Table of Contents

1. Abstract

The following work goes into detail on how Internet of Things technologies hosted on the Cloud can aid in the upkeep of plants and their surrounding environment while minimizing the amount of resources used in order to keep the plants healthy and proliferating.

The bibliography covers the state of the art and available technologies such as communication protocols and their characteristics, as well as commercially available Internet of Things devices and Cloud platform solutions as potentially viable candidates in helping to achieve the goal of this work.

With the knowledge gathered from the bibliographical research, an educated decision was made on how to develop the project. The development plan goes into detail how an ideal solution fitting the requirements would work for the given use case and outlines the functions of the system by specifying the system's components and its main deployable elements. Additionally, the planning phase sets the guidelines for the user interface design and interaction, finishing with a chronological estimation of the development efforts needed to complete the project.

The development section of this thesis covers the four main parts of realizing the previously created plans. It begins by elaborating necessary steps for putting in place the underlying infrastructure required to make the proof of concept work. It later goes into detail through the necessary configuration of the physical devices present in the system, followed by the development of the web application used to interact with the system and finally the deployment of the software components to the cloud.

Testing of the system is covered in this work and it includes a series of automated unit and integration tests, with the goal of examining the features of the system and focuses on evaluating whether the planned requirements have been fulfilled.

The works presented in this thesis are a part of a project developed at SAP Labs Hungary, with a deployed physical system at the premises of the company. The system demonstrates how Internet of Things solutions deployed on the cloud can provide a way of controlling plants' environments to keep them within favorable conditions and enable their sustained proliferation with as little manual intervention as possible.

2. Aim specification

One of the biggest goals of SAP on a global level is a meaningful contribution towards sustainability. During the last few years, SAP has been actively making a meaningful change when it comes to meeting the goals of sustainability and environmentally-friendly solutions through educating employees on best practices, providing facility support for recycling and reducing waste generation, replacing parts of its car fleet with electric vehicles, organizing campaigns such as “Bike to work Month” etc.

The aim of this project is to use Internet of Things technologies to provide a smart way of controlling the environment of plants in the office. This system is supposed to gather information about the surroundings of each plant, such as temperature, humidity, luminosity, air pressure etc. and respond in an appropriate manner to maintain the optimal conditions for the specific kind of plant. A successful deployment of this solution or one like it could enable a significantly better utilization of resources needed to sustain a proliferating environment for such plants.

Moreover, this project aims to display to both people visiting the offices of SAP Hungary and employees of the company how easily Internet of Things solutions can be integrated into a system that can enable better resource utilization.

Although at the current time this is only an internal project, the prospects of the proposed solution are quite optimistic, since the architecture is engineered to be scalable from the start and could be used in several industries, most notably in the agriculture sector where using similar technology is sometimes referred to “Precision Farming”. The economic benefits of a similar solution would be more apparent due to the sheer size of resources being utilized in a more efficient way.

3. Bibliographical research

3.1. Gateways and protocols

3.1.1. HTTP

Hypertext Transfer Protocol, or HTTP for short, is the protocol which is used as the basis of enabling data communication on the World Wide Web. Originally developed by Tim Berners-Lee and now maintained by the World Wide Web Consortium (W3C) [1], it allows for the transmission of webpages and files over the Internet.

HTTP works by having servers and user agents that connect to them, enabling the agents to consume the data. Server location is done by using a Uniform Resource Identifier (URI) or a Uniform Resource Locator (URL) which is a subset of the former set. [2]

The Hypertext Transfer Protocol works on the request-response principle, where a client, such as a web browser or a mobile application, sends a request to the HTTP server. The server's response contains a status code signifying whether the request was accepted, or there might have been an internal error in the server, or the requested resource has not been found, etc.

HTTP defines a set of methods request methods used query the server, create, update or delete a resource present on the server, or provide meaningful information about the resources and their accessibility [2]. These methods are:

- GET: Requests a representation of the specified resource. These requests should only retrieve data and have no other effect.
- HEAD: Similar to GET, however, the body of the response is not returned. Used for querying metadata while providing a smaller size of the payload returned.
- POST: Adds a new resource, specified in the body of the request, to the server.
- PUT: Used to add the resource specified under a given URI. If the URI is not present, a new resource is created. If the URI is present, the resource gets modified.
- PATCH: Used to apply partial modifications to a specified resource.
- DELETE: Removes the specified resource.
- TRACE: Used for providing debug mechanisms via performing message loop-back tests along the path to the target resource.
- OPTIONS: Used to query the list of HTTP methods supported by the server on the specified URI. This is used by Cross-Origin Resource Sharing (CORS) to enforce security restrictions for given resources. In this context, the OPTIONS request is called a pre-flight request.

- CONNECT: Creates a TCP/IP tunnel, used to facilitate SSL-encrypted communication through an unencrypted HTTP proxy.

To add end-to-end security and provide protection against man-in-the-middle attacks, a standard HTTP communication is established over Secure Sockets Layer (SSL), or nowadays more often used Transport Layer Security (TLS). This allows the entirety of the HTTP protocol to be encrypted and sent over the SSL or TLS protocol. These encryption schemes typically rely on long-term private and public keys which are used in the generation of a short-term session key, that later gets used to encrypt the communication between the sever and the client.

HTTPS plays a very important role in publicly available networks, such as Wi-Fi hotspots commonly found at cafes, libraries and universities, since anyone connected to that kind of network could relatively easily eavesdrop on the communication over plain unencrypted HTTP. The same kind of reasoning applies to scenarios where Wireless Local Area Networks (WLANs) are required to enable connectivity to many devices within an area that can be frequented by personnel who are untrusted.

In the scope of this work, the HTTP(S) protocol can be used to establish communication between components, most importantly the communication between a web-based application and a backend service exposing application programing interfaces (APIs) over this protocol as it is the de-facto standard of web-based services at the time of writing this thesis.

An alternative of HTTP is the Remote Procedure Call (RPC) protocol [3]. However, due to the age of the RFC protocol and the higher popularity of HTTP, the support for HTTP in modern-day programming languages is superior as opposed to RFC.

3.1.2. MQTT

Message Queueing Telemetry Transport, or MQTT, is a protocol based on publish-subscribe messaging. It was engineered to be a lightweight messaging transport protocol based on top of the TCP/IP protocol. Its intended use cases are in machine-to-machine (M2M) and Internet of Things scenarios. MQTT offers connections for remote locations where either a small code footprint is needed, or where the network bandwidth is very limited. [4]

An MQTT system typically is made up of clients that communicate with a server, oftentimes called a “broker”. The client can either publish or subscribe to information and every client can connect to the broker. Information is organized and communicated in groups, called topics. A client subscribes to a specific topic, and when the broker receives a message published by a client within that topic, it publishes the information from the message to all subscribers of the topic. The publisher does not need to have any knowledge of the subscribers and conversely, subscribers do not have to be aware about the publishers. The broker discards messages in a topic for which there are no subscribers.

The TCP protocol is the foundation of MQTT's data transmission. There exists a specific variant of MQTT, called MQTT-SN, which is used over unreliable transport protocols such as UDP or Bluetooth. Connection credentials are sent out in plain text format and no security or authentication measures are included by MQTT. These measures are often provided by the TCP protocol sitting below MQTT.

This protocol is defined by the ISO/IEC PRF 20922 standard. The Organization for the Advancement of Structured Information Standards (OASIS) is responsible for the definition of new improvements of the standard and it has recently published the official MQTT v5.0 standard [5], which represents a significant step forward in the refinement of the MQTT protocol. Special care was taken by OASIS to retain the backward compatibility of the standard, while updating the standard to provide the following features such as Message Properties, which adds metadata in the header of an MQTT message, Shared Subscriptions allowing for load balancing across multiple clients, Message and Session Expiry, Topic Aliases providing mappings between topic strings in messages and numbers, reducing the overall message size, etc.

Many IoT solutions leverage the power of MQTT to establish communications between devices mainly due to its low power usage, and small size of transferred data packets. Furthermore, as opposed to HTTP, an MQTT connection does not close after each request, leaving the channel of communication open and the associated overheads of opening and closing connections.

The MQTT protocol is relatively easy to set up, and there a wide range of available MQTT brokers available on the market [6], making it an interesting candidate in choosing a protocol which some of the IoT devices encompassed by this project can communicate over.

However, since the MQTT protocol operates over TCP, requiring several additional steps before a connection that is able to exchange information is established, it can affect "wake-up" and communication times. This can also affect the power usage of the device, as many IoT devices have relatively limited computing power, thus making the task of creating a new TCP session more resource intensive and ultimately increasing the overall electricity consumption of the device.

3.1.3. Zigbee

Zigbee is an IEEE 802.15.4-based specification that defines a set of high-level communication protocols used to create comparatively small, personal area networks with low-power digital radios. The intended use cases are home automation, medical device data collection, and other low-bandwidth, low-power scenarios. These Zigbee devices are designed for small scale projects which need wireless connection. [7]

Zigbee sits on top of the physical layer and media access control defined in the IEEE standard 802.15.4 defined for low-rate wireless personal area networks. Additionally, the network and application layer, Zigbee Device Objects and manufacturer-defined application objects are included in the specification. The responsibilities of Zigbee Device Objects are to keep track of device roles, to manage incoming requests for joining a network, as well as to enable device discovery and enforce security. The security in Zigbee devices is provided by using 128-bit symmetric encryption keys. The facilities of establishing and carrying out secure communications, including the generation and transport of cryptographic keys and ciphering frames are one of the defining features of the Zigbee specification.

Tree-topology and star-topology networks are supported natively by the Zigbee network layer, along with generic mesh networking. The presence of one coordinator device in a network is mandatory. In the case of a star-topology network, the coordinator must be the central node in the network. Both generic meshes and trees allow Zigbee routers to be used in order to extend the communication at the network level.

Zigbee's range is limited to distances ranging from 10 to 100 meters with line-of-sight, depending on environmental factors and the power output of the radio. However, due do the nature of the mesh-like network Zigbee devices form, data can be transferred over significantly greater distances with the help of intermediate devices. The defined data rate of the Zigbee protocol is 250 kbit/s, meaning that it is best suited for intermittent data transmissions from a sensor or input device.

There are three kinds of Zigbee devices [8]:

- Zigbee Coordinator: This device has the greatest capability. Its responsibilities is to form the root of the network tree and provide communication channels to other networks, if needed. In a Zigbee network, there is exactly one Zigbee coordinator, which stores the information regarding the network and also acts as the Trust Center, allowing the storage of security keys.
- Zigbee Router: Apart from executing an application function, a Router in the network can act as an intermediate router, which forwards data from other devices.
- Zigbee End Device: Has limited functionality, enabling communication to only the parent node (Zigbee Router or Coordinator), meaning that the End Device cannot forward any data from other devices. This enables the End Device to be asleep and not transmit any data for significant periods of time, extending battery life. Since structurally end devices are much simpler than Zigbee Routers and Coordinators, they tend to be least expensive from the types of Zigbee devices.

One of the benefits of using Zigbee devices is their availability. Many consumer grade devices can be had for a relatively small amount of money, making them very useful in

proof-of-concept scenarios, or when dealing with use cases where only consumer-grade hardware is enough. It is for these reasons why Zigbee devices are considered as a viable option for the implementation of this project.

However, Zigbee devices have a limited range and relatively low data transmission capabilities, making them unfeasible for larger scale deployments. Industrial-grade Zigbee devices are also available; they are frequently used in scenarios which benefit of having devices that are easy to relocate, such as in agriculture, in contrast to the typical deployments of programmable logic controllers (PLCs).

3.1.4. Bluetooth

Originally developed in 1994, Bluetooth is a name for a wireless technology standard intended to replace traditional cable-based connectivity. Originally standardized as IEEE 802.15.1 by the Institute of Electrical and Electronics Engineers, today it is managed by the Bluetooth Special Interest Group (SIG), which is in charge of overseeing the development of specification, managing the qualification program and protecting the trademarks. [9]

Bluetooth uses short-wavelength Ultra High Frequency (UHF) radio waves in the 2.400 to 2.485 GHz radio bands to build Personal Area Networks (PANs). It leverages a technology called frequency-hopping spread spectrum (FHSS) to transmit the radio signals over many frequency channels by using a pseudorandom sequence known only to the transmitter and the receiver.

The Bluetooth 4.0 specification [10], whose main feature is Bluetooth Low Energy (BLE), was adopted in 2010. BLE allows for smaller and lower powered devices, typically powered by a single coin-cell battery to communicate wirelessly, opening up many different opportunities. It operates in the same bands as traditional Bluetooth technology, but instead of splitting the channels into 79 1MHz-wide channels, it uses 40 2-MHz-wide channels.

When using BLE for IoT deployments, the substantially lower power consumption of BLE devices allows for BLE-enabled devices to run for extended periods of power using compact coin-cell batteries alone [11]. Moreover, the ability to form mesh networks can significantly increase the area one BLE network can cover.

However, since BLE uses a common unlicensed 2.4GHz band which is not regulated, it means that many other proprietary devices and protocols operate within this same band, potentially leading to significant interference and loss of data integrity.

Due to the fact that Bluetooth connections are encrypted, it is considered to be a reasonably secure wireless technology when used as intended. Additionally, the network changes frequencies while operating, making intrusions more difficult and less likely.

Having considered these facts regarding the Bluetooth protocol, the potential use cases such as connecting affordable sensors and similar devices were deemed relevant towards the progress of this work.

3.2. Devices

3.2.1. Raspberry Pi

The Raspberry Pi is, for all intents and purposes, a fully capable single-board computer developed by the Raspberry Pi Foundation with the goal of promoting the basics of computer science to schools and developing countries. However, the flexibility and practicality of the device was quite popular with tech enthusiasts as well and it exceeded the manufacturer's expectations as far as sales and use cases are concerned. [12]

As of December 2019, 4 major generations of Raspberry Pi have been released to the market. All models are consisted of a Broadcom system-on-a-chip (SoC) with an ARM-based Central Processing Unit (CPU) that boasts an integrated Graphical Processing Unit (GPU). The input/output (I/O) ports of the boards range from 1 to 4 USB ports, HDMI and composite video and a typical 3.5mm jack for audio output. Later iterations of the Raspberry Pi introduced General Purpose Input/Output (GPIO) pins to enable connectivity to and programmability of a wider range of devices. Ethernet ports that enable Internet connectivity were introduced to the B and B+ models of the device at a later stage. It is also possible to have Wi-Fi connectivity capabilities with the Raspberry Pi 3 and Pi Zero W models. There also are a wide range of accessories, such as cameras, displays and additional boards which can be connected to a Raspberry Pi.

Additionally, more recent versions of Raspberry Pi devices support Bluetooth and Bluetooth Low Energy communication [13]. Zigbee connectivity can be enabled by connecting either add-in boards to the Raspberry Pi, or by connecting a USB dongle to one of the available ports of the device. Due to the relatively powerful chips present on Raspberry Pi devices, especially when considering how lightweight the MQTT protocol is, a single Raspberry Pi device can act as an MQTT Broker without any significant impacts on the device's performance.

Raspbian, a Debian-based Linux distribution is provided by the Raspberry Pi Foundation as the default operating system for these devices. Due to their easy learning curve, the programming languages Python are Scratch are featured on this distribution as the main and default programming languages, with support of many other languages being provided by the operating system. There also exist a wide range of other operating systems that the Raspberry Pi can run, including Windows 10 IoT Core, FreeBSD, NetBSD, RISC OS, Android Things, SUSE, Fedora, Xubuntu and many others. [14]

The Raspberry Pi community is a very crucial part of the entire ecosystem. As it is a relatively big community of developers, teachers, users and enthusiasts who contribute to

open source projects and maintain them, it is considered to be very rich and proactive, enabling businesses and private individuals to continuously invent and develop new projects either for or with the help of the community.

Attributing to the relatively low price of the Raspberry Pi devices, they have been very popular with home automation enthusiasts. People keen on technology are modifying and extending the functionalities of Raspberry Pi devices in order to enable more flexible, more expandable and more budget-friendly solutions for home automation, compared to ones available from the commercial market.

A device called ModBerry [15], released by TECHBASE in 2014, and based on the Raspberry Pi offers extensions such as serial ports, CAN and Wire-1 buses and digital and analog I/O which are widely used in the automation industry. The design of this device allows it to be used in harsh industrial environments, overcoming Raspberry Pi's limitations when it comes to industrial IoT solutions.

Owing to the excellent community driving the ecosystem, support, open-source software and availability of Raspberry Pi devices, combined with their affordability and ease of functional extension, the Raspberry Pi is considered to be potentially one of the most useful devices when bringing the idea behind this project to life.

3.2.2. IKEA TRÅDFRI

IKEA TRÅDFRI devices are a collection of affordable consumer-grade devices intended for home and office use that have gained popularity due to their low cost and interoperability with existing systems offered by other vendors. The product line includes lightbulbs, power sockets, light panels, rolling window blinds, motion sensors and remote controllers. They rely on the Zigbee protocol to establish and provide communication to other devices.

The main and most popular devices from IKEA's smart home offering are the smart lighting bulbs and controllers. The device types in the offering can be classified as:

- Active device: Presents a device that can be actuated remotely, such as a dimmable wireless LED light bulb that fits a variety of standard sockets, a motion sensor or a power socket that can be switched on or off remotely.
- Gateway: Acts as a central coordinator of the Zigbee network consisted of active devices. It requires power and an Ethernet connection to a network. IKEA's official TRÅDFRI application provided for Android and iOS provides a detailed walkthrough of the set-up procedures needed for the gateway to work properly.
- Steering device: Used to establish initial connection to an active device or to bridge the connection between a gateway and an active device. Only necessary during the pairing process if there is a gateway present in the network, it can later be removed, and the gateway will directly communicate with the active device(s).

The main factor when considering the devices from IKEA's TRÅDFRI lineup was their availability and low price, at the cost of potentially sacrificing reliability. However, the ease of obtainability and high affordability mean replacements are quick and easy, outweighing any concerns about the devices' reliability and making them a fit for this project.

3.2.3. Philips LED CorePro

The Philips CorePro is a full-spectrum LED light with a cool light temperature of 6500K and an output of 2500 lumens. It is inexpensive and has a three-year manufacturer warranty. Moreover, it sits the A+ efficiency category, making it a very appropriate choice when there is a need to provide illumination to plants due to the broad range of light plants need to do photosynthesis and proliferate. The CorePro is very bright compared to other standard lightbulbs and lacks dimming support, however, in the case where plants need an extra source of light due to the lack of direct sunlight, it can prove to be very useful. This is why it was considered as a candidate for illuminating the plants.

3.2.4. Osram Fluora lights

The Fluora series are Osram's fluorescent lights destined to help in the culturing of plants by providing strong light in the blue and red extrema of the spectrum, which makes them a very good fit for the promotion of photo-biological processes in plants. With the help of these lights, plants in closed areas such as the home, a shopping mall, an office or the like, which do not get the necessary amount of light for proliferation, can get artificial daylight to insure their health and ultimately their growth.

The Fluora lights are sold as standard tubular lamps such as a regular household neon light with a diameter of 26mm and G13 fittings, making them a worthwhile option due to the ease of their installation and the ease of sourcing fittings for them. Additionally, they are offered in a variety of sizes (lengths), ranging from 438mm to 1500mm, meaning that they offer highly versatile and configurable solutions with off-the-shelf parts. This means that in order to mount the Fluora light, a specific fitting is needed. Luckily, the proposed plant box can be designed to accommodate these fittings, so mounting one of these lights should not pose an issue and would not hinder the project.

3.2.5. Atman AT-102 water pump

When looking into smart and IoT enabled water pumps or irrigation systems for plants, the cost can easily reach hundreds or even thousands of euros. Since a smart power socket can be had at a low cost, very similar functionality to what one of those more expensive commercially available options offer is feasible at a significantly lower price. Additionally, the AT-102 water pump, as well as similar pumps from Atman's offering, are rated for specific flow rates, meaning that a confident level of control over how much water is being delivered to the plants can be had. Unlike other more expensive solutions,

the pump lacks sensors and abilities to send data about the quantity of water it had displaced, however, this was not deemed as a significant downside and these pumps could potentially be used to water the plants.

3.2.6. Xiaomi Mi Flora sensors

The Mi Flora sensor is a relatively inexpensive sensor offered by the Chinese manufacturer Xiaomi. It has the ability to measure sunlight, temperature, soil moisture and soil fertility. Moreover, there exists a mobile application which contains more than 5.000 plant stereotypes and allows for monitoring of plants according to their type.

The sensor has two long electrodes that get implanted in the soil, a humidity sensor in one of the electrodes, as well as external (above the soil) sensors for ambient air temperature and light. It uses Bluetooth Low Energy to communicate, making it easy to integrate with other solutions. Powered by a widely-available CR2032 coin-style battery, it is rated for one year of service life when using said battery. [16]

Although the benefits of using the Mi Flora sensors come mostly from the low price and ease of integration, they can sometimes suffer performance hinderances or even fail when the manufacturer issues an over-the air firmware update [17]. Since the updates rely on using a mobile application to interface with the sensors, these circumstances can be avoided by simply not connecting the sensors to said application. Additionally, because of the low costs associated with purchasing these devices, several of them can be had and they can be implanted in the soil of the plant box to ensure accurate and reliable readings, regardless of any foreseeable circumstances.

3.2.7. Texas Instruments SensorTag

The Texas Instruments CC2650STK SimpleLink™ Bluetooth Smart SensorTag loT Kit was designed to help engineers with the development of Internet of Things and Cloud Connected product ideas and proofs of concept.

The small enclosure, powered by a coin-cell CR2032 battery, boasts a number of different sensors, including an accelerometer, a gyroscope, support for ambient temperature, humidity and lighting, and many others, easily integrable through the additional DevPack available for purchase. Its well-performing ARM Cortex M3 CPU allows for great extensibility of the default functionalities offered by the device.

Due to instability issues from previous experiences with the device and their slow process of resolution, namely a full system rewrite using the dedicated DevPack, this device was deemed unsuitable for the project.

3.3. Cloud and IoT solutions

Enabling remote connections to an IoT system is a challenging task. Due to security concerns it is recommended that IoT devices always connect to a different, dedicated network which has no access to critical components of other networks, such as a corporate network.

There are different ways of achieving this task, however, one of the simplest methods is to access the functionalities of the IoT system over the internet from a web service that handles authentication and authorization. This web service is securely connected to the IoT controller over an encrypted channel and is able to issue commands and read data from the IoT system.

"What we think of as the cloud today, [is] a service provider owning, operating and managing the computing infrastructure and resources, which are made available to users on an on-demand, pay-as-you-go basis." – said Susan Bowen, a vice president and general manager at managed service provider "Cogeco Peer 1" in an blog post for Computer Weekly [18].

As many cloud platforms offer services where a web application providing remote control of the IoT system could be deployed, choosing this type of web application hosting seemed like a reasonable solution for the task at hand. One of the greatest benefits of deploying the web service on the cloud is that there is no need to maintain the infrastructure on which the service is running. Moreover, many of the cloud platforms offer tools that aid in the management of IoT devices and the development of IoT software.

3.3.1. Amazon Web Services

It is rumored that the idea of the first widely available cloud service originated in the headquarters of Amazon. Since the company did not want to miss out on a lot of potential sales during peak days and hours on Holidays such as Black Friday and Christmas, they built their e-commerce system to be able to sustain a load and serve a given number of requests estimated to be the peak number during these high-times. This resulted in their systems only running at a fraction of the capacity they were designed to run at for most of the year, except for the short periods when the demand for their e-commerce services peaked.

That is when two of the engineers responsible for the infrastructure supporting Amazon's e-commerce business had an idea to automate a large scale of the underlying infrastructure and to "rent out" all of the available computing power that the company itself was not using to other interested parties [19]. Due to the magnitude of Amazon's existing infrastructure, this proved itself to be the first viable business option at a larger scale. Amazon founded Amazon Web Services in 2002 and launched its first cloud product offering, the Elastic Compute Cloud (EC2) in 2006.

The foundational building block of AWS' IoT offerings is the AWS IoT core component. This component allows for devices to communicate with other devices and applications running on the cloud. By default, it uses MQTT to establish communications, however, AWS claims support for the HTTP protocol and WebSockets on their website [20]. On top of that, AWS also offers other IoT services such as device auditing, device management, analytics and graphing, etc.

While providing a strong suite of IoT service offerings, AWS has some costs that are not obvious during the initial phase. It might seem that the price for sending or receiving a single message is low, however, the fact that IoT devices communicate very frequently has to be considered. An IoT system can easily send hundreds, or even thousands of messages per second, significantly increasing the costs to run the proposed system on AWS. Moreover, since the project is only meant to represent a proof of concept and is expected to be developed over the course of several months, subscribing to AWS may incur additional costs that would not make economic during the development phase.

3.3.2. Google Cloud Platform

The Google Cloud Platform (GCP) [21] is Google's offering of a set of cloud computing services running on the same infrastructure as Google's own end-user services such as Google Search, YouTube, Gmail and others. It provides Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and serverless computing environments. A serverless computing environment is an environment that requires no provisioning of resources, much like the AWS Lambda functionality provided by Amazon.

Like AWS, the Google Cloud Platform is built around a core component enabling device management, communication, security and data gathering functionalities, called the Cloud IoT Core [22]. One aspect where GCP differs from AWS is that offers integration of IoT systems with their own Cloud Machine Learning Engine.

For reasons similar to those why AWS was not considered to be a suitable pick, the Google Cloud Platform was also eliminated from the list of potential candidates where the cloud components of the proposed system would be hosted.

3.3.3. Microsoft Azure

Microsoft's Cloud Platform branded as "Microsoft Azure" is their offering in the world of cloud computing. Much like their competitors, they offer similar suites of services across the board, ranging from the abilities to host individual VMs following the IaaS model, through the possibility of running managed database services for different vendors, as well as offering a big number of Software as a Service solutions to its customers. [23]

Similar to both AWS and GCP, the Azure IoT suite is built around a core component, called the Azure IoT Central [24]. It has very similar functionalities to the ones provided by AWS and GCP including device management, security and messaging. One interesting

feature in the Azure IoT offering is the Azure Digital Twins service [25]. It allows for keeping an online, digital copy of a real-world physical object such as a room, building or even motorcycles and cars.

Once more, due to budget constraints and development cost concerns, the Microsoft Azure IoT platform was ruled out of the choice on where to host the system. All the above cloud service providers have some kind of an IoT service solution in their line of offerings, however they all share some key characteristics. First, these solutions come with a significant cost related to their operation. Second, they require very specific and cumbersome configuration of the connections between devices and the cloud. Third, many of the functionalities that are especially useful in deployments such as this, require using additional services like Amazon Lambda or Azure Automation, further increasing their cost.

3.3.4. SAP Cloud Platform

SAP Cloud Platform is a business platform founded on a multi-cloud base, allowing the use of latest cloud-native technologies and providing the benefits of a hyper scalable infrastructure. It allows the creation of intelligent, seamlessly integrated, scalable applications that can be delivered and consumed on a wide range of devices. The flexibility of the platform allows easy extension of cloud or on-premise applications by the fast addition of new functionalities to suit ever-changing demands. [26]

The SAP Cloud Platform can be hosted and accessed both as a public cloud, where the datacenter(s) hosting the cloud architecture are provisioned by SAP, or as a private cloud, where a given customer deploys the SAP Cloud Platform architecture on their own infrastructure that they are responsible of maintaining.

SAP Cloud Platform Internet of Things is a service within SAP Cloud Platform designed to allow easy onboarding, configuring and managing of remote devices by providing easily configurable IoT connectivity and large-scale device management. It allows complete lifecycle management of large numbers of Internet of Things devices starting from device onboarding and ending with device decommissioning.

By using a locally-installed IoT gateway connected to the service, sensor data collection and processing is enabled. Additionally, the gateway can be used to issue commands to connected devices, provided they have those capabilities. It is also possible to maintain the devices by sending software updates over the network. A software development kit (SDK) is provided by the service in order to enable easier creation of communication protocol adapters and interceptors. Moreover, the service provides means of establishing secure connections over the wide range of protocols supported by the platform.

An additional service and a layer of abstraction is provided by SAP Cloud Platform, under the name of Internet of Things Application Enablement (IoTAE), which provides a whole

suite of microservices intended to make development of powerful apps that leverage data storage easier and faster. [27]

These microservices allow for building applications specifically tailored to Internet of Things use cases by utilizing the Extensible Thing Model to create a digital representation of a physical object that can be controlled either automatically or manually by using the Event Management mechanism.

IoTAE additionally provides a collection of REST and OData-based services to effectively and efficiently store and leverage data to provide real-time insights about Internet of Things networks. Moreover, SAP's WebIDE, which is an Integrated Development Environment that runs in the browser, can be used to create applications with IoT-ready user interfaces.

While providing many useful microservices otherwise not available under the SAP Cloud Platform IoT service, much like the other IoT Service Solutions has an onboarding process that is tedious and prone to human errors, as it requires significant user input and a methodical approach, or relatively big development efforts to help with device onboarding automation.

Seeing how access to an SAP Cloud Platform account could be gotten at no additional costs, the exhaustive internal documentation and tooling available for this landscape, and additionally the support for hosting a web application made with the help of SAP technology, the SAP Cloud Platform can be one of the most reasonable choices for providing a place where the web application could be hosted.

3.3.5. openHAB

Open Home Automation Bus (openHAB) is an open source project whose goal is to provide community-driven and easily configurable platform that enables integration of many different IoT devices, regardless of their vendors or the protocols that they use.

Additionally, many of the devices that generally come with their own mobile application provide only limited functionality and integration possibilities. However, openHAB is user-centric, putting its main focus on the functionality that the user wants and provides ways to easily achieve what is required. [28]

openHAB abstracts away the physical devices or virtual sources of data by using the so-called items. These items allow the user to focus on what they represent, most usually in human-readable and understandable format, and not have to deal with specific IP address, MAC addresses or the similar. This allows for devices to be easily replaced within an IoT network.

When speaking about replaceability, it is very important to note that openHAB uses a modular architecture design, allowing easy and non-interrupted addition of new features,

such as bindings, which are used to provide connectivity to another system/vendor/standard, extensions, automation rules, etc. It is due to these benefits that openHAB is so popular among home automation enthusiasts who are more than keen to contribute their work to the open source project.

openHAB is able to run on a variety of platforms including Windows, macOS, Linux. openHAB is also distributed as pre-configured easy-to-install images for platforms like the Raspberry Pi (openHABian), Docker, PINE64 and others. This allows relatively fast and hassle-free installation of the openHAB platform, which is especially useful when only initial testing of the system is required.

However, the openHAB platform is not cloud-enabled on its own. There are several different approaches that can be used, including hosting the openHAB runtime on the cloud, but this would require a complex networking setup to ensure that the gateways and their connected devices can communicate with the runtime, for which they have to be on the same logical network. Luckily, the openHAB project has a cloud companion component that enables connecting the openHAB runtime to the cloud and exposing much of the related functionality of the platform over the Internet.

Due to the ease of set-up, as well as the coupled benefit that the development efforts can be focused on providing better user experiences in a shorter time, openHAB can prove to be a good choice when it comes to hosting the IoT platform for the project.

3.3.6. myopenHAB

myopenHAB is a cloud-based service instance used to connect a running openHAB system to the Internet via the Cloud Connector extension and provide features such as remote management, push notifications and integration with mobile and third-party applications, such as Amazon Alexa, IFTTT, Google Assistant, etc.

While the openHAB Foundation provides free instances of the myopenHAB cloud service instance [29], much like everything else within the openHAB project, the cloud service instance is open source, meaning that there are builds and images that can be deployed to any of the cloud service providers without many issues which usually result in better performing instances than the ones hosted by the openHAB Foundation.

The main downside of using the myopenHAB cloud service is that all of the management regarding the component such as updates, stability and any load balancing has to be handled manually, as opposed to the solutions provided by AWS, GCP, Microsoft Azure and SAP Cloud platform.

Luckily, this downside would not play a big role in the realization of this project and the functionalities of the myopenHAB service would fit the requirements of this project.

4. System plan

The below system plan illustrates the main components comprising the plant box. The plants are housed inside the box, implanted in soil. The soil also contains plant sensors that are implanted and can measure key parameters about the plant's environments. Additionally, a watering system is in charge of keeping the plants hydrated, and a lighting system provides light to the plants in cases where the plants have not been able to get an adequate amount of lighting.

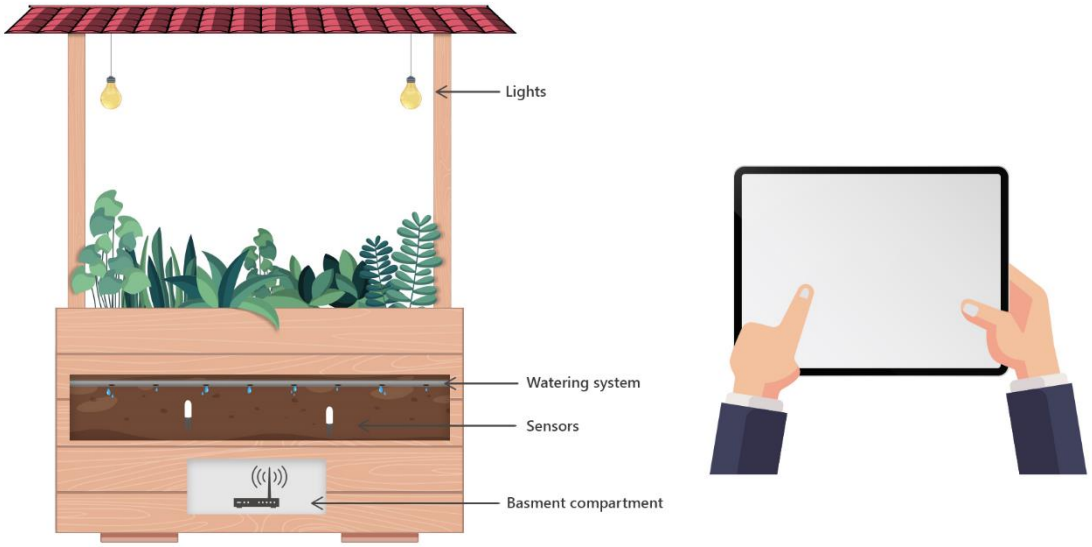

Figure 4.1 System Plan Illustration

The devices (lights, watering system, sensors) are IoT-enabled, meaning that they can be controlled from an application running on a tablet, which can monitor their status and issue commands to the devices. The gateways and any other devices responsible for the connectivity and control of these devices reside in the basement compartment of the plant box.

5. Decision-making process

One requirement for the project is to use SAP's own web front-end framework, called SAPUI5, having a user interface with a consistent look, branding and presence.

SAPUI5 is a framework based on JavaScript, HTML5 and CSS that boasts an industry-proven design and recognizable brand presence which many associate with SAP's reliability and trustworthiness. Moreover, SAPUI5 is built to support the Model-View-Controller architectural pattern and it provides support for both REST and OData JSON data models.

Because of its low cost and high practicality, while being energy efficient and suitable for lighting the plants, the Philips LED CorePro lightbulb was chosen as the main source of lighting for the deployment of the first stage of this project.

The Osram Fluora light was chosen as the lighting source for the colder seasons where some of the plants might not get appropriate levels of UV light.

Due to its reliability ratings and adjustable flow rates, the Atman AT-102 water pump presents a reasonable and low-cost solution for plant irrigation without any significant tradeoffs compared to commercial off-the-shelf irrigation systems.

IKEA's TRÅDFRI devices, namely one gateway, a dimmable LED lightbulb and two smart power sockets were chosen as the sockets can provide remote control for the water pump, Osram Fluora neon light and the Philips LED CorePro lightbulb.

The sensors chosen to monitor the plants were the Xiaomi Mi Flora sensors, due to their ability to connect over Bluetooth Low Energy. Moreover, an MQTT service can be set up for the Flora sensors that will publish their measurements on specific channels and will therefore notify all interested parties.

Because of previous decisions to use commodity hardware and the significant budget constraints for this project, openHAB was chosen as an IoT platform on which the system would be hosted, since it can be run on affordable hardware such as the Raspberry Pi and provides a REST API. Owing to the excellent community behind openHAB, there is support for many consumer off-the-shelf devices that are planned to be used in the project, making the configuration as smooth and as easy as possible.

Additionally, using the myopenHAB cloud builds that can be hosted on SAP Cloud Platform, features such as remote management and third-party application, -and system integrations are available, making openHAB the most sensible option for this project.

Many of the tedious configuration processes for onboarding IoT devices present on the IoT Service Solutions can prove to be significant slowdowns in the development process. Moreover, these solutions often come with either a subscription or a usage-based fee, for which the budget could not be allocated at stages when only delivering a proof of concept.

6. Planning

6.1. Deployment diagram

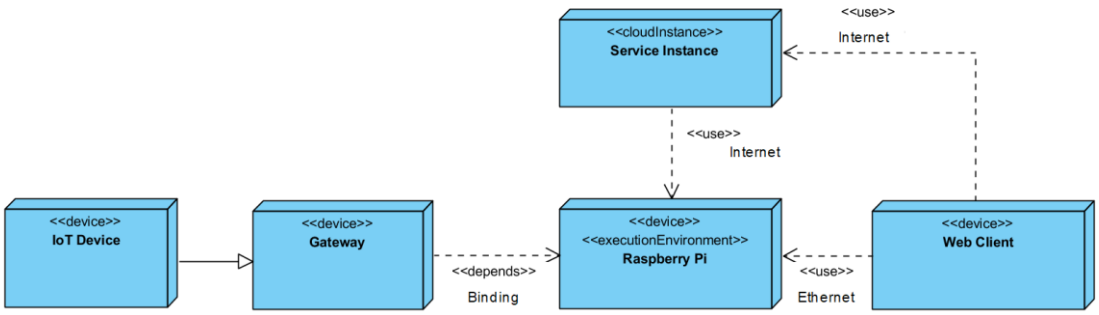

Figure 6.1 Deployment Diagram

6.2. Application requirements

Upon getting assigned to the project and receiving the prototype, the requirements had been discussed with my supervisor at the company, where the following key functionalities were set:

- Establishing two-way communication: Making sure that the backend system is working as intended and that the front-end SAPUI5 application is able to send commands and receive status updates about the connected IoT-enabled devices.
- Enabling easy device onboarding: When a device is to be added into the system, the configuration should be minimal, and the processes of onboarding should require as little input as possible from the side of the user.
- Providing automation mechanisms: It is possible to provide automation to one or more onboarded devices, following different kinds of actuations, such as a specified time of day, a change in a state of another IoT device onboarded in the system, etc.
- Setting up of dashboards: The user should be able to configure and use a dashboard up to their liking and requirements. The layout of the dashboard should be able change as to accommodate as many different configurations as the user requires.
- Providing a consistent user interface: As like with many other web applications built on the SAPUI5, this one should comply with the design guidelines and look-and-feel of the framework, providing an intuitive and easy-to-use user interface.

6.3. User Interface guidelines

SAP, with an ultimate goal of providing better user experience for its customers, has created SAP Fiori, which is a methodology of design and user interaction modelling which aims to make the interaction a user has with an application as streamlined and as simple to use as possible. The Fiori Launchpad (FLP) is the main point of entry for hundreds of SAP standard Fiori Apps written in SAPUI5 by SAP itself and is configurable to match the needs of enterprises/companies and additionally individual users as well. The Fiori methodology is also implemented in the libraries (or software development kits – SDKs) provided by SAP for developing native applications on either Android or iOS.

In order to adhere to the SAP Fiori guidelines and provide a better development experience to its teams and customers, SAP created SAPUI5, an HTML5 and JavaScript based front-end framework with many out-of-the box controls, styled with CSS to fit the SAP design language. It provides a wide range of predefined UI components which are well documented and tested. UI5 is also very flexible when it comes to running on devices with a wide range of screen sizes, ranging from small handheld device displays, up to big and high-resolution televisions. Most of the issues when developing applications needing to scale across devices are already solved by SAPUI5's responsive layout.

For this project's user interface, the goal is to provide a dashboard look and feel which is implemented not only in previous SAP web applications but can also be found in the Paper UI provided by openHAB. That means that the left side of the application contains the collapsible menus and submenus for navigation, and the right, main part of the application is from where the system and the devices are configured and controlled.

6.4. Deployment scenario

The idea behind the proposed system was to have a plant box located somewhere inside the SAP office space in Budapest, where colleagues and customers could see how a suitable environment for plants is being controlled by the Internet of Things. Additionally, the plant box should allow people to interact with the system by sending commands to the devices and observing their actions and monitoring the data collected from said devices.

To allow this, a tablet running the web application would be attached to the box. The web application can easily be hosted on SAP Cloud Platform as an HTML5 application and it can be made accessible to the tablet via the Internet.

Due to the small area where the system operates, it was decided that the IoT devices, router and the controller should be physically housed in the plant box, to make the deployment and any potential service efforts easier in the future.

6.5. Functions of the system

In this context, the word “things” refers to Internet of Things devices able to send and receive data and commands to and from a remote source. “Bindings” are abstractions that are used to provide connections and functionality to groups of devices specific to a vendor or a protocol the use for communication, for example the IKEA TRÅDFRI devices that use the Zigbee protocol.

List connected things
Add/edit/remove connected things
List bindings
Add/remove bindings
List extensions
Add/edit/remove extensions
Retrieve thing data
Issue command to thing
List automation rules
Add/edit/remove automation rules
Group connected things
Apply automation rules to thing groups
Persist thing data to database

6.6. Component diagram

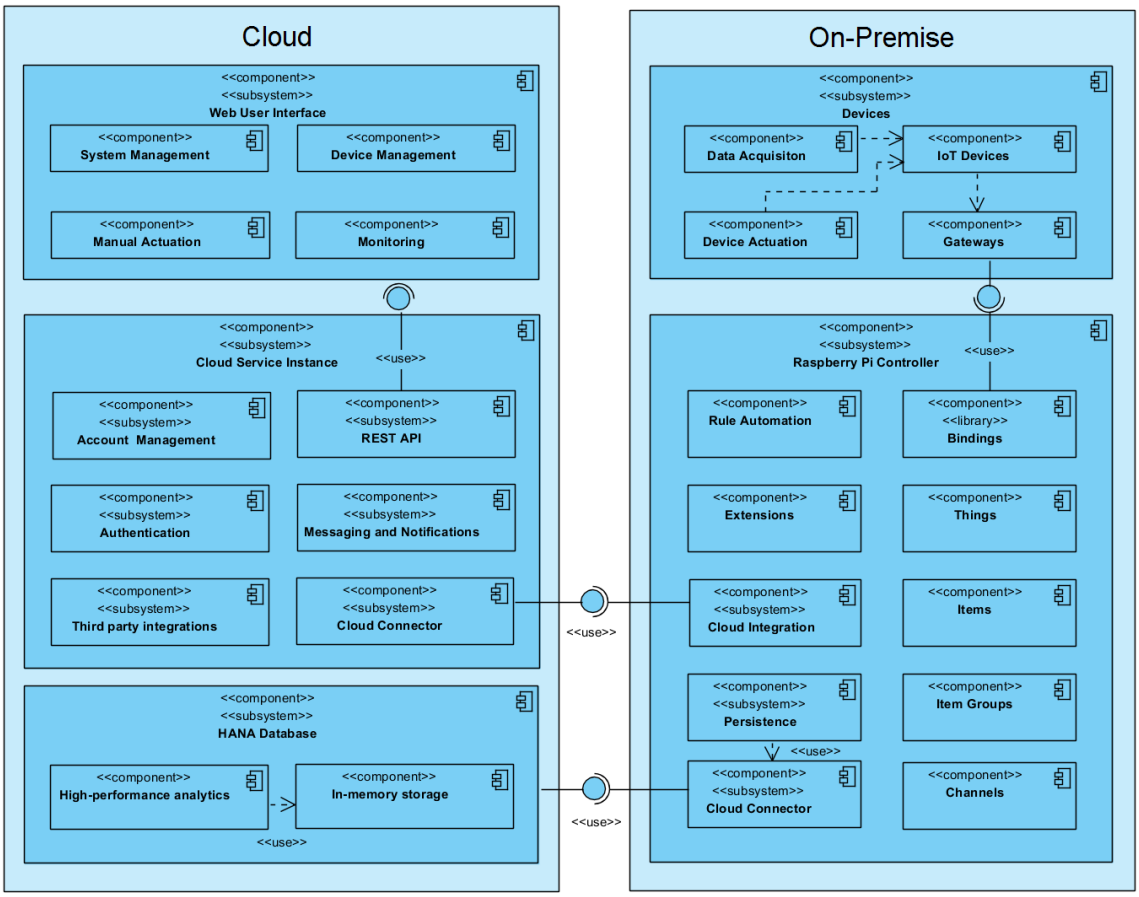

Figure 6.2 Component Diagram

When developing the system, it is quite useful to have an overview of each component and the means of interaction between them. Depicted on the component diagram are all components from the deployment diagram, including the internal subcomponents and the interfaces provided to enable communication between them.

It was decided that an SAP HANA database will be used to store the data from the system due the ease of set-up when using SAP Cloud Platform to host the components of the system and additionally because it provides a high-performance analytics engine that can use the data gathered from the IoT devices for future use, such as use cases like machine learning and artificial intelligence.

6.7. Activity diagram

The activity diagrams show the procedures of creating, reading updating and deleting bindings and things, the dependencies between them, as well as enabling their rule-based automation. In the case that there is no available rule to suit our needs, one can be easily created by providing the triggers and the subsequent actions. Optionally, when creating a rule used for automation, special conditions can be defined, allowing for much more flexible implementations.

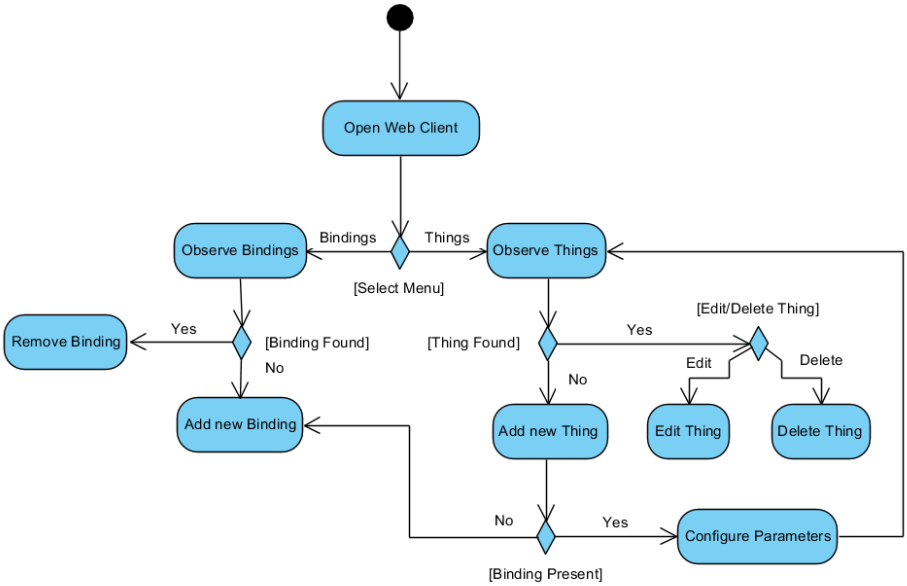

Figure 6.3 Activity Diagram 1

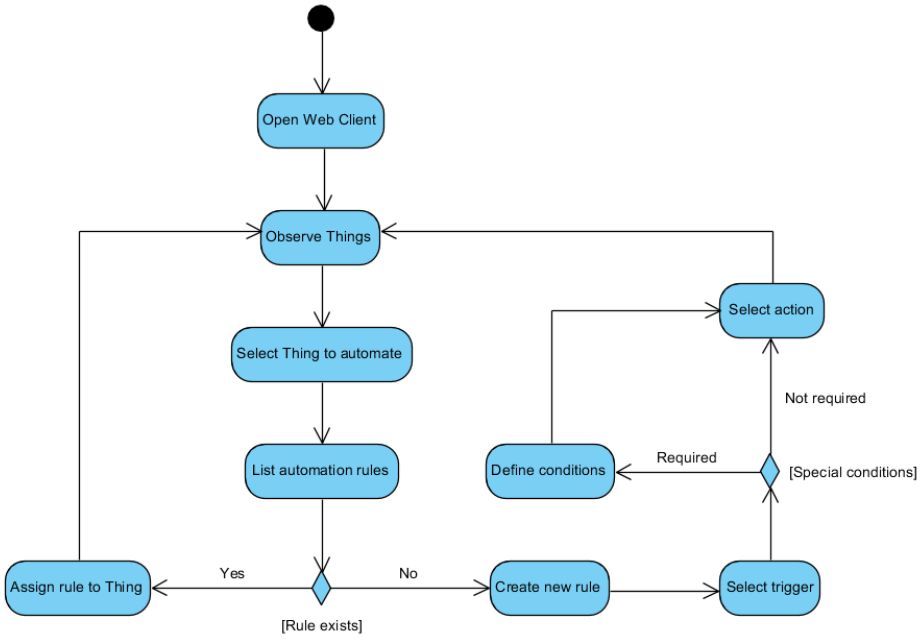

Figure 6.4 Activity Diagram 2

6.8. Wireframes

Figure 6.5 Home Page Wireframe

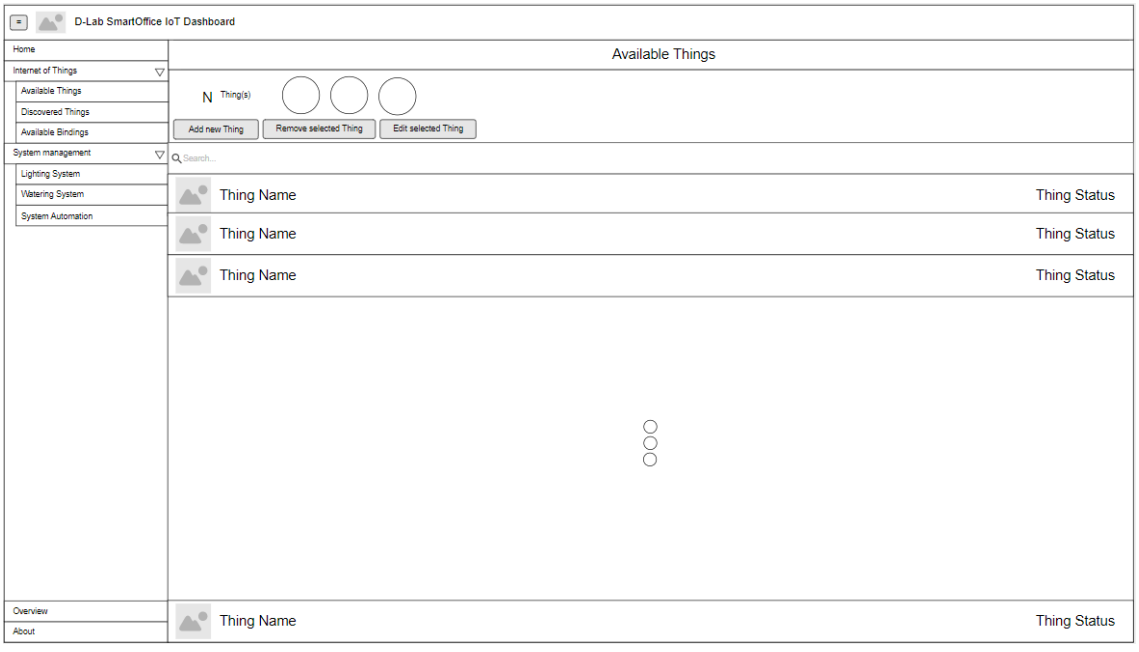

Figure 6.6 Things Page Wireframe

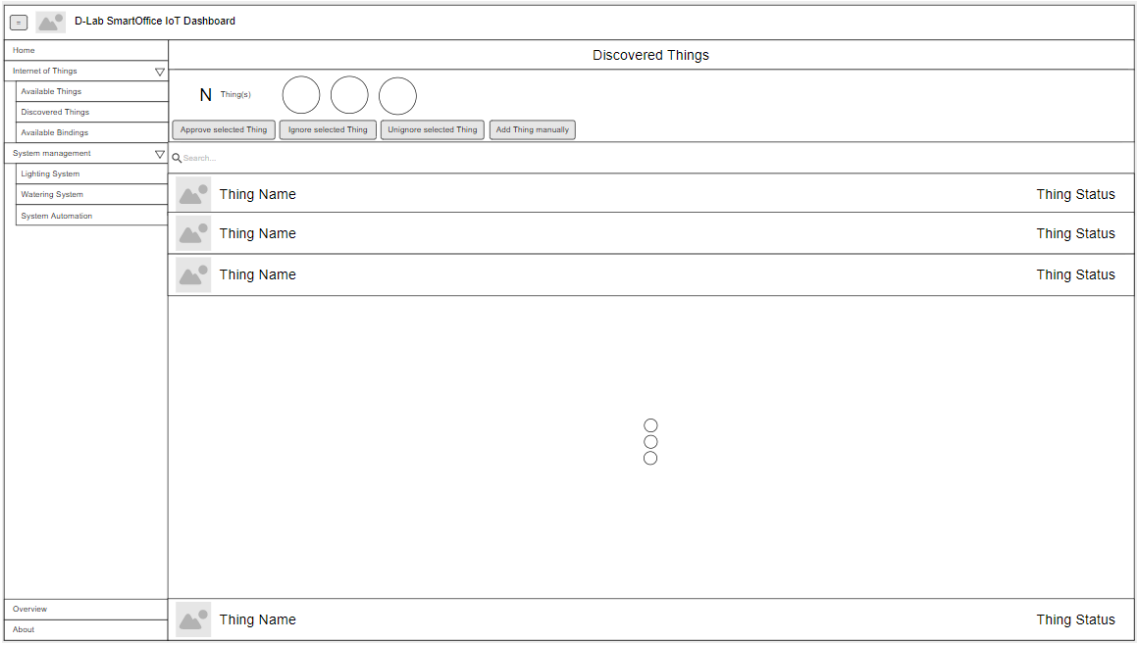

Figure 6.7 Discovered Things Page Wireframe

Figure 6.8 Bindings Page Wireframe

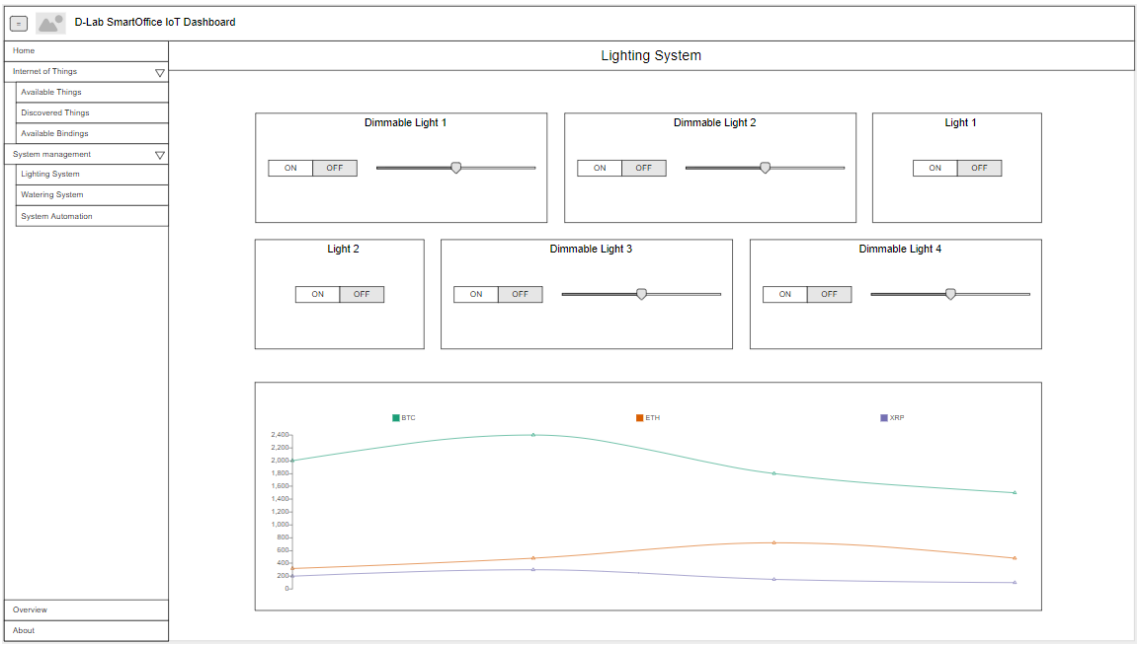

Figure 6.9 Lighting System Dashboard

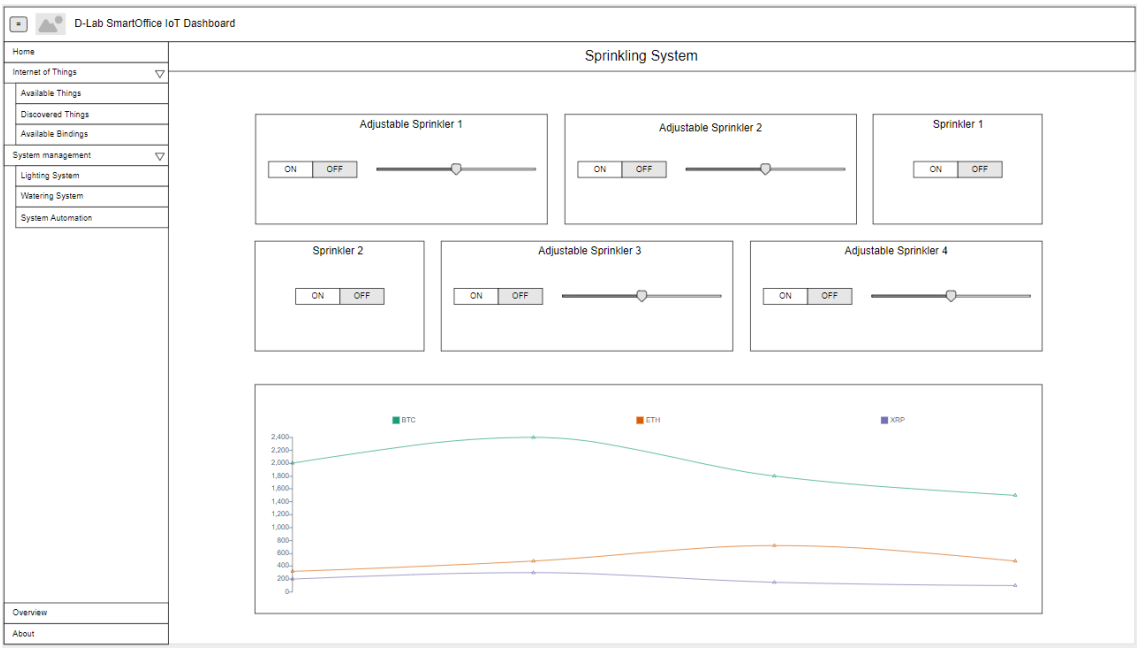

Figure 6.10 Watering System Dashboard

6.9. Gantt chart

The following Gantt chart shows the planned development cycles for this project. The thicker lines separate the months, while the thinner lines separate the weeks of the development stage.

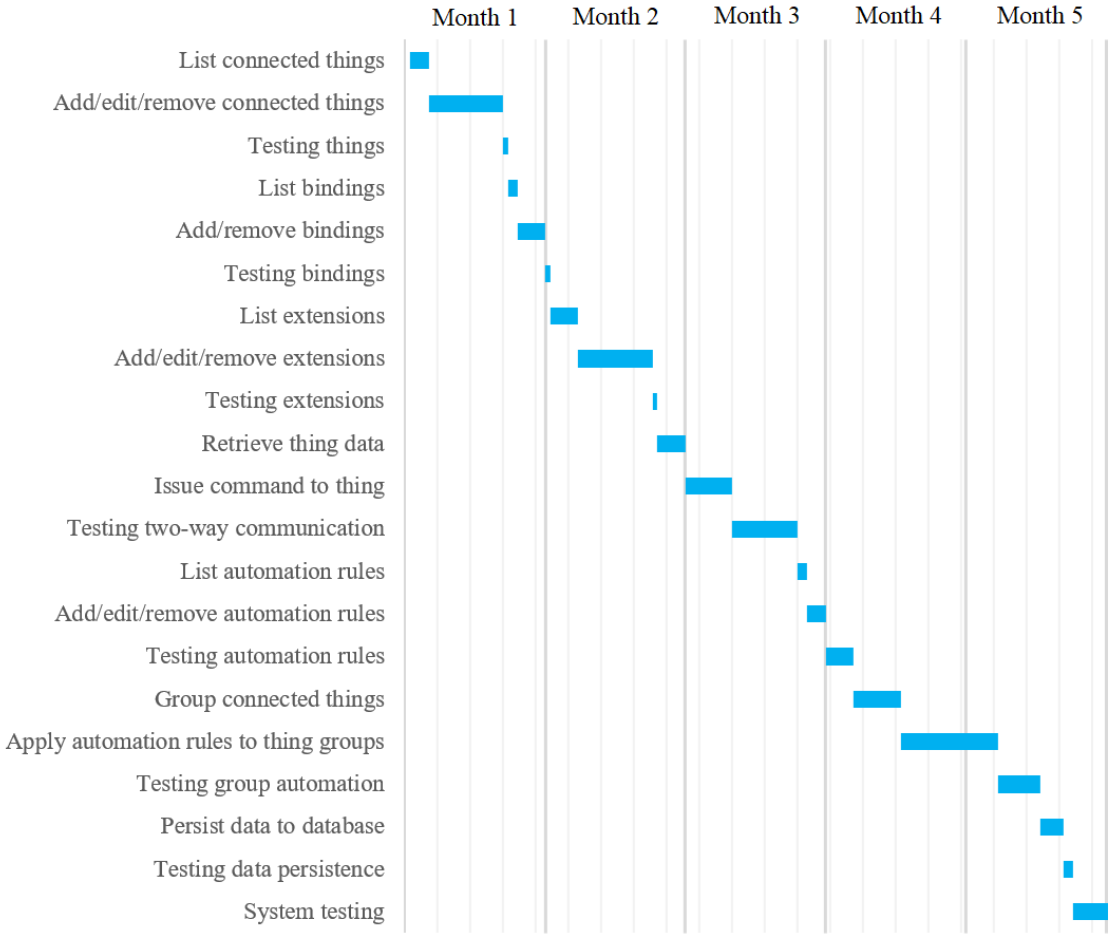

Figure 6.11 Gantt Chart

7. Development

The first stage of the development process was the handover of the previously developed user interface. The source code of the SAPUI5 application was given to me as a .ZIP format archive, upon which I took its contents, set up a git repository, and uploaded the source files to the internal GitHub server hosted by SAP.

The original SAPUI5 application contained a navigation menu and a series of screens, of which one was displaying a map that had been set up with mock data of a sensor and a mock location containing coordinates of a hypothetical sensor. This application was not connected to any service or API, as it was merely used to outline the look and feel of the user interface.

My plan was to add two systems to the application. The first would insure that IoT devices can be managed, and data and commands can be retrieved and issued. The second system would be in charge of displaying a configurable dashboard where the user can monitor and actuate the IoT devices.

7.1. Raspberry Pi setup

For the initial phase of this project and the Proof-of-Concept (PoC) stage, I decided to go with a Raspberry Pi 3 model B because I had it already available and it provides a quick and easy solution for deployment scenarios like these, especially when considering the cost of the device itself is relatively low.

7.1.1. Installing openHAB

The first setup of the Raspberry Pi 3 model B involved a ready-made image of a the Raspbian Lite Linux distribution [30] made available by the openHAB project called “openHABian”, which takes care of all setup phases and initialization via shell scripts available in the image. The only steps necessary to deploy a working instance of openHAB on a Raspberry Pi using this method is to write the openHABian image file available on the openHAB project page [31] to an SD card and insert the card into the slot of the Raspberry Pi.

After that, I needed to connect the Raspberry Pi to the Internet, which can be either by connecting a UDP cable, or if the Raspberry Pi has Wi-Fi capabilities, the service-set identifier (SSID) of the network and the pre-shared-key (PSK) should be filled in in the openhabian.conf file on the SD card before inserting it into the Raspberry Pi as values for the wifi_ssid="My Wi-Fi SSID" and wifi_psk="password123" keys respectively.

The next step is to connect power to the Raspberry Pi and wait for approximately 15 to 45 minutes, depending on the speed of the Internet connection, for the openHAB instance to be installed on the Raspberry Pi. The installation progress can be monitored by using

the http://openhabian or http://openhab URL, depending on the release of the image, or by connecting to the local IP address of the Raspberry Pi, also via HTTP.

There is a downside to this alleviated installation of the openHABian operating system on the Raspberry Pi since it is configured with default usernames and passwords. Luckily these can be easily changed by using the openHABian Configuration Tool. This tool can be accessed by running the "sudo openhabian-config" command via the console.

Alternatively, openHABian can be installed on other Linux distributions much like we would install any program. The only prerequisite for this is to have git installed on our operating system since we need to pull the openHABian repository hosted on GitHub.

The commands to manually install openHABian are the following:

```
sudo git clone https://github.com/openhab/openhabian.git /opt/openhabian
sudo ln -s /opt/openhabian/openhabian-setup.sh /usr/local/bin/openhabian-config
sudo openhabian-config
```

7.1.2. Configuring openHAB

To enable some of the necessary functional requirements of our system, openHAB needs to be configured. Mainly, we are concerned with two facets of the system: enabling Cross-Origin Resource Sharing (CORS) and Data Persistence to the database located via a JDBC URL.

CORS is a security mechanism implemented and enforced by most modern web browsers that restricts a web application running at one origin from making requests outside that origin. Additional HTTP headers are used when enforcing this policy, the so-called pre-flight requests using the OPTIONS HTTP header that help determine whether the original request is deemed safe or not by the server. If the server has determined that the request is safe, it will allow it, otherwise it will block it. [32]

We need to enable CORS on openHABian since its resources will be accessed from web applications (websites) hosted on another domain, in our case for testing that would be localhost, but with a different port, which counts as a different domain, and ultimately from the productive web application hosted on SAP Cloud Platfrom which is also considered as a different domain. To achieve this, depending on the release of openHAB, we need to modify the openhab2/services.cfg file and append the following line:

```
org.eclipse.smarthome.cors:enable=true
```

To enable persistence for our thing data to a designated URL first we need to enable the a suitable JDBC driver specific to the type of database that we are going to persist the data to and a configuration file containing the JDBC URL of the database, along with the username and password of the designated user in the database.

Because the database used in this scenario will be SAP HANA, we need to download the specific HANA drivers in order to make persistence possible. These drivers are available at https://mvnrepository.com/artifact/com.sap.cloud.db.jdbc/ngdbc.

Once downloaded, we can place the ndgbc.jar file in the openhab2/runtime/lib/ext directory and the needed configuration in the services/jdbc.cfg file as per the requirements listed here: https://www.openhab.org/addons/persistence/jdbc/.

In order to be able to configure the Raspberry Pi and the openHAB instance running on it to connect to an instance of myopenHAB running on the cloud, the openHAB Cloud Connector extension should be installed and configured. This is done easily by using openHAB's Paper UI, where in the Add-ons menu in the "MISC" tab we can select to install the openHAB Cloud Connector extension.

The next step is to configure the openHAB Cloud Connector to communicate to a service of myopenHAB running in the cloud. By default, there is a free option that is hosted by the openHAB Foundation that can be found at https://www.myopenhab.org/.

If using the option provided by the openHAB Foundation, it is needed to register and/or login into the web application and provide two crucial files from the openHAB system running on our device acting as the controller. The two files are named "UUID" and "Secret" and they can be found in the /var/lib/openhab2/uuid and the /var/lib/openhab2/openhabcloud/secret directories respectively. They both contain strings which should be copied and pasted into the running myopenHAB cloud instance via the web UI as they make sure that the communication between the cloud service and the controller device is working and encrypted.

7.1.3. Installing SAP Cloud Connector

The SAP Cloud Connector is a tool that enables connectivity between cloud and on-premise systems. The connector supports both HTTP and RFC[1] protocols. The SAP Cloud Connector can be downloaded from https://tools.hana.ondemand.com/#cloud. After extracting the archive, we can run the Cloud Connector from the go.sh script and a short while later, we can access the web UI from the IP address of the machine it was installed on, on port 8443 by default. The port can be changed by running the changeport.sh script. For the first time login, we use the username: Administrator and password: manage.

Next, we need to configure the SAP Cloud Connector to expose the cloud-hosted HANA database instance to the local machine via a JDBC URL, so that we can use that URL to persist the thing data to the database. To do so, first we need to define the SAP Cloud Platform subaccount that hosts the database that we want to connect to. After successfully connecting to the subaccount hosting the database, we can now see the "On-Premise To Cloud" option in the Cloud Connector web interface. Navigating to this part, we can now create a new service channel that would expose the HANA database to the local machine

[1] Remote Function Call – Proprietary protocol developed by SAP SE.

hosting the Cloud Connector. The service channel should point to the HANA database instance in question and should specify the instance number. This number can be any number in the range between 00 and 99 if it is not occupied by any other connections. After the successful configuration, a JDBC URL will be available on the local machine, which can be used to connect to the HANA database instance running on SAP Cloud Platform. This URL is used in the jdbc.cfg configuration file of openHAB to persist the data via the SAP HANA JDBC Driver.

7.2. Setting up the IoT devices

7.2.1. IKEA TRÅDFRI

The IKEA TRÅDFRI devices are quite easy to set up, especially when aided by the gateway that can connect to a local network. A router dedicated to host the IoT network was set up and a static IP address was assigned to the TRÅDFRI gateway. The lightbulbs, power sockets were paired using a switch remote controller, after which the switch is no longer needed, and the devices can accept commands directly from the gateway. OpenHAB auto-detected the TRÅDFRI gateway and paired devices.

7.2.2. Flower sensors

The Xiaomi Mi Flora sensors required some setting up before their readings could be sent to openHAB and observed by the system. Initially, I paired the sensors using Bluetooth to a smartphone by using the Flower Care [33] application. This initiated a firmware update where the sensors were updated to the latest firmware version provided by the vendor.

The next step was pairing the sensors to the Raspberry Pi, which was made available through the Bluetooth Low Energy interface on the Raspberry pi. To alleviate the process, I used the following MQTT daemon client: https://github.com/ThomDietrich/miflora-mqtt-daemon. The MQTT client connects to the MQTT broker made available on openHAB by default and it uses this channel to publish messages in the appropriate topics. Additionally, to properly parse and format the data, I installed the JsonPath Transformation extension [34] on openHAB.

After installing all of the required dependencies, I could run the “sudo hcitool lescan” command from the Raspberry Pi SSH console to scan the reachable Bluetooth devices in the nearby vicinity and get their physical MAC addresses. The scan revealed the two Xiaomi Mi Flora Sensors and their details, which were put into the “config.ini” file of the MQTT client.

Running the MQTT client revealed that the sensors are reachable, and their measurements are published to the proper MQTT topics. The final step in onboarding the sensors was to manually add the sensors as things to the system by creating a file each in the things and items directories of the openHAB installation, holding the details of the sensors. To

ensure that the MQTT client would run even if the Raspberry Pi is restarted, I registered the MQTT client service as a daemon that would run in the background.

7.2.3. Water pumps

The Atman AT-102 water pumps were connected to a TRÅDFRI power socket which enables them to be turned on and off. As the pumps are certified for specific flowrates, the amount of water pumped by the water can be controlled by specifying the time the specific pump will be turned on for.

7.2.4. Lights

In addition to the TRÅDFRI dimmable light, an OSRAM Fluora and a Philips LED CorePro lights were set up. The lights' power is provided by a male power plug that plugs into a TRÅDFRI power socket, therefore allowing the lights to be turned on and off by the system.

7.3. SAP Converged Cloud setup

7.3.1. Configuring the VM

A virtual machine is needed to host an instance of the myopenHAB service that provides remote control capabilities for the IoT system running on the openHAB platform. One of the choices for virtual machines provided by SAP in the SAP Converged Cloud is an Ubuntu 18.04 image. There exist other alternatives for choosing a VM image such as SUSE Linux Enterprise, Debian, consumer and server versions of Windows, etc. Due to my familiarity with the Ubuntu operating system, I chose to create a virtual machine running Ubuntu. The setup of the virtual machine is quite straight forward, with a lot of functionality already available out-of-the-box. The only things that have to be done during the initialization is to update the package repositories of the Ubuntu package manager.

7.3.2. Enabling HTTPS connections

A very important thing when setting up a web server that communicates through HTTP is to enable end-to-end encryption in order to stop malicious parties gaining access to sensitive data via man-in-the-middle attacks. In order to achieve this, HTTPS needs to be set up on the web server. The web server used by myopenHAB-cloud is nginx, which doubles its functionality as a reverse proxy.

The steps of creating SSL/TLS certificates is described in detail. For the sake of argument and not to leak any sensitive data, the domain name "example.com" will be used hereinafter.

The letsencrypt and ngingx packages need to be installed from the APT package repository. Letsencrypt is a service that provides free-of-charge TLS certificates and it is relatively easy to set up. The following command will generate the needed certificates and place them in the appropriate directory:

```
sudo letsencrypt certonly -a webroot
--webroot-path=/var/www/example.com -d example.com
```

Next, it is needed to configure the nginx server to reroute all HTTP traffic to HTTPS and use the certificates we have just created to establish encrypted channels for doing so. The sites-enabled nginx directory should contain a text configuration file matching our domain name, such as "/etc/nginx/sites-enabled/example.com". That file's contents should look like the configuration file in appendix 1.

The configuration takes care of all security settings and additionally it provides a reverse proxy which will be used to access the resources of the web application being run by NodeJS on port 3000.

7.3.3. Installing myopenHAB (openHAB-cloud)

The first step when installing the openHAB-cloud package is to clone the git repository available on the following link: https://github.com/openhab/openhab-cloud into the created /var/www/example.com/ directory. As seen from the documentation, the we will additionally need, Redis, MongoDB, Express.js, and python to be able to run openHAB-cloud on our VM.

Upon reading the provided documentation, we can see that it has been developed using NodeJS version 7.10.1. To install the appropriate version of NodeJS, I used the tool called Node Version Manager (NVM). This allowed me to select the specific version on which the openHAB-cloud service was developed, as to enable the best support possible.

After confirming that the appropriate version of NodeJS has been installed, we can proceed to install the package dependencies for openHAB-cloud by using the "npm install" command. Shortly after, the dependencies should be downloaded and the service should be ready to use after making some changes in the config.json file that can be found in appendix 2. The "registration_enabled" property is set to true in order to enable initial registration of a user authenticated by a password. After configuring the desired username and password, this property can be set to false.

To start the NodeJS application and the nginx server, the following commands should be executed:

```
sudo service nginx restart
sudo node /var/www/example.com/app.js
```

7.3.4. Establishing connections

To enable the system running openHAB to communicate to our set-up openHAB-cloud instance, we have to provide the UUID and Secret as before, with the only change that now the URL in the controller configuration itself points to our domain and the cloud instance running on it. Everything else works as it did with the free solution provided by the openHAB Foundation.

7.4. Developing the SAPUI5 application

7.4.1. Data binding

During the initial phase of familiarization with the already existing codebase, I had noticed that one of the constraints of SAPUI5, namely proper data binding was not properly implemented. Instead, the former developers relied on accessing elements by their IDs and updating their values manually, which leads to a lot of redundant, non-elegant and unmaintainable code.

To remedy this, I implemented several things. Instead of loading the mock data directly by a JSON object in the code, I created a separate file containing only the data. Using the JSONModel provided by SAPUI5, I simply loaded the data from the file into the JSONModel, giving me and potentially other developers in the future access to many of the lifecycle methods, events and binding capabilities of that class.

Next, I modified the model to reference the already created JSONModel and simply put bindings where previously ID literals would be defined. Where the data model contained a list of objects, instead of writing redundant controller code, I simply used the aggregation binding provided by SAPUI5. Although ultimately not needed, this refactoring reduced the amount of lines in the code, resulting in more elegant and guideline-compliant accessing of data, reducing both technical debt and code smells.

7.4.2. Routing

The navigation toolbar providing the navigation menus on the left side was already implemented during the previous phase of the project, however, there were two major problems with the way it had been engineered.

Firstly, the navigation was not done with a Router, opposite to what the development guidelines of SAPUI5 suggest, the “viewId” string and the appending of the key of the selected item are fragile and can fail because it was using internal naming created by the framework that could change at any time. To remedy this, I defined a variable Router on the controller level and referenced the Router component of the app to it.

Secondly, all MVC components were referenced by the navigation toolbar view, resulting in them being created and loaded, therefore slowing down the startup times and the performance of the web application.

To remedy this issue, I created routes for every specific view in the manifest.json file of the application, where each route held the view name as a property. After that, I removed the MVC references from the view and put an App in order to enable routing. Since a Router is a specific to an App, this meant that in the controller, I could simply navigate to a route matching a selected entry in the menu and the respective MVC would then be loaded and applied to the App and consequently the screen.

Two additional benefits were that no unnecessary views, controllers and models were created until the user requested them by clicking on the appropriate entries in the

navigation menu and that now the browser navigation buttons worked and could route the user through the screens of the application. This resulted in significant improvements of the performance of the application, where now it was loading in under a second and a half, as opposed to the previous solution that was loading in approximately 10 seconds, giving an improvement of loading speeds of at least 5 times.

7.4.3. Updating items' states

One critical aspect of the entire application, especially when taking into account that it is supposed to deal with a live system of potentially hundreds or even thousands of IoT devices and their respective states, is to display real-time and up-to-date data on the user interface. Initially, the controller code that was in charge of handling the data used a polling method.

One of the disadvantages using this approach is that the requests sent to the backend API do not depend on changes in the devices' states, and that in theory, should this be implemented like so, it is possible to be handling an out of date state of a device in the UI, potentially leading to system instability and unwanted results. Another disadvantage is that the state of all items might not change during a time reasonably longer than 1000 milliseconds and that means that all of the requests to the backend API and the Model refresh performed in the browser are using both computing and networking resources which do not have to be used.

In order to fix this problem and provide a better update mechanism, I investigated a couple of other, more efficient refresh mechanisms. In the first step of the research, a technology called WebSockets [35] caught my attention. Put briefly, WebSockets allow for two-way communication between a server and a client (in our case the web application running in the browser). Since it is an event driven mechanism, it is possible to send any interested party event data when something of interest occurs, for example when a device's status changes.

After doing some investigation regarding openHAB's ability to allow for WebSocket connections, I discovered that even though they are not supported at the current time, a very useful compromise, Server-Side Events (SSE) are indeed provided by the openHAB REST API. Server-Side Events, in contrast to WebSockets are only unidirectional, meaning that an application is able to subscribe to a Server-Side Event, however, it is not able to send data using the same channel. This would usually be a problem when implementing applications such as an online instant chat, however, for this use case, especially because it is very rare that the user actuates a device manually instead of relying on automation, I deemed the SSEs already available in the openHAB system as good enough to get the job done and solve this problem.

Unfortunately, I did run into some problems while using this approach. The initial way how I had set up the Server-Sent events was that in the onInit lifecycle of any controller,

the application would subscribe using a JavaScript EventSource to a specific URL of the backend API, depending on the topic of interest of that controller. So, for example, the controller in charge of Thing configuration, would subscribe to the thing topic of openHAB, the Extensions controller to the extensions topic and so on. When that EventSource received a message from the backend, it would call into the refresh method and update the data model of the particular screen.

When ultimately the complexity of the entire application increased, I ran into problems while trying to fetch from or send some commands to the backend API using HTTP. Upon some investigation, I discovered that every time a JavaScript EventSource is established, it opens a TCP/IP network socket that it uses to receive the updates from the server. However, most browsers allow for up to 6 of these connections at a given time, so if I had more screens loaded, they would open and take up an additional network socket, ultimately reaching the maximum number, after which communication with the backend by any means other than the already-existing TCP/IP sockets for the Server-Side Events was not allowed by the browser.

This limitation of the browser's ability to communicate with the backend was something of very critical nature and an effective solution had to be found rather quickly. Luckily, the SAPUI5 framework comes with an EventBus class [36], which allows programmers to leverage the publish-subscribe pattern by using this implementation. Due to the fact that it is possible to subscribe to all openHAB topics using a single EventSource and therefore using only a single TCP/IP network socket, I had figured that it is possible to write some logic inside my web application that would leverage the SAPUI5's own EventBus and send the messages to all interested subscribers internally within the browser, rather than using separate network sockets, ultimately blocking the communication with the backend and breaking the application.

I decided that it would be best to implement this logic in the only controller that is in charge of the menu on the left side of the application. My reasoning behind this was that this is the part of the application that will be visible and instantiated under any circumstance, so logic which is meant to function regardless of what is displayed on the other part of the browser window should reside there.

This makes it that the events being sent from the server are handled at a single point of the application, and the application itself is in charge of demultiplexing the events and propagating them through different channels in the application, according to their topic.

```
onInit: function () {
  …
  const oEventBus = base.getOwnerComponent().getEventBus();
  oEventBus.subscribe("things", "stateChanged", base.updateModel, this);
}
```

The previous snippet of code extracted from the Things controller is in charge subscribing to the internal EventBus, which also holds a pointer to a function that would be executed as a response to the event. Using this approach, the application only refreshes relevant data when there has been an actual change, rather than constantly polling the backend at a specified interval.

7.4.4. Configurable dashboards

One of the most important features in an application such as this is providing the users with an opportunity to setup and configure dashboards with custom controls. After a long process of research and consideration, I decided that I was going to use the GridList [37] to achieve the wanted behavior. What the GridList offers is a flexible layout, plus the option to drag and drop components by using the DragAndDrop implementation of SAPUI5.

To allow an item to be dragged and dropped to and from the GridList, the view's configuration was modified as shown in appendix 3. This enables the items' layout to be changed without manipulating any renderable components, but only relying on the data model to make the needed changes while the view's template takes care of the rendering.

Additionally, the layout of the dashboard is persisted into the browser's local storage, meaning that the user settings will get saved even if the browser is closed. This aspect of saving the layout data was chosen since it does not involve storing any identifiable or sensitive user data.

To add a new control for a device, the user can click a button, which opens a dialog box where the user can select the device in question and according to its capabilities, the user can choose which controls get put into the GridListItem. For example, if there would be a dimmable light, the user can choose to have a Slider to control the intensity of the light, a Switch to turn the light on or off, both, or none.

To edit the layout, the user can press a button on the screen, which changes the mode property of the GridList containing the device controls to Delete. When in this mode, each list item is rendered with an X button on its right corner, making it easy and intuitive on how to delete a control from the GridList. When the user wants to exit the Delete mode of the GridList, they simply click the same button that now displays the text "Done Editing" to return the GridList into its previous state.

7.5. SAP Cloud Platform setup

7.5.1. Configuring destinations

In order for the SAPUI5 application to be able to reach the openHAB API without the requests being blocked by CORS, it is needed to configure a destination in the SAP Cloud Platform Cockpit. The destination is of type HTTP and it points to the URL of the running

openHAB-cloud service. The requests get authorized by using Basic authentication, which contains the username (email address) and password of the user in the openHAB-cloud service.

7.5.2. Deploying the SAPUI5 application

SAP WebIDE provides a way to clone the repository via git to SAP Cloud Platform, from where the application can be deployed. Upon cloning the repository, there is only one thing remaining before the application can be deployed, and that is to make the destination that we had just created available to the UI5 JavaScript code. To do that, the neo-app.json file should contain the following route:

```
"routes": [
    {
        "path": "/api",
        "target": {
            "type": "destination",
            "name": "IOT_API"
        }
    }
]
```

The openHAB API is now available by using the “/api” call from the front-end.

7.6. Development summary

Throughout the development process, I created a system whose physical devices are housed in a plant box, acting as a demonstration of what can be achieved by using technologies such as the Internet of Things and the cloud.

From the web application, an operator is able to control the environment of plants by actuating devices such as lights and watering systems and the system and to observe real-time data gathered from the sensors. The user interface of this system is available on the Internet, allowing for remote control of the system. This is possible because the web application, as well as the myopenHAB component, which connects the Raspberry Pi running the openHAB platform, are hosted on SAP Cloud Platform.

The openHAB platform allows for IoT devices and gateways to be connected and interacted with. Additionally, openHAB hosted on the Raspberry Pi is able to persist the data gathered to an SAP HANA database via the Cloud Connector and the SAP HANA JDBC Drivers.

Figure 7.1 The Deployed System

The above is a picture of the deployed system on the premises of SAP Hungary. It resides in the lobby of one of the buildings and is accessible by employees of SAP and any external visitors as it is in a part of the building which is open to the public.

8. Testing

The system is accessible on the premises of SAP Hungary Kft., namely in the lobby of the S4-S5 building in Graphisoft park. It serves as an interactive demo where the users can interface with the system via a tablet running the SAPUI5 application. A user can see and update the configuration of the connected things, bindings and rules, as well as they can interact with the devices from the dashboards.

Due to the fact that the source code of the application was written while working for SAP, SAP holds the intellectual properties of the source code and it cannot be made publicly accessible. A video enclosed in the DVD attached to this thesis showcases the core functionalities of the system.

In order to be able to onboard a new IoT device, referred to as a "Thing", firstly, the user needs to install a binding from the list of supported Bindings. This is available under the "Available Things" navigation bar item. Once the binding has been installed, the user can initiate a scan using that binding from the "Discovered Things" component of the web application. The scan will look for things that are available through that binding. These things will appear in the list of discovered things, and from there they can be either approved or ignored.

When things are onboarded into the system, they are available for configuration into the dashboards. Upon clicking on the "Add New Control" button, a dialog appears that asks the user to select the thing wishing to create a control for. According to the capabilities of the thing, the user is given options to choose which capability a control should be built for.

The following image depicts the testing strategy of SAPUI5 web applications:

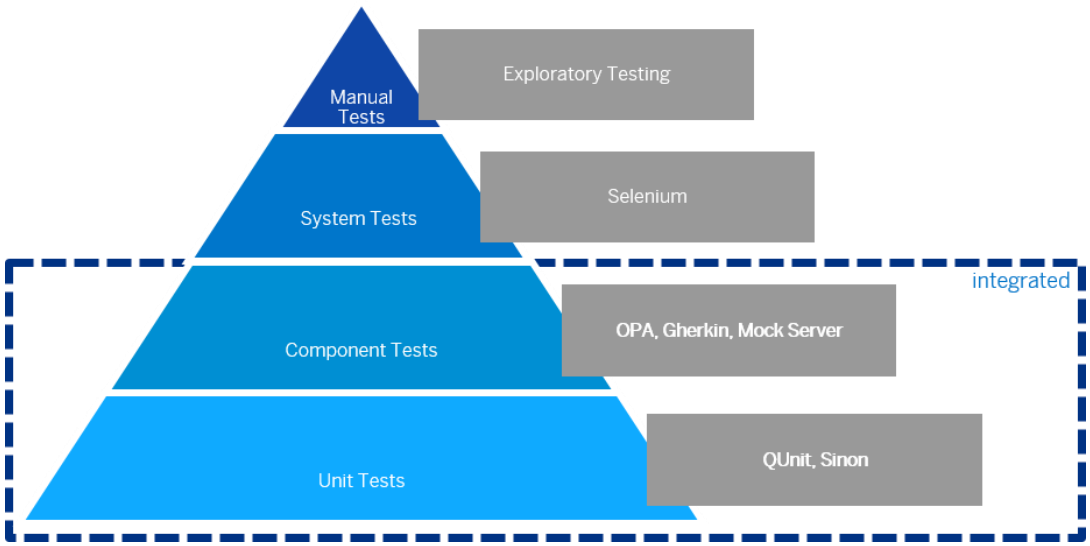

Figure 8.1 Testing Pyramid in SAPUI5

Instructions on how to run unit and integration tests and best practices are documented in the SAPUI5 online tutorials. [38] I followed these tutorials and related documentation to ensure that I do the tests of this application to the best of my ability and as compliant with the testing strategy as possible.

8.1. Mocking the backend API

In order to be able to run unit tests, I needed to have some kind of sample data. While the data can be provided purely as a JSON file, I decided to use one of the utilities available in SAPUI5, namely the Mock Server [39]. To set it up, I used the following snippet of code.

```
var oMockServer = new MockServer({
    rootUri: "https://www.backend.mock/",
    requests: [{
        method: "GET",
        path: "/rest/things",
        response: function(xhr) {
            var oMockModel = new JSONModel();
            oMockModel.loadData("mockData/Things.json", {}, false);
            xhr.respondJSON(200, {}, oMockModel.getJSON());
        }
    }]
});
oMockServer.start();
```

A similar strategy was implemented for providing mock API calls that return significant data for all other units under test and the expected responses the successful execution of the test depends on.

8.2. Unit testing

Unit tests were carried out with the help of the QUnit JavaScript testing framework [40]. A part of its capabilities is the possibility to run asynchronous tests, which makes it easy to test specific features of the web application that require asynchronous loading of data, such as loading a JSON data model from a file containing a test data set.

Since this project uses many components (units) made available by the SAPUI5 framework which have been tested by SAP, the amount of unit tests is comparatively lower to that of the integration tests covered in the next chapter.

Specific tests for the filtering functions in the MVCs responsible for listing things, bindings and rules were carried out. Possible cases where these tests can fail are when the provided data does not contain the properties being filtered, however, special attention was given to the testing data set to include as many cases as possible.

Similarly, the functions in charge of counting the specific types corresponding to the filters of the tab bar were tested using a similar method.

8.3. Integration testing

The integration tests were done by using the One Page Acceptance tests for SAPUI5 (OPA5) framework. The top-level test component is called a Journey and it contains the actual tests that will be run. Its purpose is to simulate how an end-user would interact with the application, where the user can do a specific action on which assertions are being made with the goal of evaluating whether the behavior of the application was as it had been expected.

The journeys hold references to Pages, which are the page objects which contain the part of the web application that is being tested. These pages in turn, hold the given actions and assertions for the specific part of the UI, most commonly a single MVC, which can be reused for different journeys

Additionally, a journey can contain arrangements, which are usually used to control the lifecycle of the web application, since the web application itself must be loaded in order to run the OPA5 tests. Due to the simplicity of the application, the arrangements of my testing suite only take care of starting up and shutting (tearing) down the application.

8.3.1. Navigation Journey

The navigation journey test is a simple test designed to make sure that the routing in navigation in this web application work as intended. The main actions in this journey are being done in the MainView MVC component, as it houses the navigation logic. This journey goes through all navigation pages of the application and makes sure that they are being created as requested and subsequently loaded on the screen. This integration test can fail if there are errors with the view declarations in their respective XML files, if there are syntax errors in the controller of a requested view, or if the respective view/controller does not exist.

8.3.2. Bindings CRUD Journey

The bindings CRUD (Create, Read, Update, Delete) Journey is concerned about the user's interaction when they wish to perform any of the aforementioned actions on the bindings. To evaluate the system's behavior and achieve as real-world results as possible, I started up a local openHAB instance on my Windows PC, using it as a mock for the final system and added things with different types of statuses.

The journey covers navigation to the "Available Bindings" page, checking whether the given filters work and additionally the cases where a user could install or uninstall a binding. These two cases of downloading and uninstalling a binding come with two sub-cases, where the selected binding has already been both installed and uninstalled. The

downloadable bindings cannot be uninstalled and logically, the installed bindings cannot be installed once more. In both situations, it is expected that a MessageToast control pops up and notifies the user of an action that cannot be performed.

This journey's test cases can possibly fail if the system that we test with has no bindings of a specific type or no bindings at all.

8.3.3. Things CRUD Journey

The things CRUD integration testing journey shares a significant part of its scope with the bindings CRUD journey. They differ in the part that when the add new thing button is pressed, the application should navigate to the Discovered Things component so that the user can choose one of the already discovered things or they can use one of the available bindings to scan for a new thing.

8.3.4. Things Discovery Journey

In order to test whether new things can be discovered, I created the things discovery Journey. It represents a relatively short user activity, where the user navigates to the Discovered Things page and can select to scan for new things by using one of the installed bindings. Moreover, like in the previous Journeys, the filtering functions of the list displaying the discovered things was also tested.

8.4. Evaluation

In total, 12 QUnit unit tests using a mock backend API were conducted. Additionally, 4 OPA5 Journeys consisting of 31 integration tests were also conducted. From the tests, it can be observed that the system fulfills the requirements and acceptance criteria.

Compared to other solutions researched in the bibliography, the running costs of this system are relatively lower as it uses commodity hardware and open source software. Moreover, this system has great potential in showing how SAP technologies can be integrated with other non-SAP systems and can provide a demonstration where users and observers can see their interactions with the web application turn into real-life results such as actuating a connected device's capability.

9. Conclusion

During the development of the project, I faced a significant number of challenges of varying difficulty and of different nature. The challenges spanned different domains such as software design and development, network engineering and security, and device and system management.

Due to the attributes of the thesis and the technologies used, I found it very entertaining to work on it, which made me feel motivated and empowered to overcome the challenges arising from the working process. Through the planning and development stages, I gained significant knowledge on how a project such as the one covered in this work should be carried out.

Moreover, I learned many new and useful things regarding web application development, specifically developing applications with the help of the SAPUI5 framework, as well as how both Internet of Things and cloud systems work and how to enable their secure integration to create a system that works as intended.

10.List of figures

11.References

[1] Y. Lafon, "HTTP - Hypertext Transfer Protocol," 2003. [Online]. Available: https://www.w3.org/Protocols/. [Accessed 13 December 2019].

[2] R. Fielding, "Hypertext Transfer Protocol -- HTTP/1.1," June 1999. [Online]. Available: https://tools.ietf.org/html/rfc2616. [Accessed 13 December 2019].

[3] Sun Microsystems, Inc., "RPC: Remote Procedure Call Protocol Specification Version 2," June 1998. [Online]. Available: https://tools.ietf.org/html/rfc1057. [Accessed 13 December 2019].

[4] R. Light, "MQTT man page," [Online]. Available: http://mosquitto.org/man/mqtt-7.html. [Accessed 13 December 2019].

[5] OASIS, "MQTT Version 5.0," 07 March 2019. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html. [Accessed 13 December 2019].

[6] A. Krasnopolski, "MQTT Servers/Brokers," 08 August 2019. [Online]. Available: https://github.com/mqtt/mqtt.github.io/wiki/servers. [Accessed 13 December 2019].

[7] M. Tillman and C. Hall, "What is ZigBee and why is it important for your smart home?," 15 October 2019. [Online]. Available: https://www.pocket-lint.com/smart-home/news/129857-what-is-zigbee-and-why-is-it-important-for-your-smart-home. [Accessed 13 December 2019].

[8] F-Secure, "An overview of ZigBee networks," [Online]. Available: https://www.f-secure.com/content/dam/f-secure/en/consulting/our-thinking/collaterals/digital/f-secure-zigbee-overview.pdf. [Accessed 13 December 2019].

[9] P. McDermott-Wells, "What is Bluetooth?," *IEEE Potentials*, pp. 33-35, 20 December 2004.

[10] Bluetooth SIG, "Bluetooth Low Energy," 03 December 2016. [Online]. Available: https://web.archive.org/web/20170310111443/https:/www.bluetooth.com/what-is-bluetooth-technology/how-it-works/low-energy. [Accessed 13 December 2019].

[11] N. Rockershousen, "GridConnect," 03 October 2016. [Online]. Available: https://blog.gridconnect.com/blog/general/the-internet-of-things-and-bluetooth. [Accessed 13 December 2019].

[12] E. Upton, "Ten millionth Raspberry Pi, and a new kit," 08 September 2016. [Online]. Available: https://www.raspberrypi.org/blog/ten-millionth-raspberry-pi-new-kit/. [Accessed 13 December 2019].

[13] R. Barnes, "Raspberry Pi 3: Specs, benchmarks & testing," 2016. [Online]. Available: https://magpi.raspberrypi.org/articles/raspberry-pi-3-specs-benchmarks. [Accessed 13 December 2019].

[14] Raspberry Pi Foundation, "Downloads," [Online]. Available: https://www.raspberrypi.org/downloads/. [Accessed 13 December 2019].

[15] TECHBASE Group Sp. z o.o., "Modberry," [Online]. Available: https://modberry.techbase.eu/. [Accessed 13 December 2019].

[16] Xiaomi Global Community, "Xiaomi Huahuacaocao Flower Plants Smart Monitor," [Online]. Available: https://files.xiaomi-mi.com/files/plants_monitor/Plants_monitor-EN.pdf. [Accessed 13 December 2019].

[17] EspenT, "Miflora broken after upgrading to 0.97," 13 August 2019. [Online]. Available: https://community.home-assistant.io/t/miflora-broken-after-upgrading-to-0-97/131653. [Accessed 13 December 2019].

[18] C. Donnelly, "Cloud computing: Past, present and future," 23 March 2018. [Online]. Available: https://www.computerweekly.com/blog/Ahead-in-the-Clouds/Cloud-computing-Past-present-and-future. [Accessed 13 December 2019].

[19] B. Black, "EC2 Origins," 25 January 2009. [Online]. Available: http://blog.b3k.us/2009/01/25/ec2-origins.html. [Accessed 13 December 2019].

[20] Amazon Web Services, Inc, "How does AWS IoT Core work?," 2019, [Online]. Available: https://aws.amazon.com/iot-core/. [Accessed 13 December 2019].

[21] Google Cloud, "Products and services," 2019. [Online]. Available: https://cloud.google.com/products/. [Accessed 13 December 2019].

[22] Google Cloud, "Cloud IoT Core," 2019. [Online]. Available: https://cloud.google.com/iot-core/. [Accessed 13 December 2019].

[23] Microsoft, "Azure solutions," 2019. [Online]. Available: https://azure.microsoft.com/en-us/solutions/. [Accessed 13 December 2019].

[24] Microsoft, "Azure IoT Central," 2019. [Online]. Available: https://azure.microsoft.com/en-us/services/iot-central/. [Accessed 13 December 2019].

[25] Microsoft, "Azure Digital Twins," 2019. [Online]. Available: https://azure.microsoft.com/en-us/services/digital-twins/. [Accessed 13 December 2019].

[26] SAP SE, "Product Capabilities | SAP Cloud Platform," [Online]. Available: https://www.sap.com/products/cloud-platform/capabilities.html. [Accessed 13 December 2019].

[27] S. Sundaravaradan, "SAP IoT Application Enablement - What is it?," 29 August 2017. [Online]. Available: https://blogs.sap.com/2017/08/29/sap-iot-application-enablement-what-is-it/. [Accessed 13 December 2019].

[28] openHAB Foundation, "Who We Are," 2019. [Online]. Available: https://www.openhab.org/about/who-we-are.html. [Accessed 13 December 2019].

[29] openHAB Foundation, "Become a Member," 2019. [Online]. Available: https://www.openhabfoundation.org/join/. [Accessed 13 December 2019].

[30] Raspberry Pi Foundation, "Raspbian," 2019. [Online]. Available: https://www.raspberrypi.org/downloads/raspbian. [Accessed 13 December 2019].

[31] openHAB Foundation, "openHABian - Hassle-free openHAB Setup," 2019. [Online]. Available: https://www.openhab.org/docs/installation/openhabian.html. [Accessed 13 December 2019].

[32] Mozilla , "Cross-Origin Resource Sharing (CORS)," [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS. [Accessed 13 December 2019].

[33] Beijing HHCC Plant Technology Co., Ltd, "Google Play Store," 05 December 2019. [Online]. Available: https://play.google.com/store/apps/details?id=com.huahuacaocao.flowercare. [Accessed 13 December 2019].

[34] openHAB Foundation, "JsonPath Transformation Service," 2019. [Online]. Available: https://www.openhab.org/addons/transformations/jsonpath/. [Accessed 13 December 2019].

[35] Mozilla, "The WebSocket API (WebSockets)," 28 November 2019. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API. [Accessed 13 December 2019].

[36] SAP SE, "SAPUI5 Reference Documentation - EventBus," November 2019. [Online]. Available: https://sapui5.hana.ondemand.com/#/api/sap.ui.core.EventBus. [Accessed 13 2019 December].

[37] SAP SE, "SAPUI5 Reference Documentation - GridList," 2019. [Online]. Available: https://sapui5.hana.ondemand.com/#/api/sap.f.GridList. [Accessed 13 December 2019].

[38] SAP SE, "SAPUI5 Reference Documentation - Testing," 2019. [Online]. Available: https://sapui5.hana.ondemand.com/#/topic/7cdee404cac441888539ed7bfe076e57. [Accessed 13 December 2019].

[39] SAP SE, "SAPUI5 Reference Documentation - Mock Server," 2019. [Online]. Available: https://sapui5.hana.ondemand.com/#/api/sap.ui.core.util.MockServer. [Accessed 13 December 2019].

[40] The jQuery Foundation, "QUnit: A JavaScript Unit Testing framework.," 2019. [Online]. Available: https://qunitjs.com/. [Accessed 13 December 2019].

12. Appendices

12.1. Appendix 1 – nginx configuration file

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;
    server_name example.com;
    return 301 https://$server_name$request_uri;
}

server {
    listen 443 ssl default_server;
    listen [::]:443 ssl default_server;
    server_name example.com ;

    ssl_certificate /etc/letsencrypt/live/example.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/example.com/privkey.pem;

    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    ssl_prefer_server_ciphers on;
    ssl_ciphers "EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH";
    ssl_ecdh_curve secp384r1;
    ssl_session_cache shared:SSL:10m;
    ssl_session_tickets off;
    ssl_stapling on;
    ssl_stapling_verify on;
    resolver 8.8.8.8 8.8.4.4 valid=300s;
    resolver_timeout 5s;

    add_header Strict-Transport-Security "maxage=63072000;
       includeSubdomains";
    add_header X-Frame-Options DENY;
    add_header X-Content-Type-Options nosniff;

    root /var/www/example.com;
    index index.html index.htm;

    location ~/.well-known {
        allow all;
    }

    location /css {
        alias /var/www/example.com/openhab-cloud/public/css;
    }
```

```
    location /js {
        alias /var/www/example.com/openhab-cloud/public/js;
    }
    location /img {
        alias /var/www/example.com/openhab-cloud/public/img;
    }
    location /bootstrap {
        alias /var/www/example.com/openhab-cloud/public/bootstrap;
    }
    location /font-icons {
        alias /var/www/example.com/openhab-cloud/public/font-icons;
    }
    location /fonts {
        alias /var/www/example.com/openhab-cloud/public/fonts;
    }
    location /js-plugin {
        alias /var/www/example.com/openhab-cloud/public/js-plugin;
    }
    location /downloads {
        alias /var/www/example.com/openhab-cloud/public/downloads;
    }

    location / {
        proxy_pass http://localhost:3000;
        proxy_redirect off;
        proxy_http_version 1.1;
        proxy_set_header Host $host;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto https;
    }
}
```

12.2. Appendix 2 – config.json file for NodeJS

```
{
    "system": {
        "host": "https://example.com",
        "port": "80",
        "protocol": "http",
        "logger": {
            "dir": "./logs",
            "maxFiles": "7d",
            "level": "debug",
            "morganOption": null
        },
        "subDomainCookies": false,
        "muteNotifications": false
    },
    ...
    "mongodb": {
        "hosts": [
            "127.0.0.1"
        ],
        "db": "openhab",
        "username": "username",
        "password": "password"

    },
    "redis": {
        "host": "127.0.0.1",
        "port": "6379",
        "username": "username",
        "password": "password"
    },
    ...
    "registration_enabled": true
}
```

12.3. Appendix 3 – Drag and Drop configuration

The following snippet was added to the List of the SAPUI5 XML view:

```
<f:dragDropConfig>
  <dnd:DragInfo sourceAggregation="items" />
  <dnd-grid:GridDropInfo
           targetAggregation="items" dropPosition="Between"
           dropLayout="Horizontal" drop="onDrop"
           enabled="false" id="dndConfig"/>
</f:dragDropConfig>
```

JavaScript code of the onDrop event handler:

```
onDrop: function (oInfo) {
  var oDragged = oInfo.getParameter("draggedControl"),
      oDropped = oInfo.getParameter("droppedControl"),
      sInsertPosition = oInfo.getParameter("dropPosition"),
      oGrid = oDragged.getParent(),
      oData = oDashboardModel.getData(),
      iDragPosition = oGrid.indexOfItem(oDragged),
      iDropPosition = oGrid.indexOfItem(oDropped);
  var iGridListIndex = oData[iDragPosition];
  oData.splice(iDragPosition, 1);
  if (iDragPosition < iDropPosition) {
    iDropPosition--;
  }
  if (sInsertPosition === "Before") {
    oData.splice(iDropPosition, 0, iGridListIndex);
  } else {
    oData.splice(iDropPosition + 1, 0, iGridListIndex);
  }
  oDashboardModel.setData(oData);
  oStorage.put("DashboardModel", oDashboardModel.getJSON());
}
```

12.4. Appendix 4 – List of tests

Unit tests
Should format the thing status for "ONLINE" things with Success
Should format the thing status for "OFFLINE" things with Error
Should format the thing status for "UNINITIALIZED" things with Warning
Should format the thing status for all other inputs as Information
Should set counts to 2 online, 4 offline, 3 uninitialized
Should set count to 0 if no recognized status present
Should format the binding status for non-installed bindings with Information
Should format the binding status for installed bindings with Success
Should return only bindings from extensions
Should format the newly discovered things with Information
Should format the already approved discovered things with Success
Should format the ignored discovered things with Success

Integration tests
Navigation Journey
Things CRUD Journey
Bindings CRUD Journey
Things Discovery Journey