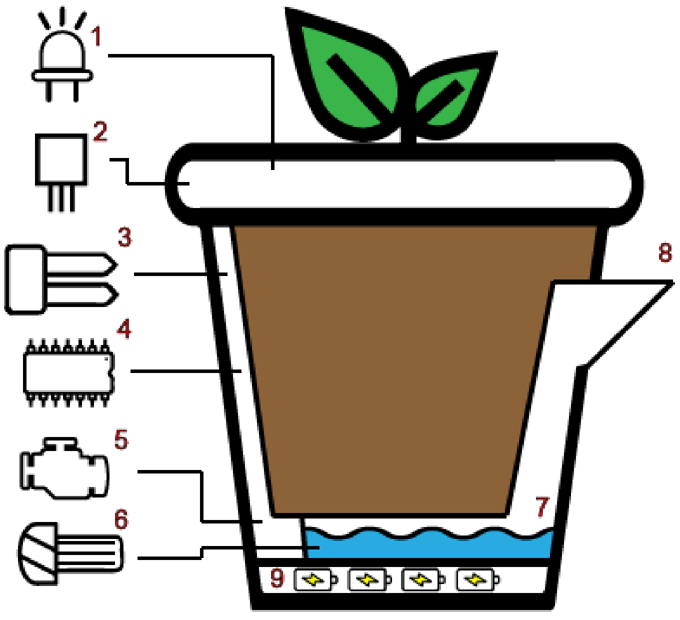
3 INTERACTION DESIGN

This section describes the interaction between the actors and interface of the system. During this report we will discern between SmartPlant, SmartPlant Pot and the SmartPlant application. When we refer to SmartPlant we mean the service as a whole. The SmartPlant Pot is the actual plant pot, which is described in subsection . Lastly, the SmartPlant is the application with the interface where users of SmartPlant can manage their plant. The SmartPlant application is described in subsection 3.2.

3.1 Actors

There are three actors in our system: the user, the SmartPlant Pot and the administrator. In the following sections the role and activities of these users will be discussed.

3.1.1 The User. The user is the consumer that bought a SmartPlant (or more than one) and now wants to track how their plant is doing via the interface. They first need to create an account on the SmartPlant application. After that they can login and visit the dashboard (as shown in figure 3). Here, the user can register their SmartPlant and specify the right plant program. Different plant programs can be bought in the store. When clicking on their registered SmartPlant, the user can see the outputs of the different sensors in the SmartPlant and if the water reservoir needs to be filled up (see figure 6. The user can also get notifications when their plant might flourish more in different circumstances, like having more sunlight.



Legend:	
1 = light sensor	5 = waterpump
2 = temperature and air humidity sensor	6 = water level sensor
3 = soil humidity sensor	7 = water reservoir
4 = NodeMCU	8 = water refill spout
	9 = battery pack

Figure 1: Schematic representation of the SmartPlant Pot

3.1.2 The SmartPlant Pot. The SmartPlant Pot has three integrated sensors that measure light, temperature, air humidity, soil humidity to analyse the environment of the plant (see figure 1). Furthermore, a water reservoir with a water level sensor is integrated in the Pot, which can be refilled by the user (for which they will get notified when necessary) via an easy-to-access spout. The sensors are described more in detail in section 4.2. The sensors are connected to the NodeMCU (also inside the Pot), which is powered by a battery pack to make it fully wireless, so the SmartPlant can be put anywhere in the room without the need of a nearby power plug. This battery pack is surrounded by a secure panelling to avoid leakage from the water reservoir. The NodeMCU is the central hub in the SmartPlant that sends the data from the sensors to the interface every three hours via WiFi (again, to make it completely wireless with no need of an ethernet cable). It also interprets the data it receives from the sensors and will activate the water pump to give the plant water when necessary.

3.1.3 The Administrator. The administrator has to perform several tasks, which when the service grows, can be automated. For these tasks the administrator can access pages, which are only accessible for user accounts with the role of administrator. The first task is that every time a new SmartPlant Pot is manufactured it must be registered in the database. The system should know which SmartPlant Pots there are and to whom they are assigned. Every new SmartPlant Pot gets a sticker on the bottom, with a unique serial number and password. Its the administrator's task that these SmartPlant Pots are added to the database. Adding these codes to the database can be partly automated, with the use of barcode scanners. The second task is that the administrator must run a script which loads the latest data from the Trefle.io API on a weekly basis. This task is further described in subsection 4.4. As the SmartPlant service grows, the task of the administrator will shift more towards a customer service perspective. The administrator can then help user's with technical issues in using the application and configuring the SmartPlant.

3.2 The Interface

A web application built on the world wide web provides a platform which is accessible through a web browser. This makes it an ideal platform, that could be reached from any location. Duan et al. [9] mentions how a application that is widely accessible would be a good foundation for applications and knowledge based systems. This is mainly due to the readability, portability and accessibility the web offers. However, one would require a stable internet connection to be able to connect to the web application. A smart device (e.g. smartphone, laptop, desktop or tablet) is required which is able to establish a connection to the internet using either WIFI, 2G, 3G, 4G, 5G or LAN (Ethernet)[1]. One of the advantages of developing a web-based application is that it removes the necessity to develop the application for different operating systems, since any smart device, as previously mentioned, possesses a web browser[2].

The final design and web-application is presented below. Figure 2 presents the login screen of the application. This is the landing page of each user that is trying to access the system, unless they are already logged in through a previous session.

When the user has an account he/she can login using their credentials, otherwise they have to register in order to create a account.

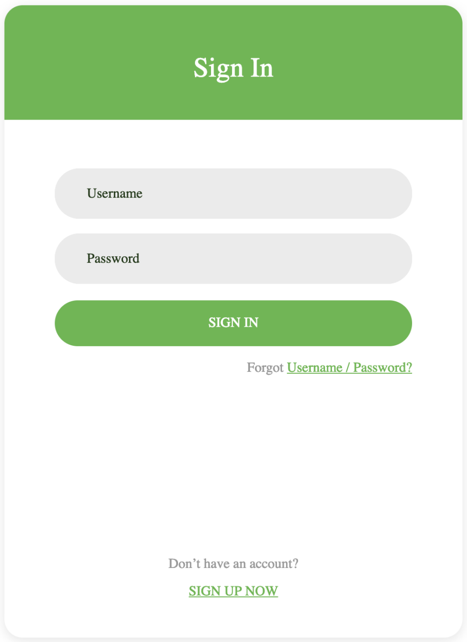
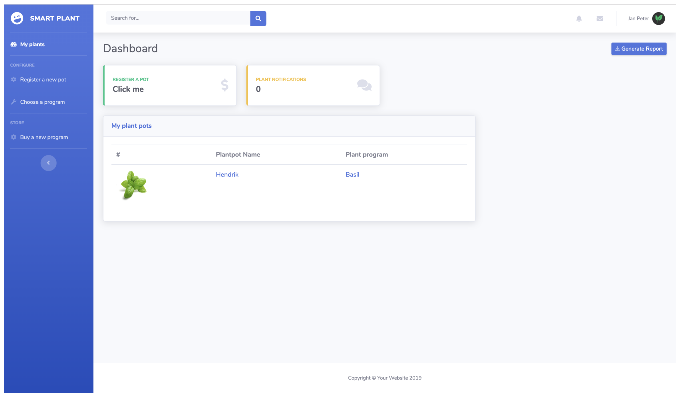


Figure 2: Login of the web-application

When the user has logged in they get presented the homepage as shown in figure 3.



#	Plantpot Name	Plant program
	Hendrik	Basil

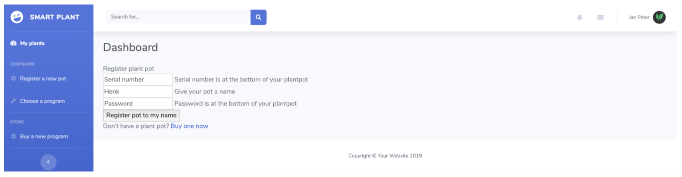
Figure 3: Homepage of the web-application

All SmartPlant Pots that have been registered will be presented on this page. However, the section "My plant pots" will present empty, when no plants have been registered. One has to activate their product first, which could be done by going to the "Register a new pot" page. On this page (see figure 4) one can activate their pot using the credentials given in the package of their purchase.

After the SmartPlant Pot has been registered one has to go to the "Choose a plant program" page to choose the specific program needed for their plant. New plant programs can be bought in the store.

In order to access the analytical information of the plant, one has to click on the name given, which is the plant "Hendrik" in this

[1]https://www.digitalunite.com/technology-guides/using-internet/connecting-internet/how-connect-internet
[2]https://www.sitepoint.com/browsers-website-support/



Dashboard

Register plant pot

Serial number — Serial number is at the bottom of your plantpot

Henk — Give your pot a name

Password — Password is at the bottom of your plantpot

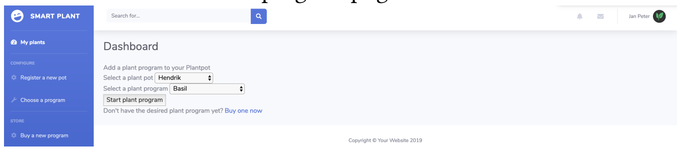
Register pot to my name

Don't have a plant pot? Buy one now

Copyright © Your Website 2019

Figure 4: Register a plant pot

program.png



Dashboard

Add a plant program to your Plantpot

Select a plant pot: Hendrik

Select a plant program: Basil

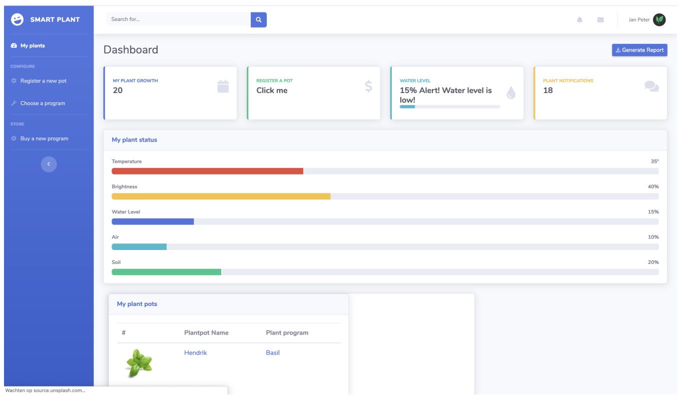
Start plant program

Don't have the desired plant program yet? Buy one now

Copyright © Your Website 2019

Figure 5: Choose plant program

case. This will redirect the user to the following page, as presented in figure 6. One could see the different kinds of data and values that are coming from the various sensors (i.e. brightness, water level, air, moisture). Also, when the water level reaches below 20%, the application will give a warning, notifying the user: "Alert! Water level is low!".



Dashboard

MY PLANT GROWTH 20

REGISTER A POT Click me

WATER LEVEL 15% Alert! Water level is low!

PLANT NOTIFICATIONS 18

My plant status

Temperature	35°
Brightness	40%
Water Level	15%
Air	10%
Soil	20%

My plant pots

#	Plantpot Name	Plant program
	Hendrik	Basil

Wachten op source.unsplash.com...

Figure 6: Analytical information about the SmartPlant Hendrik

4 SYSTEM DESCRIPTION

This section gives a description about the hardware and software used, the internal processes, services and architecture.

Figure 7 in Appendix A presents the overview of the system architecture. This architecture displays the overall flow of data, behavior and interaction amongst the various components of the SmartPlant. SmartPlant is made of four main components, namely; the back-end functionality, web application, database server and the NodeMCU. Five different sensors have been used, as can be derived from the architecture, which will be further described in the next section. These sensors send data, with an interval of three hours, to the NodeMCU[3], which is an open source Internet of Things firmware. It is a hardware component that connects through a WiFi module and is used to process and control the sensors and water pump and to retrieve and send the data. The LAMP server is used to run Apache HTTP, the webserver. This server processes the HTTP requests and instructs the NodeMCU which program to run(e.g. basil program or tomato program). Information such as sensor data, plant programs and user data are kept on the database server. The SmartPlant application is written in the front-end languages HTML, CSS and the server-side scripting language PHP. The web-application is used for the user to access the analytics about their plant and provides the user with an interface to keep track of the plant analysis. We leverage Trefle[4] API to gather plant specific data with regard to the minimum and maximum plant requirements (e.g. temperature, brightness, moisture). By combining these various components, it therefore bridges the physical and digital world by linking and connecting these various hardware components and software all together.

The overall flow of data of these components can be interpreted as: Every SmartPlant is connected to a variation of sensors and to the internet through a NodeMCU. The generated sensor data is transmitted and stored in a MySQL database which runs on a LAMP server. In this database the user data and plant programs are also stored. A web application allows users to register their Smart Pot, get insights of the plant's growing environment and load plant programs to the SmartPlant. The plant program is a growing program dedicated to a specific kind of plant which manages the water pump and informs the user about the optimal values for each sensor, so the the user can control the humidity, temperature and brightness to the optimal condition. Every three hours the SmartPlant autonomously sends the current sensor data to the database. In case a sensor malfunctions, the user is warned (see section 5.3. The following subsections will further describe the individual components.

4.1 Protocols

In order to maximize the functionality of the SmartPlant, using the appropriate protocols can extend application possibilities. This chapter focuses on describing which protocols were selected for use and why others were taken for consideration.

4.1.1 Communication protocols. **HTTP vs. MQTT** A widely applied protocol for IoT purposes is MQTT. This is considered to be a lightweight machine-to-machine messaging protocol. The consideration for this protocol was because of the fact that many other IoT devices are able to correspond with each other when implementing MQTT. However, during the development of the SmartPlant, it was decided that the hardware system would not communicate with other MQTT-devices. Therefore, a review on what would be the most efficient protocol for this application resulted into selecting HTTP as the preferred protocol. The SmartPlant uses in terms of networking very simple message interaction and HTTP is perfectly able to fulfill this task. Another reason for using HTTP was that it

[3]https://github.com/nodemcu/nodemcu-firmware

[4]https://trefle.io/

allowed the server-side system to be developed more easily, suitable for the limited amount of time that was available during the development process.

Ethernet vs. WiFi During the first prototyping phases, the SmartPlant hardware was built around an Arduino Uno as the central hub. However, while further designing the application of the SmartPlant, it quickly became clear that the end product would have to be wireless in order to be a viable product. This would also mean the internet connection of the SmartPlant would have to be wireless and therefore implement the use of WiFi. The Arduino was exchanged for a NodeMCU as this small development board has an implemented WiFi module. Another option would have been to extend the Arduino with a WiFi board. This option would have been more expensive and would also have consumed more space within the SmartPlant design.

Setting up a new SmartPlant After a user acquires a SmartPlant, a short setup protocol has to be run through. This is done by powering the SmartPlant on, which is done by pressing a button. On the concept design, this button might be placed on the bottom of the pot. After doing so, the SmartPlant will be in a "setup-phase". This means the SmartPlant broadcasts a local Wi-Fi hotspot named "SmartPlant-setup" as SSID. This local Wi-Fi hotspot has with limited access to the SmartPlant interface. When a connection is made between the SmartPlant and a Wi-Fi capable device, the user is able to configure the system by opening a new session in a web-browser to access a local web-page (for example 192.168.1.101). On this webpage, the user is able to configure the Wi-Fi connection between the SmartPlant and the Wi-Fi network that is available in the area the SmartPlant is placed in. After this is successful, the SmartPlant is fully operational.

4.1.2 *Security protocols.*

To ensure the security, privacy, and integrity of our customers' data, the communication between the customers' client (i.e. mobile application and web browser) and Smart Plant's server runs over a bidirectional encryption called Hypertext Transfer Protocol Secure (HTTPS). HTTPS is a combination of the Hypertext Transfer Protocol (HTTP) and the Transport Layer Security, which allows safe data communication over the World Wide Web [11]. This encryption authenticates the accessed website and prevents eavesdropping and tampering, meaning that the customer is securely communicating with Smart Plant's server without interference of third parties [12]. The encrypted protocol combination (i.e. HTTPS) is widely used over the internet by, e.g., banks, e-mail clients, payment transactions, e-commerce and corporate information systems [10].

To further implement security measures and ensure only authorized access to our customers' personal accounts, we integrated the bcrypt message-digest algorithm to hash and salt passwords[19]. It leverages a one-way (i.e. impossible to revert) mathematically algorithm to convert the provided password, during the registering and log in process, to a fixed size[13] and adds random additional input to improve its security[6].

4.2 Sensors

The SmartPlant is able to measure, monitor, and interact (on) with four different hardware sensors to generate an indication of how well the plant is doing and if it needs attention. In this section, each sensor is described technically while also describing its practical implementation. At the end of this section, communication between the sensors and the communication hub (NodeMCU) is described.

4.2.1 *Temperature and air humidity.*

The amount of water a plant might require depends on the dryness of the soil, temperature and the air humidity. The DHT11-sensor is a widely available sensor, which is very affordable, and measures both air humidity and temperature. An integrated 8-bit micro-controller processes the combined data and communicates through a single-pin output. The sensor needs 5V power which is a commonly used voltage on many development boards. The DHT11 can measure temperatures between 0-50 °C with an error range of ± 2 °C and humidity percentages between 20 and 90 with an error range of ± 5 percent. The pricing of an individual DHT-11 sensor varies between €4,- and €5,-.

4.2.2 *Soil humidity.*

Soil humidity is measured through a soil moisture sensor. The sensor used for the SmartPlant prototype was developed by *Velleman* and is described as the VMA303. Comparable to the DHT11-sensor, it functions at a power source of 5 volts and provides its output digitally through a single pin. The VMA303 consists out of a kit of 2 sensors (soil humidity sensor + water level sensor) which is often priced between €4,- and €5. The soil humidity sensor is shaped in the form of two prongs which are placed into plant soil .

4.2.3 *Water level.*

The SmartPlant has an automatic watering system which makes use of a water reservoir. This allows the user to refill this reservoir with larger intervals compared to the intervals of watering the plant directly. However, the water reservoir will eventually be emptied by the automatic watering system. To prevent this, the water reservoir (integrated into the pot) is provided with a water level sensor, installed into the bottom of the reservoir. This sensor functions at a voltage of 5 volts and provides digital output through a single data pin. The sensor used in the prototype is part of the earlier described VMA303 kit, costing around €4,- and €5,- making it very affordable. The sensor measures if there is contact with water and sends a "HIGH" signal when in contact and "LOW" signal when not. When it outputs "LOW", the water reservoir needs to be refilled via the spout.

4.2.4 *Light sensor.*

The sensor used in the SmartPlant prototype is commonly described as a photoresistor. The photoresistor is placed in a 5 volts circuit and depending on the amount of light which reaches to sensor surface, resulting to a variance in the analog output at the other side of the 2-pin sensor. These type of sensors are widely used in various electronic appliances and due to simple technology very inexpensive. They are commonly priced around €0,20 each. In our prototype, a 10K Ω resistor is used to protect the photoresistor from voltage spikes which can cause it to malfunction or break. In order to effectively use analog data, a scale is used to determine the light status. To give an example, the sensor's output is interpreted at a maximum of the value "1000" when being in direct sunlight. By defining a scale the system is able to distinguish levels of light the plant receives. As an example: all values in the range of 700-1000 are qualified as "in sunlight", ranges between 400-699 are qualified as "moderate lighting", ranges between 150-399 are qualified as "low lighting" and every value between 0-149 are considered to represent no light at all.

4.2.5 Sensor communication. The NodeMCU boards functions as a hub for receiving sensor data and processing this to useful interpretations. As described, all sensors communicate over a single sensor pin. As the NodeMCU has both digital and analog inputs, almost any sensor can be connected easily. All sensors consume power, provided by the power outlets on the NodeMCU board, which is powered itself by a battery pack. The SmartPlant is configured to send and receive data with an interval of three hours to efficiently consume energy and not create a sensor data overflow.

4.3 Servers

To host the web application and database we leverage the service of a third party hosting provider. The team currently does not posses the technical knowledge, resources and skill to configure and secure a server, therefore SmartPlant runs on a managed dedicated server. The hosting provider manages the server with its expertise on configuration, security, monitoring and backups. The possibility and room for future growth to handle more traffic and data storage is mandatory. Therefore this option was considered carefully to allow for scalability in case the user base and performance of SmartPlant exceeds the server capacity. In terms of costs, the preference leans towards leveraging the service of a third party over having an own server, as hardware does not have to be purchased and no dedicated personnel to manage the server is needed.

4.4 Data gathering

All data from the SmartPlants is loaded into a centralized MySQL database. The structure for this database is illustrated in figure 8. In previous research [17], a similar database structure has been proposed. However, we found that two simplifications could be made in the researchers' database structure. The simplifications made on the structure are the removal of different device types and the removal of categorisation on plant programs. We don't use different device types, because every SmartPlant should be modular. Meaning that a user must be able to choose which sensors they want to activate and deactivate. That also means that the database must accept each value, and each absent value. The second simplification, removal of Plant categories, was made because this categorisation is already made in the external API, Trefle.io, that we use.

The database structure consists of five tables. Central to this structure is the Plant table, which contains the programs that are loaded on each plant pot. This table only contains relations to other tables, and a Start- and EndTime for the program. The EndTime column is left empty until the program is over, or manually stopped by the user. The Plant table has three relations. The first relation is to the Default Plant table. The Default Plant table contains a template for the optimal growth aspects of type of plant. This template provides data for how often the water pump should provide water to the plant and tells the user if they should move the plant to a zone where there is a different temperature, brightness or humidity. The second relation is to the Sensor Data Template. In this table all the data that is created by the sensors is gathered. Since the sensors can break, or be removed by the user, some of the rows may have empty columns. The third relation is to the Plant Pot table. In order to deliver a scalable solution each SmartPlant pot comes with a unique identification number and password, which is used to identify and keep track of the customers. These unique set of values are pre-registered in the database and registered as "isUsed" is 0 (i.e. not active). Whenever a SmartPlant Pot is registered through the web application, the value "isUsed" is set to 1, as seen in figure 8. To connect to the SmartPlant Pot the IP address of the pot is needed, because we send data over the HTTP protocol. This IP address is stored under IP Address, which can accept either an IP4 or IP6 address. This connection is further described in subsection 5.1. The last table, Plant Pot table has a relation with the User Table via the UserID key. The user table contains all users. Users log in with a username and password. Their e-mail-address is registered for future marketing applications, such as news letters, and for the administrator to be able to communicate with them when there is an update or something is wrong.

Listing 1: A call to trefle.io which returns the values for one of their plants

```
cURL https://trefle.io/api/plants/109794?
    token=xxx
```

The Default Plant table gathers data from Trefle.io. This API came out in January 2019, two months before writing paper, and is still in an Alpha stage. The API is maintained by a single person. After using the API, it seems to be rather unstable. Relatively often than other well-known websites, we got "502 bad gateway" errors, when attempting access the Trefle.io website. Since botanical information about plants doesn't change that often and we regard the API as unstable, we decided to limit the amount of calls made to the API. Instead of calling this API every time a user wants to know information about their plant, we have an admin run a script on a weekly basis which performs a GET call, as seen in listing 1, to the API and gathers all the latest information regarding the plants we use in our application. In listing 1 it can be seen that the GET call requires an ID of the plant. This is stored in the table Default Plant under TrefleID. The data is then stored in Default Plant table, which the SmartPlant application queries when a user requests information about their plant through our application.

5 DISCUSSION

So far we've described the motivation, theory, interaction and components for building SmartPlant. In this section we will bring all this together and describe how the service should function as it grows and how potential errors can be solved.

As is described throughout sections 3 and 4, we've built and tested SmartPlant with one plant. However, we we're unable to add a second SmartPlant due to time and cost limitations. Thus, the description that follows in this section is based on theory and expertise of the researchers.

5.1 Connecting SmartPlant Pots

To create a network of SmartPlant Pots we use a Client-Server architecture [3]. within SmartPlant the server is the hosting party where the website and database are ran as shown in the architecture in figure 7. The clients in the service are the SmartPlant pots, which is controlled by a NodeMCU module. This NodeMCU is connected to the end-users WiFi, via the method that is described in 4.1.1 and has its own small server. On this small web server a HTML script, with dynamic data can be posted. Additionally, the NodeMCU can send data to a hard-coded URL or IP address outside the local network [1].

Within the architecture, as described in figure 7, it's shown that there is communication between the NodeMCU, its WiFi module and the LAMP server. The arrows connecting from the NodeMCU to the WiFi module is the NodeMCU posting its latest sensor data to its own little server with one HTML page. Next, the server on the WiFi module, now functioning as a client, is sending the LAMP server a message containing its IP address within the local home network of the user, the serial number of the Plant Pot and its password. This IP address is accepted through a PHP script and stored into the MySQL database (see figure 8) in the Plant Pot table in the row with the corresponding Serial Number. This step is repeated every time the NodeMCU restarted, or reconnected to WiFi.

Now that the SmartPlant central server knows the IP address of the NodeMCU it can perform HTTP GET and PUT commands to this IP address. The path where the NodeMCU stores its data is static, as shown in listing 2, and is not stored in the database. The GET command is performed every 3 hours and retrieves the latest sensor data from the HTML page that is generated by the NodeMCU. The GET command returns a JSON file, which can be interpreted by a script on the SmartPlant server. A GET command and it's returned value is shown in pseudocode in listing 2. The PUT command is performed every time a user selects a new plant program in the SmartPlant application, as shown in figure 5 under subsection 3.2. The NodeMCU provides the ability to add a basic authorization header with the Flask library [23]. This authorization header is sent with every GET and PUT call to authenticate that it is the SmartPlant central server who is sending the request. The value for the authentication header is set statically when programming the NodeMCU, and is set to the password that is on the bottom of every SmartPlant Pot.

Listing 2: Psuedo code representing a GET call to one of the SmartPlants

```
cURL https://<SmartPlant pot IP address>
/sensordata?token=xxx

Returns:
{
  "PlantPotID": 123,
  "plantID": 4,
  "sensorData": {
    "airHumidity": 12,
    "soilHumidity": 18,
    "temperature": 24,
    "brightness": 600,
    "waterLevel": 222
  }
}
```

Occasionally the NodeMCU will need to be updated with new Plant Programs and security patches. The NodeMCU provides a class for updating the NodeMCU via internet through the HTTP protocol, called ESPhttpUpdate [4]. This method requires a static IP address and a path for the patch file location. As the service grows, more HTTP calls are made by the service. The status results of these calls must be logged. The administrator can then check these logs, to find if any errors occur when clients make their calls. In order to do this, the administrator could use the generic HTTP error code categories [15]. If the administrator notices an increase in errors, then they can act upon this. More specific error cases are described in section 5.3.

5.2 Resilience against breaking of sub-services

As described in the previous subsection, both the current IP address of each NodeMCU as it connects to WiFi and the patching function for the NodeMCU require a static URL. This means that as soon as the SmartPlant Pot is programmed and shipped, this static URL can never be changed. If we ever lose the domain name of SmartPlant after the first SmartPlant Pots are shipped, they will become useless. Luckily, the NodeMCU programming interface has a method which can check if the host is available [4]. Thus, we can manually code an exception with a second, and even a third host, if the first host is unavailable. Because we want to be less dependent of one hosting party, we can host a full copy of the website at a second hosting party.

To prevent data-loss we make both a daily and monthly back-up of the central database. These get stored on the server of a second hosting provider, so that if the first hosting party has a problem, the service can continue without to much interruption. Additionally if Trefle.io breaks, we already have all the current data stored in our own database. This way we could continue with the current data and hire a botany expert to add or update plant programs when necessary.

5.3 Resilience against breaking of components

Of course the hardware can break as well. Since the Pot has many different components, there are a lot of different errors the SmartPlant can give because of malfunction of one of the parts. If the NodeMCU stops working for some reason, the Web App will give a notification that the SmartPlant can not read data anymore and either your SmartPlant needs to be repaired or replaced. In the PHP script that can accept sensor data, each sensor value is read individually and inserted into the database. This way if one of the sensors returns an error, the other sensor values are still accepted. All sensors of the NodeMCU, even temperature, return values as a positive integer. If the sensors returns no value, a -1 is automatically returned. In the SmartPlant application a -1 value is interpreted as a missing or broken sensor, which can then be displayed to the user through the interface. Furthermore, each sensor has a range of values that are logical for indoor use. For example, if the temperature sensor returns a value of 90 degrees Celsius, an error is given on the interface which depicts the unlikely high value. Since the NodeMCU is the core of the SmartPlant, all functionality of the SmartPlant will be lost, so either a quick repair by a technically-skilled user or a replacement SmartPlant Pot is necessary.

If one of the sensors breaks, there will also be a notification in the interface that certain data cannot be read anymore. Since the data is read from every sensor separately, not everything will be broken. Some functionality will be lost, but probably the SmartPlant can still work, unless it is the soil humidity sensor, because then the plant will not get automatically watered. If it is the DHT11 sensor or the light sensor, there should not be too big of a problem, only that the application can not advise the user anymore about the environment of the plant. If the water level sensor quits working, the user needs to refill the water reservoir without notification. However, since the user probably had the SmartPlant for a longer period of time already, they will know how often the Pot needs a water refill. If the water pump fails, the soil moisture sensor will pick this up eventually, since the soil will only get dryer, and this data will then correspond to a notification that the water pump is broken. The last item of hardware that can break, is the battery pack. When the Pot is low on energy, the user will be notified in the app, but can easily replace the batteries themselves through the bottom of the Pot, just like they are used to in other battery-powered products.

Since the hardware is all integrated in the Pot, it will not be easy for the user to replace them. However, since the Pot will not be too expensive and sending a technician over to the user is more expensive, the best solution is to just replace the Pot. When the user does have the technical skills to replace or repair the parts themselves, a guide is provided in the Web App to show them how it is done, since there is an access point from the outside of the pot to the area of the sensors and the NodeMCU to replace or repair the parts.

6 CONCLUSION & FUTURE WORK

This paper presented a complex system called "SmartPlant", in which users can interact with their plant through a web interface. The aim of this paper was to deliver a scalable solution that allows novice users to practice indoor gardening with no prior knowledge in terms of botany. SmartPlant fulfills this purpose by keeping track of the soil humidity, air humidity, temperature, brightness, and the water level of the water reservoir. This brings data, collected by the various sensors, to the user which assists them in the care taking of their plant. Additionally, the task of regularly watering a plant is automated. To test this system a prototype of the SmartPlant Pot and the SmartPlant application were developed and connected. We believe therefore that the aim of this paper is met. However, due to the limited time and scope of this project, we were unable to create a network of SmartPlant Pots or validate the usefulness of SmartPlant for the users. Once a network of SmartPlant Pots is built, user surveys and interviews could be conducted in order to further enhance the system. Future work also contains the automation of several processes. These points are described in the next section.

6.1 Future work

As stated in subsection 5.3, SmartPlant can return a number of errors, based on the range of values each sensor should return. Through user experience testing we could come up with a list of errors that is based on (a combination of) illogical values, and provide the users with tips on how to solve these issues. This user experience testing could also be useful for testing the service in general and see what users like and don't like and what could be improved. This could also be done with more in-depth interviews and surveys. When the product is in use, there will also need to be more plant programs available in the store, since we want the SmartPlant to be compatible for every indoor plant. To add further application use, a furtherly improved version of the SmartPlant could be made weather-resistant and therefore suited for outside use. It would be interesting to provide further research on how the SmartPlant can be made suitable for outside-purposes.

Furthermore, currently there are some flaws in terms of scalability and efficiency. For example, for each SmartPlant the administrator has to manually create the serial ID and password. In order to further enhance the system in terms of scalability and efficiency, this has to be automated in the future. Another issue is that the NodeMCU requires a static URL, as mentioned in section 5.2. This needs to be addressed in the future since we currently have no solution other then programming a second and third host address. Also, the data from the NodeMCU to the server is not secured, which should be addressed when SmartPlant is used on a large scale.

REFERENCES

[1] "Apais (alias)". 2017. PART 1 - Send Arduino Data to the Web (PHP/ MySQL/ D3.js). (2017). https://www.instructables.com/id/PART-1-Send-Arduino-data-to-the-Web-PHP-MySQL-D3js//

[2] Nikoletseas S. & Theofanopoulos G. C. Angelopoulos, C. M. 2011. A smart system for garden watering using wireless sensor networks. *Proceedings of the 9th ACM international symposium on Mobility management and wireless access* (2011), 167–170.

[3] A. Arcuri. 2017. RESTful API Automated Test Case Generation. *International Conference on Software Quality, Reliability and Security* (2017), 9–20.

[4] Arduino. 2017. Over the air updates. (2017). http://arduino-test.esp8266.com/Arduino/versions/2.0.0/doc/ota_updates/ota_updates.html#http-server

[5] Onose N. & Petropoulos M. Carey, M. J. 2012. Data Services. *Commun. ACM* 55, 6 (2012), 86–97.

[6] Crackstation. 2018. Salted Password Hashing - Doing it Right. (2018). https://crackstation.net/hashing-security.htm

[7] T. Dahbura. 2013. Arduino Tutorial: Temperature Sensor. (2013). https://www.raywenderlich.com/2675-arduino-tutorial-temperature-sensor

[8] S. Dhawan. 2007. Analogy of Promising Wireless Technologies on Different Frequencies: Bluetooth, WiFi, and WiMAX. *The 2nd International Conference on Wireless Broadband and Ultra Wideband Communications (AusWireless 2007)* (2007), 14–23.

[9] Edwards J. S. & Xu M. X. Duan, Y. 2005. Web-based expert systems: benefits and challenges. *Information Management* 42, 6 (2005), 799–811.

[10] Kasten J. Baiiley M. & Halderman A. Durumeric, Z. 2013. Analysis of the HTTPS Certificate Ecosystem. *Proceedings of the 2013 conference on Internet measurement conference* (2013), 291–304.

[11] Gettys J. Mogul J. Frystyk H. Masinter L. Leach P. & Berners-Lee T. Fielding, R. 1999. Hypertext Transfer Protocol – HTTP/1.1. (1999). https://www.rfc-editor.org/info/rfc2616

[12] Electronic Frontier Foundation. 2018. (2018).

[13] H. Halevi, S. & Krawczyk. 2006. Strengthening digital signatures via randomized hashing. *Annual International Cryptology Conference* (2006), 41–59.

[14] Gogul I. Raj M. D. Pragadesh S.K. & Sebastin J. S. Kumar, V. S. 2016. Smart Autonomous Gardening Rover with Plant Recognition Using Neural Networks. *Procedia Computer Science* 93 (2016), 975–981.

[15] Microsoft. 2018. API design. (2018). https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design

[16] K. Okayama, T. & Miyawaki. 2013. The "Smart Garden" System using Augmented Reality. *IFAC Proceedings Volumes* 46, 4 (2013), 307–310.

[17] D. Oprea, V. S. & Alexandra. 2018. Informatics solution for Smart Garden based on Sensors. *Database Systems Journal* 9, 1 (2018), 12–21.

[18] J. Pearce, J. & Murphy. 2010. Living on the hedge: creating an online smart garden watering community. *Computer-Human Interaction* 1, 1 (2010), 420–421.

[19] D. Provos, N. & Mazieres. 1999. A future adaptable password scheme. *USENIX Annual Technical Conference, FREENIX Track* (1999), 81–91.

[20] Guangling G. Lina C. & Han W. Qiang, T. 2018. Nodemcu-based Low-cost Smart Home Node Design. *IOP Conference Series: Materials Science and Engineering* 435, 1 (2018), 12–19.

[21] Lennartsson M. Williams S. Devereaux M. & Davies G. Schmutz, U. 2014. The benefits of gardening and food growing for health and wellbeing. *Garden Organic and Sustain* 1 (2014).

[22] S. Singh, P. & Saikia. 2016. Arduino-based smart irrigation using water flow sensor, soil moisture sensor, temperature sensor and ESP8266 WiFi module. *2016 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)* (2016), 1–4.

[23] TechtutorialSX. 2018. ESP32 Arduino: Basic Authentication. (2018). https://techtutorialsx.com/2018/01/03/esp32-arduino-basic-authentication/

7 APPENDIX

7.1 Appendix A

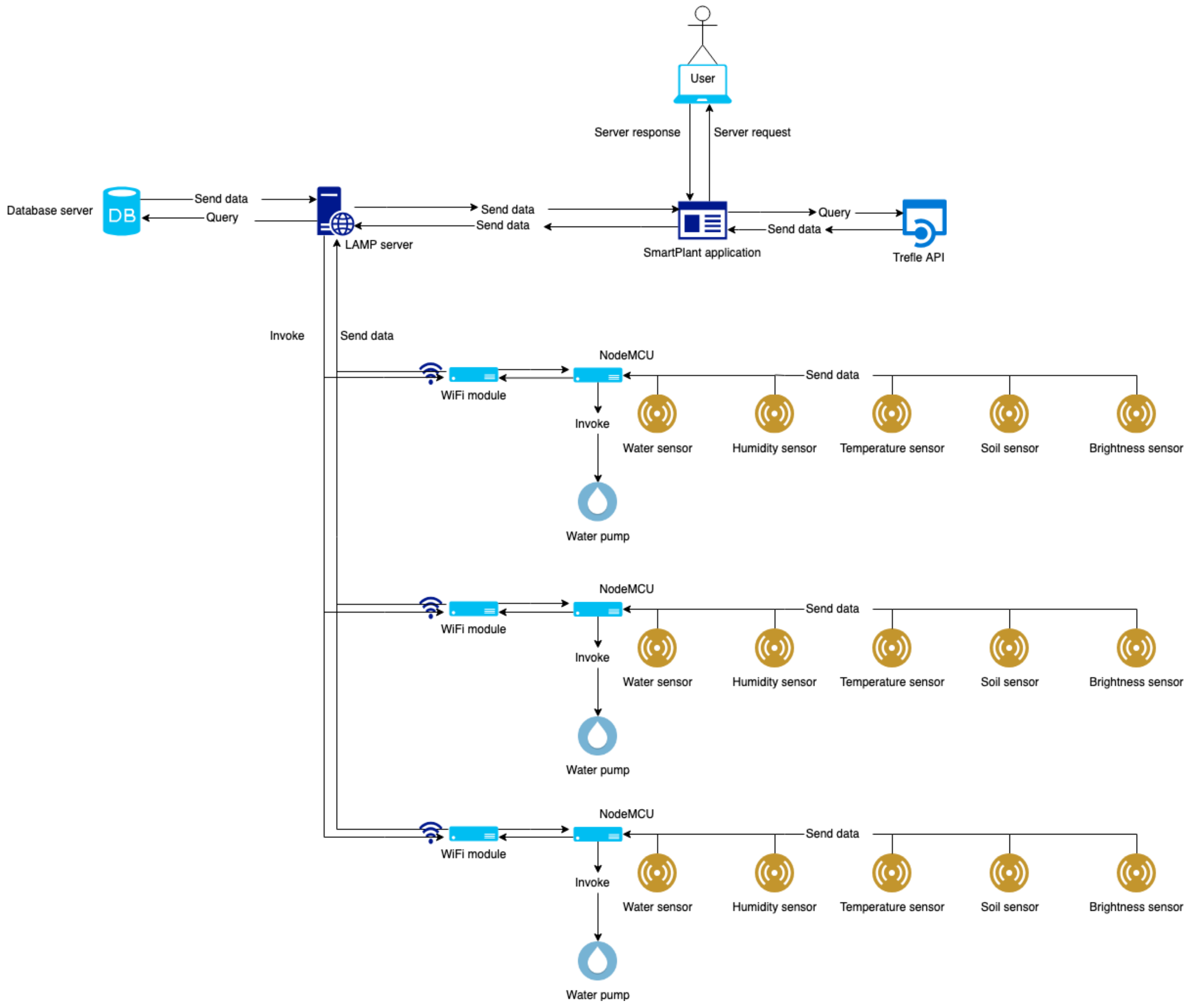


Figure 7: Overview of the system architecture

7.2 Appendix B

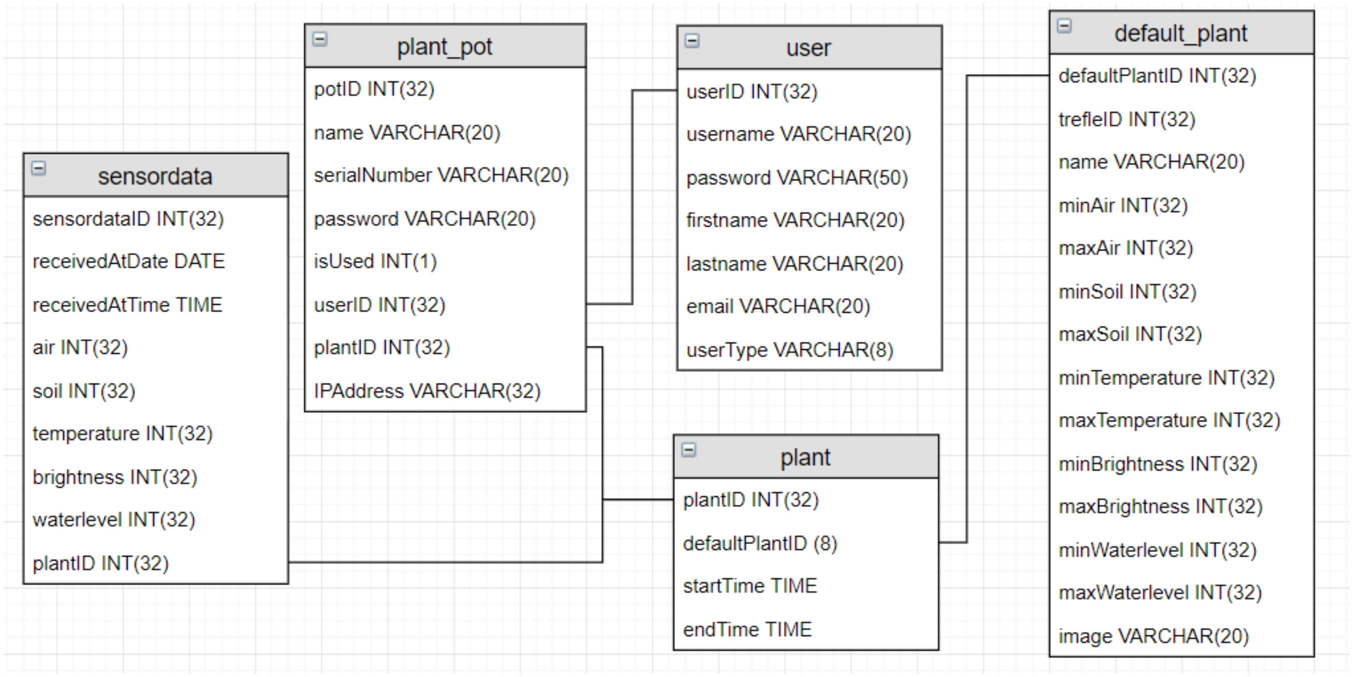


Figure 8: Overview of the relational database