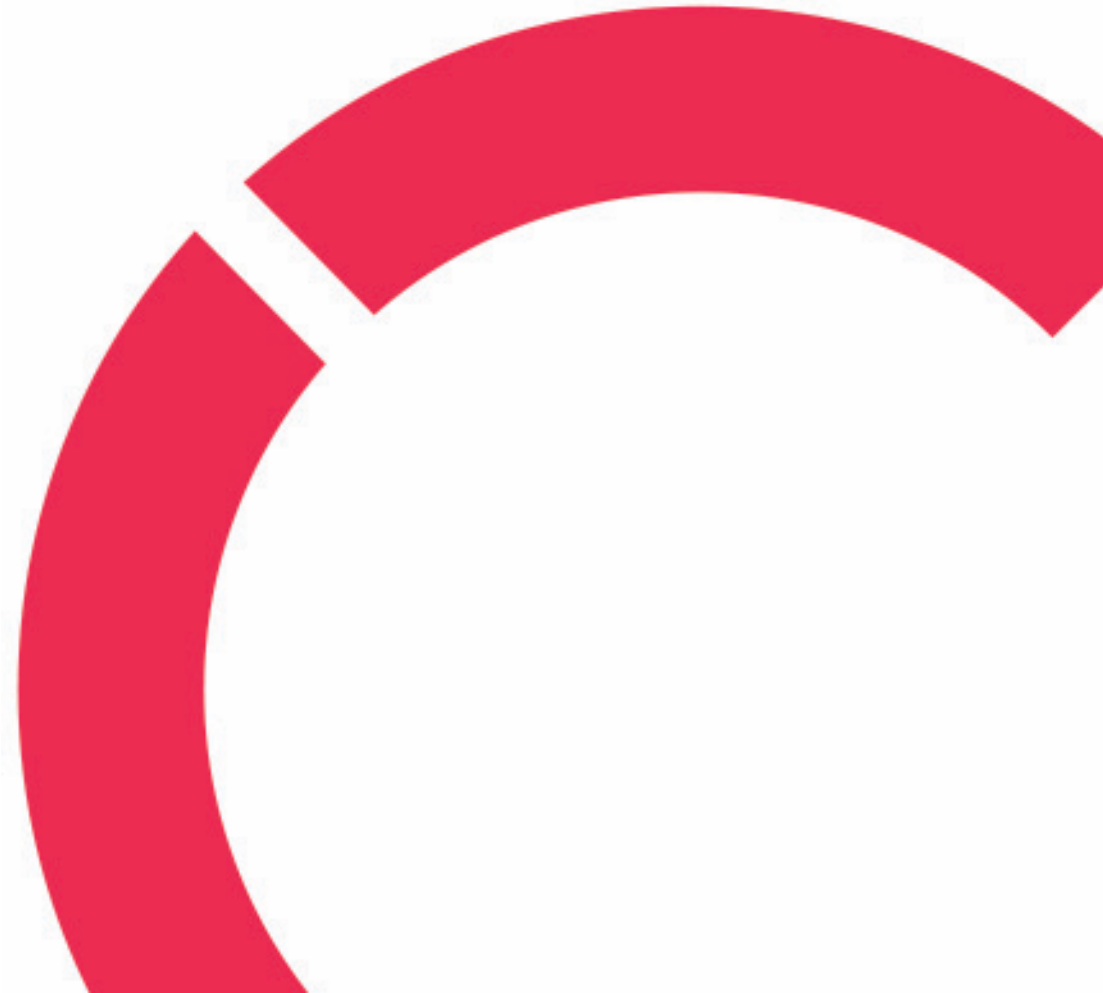
**Carl-Oscar Ahlsved**

# USB LICENSE DEVICE SHARING OVER NETWORKS

## Development of a VirtualHere and Raspberry Pi based system

**Thesis**
**CENTRIA UNIVERSITY OF APPLIED SCIENCES**
**Bachelor of Engineering, Information Technology**
**May 2023**



centria
University of Applied Sciences

**ABSTRACT**


| Centria University of Applied Sciences | Date<br>May 2023 | Author<br>Carl-Oscar Ahlsved |
|---|---|---|
| **Degree programme**<br>Bachelor of Engineering, Information Technology | | |
| **Name of thesis**<br>USB LICENSE DEVICE SHARING OVER NETWORKS. Development of a VirtualHere and Raspberry Pi based system | | |
| **Centria supervisor**<br>Jari Isohanni | | **Pages**<br>41 + 4 |
| **Instructor representing commissioning institution or company**<br>Mikael Brodin | | |

The aim of this thesis was improving an existing VirtualHere-based system running on Raspberry Pi used for sharing USB license devices over networks in Swedish audio post-production company Ljud-Bang. The original system had several shortcomings, including an inability to detect disconnected USB devices, hard-coded settings in the code, and limited user interaction possibilities. While all issues were addressed, the most significant were detection and automatic power cycling of disconnected USB devices as well as providing a means of user interaction.

First, the original system and proposed system is presented. Then background information about USB, followed by a description of the key software components used. Furthermore, considerations related to selecting USB hubs with per-port power switching used for power cycling of devices are addressed. Lastly, the developed system is presented with additional details regarding the enclosure used for the completed system.

The original system was developed using BASH scripts, which were replaced with Python and hard-coded settings were separated into JSON files. The system now incorporates the ability to detect disconnected devices, automatically power cycle them, log the event, and send email alerts to an administrator. The power cycling mechanism utilizes USB hub per-port power switching, replacing the mechanical relays used in the old system. For user interaction, a web interface was developed using Node.js and React, enabling editing of USB device configurations as well as providing means for system reboot and shutdown. Finally, the components were housed in a suitable enclosure for practical deployment.

While the system meets initial requirements, further field testing is still needed. In addition, some features are still under development. Log viewing through the web interface and adding a user database for login functionality are examples of areas requiring further improvement. Other development suggestions include enhancing device logging for better insights into device usage.



**Key words**
Raspberry Pi, Remote USB, USB, USB over IP, VirtualHere

## CONCEPT DEFINITIONS

**API**

(Application Programming Interface) is a set of rules that allows different software applications to communicate and interact with each other.

**BASH**

(Bourne-Again SHell) is a shell and scripting language used in Linux and other Unix-based operating systems.

**DESIGN SYSTEM**

The use of reusable components and guidelines to ensure consistency in visual language and user experience.

**ENDPOINT**

In USB, a communication channel between a USB device and the host computer where each endpoint is associated with a specific function or data stream.

**DOM**

(The Document Object Model) represents the objects that comprise the structure and content of a web document.

**GPIO**

(General Purpose Input/Output) refers to programmable pins on microcontrollers, such as Raspberry Pis, allowing interfacing with electronic components enabling communication and control.

**iLok**

Hardware-based digital rights management system using USB devices to securely store and manage software licenses commonly used in the audio production industry.

**JSON**

(JavaScript Object Notation) is a lightweight, text-based data format used for transmitting structured data in web applications with a human-readable syntax.

**KEY-VALUE PAIRS**

Data structures that link unique keys with corresponding values, allowing for efficient storage and retrieval.

**NAMED PIPE**

A file-like interface for communicating and sharing data through the file system.

**RPi**

Raspberry Pi

**SoC**

System On Chip

**SSH**

(Secure Shell) is a network protocol for secure remote access and communication between computers, often used for managing servers and executing remote commands.

**SSL**

(Security Sockets Layer) is a protocol for secure communications that encrypts data transmission between parties, primarily in client-server communication.

**UHUBCTL**

Command-line utility to control power to ports of USB hubs that support per-port power switching.

**USER SPACE**

In an operating system, user-space is where applications run, separate from kernel-space where the operating system's core runs.

**VPN**

Virtual Private Network

**XML**

(Extensible Markup Language) is a text-based markup language used to encode, store, and transport structured data, characterized by its flexibility and self-descriptive nature.

**CODE**



**FIGURES**



**PICTURES**



**TABLES**

# 1 INTRODUCTION

Some devices are more critical than others. If a USB license device responsible for starting software is lost or broken, it can be a stressful and unpleasant experience, especially if travelling and relying on the device for work. Moreover, sharing these devices with others can be both frustrating and time-consuming if a device must be searched for in countless office rooms. There is also a possibility that these devices contain very expensive licenses and are susceptible to theft. These are just a few examples of the challenges of sharing devices like this in their physical form. Therefore, having them located in a secure environment and sharing them over a network could be a more desirable solution. In addition, this is an excellent way of facilitating remote work, something that has become common for many due to the Coronavirus pandemic, and something which is likely to continue in the future.

There exists a plethora of ways to remotely access USB devices over a network. This technology is widely used in many areas, such as cloud services, where a local USB device can be used remotely on a virtual machine or gaming service. Another example is USB servers that share USB devices attached to the server over a network with users elsewhere running a client on their computer. An example of this type of software is VirtualHere.

This thesis covers work to replace an already existing VirtualHere system used to share USB licence devices within Swedish company LjudBang. VirtualHere allows interfacing with its functions using call-back scripts and this is used to add functionality such as timers and power cycle of devices using relays. The current system has been in operation for over five years and has some flaws that need to be addressed. One of the main goals for the new system was improving reliability through the detection of disconnected devices and automatic power cycling when this occurs, as devices commonly disconnect from the server, and this requires manual intervention. Another important aspect was to enable a means of user interaction as well as proper labelling capabilities as there currently is no way for users to interact with the system or appropriately label devices. For all this to be possible, the code needed to be rewritten with existing hard-coded configuration being separated from the code. Additionally, another power cycling method than mechanical relays needed to be used as the new system would have more ports than the old system and using relays quickly becomes bulky and impractical.

To achieve the goals the project involved the evaluation and selection of hardware, including the selection of USB hubs with port power cycling that could replace the current use of mechanical relays. Furthermore, the project included the development of new Python scripts to interface with VirtualHere and the implementation of a web-based user interface using a Node.js backend and a React frontend for user interaction. Further, central configuration files in JSON format shared by both Python and the web interface were created to replace the hard-coded settings. The thesis covers the initial development of the system. However, field testing and further development will be done after the thesis.

The structure of the thesis is as follows. The original system, requirements and proposed system are first presented in more detail. Background information about USB is then provided, followed by a description of the key software components. Furthermore, considerations regarding the selection of USB hubs with per-port power switching are discussed. Lastly, the developed system is presented. This includes the development environment, Raspberry Pi, JSON configuration files, Python scripts, the web interface, and additional details regarding the enclosure used for the completed system. Lastly, key security and reliability concerns are discussed.

# 2 ORIGINAL SYSTEM AND REQUIREMENTS

This section provides an overview of the original system for sharing USB license devices, including its limitations such as lack of device disconnection detection and hard-coded settings. It then presents the functional and non-functional requirements for the proposed system. Key improvements include using Python scripts, USB hubs with per-port power cycling capabilities, and a user-friendly interface developed with Node.js and React. The new system also introduces centralized JSON configuration files.

## 2.1 Original system

The system discussed in this thesis is used by the Sweden-based audio post-production company Ljudbang to share USB license devices both for computers within studios and between offices, as well as for people doing remote work. The company began experimenting with remote USB solutions in 2013, and the current system, which is based on the commercial software VirtualHere, was implemented in 2016. The USB devices containing licenses consist of both regular flash memory drives and specific USB license devices from iLok. Normally, iLoks are intended to be plugged into a computer, and when moved, the power is naturally disconnected. These require power cycling between uses or they will not work for the next user. Certain iLoks within the company are in high demand and hence there is also a timer functionality developed to power cycle the device after a set time. The timer settings are tied to a specific USB port on a hub and not to the device itself.

VirtualHere is highly flexible, allowing for the addition of functionality through call-back scripts on certain events. This is how both the power cycle and timer functionality are implemented. The current system uses simple BASH scripts with all settings hard-coded into the files. The hardware used consists of Raspberry Pi (RPi) and connected USB hubs, with a selection of the devices having additional power control by relays controlled from the GPIO ports of the RPi. As VirtualHere has been in use for over five years at the company, several issues have surfaced.

### 2.2 Requirements and proposed system

One major issue with the current system is that VirtualHere does not detect if a device improperly disconnects. This has been a common occurrence, especially with USB flash drives that have been plugged in for extended periods of time. Currently, the solution is to physically unplug and reconnect the devices. Often, the same devices are causing issues and would need to be replaced, but there is no organized way of tracking this. Another major issue is the lack of easy user interaction with the system. As settings are hard-coded into the BASH scripts it is hard to adjust them, and the system cannot be shut down or rebooted easily either. Additionally, due to the hard-coded settings, it is difficult to add more devices and hubs to the system. Despite not being a technical issue, the current system lacks adequate device labelling, making it difficult to locate devices. As a result of the shortcomings mentioned above and discussions with other technical staff as well as users, the requirements in table 1 were developed.

TABLE 1. Functional and non-functional requirements

Functional requirements

| |
|---|
| The system shall have power cycling functionality for each USB port. |
| The system shall have timer functionality that trigger port power cycling if set. |
| The system shall have detection of disconnected USB devices and do automatic power cycling for reconnection of devices. |
| The system shall be able to share a minimum of 40 USB devices. |
| The system shall provide means to restart or shut it down. |
| The system shall provide means for labelling of ports and devices. |

Non-functional requirements

| |
|---|
| The system shall have an easy-to-use user interface to enable non-technical users to perform basic operations and configurations. |
| The system shall log disconnected USB devices to help track malfunctioning devices in the long run. |
| The system shall be stable and not experience regular crashes. |
| The system shall have a maximum of four hours of downtime. |
| The system shall be housed in an enclosure. |
| The system shall be compatible with Mac OS. |

For the new system the functioning components of the original system, including VirtualHere and RPi, were kept since there was no obvious reason to replace them. However, the Bash scripts responsible for the core functionalities such as power cycling and timers were considered lacking and not suitable as a basis for further development. Additionally, adding detection of disconnected devices would require considerable modifications to the system. As a replacement scripting language, Python was chosen. Automatic power cycling of disconnected devices would require power cycling of all ports, not just iLoks. Instead of using mechanical relays, which would be bulky and require much work to implement for all ports, USB hubs with per-port power cycling capabilities were selected. Node.js and React were chosen for the user interface, partly to maximize learning effort and take advantage of prior knowledge. The user interface would provide ways to interact with the system, allow for easy configuration and facilitate system reboot and shutdown. The previously hard-coded configurations needed to be centralized to enable the Python scripts and the user interface to access the same settings and device data. While using a database was considered, for simplicity, it was ultimately decided to use JSON files instead.

# 3 UNIVERSAL SERIAL BUS

In this thesis, the focus is not on providing an in-depth understanding of USB, as it is a complex technology that have evolved over time. This section first offers a general overview, and then covers specific aspects relevant to the project as per-port power cycling required for power cycling of devices. The emphasis is on USB 2.0 as that is what is used for the project, but to understand some concepts and issues encountered, USB 3 is also covered to a certain extent. The latest versions of USB, such as USB4 and the use of the Type-C connector, will not be discussed, as they are not relevant to this project.

## 3.1 Overview

Universal Serial Bus (USB) is a widely used standard for interfacing computers and other digital devices with peripherals introduced by Intel in 1996. A USB system is described by three definitional areas in the USB 2.0 specification, interconnect, device, and host. The interconnect is the way USB devices are connected to and communicate with the host and includes the protocols as well as physical connectors and cables providing connections. Devices can be either hubs, which provide additional connection points, or devices that provide functions for the system such as flash drives or other peripherals. There is only one host in any USB system. A hub is integrated within the host to provide one or more connection points and is called the root hub. The USB interface to the host computer system is called the Host Controller and it is responsible for controlling data flow, managing power consumption, and ensuring correct device connections and disconnections. The USB Host Controller Interface (HCI) specification defines how USB host controllers communicate with USB devices and computer systems. (USB-IF 2000.)

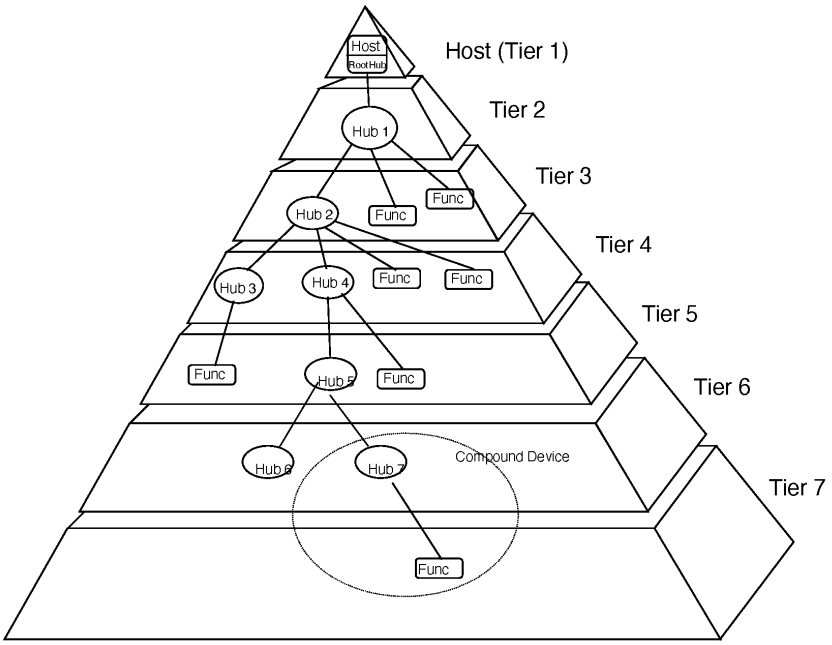



FIGURE 1. USB topology (USB-IF 2000)

The kind of physical interconnect used forms a tiered star topology, as can be seen in figure 1 where the host with the root hub is at the top and there are additional tiers below with hubs and devices. Due to timing constraints related to hub and cable transmission times a maximum of seven tiers are allowed. (USB-IF 2000.) In later specifications as USB 3.2, there are significant changes in protocols; however, the basic topology remains the same (USB-IF 2022).

### 3.2 History

The first USB specification offered two transfer speeds, initially 1,5 Mbps (low-speed) which was later increased to 12 Mbps (high-speed). As computers become more powerful and capable of processing larger amounts of data, higher data transfer rates were required. The USB 2.0 specification was introduced in 2000, providing 480 Mbps as a third transfer rate while maintaining backward compatibility. USB 2.0 also introduced more features as better power management allowing for USB devices to be powered down when not in use to reduce power consumption. With USB 2.0 a new HCI called EHCI was introduced but it coexists with a USB 1.1 host controller used for lower speeds. USB 2.0 devices are backward compatible with USB 1.1, so they can be connected to USB 1.1 computers and vice versa. (USB-IF 2022.)

The need for higher transfer rates continued, as by 2006, drive speeds commonly exceeded 100 MB/s, which was significantly higher than the 32 MB/s bandwidth offered by the 480 Mbps transfer rate available in USB 2.0. In response, USB 3.0, also called SuperSpeed, was introduced, enabling data rates up to 5 Gbps while still maintaining backward compatibility. Later, Enhanced SuperSpeed USB 3.1 and 3.2 were introduced, increasing transfer speeds even further, with USB 3.1 offering speeds up to 10 Gbps and USB 3.2 providing speeds up to 20 Gbps. (USB-IF 2022.)

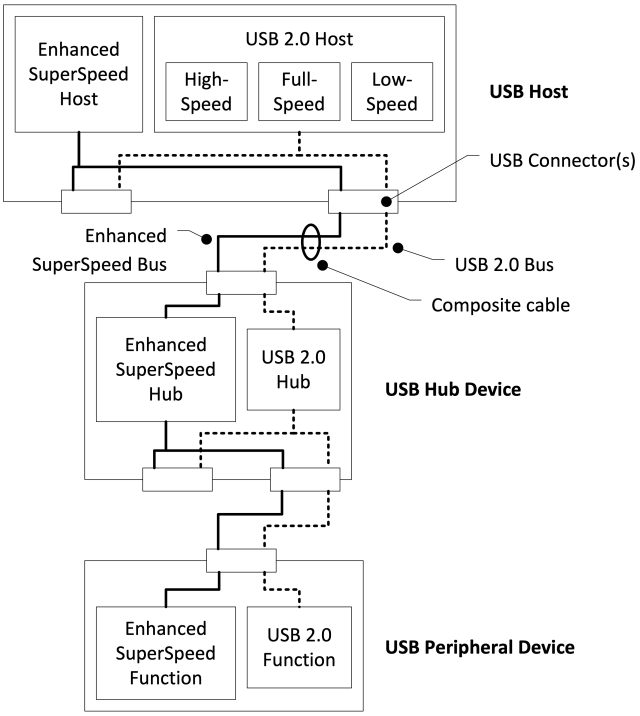



FIGURE 2. USB 3.2 dual bus system architecture (USB-IF 2022)

With USB 3.0 many technical improvements were introduced, including changes in the protocol, encoding, wiring, power management and more. Most are out of scope for this thesis, but the dual-bus architecture that enables backwards combability is worth mentioning as USB 2.0 and 3.0 co-exists side by side. This architecture provides separate paths for USB 2.0 and USB 3.0 traffic, as can be seen in figure 2 above. Data is transferred simultaneously between USB 2.0 and USB 3.0 devices via a single cable. By using separate connectors and additional wires, USB 3.0 devices are not affected by USB 2.0 traffic and both buses can be active at the same time. When connected to a USB 2.0 port, USB 3.0 devices will operate at USB 2.0 speeds. (USB-IF 2022.) USB 3.0 also introduced a new HCI controller called eXstensible Host Interface controller (xHCI) capable of also handling the older standards. The xHCI controller theoretically supports up to 255 devices with 31 endpoints each. (Intel 2019.)

### 3.3 Enumeration

When a USB hub or device is connected to a host computer, a process called enumeration is initiated to establish communication and configure the device. The process involves detecting the connection, providing power, resetting the device to have a clean starting point, assigning a unique address, and retrieving device descriptors. These are data structures containing essential information about the device and its configuration. Based on the received information, the host selects an appropriate configuration for the device and if necessary, loads device drivers. Once configured, the host computer can begin data transfers and communicate with the device using various USB transfer types and protocols. As USB allows devices to attach or detach at any time, enumeration is an ongoing process. In the enumeration process, the host communicates with the device using control transfers at what is called endpoint 0, which is a unidirectional control endpoint that is required for all USB devices and hubs. (USB-IF 2022.)

In Linux the device descriptors are also made available in a virtual filesystem. More recent kernels likely use sysfs. Here the kernel's internal data structures related to devices, drivers, and other kernel components are presented in user space through a virtual filesystem. This way, it allows users and applications to easily access this information by presenting it in a hierarchical directory structure. The sysfs filesystem contains many read-only files, but some are writable, allowing change of kernel variables. An entry is created for a USB device by the kernel under /sys/bus/usb/devices after it is successfully enumerated. The complete path forms a device path, or devpath in short. This entry contains files and directories that represent device attributes and configuration options. (Linux Kernel Organization 2023a; Linux Kernel Organization 2023b.)

### 3.4 Per-port power switching and uhubctl

USB 2.0 introduced better power management. Self-powered hubs, which do not have external power, may have optional power switches to control power to ports, while externally powered hubs require them. The control does not have to be for each individual port, however. In ganged power-switching mode, power to ports in a group is controlled simultaneously. In per-port power-switching mode, power to each port is controlled individually. (USB-IF 2000.) While it is possible to control port power by writing to the virtual sysfs filesystem as described by Stern (2014) in the Linux Kernel documentation, this is not straightforward and always reliable. Another simpler approach would be to use a third-party tool.

Uhubctl is an open-source command-line utility written in C designed to manage and control USB hubs which support per-port power switching that are compliant with USB 2.0 and USB 3.0 specifications. It is compatible with various operating systems, including Linux and macOS. The tool allows users to display information about detected hubs, including their location, device ID, and port power status. In addition, it allows control of port power. It also has a forced mode, which attempts to control individual ports on a hub even if per-port power capability is not reported by device descriptors. (MVP 2023.)

# 4 SOFTWARE

This section describes key software components of the project. It was found that the main component of the system, VirtualHere, was still competitive among other solutions. Because of its proven reliability and good support from the developer, it was decided to continue using it. Furthermore, using another software would invalidate much of the knowledge and lessons learned from the current system, making a switch even less compelling. Following VirtualHere, the technologies used for the web interface are also briefly discussed. An overview of Python follows, preceded by an overview of the RPi operating system and some Linux tools that were used.

## 4.1 VirtualHere

VirtualHere is a software solution that enables sharing of USB devices across a network. Several alternatives to VirtualHere exist on the market, but not all are suitable as the company is using Apple computers and therefore requires MacOS support. There exists, however, several competitive cross-platform products as USB Network Gate and FlexiHub by Eltima as well as products from other vendors, including USB over Network from FabulaTech and USB redirector from Incentives Pro. Additionally, hardware alternatives exist, such as Digi's AnywhereUSB line of products. (Eltima 2023; FabulaTech 2023; IncentivesPro 2023; Digi 2023.) Among these alternatives, a common denominator is that they are considerably more expensive than VirtualHere, which costs 49 USD at the time of writing, which is a fraction of what most competitors charge for similar functionality (VirtualHere 2023a).

VirtualHere works by converting USB into TCP/IP packets and then converting it back at the other end. This enables remote access to various USB devices, such as printers, scanners, webcams, and other peripherals, as though they were directly connected. VirtualHere uses a client-server architecture. The server is where the devices are connected, and the client is used on the host computer to access the devices remotely. (VirtualHere 2023b.) An overview can be seen in figure 3 below.

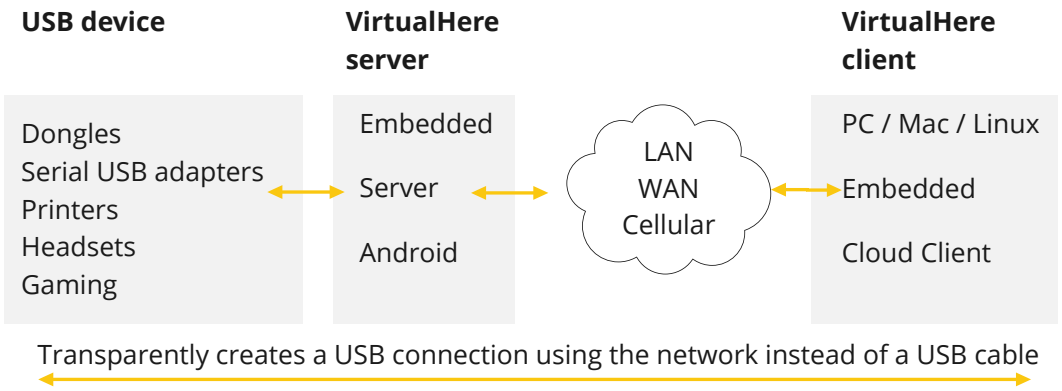



FIGURE 3. VirtualHere overview (adapted from virtualhere.com)

The VirtualHere USB Server is available for Windows, Mac OS, and Linux including Android. The Linux server is designed to be compatible with any version of the Linux kernel and with any architecture. This includes desktops, servers, Android, and various embedded systems as the RPi. All settings are stored in a text-based configuration file, and the server supports up to 122 devices and six tiers of hubs. A variety of integrations are possible through the server's ability to perform call-backs to scripts on certain events, such as when devices are enumerated on the server and used or stopped being used on the client. It uses the open Bonjour protocol for network discovery and operates on a single TCP port making firewall and remote access setup easier. It supports SSL for added security when sharing over the Internet and has a built-in system for accessing devices over the Internet, albeit at additional cost. (VirtualHere 2023c.)

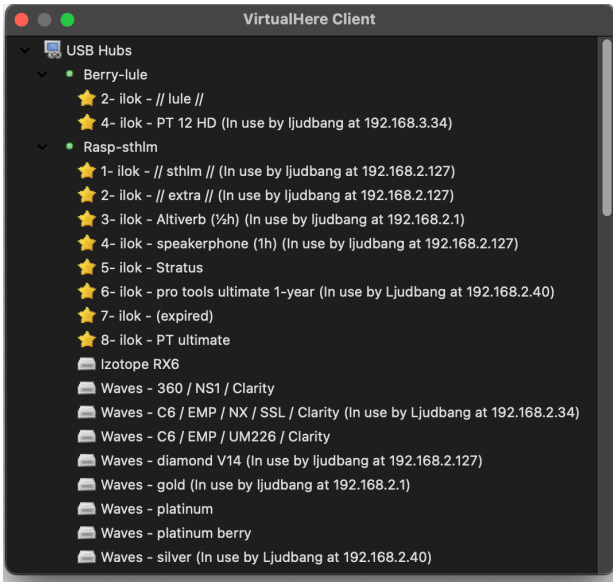



FIGURE 4. VirtualHere client window

The client is cross-platform, supporting Windows, Mac OS, and Linux. The GUI can be seen in figure 4 where a click on a device connects or disconnects it. The client can also be run as a service and controlled through an API either directly or using a named pipe. One useful command returns the client's state in XML format that also includes all connected devices. The client also supports event-based callbacks, but the number of events supported is lower than that of the server. (VirtualHere 2023d.)

### 4.2 Web interface technologies

When developing the web interface, the primary focus was on the functionality provided rather than the specific technology used. The main purpose to create a web interface was to enable user interaction with the system, offering options to edit device configuration and perform actions such as shutdown and reboot. In this project Node.js is used as a REST API that provides endpoints to the React frontend to fulfil the functionality required. For security reasons there was also need for login functionality. This part first presents a brief background on the key technologies used for the web interface as Node.js, React, and JSON. Further, additional technologies used as JWT and JSON are mentioned.

Node.js is a widely used open-source, cross-platform JavaScript runtime environment released in 2009. It offers a fast and efficient way to build JavaScript-based server-side applications. It has fast performance, non-blocking I/O, and large module ecosystem. REST APIs (Representational State Transfer Application Programming Interfaces) are architectural styles for developing web services that allow different systems to communicate and exchange data. It is based on resources, which are uniquely identifiable entities. To perform operations on resources, standard HTTP methods are used and standard HTTP response codes are given. Each request is stateless, meaning that all necessary information is included in the request. The data is often represented using JSON or XML. (Bojinov 2018.)

For preventing unauthorized use several endpoints expect a token in the header for authentication and JSON Web Token (JWT) is used for this. JWTs consist of three parts: the header, payload, and signature. The header contains information regarding the type of token and cryptographic algorithm used. The payload of a JWT contains important information about the user or entity. The signature verifies the authenticity of the token. The use of JWTs is often used to provide stateless authentication, allowing servers to verify and authorize requests without having to query a database or save session information. (Jose, M., Jones, M., & Bradley, J. 2015.) The JWT token received from the backend is also stored by

the frontend within the browser's localStorage to be retained between sessions. Localstorage is a storage of key-value pairs only accessible locally in the browser (Mozilla 2023).

The front-end is built using React, which is an open-source JavaScript library used for building user interfaces and UI components in web and mobile applications. In React, developers declare the desired structure of the user interface rather than working with the DOM. React uses a virtual representation called the Virtual DOM, which allows it to detect changes and update only the necessary parts of the UI. This enables automatic UI updates in response to changes in data or user interactions. (React 2023.) The React frontend built for the system uses Material UI (MUI) which is an open-source UI component library for React-based web and mobile application development. It is a design system based on Google's Material Design system and contains a wide range of customizable and responsive components, layout utilities, and themes that simplify the design and development process. (MUI 2023.)

JavaScript Object Notation (JSON) is used for both communication between backend and frontend as well as for configuration files. JSON represents structured data in a lightweight, human-readable format. In general, it is used to transmit data between a server and a client (web application) or within an application. Although derived from JavaScript, JSON is language-independent, meaning it can be used with any programming language. JSON data is stored as key-value pairs, where keys are strings and values are strings, numbers, booleans, objects, or arrays. For objects, curly braces are used "{}" and for arrays, square brackets are used "[]". (Ecma International 2017.)

### 4.3 Python

Initially released by Guido van Rossum in 1991, Python is a high-level interpreted programming language. It stresses simplicity and readability and uses indents for statement grouping which reduces the need for complex syntax structures. This makes it easier to read and a suitable choice for both beginners and experienced programmers. Python is general-purpose and versatile and is used for a variety of applications including web development, data science, artificial intelligence, and automation. In addition to a standard library supporting many common programming tasks, a range of external libraries is available for more specific work. Furthermore, Python has a strong and active community that contributes to its development and offers support. (Python Software Foundation 2023a.)

### 4.4 Raspberry Pi OS and utilities

The RPi in this project uses the recommended operating system provided by the RPi Foundation, known as Raspberry Pi OS. This operating system is based on Debian and tailored to the RPi (RPi Foundation 2023a). Raspbian includes numerous pre-installed tools as does most Linux distributions. A couple of common Linux tools are used. To start and keep the VirtualHere server and Node processes running, systemd is used, and for starting scripts at intervals crontab is used. Additionally, logrotate is used to manage logs.

Systemd is a system and service manager for Linux operating systems. A systemd daemon is a background process that manages system services, ensuring they start, stop, and restart as required. Several Python scripts need to be started at certain intervals and for this crontab is used which enables users to schedule tasks in crontab files. These contain a series of lines describing the tasks to be executed at what time or interval. To prevent logs from becoming excessively large, lograte is used. Logrotate is a utility designed to manage log files generated by various applications and services. Log rotation involves renaming, moving, and creating new log files to manage log sizes and make efficient use of disk space. (Negus 2015.)

To prevent unnecessary wear on the SD card, a directory in RAM is used for temporary files written by the Python scripts. Raspbian, and other Linux distributions create such folders automatically. One example is described in the Linux Foundation (2015) filesystem hierarchy standard where temporarily a directory based on the user ID is created in /run/user/ at system login. This directory serves as a memory storage location for user-specific data used at runtime. When the system restarts or the user logs out, it is automatically cleared.

# 5 HARDWARE

This section discusses several hardware choices for the project. The old system uses the RPi 2 model, which has worked reliably for many years. However, as it is old the support is about to end. As a result, the decision was made to use the newer RPi 4 version, which offers prolonged support and improved performance. The USB hubs section covers the evaluation process for hubs suitable for the project. The hubs were chosen mainly based on their per-port power control ability and their size, in order to fit enough hubs within the enclosure.

## 5.1 Raspberry Pi

In 2012, the RPi Foundation developed a credit card sized computer to provide the educational sector with an affordable platform for learning programming. The initial version of the RPi, the Model B, featured limited connectivity and only 256 MB of RAM powered by an ARM11 processor at 700MHz. Several revisions to the RPi have taken place since then, and updated models have been introduced with improved specifications and features, such as more GPIO pins, improved USB, and network support. A major strength of the RPi is the support it receives from both the Foundation and a large community of developers and enthusiasts. In addition to documentation, educational resources, and online communities the Foundation provides an official operating system, Raspberry Pi OS, which is based on Debian. (RPi Foundation 2023b.)

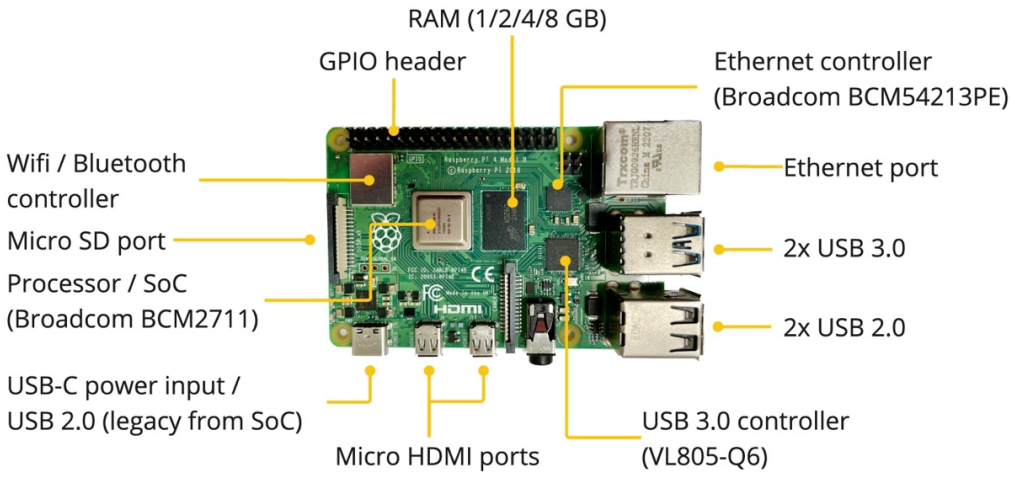



FIGURE 5. RPi 4 overview

The most recent model as of 2023, the RPi 4B, can be seen in figure 5 above. It has a quad-core ARM Cortex-A72 processor at 1.5 GHz, up to 8 gigabytes of RAM, and is the first version to have USB 3 support (RPi Foundation 2023b). The USB 3 controller is directly attached to the PCIe bus and drives the main USB ports. The old legacy controller used on previous models is found on the SoC, albeit only capable of USB 2.0, and is available for the USB-C port which can be used as both USB host and device. This controller must be manually enabled and configured as the USB-C port is primarily used for powering the PI. (Raspberry Pi Foundation 2023c.)

### 5.2 USB hubs

For the new system USB hubs with per-port power control were selected to replace the mechanical relays used in the original systems. This to be able to power cycle all USB devices connected in a practical manner. The list of USB hubs that support per-port power switching as reported by the uhubctrl project is short (MVP 2023). The hubs for this project, aside from per-port switching capabilities, needed to have dimensions to fit enough of them in a practical way inside an enclosure. Initially, a compact hub called D-Link DUB-H7 was considered as it already was in use by the company. It has seven USB 2.0 ports without any LED indicators. The data transfer is limited to 480Mbits since this is a USB 2.0 hub. The power adapter is 15W at 5V but is not required. (D-Link 2023.) However, it is an old hub and there exist several newer revisions where newer models lack per-port power cycling capabilities which is also made clear from the uhubctl page. After unsuccessful attempts to source the older model from three different sources, it was decided to explore other options. A 7-port hub from the Amazon basics line had a very compact format and was listed as compatible and therefore was tested. It features seven USB Type-A 3.1 ports with LED indicators for each port. It supports data transfer speeds of up to 5Gbps being compatible with USB 3.1, 3.0, and 2.0. The power adapter, which is rated at 36W and 12V, is required for the hub to function. There is also a 10-port version which also was evaluated that is similar but uses a 20V power supply. (Amazon 2023.) The three hubs can be seen in picture 1 below.


PICTURE 1. Amazon basics 7 & 10 port and D-Link DUB-H7 USB hubs (Amazon 2023; D-Link 2023)

Manufacturers often design larger hubs using 4-port hubs internally, which is noted by the uhubctl project (Mvp 2022). This means that a 10-port hub often contains three internally daisy-chained 4-port hubs, and a 7-port often uses two. This was the case with all hubs tested in this project. It was discovered that VirtualHere's six-tier limit could easily be exceeded if not careful. Figure 6 illustrates two scenarios involving two different hubs. In the first example, two 10-port hubs are used, and the chain of internal hubs causes some devices to end up on tier 7 and they cannot be used by VirtualHere. The other example presents a working alternative utilizing 7-port hubs where no device goes past tier 5.

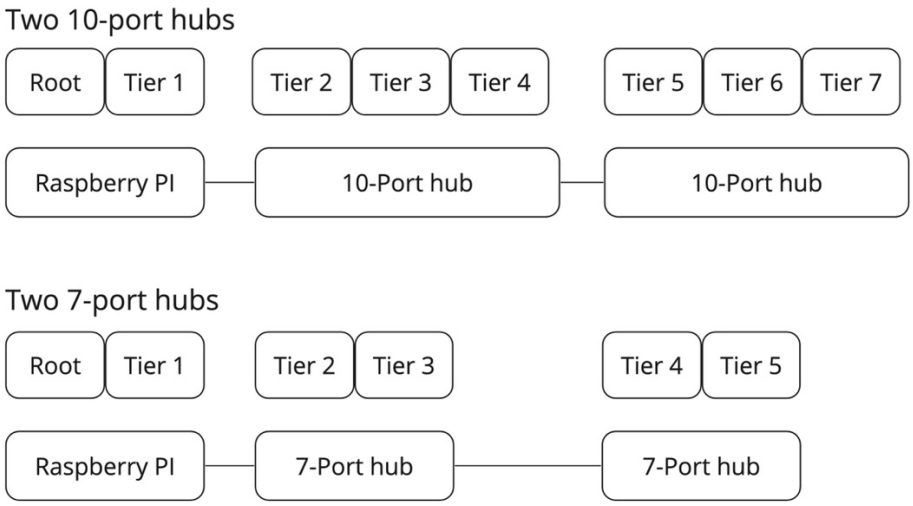
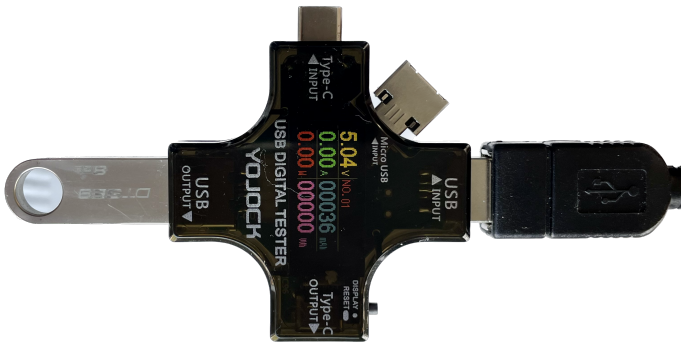



FIGURE 6. Two 10-port hubs and two 7-port hubs

Three self-powered USB 2.0 4-port hubs were additionally evaluated for this project as these consist of a single 4-port hub. These were hubs that were found to be available at the company and can be seen in picture 2 below. According to the USB 2.0 specification (USB-IF 2000), self-powered USB hubs must have power switches to control ports, although they can be managed in gangs or groups. The power switching behaviour of all hubs initially appeared to be ganged. However, when using the forced mode of uhubctl, each hub indeed featured per-port power switching capabilities. There is also the issue of limited power allowance when using self-powered hubs as there is no external power available. In USB 2.0 only 500 mA of current is supported per port (USB-IF 2000).


PICTURE 2. Tested self-powered USB 2.0 4-port hubs

To determine if the 4-port hubs could be used, preliminary testing of USB flash drive energy consumption was performed with a simple power meter as seen in picture 3 below. Device descriptors provide information about the device, including power consumption (USB-IF 2000). A difference was observed between the reported power usage in the device descriptors and the measured power usage when testing the most common types of USB devices currently in use in the original system. With the descriptor values being consistently higher than the measured values in most cases as can be seen in table 2.




PICTURE 3. YOJOCK USB digital tester

These numbers are very similar to numbers found in a previous study on power and performance characteristics of USB flash drives, where flash drives commonly used more power for writing than for reading (O'Brien, K., Salyers, D.C., Striegel, A.D., & Poellabauer, C. 2008). USB devices in the system are mostly idle and rarely see any writes. This means the measured power draw is close to what is expected in the operating environment. This would indicate that the 500 mA available to a 4-port USB 2.0 hub would be adequate for most of the USB devices in use in the original system.

TABLE 2. Measured power consumption of common USB 2.0 devices in the original system

| Device name | Size | Descriptor | Measured (read/write) |
|---|---|---|---|
| Kingston DTSE9 (flash drive) | 8 GB | 200 mA | 60 / 70 mA |
| Kingston DataTraveler G2 (flash drive) | 16 GB | 300 mA | 140 / 200 mA |
| Tranzip (flash drive) | 16GB | 300 mA | 60 / 80 mA |
| SanDisk Cruzer Blade (flash drive) | 64 GB | 224 mA | 100 / 200 mA |
| iLok | - | 60 mA | 30 / 30 mA |

In the later stages of the project more Amazon 7-port hubs were acquired and a total of six were connected to the RPi trough an additional Amazon 7-port hub acting as a distributor. This though proved to problematic with uhubctl power cycle times of up to 6s when used in USB 3.0 mode. Using a D-Link DUB-H7 as distributor hub forcing the use of USB 2.0 the power cycle time was 2.0s which was acceptable. A more serious issue was the limited number of devices enumerated. System logs on the RPi reported that the max number of devices the xHCI host supported was 32. Although the xHCI standard itself supports up to 256 devices with 31 endpoints each, the actual numbers a particular controller needs to support is not defined (Intel 2019).

The number of endpoints is also a concern in addition to the device count. USB devices must have a control endpoint, but they tend to have multiple endpoints. USB hubs also consume endpoints for ports and controllers. (USB-IF 2000.) In this case, the combination of USB devices and hubs resulted in the enumeration of only 16 devices on the RPi, considerably less than the available ports of 42. Using only the USB 2.0 D-Link model as a test, 24 devices were enumerated which still was not sufficient.

# 6 DEVELOPED SYSTEM

This section presents the developed system along with some additional comments and discussions. The development process began with an evaluation of the existing system to identify the issues to be addressed. Once the primary objectives were identified, research on USB and its use in Linux began and various hubs were acquired and evaluated. VirtualHere was then investigated to determine its different options and capabilities and it was found that keep using scripts was a more fitting approach than developing a continuously running application. JSON was chosen for the configuration files, and a configuration file structure was defined. Python scripts were developed first, followed by the Node.js backend and React frontend. Finally, scaling up the system with more hubs and devices revealed some critical concerns.

The section structure is as follows. The development environment used is briefly described first, followed by a short introduction to the entire system. The JSON configuration files are then presented followed by the Python scripts that interact with VirtualHere. Then, the web interface backend and frontend are described. Afterward, the enclosure in which the system is housed is presented, followed by a discussion of security and reliability concerns.

## 6.1 Development environment

The editor used during development was Visual Studio Code (VS Code), which is a free, open-source code editor developed by Microsoft. This editor provides a flexible and powerful editing environment, which can be enhanced with various extensions. It supports numerous programming languages and platforms and is compatible with Windows, macOS, and Linux. (Microsoft 2023.) Almost all development was conducted on the RPi itself using the SSH-Remote extension for Visual Studio Code.

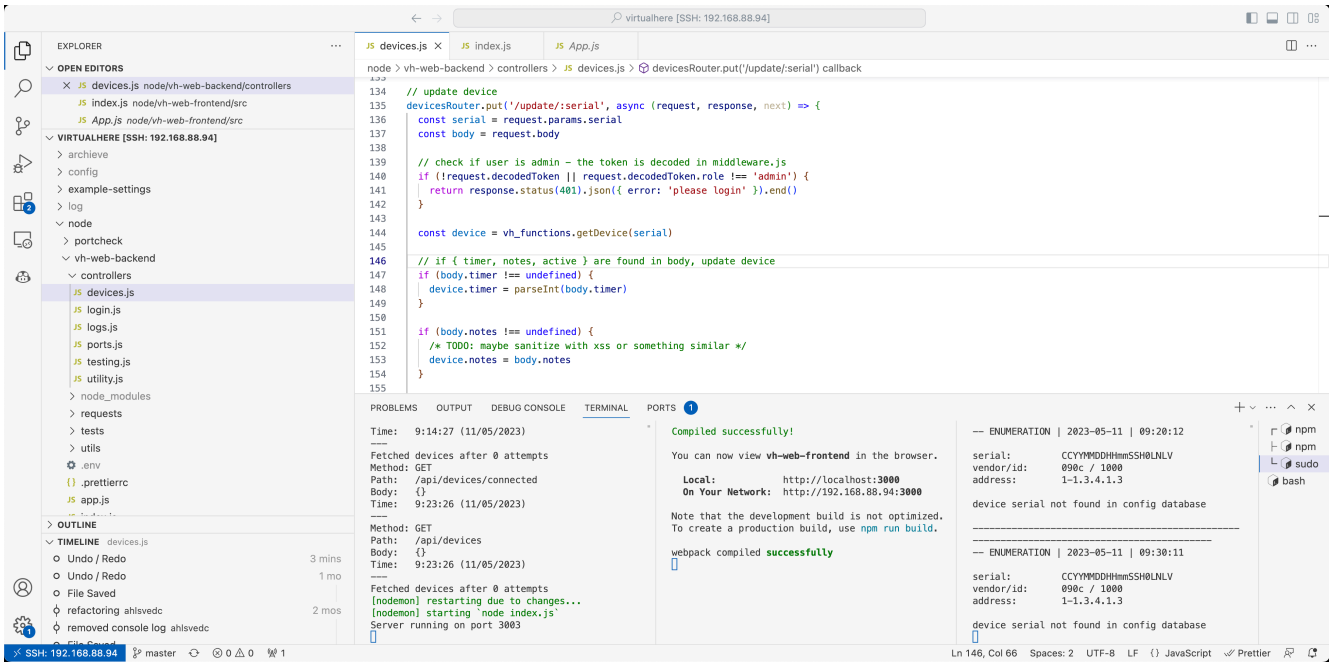



FIGURE 7. Working locally on RPi using VS code and SSH-remote extension

With this extension, a remote machine with a SSH server can be used as a development environment, and commands and many extensions can be run directly on the remote machine. Any folder on the remote machine can be opened and worked with as though it were on the local machine and this allows to use more specialized hardware than the local machine, thereby simplifying and improving the development process. (Microsoft 2021.) Figure 7 shows an example of VS Code connected to the RPi during development where local files are visible to the left and three terminals on the bottom are used to show the Node.js backend, React frontend, and VirtualHere server output running on the RPi. Additionally, git and GitHub were used for version control and as code repository. The overall structure of the project can be seen in figure 8 below.

```
├── config                      JSON configuration files
├── log                         Log files
├── node
│   ├── vh-web-backend          Node.js backend folder.
│   │   ├── controllers         Files responsible for serving endpoints
│   │   ├── requests            (used during development only)
│   │   └── utils               Mainly functions.js (common functions)
│   └── vh-web-frontend         React frontend folder.
│       ├── build               Build folder for frontend. Served by backend.
│       └── src
│           ├── components      React components
│           └── services        Files used for communication with backend
├── python                      Python scripts
└── symlink -> /run/user/1000/  Path for RAM disk. Used for temporary files.
```

FIGURE 8. Overview of project folder structure

## 6.2 Overview

The system is composed of several separate components but has two distinct parts. Scripts that interact with VirtualHere using Python and a web interface used for user interaction and configuration, powered by Node.js and React. Both parts share the same configuration files and use uhubctl for controlling power of USB hub ports. Figure 9 below provides an overview.

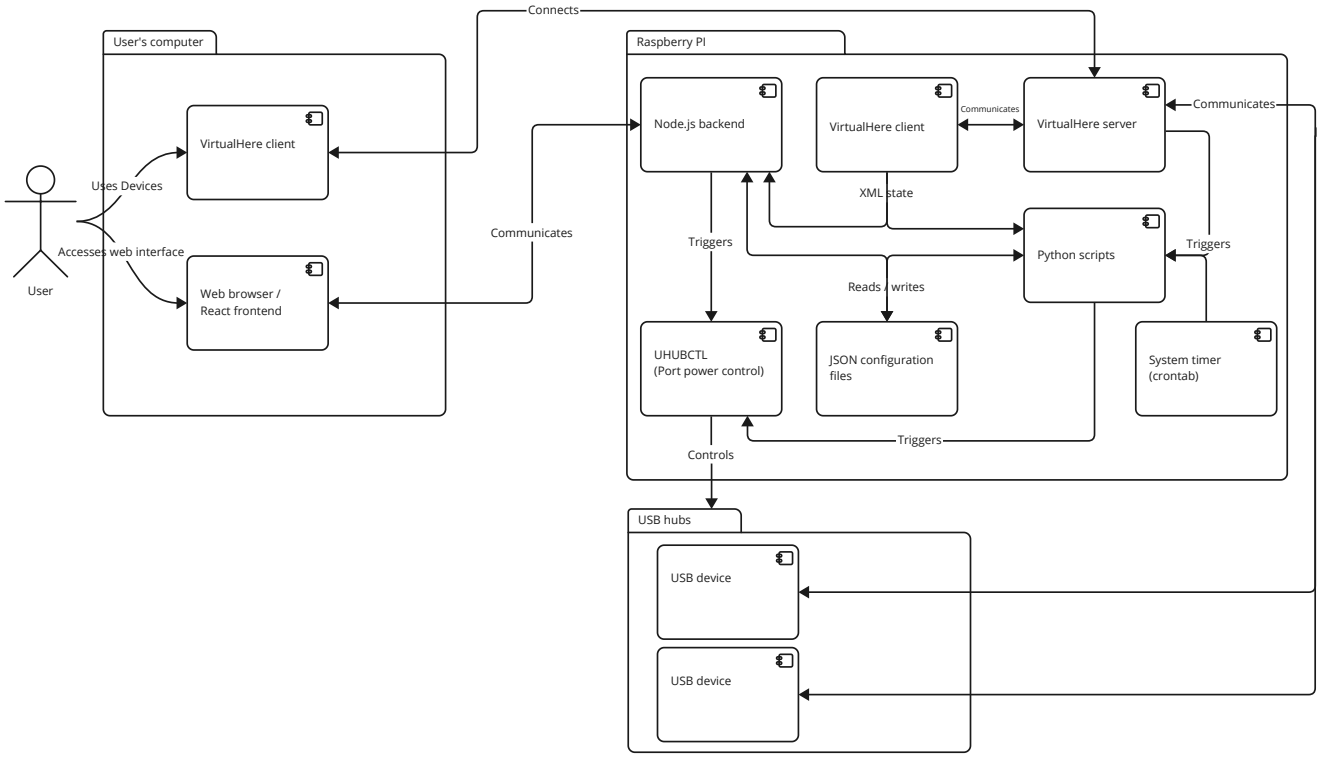



FIGURE 9. Overview of system components

The Python scripts extends the functionality of the VirtualHere server primarily by adding the ability to power cycle a USB device thus also the ability to disconnect a user after a specified time. This enables iLok devices to power cycled and consequently used by another user. In addition, scripts run at set intervals to check if devices are disconnected and attempts to power cycle the USB ports to restore them if possible. Another script alerts by email at the end of day if a device has disconnected more than a set threshold. The VirtualHere client running on the RPi is connected to the VirtualHere server on the same host and serves as a source for detecting connected devices by using XML state output of client. The Node.js backend serves a REST API as well as a web interface developed in React. The backend provides a variety of REST API endpoints that the React frontend utilizes to read and write configuration data. Furthermore, there are endpoints that enable the system to be rebooted or shut down.

To re-use code the Python scripts use a vh_functions.py module that each script imports. It contains common functions, such as reading and writing configuration, translation functions, as well as functionality for power cycling USB ports with uhubctl. The same applies to Node.js where a corresponding vh_functions.js file is used. Additionally, there is also an environment file used by both Python and Node.js.

### 6.3 Raspberry Pi

The required steps to get the system up and running on a RPi running a fresh install of Raspberry Pi OS Lite is to install Git, Node.js, uhubctl, VirtualHere server, and VirtualHere client. Then clone the git repository for the project from GitHub and copy the environment variables, as they are not part of the repository. To start all services at boot they need to be added to systemd. Lastly, crontab need to be configured to run required Python scripts at intervals.

The choice of the RPi for this project was based on several factors. First, the RPi has an excellent track record within the company, with multiple devices operating continuously for years without issues. Second, the RPi has strong support from both the RPi Foundation, community, and VirtualHere. The small size of the RPi further makes it easy to fit in a small enclosure. Additionally, the GPIO pins provide the option for a hardware interface with buttons and displays. In the end, however, a hardware interface was not required, as the user interface was implemented using a web interface. During the thesis process, a component shortage made it difficult to obtain RPi devices. The use of another platform was considered but never realized as the advantages still outweighed the disadvantages.

### 6.4 Configuration files

JSON was chosen for the configuration files as it is a commonly used technology and therefore works well with both the web interface and Python scripts. Initially, USB device path addresses in Linux were used to identify devices in the configuration files. While this method was functional, it resulted in an inflexible structure, which was one of the issues with the old system. Using USB serial numbers rather than addresses enabled a more flexible structure, since the JSON file did not need to reflect the physical arrangement of hubs and devices in any way. As unique serial numbers are required, USB devices with-

out them cannot be used, which is a limitation of the new system. Observations from devices tested during development showed that a selection of USB flash drives did not have a proper serial number.

The use of a database was considered, but files were chosen for simplicity to avoid the Python scripts being dependent on a database to function as configuration changes are rare. It is expected that the system will have minimal writes over time. However, there could be issues if a Python script and the backend write to a file simultaneously. To solve possible issues with write conflicts a file locking mechanism could be used for writes (Molay, B. 2003). The REST API could also be used in Python for file access via HTTP requests, although this would make the scripts dependent upon the backend and a database could as well be used.

The project contains two types of configuration files. Firstly, USB device configuration, which provides information about USB devices used in the system. Secondarily, port locations containing physical port names. The device JSON configuration file consists of an array of objects. Each object holds key-value pairs concerning a specific device and code 1 below depicts a single entry. The serial number is the primary identifier used to locate the device. Usually, the address of the device is passed from VirtualHere but when the device is disconnected, and the system tries to power cycle the port the latest known address is needed. The product ID is used in the web interface to sort devices by type. The state of the device indicates whether it is in use or not. The timer is for power cycle of devices and is expressed in seconds, where zero is no timer. The nickname is only intended for ease of reference in logs and the web interface and is the nickname used in VirtualHere. Logging if set to zero means no data will be logged, and if above zero logging is on. Typically, notes are used to provide extra information, such as if the device is disconnected and used elsewhere which helps organization and labelling. In the future more key-value pairs can be added as needed.

```
[
    {
        "serial": "3988A22A_7F293214",
        "address": "1-1.1.2.4.4",
        "product_id": "5036",
        "state": 1,
        "timer": 300,
        "nickname": "ilok - test",
        "logging": 1,
        "notes": "This is just a test note"
    }
]
```

CODE 1. Single device JSON device configuration example

The port location JSON file consists of the USB port address as key and location name as value. It is used to show device location in the web interface and as such help with organization and labelling of devices. The hub addresses provided below in code 2 refer to the Amazon Basics 7-port hub which is labelled HUB A within the enclosure.

```
[
    {"1-1.3.4.3.3": "HUB A - port 1"},
    {"1-1.3.4.3.2": "HUB A - port 2"},
    {"1-1.3.4.3.1": "HUB A - port 3"},
    {"1-1.3.4.3.4.3": "HUB A - port 4"},
    {"1-1.3.4.3.4.2": "HUB A - port 5"},
    {"1-1.3.4.3.4.1": "HUB A - port 6"},
    {"1-1.3.4.3.4.4": "HUB A - port 7"}
]
```

CODE 2. Port location JSON example

### 6.5 Python scripts

Python scripts are used for the integration with the VirtualHere server. The following VirtualHere server events are used, onBind, onUnbind and onEnumeration. OnBind occurs when a device is used in the client and the start_use.py script is then executed by the VirtualHere server. Whenever a device is stopped from being used on the client, the onUnbind event is triggered, which executes the stop_use.py script. Finally, the onEnumeration event occurs at device enumeration and executes the device_enumeration.py script that updates the device configuration if necessary. Using crontab, the port_check.py script is executed every ten minutes to check for disconnected devices, and the email_alert.py script is executed at the end of the day to alert the administrator if any devices have disconnected more frequently than a set threshold. The scripts will be described in detail below with flowcharts available in appendix 4.

To pass device data between VirtualHere and Python scripts, the VirtualHere server is configured to write data to a file on events which is then read by the Python scripts. Using files for communication between processes can be problematic. The writing process must finish before reading can begin, resulting in files generally being unsuitable for real-time communication (Molay, B. 2003). In this case, the VirtualHere server consistently finishes writing before triggering the Python scripts. Named pipes could otherwise been used as an alternative. The code 3 below is from the VirtualHere config.ini file where the behaviour is specified for events. The onBind event depicted occurs when a device is being used in the client. Each item between dollar signs represents data provided by VirtualHere. Rest are standard Linux commands. For readability, file paths have been removed. The process involves echoing a JSON

object into the vh_start_arguments.json file, and then executing the start_use.py script, which then internally reads the JSON file to pass the data.

```
onBind=echo
'{
   "address":"$ADDRESS$",
   "devpath":"$DEVPATH$",
   "vendor_id":"$VENDOR_ID$",
   "product_id":"$PRODUCT_ID$"
}'
> vh_start_arguments.json && python3 start_use.py&
```

CODE 3. VirtualHere server event config example for onBind

### 6.5.1 Enumeration

The enumeration script is triggered by the VirtualHere server whenever a device is enumerated, which occurs when a device is plugged into a USB port or when a port is power cycled. VirtualHere is configured to pass the data that can be seen in code 4 below. The purpose of the script is to update the device address and nickname in the configuration if they have changed. Otherwise, if a device is removed and replaced by another, the system would assume that the old device is missing. This would result in an incorrect power cycle of the port by the port check script.

```
{
   "address":"$ADDRESS$",
   "devpath":"$DEVPATH$",
   "vendor_id":"$VENDOR_ID$",
   "product_id":"$PRODUCT_ID$",
   "nickname":"$NICKNAME$",
   "serial":"$SERIAL$",
   "product":"$PRODUCT$"
}
```

CODE 4. VirtualHere onEnumeration data

### 6.5.2 Start use

The start use script is triggered by VirtualHere server when a user starts using a device in the client. The main purpose of the script is to allow timers for iLok devices. The data shown in code 5 is passed from VirtualHere. Unfortunately, VirtualHere does not pass the device's serial which is needed to find the

device configuration as the serial is used as the identifier. The script instead retrieves it from the Linux sysfs filesystem using the device devpath which is passed.

```
{
   "address":"$ADDRESS$",
   "devpath":"$DEVPATH$",
   "vendor_id":"$VENDOR_ID$",
   "product_id":"$PRODUCT_ID$"
}
```

CODE 5. VirtualHere onBind data

Once the serial is retrieved the script checks if it is found in the device configuration file. In that case the script checks if a timer other than zero is set. If true, the script waits for the specified time and power cycles the port. To cycle ports using uhubctl, the Python subprocess module is utilized, which allows new processes to be created while also connecting to the input and output of the process (Python Software Foundation 2023b). This enables the uhubctl utility to be run from within Python while passing required variables as hub and port. An example of this can be seen in code 6 below.

```
subprocess.run(['sudo', 'uhubctl',
                '-a',               # action
                '2',                # cycle power
                '-d',               # cycle time
                '2',
                '-l',               # limit to hub
                hub,
                '-p',               # port
                port])
```

CODE 6. Python subprocess with uhubctl

The script continues to run until the timer has elapsed. A user may however stop using the device before the timer expires and another user may now use the device. To prevent power cycles in this case, incorrectly disconnecting the current user, the script writes a file with a random timer ID with a filename of the address. Before power cycling the port, the script confirms whether the timer ID still matches. If another user would have started using the device, the timer ID would have been overwritten and would no longer match. Additionally, the timer ID gets a pending status for a brief period after the power cycle before the ID file is deleted. This state is used by the stop use script to prevent further unnecessary power cycling.

### 6.5.3 Stop use

Stop use is triggered by the VirtualHere server when a user stops using a device in the client or when power is cycled for a device. The script is primarily used to power cycle iLoks. Unfortunately, this script is also triggered whenever the start use script power cycles a device. To prevent unnecessary power cycles the timer ID is checked for the pending state, which indicates that the device has recently been power cycled by the start use script and should not be power cycled again.

### 6.5.4 Port check and alert

The port check script is executed by crontab at 10-minute intervals. Its primary function is to identify disconnected devices and initiate a power cycle to potentially bring them back. Additionally, if logging is enabled for a device, an entry is added to a power cycle log to keep track of problematic devices. Testing was done to determine whether it is possible to detect disconnected USB devices directly in Linux. Reproducing the intermittent fault of disconnected devices was extremely difficult, making consistent testing almost impossible. Reading the systfs directories and USB descriptors proved challenging and therefore using VirtualHere client state was deemed more robust and simpler. The script determines connected devices by utilizing the VirtualHere client operating locally on the RPi, which can output its state in XML format. The library xmltodict is used to, as the name implies, convert the XML to a Python dict which is a data structure which resembles the structure of JSON (PyPI 2022). By comparing the serial numbers found in the XML output to the devices expected in the JSON device configuration file, the script can determine whether a device is missing, and a power cycle should be done for a device.

An additional alert script is executed by crontab at the end of each day. Its primary purpose is to notify the system administrator via email about any problematic devices for that day. The script searches the power cycle log generated by the port check script and counts the occurrences of each device. The logs are rotated every day meaning only one day of entries are counted. If a device's count surpasses a set threshold, an email alert is sent to the administrator. The email is sent using the standard Python smtplib library.

### 6.6 Web interface

When developing the web interface, the primary focus was on the functionality provided rather than the specific technology used. The main purpose of it is to enable user interaction with the system, offering options to edit device configuration and perform actions such as shutdown and reboot. This section first presents the REST API backend followed by the React based frontend.

#### 6.6.1 Backend

The Node.js backend functions as a REST API that provides endpoints to the React frontend that it also serves. The main resource URL endpoints of the API can be found in table 3 below. The endpoints are described in more detail in appendix 1. Important endpoint functionality includes device JSON configuration file access to be able to add, edit and delete devices. Further, endpoints for the execution of shell commands are important. This includes uhubctl for power cycling ports and the VirtualHere client for obtaining state data in XML format. A similar approach as in Python is used to execute shell commands, but with Node.js the exec method from the module child_process is used (Node.js Foundation 2023). To parse the XML to suitable JavaScript objects the library node-xml2js is used (Leonidas-from-XIV 2023).

TABLE 3. REST API resource URLs

| | |
|---|---|
| /api/devices | Endpoints concerning devices |
| /api/ports | Endpoints concerning ports |
| /api/logs | Endpoints for logs |
| /api/utility | Endpoints for utility functions as reboot and shutdown |
| /api/login | Endpoint for login |

One issue encountered while working with the VirtualHere client to output XML was that the client would not reliably work. This caused issues in the frontend as it needed the data. To solve this the backend tries to run the command three times with a delay in between before aborting. This solved the issue completely. Another solution would have been to make the frontend fetch the data several times until successful.

### 6.6.2 Frontend

The react web interface frontend has two parts: the device list and the utility page. The central part is the device list, where devices can be listed in a tabular format. By selecting a device from the list, it can be added, edited, or deleted. Another useful feature is the ability to manually power cycle devices, which also helps in locating devices as this causes the LEDs on the USB hub ports to blink. The utility page provides other functionality such as restarting and shutting down the system. To access key features, user login is first required. At the time of writing, however, there is no central user database implemented.

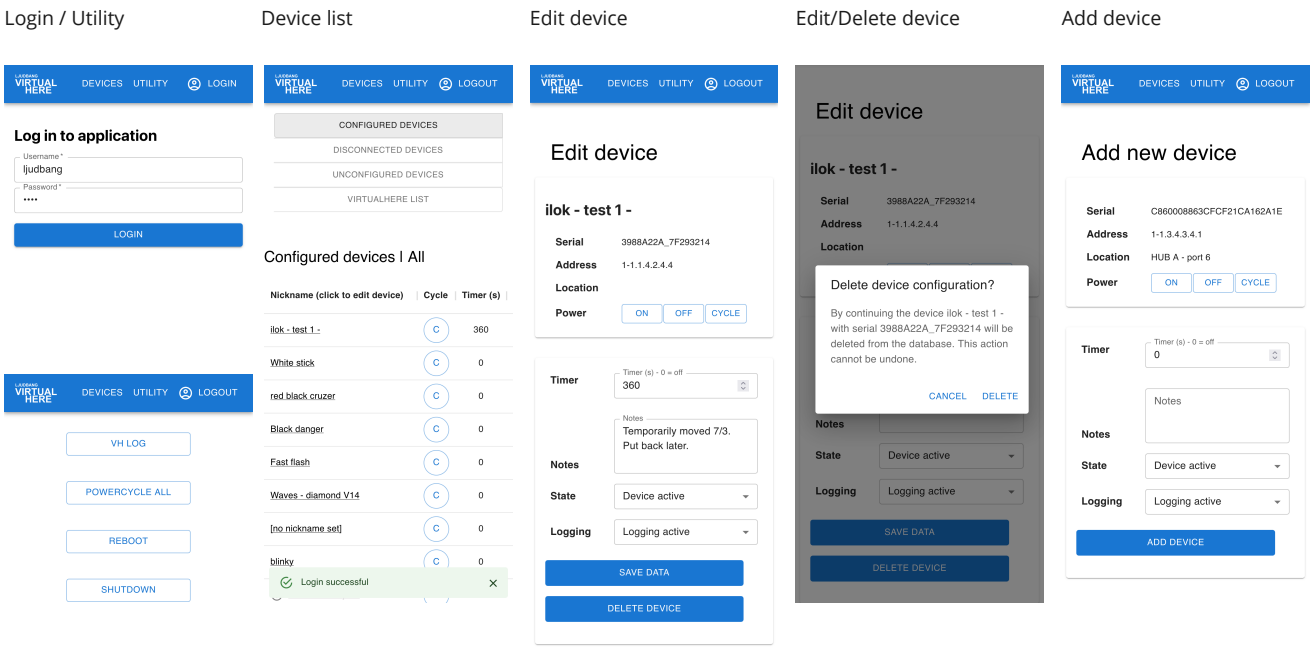



FIGURE 10. Frontend in mobile view

Mobile was the priority in the development of the frontend since it would likely be used in the server room on a mobile device. Above in figure 10 are examples of pages in mobile view. The desktop views with further descriptions of the interface can be found in appendix 3. The web interface does not require authorization for access to the device list, though any other operation requires login, or the option will be greyed out or the operation will cause a notification and fail. If username and password is correct at login a JWT token is received and stored in a variable and in the browser's localStorage. The logout functionality clears localStorage and stored variables.

A DataGrid table from MUI is used to show device lists, and each column can be sorted and hidden without additional frontend code. The most important information is visible at first, including nickname,

a power cycle button, and timer setting. Scrolling to the right of the main device list reveals the other columns as notes, location, serial and other information. By clicking on the device nickname the device can be edited. On this page the device power can be controlled as well, and device configuration as timer, notes, state, and logging be edited. Aside from saving device data, the device can be deleted, which will display an additional confirmation dialog. Unconfigured devices can be added by clicking the serial in the main device list and this provides the same user interface as when editing a device. The different device lists are created by comparing devices found in the JSON configuration file and connected devices detected by the VirtualHere client. Besides configured and connected devices, disconnected and unconfigured device lists can this way be created. Port names are added separately to all the lists.

The utility page's main purpose is to make it possible to reboot and shutdown the RPi. There is also a power cycle all function that triggers a power cycle of all ports found in the ports JSON configuration file to quickly reset all the USB devices in the system. Selecting any of the options will show an additional dialog to confirm the action. The VirtualHere log functionality is experimental at the time of writing and currently shows the main VirtualHere server output. It has a simple auto-update function that polls the backend every 2 seconds. The current implementation of fetching logs from the backend is by capturing the screen output of the VirtualHere server and writing it to a file served by the backend. As it turns out, when writing screen output to a file it gets buffered, resulting in not everything being written to the file in the right order. This needs further attention, although the most appropriate solution is to implement proper logging instead of using screen output.

At first much of the business logic of creating different device lists was in the backend. Most types of device lists are dependent on getting the XML status from the VirtualHere client running on the RPi, and this is a much slower process than reading JSON files. This was making the response of the frontend slow as it had to wait for the VirtualHere client XML output on all requests. Fetching data from separate endpoints on the backend and then combining the data on the frontend solved the issue.

### 6.7 Enclosure and USB hubs

It was necessary for the enclosure to fit within a 19" rack since these racks are used in the server rooms. The idea of using a rack drawer was considered at first, but internal cabling and heat concerns ruled that out in favour for a 10" rack cabinet that also could be moved more easily. For the enclosure design overall the most important part was easy access to the USB devices and adequate cooling. Having a lock

was also advantageous to prevent theft. The enclosure selected for the project is a 10-inch, 6U steel server cabinet manufactured by HMF. The cabinet has a lockable safety glass door and removable side panels for easy access. Furthermore, it has good airflow and provides space for a 120 mm fan at the top. (HMF 2023.) The case used for the RPi is manufactured by Flirc and made of aluminium. The whole case serves as a large heat sink which can be seen in picture 4 below. GPIO and all the main connectors are accessible through slots at the bottom, as is the SD card. (Flirc 2023.)

PICTURE 4. Flirc RPi 4 aluminium case (Flirc 2023)

The hubs used are mentioned in section 5.2. The Amazon Basics 7-port hub is used for devices and to connect the Amazon hubs to the RPi the D-Link DUB-H7 is used. The 10-port variant of the Amazon basics hub could have been used if being careful with not exceeding the six tiers permitted by VirtualHere. The 10-port though uses a 20V power supply that could be harder to quickly source if it would break. The 7-port version uses 12V which is commonly used within the company. Using self-powered 4-port hubs was deemed impractical in the end due to the large number of hubs that would be required.

To address the problem with limited device enumeration on the RPi encountered when scaling up the system with more hubs, tests were carried out using the RPi USB-C port. This port has the legacy USB 2.0 controller used in previous RPi models (RPi Foundation 2023c). By distributing the USB hubs between the primary USB 3.0 xHCI controller and the legacy eHCI controller associated with the USB-C port, an increased number of devices could be enumerated. In addition, this resolved the problem of the slow power cycle of six seconds that occurred when USB 3.0 was used alone. Unfortunately, only 31 USB devices were available for testing during this process, a number that the legacy controller was able to handle by itself. As a result, the system is expected to be able to utilize all 42 USB ports available from the six 7-port hubs used.

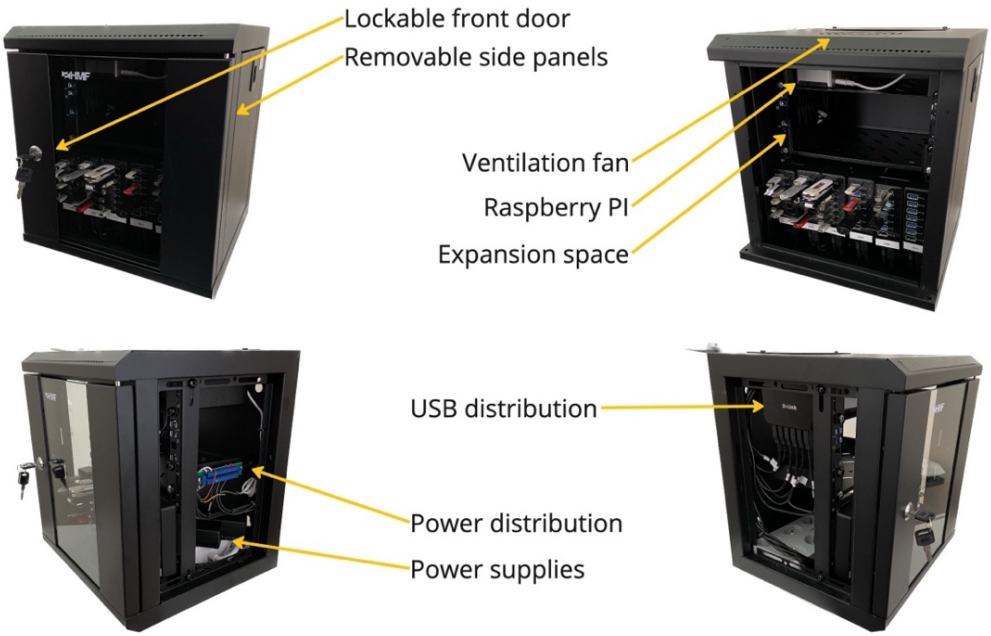



FIGURE 11. Overview of case

Two extra shelves were added. One for the RPi, while the other is for future expansion. The six Amazon USB hubs were installed horizontally on empty rack panels in the front to facilitate easy organization and accessibility. Figure 11 above illustrates how the cabinet space is utilized. To save space and reduce heat generation, the number of power supplies for the USB hubs was decreased from six to two. This was based on the observation that the power consumption of most USB devices was much less than reported by the device descriptors. On the right side of the enclosure, two 12V 36W power supplies are used to power the six USB hubs via a power distribution block. The RPi has its own power supply. The other side is dedicated to connecting the device USB hubs to the RPi.

### 6.8 Security and reliability

One of the means to keep the system safe is to keep it isolated both physically and network wise. The system will be used in a locked server room with no access to unauthorized third parties. Further, a wired network connection will be used, and the system will not be directly exposed to the Internet as it is behind company firewalls. Remote access to the system will be provided by the company's VPN solution. As a low-cost device, the design of the RPi is not primarily focused on security, and it should be protected from nonauthorized access (Sainz-Raso, Martin, Diaz & Castro 2019). The fact that the system is locked away and not directly connected to the Internet solves some security concerns. The licenses cannot easily be stolen without access to both the physical USB device and the license credentials. For

the Raspberry Pi OS the default username and password on Raspbian are significant security risks and should be changed. Further, considering how to secure SSH if it will be used should be done as well. (Le, Grande, Carmine, Thompson & Khan Mohd 2022). In addition to using a strong password, the password-based SSH access will be disabled on the system, and only key-based access permitted. Currently, the web interface uses unsecured HTTP communication which presents a risk, although the web interface is only intended to be used on the local network. While used remotely, the connection is encrypted by the company VPN service. It is possible that SSL will be added in the future. The login functionality using JWT tokens prevents unauthorized access to the web interface itself. Reliability of the system can be ensured partly by operating multiple systems with extra capacity in the event of failure. Spare parts will also be available for quick replacement. Open-source project Pi clone will be used to make bootable backups of the system SD card for quick recovery (Raspberry Pi-UI 2023).

# 7 CONCLUSIONS

The aim of this thesis was to further develop an existing VirtualHere-based system used for sharing USB license devices over a network. The original system, in operation for over five years, had several shortcomings, including the lack of detection of improperly disconnected USB devices, hard-coded settings in the code, and limited user interaction. The main goals were to add detection and automatic power cycle of disconnected USB devices to the VirtualHere server, as well as provide a user interface for interaction with the system. The original system was poorly coded using BASH scripts and thus the code interfacing with VirtualHere was rewritten in Python with the settings defined within separate JSON configuration files. The ability to detect disconnected devices was added, along with the possibility of automatically power cycling devices, logging the event, and alerting an administrator by email. The power cycle on the new system employs USB hub per-port power switching instead of mechanical relays used in the old system. A web interface using Node.js and React was developed to enable user interaction, allowing easy editing of USB device configurations as well providing means for rebooting and shutting down the system. In the end, the components were placed in a suitable enclosure.

The system meets the initial requirements and presents a more reliable and easy-to-use system, although not all features are completed. Viewing logs via the web interface is an experimental feature, and there is no user database for the login functionality. The web interface also uses unencrypted HTTP traffic. Further field testing is also necessary to identify any unforeseen issues and test performance in the field. To address potential issues and improve functionality, further development is likely to be required.

Many valuable lessons were learned during the development process. One significant lesson was the importance of testing for scalability early on. It was not until a relatively late stage in the development process that the system was scaled up with more USB devices and hubs attached. This resulted in performance issues and revealed limitations in how the RPi 4 handles an increased number of USB devices. A better approach would have been to anticipate scalability issues from the beginning and plan the system accordingly. It was also learned that a consistent programming language and function definition would have improved the development process. Initially, it was intended to minimize the use of Node.js and focus more on Python. Nevertheless, as the project progressed, more functionality was added to the web interface and more functions were ported to Node.js, resulting in several versions of functions that needed to be maintained. The development process would have been simplified if there was a more cohesive approach, either by utilizing Node.js or Python fully. The chip shortage issue also impacted the

project, making it challenging to obtain RPi devices. Considering a more general approach, rather than one centred around the RPi ecosystem, could reduce supply chain limitations in the future.

Further development suggestions include developing the logging capability. This could provide valuable insights into device usage, which in turn could help make informed business decisions about license needs. To reduce the disconnection rate of USB flash drives, further power consumption measurements could be conducted on devices running in both USB 2.0 and USB 3.0 modes to determine whether power consumption and consequently heat generation differ greatly between the two modes. The system now uses USB 2.0 but USB 3.0's improved power management and architecture may decrease heat and possibly increase reliability. Implementing unit, integration, and stress tests could improve system reliability and performance as well as identify potential issues. The most effective option would be to implement a deployment pipeline with automated testing, build, and deployment. At this stage, however, due to the small number of systems that will be in use, it has not been considered a priority.

# 8 REFERENCES

Amazon. 2023. Amazon basics USB 3.1 hub. Available at: https://www.amazon.de/AmazonBasics-USB-Hub-Anschl%C3%BCssen-Netzadapter-Schwarz/dp/B076YRSWH3. Accessed 24 May 2023.

Bojinov, V. 2018. *RESTful Web API Design with Node. js 10, Third Edition: Learn to Create Robust RESTful Web Services with Node. js, MongoDB, and Express. js*, 3rd Edition.

Digi. 2023. AnywhereUSB Plus. Available at: https://www.digi.com/products/networking/infrastructure-management/usb-connectivity/usb-over-ip/anywhereusb. Accessed 9 May 2023.

D-Link. 2023. DUB-H7 7-Port USB 2.0 Hub. Available at: https://eu.dlink.com/fi/fi/products/dub-h7-7-port-usb-2-0-hub. Accessed 24 May 2023.

Ecma International. 2017. *ECMA-404 The JSON Data Interchange Standard.* 2nd edition. Geneva: Ecma International.

Eltima. 2023. Eltima Software. Available at: https://www.eltima.com. Accessed 9 May 2023.

FabulaTech. 2023. USB over Network - Share USB devices over network or the Internet. Available at: https://www.fabulatech.com/usb-over-network.html. Accessed 9 May 2023.

Flirc. 2023. Flirc Raspberry Pi 4 Silver. Available at: https://flirc.tv/products/flirc-raspberrypi4-silver. Accessed 9 May 2023.

HMF. 2023. 10-Zoll Serverschränke. Available at: https://hmf.de/10-zoll-serverschraenke. Accessed 9 May 2023.

IncentivesPro. 2023. IncentivesPro - USB Redirection and Remote Scanning over Network, Remote Desktop or Internet. Available at: https://www.incentivespro.com. Accessed 9 May 2023.

Intel. 2019. eXtensible Host Controller Interface for Universal Serial Bus (xHCI) Requirements Specification. Available at: https://www.intel.com/content/dam/www/public/us/en/documents/technical-specifications/extensible-host-controler-interface-usb-xhci.pdf. Accessed 10 January 2023.

Jose, M., Jones, M., & Bradley, J. 2015. JSON Web Token. Available at: https://tools.ietf.org/html/rfc7519. Accessed 9 May 2023.

Le, C., Grande, A. M., Carmine, A., Thompson, J. & Khan Mohd, T. 2022. *Analysis of Various Vulnerabilities in the Raspbian Operating System and Solutions*.

Leonidas-from-XIV. 2023. node-xml2js. Available at: https://github.com/Leonidas-from-XIV/node-xml2js. Accessed 11 May 2023.

Linux Foundation. 2015. Filesystem Hierarchy Standard. Available at: https://refspecs.linuxfoundation.org/FHS_3.0/fhs/index.html. Accessed 18 April 2023.

Linux Kernel Organization. 2023a. USB Driver API. Available at: https://www.kernel.org/doc/html/latest/driver-api/usb/index.html. Accessed 20 January 2023.

Linux Kernel Organization. 2023b. Sysfs filesystem documentation. Available at: https://www.kernel.org/doc/html/latest/filesystems/sysfs.html. Accessed 20 January 2023.

Microsoft. 2021. Developing on Remote Machines using SSH and Visual Studio Code. Available at: https://code.visualstudio.com/docs/remote/ssh. Accessed 9 May 2023.

Microsoft. 2023. Visual Studio Code. Available at: https://code.visualstudio.com. Accessed 9 May 2023.

Mozilla. 2023. Window.localStorage. Available at: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage. Accessed 9 May 2023.

MUI. 2023. MUI: A popular React UI framework. Available at: https://mui.com. Accessed 9 May 2023.

Negus, C. 2015. *Linux Bible, 9th Edition*.

Node.js Foundation. 2023. Child Process. Available at: https://nodejs.org/api/child_process.html. Accessed 9 May 2023.

O'Brien, K., Salyers, D., Striegel, A. & Poellabauer, C. 2008. *Power and performance characteristics of USB flash drives*.

Python Software Foundation. 2023a. Python Documentation. Available at: https://www.python.org/doc. Accessed 24 May 2023.

Python Software Foundation. 2023b. subprocess - Subprocess management. Available at: https://docs.python.org/3/library/subprocess.html. Accessed 11 May 2023.

PyPI. 2022. xmltodict 0.13.0 - XML to Python dictionary. Available at: https://pypi.org/project/xmltodict/. Accessed 9 May 2023.

Raspberry Pi-UI. 2023. PiClone. Available at: https://github.com/raspberrypi-ui/piclone. Accessed 9 May 2023.

React. 2023. React – A JavaScript library for building user interfaces. Available at: https://react.dev. Accessed 9 May 2023.

RPi Foundation. 2023a. Raspberry Pi OS. Available at: https://www.raspberrypi.com/software/. Accessed 9 May 2023.

RPi Foundation. 2023b. Raspberry Pi. Available at: https://www.raspberrypi.org. Accessed 18 April 2023.

RPi Foundation. 2023c. RPi documentation - the config.txt file. Available at: https://www.raspberrypi.com/documentation/computers/config_txt.html. Accessed 18 April 2023.

Sainz-Raso, J., Martin, S., Diaz, G. & Castro, M. 2019. Security Vulnerabilities in Raspberry Pi-Analysis of the System Weaknesses. *IEEE consumer electronics magazine,* 47-52.

Stern, A. 2014. Power Management for USB. Available at: https://www.kernel.org/doc/html/v4.16/driver-api/usb/power-management.html. Accessed 20 April 2023.

Systemd Project. 2023. systemd. Available at: https://systemd.io. Accessed 2 February 2023.

USB-IF. 2000. USB 2.0 Specification. Available at: https://www.usb.org/document-library/usb-20-specification. Accessed 27 July 2023.

USB-IF. 2022. USB 3.2 Revision 1.1. Available at: https://usb.org/document-library/usb-32-revision-11-june-2022. Accessed 18 April 2023.

VirtualHere. 2023a. Purchase. Available at: https://www.virtualhere.com/purchase. Accessed 8 June 2023.

VirtualHere. 2023b. VirtualHere. Available at: https://www.virtualhere.com/. Accessed 18 April 2023.

VirtualHere. 2023c. USB Server Software. Available at: https://www.virtualhere.com/usb_server_software. Accessed 18 April 2023.

VirtualHere. 2023d. USB Client Software. Available at: https://www.virtualhere.com/usb_client_software. Accessed 18 April 2023.

# APPENDIX 1/1

## REST API endpoints

| HTTP | Routes | Description | Auth | Expected JSON Object |
|---|---|---|---|---|
| Devices | | | | |
| GET | /api/devices/connected | Returns a list of connected devices from XML data of VirtualHere client | No | |
| GET | /api/devices/portinfo | Returns data from a Python script regarding port information. Experimental. | No | |
| GET | /api/devices/:serial | Returns a device by its serial number. | No | |
| GET | /api/devices/:serial/powercycle | Power cycles a device by its serial number. | Yes | |
| POST | /api/devices/powercycle | Power cycles a device by its address. CommandType: 0 = off, 1 = on, 2 = power cycle. | Yes | {"address": "string", “commandType: “int”} |
| PUT | /api/devices/update/:serial | Updates a device by its serial number. | Yes | {"location": "string", "description": "string"} (rest optional) |
| POST | /api/devices/add | Adds a new device. | Yes | {"serial": "string", "location": "string", "description": "string"} (rest optional) |
| DELETE | /api/devices/:serial | Deletes a device by its serial number. | Yes | |
| GET | /api/devices | Returns a list of all devices found in device JSON config file. | No | |
| Ports | | | | |
| GET | /api/ports | Returns a list of all ports found in ports JSON config file. | No | |
| GET | /api/ports/powercycleall | Power cycles all ports found in ports JSON config file. | Yes | |
| Login | | | | |
| POST | /api/login | Authenticates a user and returns a JWT token, username, and role. | No | {"username": "string", "password": "string"} |
| Logs | | | | |
| GET | /api/logs/vh_server | Returns the vh_server.log file. | No | |
| GET | /api/logs/vh_server/:day | Returns the vh_server.log file for a specific day. Experimental. | No | |
| Utility | | | | |
| GET | /api/utility | Returns a list of all ports. | No | |
| GET | /api/utility/reboot | Reboots the system. | Yes | |
| GET | /api/utility/shutdown | Shuts down the system. | Yes | |
| GET | /api/utility/powercycleall | Power cycles all ports found in ports JSON config file. | Yes | |

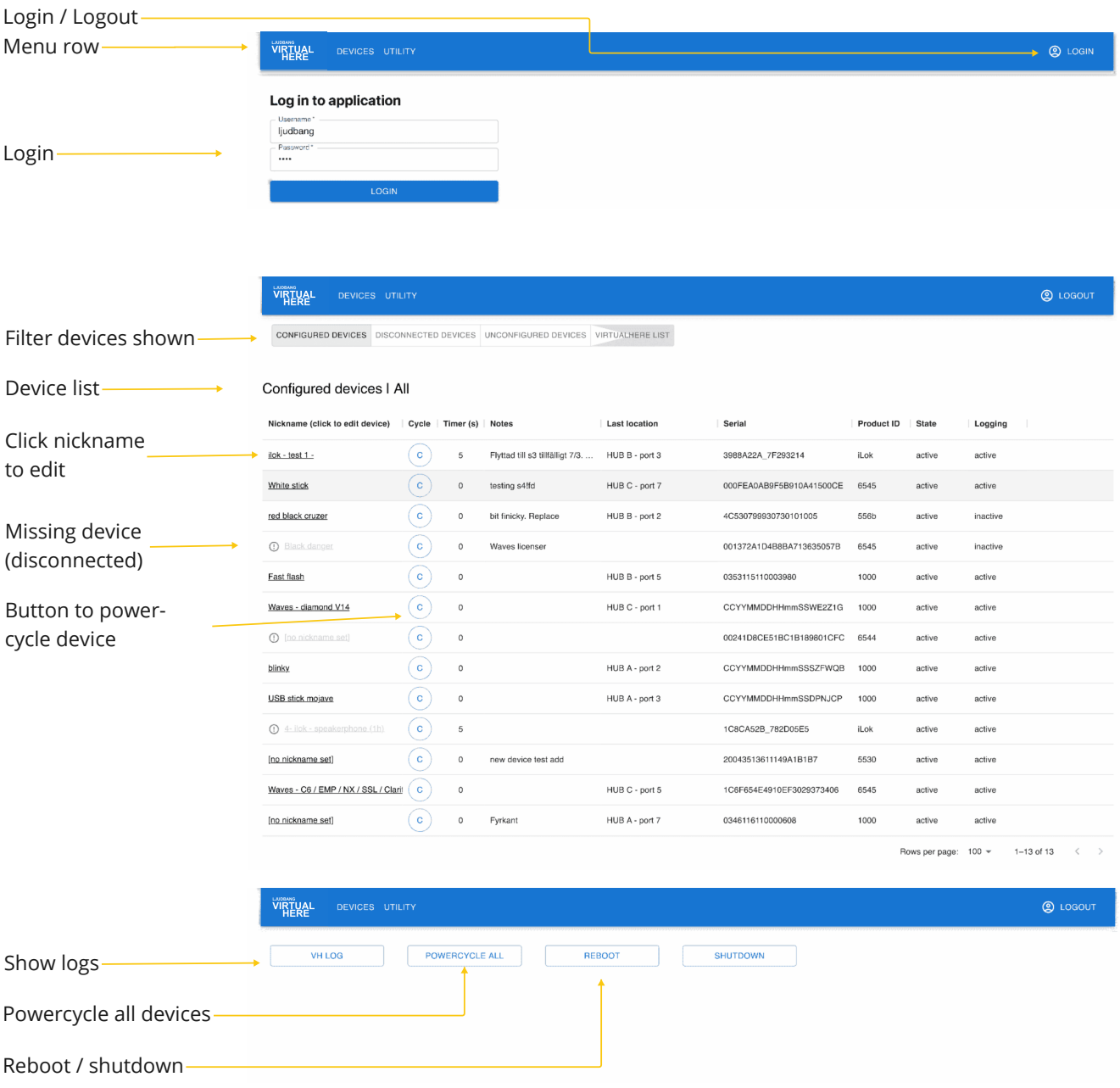
APPENDIX 2

MUI components used in React frontend

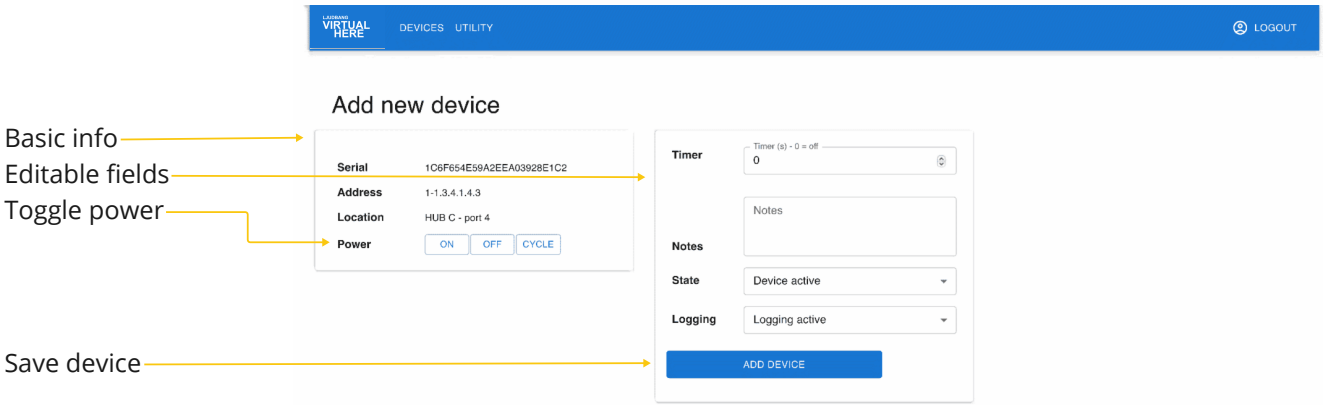
| | |
|---|---|
| Appbar | Used for branding, screen titles, navigation, and actions related to current screen. |
| Box | Manages layout. Used to arrange and structure elements. |
| Button | Clickable button |
| DataGrid | Show data in a grid format. Has interactive functions as sorting, filtering, and pagination. |
| Dialog | Informs users. Can contain critical information that require input from user. |
| Grid | Manages layout. Creates a grid of items that adapt to screen size and orientation. |
| Select | Select option from list of options. |
| Snackbar | Provide brief notifications (also known as a toast) |
| Stack | Manages layout. Creates vertical or horizontal one-dimensional layouts. |
| Textfield | For text input. |
| Tooltip | Provide users with information when hovering over an element. |

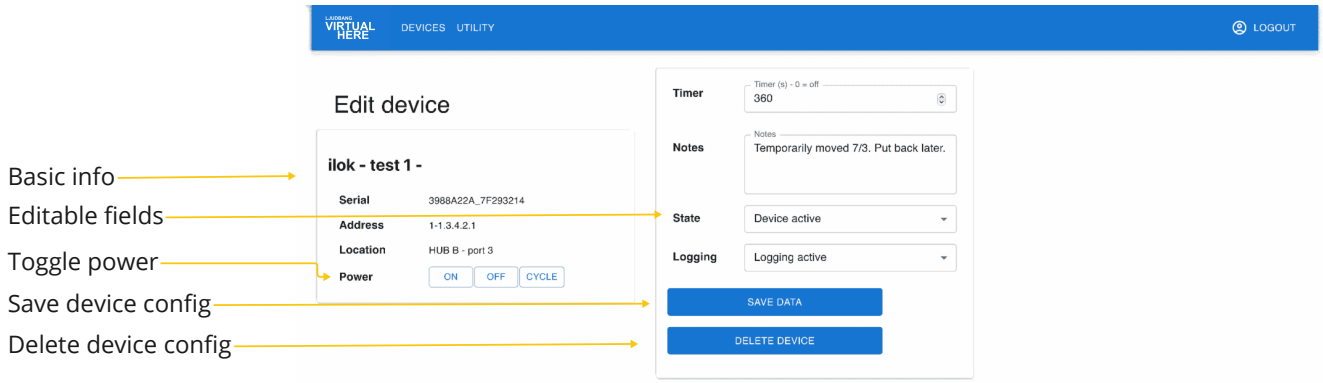
APPENDIX 3/1

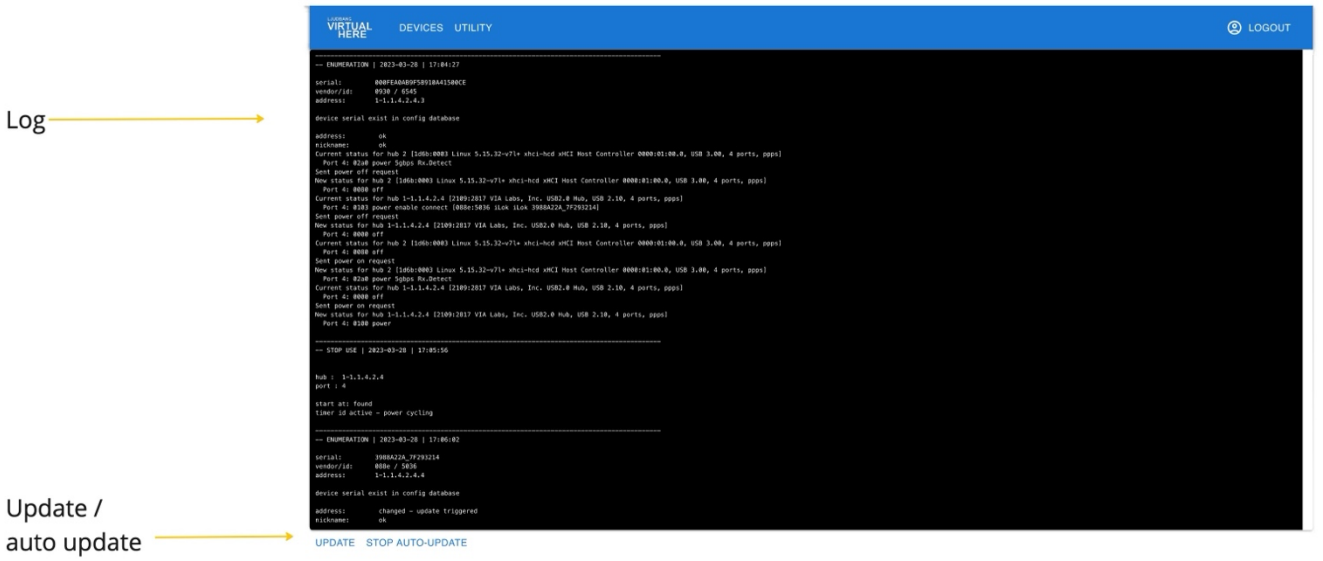
Web interface frontend in desktop view



# APPENDIX 3/2


VIRTUAL HERE
DEVICES
UTILITY
LOGOUT
Add new device
Basic info
Editable fields
Toggle power
Save device
Serial
1C6F654E59A2EEA03928E1C2
Address
1-1.3.4.1.4.3
Location
HUB C - port 4
Power
ON
OFF
CYCLE
Timer
Timer (s) - 0 = off
0
Notes
Notes
State
Device active
Logging
Logging active
ADD DEVICE

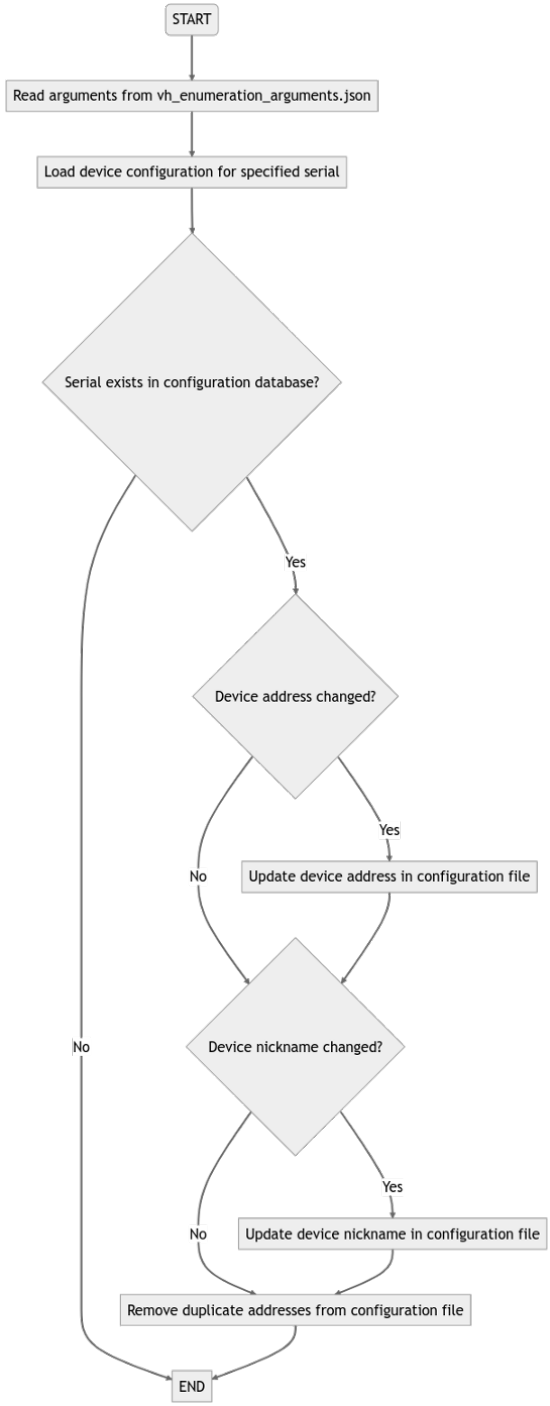

VIRTUAL HERE
DEVICES
UTILITY
LOGOUT
Edit device
Basic info
Editable fields
Toggle power
Save device config
Delete device config
ilok - test 1 -
Serial
3988A22A_7F293214
Address
1-1.3.4.2.1
Location
HUB B - port 3
Power
ON
OFF
CYCLE
Timer
Timer (s) - 0 = off
360
Notes
Notes
Temporarily moved 7/3. Put back later.
State
Device active
Logging
Logging active
SAVE DATA
DELETE DEVICE


VIRTUAL HERE
DEVICES
UTILITY
LOGOUT
Log
Update /
auto update
UPDATE
STOP AUTO-UPDATE

APPENDIX 4/1

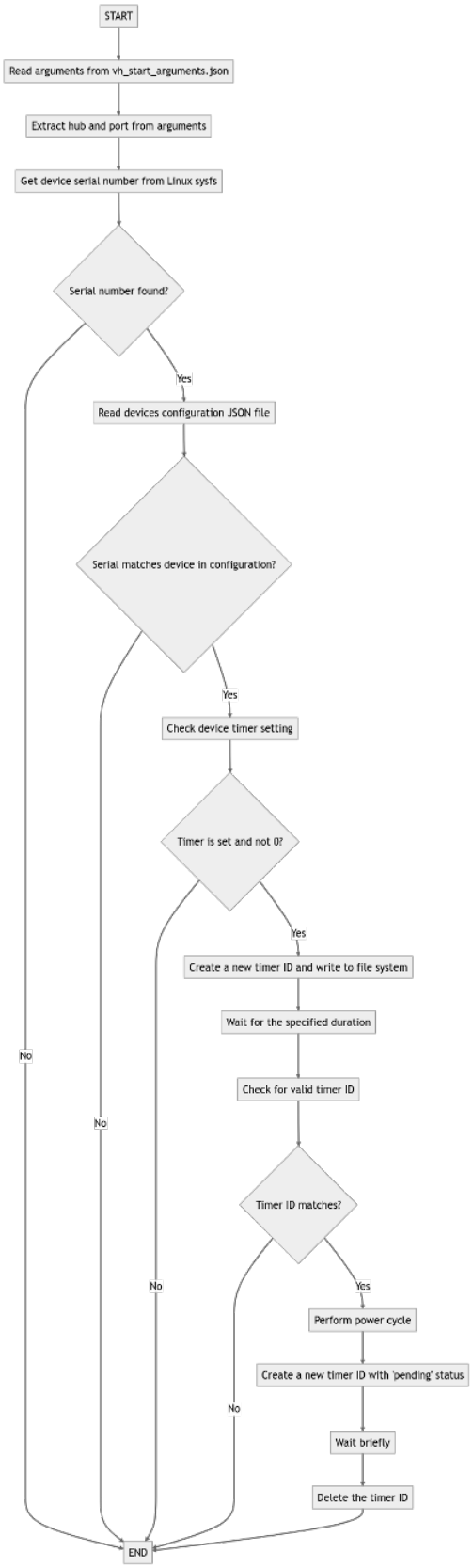
Python script flowcharts

device_enumeration.py

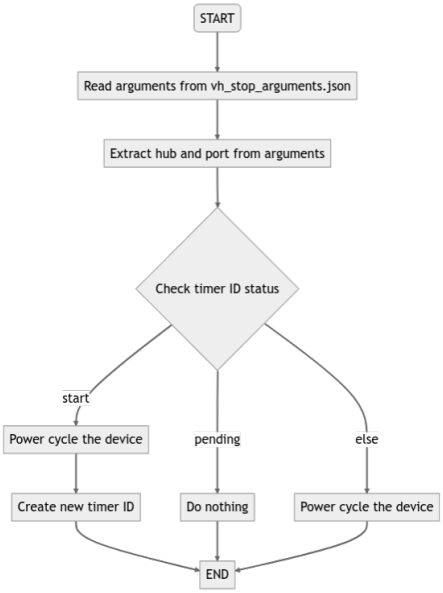


APPENDIX 4/2

start_use.py

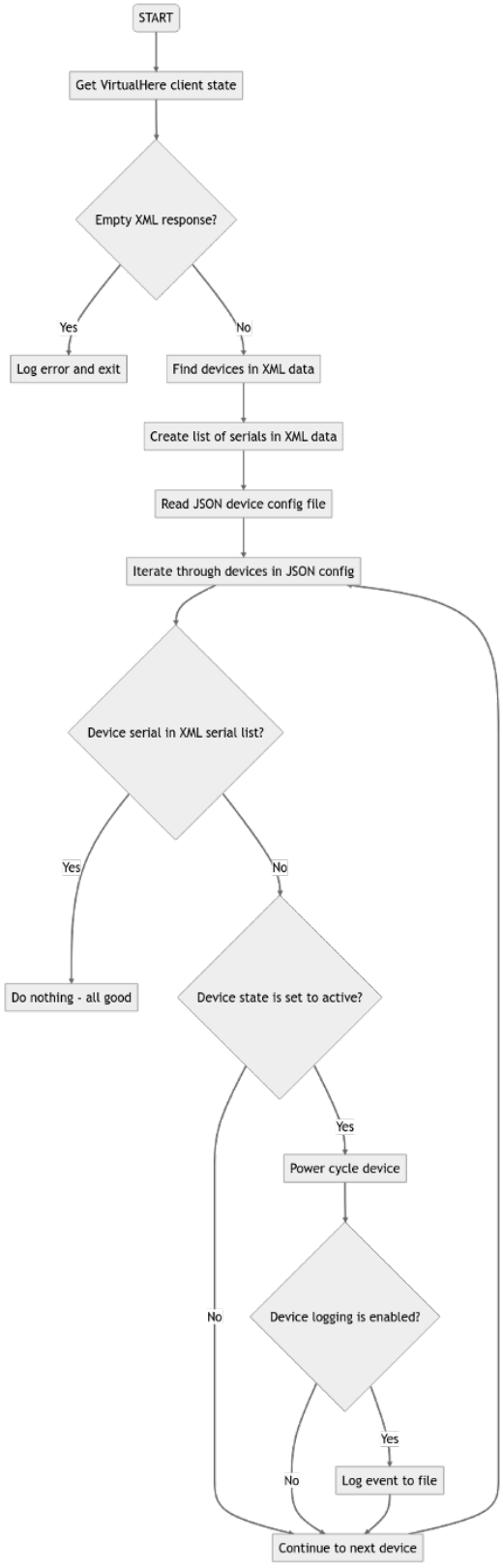



stop_use.py



APPENDIX 4/3

port_check.py




email_alert.py