ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

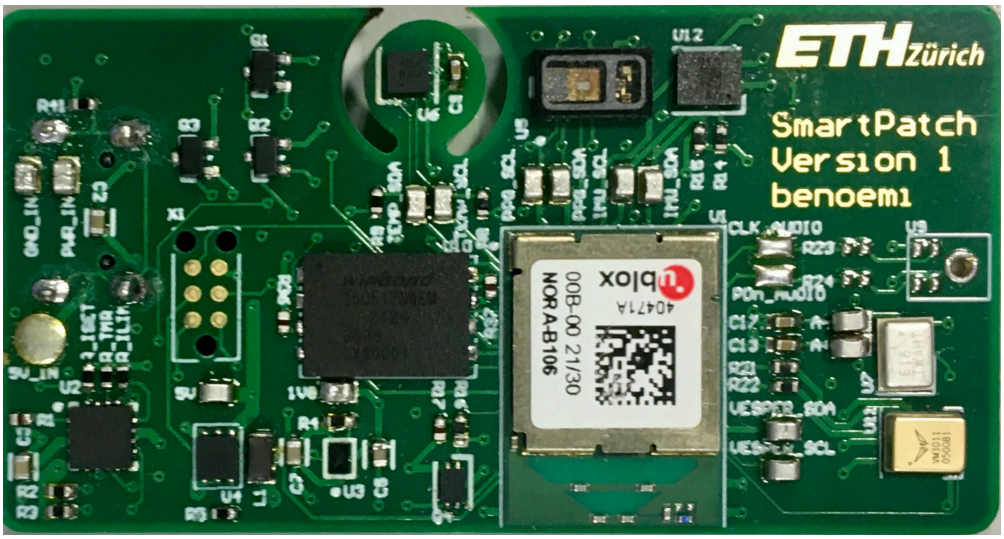
Center for Project-Based Learning
PBL

Department of Information Technology and
Electrical Engineering

Autumn Semester 2021

SmartPatch Project
SmartPatch System

Bachelor Project



Cyrill Knecht
cyrknech@student.ethz.ch

January 2022

Supervisors: Dr. Michele Magno, michele.magno@pbl.ee.ethz.ch
Silvano Cortesi, cortesis@ethz.ch, Christian Vogt, christian.vogt@pbl.ee.ethz.ch

Professor: Prof. Dr. Sebastian Kozerke, kozerke@biomed.ee.ethz.ch

Acknowledgements

I'd like to thank my group members for making the collaboration easy and fun. Also I'd like to thank Silvano and Christian for their constant support. Further, I'd like to thank the PBL for hosting this unique flagship project.

Abstract

The SmartPatch Project aims at developing a health sensor to collect data which could help in early detection of sepsis. The primary target group for this sensor, called SmartPatch from now on, are patients in intensive care units. The SmartPatch should be designed as a non-invasive, small, low energy and low cost device. In addition it should have a battery life of a few days and should be able to wirelessly transmit data to a nearby base station.

The goal of this thesis was to implement a system surrounding the actual SmartPatch, to access and store data and to make the use of SmartPatches in an easy and efficient manner.

To achieve this, an open source IoT platform called Thingsboard was used to visualize and store all SmartPatch sensor data. The SmartPatch Basestation, a Raspberry Pi running the SmartPatch Basestation Software, was used as a gateway between the actual SmartPatch and the Thingsboard server. Further, the cross-platform SmartPatch Connector App was implemented using the flutter framework. The app provides the possibility to map patients to SmartPatches.

To date, multiple Basestations can be connected to a Thingsboard server and each of those Basestations can connect to up to ten SmartPatches. For future large-scale deployment, there should be a rework of the security architecture and there also remain possibilities to improve efficiency.

Declaration of Originality

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor. For a detailed version of the declaration of originality, please refer to Appendix B

Cyrill Knecht,
Zurich, January 2022

Contents

List of Figures

List of Tables

List of Acronyms

API . . . . . Application Programming Interface

APK . . . . . Android application package

app . . . . . Application

BLE . . . . . Bluetooth Low Energy

IoT . . . . . Internet of Things

JSON . . . . . JavaScript Object Notation

MQTT . . . . . Message Queuing Telemetry Transport

PBL . . . . . Center for Project-Based Learning

QR . . . . . Quick Response

TLS . . . . . Transport Layer Security

UI . . . . . User Interface

Chapter 1

Introduction

This chapter is partly co-authored by Noemi Bernstein, Amane Zürrer, Andreas Hunziker and Robin Hunziker.

"Sepsis is a life-threatening condition that arises when the body's response to infection causes injury to its own tissues and organs. With early intervention, it is possible to mitigate the course of the disease, which reduces the risk of a fatal outcome. Due to the costs and difficulties of full-time monitoring of patients in intensive care units, this flagship project aims at developing a continuous patient monitoring system using a low cost health sensor. In collaboration with the University Hospital of Lausanne, this health sensor will then be used for early detection of sepsis."[A]

For this purpose, the team working on this flagship project developed a concept of a health sensor and its surrounding system. From this point on the actual health sensor developed in the project will be referred to as SmartPatch and the whole system that was implemented to use it, including the actual patch will be referred to as the SmartPatch System.

The initial concept of this SmartPatch System included the SmartPatch that should be placed on a patient's body. This SmartPatch should be able to measure:

- Heart rate
- Respiratory rate
- Blood oxygen saturation
- Body skin temperature
- Patient activity

Further the initial concept included a base station for the SmartPatch, our specific base station will from now on be referred to as *Basestation*, that should be in a room with the SmartPatches. The SmartPatch sensor data should then be sent over BLE to the Basestation and then there should be some way to display this data on a UI.

From there on, the project was split into several independent parts between the group members:

- Noemi Bernstein - Hardware and Packaging[3]
- Amane Zürrer - Firmware and Data Processing[4]
- Andreas Hunziker - Bluetooth Low Energy[5]
- Cyrill Knecht - SmartPatch System
- Robin Hunziker - Indoor Localization and Audio Acquisition[6]

This report will only explain my specific part of the SmartPatch flagship project, the SmartPatch System. Part of this work package were the Basestation Software, excluding the BLE interface and the data processing, the data accumulation and visualization in a UI, that is referred to as the SmartPatch UI and the implementation of an application to map patients to SmartPatches, referred to as the SmartPatch Connector App.

In Fig. 1.1 an overview over the final SmartPatch System that was developed is shown. All parts of it will be explained in detail in the Architecture and Implementation chapters.

Whenever there were intersections with work of other group members in this report, their individual report was referenced. Consider reading all reports of the group to get an overview of the capabilities of the SmartPatch System as a whole.

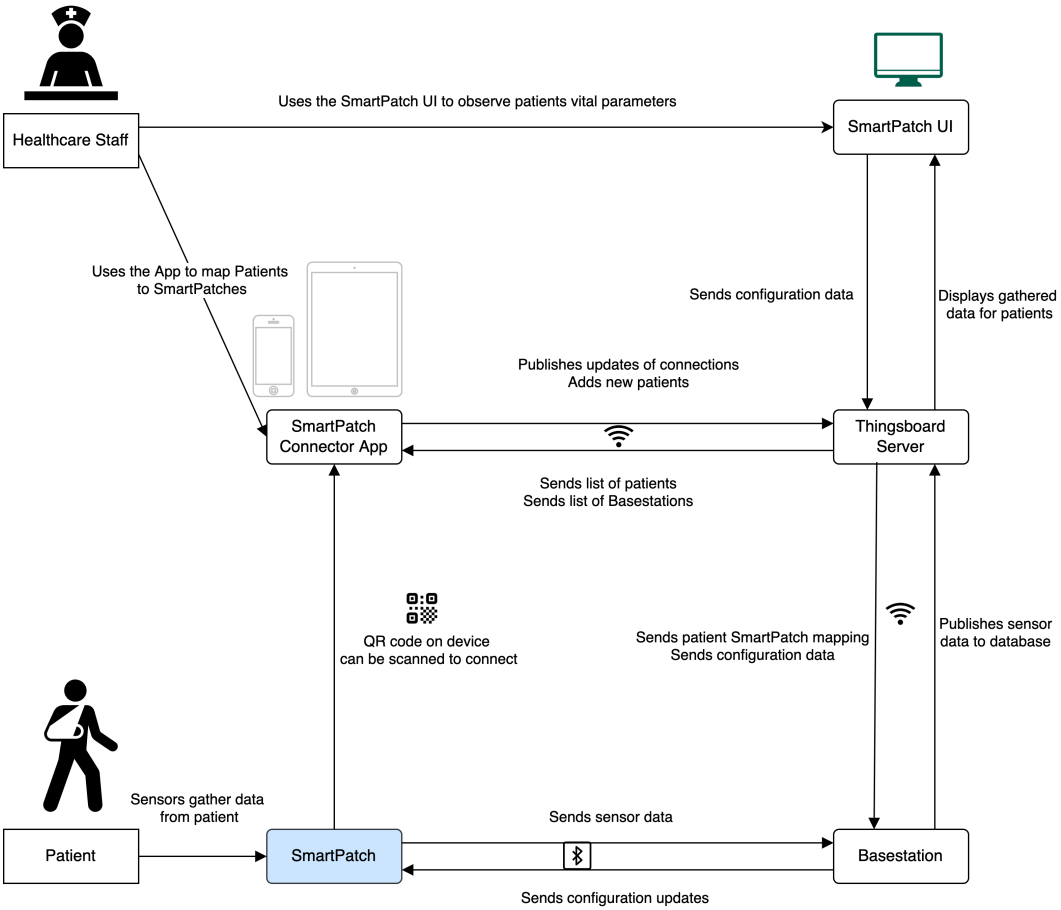


Figure 1.1.: Overview of the final SmartPatch System

Chapter 2

Background and Related Work

2.1. Background

The most important platforms and protocols that were used in the Implementation of the SmartPatch System are briefly described in this section.

2.1.1. MQTT

Message Queuing Telemetry Transport (MQTT) is a widely used messaging protocol for the Internet of Things (IoT). In Fig. 2.1 its fundamental architecture is shown. In short, MQTT is a publish/subscribe messaging protocol that is really lightweight and robust. Messages are published from an MQTT client to an MQTT broker. For each message the publishing client has to define a topic. Another MQTT client can then subscribe to a certain topic and receive all messages published to this topic.

So for example Device 1 sends a message with content "/sensordata/heartrate: 60" to its MQTT broker. The value "60" is then saved under the topic "/sensordata/heartrate" on the broker. When Device 2 was subscribed to the topic "sensordata/heartrate" when the message was published, it will now also receive the value "60".

MQTT requests can be sent using an MQTT client library. Such clients are implemented in various different programming languages.

Oftentimes MQTT brokers will also require some kind of authentication. In this project, authentication with access tokens was used.

MQTT also provides the possibility to encrypt its messages using Transport Layer Security (TLS) and to authenticate clients using several authentication protocols.

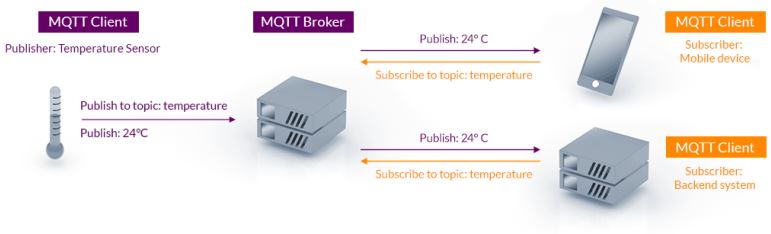


Figure 2.1.: The MQTT Architecture, [1]

2.1.2. Thingsboard IoT Platform

Thingsboard is an open-source IoT platform that can be used in the cloud or as a self hosted on-premises solution. All of Thingsboards functionality can be seen in Fig. 2.2. It acts as an MQTT Broker, provides a database, an API to interact with it and lots of additional features to support building IoT applications. Thingsboard supports various different database solutions. For this project, the standard relational PostgreSQL database was used and the in-memory option for the queue was chosen, since the are more than sufficient for the amount of data and traffic collected in this early stage of the project.

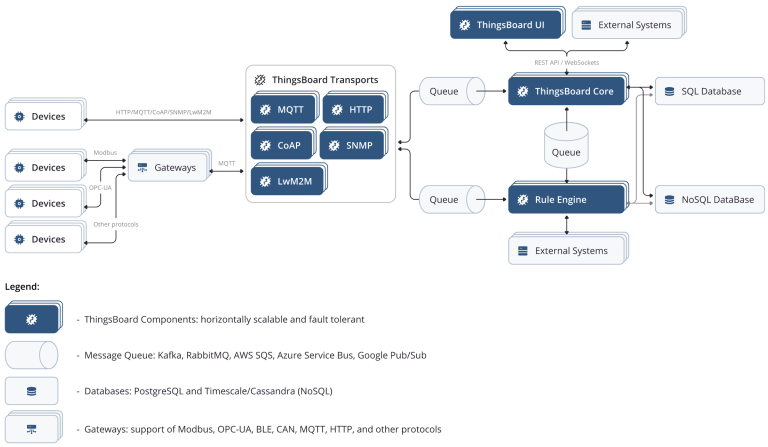


Figure 2.2.: The Thingsboard CE Architecture, [2]

2.1.3. Flutter

Flutter is an open source framework by Google. It can be used to create cross-platform applications with an emphasis on mobile applications. Using Flutter, an application needs to be written only once and can then be deployed on Android and iOs devices.

Flutter also provided useful packages for this project, mainly a client for the Thingsboard API and a package to scan QR codes.

2.2. Related Work

Not a lot of related work can be found, because this part of the SmartPatch project is a collection of different implementations for a system to accompany a health sensor.

Somewhat related, "A Web of Thing Middleware for Enabling Standard Web Access over BLE based Healthcare Wearable Device" by Bhawiyuga et al.[7] provides an oversight of different papers describing possible architectures for smart healthcare systems.

Chapter 3

Architecture

3.1. Concept

In this section an overview over the concept of the SmartPatch System is given, following the workflow when installing and using it in a medical facility. For all major steps there are visualizations that show exactly what happens throughout the whole system at this point in time. If details are left unclear, the Implementation chapter will give a deeper insight into the separate parts of the system. To begin, in Fig. 3.1 an overview of the SmartPatch System is shown.

To use a SmartPatch, a Basestation has to be in every room in the hospital where patients are using a SmartPatch. What happens in the SmartPatch System when a Basestation is powered up is shown in Fig. 3.2.

The healthcare staff then has to place a SmartPatch on a patient. They can use the app developed in this thesis on their phone to scan a QR code with the device MAC address placed on the SmartPatch. This QR code is there for convenience. In a future version using the NFC capabilities of the SmartPatch would probably be an even faster solution. What happens on the SmartPatch Connector App startup is shown in Fig. 3.3.

In the SmartPatch Connector App, they can now choose an existing patient from the Thingsboard server database or add a new Patient that is using the SmartPatch. When adding a new patient the process shown in Fig. 3.4 takes place.

The SmartPatch Connector App then publishes these updates to the Thingsboard server using MQTT. Now all Basestations receive an update with the new connection, since they are subscribed to this MQTT topic.

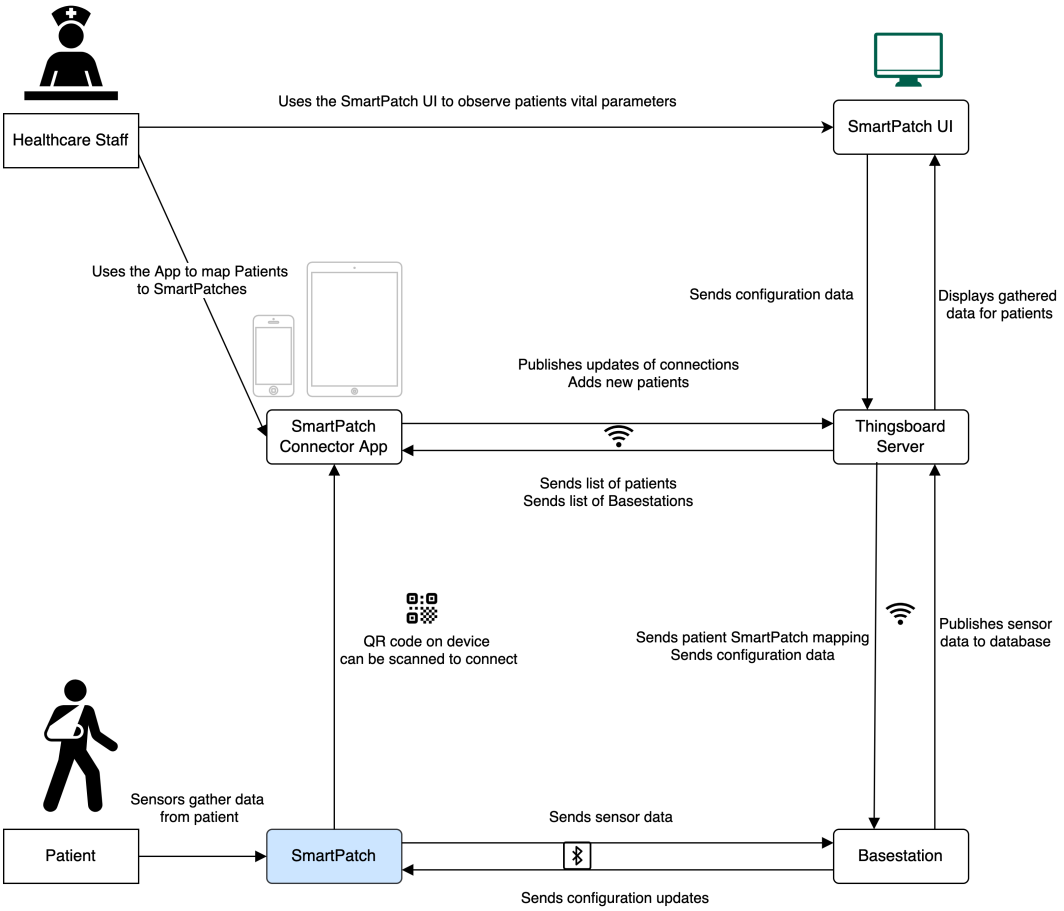


Figure 3.1.: Overview of the SmartPatch System

After receiving the update, every Basestation tries to connect to the new SmartPatch using BLE. If one of them is successful, it will receive sensor data from the SmartPatch. This connection process is shown graphically in detail in Fig. 3.5.

The received sensor data is then processed on the Basestation and afterwards published to the patients profile in the Thingsboard server database as shown in Fig. 3.6.

Also for a better understanding of the whole SmartPatch System data pipeline, it is shown in Fig. 3.7.

The healthcare staff can now observe the patients vital parameters in real-time in the SmartPatch UI. Also researchers could use the data from the Thingsboard server database using a python script provided in the Basestation Software.

When a SmartPatch has no more power(can be seen in the SmartPatch UI) or enough data was gathered, healthcare staff can remove the SmartPatch and then scan its QR code

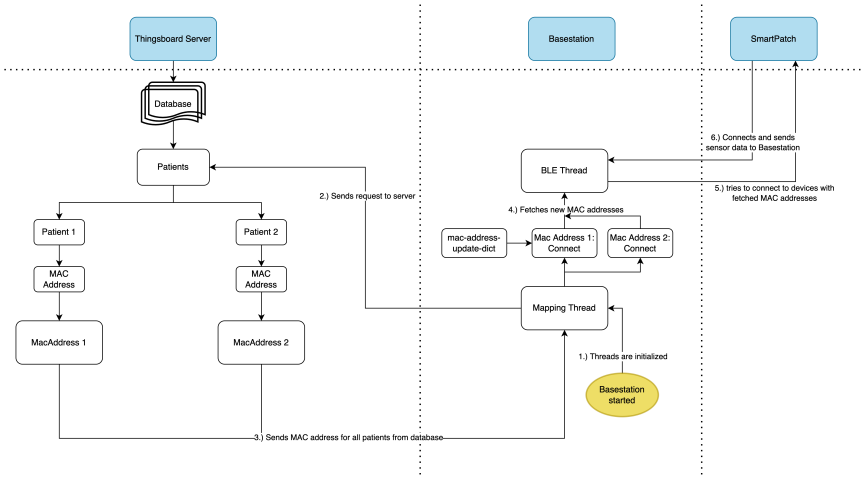


Figure 3.2.: Basestation Startup Procedure in the context of the SmartPatch System

or choose its patient name to disconnect it from a patient in the SmartPatch Connector app. This then sends an update to the Thingsboard server database, which is again received on every Basestation. This process is shown in Fig. 3.8.

The SmartPatch UI can also be used to configure a Basestation and to configure all SmartPatches currently connected to a Basestation. There are possibilities to enable or disable the logging of raw data, saving the data locally on the Basestation instead of publishing it to the Thingsboard server and more.

If a new Basestation is powered, it automatically starts the Basestation software and loads all current patient-SmartPatch connections again and tries to connect to every one of them. While this could be time consuming in large-scale use of the SmartPatch system, it is really useful for small-scale use cases since no connection is lost when losing power on the Basestation and one can also restart it without accessing its UI. For a large-scale solution, SmartPatches should also be directly mapped to a Basestation or a room, to avoid trying to connect to hundreds of devices at startup.

3.2. Hardware and Software Decisions

3.2.1. Thingsboard server

The Thingsboard server is the heart of the SmartPatch System. All communication except from the BLE connection from the Basestation to the SmartPatch is handled via the server. Thingsboard was chosen as an IoT platform for various reasons. It is open-source, brings an in-built MQTT broker and a database to save the sensor data to. It

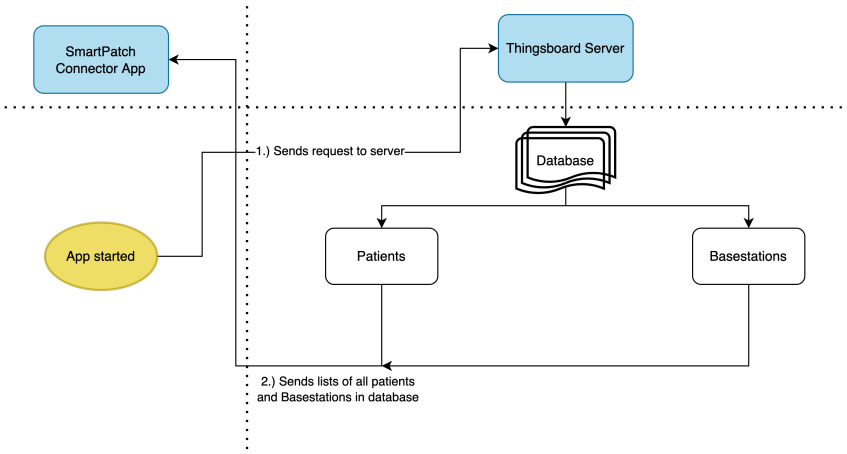


Figure 3.3.: SmartPatch Connector App Startup Procedure in the context of the SmartPatch System

also offers a lot of functionality to use in its Thingsboard API, and a possibility to easily build a User Interface.

Therefore Thingsboard was the perfect choice for the heart of the SmartPatch System. At first, a self-built solution was considered, but a Thingsboard Community Edition server has all the possibilities one would have to develop already built in.

Thingsboard servers can be hosted on lots of devices. In the test setup, the server is hosted on a Raspberry Pi 4B but it could easily be hosted on a more powerful platform.

3.2.2. Basestation

At the moment, a Raspberry Pi 4B is used as Basestation device. It was chosen because of its small form factor and its vast capabilities. But the Basestation software could also be ran on any device capable of running Python scripts and handling the used BLE dongle.

Python was chosen as programming language, since it entails a lot of crucial libraries for the Software, including a client for the Thingsboard API.

3.2.3. SmartPatch Connector App

The SmartPatch Connector app uses the Flutter framework, a cross-platform application framework. It was used because of its various different deployment possibilities. Apps can be deployed without any changes to Android and iOs, but also as web applications.

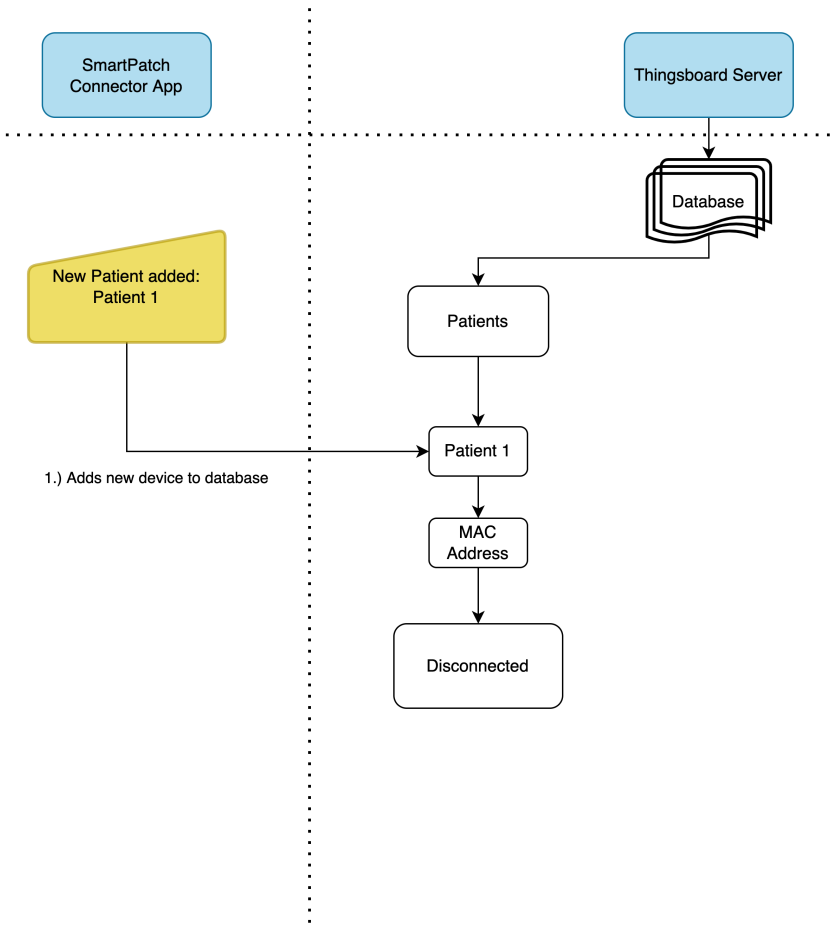


Figure 3.4.: Adding a new Patient in the SmartPatch Connector App in the context of the SmartPatch System

Furthermore, a client for the Thingsboard API exists in dart, the programming language used to develop flutter apps.

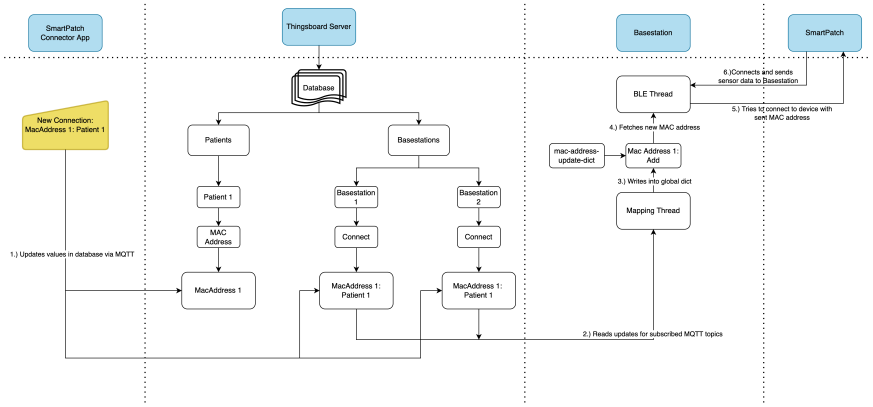


Figure 3.5.: Connecting a patient to a SmartPatch in the context of the SmartPatch System

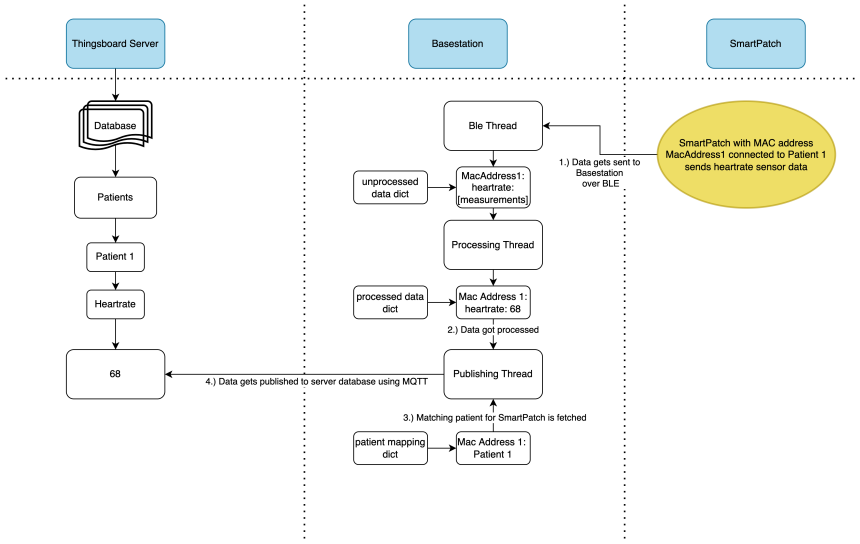


Figure 3.6.: Patient data publishing from SmartPatch in the context of the SmartPatch System

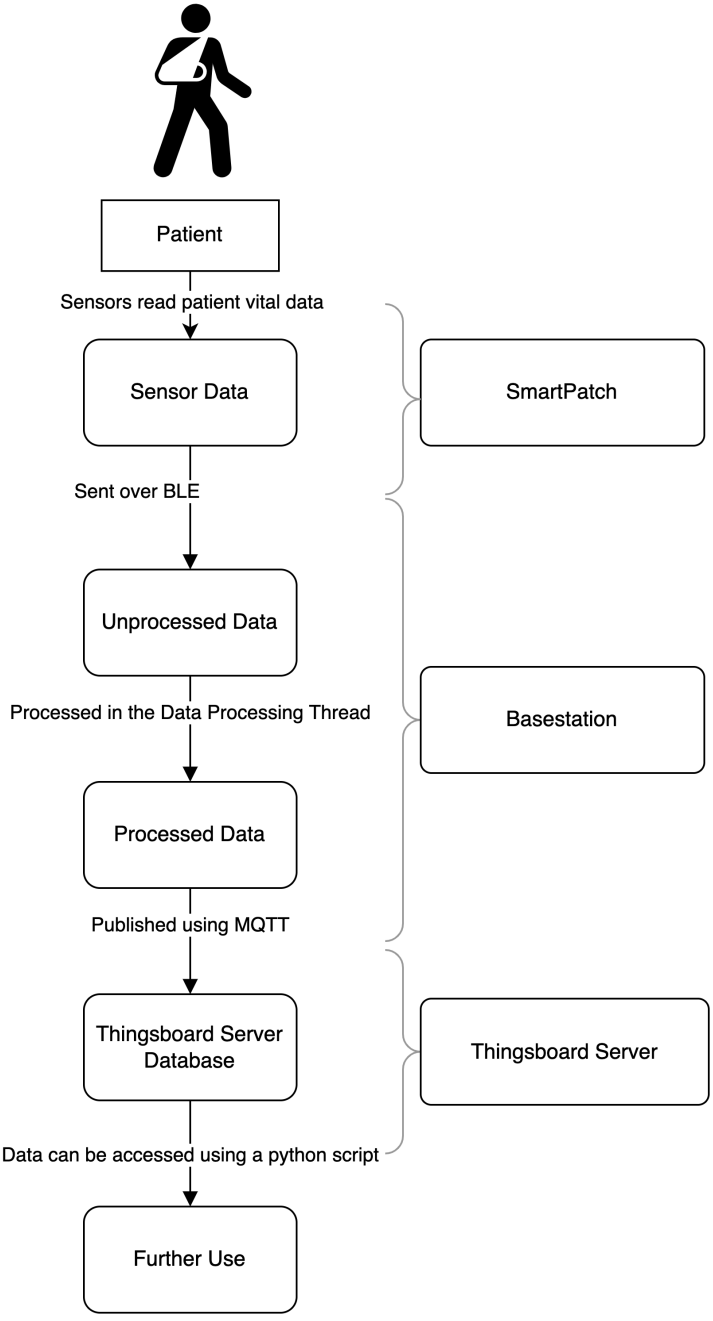


Figure 3.7.: Overview of the SmartPatch System Data Pipeline

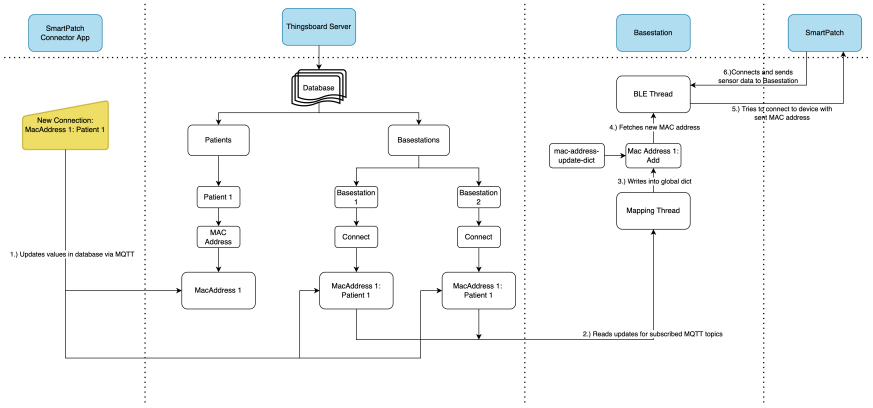


Figure 3.8.: Disconnecting a SmartPatch from a Patient in the SmartPatch Connector App in the context of the SmartPatch System

Chapter 4

Implementation

4.1. SmartPatch UI

The SmartPatch UI was built using the Thingsboard platform. The UI is a Thingsboard dashboard, Thingsboards possibility to make custom user interfaces for customers to visualize their data. This SmartPatch UI dashboard is built from different dashboard states or pages which will be explained in detail below. Every page consists of widgets, that show data from the Thingsboard server database. In the SmartPatch UI those widgets are mostly used to display SmartPatch sensor data for a patient, but also to generate lists of all patients and Basestations that are in the database or all SmartPatches that are currently connected.

On the Thingsboard platform, devices can be added to display and save telemetry data and attributes for them. Those devices need to have a device type. The two most used device types in the implementation of the SmartPatch UI are shown in Fig. 4.1 and Fig. 4.2. Because, for the SmartPatch System, the data needs to be associated with hospital patients and not the actual SmartPatches, a workaround was applied. All patients are saved as devices of type Patient. With this trick, data can be saved to its actual patient in the database. The SmartPatches and Patients can then be mapped to each other using the SmartPatch Connector App.

4.1.1. Home Page

In Fig. 4.3, the home page of the SmartPatch UI is shown. This page will be shown when a customer logs in with their Thingsboard account. It displays all patients that are currently in the database. A patients data can be accessed with a double click, changing the dashboard state to the SmartPatch Dashboard of the selected patient. The homepage also displays an overview of all currently connected SmartPatches and links to the Admin

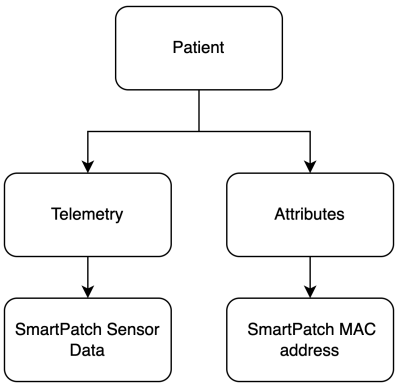


Figure 4.1.: Device Type Patient in Thingsboard Database

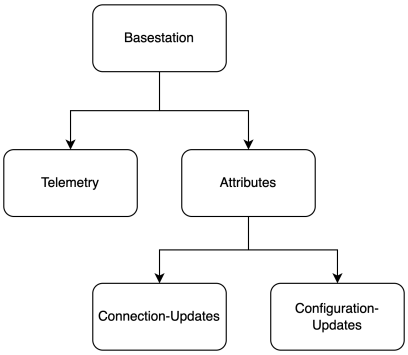


Figure 4.2.: Device Type Basestation in Thingsboard Database

Dashboard state and provides a link for a future integration of the Indoor Localization System.

4.1.2. SmartPatch Dashboard

In Fig. 4.4 and Fig. 4.5, a SmartPatch Data Dashboard, displaying simulated sensor data, is shown. It displays all the latest values from the database for the selected patient. Furthermore it displays all sensor data over time in graphs. The displayed time window of these figures can be adjusted from within the dashboard. Using the displayed last connection time, one can check at a quick glance, if the SmartPatch of the patient is still active and sending sensor data. The displayed data is telemetry data that is sent from a Basestation to the selected Patient.

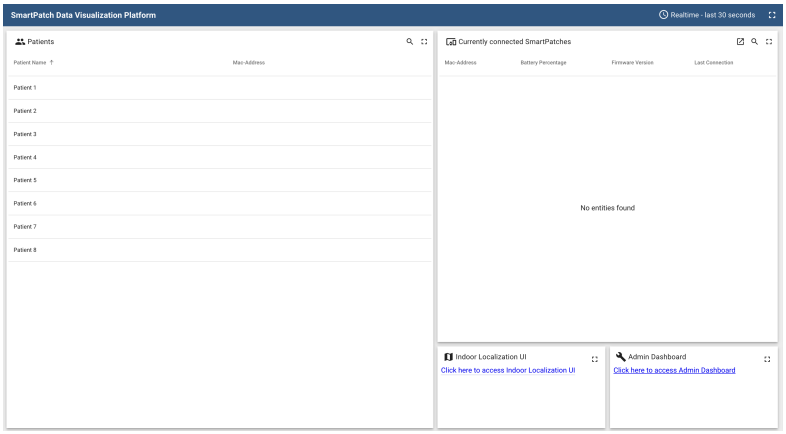


Figure 4.3.: The SmartPatch UI Home Page

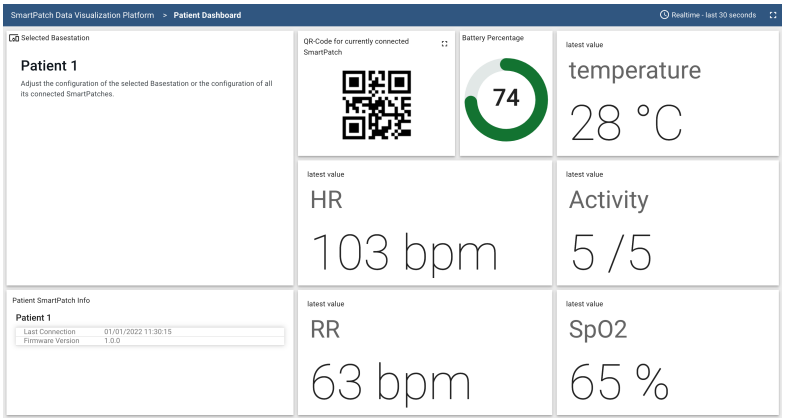


Figure 4.4.: The SmartPatch UI Data Page 1/2

4.1.3. Admin Dashboard

The Admin Dashboard shown in Fig. 4.6 and Fig. 4.7 is not intended to be used by customers. It is, as the name suggests, only for the ones maintaining the SmartPatch System. It provides the possibility to select a Basestation and then change its configuration. It is also possible to change the configuration for all SmartPatches connected to a selected Basestation. This is achieved using a Thingsboard feature called attributes. Each Basestation is subscribed using MQTT to its configuration and update attributes and therefore updates them when they are changed in the UI. SmartPatch configuration updates can be updated instantaneously, but Basestation configuration updates are only applied once the Basestation is restarted, because the Basestation uses its configuration on startup.

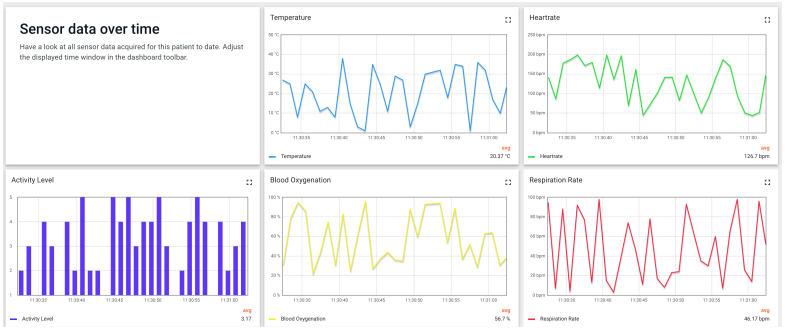


Figure 4.5.: The SmartPatch UI Data Page 2/2

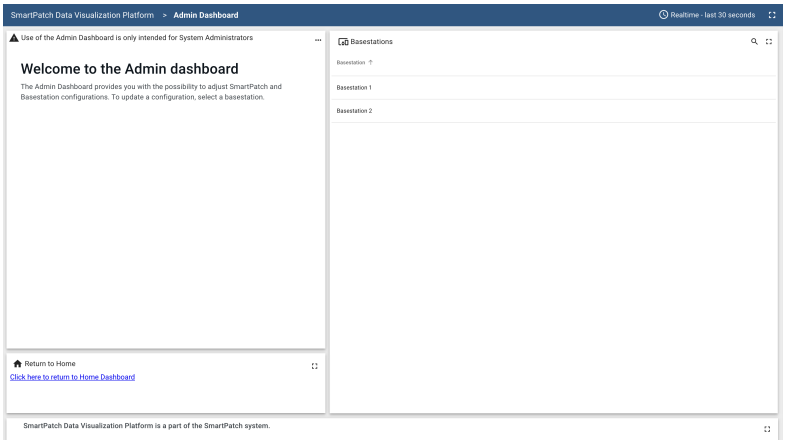


Figure 4.6.: The SmartPatch UI Admin Page

4.2. SmartPatch Basestation Software

The SmartPatch Basestation Software is a Software package implemented in Python. It uses the Thingsboard API, MQTT and BLE to connect to SmartPatches and upload their sensor data to a Thingsboard server database.

4.2.1. Initialization

The SmartPatch Basestation Software starts on startup of the Basestation device, if installed correctly. A terminal window opens automatically on startup of the Basestation and in this window the command line output of the Software can be seen for debugging purposes. Then the Software starts initializing, as shown in 4.8.

At first, the program connects to your Thingsboard server and tries to fetch configuration data and an already existing mapping of patients to SmartPatches. Then it starts

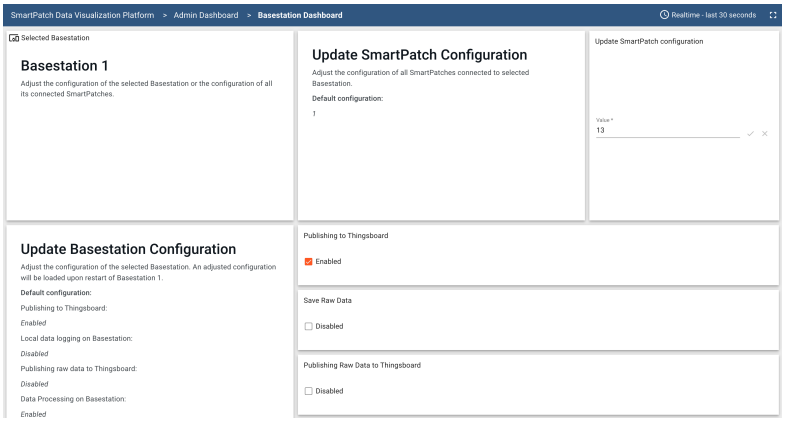


Figure 4.7.: The SmartPatch UI Basestation Page

building the Basestation part of the data pipeline shown in Fig. 4.9. Upon finishing the initialization, it waits for updates or new data forever.

4.2.2. Threads

The Basestation Software consists of several threads working simultaneously. To avoid conflicts, all global data structures are accessed with thread locks.

Mapping Thread

The first thread to be started is the Mapping Thread, shown graphically in Fig. 4.10. When initialized, this thread fetches a list of all devices of type Patient from the Thingsboard server database. Additionally, it fetches the MAC address attribute for each patient. This data is then written to a global dict, excluding all the patients that have "disconnected" as their MAC address. This global dict is used to map the devices to their patients. The MAC addresses are also written to another global dict with the MAC addresses as keys and "add" as value. This dict is an update dict that is used by the BLE thread to connect to devices. At Basestation startup all device-patient-pairings are added to the update dict to ensure that no connection is lost.

After that initial mapping, the Basestation subscribes to different attributes of its according Basestation device in the Thingsboard database using MQTT. These are used for connection updates and configuration updates, sent from the SmartPatch Connector App or the SmartPatch UI.

Then, the main loop of the mapping thread starts, where it waits for new updates of these attributes. Depending on the updated attribute, a different callback is called. Basestation configuration updates will be displayed, but will only have an effect after

restarting the Basestation Software. SmartPatch configuration updates on the other hand will be passed on immediately to all SmartPatches connected to a Basestation. Connection updates stemming from the SmartPatch Connector app are the main source of updates. These updates will add or delete entries in the patient-SmartPatch mapping dict and also in the update dict accordingly.

Ble Thread

The Ble Thread receives sensor data from SmartPatches. For more information on this thread, refer to Andreas Hunziker's part[5] of the SmartPatch project.

Processing Thread

The Processing Thread, shown in Fig. 4.11, fetches unprocessed data, processes it and then saves it as processed data. For more information on this thread, refer to Amane Zürrer's part[4] of the SmartPatch project.

Publishing Thread

The Publishing Thread, shown graphically in Fig. 4.12, takes data out of the processed data data structure and publishes it to a Thingsboard server database.

It therefore checks for available data and waits to check again after a configured amount of time if no data was found. It then iterates through this data and exchanges the MAC address from the SmartPatch that sent it, with the patient name that was mapped to this SmartPatch. Then it publishes all data for this patient name, that is at the same time the MQTT access token, that is needed to publish to this patient in the Thingsboard server database.

When this process is finished for all data that was fetched, the loop starts anew.

This architecture is not the most efficient, since every MQTT reconnect is costly. Therefore the architecture could be further speed up by keeping the MQTT connections alive for as long as possible. But since no faster architecture was needed for the current system's throughput requirements, performance improvements were not the main focus of this version of the Basestation Software.

Raw Data Thread

The Raw Data Thread, shown Fig. 4.13, is not enabled by default. If it is enabled, it first checks the rest of the configuration. If publishing the raw data is enabled, the same procedure as described for the Publishing Thread will take place. The only differences are, that the data will be fetched from the unprocessed data data structure and will be published to a device of type Raw with patient name + " Raw Data" as its device name. This is to strictly separate the raw data from the processed data, when saving it to the database. If publishing the raw data is disabled, all unprocessed data will just be logged to a local file on the Basestation device.

In Fig. 4.14 a visualization of the augmented data pipeline with the Raw Data Thread included is given.

This thread is only an experimental implementation, designed to show that logging the raw SmartPatch data is possible. Since the thread just periodically logs the unprocessed data, it highly depends on being in sync with the Data Processing Thread. If a future version of the SmartPatch System would have raw data logging as a main goal, an extra step in the data pipeline should be inserted. The SmartPatch sensor data would be saved in a raw data data structure from the Ble Thread, then published as raw data and written into the unprocessed data dictionary by the Raw Data Thread.

4.2.3. Basestation Settings

In Table 4.1 an overview over all important Basestation settings is given. Those that are not mentioned are rather self-explanatory. All described settings can be found in the Basestation Software Package in the Settings module.

4.2.4. Basestation Global Variables

In Table 4.2 all global variables used for the Basestation software are explained. They were introduced to exchange data between threads. All global variables ore only altered using their matching global thread lock.

4.3. SmartPatch Connector App

The SmartPatch Connector App was developed to make it possible to map SmartPatches dynamically to patients without altering any code. It was implemented using the flutter framework.

4.3.1. Concept

The basic concept of the Connector App is, that the attribute MAC address for a patient in a Thingsboard server database gets changed using the Thingsboard API. Further an update attribute is published to every Basestation device in the Thingsboard server database. The Basestations are subscribed to these updates and try to connect or disconnect the devices that are mentioned. This indirect path of communication was chosen, so that the Basestations do not have to be connected to the internet, for security reasons.

4.3.2. Pages

The app consists of several different pages which are shown and explained in detail in this chapter.

Home Page

The Home Page shown in Fig. 4.15 is the first page one can see when starting the app. It links to the different functionalities of the app. One can navigate to the settings page to adjust the app settings. One can also access the SmartPatch UI in your mobile browser shown in Fig. 4.16. Most importantly one can navigate to the Connect and Disconnect Pages. If one navigates to the latter two pages, at first, a list of all patients that are currently in your Thingsboard server database gets fetched. This is implemented using the Thingsboard API to fetch all devices of the type patient from the Thingsboard server database. Furthermore on every button press on the Home Page, all settings are updated, to ensure connecting to the desired Thingsboard server.

Connect Page

In the Connect Page shown in Fig. 4.17, one can select a SmartPatch MAC address by scanning a QR code. This process is shown in Fig. 4.18. The QR code scanner was tested on both iOs and Android, and uses a flutter package that already implements the QR code scanner. This method of entering a SmartPatch MAC address was chosen, since it is faster and a lot less tedious than entering it by hand, but also easy to implement and the QR code can just be attached to the SmartPatch.

After one has successfully scanned a SmartPatch MAC address, one has to select a patient from a drop down list of all existing patients that was fetched upon accessing the page. If the patient to map to the SmartPatch is not already in the database, there exists a possibility to add him directly from here with a button, navigating to the Add New Patient page.

When a MAC address and a patient where selected, one can press the connect button to publish this update. The MAC address patient pair gets uploaded to the Attribute Connect as a JSON object on all Basestation devices in the Thingsboard database. This is what triggers the instant updates on the Basestations. Furthermore the selected patient in the Thingsboard database also gets its MAC address attribute updated to the selected MAC address. This is done to display the connections in the SmartPatch UI and to ensure that a Basestation can fetch all existing connections on startup. Also, on press of the connect button, all previous connections for this SmartPatch are deleted from the Thingsboard database, to make sure that the data is not sent to multiple patients by accident.

Disconnect Page

On the Disconnect Page shown in Fig. 4.19, one can disconnect a SmartPatch from a patient. One can either select a SmartPatch using the in-built QR code scanner, or choosing a patient from the previously fetched patient list. If both are selected, the patient name will be prioritized.

On the press of the disconnect Button, all Basestations receive an update to the Attribute Disconnected, with the patient name and MAC address pair as its value. Further, the patients MAC address attribute is changed to disconnected.

Add New Patient Page

The Add New Patient Page shown in Fig. 4.20 is the only way, besides a python script in the Basestation Software package, to automatically provide new patients to the Thingsboard database in the right manner, so that they can be used with the SmartPatch System.

The page is simple, providing the possibility to enter a patient name and adding it to the database on a button press. When pressing the add new patient Button, two new devices are added to the Thingsboard database. One device from type Patient, with the patient name as device name and one device from type Raw with patient name + " Raw Data" as device name. While the first device is the patient entry used for the SmartPatch UI, the second device is solely created to hold this patients raw data, if it is saved to the database. The device from type Patient also gets its MAC address set to "disconnected". This is done because because querying empty attributes could lead to problems.

All Thingsboard devices need an access token to publish to them using MQTT or the Thingsboard API. Both devices that are created have their device name as their access token. This is done for convenience, because one does not have to worry about the access tokens anymore. Of course this approach could pose a security threat, but it could be solved by encoding the patient name with a predefined function and using this encoding

as the access token. But since security was not the most important concern for this first version of the SmartPatch System, this was not implemented yet.

Success Page

After executing a connect or disconnect without an error, the app navigates to the Success Page shown in Fig. 4.21. It gives a visual confirmation that the request was sent to the Thingsboard server and provides the possibility to return to the Home Page.

Settings Page

In the Settings page shown in Fig. 4.22 one can adjust the app settings for the Thingsboard server URL, username and password, and a device maximum. While the first setting is really important, since it is the only thing that prevents us from having to hard code the Thingsboard server URL for every new Thingsboard Server, the other settings could remain hard coded and be deactivated for customer use. The Thingsboard server URL setting is therefore also the only setting field that is required, in order to apply the settings. Upon application of the settings, they are saved as shared preferences. Shared preferences are a possibility provided in flutter, to save key-value-pairs without explicitly using a database.

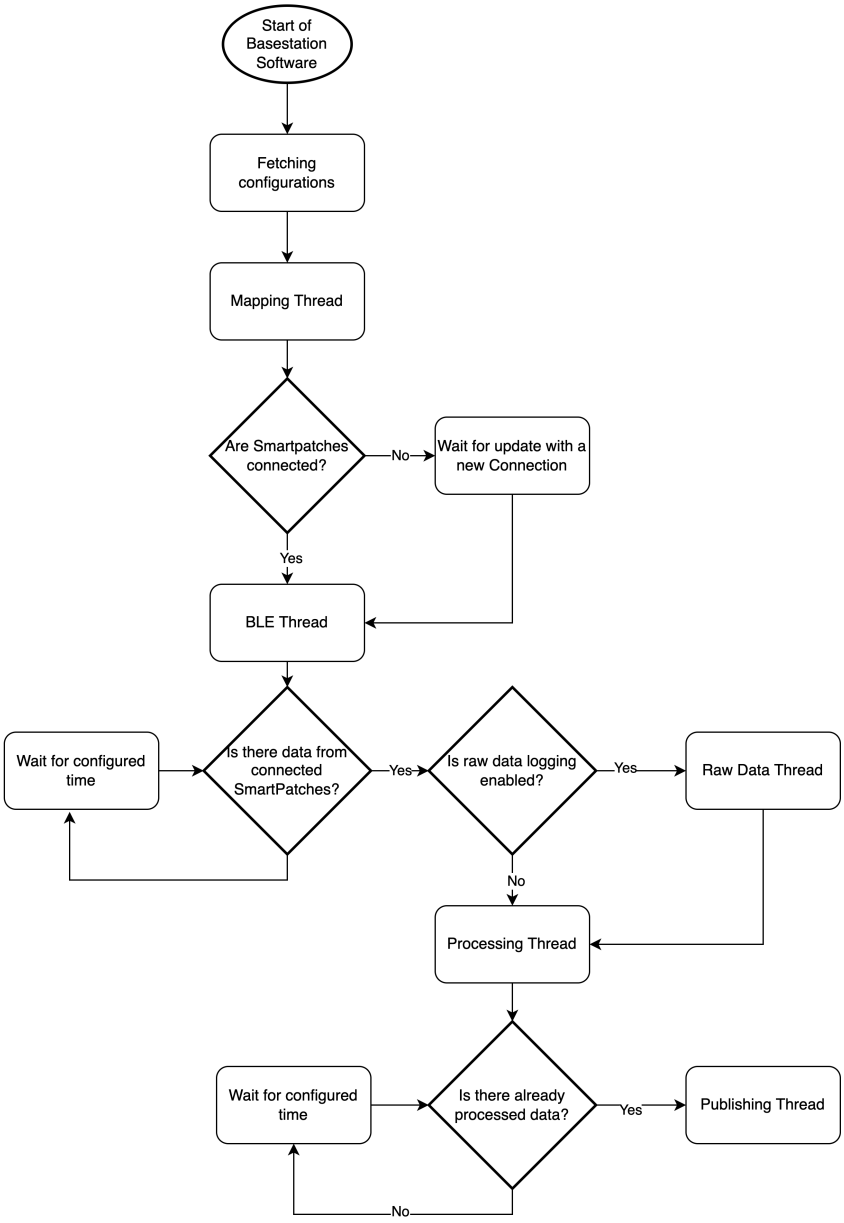


Figure 4.8.: Initialization Flowchart of Basestation Software

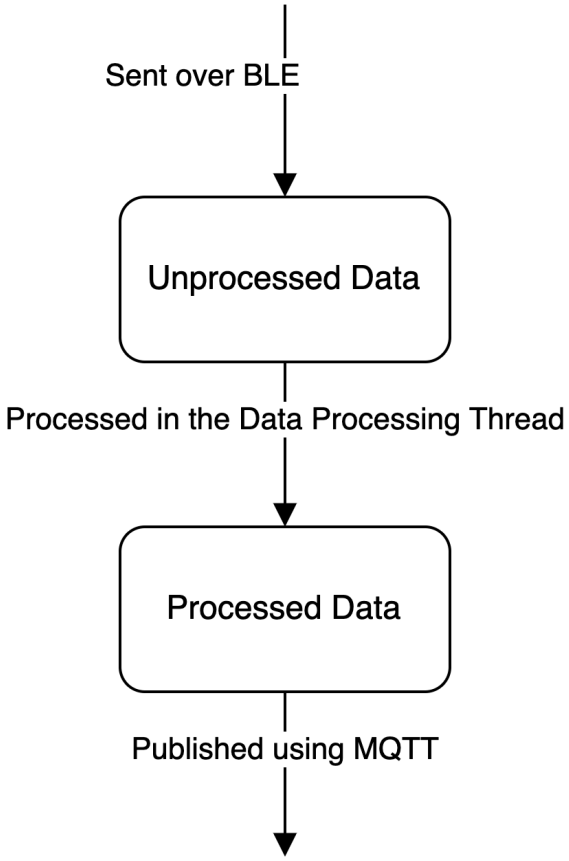


Figure 4.9.: Basestation Data Pipeline

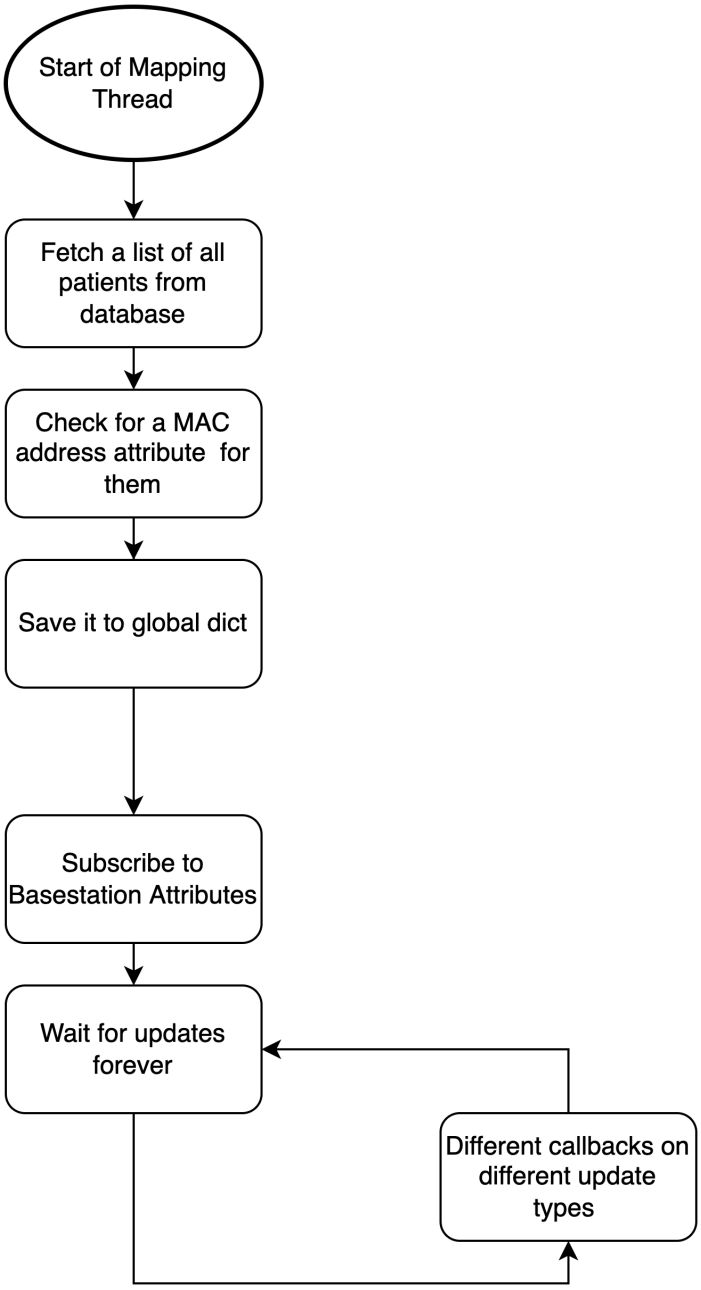


Figure 4.10.: Mapping Thread Flowchart

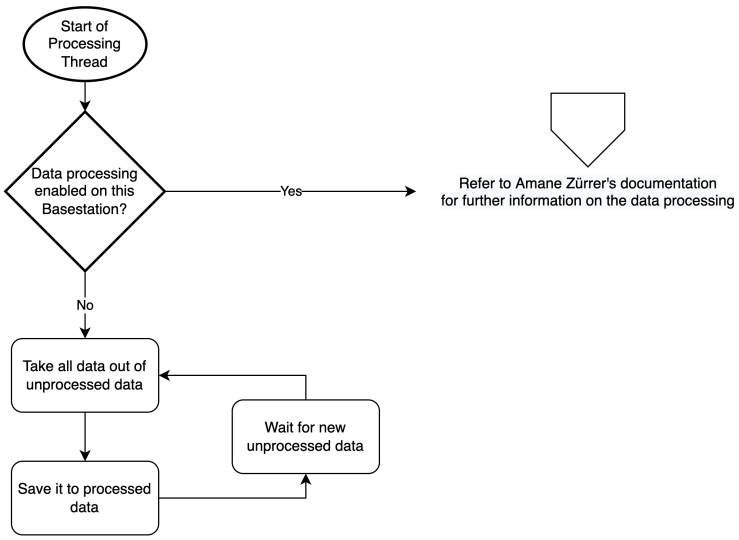


Figure 4.11.: Processing Thread Flowchart

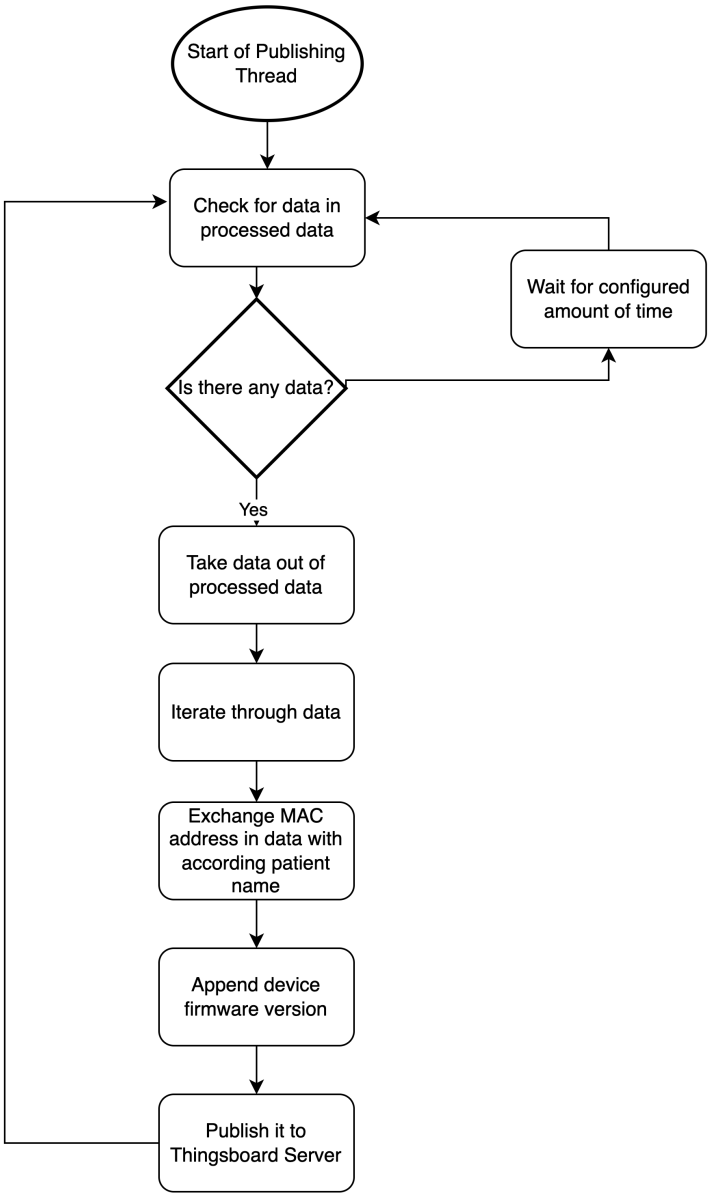


Figure 4.12.: Publishing Thread Flowchart

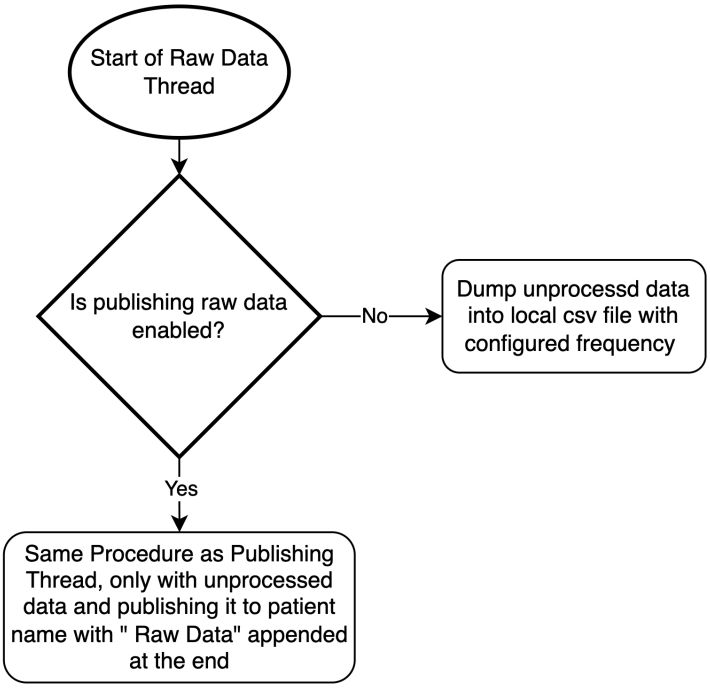


Figure 4.13.: Raw Data Thread Flowchart

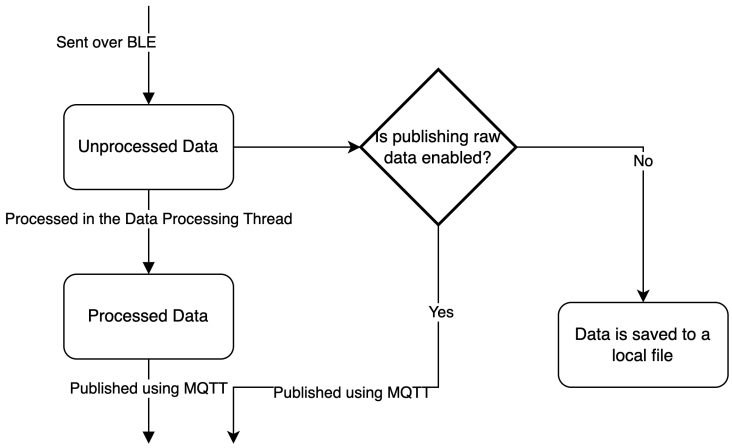


Figure 4.14.: Basestation Data Pipeline with Raw Data Thread

Name	Function	Configurable in Thingsboard UI
Software Version	Software version of Basestation, change on update.	No
Basestation Name	Name of this Basestation in Thingsboard server database. *Basestation has to be provisioned to database first before use.*	No
Broker	Thingsboard server IP address. *Has to be configured before use.*	No
Port	Thingsboard server data listening port.	No
Username	Thingsboard server username.	No
Password	Thingsboard server password.	No
Max Patients	Maximum count of patients handled by Basestation.	No
Device Maximum	Maximum count of connected SmartPatches.	No
Update Attributes List	Attributes of Basestation device in Thingsboard database that are subscribed to using MQTT.	No
Basestation Config Params List	List of all Basestation configuration parameters.	No
Process Data	If enabled, data will be processed on Basestation.	Yes
Publish To Thingsboard	If enabled, processed data will be published to Thingsboard server database.	Yes
Local Data Logging	If enabled, processed data will be saved locally on Basestation device. Automatically disabled when Publish to Thingsboard is enabled.	Yes
Save Raw Data	If enabled, unprocessed data will be saved locally on Basestation. Only works if processed data is either published or saved locally as well.	Yes
Publish Raw Data	If enabled, unprocessed data will be saved published to Thingsboard server database. Only works if processed data is either published or saved locally as well.	Yes

Table 4.1.: Important Basestation Settings

Name	Function
connected_devices	All devices currently connected to Basestation over BLE.
mac_address_update	Dict with all connection updates from Thingsboard server. SmartPatch MAC addresses as keys and "add" or "remove" as values.
patient_mapping	Dict with current mapping of SmartPatches to patients. SmartPatch MAC addresses as keys and patient names as values.
smartpatch_config_update	Integer with current updated SmartPatch configuration.
unprocessed_data	Nested dict structure of data produced in BLE Thread. For exact data structure consult code documentation.
processed_data	Nested dict structure of data produced in Processing Thread. For exact data structure consult code documentation.

Table 4.2.: Overview of global variables on the Basestation

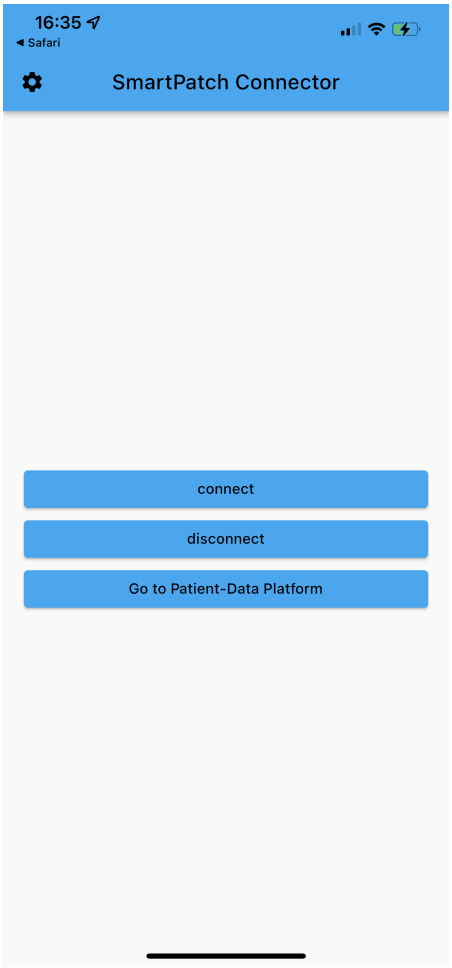


Figure 4.15.: The SmartPatch Connector App Home Page

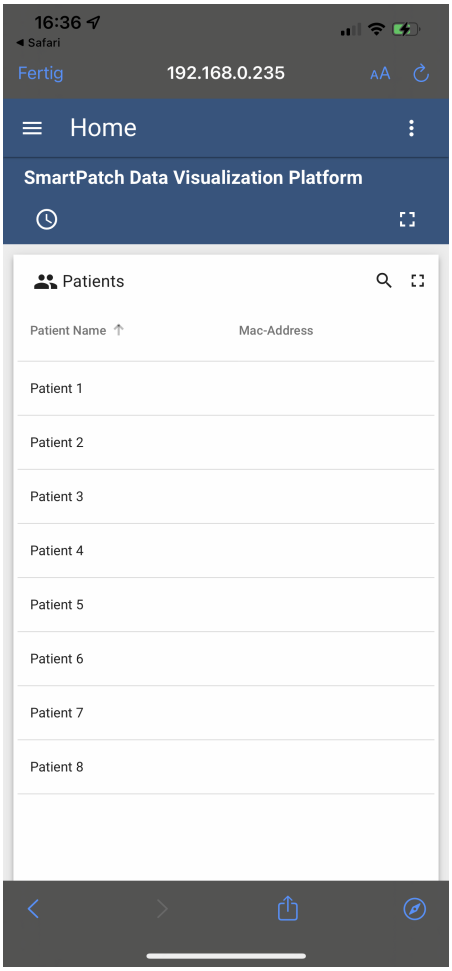


Figure 4.16.: The mobile version of the SmartPatch UI accessed using the SmartPatch Connector App

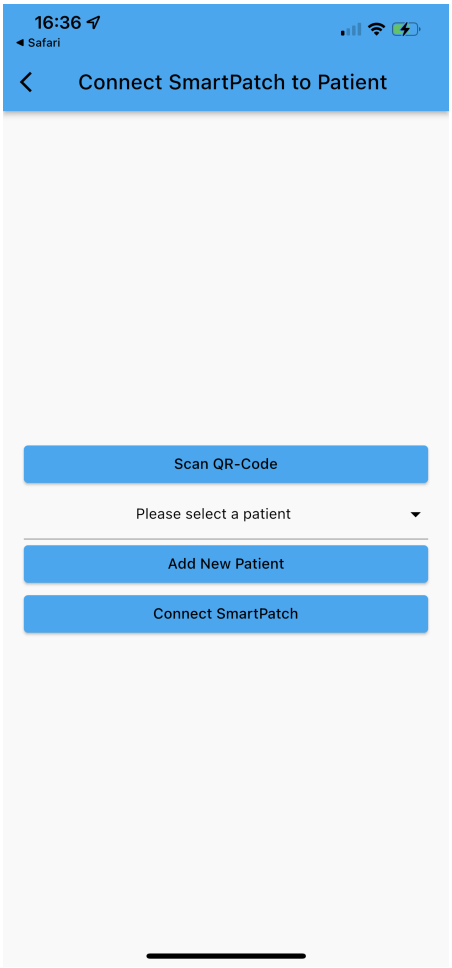


Figure 4.17.: The SmartPatch Connector App Connect Page

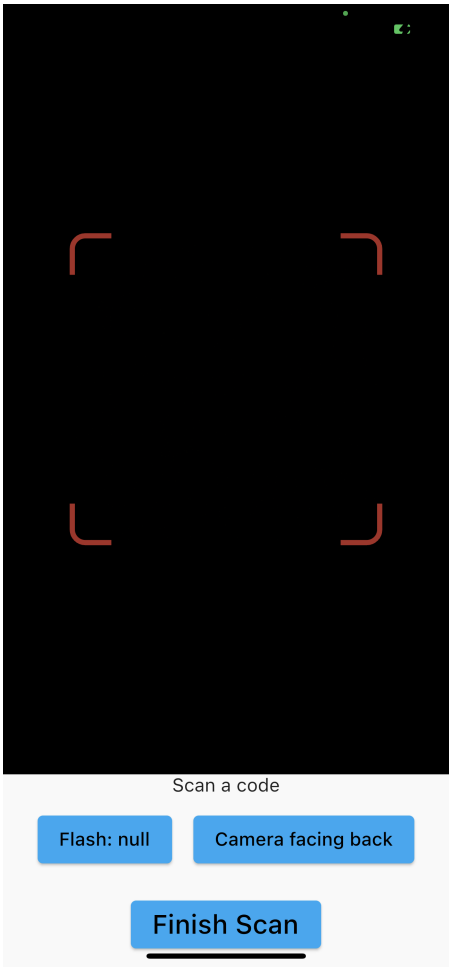


Figure 4.18.: The in-app QR code scanner

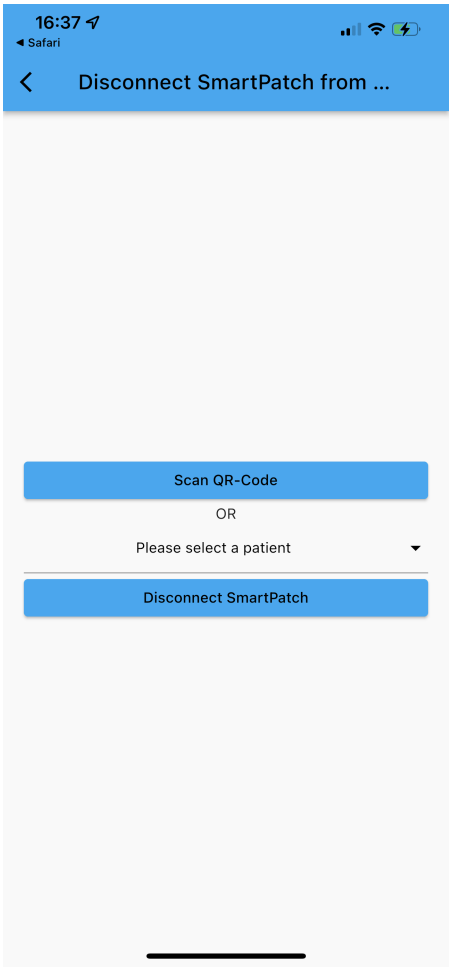


Figure 4.19.: The SmartPatch Connector App Disconnect Page

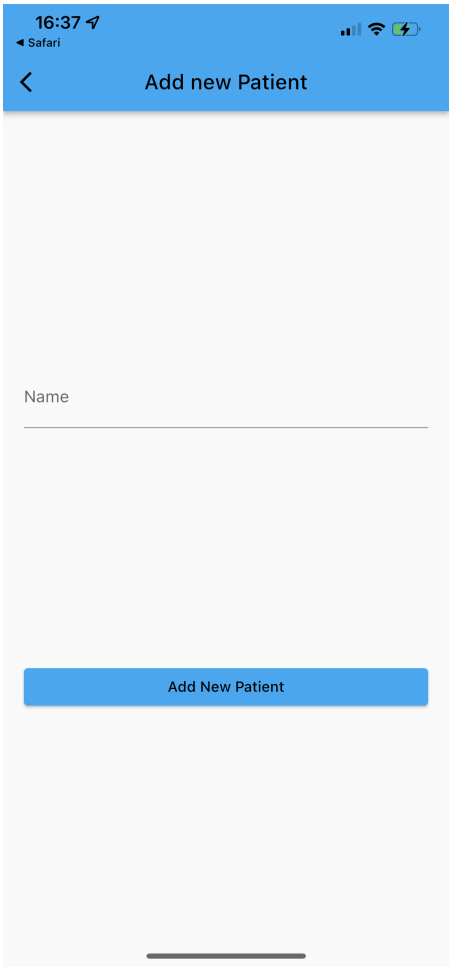


Figure 4.20.: The SmartPatch Connector App Add New Patient Page

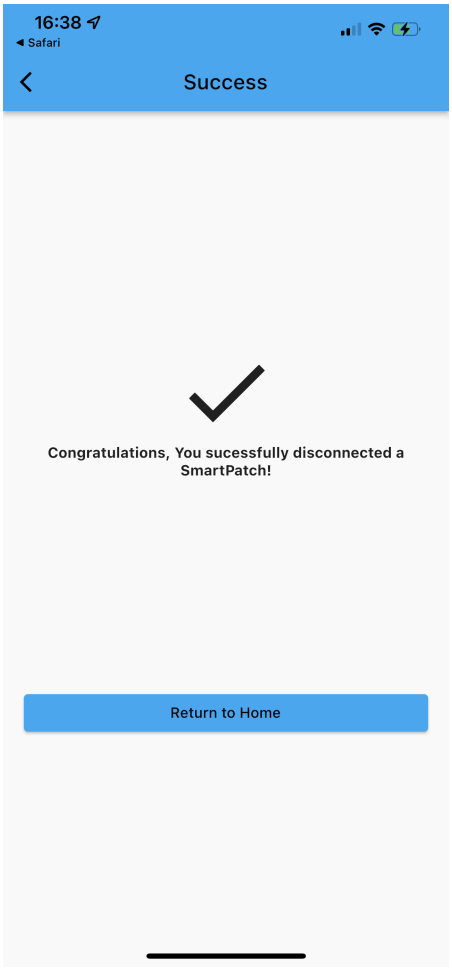


Figure 4.21.: The SmartPatch Connector App Success Page

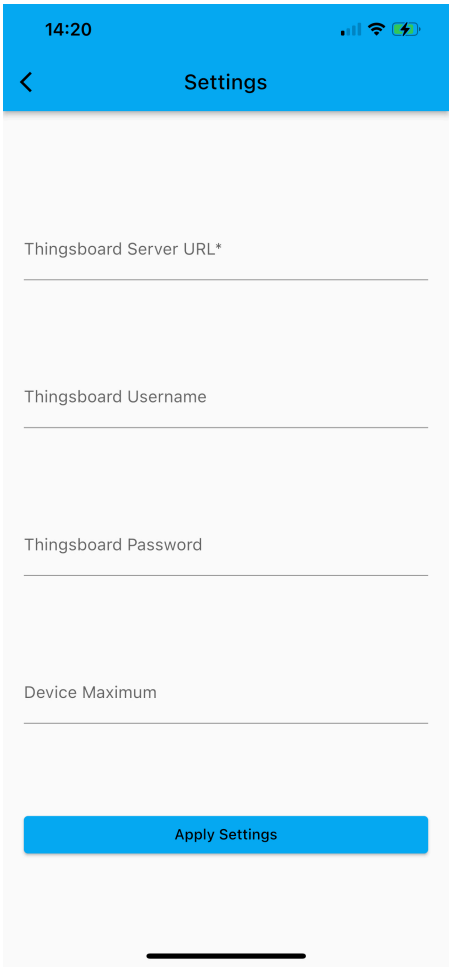


Figure 4.22.: The SmartPatch Connector App Settings Page

Chapter 5

Results and Discussion

5.1. Results

All tests were performed using a Raspberry Pi 4B, running Raspbian Buster, as Basestation Device. The Thingsboard server was either hosted on the same Raspberry Pi 4B as the Basestation Software or a different Raspberry Pi 4B running Raspbian Buster. Various mobile phones were used to run the SmartPatch Connector App. Basestations, Server and mobile phone were all connected to the same wireless network during the tests. In the tests that used a BLE dongle, a "EDIMAX BT-8500" [8] was used.

5.1.1. SmartPatch Tests

For each SmartPatch test, all SmartPatches were added to the SmartPatch System using the SmartPatch Connector App. For all SmartPatches, publishing the sensor data to a Thingsboard server was enabled. For all tests, successful transmission of data to the Thingsboard server for the chosen duration was counted as pass.

Multiple Tests were made on the final SmartPatch System as well as in earlier stages of the development. For the tests shown in Table 5.1, the final version of the SmartPatch, coupled with the final version of the SmartPatch System was used.

For the tests shown in Table 5.2, simulated SmartPatches on "nrf52 development kits" [9] were used. These boards sent random already processed measurements of health data over BLE. Therefore the data processing on the Basestation was disabled during the tests. Duration for all tests in Table 5.2 was 5 minutes.

Test Setup	Duration	Result
1 SmartPatch connected to SmartPatch System using BLE dongle	1 hours	Pass
2 SmartPatches connected to SmartPatch System using BLE dongle	1 hour	Pass
1 SmartPatch connected to SmartPatch System	6 hours	Pass
2 SmartPatches connected to SmartPatch System	1 hour	Pass
1 SmartPatches connected to SmartPatch System also publishing raw sensor data	10 min	Pass
2 SmartPatches connected to SmartPatch System also publishing raw sensor data	10 min	Pass

Table 5.1.: Tests with final SmartPatch Version

Test Setup	Result
1 SmartPatch connected to SmartPatch System	Pass
2 SmartPatches connected to SmartPatch System	Pass
3 SmartPatches connected to SmartPatch System	Pass
4 SmartPatches connected to SmartPatch System	Pass
5 SmartPatches connected to SmartPatch System	Pass
6 SmartPatches connected to SmartPatch System	Pass

Table 5.2.: Earlier tests with simulated SmartPatches

5.1.2. Basestation Tests

For the tests shown in Table 5.3, Basestations with the final version of the Basestation Software were used, connected to a Thingsboard server with the final SmartPatch UI installed.

The ability to configure the connected Basestations from the Thingsboard server and see those updates in the output of the Basestations was interpreted as a passed test. Failure to send those updates and therefore a failed connection would result in a failed test.

Test Setup	Result
1 Basestation connected to SmartPatch System	Pass
2 Basestations connected to SmartPatch System	Pass

Table 5.3.: Tests with multiple Basestations

5.2. Discussion

5.2.1. SmartPatch Tests

All tests with varying numbers of SmartPatches succeeded. While this is a promising result, the theoretical limit of ten SmartPatches connected to a Basestation could not be tested, due to a lack of SmartPatches for testing. This is also the reason why only tests with up to two SmartPatches were done with the real final SmartPatch version.

While only publishing processed data should not pose a problem as shown with the simulated SmartPatches in Table 5.2, also publishing raw sensor data for more than two SmartPatches could lead to performance issues. This assumption stems from the fact that while the test with two SmartPatches also publishing raw data passed, some performance decreases in the live data display on the SmartPatch UI could already be seen.

5.2.2. Basestation Tests

There were only a few tests with multiple Basestations but they proved sufficiently that the Systems is also functional with multiple Basestations. However, a load test with a large number of Basestations was impossible, due to a lack of devices.

Chapter 6

Conclusion and Future Work

6.1. Conclusions

A working version of the SmartPatch System was implemented. It provides possibilities to easily map SmartPatches to patients and to gather and display their data. Basestations can theoretically handle up to ten SmartPatches at once and tests for up to six SmartPatches at once have passed.

With the Thingsboard platform, a fitting IoT platform was chosen that yields possibilities to scale the system or adapt it in the future. All SmartPatch and patient data is visualized in the SmartPatch UI, in real-time and can be accessed with a python script.

The System is now ready for in depth in field testing and use.

6.2. Future Work

At this point there exists a first working version of the SmartPatch System. Even though it is sufficient for small-scale tests, for large-scale testing there could and should be some adjustments made.

6.2.1. Security

The topic of security was not handled in this first rendition of the SmartPatch System. Therefore a future project should certainly look into it. For example, there exists the possibility, also implemented by the Thingsboard platform, to use TLS when using MQTT. This would be a first step towards a more secure system.

6.2.2. Deployment

The whole SmartPatch System was only tested in a local network. In order to deploy it, one would still need to host a Thingsboard server at PBL, and tunnel to a network used at the test location.

Further, the SmartPatch Connector App can at the moment only be distributed for Android, using an APK file. To distribute it for iOS, at least an Apple Developer membership would be needed.

6.2.3. Efficiency

Efficiency was not the first priority in the development of the Basestation Software. So in order to handle raw data logging, which is only implemented experimentally or lots of SmartPatches and Connections at once, there could be made some adjustments. The whole publishing architecture could for example be handled asynchronous and MQTT connections could be upheld for as longs as possible to avoid time-ineffective re-connections.

Appendix A

Task Description

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CENTER FOR PROJECT-BASED LEARNING

Fall Semester 2021

D-ITET Flagship Project SmartPatch

Design of wearable electronics for medical data collection

Bachelor Thesis

Cyrill Knecht
cyrknecht@student.ethz.ch

27. October 2020

Advisors: Michele Magno, michele.magno@pbl.ee.ethz.ch
Christian Vogt, christian.vogt@pbl.ee.ethz.ch
Silvano Cortesi, cortesis@ethz.ch

Professor: Prof. Dr. Sebastian Kozerke, kozerke@biomed.ee.ethz.ch

Handout Date: 07.10.2021
Due Date: 13.01.2022

Project Goals

Sepsis is a life-threatening condition that arises when the body's response to infection causes injury to its own tissues and organs. With early intervention, it is possible to mitigate the course of the disease, which reduces the risk of a fatal outcome. Due to the costs and difficulties of full-time monitoring of patients in intensive care units, this flagship project aims at developing a continuous patient monitoring system using a low cost smart patch. In collaboration with the University Hospital of Lausanne, this smart patch will then be used for early detection of sepsis. PBL tries to give the students an opportunity to organize themselves independently in a group and to work in this very group on the project. A guideline for the development of the monitoring system is the SmartPatch v2 developed by PBL, which already tries to achieve the same goal of early sepsis detection by means of accelerometer, gyroscope, microphone, PPG and body skin temperature sensor. In addition, the possibility of indoor localization using BLE 5.2 will be evaluated, which can be used to track patients within their room or hospital. The final product develped during this flagship project should offer the following features:

- be able to reliably and accurately measure:
  - Heart rate (HR)
  - (*) Respiratory rate (RR)
  - Oxygen saturation (SpO2)
  - Body skin temperature (T)
  - orientation and movement (IMU)
  - record audio
  - (*) detect certain audio signatures (fatigue, coughing or similars)
  - (*) detect fatigue, restlessness, sleeping position from gathered data
  - its current position using BLE Direction Finding
- be able to wireless stream the (possibly processed) data to the base station using Bluetooth Low Energy (BLE)
- have a high energy efficiency
- (*) be easy to use and clean (for medical staff)
- be resistant to the environment of deployment (high humidity, movement)

Additionally a base station should be implemented, located in the same room as the patch which:

- can receive data from the patch via BLE (*) from multiple patches/patients
- can manage the received data
  - (*) process the acquired data

  - store said data
  - make said data accessible at PBL via internet
- have a GUI to show the data
  - intuitive, comprehensive, reliable
  (*) being accessible over the internet
- implement/receive the results from BLE indoor localization

Tasks

The project will be split into tasks for each student and into three phases, as described below:

Phase 1 (Week 1-4)

1. Evaluation of different software components for the base station and data cloud (separated, but could be run from the same machine).
2. Initialization of the used software.

Phase 2 (Week 5-11)

1. Implementation of an application for patients to device mapping.
2. Implementation of a tabular overview of the devices with the corresponding firmware version, SpO2 value, HR value, battery level, ...

Phase 3 (Week 12-14)

1. Integration and visualization (over time) of algorithms for SpO2 and HR detection of the PPG within the data cloud (e.g. using widgets).
2. System integration of the BLE architecture to the base station (with the work of the other group members).
3. In-field test.
4. Preparation of the presentation and finalization of the report.

Milestones

By the end of **Phase 1** the following should be completed:

- Decision and setup of software for the base station and data cloud.

By the end of **Phase 2** the following should be completed:

- An application for patient to device mapping is implemented and integrated into the base station / cloud solution.
- Widgets for additional information of the patches should be implemented.
- Tabular overview of all devices inside the cloud visualization.

By the end of **Phase 3** the following should be completed:

- Final design and in-field test.
- Final report and presentation.

Project Organization

Weekly Report

There will be a weekly report sent by the candidate at the end of every week. The main purpose of this report is to document the project's progress and should be used by the student as a way to communicate any problems that arise during the week.

Project Plan

Within the first month of the project, you will be asked to prepare a project plan. This plan should identify the tasks to be performed during the project and sets deadlines for those tasks. The prepared plan will be a topic of discussion of the first week's meeting between you and your advisers. Note that the project plan should be updated constantly depending on the project's status.

Final Report and Paper

The final report has to be presented at the end of the project and a digital copy need to be handed in. Note that this task description is part of your report and has to be attached to your final report.

Final Presentation

There will be a presentation (15 min presentation and 5 min Q&A) at the end of this project in order to present your results to a wider audience. The exact date will be determined towards the end of the work.

References

References will be provided by the supervisors by mail and at the meetings during the whole project.

Appendix B

Declaration of Originality

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
	C. Knecht

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.

Glossary

Basestation An electronic device, responsible for collecting data from multiple SmartPatches, processing some of the data and sending it to a cloud.

Python A high-level general-purpose programming language..

Raspberry Pi A small and cheap single-board computer[10].

SmartPatch A wearable electronic device for medical data collection.

SmartPatch Connector App A crossplatform mobile application, to handle SmartPatches using the SmartPatch System.

SmartPatch System The SmartPatch ecosystem consisting of everything that was developed to be used with a SmartPatch. At the moment it includes the SmartPatch itself, the SmartPatch Basestation Software, the SmartPatch UI and the SmartPatch Connector App.

SmartPatch UI The SmartPatch UI is a User Interface for visualizing collected SmartPatch Data and changing configurations in the SmartPatch System.

Bibliography

[1] MQTT, "MQTT – official website," accessed 23-December-2021]. [Online]. Available: https://mqtt.org/

[2] Thingsboard IoT Platform, "Thingsboard – official website," accessed 23-December-2021]. [Online]. Available: https://thingsboard.io/

[3] N. Bernstein, "Smartpatch - design of wearable electronics," January 2022, unpublished.

[4] A. Zürrer, "Smartpatch project firmware and data processing," January 2022, unpublished.

[5] A. Hunziker, "Smartpatch project bluetooth low energy," January 2022, unpublished.

[6] R. Hunziker, "Smartpatch - indoor localization and audio acquisition," January 2022, unpublished.

[7] A. Bhawiyuga, E. S. Pramukantoro, and A. P. Kirana, "A web of thing middleware for enabling standard web access over ble based healthcare wearable device," in *2019 IEEE 1st Global Conference on Life Sciences and Technologies (LifeTech)*, 2019, pp. 265–267.

[8] EDIMAX. (2021) BT-8500. [Online]. Available: https://www.edimax.com/edimax/merchandise/merchandise_detail/data/edimax/global/bluetooth/bt-8500/

[9] Nordic Semiconductor. (2021) nRF52 DK. [Online]. Available: https://www.nordicsemi.com/Products/Development-hardware/nrf52-dk

[10] Raspberry Pi, "Raspberry Pi – official website," accessed 20-December-2021]. [Online]. Available: https://www.raspberrypi.com

[11] Flutter, "Flutter – official website," accessed 23-December-2021]. [Online]. Available: https://flutter.dev/