Delft University of Technology

BAP

EE3L11

Design of a Telehealth System

Authors:
Geert Jan Meppelink (4692810)
Yavuzhan Mercimek (4658582)

June 19, 2020

Health2Go

TU Delft Delft University of Technology

Abstract

This study proposes a system for constant monitoring of ECG and respiration signals using a wearable. The proposed system uses capacitively-coupled electrodes for the measuring of the ECG-signal and a resistive strain sensor for the measuring of the respiration signal. The system applies the strain sensor to the abdomen of a patient, and integrates the electrodes with the rest of the components into clothing to maximize comfort. A battery life of at least 12 hours before changing the battery to recharge is estimated. Options for changing the system or its components to favour certain applications are discussed. A graphical user interface is developed which includes a login screen based on the SHA-256 hashing algorithm, a patient tab that visualizes stress and other important features, and a physician tab that also includes the raw data and options for contacting or adding a patient. The graphical user interface uses pre-measured data stored on a Microsoft Azure server.

Preface

This report was written in the context of the Bachelor Graduation Project to obtain the Electrical Engineering Bachelor at Delft University of Technology. We would like to thank dr. Carolina Varon Perez for her continuous help and support throughout the project. We also want to express our sincere gratitude to both dr. Ioan Lager and dr. Carolina Varon Perez for giving us the opportunity to continue the project amid the Covid-19 situation. We would also like to thank prof.dr.Leo de Vreede and dr. Francesco Fioranelli for taking the time to be on the jury for our final assessment.

We would also like to thank our other group members, Talha, Enes, Isar and Bob, who have worked very hard together with us. Without their contributions, this would not have been possible. We had daily meetings with the group, which was divided into three subgroups, and biweekly meetings with Carolina. Their insight has greatly contributed to our progress throughout this project.

- *Geert Jan Meppelink*
- *Yavuzhan Mercimek*

Contents

Chapter 1

Introduction

The WHO declared stress as the health epidemic of the 21st century. Everyone has some form of stress during the day, which can cause serious health issues [1]. Prolonged stress has been associated with a multitude of health issues, such as psychiatric disorders such as anxiety, depression, and Alzheimer's [2] and cardiovascular diseases [3].

Formerly, people had to visit their doctor or physician in person to get their health monitored and to get an analysis. This requires a lot of time and commuting which is not always necessary. This is why Telehealth systems have been emerging in the past years, which makes remote health monitoring straightforward. These systems allow the physicians to view their patient's important health signals and to give them an analysis without having to meet in person. These systems have also been allowing patients to view their own health related information which can help them improve their own health. The novel and coming Telehealth systems would improve on these by integrating a better network for a physician to view the analysis of multiple patients, and a better visual platform for a patient to monitor their own health.

With this in mind, the goal of this project is to create a Telehealth system, which is capable of detecting whether a person is stressed using their Electrocardiogram(ECG) signal and their respiratory signal (RS). This information will then be used to give the users feedback on their stress levels during the day and will be sent to their physician. This project will combine the knowledge of signal processing, cardiovascular biology, computer science and psychology to achieve these requirements.

1.1 Current technology

While wearable systems capable of ECG sensing already exist, [4], with some even having an application to directly monitor the sensor information [5], their use is mostly focused on the detection and prevention of cardiovascular diseases. Research regarding wearable respiration sensors has also been conducted [6]. Papers which link respiratory pattern and heart rate variability to stress have been published [7]. However, these papers do not combine a ECG and RS sensor which is used for real-time stress detection in combination with a suitable graphical interface.

1.2 Problem definition

The tasks of this project have been divided into three parts:

- **Pre-processing [8]:** The raw data coming from the wearable is processed by filtering and doing a quality assessment on the signals. This is done to remove artifacts from the raw data for a clean and reliable information extraction. The reliability of the signal is indexed based on a three step decision system. The result of the quality indexing indicates if a signal is good or bad.
- **Stress detection [9]:** The detection of the stress levels will be done using the pre-processing part and machine learning, conditioning it to classify and detect stress. Detecting stress is done via features, which will be extracted from the filtered ECG and respiratory signals provided by the pre-processing group [8]. To obtain the features, signal processing steps, like filtering and the wavelet transform have been used.

- **System design:** The original task of the system design subgroup was to design a whole Telehealth system, however, due to the COVID-19 pandemic, this goal was redefined. Instead the focus was shifted to integrating the results of the other subgroups into a intuitive Graphical User Interface(GUI). An overview of the recording system will be given in addition to a detailed design and implementation of a software platform.

The problem this thesis will assess is the creation of a coherent system capable of acquiring ECG and respiratory data as well as displaying relevant information for the assessment of stress, after the necessary calculations are done by the other two subgroups. This includes all the components in between, such as data transmission and server storage. Merging these existing technologies mentioned in section 1.1 with additions created during this project will provide novelties, such as autonomous stress detection and an easy-to-use interface.

1.3 Paper structure

Firstly, in Chapter 2, the requirements and trade-offs of the monitoring system and graphical user interface are defined. In Chapter 3, the different parts of the system are defined and recommendations for them discussed. After that, in Chapter 4, the graphical user interface is explicated. The results of the graphical user interface and design of the system are discussed and compared to the programme of requirements in Chapter 5, and a conclusion is given in Chapter 6.

Chapter 2

Programme of Requirements

In this chapter, the programme of requirements is defined, which serves as a reference for the execution and achieved results of the project.

2.1 Project limitations

2.1.1 Ethical limitations

Telehealth systems can use a variety of sensors to acquire the data from the user. These sensors collect sensitive personal information, which needs to be protected. For this reason, there are a number of laws and protections which every Telehealth system should uphold and will protect your private information from being leaked [10]. Such as the HIPAA (Health Insurance Portability and Accountability Act) [11], which requires healthcare providers to follow procedures that will protect patient health information, in the USA. The European equivalent, the GDPR (General Data Protection Regulation) [12], is a more general data protection regulation not only exclusive to the healthcare sector. One of the requirements is that no patient will be monitored unless they know it and agree to it. An agreement is to be signed for this.

Security

Typical security threats for Telehealth systems include [10]:

- Breach of confidentiality during collection of sensitive data or during transmission to the provider's system.
- Unauthorized access to the wearable of the user
- Untrusted distribution of software and/or hardware to the patient

A number of techniques are used to prevent such risks, with the most notable being Data encryption. Data encryption secures the data by processing the true data through a complex mathematical algorithm before storing or transmitting it [10].

Equality

An ethical issue arises when certain populations are deprived of Telehealth service due to their lack of access to technology, or their lack of technological knowledge [13]. This would cause an unfair distribution of Telehealth aid. As Telehealth is also needed in remote places with little to no access to modern health care. These places, however, tend to also be the places with limited access to internet bandwidth, a core necessity of Telehealth [13].

From these statements we can conclude that this system needs to adhere to the following limitations:

- The acquired data must only be used for what is stated, which is to detect if a individual is stressed using their ECG and respiratory signal.
- User data must be secured and only accessible by someone with specific permission.
- The system should not be limited to advantaged populations, but should be as accesible as possible

2.1.2 Physical limitations

In addition to the ethical limitations, designs generally also have some physical bounds. This project is generally limited by its compatibility and its ease of access. They will help us to determine the boundaries for the physical part of the system, which will be designed based on these goals. These physical limitations are:

- The system should be wearable and not hinder the daily activities of the user.
- The system should be non obtrusive and non-visible when worn under clothing to protect the patients privacy.

2.2 Project requirements

Keeping in mind the limitations given in section 2.1, the PoR for this subgroup can be given as:

- The system needs to work as close to real time as possible: ECG data needs to be updated at least every 10 seconds and the respiration data at least every 30 seconds. The stress indicator should be computed at least within the segment length to ensure real-time visualization. This segment length is taken to be 2 minutes maximum as stated by the stress detection subgroup [9].
- The battery life of the system should be at least 7 days to limit the amount of times the battery has to be replaced.
- The total weight of the of the complete system should be less than 250 grams. This is about the same weight as a smartphone on the heavier end of the spectrum, and should be adequate for a wearable system.
- The Graphical User Interface (GUI) needs to have a login screen to restrict the access of personal information to those permitted.
- The GUI is to display whether or not a person is stressed, and if so provide a calming exercise and an option to play a calming song.
- The GUI is to display data retrieved from the ECG and respiration signals, such as the respiratory rate, heart rate and the processed signals.
- Data must be stored on a server and remotely accessible.

Chapter 3

System Design

Would the project not have been hindered by the current corona virus pandemic, a bigger part of the actual system could have been implemented. This is not possible anymore, but that does not inhibit the task of designing an actual system that could complete the functionalities of the current project instead. Instead of pre-measured ECG and respiration data, the data will be measured by sensors that can comfortably be worn by someone all day long. The system will use the developed pre-processing and stress detection parts to process the data and display the processed data on both a patient wearable or smartphone and a physician web interface. The data will have to be shared in between these parts and be stored on a server for later accessing as well. In this chapter the possibilities for such a system will be discussed. A general overview of such a system is given and the separate parts are laid out and characterized. The specifications of different options are determined, and their advantages and shortcomings discussed. Also, the way the data can be handled and transmitted between the different parts of the system is analyzed.

3.1 General overview

The system consists of four distinctive parts, which can be seen in figure 3.1:

- Wearable sensors, which will collect the ECG and respiratory data from the patient. This part also includes data transmission to the smartphone/wearable.
- A patient smartphone or wearable, where the patient can view their own data. The smartphone/wearable is also responsible for relaying data to the server.
- A server that stores the patient and login data.
- A remote computer capable of accessing this data from the server and displaying them for the physician.

Further elaboration of the methodology of the system design can be found in the following sections.

3.2 Wearable sensors

The wearable sensors part of the system is responsible for obtaining the analogue physiological ECG and respiration rate data, converting it to digital data and transmitting it to a patient wearable/smartphone for further processing and visualization. One of the main requirements of this part of the system is the comfort for the wearer, as the system is designed for long-term use. A schematic overview of the designed wearable part of the system can be seen in Fig. 3.2. The blocks within this system will be elaborated on in the different subsections of this section.

3.2.1 ECG measurement

An electrocardiogram (ECG) is a graph of voltage versus time of the electrical activity of the heart. This electrical activity occurs due to continuous depolarization and repolarization of the heart. This polarization is due to the heart muscles, just like other muscles, being electrically stimulated. The inside of a heart cell is negatively charged relative to the outside. This means that when the are stimulated, the cells depolarize

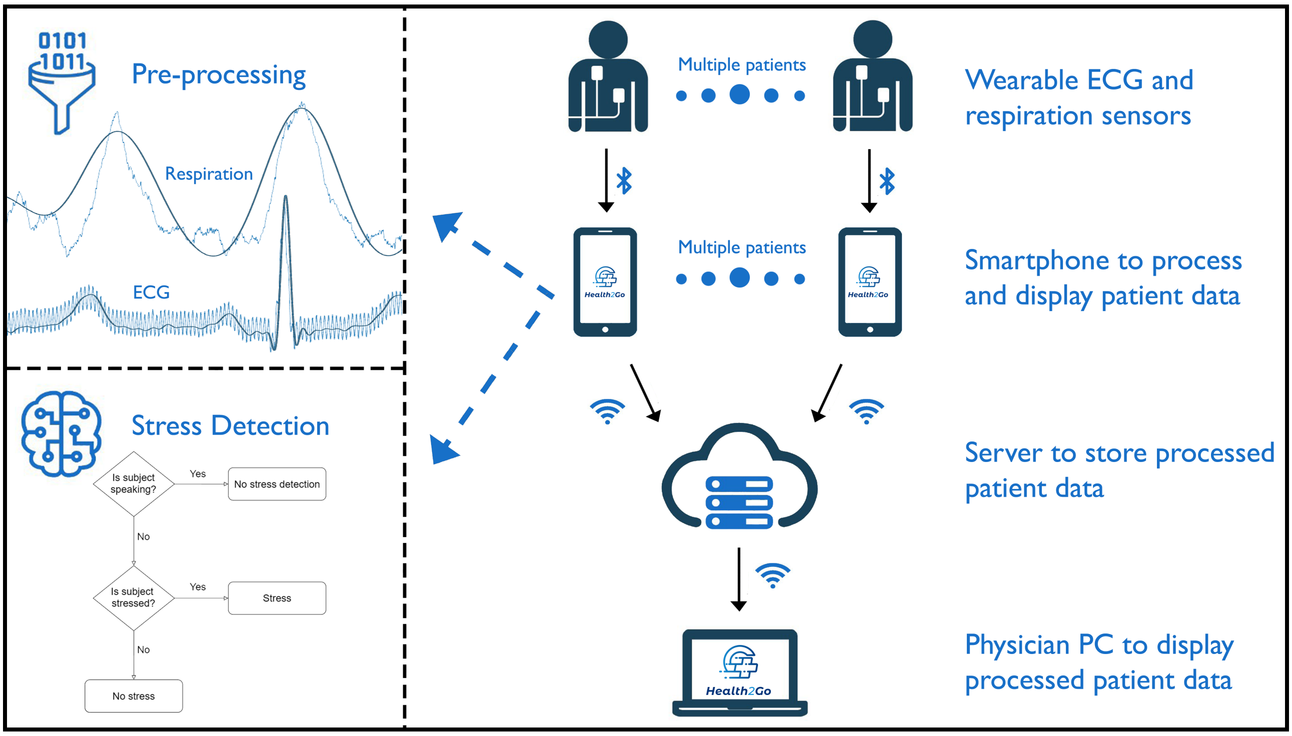


Figure 3.1: General overview of the system design including stress detection and pre-processing.

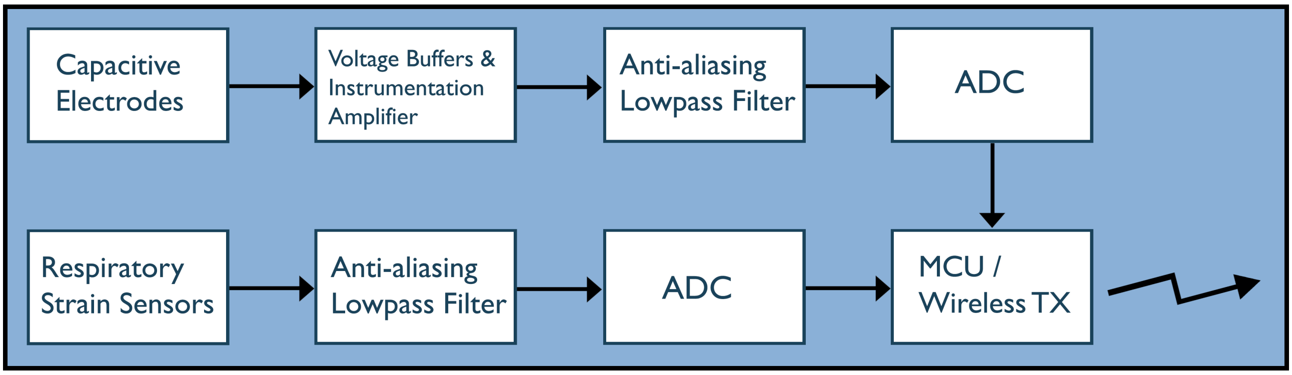


Figure 3.2: Schematic representation of the wearable sensor architecture

and the muscles contract. After which they repolarize and the muscles loosen. This electrical activity can be measured using electrodes and subsequently processed. The sensor system needed to detect the ECG consists of four parts:

- Electrodes
- Amplifier
- Anti-aliasing low-pass filter (LPF)
- Analog-to-digital converter (ADC)

Electrodes

The electrodes are the heart of the ECG measuring system. In a conventional system, called resistive or wet ECG, 12 or 15 Ag-AgCl electrodes are attached to different parts of the body using gel to improve conduction. Even though this kind of system provides high quality signals, they are inconvenient and could cause allergic reactions or even inflammation due to long-term use of the gel that was being used to attach them [4]. As stated before, due to the way that the system will be used, the goal is to design a system that hinders the user

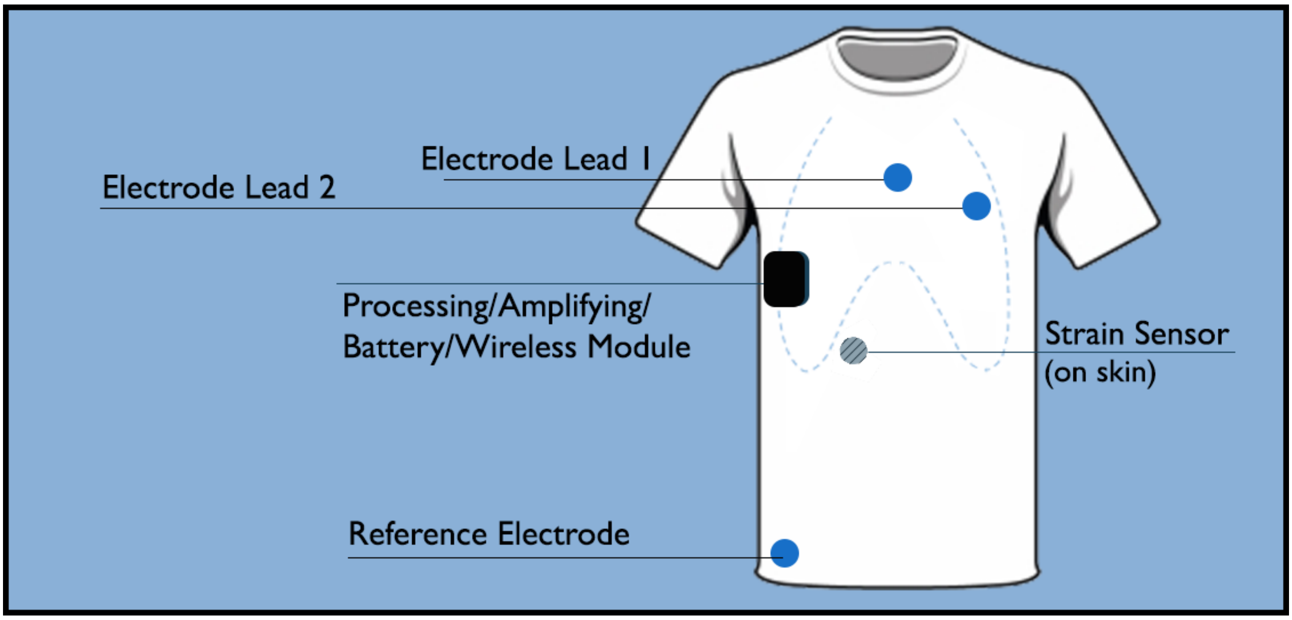


Figure 3.3: Schematic of the wearable sensors part integrated into a t-shirt

as little as possible. In such systems, the usage of non-contact capacatively-coupled ECG (CC-ECG) sensors are preferred. In contrary to the standard twelve-electrode ECG, a three-electrode ECG system is used. Even though it is desirable to have as few electrodes as possible to reduce size and weight, the removal of the third electrode is challenging, as two-electrode acquisition systems have significantly higher electromagnetic interference (EMI) and lower signal-to-noise ratio (SNR) [14]. These sensors are not placed directly on the body. Instead, a layer of insulator is placed between the skin and the metal electrode. Normal clothing can be used as an insulator to improve the wearability for the patient. This provides a good option for integrating the sensors to minimize their inconvenience. A schematic overview of the recommended way of incorporating the sensors into a t-shirt is shown in Fig. 3.3. Nevertheless, to maximize the working, the material of the clothing should have a high dielectric constant and be very thin. The use of cotton cloth as an insulator produces comparable ECG signals to wet ECG systems, while for example wool disturbs the signal [4]. The electrodes, skin and insulators form capacitances. Two of these electrodes, placed on the upper body, are used to measure a body surface's potential difference. The third electrode is used as a low-impedance path for noise reduction. Used this way, a three-electrode ECG system provides sufficient sensitivity [14].

This method has several advantages over the wet (resistive) ECG method. Apart from the added comfort of not having a dozen electrodes attached to you, the sensors can be installed in other objects, like furniture for example, to monitor a patient without them having to wear the sensors. One way to implement such sensors is to attach them to clothes that can be worn by the patient. This way, the patient can wear them with minimal discomfort. But, these sensors do not only have advantages. They have some shortcomings when compared to resistive systems. The signal quality is not as good as with ECG-systems connected to the skin, because there is a significantly higher impedance between the signal sensor and the source. The system can also generate high unwanted currents that can overwhelm the ECG current due to movement artefacts. When the position of the electrodes changes, so does the capacitance. This will generate an unwanted current that may overwhelm the ECG signal. And lastly, due to the high impedance, the electrodes suffer from power-line contamination due to capacitive coupling.

There are a few constraints on the kind of electrode that will be used. The impedance between the skin and the electrode should not be too high, so, the electrode should be large enough to ensure this, but small enough to fit comfortably in a piece of clothing. The characteristics of the piece of clothing that the electrode is attached to are important as well. To minimise artefacts due to movement, the electrodes should stay at their relative positions as much as possible. Therefore, it is recommended to incorporate them into a piece of more tight clothing, like to a bra, a tight undershirt or even a halter designed for such systems.

ECG amplification

Amplification of the ECG signal is needed in order to fit the signal to the range of the ADC (-1.65 to +1.65V [15], see subsection *ECG analog-to-Digital conversion*). Also, due to the high input impedance (in the range of 10GΩ [4]) of the ECG sensor, a voltage buffer with high input and low output impedance should be used in order to match to the low impedance of the amplifier. The amplifier is a big part in reducing the noise in the system. Common-mode rejection (CMR) is one of the most important performance parameters in an ECG measuring system. This is due to the large amounts of electromagnetic interference (EMI) that is coupled to the system. This can be coupled through the patients skin to their body, or via the electrodes or other elements. This interference, which originates from the 50 Hz (Europe/Asia) power lines, ends up as a common-mode (CM) noise in the system. Due to mismatches in the system, this interference also results in differential-mode (DM) noise, which deteriorates the systems CMR performance.

To combat this degradation, different options can be implemented to improve the CMR performance. One straightforward option is to shield the system to reduce the amount of power-supply interference that enters the system. This way, other additional environmental noise is also reduced. This could be a good option for the cables, where mismatches might occur more often. Too much shielding might add too much weight to the system and reduce the comfort. Therefore, shielding of other parts of the system is not preferred. To help improve the CMR performance though, the third electrode can be used as a reference input for the voltage buffers. This is called a driven right leg circuit (DRL), due to the third electrode commonly being applied to the right leg, farthest away from the heart [16]. The system improves CMR by sensing the input common-mode voltage at the voltage buffer.

In addition to all these techniques to improve CMR performance, voltage buffers and amplifiers with good qualities should be chosen. For the voltage buffer, low noise and voltage input offset are desirable, as this reduces the common-mode noise. The LMC6001 Ultra [17] Would be a suitable amplifier for these reasons. For the differential amplifier, the INA106 [18] is recommended for its low noise ($1\mu\ V_{p-p}$), reasonable gain (60 dB) and a high CMRR (110 dB).

ECG anti-aliasing low-pass filter

In the pre-processing part of the complete system the required filtering is done digitally to prepare it for stress detection. However, an anti-aliasing filter is required before feeding the signal to an ADC. According to the Nyquist theorem, the sampling rate should be at least twice as high as the highest frequency component of the signal. According to the American Heart Association (AHA), information in the ECG can still be found up to 150 Hz for adults, adolescents and children. It is stated that data sampled at 500 samples per second (sps/Hz) is needed to correctly convey this information [19]. This- means that a low-pass filter with a cutoff frequency at 150 Hz with sufficient attenuation at 250 Hz could be used to prevent aliasing. A single-pole passive RC filter with a -3db frequency of 150 Hz offers enough attenuation (-4,4 dB) at the Nyquist frequency of 250 Hz, and it has the advantage that it does not need any extra power or amplification. Therefore, it will take up less space and weigh less compared to an active filter. This is the preferred option. If needed, an option could be to slightly increase the sampling rate to combat any aliasing that might occur from the frequencies in the transition band. A different option would be to use higher order active filters to improve the roll-off, but this would be at the expense of having a bigger system that also uses more battery.

ECG analog-to-Digital conversion

For the ADC, the AD7779 [15] is proposed. It has a high signal to noise ratio (108dB) and resolution (24bit), it has low power consumption (3.37 - 10.75mW), and it has eight ADCs in it, which can be used simultaneously. This is especially handy as this makes it possible to use it as an ADC for the respiratory signal as well. This reduces the complexity of the system as well as the costs. The costs of the system will be extensively reviewed in the additional business plan of the project, which will be issued at the same time as this thesis.

3.2.2 Respiratory measurement

The respiratory rate of a person is the rate at which their breathing happens, usually measured in breaths per minute. This signal is traditionally measured by using a respiratory flow sensor attached to a face mask. These medical flow sensors are often not only large and expensive, but they are in no way comfortable if someone would need to wear it for longer amounts of time. A good wearable sensor system for respiration rate would be lightweight, flexible, durable and robust to motion artefacts that will appear during normal use.

The sensor system for measuring of the respiration consists of three parts.

- Respiratory sensors
- Anti-aliasing low-pass filter (LPF)
- Analog-to-Digital converter (ADC)

Respiratory sensors

Wearable sensors for respiratory monitoring can consist of various types of sensors that can be attached to a person in numerous ways. They could be worked into clothes, attached to belts or just placed on the skin. When employing such sensors, it is important to know what type of changes are expected to be registered during breathing. There will be an airflow in and out of the mouth and nose, the lung volume will increase and decrease and the concentration of oxygen and carbon dioxide in the blood will change. Different options for sensors include [20]:

- Pressure sensors
- Acoustic sensors
- Humidity sensors
- Oximetry sensors
- Accelerometers
- Resistive sensors

Due to the constraints of the system, some sensors are clearly more useful for this application than others. The system should be as lightweight and comfortable as possible. Pressure sensors could, for example, be integrated in a belt that a person could wear around the chest to measure the expansion during breathing. Another usage of belts comes in the form of resistive or acceleration sensors. These belts measure the variations in resistance and position, respectively. However, because such bands are rather bulky and prone to moving, they are not suitable for systems that require constant daily monitoring without interfering in daily lives [6]. In the proposed system though, resistive sensors still form a good option for measuring. However, not in the form of belts, but in the form of small piezo-resistive strain sensors. These small sensors are based on the materials ability to increase resistance with respect to strain. When applied to a part of the chest that moves during breathing, such as the ribcage, the variations in resistance can be measured with the use of a voltage divider to produce a respiration rate signal that can be used by the pre-processing and stress detection parts of the system. The proposed method of integrating this sensor into the system can be seen in Fig. 3.3, where the strain sensor is applied to the skin of the abdomen.

Respiratory anti-aliasing low-pass filter

As stated in the anti-aliasing section of the ECG sensors. No filtering of noise due to signal acquisition is needed, but rather to prevent noise due to aliasing when converting from an analogue signal to a digital one. According to [21], useful information in the respiration signal can be found up until 0.5 Hz. This is also used by the pre-processing subgroup [8]. This means that the sample rate of the respiration rate will be way lower than the sample rate of the ECG-signal.The same idea for a first order low-pass filter can be used, this time with the cut-off frequency at 0.5 Hz. A first order low-pass filter has a 20 dB/decade roll-off, which equals a 6 dB/octave roll-off. This means that the signal will be attenuated with 6 dB more at 1 Hz compared to 0.5 Hz . As the Nyquist theorem states, the sample rate should be twice as high as the highest frequency component. The ADC sampling rate can be set to any value [15], so the sampling rate can be rather low. A sampling rate of 4 Hz should already be sufficient, as this is the minimum required for the stress detection algorithm to work properly [**stressdetectiojan**]. The sample rate can also be chosen to be marginally higher to improve the quality of the signal, as this has relatively little effect on the total amount of data that needs to be processed, because the total data also includes the ECG-data sampled at 500 Hz.

Respiratory analog-to-digital conversion

The same ADC that is used for the ECG-sensors part can also be used for the respiratory sensors part, as it integrates options for sampling multiple analogue signals at once. As stated before, this helps to reduce the size and cost by using the same element.

3.2.3 Microcontroller unit & Bluetooth Module

The wearable part of the system concludes with a microcontroller unit (MCU) that integrates the processor of the system with a Bluetooth module for wireless connection to the next stage. The analog-to-digital converters of both the ECG and respiratory stages are attached to this module. A schematic representation of the wearable sensor architecture can be seen in Fig. 3.2.

The proposed microcontroller is the MAX32665 [22]. This microcontroller was chosen for its built-in Bluetooth Low Energy (BLE) module, its relatively high processing power (up to 96 MHz) and its dynamic voltage scaling to minimize power consumption. An 8-channel input sigma-delta ADC is also present on the microcontroller, but it cannot sample multiple analogue signals simultaneously, so it offers no possibilities for reducing the amount of ADC's in the system.

3.2.4 Power supply

As the system will be used for long-term monitoring, it is important that a battery with relatively high capacity is used to ensure that the batteries do not need to be replaced and recharged often. As the system will work continuously, using non-rechargeable batteries would result in the usage of a high amount of batteries. Therefore, a good recommendation would be to use rechargeable batteries in a pair, so that when one battery is being used, the other can recharge. A higher battery capacity also means a bigger size, so there is a trade-off between size and weight of the system on one hand, and the time it takes before having to switch the batteries on the other hand. In the programme of requirements, it is stated that the battery life of the system should be seven days. However, there is less sense in using way bigger batteries when rechargeable batteries are being used instead of non-rechargeable ones A better estimate is that the battery pair should keep the system working for at least one day, so one battery should be able to sustain the system for 12 hours. Relatively small lithium rechargeable batteries form a good compromise. Two 3.7 V thin and relatively small (33x31x5mm) with a respectable 500 mAh capacity were chosen. To provide the voltages for the amplifiers a very small low-power DC/DC voltage converter TPS61040 can be used [23]. This setup could provide the system with approximately 40 mA throughout the day. In the paper of Nemati et al. [4], a current drain of 25mA is measured using a similar system which only monitors ECG. An estimate of the power consumption using the data-sheets of the needed components shows that a consumption in the same order of about 25 mA is to be expected. This assumption was made based on the provided power consumption with the ADC channels sampling at 2 kHz and the processor in constant transmission.

Protection and isolation

Isolation is important in medical equipment, as the patient is a part of the system and should be protected. The patient must be protected from electric shock from the system, and the system must be protected from extreme voltages generated by emergency use of a defibrillator. To ensure this protection, isolation of the power and signals is required. Isolation of the signals used to be done using optocouplers, which transfers electrical signals between two isolated circuits using light. However, optocouplers tend to have poor analogue linearity and are not suitable for direct coupling of precision analogue signals. Using a digital isolator however would negate these disadvantages, while using less power as well. The proposed digital isolator is the ADuM2400 [24]. This isolator integrates high presicion data transfer at minimal power (about 1,5 mA at two channels). To seperate the power of the analogue front end from the rest of the system, another battery could be used. The same 3,7 V/500 mAh battery used for the digital part of the system can be used for the analogue front end. A current limiting device is needed to keep the current at a safe level. A limit of 10 A rms is defined by the American College of Cardiologists [25]. Solutions in the form of a resistance placed in the signal path (10s of Kilo-ohms) or current limiting devices.

3.3 Data processing options

3.3.1 Signal pre-processing and stress detection

In the report of the pre-processing [8], and stress detection [9] subgroups, the mean computation time and standard deviation have been given for their respective parts. These computation times are for segments of 10 seconds for the pre-processing and 80 seconds for the stress detection, their times can be found in Table 3.1. The micro-controlling unit (MCU) chosen for this project is the MAX32665 [22], which has an ARM Cortex-M4 [26] processor with a clock speed of up to 96MHz, which is considerably slower than the processors used for the development of the functions. Comparing the performance of different processors is not possible simply by comparing their clock speeds. This is due to the difference in architecture (ARM versus Kaby Lake [27]/Broadwell [28]), which gives each processor a different instruction set to work with. The Cortex-M4 was announced in 2010, while Broadwell was announced in 2014, this is a considerable time difference in terms of technology and one could expect that the much more powerful and younger processor has a broader instruction set. It should however be noted that the Cortex-M4 has special instructions included designed for Digital signal processing, which could make up for the lower clock speed. However, if a comparison were to be made between the different processors purely based on clock speed, and assuming that clock speed is linear to performance, the wearable MCU, the MAX32665 is expected to perform the processing within the segment length of 10 seconds. For example, a comparison can be made between the clock speed of the Cortex-M4 and the i7-5000U, which was used in one of the tests for the computation time. The ratio, $Ratio_{CPU}$, between the two clock speeds would be:

$$Ratio_{CPU} = \frac{2.4 \times 10^9}{96 \times 10^6} = 25$$

Which means that with an identical architecture, and thus instruction set, the Intel i7-5000U would be 25 times faster that the Cortex-M4.

For the calculation it is assumed that the time to perform the stress detection is evenly distributed between the 8 segments of 10 seconds. Also, it is assumed that the computation times are normally distributed, so the empirical rule of statistics states that 99.73% of all values are between six standard deviations around the mean. So, the maximum computation is assumed to be within three standard deviations added to the mean:

$$\text{Maximum computation time} = t_{max} = 0.0379 + 3 \times 0.0128 + 0.1278 + 3 \times 0.0176 = 0.2569 \text{ seconds}$$

$$\text{Cortex max computation time } = Ratio_{CPU} \times t_{max} = 25 \times 0.2569 = 6.4225 \text{ seconds}$$

This is however the maximum computation time, when the MCU has to processing the data as well as check for stress, which is only done once every 8 segments. During the other segments however, only the processing of the signals has to be done. The maximum for the standard computation time, $t_{standard,max}$ is calculated similarly as to t_{max}:

$$t_{standard,max} = 0.0379 + 3 \times 0.0128 = 0.0763 \text{ seconds}$$

$$\text{Cortex standard computation time} = Ratio_{CPU} \times t_{standard,max} = 25 \times 0.0763 = 1.9075 \text{ seconds}$$

From this calculation we can conclude that the MAX32665 would be fast enough to do the calculations needed for the processing and stress detection, if the only variable would be the clock speed. However, as is explained above, the clock speed is not the only variable. The system should be tested in real life in order to decisively conclude whether the MAX32665 is indeed powerful enough to do the full computation. It should however be noted that the calculations for the computation costs were done using data that had sample rates of 1000Hz for the ECG and 250Hz for the RS. The sample rates that are used in the design for the implemented system is lower (500Hz and 4Hz respectively). This would improve the computation time.

If however after the research it is concluded that the MCU is not fast enough, it would not prove to be a problem, as the data processing can be done after the data has been transmitted to the phone, which has a much stronger processor. Alternatively, the segment length could be made longer, which would give the MCU enough time to process the data.

	Mean time (s)	Standard deviation (s)	CPU used
Pre-processing	0.0379	0.0128	i5-8300H @2.3GHz
Stress detection	0.1278	0.0176	i7-5500U @2.4GHz

Table 3.1: Table with the mean time and variance of the computation on their respective CPU.

3.3.2 Wireless transmission

Data transmission will be executed across multiple devices. For this reason, multiple methods of wireless data transmission will be needed depending on factors like power consumption, cost, range and bitrate. The two ways of telecommunication will be through Bluetooth Low Energy (BLE) and WiFi/LTE.

The connection between the sensors and the smartphone should be done using BLE, this is due to the significant difference in energy consumption between them: the power consumption of WiFi is approximately $10^3 - 10^4$ times higher than that of BLE [29]. The power consumption of LTE is even higher than that of Wifi [30]. Thus, by choosing BLE for the wireless communication between the sensors and the smartphone, the battery life of the sensors will be significantly longer compared to when other modes of wireless communication would be used. There are some drawbacks with using BLE, such as the lower bitrate and range. The range should however not pose a problem as the connection will be between a sensor on the body and a phone, which is generally held in close proximity.

The data that will be sent are the processed ECG and respiratory signal. A quality indicator, which indicates whether the signal is of bad quality according to the pre-processing subgroup, and the stress indicator. If a person is indeed stressed, the unfiltered ECG will also be transmitted to the server in order for the physician to analyze the raw data. There is a trade-off between the amount of patient information sent to the server, and the power consumption and server storage. One option would be to constantly send all the data to the server. This would require the most power and server storage. Another implementation could be to only send the data around the time when stress is detected, or only a scheme of the times of day that a person experiences stress. Different options can be implemented for different practices. If the system would not be constantly transmitting, it would reduce the power consumption. Also, fewer data sent means less server storage. Another option would be to greatly down-sample the processed ECG and respiratory data after the stress detection part has already taken place. This means that a lot less data has to be sent.

The Data transmission is done via BLE with a bitrate of 1 Mbps and 2 Mbps. Most of the data will be of the ECG which will be sampled at 500 Hz. Due to the circumstances during this project, no data from the proposed system is available and thus the actual file sizes can not be given. During the course of this project, however, ECG data sampled at 1000Hz has been used. By simply dividing the data up as if it were sampled at 500Hz in segments of 10 seconds, the average file size of a similar ECG sensor sampled at 500Hz has been determined to be 35Kb. This is small enough to be sent without any delays.

Data transmission from the phone to the servers and vice versa will be done through the connection of the phone, whether it is connected trough WiFi or via LTE. This due to one reason, which is connectivity. The simplest and most accessible way to transmit data to the server is via an internet connection. Connection between the smartphone and the server will be every 10 seconds in order to keep the displayed information as close to real time as possible.

3.4 Server storage

For this project, data storage has been done using Microsoft Azure [31]. The reason for choosing Azure was partly that it provides a broad spectrum of services which could be used to further improve the development of the GUI. The other part is that Azure is relatively simple to connect to MATLAB for basic tasks, such as reading and writing data to an Azure BLOB-storage [32]. One disadvantage of Azure however is that MATLAB cannot read '.mat' files from Azure, only '.csv' and '.txt' can be read from the storage without actually downloading data. For this reason, the file extension used for data storage will be '.csv'.

One thing that should be noted is that for testing purposes, the data used during this project is stored in segments of 50 seconds for both the ECG and RS. This is due to not having a complete system which can continuously send data every 10 seconds.

3.5 Alternative design

This design relies on items that could be considered as a luxury: multiple sensors, ADC's and filters, which can quickly increase the cost of the system. In addition to this, not everyone has access to the internet. As explained in section 2.1.1 of chapter 2, new Telehealth systems should be as widely available as possible. For this reason, an alternative design is proposed. This design will consist of the same components as the original design, with some features changed. A need for a separate respiratory sensor , for example, is not crucial, as this could be extracted from the ECG-signal, the cutoff frequencies can be taken at 0.05 and 1 Hz of the ECG [33], and could be extracted. However, this might worsen the stress detection. Now a cheaper processor can be used with less processing power. Compromises can also be made on the need for an internet connection. The user could connect his phone to a physical storage device, like a USB-stick or a personal computer for example, from which later on the physician could retrieve the data and observe it another time. This can also be used to do the processing and stress detection, which would have normally been done on the MCU. This further reduces the need of an MCU with high processing power, making the overall system cheaper. Alternatively, the memory on the MCU could be increased and the data could be stored internally, from where the physician can later on download the data locally via Bluetooth.

3.6 Use of personal information

The goal of this project is to create a system in addition to an application, intended for public use, which collects and processes personal information of its users, it should be noted that the application developed during this project should adhere to the GDPR before being publicly released. As this design will not be fully physically realized in the scope of this project, steps will be taken to ensure the protection of personal data according to the GDPR, but will in no way be sufficient to comply with the GDPR.

Chapter 4

Graphical User Interface: Health2Go

The GUI, named Health2Go, is the part that connects the users to the functionalities of the app. There will be two different GUI tabs, one for the patient and one for the physician. The patient GUI will display the important signals, such as the filtered and unfiltered respiration rate, and the Heart Rate (HR). It will also be able to help the patient calm down through a breathing exercise. Besides the signals available on the patient GUI, the physician GUI will also show the filtered and unfiltered ECG signal, and allows for quick contact with the patient. As the physician must be able to monitor multiple patients at once, the physician GUI offers the option to switch between different patients from within the GUI. To ensure the safety of the patient data, both GUI's can only be accessed from a login screen with the right username and password combination. The right combinations can grant access to either the patient or the physician GUI, based on the security access levels.

The GUI presented below has been made using Matlab(R2020a) App Designer and its code can be found in Appendix A.1

4.1 Login Screen

The login screen is the first and only tab that is visible when the app has just launched. The purpose of the login screen is very straightforward. Grant access to people with the right login credentials, and forward them to the corresponding GUI. Of course, when the right credentials are not entered, access will be denied, and they will stay on the login screen. The appearance of the login screen can be seen in Fig. 4.1.

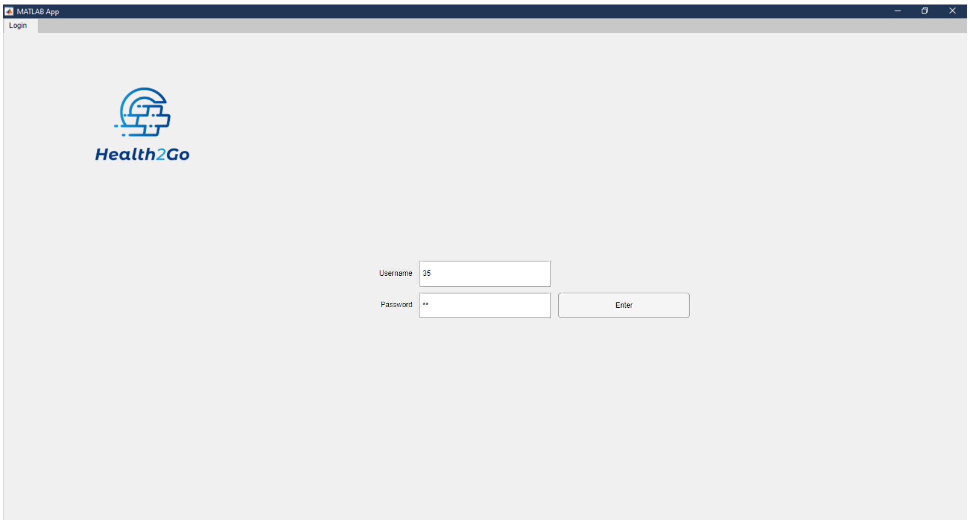


Figure 4.1: Visual of the login screen

4.1.1 Functionalities

The login screen has two visible text fields and one visible button. The first text field is for filling in the username, and the second field for the password. As is traditional for most password fields, the password itself is replaced with asterisks while typing. The characters are not masked, but changed. If someone were to copy the password field, they would only get asterisks. The actual password is namely stored somewhere else, and then together with the username compared to the known combinations. The way this is done is by using a third, invisible text field. While typing, the characters of the password are changed to asterisks one by one. Every time before that happens though, the first character that is not an asterisk is copied to the invisible field, and concatenated with what was already in there (Appendix A.15).

When the enter button is pushed, the combination of username and password are checked to see if there is access connected to the combination, and if so, which user interface should be displayed. The way this is done is as follows. The login data is stored on the Microsoft Azure server in a file with three columns. The first column is the username, the second is the hashed password, and the third is a distinct ID. The hashing of passwords was done using an SHA-256 hashing mechanism. The SHA-256 mechanism was used for a couple of reasons. Firstly, due to the way passwords are handled within the system, there is no need for two-way encryption. The entered password is hashed and then compared to what is stored on the server. SHA-256 is a one-way hashing system, which is enough for the purpose of this project. It provides better security than similar hashing functions, such as MD5 and SHA-1 [34]. It should however be noted that using only SHA-256 for password hashing is not enough as it is still vulnerable to commonly used tactics such as rainbow tables, which are pre-computed tables with commonly used passwords. The use of SHA-256 in this project was only as a proof-of-concept and a step in the right direction rather than a true security measurement. The use of a salt for example, would increase security and prevent unauthorized access to the system.

When enter is pressed, the app will look for the username within the first column of the file. When the username is located, the hashed password is compared to the entry of the second column of the same row, where the corresponding hashed password is stored. If this is also equal, the person will be forwarded to their personal screen based on their ID tag in the third column, which corresponds either to a patient GUI with their own data, or the physician GUI with all the data available, the code for this can be found in Appendix A.7.

4.2 Patient Graphical User Interface

The Patient tab is meant for personal use only. A patient that is being monitored can access relevant information from here.

When a patient opens the application a large green ring and a graph on the left side of the application, as seen in figure 4.2, will be shown. This green ring will turn red depending on the outcome of the stress detection algorithm [9]. In the ring, the heart rate of the user will be displayed with a 10 second delay. This delay has been explained in section 3.3 of this thesis. Below the ring, the respiratory signal of the patient is displayed (figure 4.3), This graph will be plotted in a scrolling manner in order to simulate a continuous connection, even though the data is only updated every 10 seconds. Both the heart rate as well as the respiratory signal displayed have first been processed and calculated by the pre-processing subgroup [8]. The reason for displaying only the heart rate and respiratory signal instead of both signals and both rates was, aside from clutter on the screen, that a user understands and knows what a healthy heart rate is supposed to be, but cannot retrieve any useful information from the ECG signal. On the other hand, a user most likely does not know what a healthy respiration needs to be. He/She can however intuitively feel when he/she is breathing too fast. By also giving a visual aide in the form of a graph which shows their breathing pattern, it will help them to better gain control of their breathing.

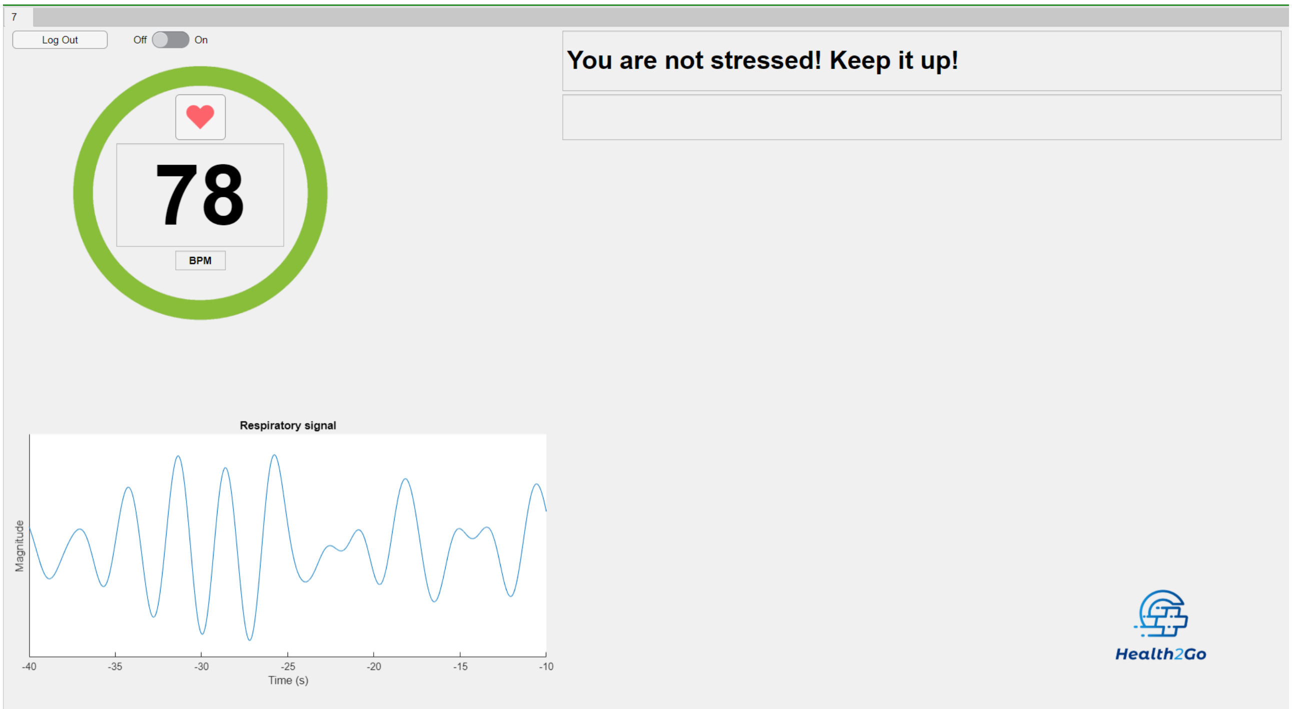


Figure 4.2: Interface when the patient is not stressed.

When stress is detected for a user by the stress detection subgroup, the green ring will turn red, as seen in figure 4.3 (Appendix A.19). In addition to this, a breathing exercise will be displayed, to help the user calm down. Feedback will also be given in the text box in the upper right corner to help the user match their breathing with the exercise: if the breathing rate is not within regular margins, the subject will be encouraged to slow down or speed up until the desired respiratory rate is reached (Appendix A.6). While the breaths per minute which the exercise encourages, approximately 6.67 breaths per minute, is lower than normal for a non-respiratory compromised adult (12-20 breaths per minute [35]), slow breathing exercise have proven to positively influence relaxation [36]. The user is also given the option to listen to the song 'Miserere Mei' by Gregio Allegri which helps reduce stress in patients, according to Myriam V Thoma et al. [37] (Appendix A.11, A.16).

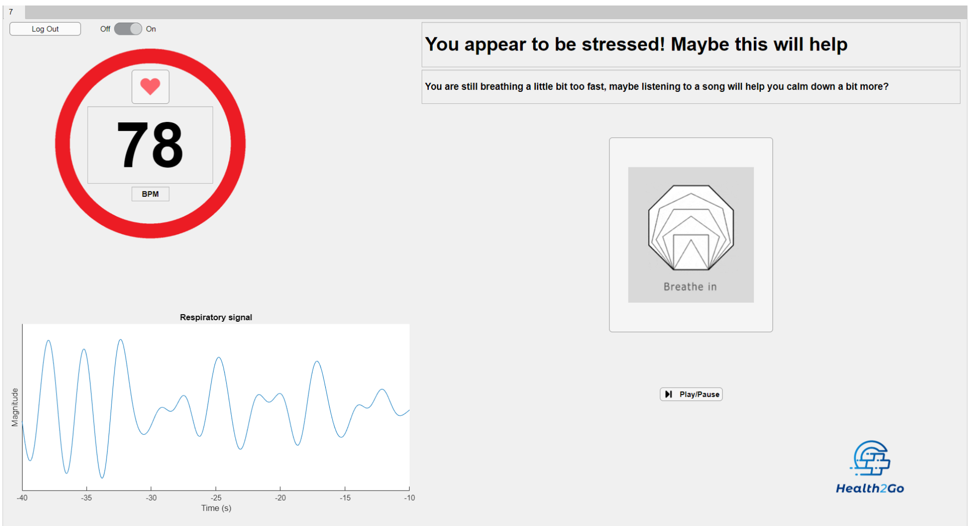


Figure 4.3: Interface when the patient is stressed.

4.3 Physician Graphical User Interface

The physician tab is meant for physicians who need to check up on their patients. It provides more detailed information and an option to contact their patients.

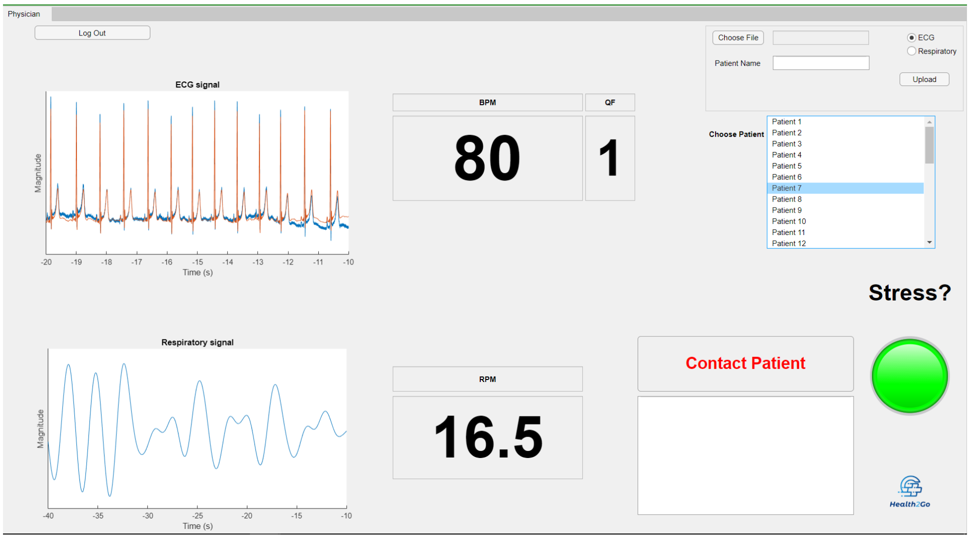
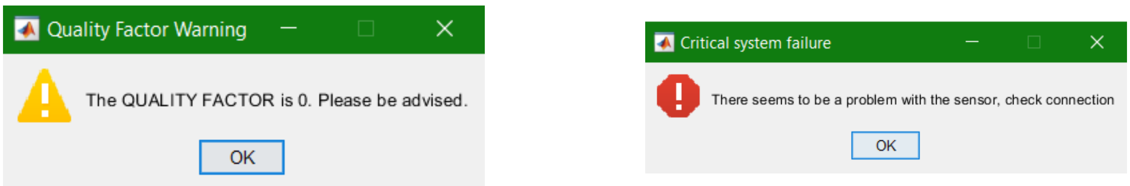


Figure 4.4: Interface for the physician.

Figure 4.4 shows the screen a physician will be shown when logged in. The upper left graph shows both the processed (red) and unprocessed (blue) ECG signal of the patient. Below, in the lower left corner, the processed respiratory signal can be seen. These graphs will be plotted in a scrolling manner in order to simulate a continuous connection, even though the data is only updated every 10 seconds. Next to both graphs the heart rate, as well as the respiration rate, are displayed (Appendix A.10, A.17). Additionally, a binary quality factor indicator is displayed. If the quality factor is 0 for three consecutive segments, the signal is of bad quality and a warning will be displayed to make the physician aware of this (figure 4.5a). The system will also notify the physician when there is a prolonged period of time (10 seconds) where there are inputs missing from either the ECG or the respiratory signal, which can be seen in figure 4.5b .



(a) Warning notification when the quality factor is 0.

(b) Error notification when there is missing data

Figure 4.5: Notifications of the system

The 'Contact Patient' button together with the text box underneath can be used to quickly send a mail to the patient. The empty text area can be used to fill in a personal message to the patient. A notification will be displayed when the mail has been sent (figure 4.6)(Appendix A.5). The green disc, next to the 'contact patient', button works similarly with the green ring in the patient tab. When it is green, the patient is not stressed, and when it turns red, it indicates that the patient is stressed.

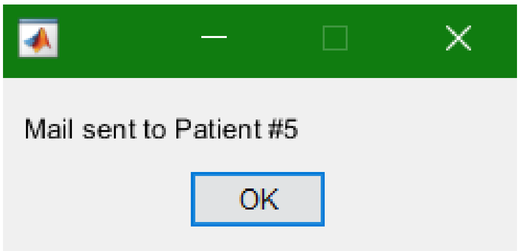
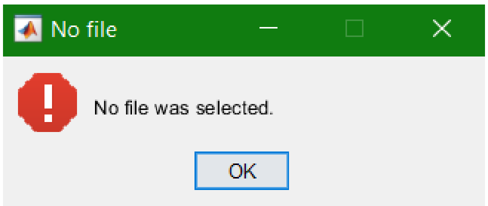
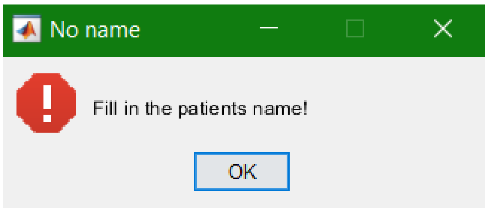


Figure 4.6: Notification for when mail is sent and to whom.

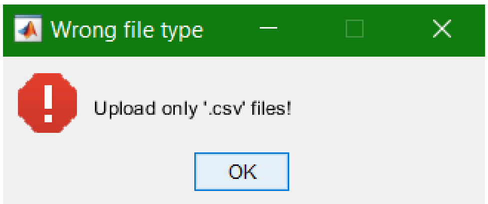
If the physician already has ECG or respiratory data from one of their patients, the data can easily be added via the upload system on the upper right side of the interface (Appendix A.3). The file uploaded has to be a .csv file and a warning will be given when the uploaded file does not have the .csv extension. Figure 4.7 shows all the warnings that will be given depending on the type of error the user makes. Furthermore, the physician can indicate whether the uploaded file contains ECG or respiratory information and the name of the patient whom this information belongs to. If the patient does not exist, the new patient will be added to the list below the upload button from where the physician can now also select him/her and review their data (Appendix A.2). This data will however only be stored locally in a subfolder accessible by the application.



(a) Notification when no file is selected when pushing the 'upload button'.



(b) Notification when the patients name is missing



(c) Notification when a file of the incorrect type is uploaded.

Figure 4.7: Notifications when uploading a file

Chapter 5

Results and Discussion

Due to the current circumstances, the whole system design could not be implemented. So, we are not able to verify some of the design choices that were made based on the programme of requirements, Section 2. Approximations regarding parameters as weight, battery life and computational power have been made, but cannot be confirmed.

5.1 Results

5.1.1 System Design results

The goal of the system was to design a non-invasive ECG and respiration monitoring system. The system should conform to physical as well as electrical attributes to ensure proper working. In the programme of requirements (PoR) section, Chapter 2, it is stated that the battery life of the system should be at least 7 days. For the final design, rechargeable batteries are recommended. These batteries offer a good resolution between weight/size and capacity. Literature and approximations point to the fact that such a battery should be able to sustain the system for at least 12 hours up to a day before having to be recharged. However, we are not capable of confirming this. The weight of the system is estimated to be well within the 250 gram maximum set in the PoR. However, this is also difficult to estimate due to the way it might be implemented, and impossible to confirm for now.

5.1.2 Graphical User Interface results

The Graphical User Interface (GUI) has actually been made. Such as, a comparison can be made between the final GUI and the expected goals from the PoR. The GUI needed to have a login screen with personal credentials to prevent unsolicited access to their data. Such a screen was made, with pre-determined username and password combinations that were hashed using the SHA-256 hashing mechanism. The GUI is able to display when a person is stressed according to stress detection [9]. A breathing exercise and a calming song are provided when this occurs. The heart rate of the patient and their filtered respiration rate are visible within the patient tab. Within the physician tab, the raw and processed ECG and respiration data are also visualized. A way of contacting the patient directly through the GUI via e-mail was implemented. Also, within the physician tab there is the option of adding extra local data. The requirement for remote storing and accessing of data is also met. The data is stored on a Microsoft Azure server [31].

It was stated that the system should be updated within certain time-frames to keep it as close to real time as possible. The achievements of the stress detection [9] and pre-processing [8] subgroups made this possible. The segment length was taken to be 80 seconds. So, the stress indicator is able to be updated every 80 seconds, and the ECG and respiration are updated every 10 seconds.

5.2 Discussion

There are a lot of different systems for the monitoring of ECG or respiration. There are little to no systems that try to do this at the same time. Every system, ours included has been designed with their respective goals in mind. Our goal being to try and monitor this data without interfering in the movement of daily lives.

Additional changes could be made within the system to further improve comfortability for daily use. For example, the number of electrodes could be reduced from three to two, but as this would increase the amount of noise and electromagnetic interference, components could have to be added that negate the benefits of removing it. It is also possible however that better pre-processing could deal with this amount of extra noise and make it work. This however would increase the computational costs and might inhibit the system of working on a wearable. A lot of these trade-offs could be seen when the system was designed. The choices made and components proposed are the options that in our eyes could ensure the system to work within the boundaries that were determined. These boundaries also state that the system should hinder the patient wearing it as little as possible. We are sure that improvements can be made within that part. The way the respiration strain sensor is now attached, directly to the skin and not processed into a shirt, definitely lowers the comfort. However, if the pre-processing and stress-detection parts would be able to recognize and process the respiration data from the ECG data, this part might be left out completely, to drastically improve the size and wearability of the system.

The proposed functionalities of the graphical user interface (GUI) are all achieved. Some of the functionalities are there as a proof-of-concept rather than a product to be used though. The way the login protection is achieved now is by using a hashing mechanism (SHA-256) that is not solely used within an actual GUI that handles private data, but in combination with a salt. The other functions are achieved as envisioned, even though they might not be as good looking as intended. The usage of Matlab App Designer restricts the appearance and functionalities of the GUI, as it is for example not very well designed for the use of writing to and reading from online storage.

Chapter 6

Conclusion

The designed system uses sensors, components and acquisition systems that are proven to work within the same applications. The patient should be able to wear the system without too much hinder in their daily lives. The real world applicability is expected, but to be tested.

The Graphical User Interface was developed and works as according to expectations and requirements. A login screen based on the SHA-256 hashing algorithm is made, which is good for developing purposes but not for real world applications. The GUI incorporates a patients heart rate, respiration data, stress indicator and a breathing exercise and calming song to calm down the patient when stressed. The physician GUI also includes the raw and filtered ECG data, quality factors, an option for contacting the patient as well as the option for adding local ECG or respiration data. The system integrates the pre-processing [8] and stress detection [9] parts well and uses them to process the needed data.

6.1 Recommendation & Future Work

Future work should focus on testing and improving the wearable system. The weight, size and comfort might need to be improved to maximize its value for the proposed application; constant monitoring without hinder. The options for extracting respiration data from the ECG to remove certain system parts needs to be looked into. It is recommended that Matlab App Designer is not to be used for the purpose of creating a visually pleasing GUI, and also for the reasons that it offers few options for integrating server capabilities.

Bibliography

[1] Mika Kivimäki et al. "Work stress and risk of cardiovascular mortality: prospective cohort study of industrial employees". In: *BMJ* 325.7369 (2002), p. 857. DOI: 10.1136/bmj.325.7369.857.

[2] A-M Bao, G Meynen, and DF Swaab. "The stress system in depression and neurodegeneration: focus on the human hypothalamus". In: *Brain research reviews* 57.2 (2008), pp. 531–553.

[3] Timo Heidt et al. "Chronic variable stress activates hematopoietic stem cells". In: *Nature medicine* 20.7 (2014), p. 754.

[4] Ebrahim Nemati, M. Deen, and Tapas Mondal. "A wireless wearable ECG sensor for long-term applications". In: *IEEE Communications Magazine* 50.1 (2012), pp. 36–43. ISSN: 0163-6804. DOI: 10.1109/mcom.2012.6122530.

[5] Ozkan Haydar et al. "A portable wearable tele-ECG monitoring system". In: *IEEE Transactions on Instrumentation and Measurement* 69 (2020), pp. 173–182. DOI: 10.1109/TIM.2019.2895484.

[6] Michael Chu et al. "Respiration rate and volume measurements using wearable strain sensors". In: *npj Digital Medicine* 2.1 (2019). ISSN: 2398-6352. DOI: 10.1038/s41746-019-0083-3. URL: https://dx.doi.org/10.1038/s41746-019-0083-3.

[7] Alberto Hernando et al. "Inclusion of Respiratory Frequency Information in Heart Rate Variability Analysis for Stress Assessment". In: *IEEE Journal of Biomedical and Health Informatics* 20 (Apr. 2016), pp. 1016–1025. DOI: 10.1109/JBHI.2016.2553578.

[8] Enes Kinaci and Talha Kuruoglu. "Thesis Pre-processing". In: (2020).

[9] Isar Meijer and Bob Morssink. "Stress Detection System Using ECG and Respiratory Signals." In: (2020).

[10] Joseph L. Hall and Deven McGraw. "For Telehealth To Succeed, Privacy And Security Risks Must Be Identified And Addressed". In: *Healt Affairs* (2014). DOI: 10.1377/hlthaff.2013.0997.

[11] *HIPAA regulations*. URL: https://www.cdc.gov/phlp/publications/topic/hipaa.html.

[12] *GDPR*. URL: https://gdpr-info.eu/.

[13] Fateme Moghbeli, Mostafa Langarizadeh, and Ali Aliabadi. "Application of Ethics for Providing Telemedicine Services and Information Technology". In: *Medical Archives* 71.5 (2017), pp. 351–355. DOI: 10.5455/medarh.2017.71.351-355.

[14] Branko Babusiak, Stefan Borik, and Maros Smondrk. "Two-Electrode ECG for Ambulatory Monitoring with Minimal Hardware Complexity". In: *Sensors* 20.8 (2020), p. 2386. ISSN: 1424-8220. DOI: 10.3390/s20082386.

[15] *Data Sheet AD7779*. URL: https://www.analog.com/media/en/technical-documentation/data-sheets/AD7779.pdf.

[16] Bruce B. Winter and John G. Webster. "Driven-right-leg circuit design". In: *IEEE Transactions on Biomedical Engineering* BME-30.1 (1983), pp. 62–66. ISSN: 0018-9294. DOI: 10.1109/tbme.1983.325168.

[17] *LMC6001 Ultra, Ultra-Low Input Current Amplifier*. URL: http://www.ti.com/lit/ds/symlink/lmc6001.pdf?ts=1591195580653.

[18] *INA106, Precision Gain = 10 DIFFERENTIAL AMPLIFIER*. URL: https://www.ti.com/lit/ds/symlink/ina106.pdf?ts=1591821972344&ref_url=https://www.ti.com/product/INA106.

[19] Paul Kligfield et al. "Recommendations for the Standardization and Interpretation of the Electrocardiogram". In: *Circulation* 115.10 (2007), pp. 1306–1324. ISSN: 0009-7322. DOI: 10.1161/circulationaha.106.180200.

[20] Taisa Daiana Da Costa et al. "Breathing Monitoring and Pattern Recognition with Wearable Sensors". In: (2019). DOI: 10.5772/intechopen.85460.

[21] Laura Mason. *Signal processing methods for non-invasive respiration monitoring.* University of Oxford Oxford, 2002.

[22] *Data Sheet MAX32665.* URL: https://datasheets.maximintegrated.com/en/ds/MAX32665-MAX32668.pdf.

[23] *TPS6104x Low-Power DC-DC Boost Converter in SOT-23 and WSON Packages.* URL: https://www.ti.com/lit/ds/symlink/tps61040.pdf?ts=1591976240582&ref_url=https%5C%253A%252F%252Fwww.ti.com%252Fproduct%252FTPS61040.

[24] *Quad-Channel Digital Isolators.* URL: https://www.analog.com/media/en/technical-documentation/data-sheets/ADuM2400_2401_2402.pdf.

[25] Bill Crone. “Mitigation Strategies for ECG Design Challenges”. In: *Analog Devices* (2011).

[26] *Cortex-M4 Technical Reference Manual.* URL: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0439b/DDI0439B_cortex_m4_r0p0_trm.pdf.

[27] *7th Generation Intel® Processor Family.* URL: https://cdn.cnetcontent.com/a5/fb/a5fbeb54-3807-4b86-a1fe-f28a862e0bc7.pdf.

[28] *5th Generation Intel® Core™ Processors Based on the Mobile U-Processor Line.* URL: https://www.intel.com/content/www/us/en/design/products-and-solutions/processors-and-chipsets/broadwell-u-y/technical-library.html?grouping=EMT_Content%5C%20Type&sort=title:asc.

[29] Ali Abedi, Omid Abari, and Tim Brecht. “Wi-LE: Can WiFi Replace Bluetooth?” In: (2019), pp. 117–124.

[30] Junxian Huang et al. “A close examination of performance and power characteristics of 4G LTE networks”. In: (2012), pp. 225–238.

[31] Adam Boeglin et al. *Building a telehealth system on Azure.* URL: https://docs.microsoft.com/en-us/azure/architecture/example-scenario/apps/telehealth-system.

[32] *Work with Remote Data.* URL: https://www.mathworks.com/help/matlab/import_export/work-with-remote-data.html#mw_11daa39e-c1f6-475d-927b-87dc94718b99.

[33] C. Varon et al. “A Comparative Study of ECG-derived Respiration in Ambulatory Monitoring using the Single-lead ECG”. In: *Scientific Reports* 10.1 (2020).

[34] Nicolas Sklavos and Odysseas Koufopavlou. “On the hardware implementations of the SHA-2 (256, 384, 512) hash functions”. In: *Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS'03.* Vol. 5. IEEE. 2003, pp. V–V.

[35] T Flenady, T Dwyer, and J Applegarth. “Accurate respiratory rates count: So should you!” In: *Australas Emerg Nurs J.* 20 (2017), pp. 45–47. DOI: https://doi.org/10.1016/j.aenj.2016.12.003.

[36] Monika Mourya et al. “Effect of slow-and fast-breathing exercises on autonomic functions in patients with essential hypertension”. In: *The journal of alternative and complementary medicine* 15.7 (2009), pp. 711–717.

[37] Myriam V Thoma et al. “The effect of music on the human stress response”. In: *PloS one* 8.8 (2013).

Appendix A

MATLAB Code

A.1 app1.m

```
classdef app1 < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                    matlab.ui.Figure
        VisGroup                    matlab.ui.container.TabGroup
        LoginTab                    matlab.ui.container.Tab
        GridLayout3                 matlab.ui.container.GridLayout
        UsernameLabel               matlab.ui.control.Label
        Usernamefield               matlab.ui.control.EditField
        Passwordfield               matlab.ui.control.EditField
        EnterButton                 matlab.ui.control.Button
        PasswordLabel               matlab.ui.control.Label
        EditFieldLabel              matlab.ui.control.Label
        truepassword                matlab.ui.control.EditField
        Image                       matlab.ui.control.Image
        PatientXTab                 matlab.ui.container.Tab
        GridLayout                  matlab.ui.container.GridLayout
        calmgifje1                  matlab.ui.control.Button
        sound                       matlab.ui.control.Button
        stresstext                  matlab.ui.control.EditField
        StressSwitch                matlab.ui.control.Switch
        redring                     matlab.ui.control.Image
        greenring                   matlab.ui.control.Image
        Button                      matlab.ui.control.Button
        HRshower                    matlab.ui.control.EditField
        BPMtext                     matlab.ui.control.EditField
        Repsax                      matlab.ui.control.UIAxes
        LogOutButton                matlab.ui.control.Button
        Image2                      matlab.ui.control.Image
        stresstext_2                matlab.ui.control.EditField
        PhysicianTab                matlab.ui.container.Tab
        GridLayout2                 matlab.ui.container.GridLayout
        Repsax2                     matlab.ui.control.UIAxes
        ECGax                       matlab.ui.control.UIAxes
        stressLamp                  matlab.ui.control.Lamp
        ContactPatientButton        matlab.ui.control.Button
        HRshower_2                  matlab.ui.control.EditField
        BPMtext_2                   matlab.ui.control.EditField
        ChoosePatientListBoxLabel   matlab.ui.control.Label
        ChoosePatientListBox        matlab.ui.control.ListBox
```

```
        QFshower                    matlab.ui.control.EditField
        BPMtext_3                   matlab.ui.control.EditField
        LogOutButton_2              matlab.ui.control.Button
        BPMtext_4                   matlab.ui.control.EditField
        RRshower                    matlab.ui.control.EditField
        StressLabel                 matlab.ui.control.Label
        Image3                      matlab.ui.control.Image
        contacttext                 matlab.ui.control.EditField
        ButtonGroup                 matlab.ui.container.ButtonGroup
        ECGButton                   matlab.ui.control.RadioButton
        RespiratoryButton           matlab.ui.control.RadioButton
        UploadButton                matlab.ui.control.Button
        ChooseFileButton            matlab.ui.control.Button
        PatientNameLabel            matlab.ui.control.Label
        PatientNameEditField        matlab.ui.control.EditField
        filenamepathtext            matlab.ui.control.EditField
        InvGroup                    matlab.ui.container.TabGroup
        Tab                         matlab.ui.container.Tab
    end


    properties (Access = public)
        % globa variables used troughout the whole system
        flag
        totaalgeenpersoonlijkeinformatie
        trueww2
        musicval = 1;
        welkenummerisdit
        filename
        path
        ECGsignal
        Respsignal
        fs_resp = 250;
        fs_ecg = 1000;
        repscounter
        stress
    end

    properties (Access = private)
        Property % Description
        player = ''; % initialize music player

    end

    methods (Access = public)



        function  musicfunc(app)
            try
                [y,Fs] = audioread('Allegri-Miserere.mp3');  %music which plays
                app.player =audioplayer(y,Fs);
            catch
            end
        end
    end

    methods (Access = private)
```

```
        function  login(app)
            onlineloginhalen(app); % check online for login data
            username = app.Usernamefield.Value; %username
            pw = string(app.truepassword.Value); %password
            loc = find(strcmp(app.totaalgeenpersoonlijkeinformatie(:,1), username)); %check
             ↪ for username in database
            haspw = DataHash(pw, 'SHA-256', 'Base64');
            if haspw == app.totaalgeenpersoonlijkeinformatie(loc,2) %check if pw is correct
                ID = str2double(app.totaalgeenpersoonlijkeinformatie(loc,3)); %display
                 ↪ relavent screen depending on logindata
                if ID == 35 % open physiciantab
                    app.LoginTab.Parent = app.InvGroup;
                    app.PatientXTab.Parent = app.InvGroup;
                    app.PhysicianTab.Parent = app.VisGroup;
                    app.VisGroup.SelectedTab = app.PhysicianTab;

                else % open patient tab
                    app.LoginTab.Parent = app.InvGroup;
                    app.PatientXTab.Parent = app.VisGroup;
                    app.PhysicianTab.Parent = app.InvGroup;
                    app.VisGroup.SelectedTab = app.PatientXTab;
                    app.PatientXTab.Title = string(ID);
                    j = 1;
                    counter = 0;
                    localornot = 1;
                    stapregelaar(app, j, counter, localornot , ID)
                end
            else
                opts = struct('WindowStyle','modal',...
                    'Interpreter','tex');
                errordlg('Incorrect Username or Password',...
                    'Error', opts);
            end

        end

        function  onlineloginhalen(app)
            setenv('MW_WASB_SAS_TOKEN',
 ↪ '?sv=2019-10-10&ss=bfqt&srt=sco&sp=rwdlacupx&se=2020-09-23T21:59:27Z&st=2020-05-11T13:59:27Z&spr=ht
            setenv('MW_WASB_SECRET_KEY',
             ↪ '+bp+3xekejItpGtJaw5e5g4ZkV1kuGL9XC2sqrbTlq8jd7eSUqw5MFg1bBRI9/4uaE/KeRo97gSPmdO9IrAV3Q
             ↪ %server keys to acces data
            loc = 'wasbs://werk@inloggegevens.blob.core.windows.net/'; %location of data
            loc  = append(loc, 'xxx2.csv'); %reorganize data
            ds = tabularTextDatastore(loc,'FileExtensions',{'.csv'});
            pf = read(ds);

            A = string(table2cell(pf));

            A(:,3) = append('0',A(:,3));


            app.totaalgeenpersoonlijkeinformatie = A;
        end
```

```
        function mainECGrunner(app, processed_signal, g, c, ~) %plot graphs
            %This function displays the ECG signal, heart rate, Respiratory signal and
            % respiratory rate depending on which screen is accessed. It also handles
            % errors which rise from NaN's and from the quality factors.

            ecgsignali = transpose(processed_signal{1,1}); %raw ECG signal
            ecgsignalf =transpose(processed_signal{3,1}); %filtered ECG signal

            respsignalf = transpose(processed_signal{9,1}); %filtered Reps signal
            s = 50;
            if app.VisGroup.SelectedTab == app.PhysicianTab
                try

                    nananan = isnan(processed_signal{1,1}); % checks for NaNs
                    nananan2 = find(nananan == 1);

                    app.RRshower.Value = string(60*(processed_signal{8,1})); %print HR
                    app.HRshower_2.Value = string(round(processed_signal{6,1}));
                    app.QFshower.Value = string(processed_signal{4,1});
                    if processed_signal{4,1} < 0.5 && app.VisGroup.SelectedTab == app.PhysicianTab
                        opts = struct('WindowStyle','modal',...
                            'Interpreter','tex');
                        warndlg('The QUALITY FACTOR seems to be low. Please be advised.',...
                            'Quality Factor Warning', opts);
                    end
                    if length(nananan2) > 100 && app.VisGroup.SelectedTab == app.PhysicianTab
                        opts = struct('WindowStyle','modal',...
                            'Interpreter','tex');
                        errordlg('There seems to be a problem with the sensor, check connectio',...
                            'Critical system failure', opts);

                    end

                    for i = 0 : s : 10*app.fs_resp -1
                        j = 4 *i;


                        if g == 0 && c== 0

                            xecg = -20 : 1/app.fs_ecg : -10 -1/app.fs_ecg;
                            yecgi = ecgsignali(1+ j : j+10*app.fs_ecg);
                            yecgf = ecgsignalf(1+ j : j+10*app.fs_ecg);


                        else
                            xecg = -20 : 1/app.fs_ecg : -10 - 1/app.fs_ecg;
                            yecgi = ecgsignali(1+ j : j+10*app.fs_ecg);
                            yecgf = ecgsignalf(1+ j : j+10*app.fs_ecg);

                        end
```

```
                        xreps = -40 : 1/app.fs_resp: -10 -1/app.fs_resp;
                        yrepsf = respsignalf(1+i : i + 30*app.fs_resp);


                        plot(app.ECGax, xecg, yecgi, xecg, yecgf)
                        plot(app.Repsax2, xreps, yrepsf);


                        pause(s/app.fs_resp)


                    end


                catch
                end
            elseif app.VisGroup.SelectedTab == app.PatientXTab
                app.HRshower.Value = string(round(processed_signal{6,1}));
                feedbackloop(app, processed_signal)
                for i = 0 : s : 10*app.fs_resp -1
                    xreps = -40 : 1/app.fs_resp: -10 -1/app.fs_resp;
                    yrepsf = respsignalf(1+i : i + 30*app.fs_resp);
                    plot(app.Repsax, xreps, yrepsf);

                    pause(s/app.fs_resp)
                end
            end



        end



        function add2list(app) %adds uploaded patient info to patientlist in the pysician
    ↪ tab

            try
            x2 = dir('extraECG');
            x = dir('extrareps');

            t1 = transpose(struct2cell(x));
            t2 = transpose(struct2cell(x2));
            try
                t1 = strrep(string(t1), '.csv','');
                t1 = strrep(t1, 'Resp_','');
                t1 = cellstr(t1);

                t2 = strrep(string(t2), '.csv','');
                t2 = strrep(t2, 'ECG_','');
                t2 = cellstr(t2);
            catch
            end

            xl = length(x2) -2 ;
            diffecg = setdiff(t1(:,1) , t2(:,1)) ;

```

```
            f = app.ChoosePatientListBox.Items;
            f2 = app.ChoosePatientListBox.ItemsData;

            i = 0; %#ok<NASGU>
            k = 0; %#ok<NASGU>
            for i = 1 : xl +1
                if i < xl+1
                    f2(34+i) = t2(i+2,1);
                    f(34+i) = t2(i+2,1);
                elseif i == xl +1
                    f2(i+34) = cellstr(blanks(1));
                end
            end

            if isempty(diffecg) == 0
                diffl1 = length(diffecg);
                for  k = 1 : diffl1 +1
                    if k < diffl1+1
                        f2(33 + i +k) =diffecg( k,1);
                        f(33+i +k) = diffecg( k,1);
                    elseif k == diffl1 + 1
                        f2( i + k +33) = cellstr(blanks(1));
                    end
                end

            end
            f = strrep(f, '.csv','');
            f = strrep(f, 'ECG_','');
            f = strrep(f, 'Resp_','');
            app.ChoosePatientListBox.Items = f;
            app.ChoosePatientListBox.ItemsData = f2;

            catch
            end
        end




        function stapregelaar(app, j, counter, localornot , patient)
            % this code checks which data needs to be downloaded from the
            % server for, dowloads it and sends the revelant part to
            % MainECGrunner to be displayed, after it has been processed in MainECG. IF the data is locally
            % availabe,nothing will be downloaded and will be processed
            % from the main storage directly
            %NOTE: when fully realized, either this function needs to be
            %redone, or the way data is read from the server needs to be
            %according to how it is read here.
            if localornot == 1

                app.ECGsignal = downloadECG(patient, j);
                app.Respsignal = downloadreps(patient, j);


            end
            try
```

```
        while app.VisGroup.SelectedTab == app.PhysicianTab | app.VisGroup.SelectedTab == app.PatientXTab %#ok<OR2>

            g = app.fs_ecg* 50 * (j-1); %50 sec of ecg steps
            c = app.fs_ecg * 10 * counter; % 10 sec sub steps ecg

            r = app.fs_resp*50*(j-1); %50sec resp steps
            eps = app.fs_resp*10*(counter+1);

            if g == 0 && c == 0
                ECGred = app.ECGsignal( g+ c + 1 : g+c+app.fs_ecg*20, :);
            else
                ECGred = app.ECGsignal( g+ c -app.fs_ecg*10 + 1 : g+c+app.fs_ecg*10, :);
            end

            if j == 1 && counter < 2 && localornot == 1
                Repsred = app.Respsignal(r + eps +1 : r + eps + app.fs_resp*10);
            else
                Repsred = app.Respsignal(r + eps - app.fs_resp*30 +1 : r + eps + app.fs_resp*10);

            end


            processedsignal = MainECG(0 , ECGred, length(ECGred), length(Repsred), Repsred);

            mainECGrunner(app, processedsignal, g, c, localornot)
            counter = counter + 1;


            if counter == 3 && localornot == 1
                t = j + 1;

                ecg = downloadECG(patient, t);
                ecgcomb = cat(1, app.ECGsignal , ecg);
                app.ECGsignal = ecgcomb;

                reps = downloadreps(patient , t);
                repscomb = cat(1, app.Respsignal, reps);
                app.Respsignal = repscomb;
            elseif counter  == 5
                counter = 0;
                j = j + 1;
            end

            if mod(5*j + c, 8) == 0
                app.stress = randi(2)-1;
                pause(2)
            end



        end
    catch
    end
end
```

```

        function feedbackloop(app, processed_signal)
            %Gives feedback on the breathing pattern of the patient
            %depending on how fast he is breathing.
            if app.stress == 1
                if (processed_signal{8,1}) < 0.9*(1/9)
                    app.stresstext_2 = 'You are breathing too slow. You are probably going to die.....';
                elseif (processed_signal{8,1}) >= 0.9*(1/9) && (processed_signal{8,1}) <= 2*(1/9)
                    app.stresstext_2 = 'You breathing is excellent! Keep up the pace!';
                elseif (processed_signal{8,1}) > 2*(1/9) && (processed_signal{8,1}) <= 2.5*(1/9)
                    app.stresstext_2 = 'You are still breathing a little bit too fast, maybe listening to a song will help you calm down a bit more?';
                elseif (processed_signal{8,1}) > 2.5*(1/9)
                    app.stresstext_2 = 'You are breathing too fast. Try matching you breaths with the exercise shown below';
                end
            else
                app.stresstext_2 = '';
            end
        end
    end




    % Callbacks that handle component events
    methods (Access = private)

        % Code that executes after component creation
        function startupFcn(app)
           %everything that needs to be initialized in the beginning can be
           %found here, such as server keys, folder locations etc.

            if isempty(gcp('nocreate'))
                parpool('local');
            end
            setenv('MW_WASB_SAS_TOKEN',
            ↪ '?sv=2019-10-10&ss=bfqt&srt=sco&sp=rwdlacupx&se=2020-09-23T21:59:27Z&st=2020-05-11T13:5
            setenv('MW_WASB_SECRET_KEY',
            ↪ '+bp+3xekejItpGtJaw5e5g4ZkV1kuGL9XC2sqrbTlq8jd7eSUqw5MFg1bBRI9/4uaE/KeRo97gSPmdO9IrAV3Q


            addpath('ECG_preprocess')
            addpath('extraECG')
            addpath('extrareps')
            app.LoginTab.Parent = app.VisGroup;
            app.PatientXTab.Parent = app.InvGroup;
            app.PhysicianTab.Parent = app.InvGroup;
            app.VisGroup.SelectedTab = app.LoginTab;

            musicfunc(app);
            app.flag = 0;
            set(app.greenring,'visible','on')
            set(app.redring,'visible','off')
            app.stresstext.Value = ['You are not stressed!' ...
```

```
                ' Keep it up!'];
            set(app.calmgifje1, 'visible', 'off')
            set(app.sound, 'visible', 'off')

            app.stress = 0;
        end

        % Value changed function: StressSwitch
        function StressSwitchValueChanged(app, event)
            value = string(app.StressSwitch.Value);
            app.stress = 1;
            if value == 'On' %#ok<BDSCA>
                set(app.redring,'visible','on')
                set(app.greenring,'visible','off')
                app.stresstext.Value = ['You appear to be stressed!' ...
                    ' Maybe this will help'];
                set(app.calmgifje1, 'visible', 'on')
                set(app.sound, 'visible', 'on')
            else
                set(app.greenring,'visible','on')
                set(app.redring,'visible','off')
                app.stresstext.Value = ['You are not stressed!' ...
                    ' Keep it up!'];
                set(app.calmgifje1, 'visible', 'off')
                set(app.sound, 'visible', 'off')
            end


        end

        % Button pushed function: sound
        function soundButtonPushed(app, event)
            % plays the music when the sound button is pushed
           if app.flag == 0
               if isplaying(app.player) == false
                   play(app.player)
                   app.flag = 0;
               else
                   pause(app.player)
                   app.flag = 1;
               end
           else
               resume(app.player)
               app.flag = 0;
           end
        end

        % Value changed function: ChoosePatientListBox
        function ChoosePatientListBoxValueChanged(app, event)
            %changes the patient information that is displayed, downloads
            %it when needed, or when it is local information just retrieves
            %it from the relavent folder
            app.ECGax.cla;
            app.Repsax2.cla;
            j = 1;
            counter = 0;
            app.repscounter = 0;
            app.contacttext.Value = '';
```

```
        id = (app.ChoosePatientListBox.Value);
        if str2double(id) < 35
            localornot = 1;
            counter = 0;
            j = 1;

        elseif isnan(str2double(id))
            idecg = append('ECG_', id, '.csv');
            idresp = append('Resp_', id, '.csv');
            %                 cd extraECG
            try
                app.ECGsignal = csvread(idecg);
            catch
                warndlg('No ECG signal can be found for this patient',...
                    'No signal');
                app.ECGsignal = zeros(100000, 1);
            end
            %                 cd ..\
            %                 cd extrareps
            try
                app.Respsignal = csvread(idresp);

            catch
                warndlg('No respiratory signal can be found for this patient',...
                    'No signal');
                app.Respsignal = zeros(1000000, 1);
            end
            %                 cd ..\
            localornot = 0;


        end

        stapregelaar(app, j, counter, localornot, id)

    end

    % Button pushed function: ContactPatientButton
    function ContactPatientButtonPushed(app, event)
        % when the button is pushed, a mail will be send to the patient
        % with a personalizable text
        mail = 'dontdiefromstress@gmail.com'; %Your GMail email address
        password = 'Ugoodm8?';  %Your GMail password
        setpref('Internet','SMTP_Server','smtp.gmail.com');
        setpref('Internet','E_mail', mail);
        setpref('Internet','SMTP_Username', mail);
        setpref('Internet','SMTP_Password', password);
        props = java.lang.System.getProperties;
        props.setProperty('mail.smtp.auth','true');
        props.setProperty('mail.smtp.socketFactory.class', 'javax.net.ssl.SSLSocketFactory');
        props.setProperty('mail.smtp.socketFactory.port','465');
        n = app.ChoosePatientListBox.Value;
        nn = 'Dear patient #';
        nnn = strcat(nn, n);
        nnnn = string(app.contacttext.Value);
        sendmail('yavuzhanmercimek@gmail.com',nnn, nnnn)
        message = strcat('Mail sent to Patient #', n);
```

```
            msgbox(message)
        end

        % Button pushed function: LogOutButton
        function LogOutButtonPushed(app, event)
            % return to loginscreen when pushed
            app.LoginTab.Parent = app.VisGroup;
            app.PatientXTab.Parent = app.InvGroup;
            app.PhysicianTab.Parent = app.InvGroup;
            app.VisGroup.SelectedTab = app.LoginTab;
            app.Passwordfield.Value = '';
            app.truepassword.Value = '';
            app.trueww2 = '';

        end

        % Button pushed function: LogOutButton_2
        function LogOutButton_2Pushed(app, event)
            % return to loginscreen when pushed
            app.LoginTab.Parent = app.VisGroup;
            app.PatientXTab.Parent = app.InvGroup;
            app.PhysicianTab.Parent = app.InvGroup;
            app.VisGroup.SelectedTab = app.LoginTab;
            app.Passwordfield.Value = '';
            app.truepassword.Value = '';

        end

        % Button pushed function: EnterButton
        function EnterButtonPushed(app, event)
                % runs these functions when pushed
            login(app)
           add2list(app)
        end

        % Value changing function: Passwordfield
        function PasswordfieldValueChanging(app, event)
            %Swaps the entered characters with *, and places them
            %characters in a invisible text box, from it will be read later
            %on.
            value = event.Value;

            newStr = erase(value,'*');
            l = length(value);
            n = '';
            for i = 0 : l-1

                n = strcat(n,'*');
            end
            app.Passwordfield.Value = n;
            if l == 1
                trueww = '';
                app.trueww2 = strcat(trueww, newStr);
            else
                app.trueww2 =strcat(app.trueww2, newStr);
            end

            app.truepassword.Value = app.trueww2;
```

```
        end

        % Button pushed function: ChooseFileButton
        function ChooseFileButtonPushed(app, event)
         %Opens explorer to select a file
          [app.filename, app.path] = uigetfile;
         app.filenamepathtext.Value = app.filename;


        end

        % Button pushed function: UploadButton
        function UploadButtonPushed(app, event)
            %Places files in a  folder depending on the type of information
            %in the file. Will give error if incorrect file type is
            %given.
            try
                [~, ~ , fExt] = fileparts(app.filename);
            catch
            end

            name = app.PatientNameEditField.Value;
            if isempty(fExt) == 1
                errordlg('No file was selected.',...
                    'No file');
            elseif convertCharsToStrings(fExt) == '.csv' %#ok<BDSCA>

                if app.ECGButton.Value == 1 && app.RespiratoryButton.Value == 0
                    namecat = strcat('ECG_', name, '.csv');
                    z = append(app.path, app.filename);
                    copyfile(z , 'extraECG')
                    cd 'extraECG'
                elseif app.RespiratoryButton.Value == 1 && app.ECGButton.Value == 0
                    namecat = strcat('Resp_', name, '.csv');
                    z = append(app.path, app.filename);
                    copyfile(z , 'extrareps')
                    cd 'extrareps'
                end


                if isempty(name) == 1
                    errordlg('Fill in the patients name!',...
                        'No name');
                elseif strcmp(app.filename, namecat) == 0
                    copyfile(app.filename, namecat)
                    delete(app.filename)
                end


            else
                errordlg('Upload only ''.csv'' files!',...
                    'Wrong file type');
            end

            app.filenamepathtext.Value = '';
            app.PatientNameEditField.Value = '';
```

```
            cd ..\
            add2list(app)
        end

        % Close request function: UIFigure
        function UIFigureCloseRequest(app, event)
           %closes parallel loop when closing the window
            delete(gcp('nocreate'))
            delete(app)

        end
    end

    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create UIFigure and hide until all components are created
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.Position = [100 100 1238 739];
            app.UIFigure.Name = 'MATLAB App';
            app.UIFigure.CloseRequestFcn = createCallbackFcn(app, @UIFigureCloseRequest,
            ↪  true);

            % Create VisGroup
            app.VisGroup = uitabgroup(app.UIFigure);
            app.VisGroup.Position = [1 0 1238 740];

            % Create LoginTab
            app.LoginTab = uitab(app.VisGroup);
            app.LoginTab.Title = 'Login';

            % Create GridLayout3
            app.GridLayout3 = uigridlayout(app.LoginTab);
            app.GridLayout3.ColumnWidth = {'2x', '1x', '1x', '1x', '2x'};
            app.GridLayout3.RowHeight = {'10x', '1x', '1x', '1.5x', '1.5x', '1x', '10x'};
            app.GridLayout3.ColumnSpacing = 12;
            app.GridLayout3.Padding = [12 10 12 10];

            % Create UsernameLabel
            app.UsernameLabel = uilabel(app.GridLayout3);
            app.UsernameLabel.HorizontalAlignment = 'right';
            app.UsernameLabel.Layout.Row = 4;
            app.UsernameLabel.Layout.Column = 2;
            app.UsernameLabel.Text = 'Username';

            % Create Usernamefield
            app.Usernamefield = uieditfield(app.GridLayout3, 'text');
            app.Usernamefield.Layout.Row = 4;
            app.Usernamefield.Layout.Column = 3;

            % Create Passwordfield
            app.Passwordfield = uieditfield(app.GridLayout3, 'text');
            app.Passwordfield.ValueChangingFcn = createCallbackFcn(app,
            ↪  @PasswordfieldValueChanging, true);
            app.Passwordfield.Layout.Row = 5;
```

```
            app.Passwordfield.Layout.Column = 3;

            % Create EnterButton
            app.EnterButton = uibutton(app.GridLayout3, 'push');
            app.EnterButton.ButtonPushedFcn = createCallbackFcn(app, @EnterButtonPushed,
            ↪  true);
            app.EnterButton.Layout.Row = 5;
            app.EnterButton.Layout.Column = 4;
            app.EnterButton.Text = 'Enter';

            % Create PasswordLabel
            app.PasswordLabel = uilabel(app.GridLayout3);
            app.PasswordLabel.HorizontalAlignment = 'right';
            app.PasswordLabel.Layout.Row = 5;
            app.PasswordLabel.Layout.Column = 2;
            app.PasswordLabel.Text = 'Password';

            % Create EditFieldLabel
            app.EditFieldLabel = uilabel(app.GridLayout3);
            app.EditFieldLabel.HorizontalAlignment = 'right';
            app.EditFieldLabel.Visible = 'off';
            app.EditFieldLabel.Layout.Row = 7;
            app.EditFieldLabel.Layout.Column = 1;
            app.EditFieldLabel.Text = 'Edit Field';

            % Create truepassword
            app.truepassword = uieditfield(app.GridLayout3, 'text');
            app.truepassword.Editable = 'off';
            app.truepassword.Visible = 'off';
            app.truepassword.Layout.Row = 7;
            app.truepassword.Layout.Column = 3;

            % Create Image
            app.Image = uiimage(app.GridLayout3);
            app.Image.Layout.Row = 1;
            app.Image.Layout.Column = 1;
            app.Image.ImageSource = 'Health2Go-01.png';

            % Create PatientXTab
            app.PatientXTab = uitab(app.VisGroup);
            app.PatientXTab.Title = 'Patient X';

            % Create GridLayout
            app.GridLayout = uigridlayout(app.PatientXTab);
            app.GridLayout.ColumnWidth = {42, 61, 60, 60, 60, 61, '1x', '1.2x', 70, 100, 70,
            ↪  '1.2x'};
            app.GridLayout.RowHeight = {22, 46, 54, 45, 74, 23, 63, 59, 74, '1x', 22,
            ↪  '2.51x'};
            app.GridLayout.ColumnSpacing = 10.3076923076923;
            app.GridLayout.RowSpacing = 4.30769230769231;
            app.GridLayout.Padding = [10.3076923076923 4.30769230769231 10.3076923076923
            ↪  4.30769230769231];

            % Create calmgifje1
            app.calmgifje1 = uibutton(app.GridLayout, 'push');
            app.calmgifje1.Icon = 'calmgifje1.gif';
            app.calmgifje1.IconAlignment = 'center';
            app.calmgifje1.Layout.Row = [5 9];
```

```
        app.calmgifje1.Layout.Column = [9 11];
        app.calmgifje1.Text = '';

        % Create sound
        app.sound = uibutton(app.GridLayout, 'push');
        app.sound.ButtonPushedFcn = createCallbackFcn(app, @soundButtonPushed, true);
        app.sound.Icon = 'playpauseicon.png';
        app.sound.FontWeight = 'bold';
        app.sound.Layout.Row = 11;
        app.sound.Layout.Column = 10;
        app.sound.Text = 'Play/Pause';

        % Create stresstext
        app.stresstext = uieditfield(app.GridLayout, 'text');
        app.stresstext.Editable = 'off';
        app.stresstext.FontSize = 30;
        app.stresstext.FontWeight = 'bold';
        app.stresstext.BackgroundColor = [0.9412 0.9412 0.9412];
        app.stresstext.Layout.Row = [1 2];
        app.stresstext.Layout.Column = [8 12];

        % Create StressSwitch
        app.StressSwitch = uiswitch(app.GridLayout, 'slider');
        app.StressSwitch.ValueChangedFcn = createCallbackFcn(app,
        ↪  @StressSwitchValueChanged, true);
        app.StressSwitch.Layout.Row = 1;
        app.StressSwitch.Layout.Column = [3 4];

        % Create redring
        app.redring = uiimage(app.GridLayout);
        app.redring.Layout.Row = [2 7];
        app.redring.Layout.Column = [2 6];
        app.redring.ImageSource = 'redring.png';

        % Create greenring
        app.greenring = uiimage(app.GridLayout);
        app.greenring.Layout.Row = [2 7];
        app.greenring.Layout.Column = [2 6];
        app.greenring.ImageSource = 'greenring (2).png';

        % Create Button
        app.Button = uibutton(app.GridLayout, 'push');
        app.Button.Icon = 'giphy.gif';
        app.Button.IconAlignment = 'center';
        app.Button.Layout.Row = 3;
        app.Button.Layout.Column = 4;
        app.Button.Text = '';

        % Create HRshower
        app.HRshower = uieditfield(app.GridLayout, 'text');
        app.HRshower.Editable = 'off';
        app.HRshower.HorizontalAlignment = 'center';
        app.HRshower.FontSize = 100;
        app.HRshower.FontWeight = 'bold';
        app.HRshower.BackgroundColor = [0.9412 0.9412 0.9412];
        app.HRshower.Layout.Row = [4 5];
        app.HRshower.Layout.Column = [3 5];

```

```
            % Create BPMtext
            app.BPMtext = uieditfield(app.GridLayout, 'text');
            app.BPMtext.Editable = 'off';
            app.BPMtext.HorizontalAlignment = 'center';
            app.BPMtext.FontWeight = 'bold';
            app.BPMtext.BackgroundColor = [0.9412 0.9412 0.9412];
            app.BPMtext.Layout.Row = 6;
            app.BPMtext.Layout.Column = 4;
            app.BPMtext.Value = 'BPM';

            % Create Repsax
            app.Repsax = uiaxes(app.GridLayout);
            title(app.Repsax, 'Respiratory signal')
            xlabel(app.Repsax, 'Time (s)')
            ylabel(app.Repsax, 'Magnitude')
            app.Repsax.PlotBoxAspectRatio = [2.32167832167832 1 1];
            app.Repsax.XLim = [-40 -10];
            app.Repsax.YTick = [];
            app.Repsax.YGrid = 'on';
            app.Repsax.Layout.Row = [9 12];
            app.Repsax.Layout.Column = [1 7];

            % Create LogOutButton
            app.LogOutButton = uibutton(app.GridLayout, 'push');
            app.LogOutButton.ButtonPushedFcn = createCallbackFcn(app, @LogOutButtonPushed,
            ↪ true);
            app.LogOutButton.Layout.Row = 1;
            app.LogOutButton.Layout.Column = [1 2];
            app.LogOutButton.Text = 'Log Out';

            % Create Image2
            app.Image2 = uiimage(app.GridLayout);
            app.Image2.Layout.Row = 12;
            app.Image2.Layout.Column = 12;
            app.Image2.ImageSource = 'Health2Go-01.png';

            % Create stresstext_2
            app.stresstext_2 = uieditfield(app.GridLayout, 'text');
            app.stresstext_2.Editable = 'off';
            app.stresstext_2.FontSize = 30;
            app.stresstext_2.FontWeight = 'bold';
            app.stresstext_2.BackgroundColor = [0.9412 0.9412 0.9412];
            app.stresstext_2.Layout.Row = 3;
            app.stresstext_2.Layout.Column = [8 12];

            % Create PhysicianTab
            app.PhysicianTab = uitab(app.VisGroup);
            app.PhysicianTab.Title = 'Physician';

            % Create GridLayout2
            app.GridLayout2 = uigridlayout(app.PhysicianTab);
            app.GridLayout2.ColumnWidth = {43, '1.33x', 100, '1x', 130, '1.12x', 79, 60, 79,
            ↪ 104, 93, '1x', 68, 20, 29, 41};
            app.GridLayout2.RowHeight = {22, '1.5x', 17, 28, '1.88x', 69, 69, 39, '2.5x',
            ↪ '1.31x', 40, '1x', 22, 14, 43, 50, 18};
            app.GridLayout2.ColumnSpacing = 4.22222222222222;
            app.GridLayout2.RowSpacing = 7.13333333333333;
```

```
            app.GridLayout2.Padding = [4.22222222222222 7.13333333333333 4.22222222222222
            ↪  7.13333333333333];

            % Create Repsax2
            app.Repsax2 = uiaxes(app.GridLayout2);
            title(app.Repsax2, 'Respiratory signal')
            xlabel(app.Repsax2, 'Time (s)')
            ylabel(app.Repsax2, 'Magnitude')
            app.Repsax2.PlotBoxAspectRatio = [1.87458745874587 1 1];
            app.Repsax2.XLim = [-40 -10];
            app.Repsax2.YTick = [];
            app.Repsax2.YGrid = 'on';
            app.Repsax2.Layout.Row = [10 17];
            app.Repsax2.Layout.Column = [1 5];

            % Create ECGax
            app.ECGax = uiaxes(app.GridLayout2);
            title(app.ECGax, 'ECG signal')
            xlabel(app.ECGax, 'Time (s)')
            ylabel(app.ECGax, 'Magnitude')
            app.ECGax.PlotBoxAspectRatio = [1.85947712418301 1 1];
            app.ECGax.XLim = [-20 -10];
            app.ECGax.YTick = [];
            app.ECGax.YGrid = 'on';
            app.ECGax.Layout.Row = [3 8];
            app.ECGax.Layout.Column = [1 5];

            % Create stressLamp
            app.stressLamp = uilamp(app.GridLayout2);
            app.stressLamp.Layout.Row = [10 12];
            app.stressLamp.Layout.Column = [13 16];

            % Create ContactPatientButton
            app.ContactPatientButton = uibutton(app.GridLayout2, 'push');
            app.ContactPatientButton.ButtonPushedFcn = createCallbackFcn(app,
            ↪  @ContactPatientButtonPushed, true);
            app.ContactPatientButton.BackgroundColor = [0.9412 0.9412 0.9412];
            app.ContactPatientButton.FontSize = 26;
            app.ContactPatientButton.FontWeight = 'bold';
            app.ContactPatientButton.FontColor = [1 0 0];
            app.ContactPatientButton.Layout.Row = [10 11];
            app.ContactPatientButton.Layout.Column = [10 12];
            app.ContactPatientButton.Text = 'Contact Patient';

            % Create HRshower_2
            app.HRshower_2 = uieditfield(app.GridLayout2, 'text');
            app.HRshower_2.Editable = 'off';
            app.HRshower_2.HorizontalAlignment = 'center';
            app.HRshower_2.FontSize = 100;
            app.HRshower_2.FontWeight = 'bold';
            app.HRshower_2.BackgroundColor = [0.9412 0.9412 0.9412];
            app.HRshower_2.Layout.Row = [5 6];
            app.HRshower_2.Layout.Column = [6 8];

            % Create BPMtext_2
            app.BPMtext_2 = uieditfield(app.GridLayout2, 'text');
            app.BPMtext_2.Editable = 'off';
            app.BPMtext_2.HorizontalAlignment = 'center';
```

```
            app.BPMtext_2.FontWeight = 'bold';
            app.BPMtext_2.BackgroundColor = [0.9412 0.9412 0.9412];
            app.BPMtext_2.Layout.Row = 4;
            app.BPMtext_2.Layout.Column = [6 8];
            app.BPMtext_2.Value = 'BPM';

            % Create ChoosePatientListBoxLabel
            app.ChoosePatientListBoxLabel = uilabel(app.GridLayout2);
            app.ChoosePatientListBoxLabel.HorizontalAlignment = 'right';
            app.ChoosePatientListBoxLabel.FontWeight = 'bold';
            app.ChoosePatientListBoxLabel.Layout.Row = 5;
            app.ChoosePatientListBoxLabel.Layout.Column = 11;
            app.ChoosePatientListBoxLabel.Text = 'Choose Patient';

            % Create ChoosePatientListBox
            app.ChoosePatientListBox = uilistbox(app.GridLayout2);
            app.ChoosePatientListBox.Items = {' Patient 1', ' Patient 2', ' Patient 3', '
            ↪ Patient 4', ' Patient 5', ' Patient 6', ' Patient 7', ' Patient 8', '
            ↪ Patient 9', ' Patient 10', ' Patient 11', ' Patient 12', ' Patient 13', '
            ↪ Patient 14', ' Patient 15', ' Patient 16', ' Patient 17', ' Patient 18', '
            ↪ Patient 19', ' Patient 20', ' Patient 21', 'Patient 22', 'Patient 23',
            ↪ 'Patient 24', 'Patient 25', 'Patient 26', 'Patient 27', 'Patient 28',
            ↪ 'Patient 29', 'Patient 30', 'Patient 31', 'Patient 32', 'Patient 33',
            ↪ 'Patient 34'};
            app.ChoosePatientListBox.ItemsData = {'1', '2', '3', '4', '5', '6', '7', '8',
            ↪ '9', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '20', '21',
            ↪ '22', '23', '24', '25', '26', '27', '28', '29', '30', '31', '32', '33',
            ↪ '34', ''};
            app.ChoosePatientListBox.ValueChangedFcn = createCallbackFcn(app,
            ↪ @ChoosePatientListBoxValueChanged, true);
            app.ChoosePatientListBox.Layout.Row = [5 7];
            app.ChoosePatientListBox.Layout.Column = [12 15];
            app.ChoosePatientListBox.Value = '1';

            % Create QFshower
            app.QFshower = uieditfield(app.GridLayout2, 'text');
            app.QFshower.Editable = 'off';
            app.QFshower.HorizontalAlignment = 'center';
            app.QFshower.FontSize = 80;
            app.QFshower.FontWeight = 'bold';
            app.QFshower.BackgroundColor = [0.9412 0.9412 0.9412];
            app.QFshower.Layout.Row = [5 6];
            app.QFshower.Layout.Column = 9;

            % Create BPMtext_3
            app.BPMtext_3 = uieditfield(app.GridLayout2, 'text');
            app.BPMtext_3.Editable = 'off';
            app.BPMtext_3.HorizontalAlignment = 'center';
            app.BPMtext_3.FontWeight = 'bold';
            app.BPMtext_3.BackgroundColor = [0.9412 0.9412 0.9412];
            app.BPMtext_3.Layout.Row = 4;
            app.BPMtext_3.Layout.Column = 9;
            app.BPMtext_3.Value = 'QF';

            % Create LogOutButton_2
            app.LogOutButton_2 = uibutton(app.GridLayout2, 'push');
            app.LogOutButton_2.ButtonPushedFcn = createCallbackFcn(app,
            ↪ @LogOutButton_2Pushed, true);
```

```
        app.LogOutButton_2.Layout.Row = 1;
        app.LogOutButton_2.Layout.Column = 2;
        app.LogOutButton_2.Text = 'Log Out';

        % Create BPMtext_4
        app.BPMtext_4 = uieditfield(app.GridLayout2, 'text');
        app.BPMtext_4.Editable = 'off';
        app.BPMtext_4.HorizontalAlignment = 'center';
        app.BPMtext_4.FontWeight = 'bold';
        app.BPMtext_4.BackgroundColor = [0.9412 0.9412 0.9412];
        app.BPMtext_4.Layout.Row = 11;
        app.BPMtext_4.Layout.Column = [6 8];
        app.BPMtext_4.Value = 'RPM';

        % Create RRshower
        app.RRshower = uieditfield(app.GridLayout2, 'text');
        app.RRshower.Editable = 'off';
        app.RRshower.HorizontalAlignment = 'center';
        app.RRshower.FontSize = 92;
        app.RRshower.FontWeight = 'bold';
        app.RRshower.BackgroundColor = [0.9412 0.9412 0.9412];
        app.RRshower.Layout.Row = [12 15];
        app.RRshower.Layout.Column = [6 8];

        % Create StressLabel
        app.StressLabel = uilabel(app.GridLayout2);
        app.StressLabel.HorizontalAlignment = 'center';
        app.StressLabel.FontSize = 36;
        app.StressLabel.FontWeight = 'bold';
        app.StressLabel.Layout.Row = [8 9];
        app.StressLabel.Layout.Column = [13 16];
        app.StressLabel.Text = 'Stress?';

        % Create Image3
        app.Image3 = uiimage(app.GridLayout2);
        app.Image3.Layout.Row = [15 17];
        app.Image3.Layout.Column = [13 16];
        app.Image3.ImageSource = 'Health2Go-01.png';

        % Create contacttext
        app.contacttext = uieditfield(app.GridLayout2, 'text');
        app.contacttext.Layout.Row = [12 16];
        app.contacttext.Layout.Column = [10 12];

        % Create ButtonGroup
        app.ButtonGroup = uibuttongroup(app.GridLayout2);
        app.ButtonGroup.Layout.Row = [1 4];
        app.ButtonGroup.Layout.Column = [11 16];

        % Create ECGButton
        app.ECGButton = uiradiobutton(app.ButtonGroup);
        app.ECGButton.Text = 'ECG';
        app.ECGButton.Position = [254 87 58 22];
        app.ECGButton.Value = true;

        % Create RespiratoryButton
        app.RespiratoryButton = uiradiobutton(app.ButtonGroup);
        app.RespiratoryButton.Text = 'Respiratory';
```

```
            app.RespiratoryButton.Position = [254 66 83 22];

            % Create UploadButton
            app.UploadButton = uibutton(app.ButtonGroup, 'push');
            app.UploadButton.ButtonPushedFcn = createCallbackFcn(app, @UploadButtonPushed,
            ↪  true);
            app.UploadButton.Position = [242 21 79 22];
            app.UploadButton.Text = 'Upload';

            % Create ChooseFileButton
            app.ChooseFileButton = uibutton(app.ButtonGroup, 'push');
            app.ChooseFileButton.ButtonPushedFcn = createCallbackFcn(app,
            ↪  @ChooseFileButtonPushed, true);
            app.ChooseFileButton.Position = [11 87 82 22];
            app.ChooseFileButton.Text = 'Choose File';

            % Create PatientNameLabel
            app.PatientNameLabel = uilabel(app.ButtonGroup);
            app.PatientNameLabel.HorizontalAlignment = 'center';
            app.PatientNameLabel.Position = [11 47 80.0918851435706 22];
            app.PatientNameLabel.Text = 'Patient Name';

            % Create PatientNameEditField
            app.PatientNameEditField = uieditfield(app.ButtonGroup, 'text');
            app.PatientNameEditField.Position = [107 47 87 22];

            % Create filenamepathtext
            app.filenamepathtext = uieditfield(app.ButtonGroup, 'text');
            app.filenamepathtext.Editable = 'off';
            app.filenamepathtext.BackgroundColor = [0.9412 0.9412 0.9412];
            app.filenamepathtext.Position = [107 87 86 22];

            % Create InvGroup
            app.InvGroup = uitabgroup(app.UIFigure);
            app.InvGroup.Position = [470 -124 100 30];

            % Create Tab
            app.Tab = uitab(app.InvGroup);
            app.Tab.Title = 'Tab';

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = app1

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            % Execute the startup function
```

```
            runStartupFcn(app, @startupFcn)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```

A.2 add2list.m

```
function add2list(app) %adds uploaded patient info to patientlist in the pysician tab

try
    x2 = dir('extraECG');
    x = dir('extrareps');

    t1 = transpose(struct2cell(x));
    t2 = transpose(struct2cell(x2));
    try
        t1 = strrep(string(t1), '.csv','');
        t1 = strrep(t1, 'Resp_','');
        t1 = cellstr(t1);

        t2 = strrep(string(t2), '.csv','');
        t2 = strrep(t2, 'ECG_','');
        t2 = cellstr(t2);
    catch
    end

    xl = length(x2) -2 ;
    diffecg = setdiff(t1(:,1) , t2(:,1)) ;

    f = app.ChoosePatientListBox.Items;
    f2 = app.ChoosePatientListBox.ItemsData;

    i = 0; %#ok<NASGU>
    k = 0; %#ok<NASGU>
    for i = 1 : xl +1
        if i < xl+1
            f2(34+i) = t2(i+2,1);
            f(34+i) = t2(i+2,1);
        elseif i == xl +1
            f2(i+34) = cellstr(blanks(1));
        end
    end

    if isempty(diffecg) == 0
        diffl1 = length(diffecg);
        for  k = 1 : diffl1 +1
            if k < diffl1+1
```

```
                f2(33 + i +k) =diffecg( k,1);
                f(33+i +k) = diffecg( k,1);
            elseif k == diffl1 + 1
                f2( i + k +33) = cellstr(blanks(1));
            end
        end

    end
    f = strrep(f, '.csv','');
    f = strrep(f, 'ECG_','');
    f = strrep(f, 'Resp_','');
    app.ChoosePatientListBox.Items = f;
    app.ChoosePatientListBox.ItemsData = f2;

catch
end
end
```

A.3 ChooseFileButtonPushed.m

```
function ChooseFileButtonPushed(app, event)
%Opens explorer to select a file
[app.filename, app.path] = uigetfile;
app.filenamepathtext.Value = app.filename;


end
```

A.4 ChoosePatientListBoxValueChanged.m

```
function ChoosePatientListBoxValueChanged(app, event)
%changes the patient information that is displayed, downloads
%it when needed, or when it is local information just retrieves
%it from the relavent folder
app.ECGax.cla;
app.Repsax2.cla;
j = 1;
counter = 0;
app.repscounter = 0;
app.contacttext.Value = '';
id = (app.ChoosePatientListBox.Value);
if str2double(id) < 35
    localornot = 1;
    counter = 0;
    j = 1;

elseif isnan(str2double(id))
    idecg = append('ECG_', id, '.csv');
    idresp = append('Resp_', id, '.csv');
    %                 cd extraECG
    try
        app.ECGsignal = csvread(idecg);
    catch
        warndlg('No ECG signal can be found for this patient',...
            'No signal');
        app.ECGsignal = zeros(100000, 1);
    end
```

```
    %                 cd ..\
    %                 cd extrareps
    try
        app.Respsignal = csvread(idresp);

    catch
        warndlg('No respiratory signal can be found for this patient',...
            'No signal');
        app.Respsignal = zeros(1000000, 1);
    end
    %                 cd ..\
    localornot = 0;


end

stapregelaar(app, j, counter, localornot, id)

end
```

A.5 ContactPatientButtonPushed.m

```
function ContactPatientButtonPushed(app, event)
% when the button is pushed, a mail will be send to the patient
% with a personalizable text
mail = 'dontdiefromstress@gmail.com'; %Your GMail email address
password = 'Ugoodm8?';  %Your GMail password
setpref('Internet','SMTP_Server','smtp.gmail.com');
setpref('Internet','E_mail', mail);
setpref('Internet','SMTP_Username', mail);
setpref('Internet','SMTP_Password', password);
props = java.lang.System.getProperties;
props.setProperty('mail.smtp.auth','true');
props.setProperty('mail.smtp.socketFactory.class', 'javax.net.ssl.SSLSocketFactory');
props.setProperty('mail.smtp.socketFactory.port','465');
n = app.ChoosePatientListBox.Value;
nn = 'Dear patient #';
nnn = strcat(nn, n);
nnnn = string(app.contacttext.Value);
sendmail('yavuzhanmercimek@gmail.com',nnn, nnnn)
message = strcat('Mail sent to Patient #', n);
msgbox(message)
end
```

A.6 add2list.m

```
function feedbackloop(app, processed_signal)
%Gives feedback on the breathing pattern of the patient
%depending on how fast he is breathing.
if app.stress == 1
    if (processed_signal{8,1}) < 0.9*(1/9)
        app.stresstext_2 = 'You are breathing too slow. You are probably going to die.....';
    elseif (processed_signal{8,1}) >= 0.9*(1/9) && (processed_signal{8,1}) <= 2*(1/9)
        app.stresstext_2 = 'You breathing is excellent! Keep up the pace!';
    elseif (processed_signal{8,1}) > 2*(1/9) && (processed_signal{8,1}) <= 2.5*(1/9)
        app.stresstext_2 = 'You are still breathing a little bit too fast, maybe listening
         ↪  to a song will help you calm down a bit more?';
```

```
    elseif (processed_signal{8,1}) > 2.5*(1/9)
        app.stresstext_2 = 'You are breathing too fast. Try matching you breaths with the
         ↪  exercise shown below';
    end
else
    app.stresstext_2 = '';
end
end
end
```

A.7 login.m

```
function  login(app)
onlineloginhalen(app); % check online for login data
username = app.Usernamefield.Value; %username
pw = string(app.truepassword.Value); %password
loc = find(strcmp(app.totaalgeenpersoonlijkeinformatie(:,1), username)); %check for username
 ↪  in database
haspw = DataHash(pw, 'SHA-256', 'Base64');
if haspw == app.totaalgeenpersoonlijkeinformatie(loc,2) %check if pw is correct
    ID = str2double(app.totaalgeenpersoonlijkeinformatie(loc,3)); %display relavent screen
     ↪  depending on logindata
    if ID == 35 % open physiciantab
        app.LoginTab.Parent = app.InvGroup;
        app.PatientXTab.Parent = app.InvGroup;
        app.PhysicianTab.Parent = app.VisGroup;
        app.VisGroup.SelectedTab = app.PhysicianTab;

    else % open patient tab
        app.LoginTab.Parent = app.InvGroup;
        app.PatientXTab.Parent = app.VisGroup;
        app.PhysicianTab.Parent = app.InvGroup;
        app.VisGroup.SelectedTab = app.PatientXTab;
        app.PatientXTab.Title = string(ID);
        j = 1;
        counter = 0;
        localornot = 1;
        stapregelaar(app, j, counter, localornot , ID)
    end
else
    opts = struct('WindowStyle','modal',...
        'Interpreter','tex');
    errordlg('Incorrect Username or Password',...
        'Error', opts);
end

end
```

A.8 LogOutButton_2Pushed.m

```
function LogOutButton_2Pushed(app, event)
% return to loginscreen when pushed
app.LoginTab.Parent = app.VisGroup;
app.PatientXTab.Parent = app.InvGroup;
app.PhysicianTab.Parent = app.InvGroup;
app.VisGroup.SelectedTab = app.LoginTab;
app.Passwordfield.Value = '';
```

```
app.truepassword.Value = '';

end
```

A.9 LogOutButtonPushed.m

```
function LogOutButtonPushed(app, event)
% return to loginscreen when pushed
app.LoginTab.Parent = app.VisGroup;
app.PatientXTab.Parent = app.InvGroup;
app.PhysicianTab.Parent = app.InvGroup;
app.VisGroup.SelectedTab = app.LoginTab;
app.Passwordfield.Value = '';
app.truepassword.Value = '';
app.trueww2 = '';

end
```

A.10 mainECGrunner.m

```
function mainECGrunner(app, processed_signal, g, c, ~) %plot graphs
%This function displays the ECG signal, heart rate, Respiratory signal and
% respiratory rate depending on which screen is accessed. It also handles
% errors which rise from NaN's and from the quality factors.

ecgsignali = transpose(processed_signal{1,1}); %raw ECG signal
ecgsignalf =transpose(processed_signal{3,1}); %filtered ECG signal

respsignalf = transpose(processed_signal{9,1}); %filtered Reps signal
s = 50;
if app.VisGroup.SelectedTab == app.PhysicianTab
    try

        nananan = isnan(processed_signal{1,1}); % checks for NaNs
        nananan2 = find(nananan == 1);

        app.RRshower.Value = string(60*(processed_signal{8,1})); %print HR
        app.HRshower_2.Value = string(round(processed_signal{6,1}));
        app.QFshower.Value = string(processed_signal{4,1});
        if processed_signal{4,1} < 0.5 && app.VisGroup.SelectedTab == app.PhysicianTab
            opts = struct('WindowStyle','modal',...
                'Interpreter','tex');
            warndlg('The QUALITY FACTOR seems to be low. Please be advised.',...
                'Quality Factor Warning', opts);
        end
        if length(nananan2) > 100 && app.VisGroup.SelectedTab == app.PhysicianTab
            opts = struct('WindowStyle','modal',...
                'Interpreter','tex');
            errordlg('There seems to be a problem with the sensor, check connectio',...
                'Critical system failure', opts);

        end

        for i = 0 : s : 10*app.fs_resp -1
            j = 4 *i;


```

```
            if g == 0 && c== 0

                xecg = -20 : 1/app.fs_ecg : -10 -1/app.fs_ecg;
                yecgi = ecgsignali(1+ j : j+10*app.fs_ecg);
                yecgf = ecgsignalf(1+ j : j+10*app.fs_ecg);


            else
                xecg = -20 : 1/app.fs_ecg : -10 - 1/app.fs_ecg;
                yecgi = ecgsignali(1+ j : j+10*app.fs_ecg);
                yecgf = ecgsignalf(1+ j : j+10*app.fs_ecg);

            end

            xreps = -40 : 1/app.fs_resp: -10 -1/app.fs_resp;
            yrepsf = respsignalf(1+i : i + 30*app.fs_resp);


            plot(app.ECGax, xecg, yecgi, xecg, yecgf)
            plot(app.Repsax2, xreps, yrepsf);


            pause(s/app.fs_resp)


        end


    catch
    end
elseif app.VisGroup.SelectedTab == app.PatientXTab
    app.HRshower.Value = string(round(processed_signal{6,1}));
    feedbackloop(app, processed_signal)
    for i = 0 : s : 10*app.fs_resp -1
        xreps = -40 : 1/app.fs_resp: -10 -1/app.fs_resp;
        yrepsf = respsignalf(1+i : i + 30*app.fs_resp);
        plot(app.Repsax, xreps, yrepsf);

        pause(s/app.fs_resp)
    end
end



end
```

A.11 musicfunc.m

```
        function  musicfunc(app)
            try
                [y,Fs] = audioread('Allegri-Miserere.mp3');  %music which plays
                app.player =audioplayer(y,Fs);
            catch
            end
        end
    end
```

A.12 onlineloginhalen.m

```
function  onlineloginhalen(app)
setenv('MW_WASB_SAS_TOKEN',
    '?sv=2019-10-10&ss=bfqt&srt=sco&sp=rwdlacupx&se=2020-09-23T21:59:27Z&st=2020-05-11T13:59:27Z&spr=ht
setenv('MW_WASB_SECRET_KEY',
    '+bp+3xekejItpGtJaw5e5g4ZkV1kuGL9XC2sqrbTlq8jd7eSUqw5MFg1bBRI9/4uaE/KeRo97gSPmdO9IrAV3Q==')
    %server keys to acces data
loc = 'wasbs://werk@inloggegevens.blob.core.windows.net/'; %location of data
loc  = append(loc, 'xxx2.csv'); %reorganize data
ds = tabularTextDatastore(loc,'FileExtensions',{'.csv'});
pf = read(ds);

A = string(table2cell(pf));

A(:,3) = append('0',A(:,3));


app.totaalgeenpersoonlijkeinformatie = A;
end
```

A.13 PasswordfieldValueChanging.m

```
function PasswordfieldValueChanging(app, event)
%Swaps the entered characters with *, and places them
%characters in a invisible text box, from it will be read later
%on.
value = event.Value;

newStr = erase(value,'*');
l = length(value);
n = '';
for i = 0 : l-1

    n = strcat(n,'*');
end
app.Passwordfield.Value = n;
if l == 1
    trueww = '';
    app.trueww2 = strcat(trueww, newStr);
else
    app.trueww2 =strcat(app.trueww2, newStr);
end

app.truepassword.Value = app.trueww2;


end
```

A.14 soundButtonPushed.m

```
function soundButtonPushed(app, event)
% plays the music when the sound button is pushed
if app.flag == 0
    if isplaying(app.player) == false
        play(app.player)
        app.flag = 0;
```

```
    else
        pause(app.player)
        app.flag = 1;
    end
else
    resume(app.player)
    app.flag = 0;
end
end
```

A.15 PasswordfieldValueChanging.m

```
function PasswordfieldValueChanging(app, event)
%Swaps the entered characters with *, and places them
%characters in a invisible text box, from it will be read later
%on.
value = event.Value;

newStr = erase(value,'*');
l = length(value);
n = '';
for i = 0 : l-1

    n = strcat(n,'*');
end
app.Passwordfield.Value = n;
if l == 1
    trueww = '';
    app.trueww2 = strcat(trueww, newStr);
else
    app.trueww2 =strcat(app.trueww2, newStr);
end

app.truepassword.Value = app.trueww2;


end
```

A.16 soundButtonPushed.m

```
function soundButtonPushed(app, event)
% plays the music when the sound button is pushed
if app.flag == 0
    if isplaying(app.player) == false
        play(app.player)
        app.flag = 0;
    else
        pause(app.player)
        app.flag = 1;
    end
else
    resume(app.player)
    app.flag = 0;
end
end
```

A.17 stapregelaar.m

```
function stapregelaar(app, j, counter, localornot , patient)
% this code checks which data needs to be downloaded from the
% server for, dowloads it and sends the revelant part to
% MainECGrunner to be displayed, after it has been processed in MainECG. IF the data is locally
% availabe,nothing will be downloaded and will be processed
% from the main storage directly
%NOTE: when fully realized, either this function needs to be
%redone, or the way data is read from the server needs to be
%according to how it is read here.
if localornot == 1

    app.ECGsignal = downloadECG(patient, j);
    app.Respsignal = downloadreps(patient, j);


end
try
    while app.VisGroup.SelectedTab == app.PhysicianTab |  app.VisGroup.SelectedTab == app.PatientXTab %#ok<OR2>

        g = app.fs_ecg* 50 * (j-1); %50 sec of ecg steps
        c = app.fs_ecg * 10 * counter; % 10 sec sub steps ecg

        r = app.fs_resp*50*(j-1); %50sec resp steps
        eps = app.fs_resp*10*(counter+1);

        if g == 0 && c == 0
            ECGred = app.ECGsignal( g+ c + 1 : g+c+app.fs_ecg*20, :);
        else
            ECGred = app.ECGsignal( g+ c -app.fs_ecg*10 + 1 : g+c+app.fs_ecg*10, :);
        end

        if j == 1 && counter < 2 && localornot == 1
            Repsred = app.Respsignal(r + eps +1 : r + eps + app.fs_resp*10);
        else
            Repsred = app.Respsignal(r + eps - app.fs_resp*30 +1 : r + eps + app.fs_resp*10);

        end


        processedsignal = MainECG(0 , ECGred, length(ECGred), length(Repsred), Repsred);

        mainECGrunner(app, processedsignal, g, c, localornot)
        counter = counter + 1;


        if counter == 3 && localornot == 1
            t = j + 1;

            ecg = downloadECG(patient, t);
            ecgcomb = cat(1, app.ECGsignal , ecg);
            app.ECGsignal = ecgcomb;

            reps = downloadreps(patient , t);
```

```
            repscomb = cat(1, app.Respsignal, reps);
            app.Respsignal = repscomb;
        elseif counter  == 5
            counter = 0;
            j = j + 1;
        end

        if mod(5*j + c, 8) == 0
            app.stress = randi(2)-1;
            pause(2)
        end



    end
catch
end
end
```

A.18 startupFcn.m

```
function startupFcn(app)
%everything that needs to be initialized in the beginning can be
%found here, such as server keys, folder locations etc.

if isempty(gcp('nocreate'))
    parpool('local');
end
setenv('MW_WASB_SAS_TOKEN',
↪  '?sv=2019-10-10&ss=bfqt&srt=sco&sp=rwdlacupx&se=2020-09-23T21:59:27Z&st=2020-05-11T13:59:27Z&spr=ht
setenv('MW_WASB_SECRET_KEY',
↪  '+bp+3xekejItpGtJaw5e5g4ZkV1kuGL9XC2sqrbTlq8jd7eSUqw5MFg1bBRI9/4uaE/KeRo97gSPmdO9IrAV3Q==')


addpath('ECG_preprocess')
addpath('extraECG')
addpath('extrareps')
app.LoginTab.Parent = app.VisGroup;
app.PatientXTab.Parent = app.InvGroup;
app.PhysicianTab.Parent = app.InvGroup;
app.VisGroup.SelectedTab = app.LoginTab;

musicfunc(app);
app.flag = 0;
set(app.greenring,'visible','on')
set(app.redring,'visible','off')
app.stresstext.Value = ['You are not stressed!' ...
    ' Keep it up!'];
set(app.calmgifje1, 'visible', 'off')
set(app.sound, 'visible', 'off')

app.stress = 0;
end
```

A.19 StressSwitchValueChanged.m

```
function StressSwitchValueChanged(app, event)
value = string(app.StressSwitch.Value);
app.stress = 1;
if value == 'On' %#ok<BDSCA>
    set(app.redring,'visible','on')
    set(app.greenring,'visible','off')
    app.stresstext.Value = ['You appear to be stressed!' ...
        ' Maybe this will help'];
    set(app.calmgifje1, 'visible', 'on')
    set(app.sound, 'visible', 'on')
else
    set(app.greenring,'visible','on')
    set(app.redring,'visible','off')
    app.stresstext.Value = ['You are not stressed!' ...
        ' Keep it up!'];
    set(app.calmgifje1, 'visible', 'off')
    set(app.sound, 'visible', 'off')
end


end
```

A.20 UIFigureCloseRequest.m

```
function UIFigureCloseRequest(app, event)
%closes parallel loop when closing the window
delete(gcp('nocreate'))
delete(app)

end
```

A.21 UploadButtonPushed.m

```
function UploadButtonPushed(app, event)
%Places files in a  folder depending on the type of information
%in the file. Will give error if incorrect file type is
%given.
try
    [~, ~ , fExt] = fileparts(app.filename);
catch
end

name = app.PatientNameEditField.Value;
if isempty(fExt) == 1
    errordlg('No file was selected.',...
        'No file');
elseif convertCharsToStrings(fExt) == '.csv' %#ok<BDSCA>

    if app.ECGButton.Value == 1 && app.RespiratoryButton.Value == 0
        namecat = strcat('ECG_', name, '.csv');
        z = append(app.path, app.filename);
        copyfile(z , 'extraECG')
        cd 'extraECG'
    elseif app.RespiratoryButton.Value == 1 && app.ECGButton.Value == 0
        namecat = strcat('Resp_', name, '.csv');
```

```
        z = append(app.path, app.filename);
        copyfile(z , 'extrareps')
        cd 'extrareps'
    end


    if isempty(name) == 1
        errordlg('Fill in the patients name!',...
            'No name');
    elseif strcmp(app.filename, namecat) == 0
        copyfile(app.filename, namecat)
        delete(app.filename)
    end


else
    errordlg('Upload only ''.csv'' files!',...
        'Wrong file type');
end

app.filenamepathtext.Value = '';
app.PatientNameEditField.Value = '';
cd ..\
add2list(app)
end
```