UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Janar Juusu

Web interface for accelerating IoT device provisioning and configuration at SmartVent

Bachelor's Thesis (9 ECTS)

Supervisor(s): Pelle Jakovits

Tartu 2018

Veebiliides IoT seadme seadistamise kiirendamiseks SmartVendis

Lühikokkuvõte:

SmartVent IoT juhtseadmete ühendamine oli siiani täielikult manuaalne töö, sealhulgas käsitsi kirjutatud andmebaasi kirjed. See lähenemine oli ajakulukas ning oli suur oht vigade tekkimiseks. Lõputöö käigus loodi veebirakendus, mis automatiseerib osa IoT juhtseadme ühendamise protsessist luues automaatselt vajalikud andmebaasi kirjed ning koostab seadme konfiguratsiooni. Kasutajalt küsitakse protsessi käigus andmeid hoone ja selle tehnoseadmete kohta, mille külge juhtseade paigaldatakse. Antud tööriist tegi ühendamise palju kiiremaks, lihtsamaks ning on väiksem oht inimvigade tekkimiseks.

Võtmesõnad:

IoT, automatiseerimine, seadme konfigureerimine, seadme integratsioon

CERCS: P170, Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine

Web interface for accelerating IoT device provisioning and configuration at SmartVent

Abstract:

Setting up SmartVent IoT controllers was completely manual work, including hand-written database queries. This approach was very time-consuming and error-prone. During this thesis a web application was created, which automates a portion of the IoT controller setup by automatically creating the database queries and generating device configuration. The user is prompted to enter details about the building and it's technical systems to which the controller will be installed. This tool made setup process much quicker, simpler and has less potential for user errors.

Keywords:

IoT, automation, device provisioning, device integration

CERCS: P170, Computer science, numerical analysis, systems, control

Table of Contents

1. Introduction

Internet of Things is an increasing trend of connecting things we daily use into the cloud. The main idea is to collect information about our surroundings to understand it and take necessary actions. This can improve everyday user's quality of life and gives a clear overview what else could be made better (Diaz, Martin, & Rubio, 2016).

Devices such as mobile phones have become connected to the cloud very quickly as they are replaced and upgraded every few years. Compared to industrial technical systems this has been very rapid progression, because industrial solutions, e.g. ventilation hardware, are designed to last for much longer, between 20 and 50 years with no major upgrades.

SmartVent is a startup that deals with modernizing buildings' climate and ventilation systems. The intention is to give the building owner a clear overview and status of the building technical hardware as well as air quality, mainly CO2. To achieve this, controllers[1] are introduced to collect this data and send it to a central storage in the cloud, where it can be analyzed and visualized.

Working at an IT startup requires workers to be vigilant and serve their customers on time. Demand of the customers must be satisfied in order to stay competitive on the market, therefore our product must be scalable. In order to achieve that, we are in need of a solution to accelerate our controller provisioning and configuration to minimize the amount of manual work and speed up installation process. As a result we will be able to spend less time with each controller during the setup.

Nowadays device integration is largely manual work as every company has custom architecture and different setup procedures. This is also the case at SmartVent – the architecture has been engineered by themselves using custom and/or self-assembled controllers, there is no ready-to-use compatible integration system directly available and thus a custom solution is needed to solve this issue.

SmartVent is currently struggling with provisioning new devices to their cloud platform. A lot of manual work is involved, for example hand-written database queries. This is very

[1] A controller is a physical gateway, which collects data from technical systems and sends it to the cloud. Depending on the needs and connectivity, it also controls the operation of these systems.

tedious and time-consuming for workers and has high risk for mistakes. This also allows SmartVent to deal with potential sudden increases in the amount of locations that need to be set up.

During this thesis, a web-based software solution for connecting these IoT controllers to SmartVent Cloud will be created. The SmartVent Cloud is an in-house built platform for visualizing data that has been collected from the buildings. In addition to that, a comparison will be made with an existing IoT platform named Cumulocity, which is a commercial platform for IoT devices.

The paper is organized as follows. In this section we have an introduction to general overview of Internet of Things and SmartVent. Section two gives a more in-depth view into the advantages and potential of IoT and IoT platforms. In third chapter we describe the current system architecture that is used in SmartVent. Parts four and five are dedicated to the solution: what are the requirements, technical overview, the final solution and comparison to an existing IoT platform. In chapter six we verify if requirements were fulfilled and whether the solution has brought improvements to the workflow. Finally in section seven we have conclusion and future improvement possibilities.

2. State of the Art

This chapter gives an overview of the state of the art in Internet of Things and what kind of opportunities it can bring. We also take a look at IoT platforms and different configuration management tools regarding device provisioning.

2.1 Internet of Things

Internet of Things (abbreviated IoT) is a global infrastructure of physical and virtual objects, devices, vehicles, buildings and other items embedded with electronics ("things"). These enable advanced services by interconnecting things based on their technologies (Wortmann & Flüchter, 2015). Bringing devices to the cloud gives the possibility to monitor and analyze the daily life and conditions we live in. This also provides the opportunity to provide better services and environment based on the user demands (Gubbi, Buyya, Marusic, & Palaniswami, 2013).

These "things" are often low-powered and have the hardware capability for sending and receiving data, but are resource constrained for larger computational tasks, such as data processing. It is common practice to connect them to the cloud, where the available resources are virtually unlimited.

Most noticeable effects of Internet of Things for a private user could be in domestic and working life, starting from environment monitoring and automated home solutions, such as gates and doors, to intelligent transportation of people and goods and e-health services (Stergiou, Psannis, Kim, & Gupta, 2018). Some examples include:

- Efficient collecting and sharing of healthcare-related information. For instance, a patient's heart rate can be collected by sensors and sent directly to the doctor's office. By using mobile devices, healthcare services can be personalized (Xu, He, & Li, 2014).
- In China, intelligent video cameras and sensor networks are used for real-time environment monitoring for fire alerts, which can then trigger alarms and message rescue services (Ying-cong & Jing, 2013).
- IoT technologies are used for preventing and reducing accidents in mining industry by forecasting disasters and improving communication between surface and underground. Chemical and biological sensors are used for measuring hazardous environments to prevent health issues (Xu, He, & Li, 2014).

2.2 IoT platforms

With the rise of IoT devices, many cloud providers and companies have started offering IoT platforms for users for convenient day-to-day use. These platforms aim to make collecting data from devices as simple as possible without requiring much expertise on the user side.

There are many different kind of platforms. Some provide only the connectivity to the devices, for example Amazon Web Services IoT, whereas others also provide processing power for applications and services, for example IBM Watson IoT Platform. Platforms like Google Nest are built for specific hardware, which support only very limited range of devices (Scully, 2016).

SmartVent was offered to test out IoT platform called Cumulocity. It was recently made available locally in Estonia by Telia and is advertised as "zero model configuration" platform (Cumulocity GmbH, n.d.).

Cumulocity is a fully-featured IoT platform that is designed for collecting and visualizing data from IoT devices, for example air quality sensors, lightbulbs, noise meters (Kaldvee, 2017). They support wide variety of devices. Their provisioning process is very simple thanks to having built necessary tools and applications for devices and making them open-source.

Cumulocity device connecting uses 'plug and play' approach. The user needs to install and run Cumulocity agent on the device. It will then automatically gather the device information and register the device on the cloud. The agent can also detect sub-devices, for example sensors, and configure them automatically as well. The user does not have to carry out any other tasks in the cloud (Cumulocity GmbH, n.d.).

Another advantage for Cumulocity is that there are quite many open-source tools and libraries available. These can be used to enhance user experience within the platform or include support to even more devices (Cumulocity GmbH, n.d.). Finally they have made very extensive API documentation available to everyone to make building new features very simple.

Biggest disadvantages for Cumulocity are it's cost and the inability to host on own infrastructure. Cumulocity is a commercial platform and platform itself is not open-source. Another major drawback is limited possibilities for user-triggerable actions. Cumulocity has

good support for device's built-in functionality and user can create rules for automatic actions, however there does not appear to be a feature for user to manually trigger controls.

Cumulocity does bring some very good features, therefore it was decided to make a more in-depth look at Cumulocity, which can be found in a later section. A comparison will be made with the custom solution and then it can be determined which solution fits SmartVent use cases and needs better.

2.3 Configuration management tools

SmartVent's one of the biggest challenges right now is getting a new IoT device into a fully functional SmartVent controller as fast as possible and with least amount of manual work. There are multiple configuration management tools available, that can provision a new device or even multiple devices quickly and have them running in minutes. These tools also make sure, that every node is running same software in same configuration. This allows for easier maintenance in the future and quicker recovery in case of disasters (Heidi, 2016).

SmartVent is currently using a tool named Ansible for loading configuration to the device. This uploads necessary files from the computer to the device and configures the operating system for security and stability. A disadvantage for Ansible is that it does not store current state. It carries out series of tasks, but does not maintain any information on what is actually configured on the device. In addition to that, Ansible is a command-line tool utility, which is not easy-to-use by less experienced users in the team.

When Ansible is working on push-based approach, where user starts pushing files to the device, then Chef[2], for example, is based on pulling the data from a server. When a new device is added, a piece of specialized software is run on the node, which will download the files and configuration. When an update is published, the device can reconfigure itself automatically, whereas with Ansible user would have to start the configuration reload process manually.

While both approaches are quite positive, then fully basing a device provisioning process on them is not very feasible. SmartVent stores some of the device configuration in a separate

[2] Chef is a configuration management tool for automating IT infrastructure.

database, which neither tool can update in a convenient way. There are other similar tools out there, but they do not solve the problem at hand.

3. Overview of current setup

In this chapter we explain the briefly explain the terminology and technology that are important to this thesis. There is also an in-depth layout of SmartVent architecture, current device provisioning process and it's flaws which are being addressed in this thesis.

3.1 Glossary

A **location** indicates a specific building, where a SmartVent controller has been setup. Location name is either the building name or the address. Optionally, we can also include the location's geographical coordinates and extras, such as links to building's blueprints, schematics and/or the air handling unit's own remote servicing system. Every location is tied to a specific account, which refers the company name in charge of the building and is used for easier overview and user permissions management.

An **area** is the smallest controllable unit in a SmartVent building. It indicates a specific part of the building, for example a room, a floor or a section. This is decided in the planning phase and is highly dependant on the ventilation system setup, for example in some buildings ventilation pipes are only installed per floor.

An **air handling unit** (abbreviated **AHU**) is responsible for circulating indoor air by pulling fresh air in from the outside, pushing it to the rooms (also known as supplying air) and pulling out indoor air (also known as exhaust) and pushing it back outside.

Priority mode means turning all the ventilation hardware in a specific area to the configured maximum for rapid ventilation. This is mostly used in situations, where an uncommon event is about to occur and this area should be ventilated more than other areas. An example is when many people have gathered within a single area and the CO2 level rises quickly.

SmartVent uses air quality **sensors**, that can measure only CO2 or CO2, relative humidity (RH) and temperature. Some of the sensors are connected via physical wiring, while others are connected using wireless protocols such as LoRa or EnOcean, in cases where wiring is not approved by the building manager.

Actuators are hardware devices, that transform input signal to physical movement using an electric motor. In our case, we use Variable Air Volume (shortly VAV) valves, which are controlled by actuators and affect the amount of air moving within the pipe to or from the areas.

ModBus is a serial protocol for communicating with industrial electronic devices. Every sensor, actuator and AHU is given a unique ModBus address depending on the physical wiring. Data is stored in each device's registries and coils and is the data source for the controller. This protocol has very wide compatibility with existing ventilation hardware.

MQTT is a publish/subscribe-based communication protocol, where a message can be published on a certain topic and that message is then forwarded to all subscribers on that topic by the MQTT broker. Compared to other standard protocols, such as HTTP, it is lightweight, making it suitable for low-power or limited network bandwidth environments.

A **controller** or a device is Raspberry Pi-based[3] computer, which is running SmartVent software. The controller is connected to all sensors, actuators and the air handling unit over ModBus protocol. The software reads the data from the sensors and sends all the data to SmartVent Cloud over MQTT. The controller has a specific ID and API key, which is stored on the device and is used for downloading rest of it's configuration from the cloud. One location has single controller.

3.2 Current system layout

A thorough illustration of SmartVent architecture can be seen on figure 1. The controller is setup in the building and is actively sending data to our MQTT broker[4] every 1-2 minutes. MQTT bridge[5] receives the data and saves it to the database, where it can be accessed by our web application for displaying real-time and historical (up to 3 months) air quality measurements, air handling unit statistics and alarms. All commands that are given via web interface (for example, priority mode) are sent to the MQTT bridge, which sends out an MQTT message, that is received by the respective controller, which in turn carries out the command action.

[3] Raspberry Pi is credit-card sized computer for light workloads and low power consumption. https://www.raspberrypi.org/

[4] MQTT broker is the server, where all clients are connected to, and handles messaging between clients.

[5] MQTT bridge is in-house software for connecting database and web application to the MQTT broker.

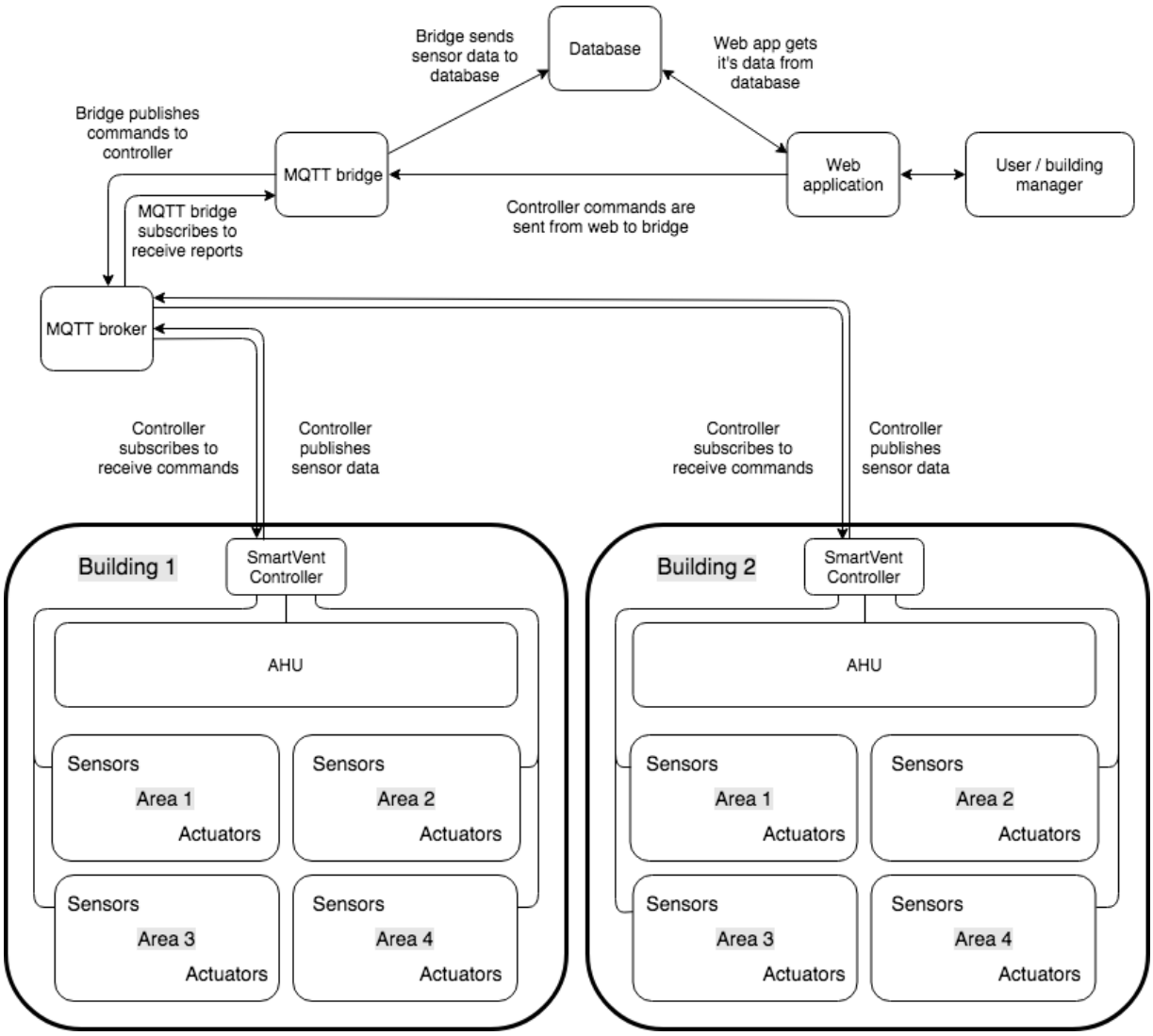

Figure 1. Overview of SmartVent systems architecture.

One of the biggest challenges in this architecture is provisioning devices in new locations. Every building has slightly different configuration and hardware, which means the setup process will be slightly different, which is one of the reasons why we initially stayed with a manual solution for setting up the sites.

Currently the setup process is the following:

- Add a location record to the database.
- Create a random string to be used as controller API key.
- Add a controller record to the database.
- Add all the areas to the database.

- Identify air handling unit, it's connection address and features and create a record in the database.
- Identify actuators and their specifications and add them to the database.
- Identify sensors and their specifications and add them to the database.
- Connect the physical wires between components, pair wireless sensors with receiver.
- Generate VPN[6] and MQTT client certificates.
- Generate device configuration file that includes MQTT and web app URLs, API key, certificate files and ModBus device interface.
- Load our software, certificates and configuration file to the device with Ansible[7].
- Test whether device connects to MQTT.
- Test whether device is able to read all the sensors and values are legitimate (to make sure we do not have a faulty sensor or connection).
- Test whether device sends the data correctly.
- Test whether software was configured correctly.

One step of this process is already automated with Ansible which loads SmartVent software and configures the device for networking, security, remote access, logging and monitoring. However all other tasks are done manually, meaning:

- A lot of time is spent on manually creating database queries and validating them.
- Inserting records to database by hand can result in errors, that can cause problems later on.
- Generating device configuration file is done manually and can result in device mal-function.
- Certificate generation is limited to system administrators only, requiring them to be available for the setup task.
- Testing can only be done by someone with IT-knowledge and access, who can verify that connections are successful.

[6] Virtual Private Network (VPN) is tool for connecting from public internet to an internal secure network.

[7] Ansible is configuration management tool for automated infrastructure setup.

4. Location Setup tool

In this chapter we take a look at the created custom solution to speed up device provisioning, named Location Setup tool. A web application is created, that is easy-to-use and gathers all the information necessary to install SmartVent services to a new location. It's secondary purpose is to update an already existing location and it's subobjects, for example areas or the air handling unit. This feature is used in cases where changes are made, for example sensors are moved to another room or a new area is created.

This tool will not be a fully automatic solution, because the user is required to enter the hardware details itself. This is a technical limitation, because ModBus protocol does not support service discovery and there are no standards regarding where devices are connected and what features they have.

In the first section application requirements are defined. Section two gives technical overview of what tools, libraries and programming languages are used for building process. Section three has detailed information and screenshots of the final result.

4.1 Application requirements

Application requirements are divided into two groups. Functional requirements are specifically about application functionality and features. Non-functional requirements describe security, user experience and usability, performance and environmental aspects of the application.

4.1.1 Application functional requirements

- The application must authenticate the user by @smartvent.ee email.
- The application must be able to gather data about the location and the building, it's areas along with all the hardware information, that can be saved in our current database. Most information points were brought out in the introduction of this thesis.
- The application should be capable of adding a single location at a time.
- The application should be capable of adding multiple areas to a location at a time.
- The application should be capable of adding multiple sensors to an area at a time.
- The application should be capable of adding multiple actuators to an area at a time.
- The application must not allow adding sensors or actuators if no areas are defined.

- The application must conform to different types of devices that are used for sensors and actuators.
- The application must be capable of identifying possibly invalid values (e.g. word in a number field) and notifying the user about these errors.
- The application should use paginated views in order to comfortably move step-by-step through the process.
- The application must keep the filled form data entered on other pages when switching between them.
- The application must show all the inserted data to the user at the end of the process for final verification.
- The application must generate a client certificate and key for authenticating the location's controller's MQTT connection.
- The application must generate a client certificate and key for the controller's VPN connection. This must not be same as MQTT connection pair.
- The application must generate a key-value based device configuration file.
- The application must automatically send these files above in a single zip container to the user.
- The application should have the capability to edit/change existing location and it's subitems' configuration.

4.1.2 Application non-functional requirements

- The application must be easy to understand to the user. All text fields must have clear description what kind of information must be inserted there and clear indication whether the field is required.
- The application must scale visually on different devices, from mobile devices to laptops. All the page content must be visible and more-or-less the same layout on a mobile device as it would be on a desktop computer.
- The application must response to user actions within 1 second. Only exception is location saving, which can take longer due to high amount of tasks.
- The application must have access to SmartVent Root CA certificate and private key, however the key must not be directly supplied with the application.

- The key password must be passed to the application separately from the application deployment, e.g. application is set up in a Docker[8] container, however the password and key are supplied securely via environment variables.
- Application must have a secure (TLS 1.2) connection to our PostgreSQL[9] database.
- Traffic between the user and the application must be transported over HTTPS using TLS 1.2 according to NIST[10] guidelines.
- Application must use prepared statements for database queries to prevent SQL injection attacks.
- Application must use CSRF tokens for POST requests to prevent cross-site scripting attacks (XSS).

4.2 Technical overview

It was decided to build a web application in order to be easily accessible and requiring less technical knowledge as opposed to using scripting, for example in Bash shell. This means that all the fields on the page are visible in UI, making it very clear what goes where. In addition to that it does not require additional tools or applications on the user's computer, except for a web browser which is installed on most, if not all, devices. Going the scripting route would have meant that the user first needs to download all our dependencies, which may become a problem on different operating systems (some packages may not be available for Windows, different package managers for Ubuntu/CentOS/macOS etc).

Web application is divided into two parts, back-end and front-end. Back-end is responsible for handling data between the user and database, whereas front-end manages the presentation layer.

[8] Docker is tool for operating-system-level virtualization also known as containerization.

[9] PostgreSQL is open-source relational database management system.

[10] National Institute of Standards and Technology publishes recommended security guidelines for enterprises to follow, taking into account current threats.

4.2.1 Back-end

The web application back-end side, which code is directly on our server, is written in TypeScript, which gets compiled and executed. The decision to go with TypeScript came down mainly for these reasons:

- We wanted to choose a language, that would be easy to understand and maintain by any developer of our team. TypeScript is very similar to JavaScript, which is also used in web frontend development.
- TypeScript is statically typed[11], which helps to prevent some errors during the coding process, unlike plain JavaScript. This makes sure, that correct value types are used, e.g. if a number is expected, then the value can not be a string.
- Node.js applications can be easily deployed to cloud-based runtimes such as Amazon Web Services Lambda and Google App Engine.

4.2.2 Front-end

Front-end part of the application is done using standard HTML, CSS and JavaScript. In addition, React.js library will be used for easier page building and Bootstrap[12] Material Design[13] for the visuals. Content will be served to the user by the same web server that runs back-end code. It is possible to bring it away into a separate environment, however at our current scale, this will not bring any noticeable difference and would only complicate the project.

Front-end also uses browser's local storage feature for storing the data from start to end of the process. This allows collecting data in one place before sending it to the back-end. This also allows the possibility of keeping the data between page switching.

4.3 Scope

The main intention is to build a tool for quick device provisioning and location setup process. Modifying the location of an existing controller was also initially planned, because it

---

[11] Statically typed means checking and enforcing type values for variables during compiling as opposed to runtime.

[12] Bootstrap is free and widely used framework for faster front-end development.

[13] Material Design is design language developed by Google for improved user experience.

is needed for cases where hardware is changed, however due to time and technical constraints, this has been moved to future improvements.

VPN certificate creation has been moved out of the scope for this thesis. This is due to the fact that VPN currently used is a worker's personal. Once a switch is made to company's VPN, this feature will be finalized. This does not affect MQTT client certificate generation.

4.4 Main functionality

First, the application is loaded only if user is successfully authenticated using @smartvent.ee email, which is done automatically on every page load (see Figure 2). If the user is not authenticated, he/she will be redirected to Google account sign-in page. An integration was made with Google G-Suite[14] using Single Sign-on[15] to handle this user authentication against Google servers. SmartVent is currently a small team and does not have any team members, who should be restricted, therefore it was decided to allow every person with the correct email to access the page.

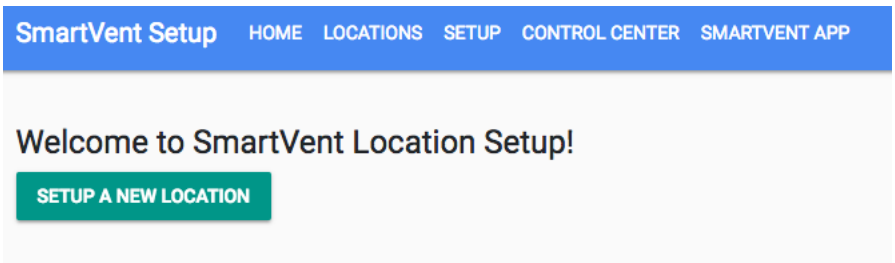

Figure 2. Front page of the application after successful user authentication.

Location setup part of the app is divided into pages: location, air handling unit, areas, sensors, actuators. After configuring the devices on a page, the data is saved to user's browser's local storage for persistence. This allows switching between pages without losing any previously entered information and also allows us to send the data to the backend in one request, simplifying the backend processing.

[14] Google G-Suite is collection of productivity and collaborations tools and software targeted for businesses, that also supports user management.

[15] Single Sign-on is a feature allowing to sign in to a service using an existing account in another service, e.g. Google account.

First step in the setup is providing information about the location, example can be seen on figure 3. This includes it's name, the account it belongs to, geographical coordinates (optional) and extras. Extras are links to external resources (URLs) related to the location.

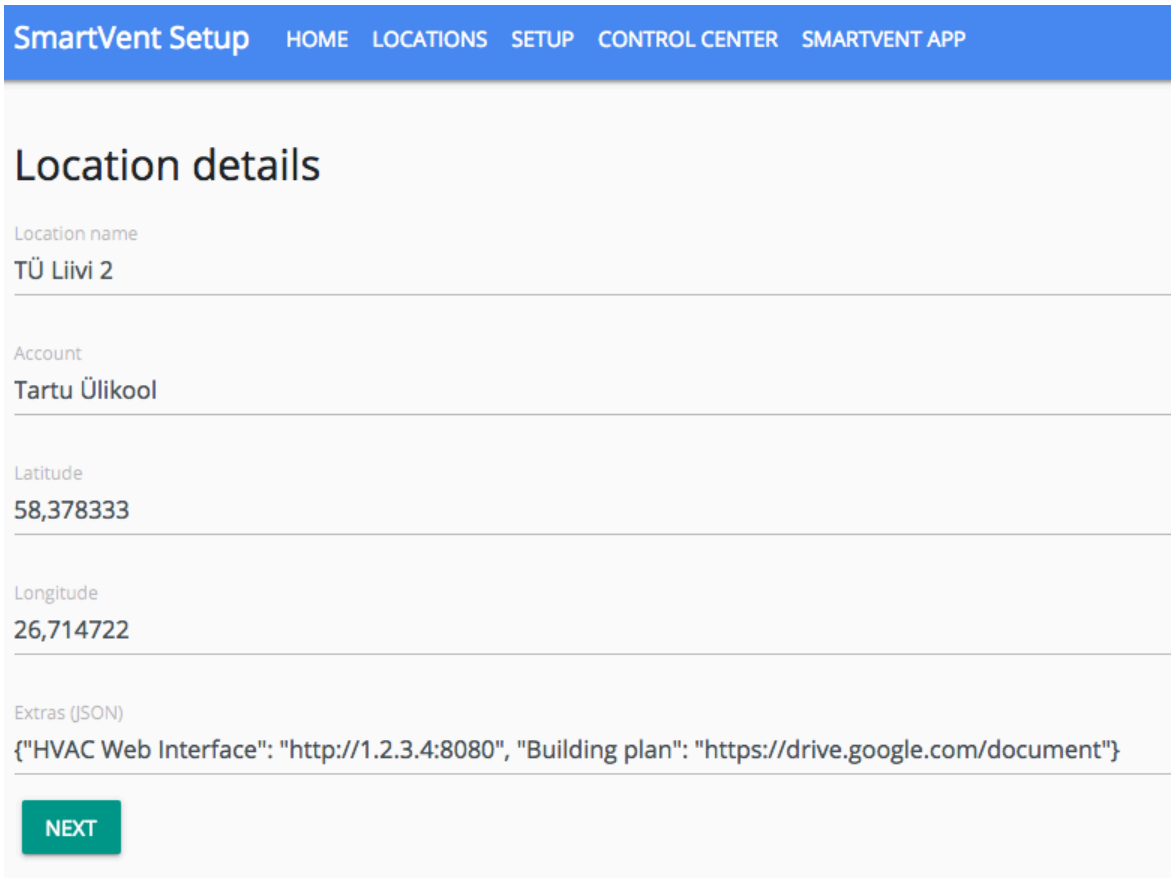

Figure 3. Location setup page with example data.

On the next page we tell what air handling unit is in the building and it's specifications (see Figure 4). It's model can be chosen from list of models, that are used in existing locations, or a new one can be written. It is possible to skip this step, if the building does not have or it's details are not finalized. In the latter case, the air handling unit can be added directly through the database or by modifying existing configuration, but this feature will be included in the future.

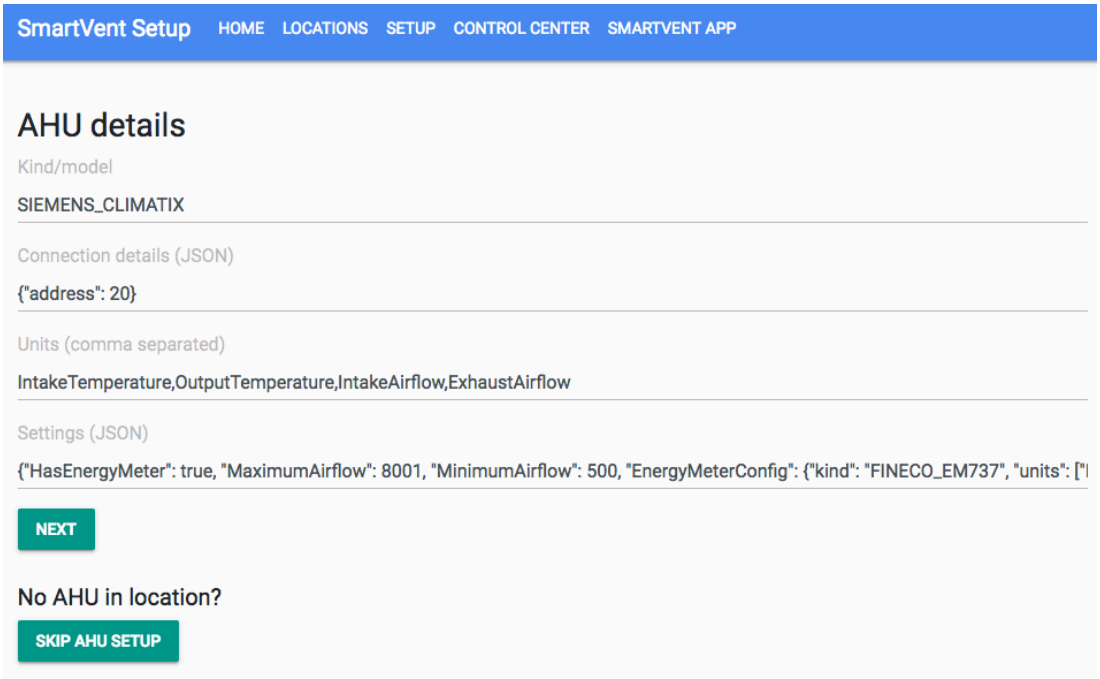

Figure 4. Air handling unit setup page with example data.

The third part of the setup is configuring the areas (see Figure 5). There is a button for dynamically adding new area, which displays a container with name and priority time fields. Both are required, but can be changed later within our overview application.

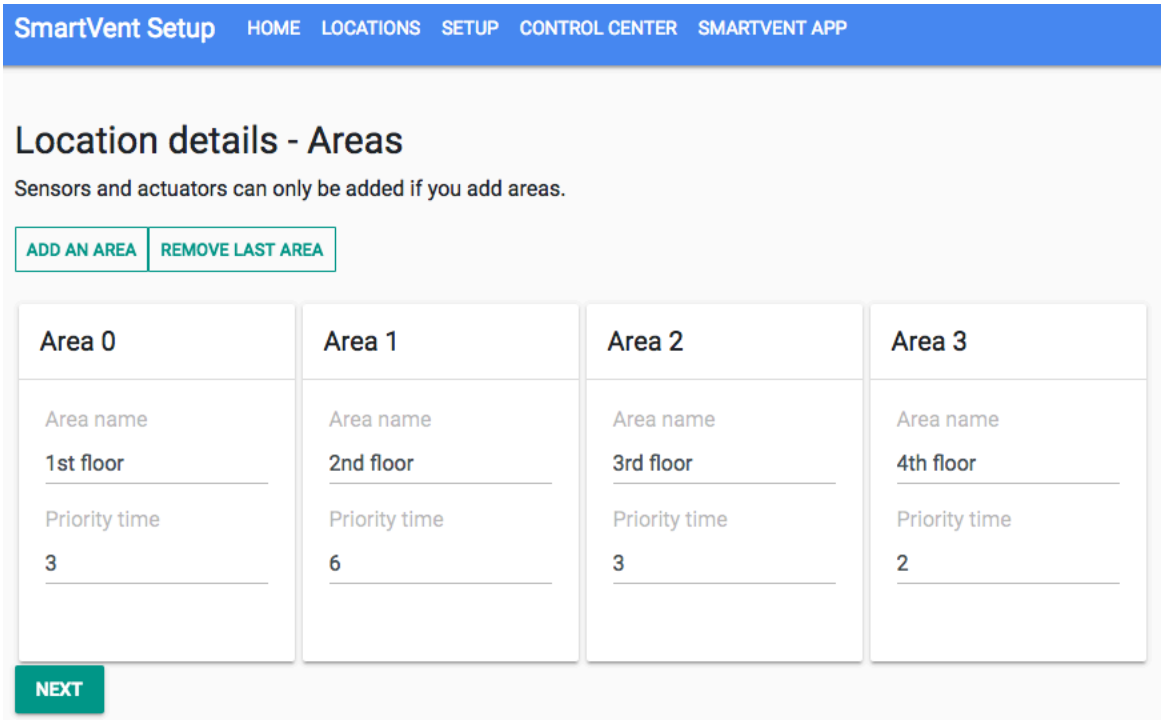

Figure 5. Areas setup page with example data.

Areas are prerequisite to adding sensors and actuators, meaning if there are no areas configured, there can not be any sensors or actuators added. If that is the case, user is sent to the verification page immediately.

If areas do exist, then next step is setting up the sensors (see Figure 6). Just like with areas, there is a button for dynamically adding new containers. Sensors belong to an area. Each sensor model can have different sources/registries for reading air quality values, so we also have to record the sensor identification data. Sensor model can be chosen from list of already used sensor types or user can write a new one.

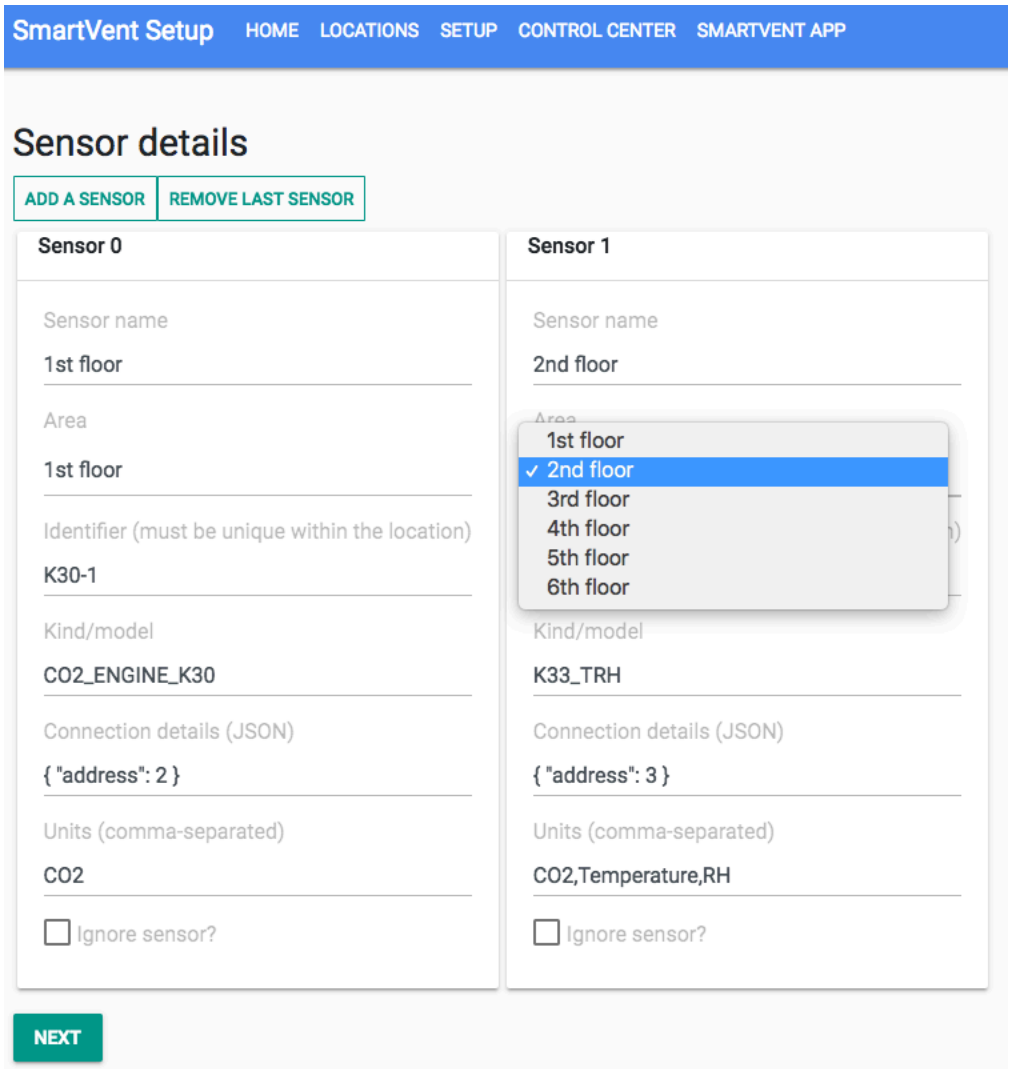

Figure 6. Sensor setup page with example data.

Then we have actuators setup (see Figure 7). Most of the time these are just controlling Variable Air Volume valves within ventilation pipes. These have very specific values regarding how much air can be moved at minimum and maximum and are determined by

ventilation hardware installation team and handed over. User can then just enter these values into a field. Just like sensors, actuators belong to an area and can have a pre-existing model or a new one provided.

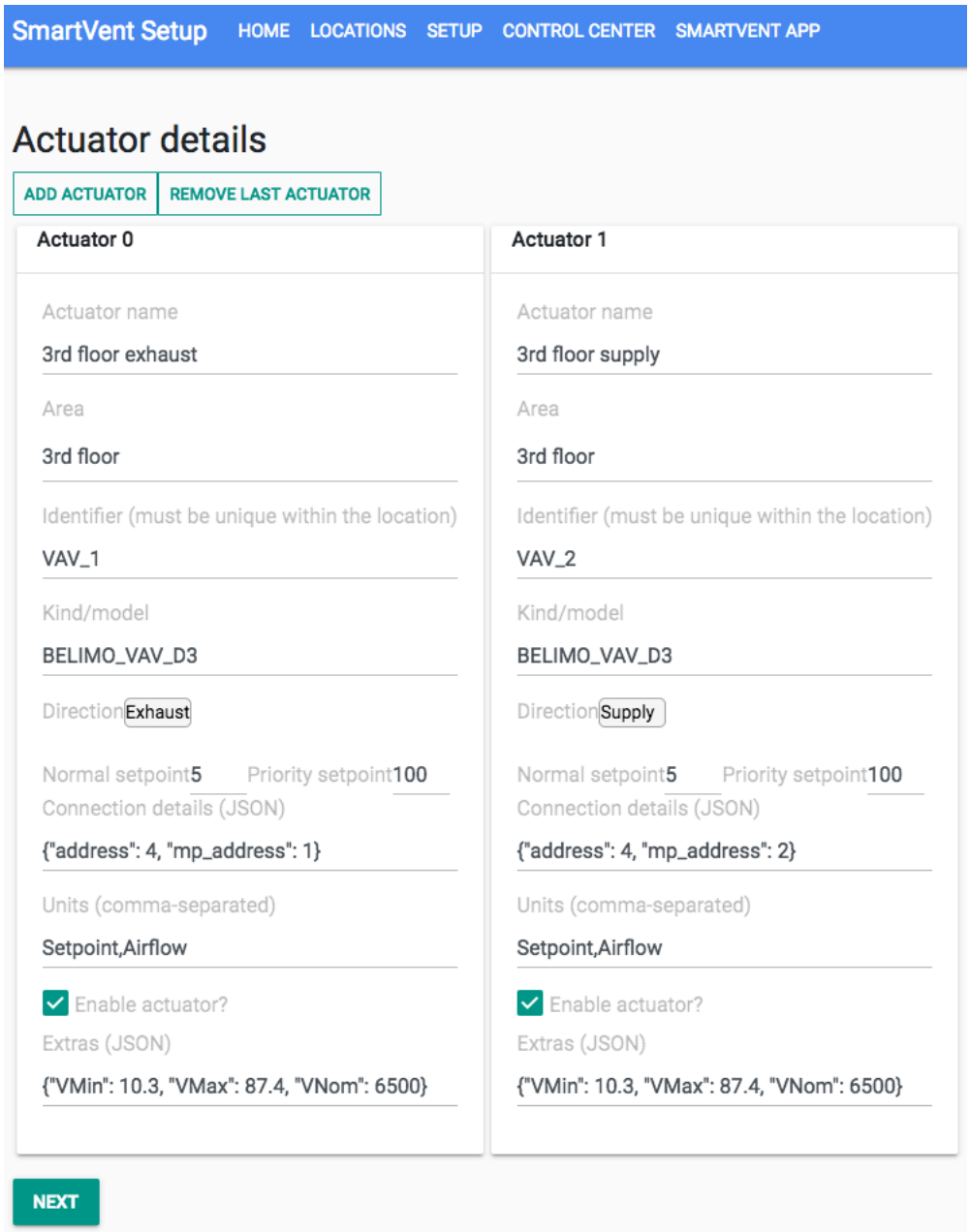

Figure 7. Actuator setup page with example data.

In some steps, data is provided in JSON[16] form. This is due to technical limitations of React and web forms, where it is rather difficult to get data in standard JSON form, which can be easily read by SmartVent controller or the main web application. An attempt was made to use key-value based fields, however the data format was not suitable for existing services. After a brief discussion with the team, it was decided to keep standard JSON for now and work on a better solution in the future.

The final step is data verification (see Figures 8 to 9). The user is displayed all the information that he has entered and it is his responsibility to make sure that all the info is correct. There is an option to go back and fix any mistakes.

[16] JavaScript Object Notation is a lightweight human-readable data object consisting of key-value pairs.

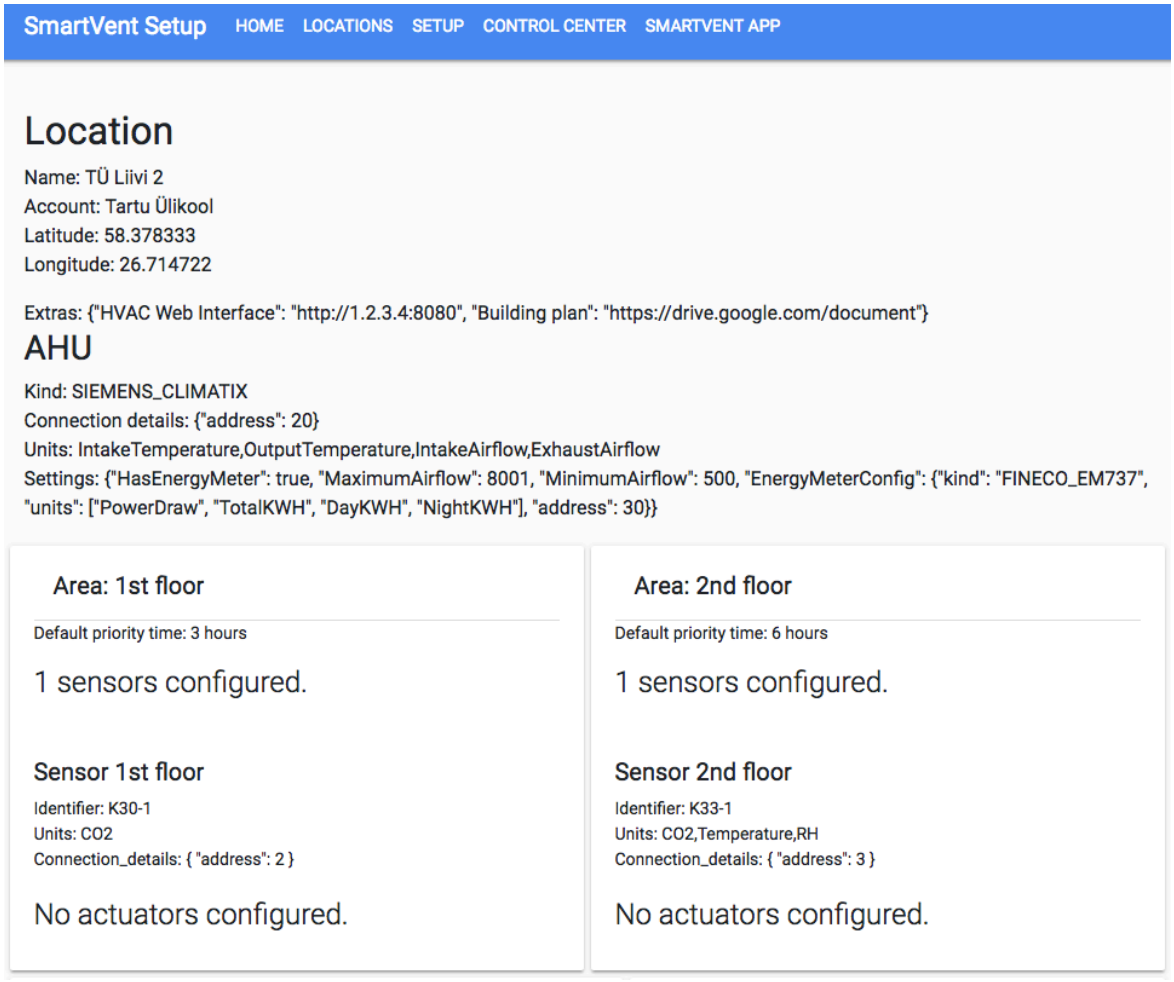

Figure 8. First part of the verification page with same example data as in previous figures. Page is large in size and is therefore split into parts.

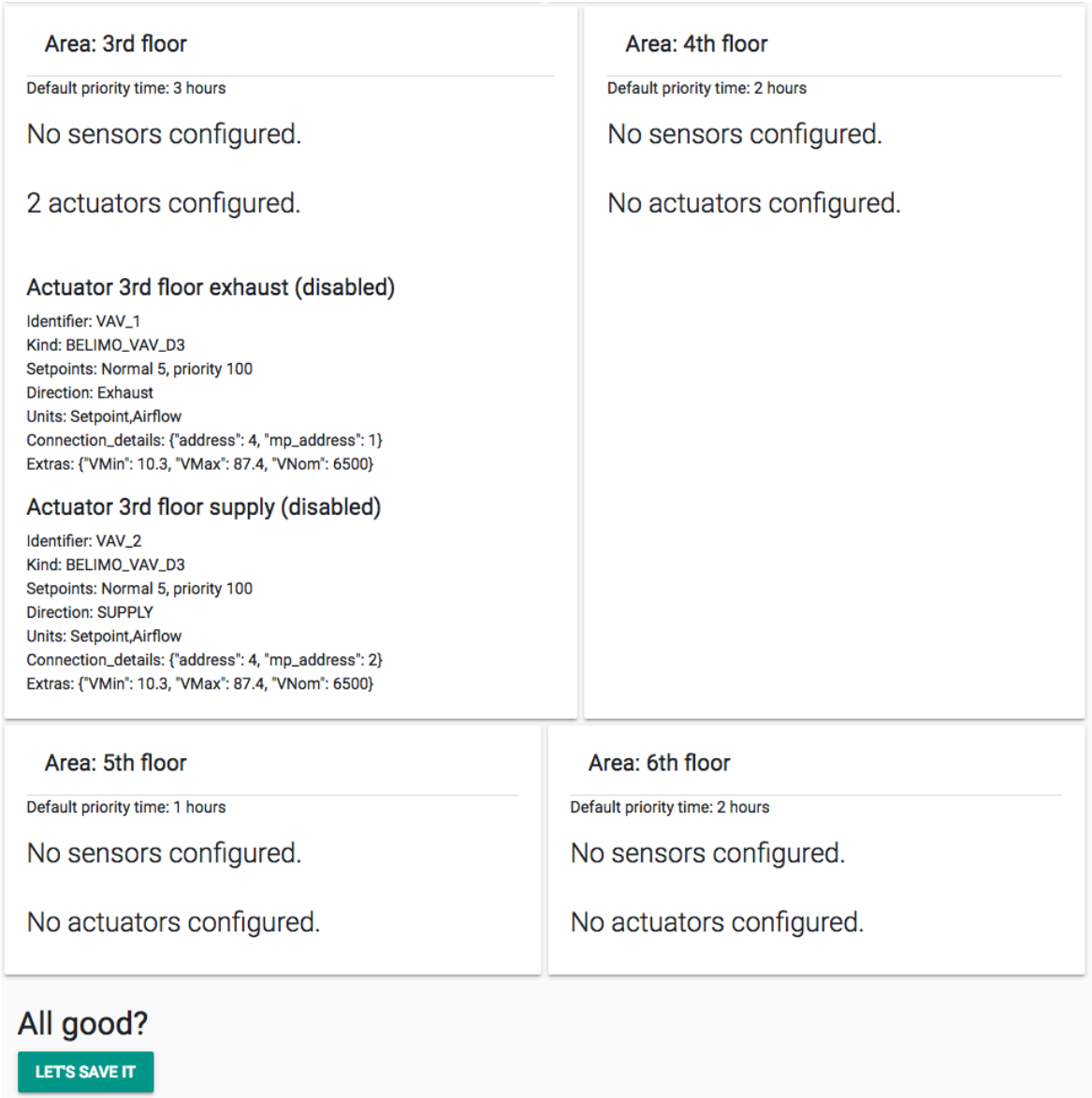

Figure 9. Second part of the verification page.

If everything is correct, then user can click the save button and the backend starts doing the following tasks:

- First task is to save all the data to our PostgreSQL database into respective tables. For this we are using SQL prepared statements, where the query is just filled with variables and then sent to the database for processing. Should any one query fail, an error is displayed to the user and the transaction will be rolled back.
- Secondly the app will generate the device configuration file (refer to Figure 10 for an example).

- Thirdly, the backend will use OpenSSL[17] and EasyRSA[18] to generate certificates and private keys for the controller's secure communication to our infrastructure.
- Finally, certificates, keys and device configuration is packed into a single zip-archive.

Figure 10. Device configuration file. Critical information has been hidden.

If all the tasks above are completed successfully, user is shown a message and a button is given to download the configuration and certificates (see Figure 11). When clicked, download of the zip file will begin. These files (see Figure 12) must then be copied to the controller by the user using the automated controller configration script in Ansible.

[17] OpenSSL is open-source software library for secure communication protocols. https://www.openssl.org/

[18] EasyRSA is a utility for managing public key infrastructure certificates. https://github.com/OpenVPN/easy-rsa

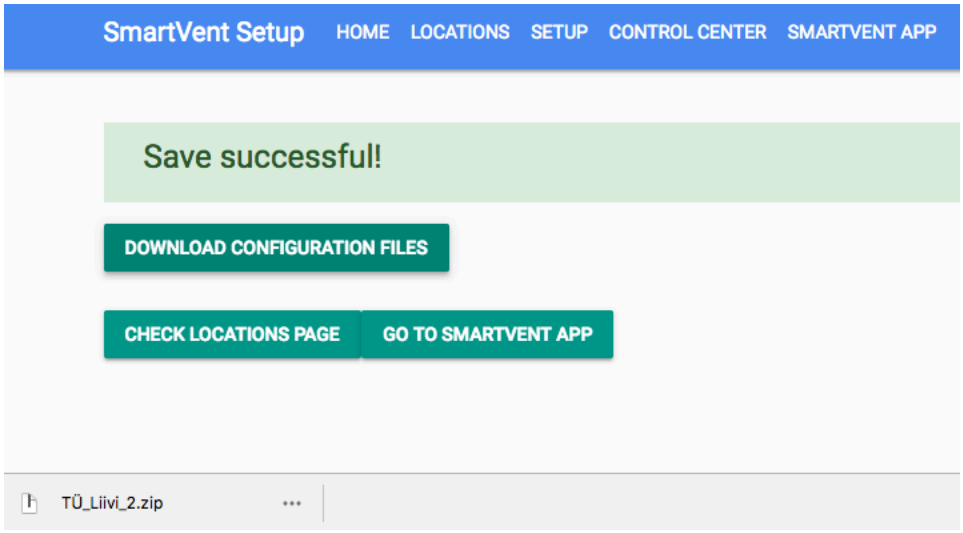

Figure 11. Successfully created location and downloaded configuration files.

```
➜ TÜ_Liivi_2 ls
total 32
-r--r--r--@ 1 janar  staff   2.1K May 14 18:32 CA.crt
-rw-r--r--@ 1 janar  staff   392B May 14 18:39 TÜ_Liivi_2.json
-rw-r--r--@ 1 janar  staff   1.7K May 14 18:39 certificate.crt
-rw-r--r--@ 1 janar  staff   1.6K May 14 18:39 certificate.key
➜ TÜ_Liivi_2 openssl x509 -noout -text -in certificate.crt
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 4107 (0x100b)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=EE, ST=Tartu, O=TarkVent O\xC3\x9C, OU=TarkVent O\xC3\x9C Certificate Authority, CN=TarkVent O\xC3\x9C Root CA
        Validity
```

Figure 12. Successfully downloaded files and valid certificate by SmartVent Root CA.

5. Analysis of the Cumulocity platform

Before building the custom solution, an idea was suggested to use an existing IoT platform as the base for integrating controller to the cloud. Cumulocity platform is advertised as very simple tool for integrating IoT devices to the cloud using 'plug and play' approach and they are working on making this process as seamless as possible. As Cumulocity is very widely used and has support for our devices, research was made which option would be more convenient and useful for the startup, whether building our own solution or using a pre-existing one.

An account was acquired for Cumulocity platform with help from University of Tartu Mobile & Cloud Lab to look deeper and test out these advertised features. We investigated different device provisioning methods and analysed each step.

First major advantage to Cumulocity is the support for wide variety of IoT devices. This includes Raspberry Pi-based controllers, that SmartVent uses. They have created specialized software for running on the devices called agents. These agents automatically collect device identification and feature data and send it to the cloud, requiring no input from the user. This also means, that the agent needs to be constantly running on the controller to maintain periodic connection with the cloud.

Cumulocity has built-in configuration management tool, which allows for seamless configuration and software changes without the need to connect to the device directly. The agent makes sure that the device has the latest configuration.

Finally, Cumulocity is hosted on their own infrastructure. This means that we do not have to worry about maintaining servers or network and eases our workload. However this also means that in case of connectivity issues or downtime, we are unable to carry out fixes and are dependant on their platform. Compared to using our own cloud, downtime would mean that we have some control and can set up a backup environment which can be connected to.

Moving on to the disadvantages, switching to Cumulocity would mean lock-in to their platform. If in the future we would have to stop using Cumulocity for some reason, we would have to create new solution for device provisioning. SmartVent is trying to avoid vendor lock-in in order to find the best possible way to run the services. Having a custom solution would allow customization as time progresses.

In terms of management, Cumulocity does not offer remote management tools directly, but they do have options to passthrough VNC[19] connection if the device supports it, for example. This means that we would still need a VPN to remotely connect to the controller, should SmartVent software or Cumulocity agent stop working.

Finally, Cumulocity is a commercial platform and they charge a fee depending on the amount of devices. This is not very cost-efficient for SmartVent at this stage. Secondly, most other tools have been custom built by the team to fit the needs and having another one made with no financial obligations would be much safer route to take.

[19] Virtual Network Computing is a graphical desktop sharing system over the network, similar to Windows Remote Desktop.

6. Verification

After having built the custom solution, we must make sure that all requirements are fulfilled and application has necessary features. In addition to that, application was tested in a very basic real-life scenario and we have gathered initial feedback.

Built-in unit and integration tests were not implemented at this moment. This part was postponed in order to achieve a minimum viable product and are going to be added in the future.

Authentication with Google G-Suite and @smartvent.ee email was successfully implemented. Users are forbidden from using the application if they are not authenticated with correct email.

Component adding and database queries have been successfully implemented and they create database rows as required. The process is quick and will make a clean rollback should an error occur. The data is sanitized using SQL prepared statements, which avoid any SQL injection attacks.

Two new locations have been set up using this tool. Unfortunately these were very basic locations, consisting of only air handling units. The feedback was positive, however there were some errors with some data entries. At first we attempted to use two fields instead of raw JSON, however the data could not have been stored in a suitable form to be saved to the database. Temporary workarounds were implemented in the backend to make sure that data is in correct format. Due to complexity of the data structure, a cleaner solution will be decided down the road.

The user interface is not currently in the best shape. For example, the margins are not consistent in places and locations page does not show all the data in readable way. This is currently acceptable as these are not high priority at this moment, functionality and usability are more important. This is an internal tool for the team only and will not be provided to customers.

In addition to that, switching between pages maintains data in forms only on location and air handling unit pages. Areas, sensors and actuators setup pages use a dynamic page building method, which does not put data back to the fields.

Certificates for MQTT and device configuration are created correctly and have been tested as well. VPN certificate generation is postponed due to currently using a worker's personal VPN. Once a switch is made to company VPN, then this feature is going to be implemented.

We have tested the application on few different devices as well as operating systems to test user interface scaling and user experience. On Apple Safari browser, we noticed that HTML datalist elements are not supported on that browser. Datalists are used for selecting a value from existing values or inserting own one. This rendered some fields broken, for example choosing account in location. A workaround for this is to use another web browser, for example Google Chrome.

Changing the location of an existing controller is postponed due to technical issues regarding building user interface. This turned out to be much more complicated to be done in a convenient matter and needs more thorough initial planning.

Security configuration has been checked and meets the requirements. TLS 1.2 connection is enforced by SmartVent infrastructure to connect to the application and from application to the database. CSRF tokens are required by web server for all POST requests and request is denied, if it is missing.

7. Conclusion

Before the creation of this Location Setup tool, the setup process was done manually and consisted of two main parts:

1) configuring the controller and devices in the database and on the device itself
2) loading correct software to the controller.

This process used to take approximately 5-6 hours per controller for a rather small building (approximately 4 areas, 4 sensors and 4 actuators). About half of this time is spent on setting correct configuration, loading the software and testing it. The most time-consuming part is testing, verifying, and fixing if necessary, because we have to check every connected component and make sure that the value is real.

After an in-depth comparison between custom solution and Cumulocity platform, a decision was made to not use Cumulocity for IoT controller provisioning workflow. Vendor lock-in is a serious obstacle as SmartVent is attempting to avoid building their infrastructure around one specific platform. It is also financially not reasonable for SmartVent to start using a commercial platform. Therefor a custom solution was built in web application form.

This new tool allows anyone from the team to carry out the setup process as opposed to older method, which required someone to handle the database and someone to handle the wiring. This also means, that all SQL queries are done automatically by the server and there is much smaller chance for a human error.

The Location Setup tool was tested in two new locations and we achieved a time saving of approximately 10 to 15 minutes per location. These locations were very simple and did not have much hardware, therefore the gains were small. However when we can test the setup tool on larger buildings, the number is definitely larger.

7.1 Future improvements

At this moment some future improvements are due to slight architecture changes. Due to demand and complexity in some buildings' layouts there are cases where multiple controllers are needed to handle larger sites. For such case, the system should be modified to support a single main controller and a set of sub-controllers.

As mentioned in the scope modifying the location of an existing controller was pushed forward in the roadmap due to technical limitations and time constraints. This is also a needed feature in cases where hardware is changed, however it has smaller priority at this moment.

VPN certificate generation will be added as soon as company VPN has been setup and all existing controllers are switched over to that one. Currently SmartVent is using a worker's private VPN due to historic reasons. Switching to company VPN requires generating new certificates, therefore will be finalized later on.

As the application deals with JSON-based[20] data, it is suggested to use JSON schema validations, which check the values types and other constraints, such as number minimum and maximum, in the data. This assists in data validation by not requiring complicated algorithms to run all the checks.

In addition to that, some form elements are accepting raw JSON data, which is not very convenient, however was accepted by the SmartVent team for the time being. A better approach for this needs to be figured out and then implemented to the application.

Implementing code, connectivity and device tests is already in the roadmap. Code tests make sure that every piece of code runs as required. Connectivity tests would attempt connecting to the MQTT and VPN using freshly generated certificates and make sure these are functional. Controller tests would make sure that device is correctly running our software, is connected to our cloud MQTT and VPN and is able to collect data from the air handling unit, sensors and actuators.

[20] JavaScript Object Notation is a lightweight human-readable data object consisting of key-value pairs.

8. Bibliography

Cumulocity GmbH. (n.d.). *Concepts guide - Interfacing devices*. Retrieved May 2018, from Cumulocity: http://cumulocity.com/guides/concepts/interfacing-devices/

Cumulocity GmbH. (n.d.). *Features*. Retrieved May 2018, from Cumulocity: https://www.cumulocity.com/features/

Cumulocity GmbH. (n.d.). *Web SDK for plugins*. Retrieved May 2018, from Cumulocity Guides: http://cumulocity.com/guides/web/introduction/

Diaz, M., Martin, C., & Rubio, B. (2016, May). State-of-the-art, challenges, and open issues in the integration of Internet of things and cloud computing. *Journal of Network and Computer Applications, 67*, 99-117.

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013, September). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future generation computer systems, 29*(7), 1645-1660.

Heidi, E. (2016, March 24). *DigitalOcean*. Retrieved May 2018, from An Introduction to Configuration Management: https://www.digitalocean.com/community/tutorials/an-introduction-to-configuration-management

Kaldvee, M.-L. (2017). *Cumulocity platvorm.* Bachelor's thesis, University of Tartu, Institute of Computer Science, Tartu.

Scully, P. (2016, January 26). *5 Things To Know About The IoT Platform Ecosystem*. Retrieved May 2018, from IoT Analytics: https://iot-analytics.com/5-things-know-about-iot-platform/

Stergiou, C., Psannis, K. E., Kim, B.-G., & Gupta, B. (2018, January). Secure integration of IoT and Cloud Computing. *Future Generation Computer Systems, 78*, 964-975.

Wortmann, F., & Flüchter, K. (2015). Internet of things. *Business & Information Systems Engineering, 57*(3), 221-224.

Xu, L., He, W., & Li, S. (2014, January 16). Internet of Things in Industries: A Survey. *10*(4), 2233-2243.

Ying-cong, Z., & Jing, Y. (2013). A Study on the Fire IOT Development Strategy. *Procedia Engineering, 52*, 314-319.

Appendix

I. Application snapshot

A snapshot of the application's code in a zip-archive has been supplied with this thesis. The archive contains a README.md file with instructions on how to run the application. A test CA has been included for certificate generation.

Running the full application locally requires a Postgres instance with specific database structure and necessary environment variables. In addition to that the application uses Font Awesome Pro for visual elements, which is a commercial product and requires a key. Due to limitations imposed by SmartVent confidentiality, we are unable to provide this database structure at this moment. We can provide a Docker image, that has necessary modules installed to run the project. If you would like to get access to this, please contact over email janar@smartvent.ee.

As an alternative, a demo environment has been setup at https://setup.test.smartvent.app. For testing purposes, the @smartvent.ee email requirement has been removed. Production environment is available at https://setup.smartvent.app.

II. License

Non-exclusive licence to reproduce thesis and make thesis public

I, **Janar Juusu**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:
   1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and
   1.2. make available to the public via the university's web environment, including via the DSpace digital archives, as of **14.05.2020** until expiry of the term of validity of the copyright,

Web interface for accelerating IoT device provisioning and configuration at SmartVent,

supervised by **Pelle Jakovits**,

2. I am aware of the fact that the author retains these rights.
3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, **14.05.2018**