VNS Monitoring System

Index

1. Introduction
    1. Objective and Purpos
    2. Capability of our device
    3. Hardware used
2. Design and Schematic
    1. Body design
    2. Schematic
3. Code Manuel
    1. Hardware Code Explanation
        1. Setting up environment for programming raspberry pi pico w with the code
        2. Configuring Arduino IDE to program raspberry pi pico w
        3. Checking for required Library Availability
        4. The code
    2. VNS Manager code
    3. VNS QR Generator code
4. User Manuel
5. Appendix A

Introduction

Picture of the Prototype

1.1) Objective and Purpose

The project aims to enable large-scale collection of reliable electrocardiogram (ECG) data from human patients. This data will serve as a foundation for extracting a comprehensive range of physiological parameters including, but not limited to:

- Heart rate variability (HRV)
- Respiratory sinus arrhythmia (RSA)
- QT interval variability
- P-wave dispersion
- T-wave alternans
- QRS complex characteristics
- ST segment analysis

These parameters are essential for monitoring the health status of a diverse population and evaluating the impact of specific activities when practised for a long time , such as in our case monitoring the effect of practising yoga and different breathing exercises on vagal tone and overall cardiovascular health .

1.2) Capability of our device

Our prototype is specifically designed to capture single-channel ECG data from patients using electrodes attached to their bodies. This data is recorded and stored in 16-bit format on an SD card, saved as a text file named in the format "YYYY-MM-DD--hh-mm-ss-ID". Here, 'ID' denotes the unique identifier assigned to each patient, 'YYYY-MM-DD' represents the date of storage, and 'hh-mm-ss' indicates the time of the reading. These files are organized within a folder named after the patient's ID.

Data is transferred to the machine via WiFi, facilitated by a built-in web server accessed from a connected device. The recorded data can be processed using our software solution to extract a range of parameters such as LF/HF ratio, heart

rate, IBI (Inter-Beat Interval), RMSSD (Root Mean Square of Successive Differences), and CVI (Cardiac Vagal Index). These parameters can be viewed graphically in individual reports or collectively over time for a single patient.

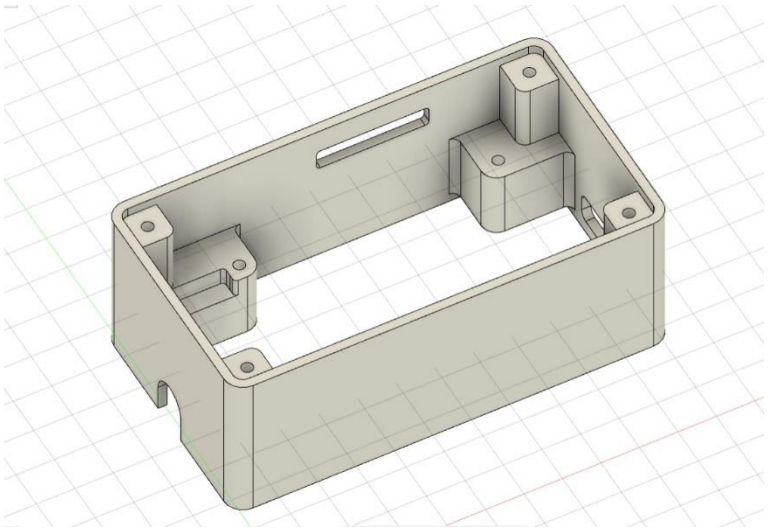
Our solution is highly flexible as users have access to raw ECG data, enabling custom statistical analyses and insights based on the collected data.

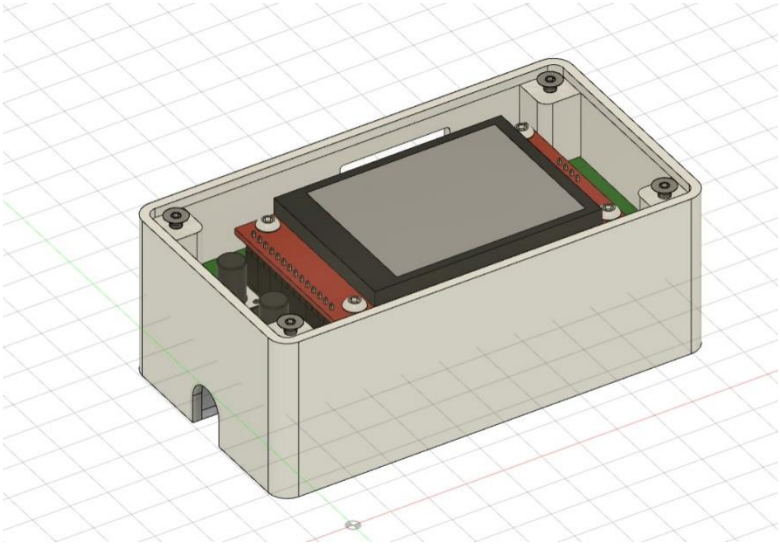
1.3) Hardware Used

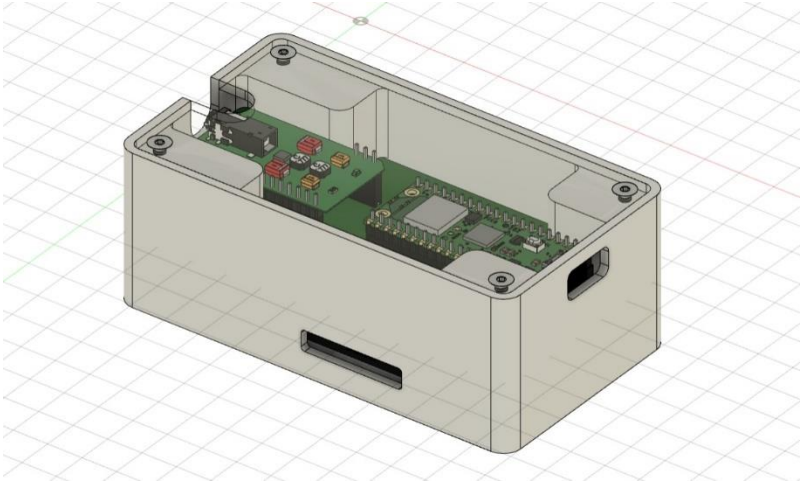
1. Raspberry PI Pico W: This microcontroller serves as the main control unit responsible for managing the various states of the machine. It was chosen for its extensive documentation, built-in WiFi capabilities, and cost-effectiveness, ensuring sufficient processing power for the project.
2. ECG CLICK 4: This sensor, utilising the BMD101 SOC, was selected because of its clinically accurate ECG data recording capabilities supported by multiple research papers and integrated ADC converter and hardware filters.
3. ILI934 2.4" TFT Display: Selected for its suitability and generic nature, this display meets the project's requirements effectively.
4. 32 GB SD Card: Chosen because it is the largest size supported by the FAT32 protocol used in the Raspberry PI Pico for data storage.

Design and Schematic

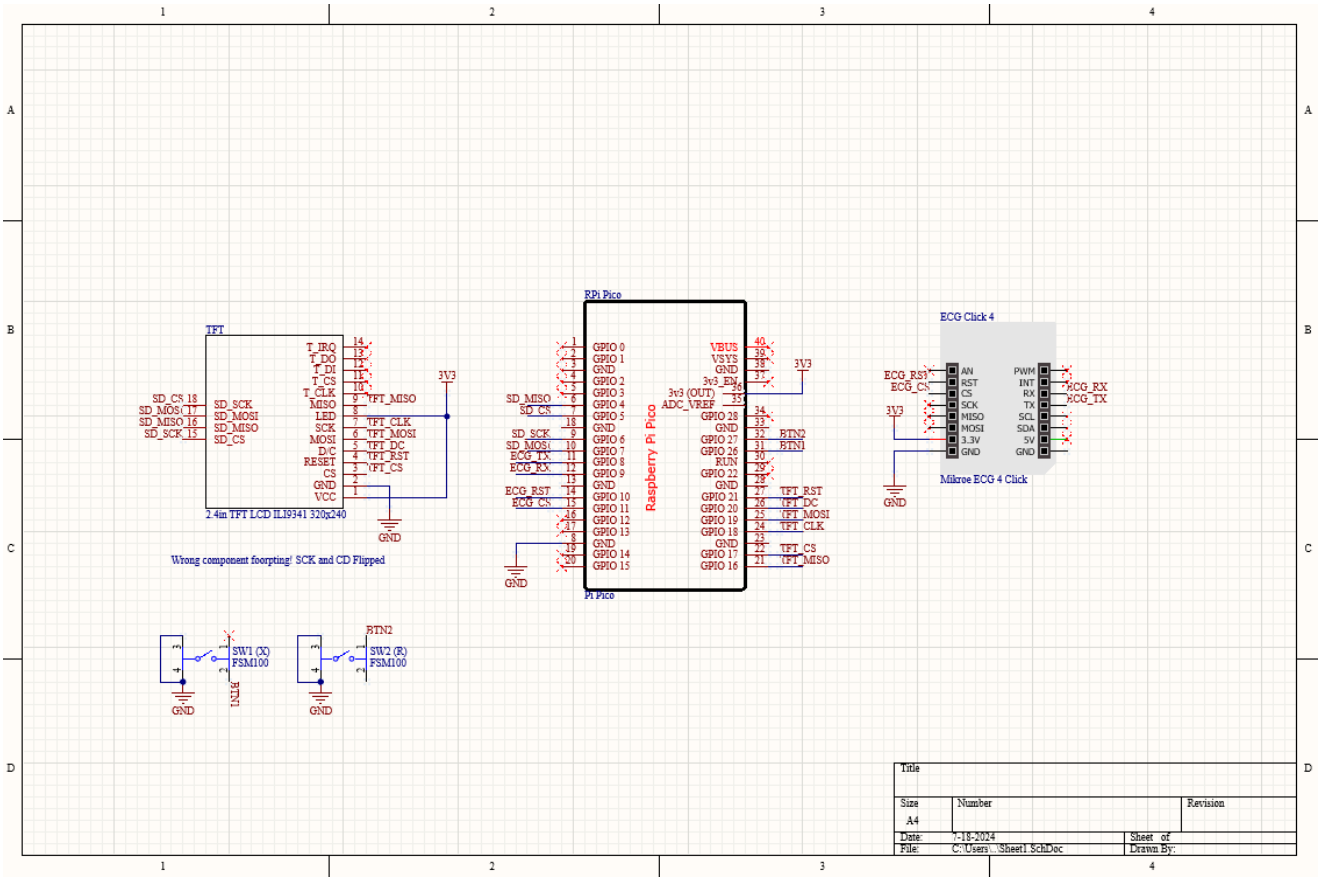
2.1) Body design







2.2) Schematic



Code Manuel

3.1) Hardware code

This Section provides a step-by-step guide on programming the Raspberry Pi Pico W with the given program, and it also explains how the code works to facilitate future upgrades

3.1.1) Setting Up environment for programming raspberry pi pico w with the code

Installing Arduino ide in system

Downloading and installing the Arduino IDE is straightforward across Windows, Linux, and macOS. Here are the steps for each operating system:

For Windows:

1. **Visit the Arduino Website**:
   - Go to the official Arduino download page: Arduino Software.
2. **Download the Installer**:
   - Under the "Windows" section, click on the link to download the Windows installer.
   - Choose between the "Windows Win 7 and newer" or "Windows ZIP file for non-admin install" options based on your requirements.
3. **Run the Installer**:
   - Once the download is complete, open the installer file.
   - Follow the on-screen instructions to complete the installation. Typically, this involves agreeing to the license agreement, choosing an installation location, and selecting components to install.
4. **Complete the Installation**:
   - Click "Install" and wait for the process to finish.
   - Once installed, you can launch the Arduino IDE from the Start menu or the desktop shortcut.

For Linux:

1. **Visit the Arduino Website**:
   - Go to the official Arduino download page: Arduino Software.
2. **Download the Installer**:
   - Under the "Linux" section, choose the appropriate version for your distribution (32-bit, 64-bit, or ARM).
3. **Extract the Files**:

- Open a terminal and navigate to the directory where the downloaded file is located.

  Use the tar command to extract the files. For example:
  bash
  tar -xvf arduino-*.tar.xz

4. **Run the Install Script**:

Navigate into the extracted directory:
bash
cd arduino-*
Run the installation script with the following command:
bash
sudo ./install.sh

5. **Launch the Arduino IDE**:
   - You can start the Arduino IDE from your application launcher or by typing arduino in the terminal.

For macOS:

1. **Visit the Arduino Website**:
   - Go to the official Arduino download page: Arduino Software.
2. **Download the Installer**:
   - Under the "Mac OS X" section, click on the link to download the macOS installer.
3. **Open the Disk Image**:
   - Once the download is complete, open the .dmg file.
   - Drag the Arduino application into the Applications folder.
4. **Launch the Arduino IDE**:
   - Go to the Applications folder and double-click on the Arduino app to launch it.
   - You may need to bypass the Gatekeeper security by right-clicking on the Arduino app and selecting "Open," then confirming you want to open the app.

After following these steps, the Arduino IDE should be installed and ready to use on your respective operating system.

3.1.2) Configuring Arduino IDE to program raspberry pi pico w

Configuring the Arduino IDE to program the Raspberry Pi Pico involves several steps. Here's a detailed guide:

Prerequisites:

- Ensure you have the latest version of the Arduino IDE installed.
- A USB cable to connect the Raspberry Pi Pico to your computer.

Steps to Configure Arduino IDE for Raspberry Pi Pico:

1. Install the Raspberry Pi Pico Board in Arduino IDE:

1. **Open Arduino IDE**:
   - Launch the Arduino IDE on your computer.
2. **Add the Board Manager URL**:
   - Go to File > Preferences.

     In the "Additional Board Manager URLs" field, add the following URL:
     https://github.com/earlephilhower/arduino-pico/releases/download/global/package_rp2040_index.json

   - If there are already other URLs in the field, separate them with a comma.
3. **Open the Board Manager**:
   - Go to Tools > Board > Board Manager.
   - In the search bar, type rp2040 or Raspberry Pi Pico.
   - Install the package named "Raspberry Pi Pico/RP2040".

2. Connect the Raspberry Pi Pico:

1. **Prepare the Pico for USB Programming**:
   - Hold down the BOOTSEL button on the Pico.
   - While holding the button, connect the Pico to your computer using a USB cable.
   - Release the BOOTSEL button after connecting. The Pico should mount as a removable storage device named RPI-RP2.

3. Select the Board and Port:

1. **Select the Board**:
   - Go to Tools > Board.
   - Select Raspberry Pi Pico or the appropriate RP2040 board you installed earlier.
2. **Select the Port**:
   - Go to Tools > Port.
   - Select the port that corresponds to your connected Pico. It should appear as something like COMx on Windows, /dev/ttyACMx or /dev/ttyUSBx on Linux, and /dev/cu.usbmodemXXXX on macOS.

4. Upload a Blink Example:

1. **Open an Example Sketch**:
   - Go to File > Examples > 01.Basics > Blink.
2. **Upload the Sketch**:

- Click on the Upload button (right arrow) in the Arduino IDE.
- The IDE will compile the sketch and upload it to the Pico. You should see the onboard LED of the Pico blinking if the upload is successful.

Troubleshooting:

- **If the Pico does not appear as a port:**
  - Ensure the Pico is in boot mode by pressing the BOOTSEL button while connecting it to the computer.
  - Try using a different USB cable or port.
- **If the upload fails:**
  - Double-check the board and port selections in the Tools menu.
  - Ensure the board package for the RP2040 is installed correctly.

Additional Resources:

- Raspberry Pi Pico Getting Started Guide
- Arduino-Pico GitHub Repository

3.1.3) Checking for required Library Availability

The Following libraries are needed to run code

- Arduino.h
- Stdint.h
- avr/pgmspace.h
- Adafruit_GFX.h
- SD.h
- SPI.h
- FreeSerif12pt7b.h
- SPI.h
- Adafruit_ILI9341.h
- ui_bitmap.h
- Adafruit_Debounce.h
- WiFi.h
- WebServerSecure.h
- FS.h
- LEAmDNS.h

The following steps can be followed to ensure if libraries are available or not

Open an Example :

- Go to File > Examples >Library_name>Example_skecth

The following steps are can followed to install the missing libraries

Using the Library Manager (Recommended Method)

1. **Open Arduino IDE:**
   - Launch the Arduino IDE on your computer.
2. **Open the Library Manager:**
   - Go to Sketch > Include Library > Manage Libraries...
3. **Search for the Library:**
   - In the Library Manager window, use the search bar to find the library you need.
4. **Install the Library:**
   - Select the library from the list.
   - Click the "Install" button. If multiple versions are available, you can select the version you need from the dropdown menu.
5. **Include the Library in Your Sketch:**
   After installing, include the library in your sketch using:
   #include <LibraryName.h>

Manual Installation of Libraries

If the library is not available through the Library Manager, you can manually install it. Here's how:

1. **Download the Library:**
   - Go to the library's repository or website (e.g., GitHub).
   - Download the library as a .zip file.
2. **Add the Library to Arduino IDE:**

Windows and macOS:

1. **Open Arduino IDE:**
   - Launch the Arduino IDE on your computer.
2. **Add the .zip Library:**
   - Go to Sketch > Include Library > Add .ZIP Library...
   - Navigate to the downloaded .zip file and select it.
   - The library will be added to your Arduino libraries.

Linux:

1. **Open Arduino IDE:**
   - Launch the Arduino IDE on your computer.
2. **Add the .zip Library:**
   - Go to Sketch > Include Library > Add .ZIP Library...
   - Navigate to the downloaded .zip file and select it.
   - The library will be added to your Arduino libraries.

3. **Include the Library in Your Sketch:**
    - After installing, include the library in your sketch using: `#include <LibraryName.h>`

Alternative Manual Installation Method (Advanced)

If you prefer or need to manually install libraries by placing files directly into the Arduino libraries directory, follow these steps:

1. **Download the Library:**
    - Download the library as a .zip file from the repository or website.
2. **Extract the Library:**
    - Extract the .zip file to obtain the library folder.
3. **Move the Library Folder:**
    - Locate the Arduino libraries directory on your computer:
        - **Windows**: Documents > Arduino > libraries
        - **macOS**: Documents > Arduino > libraries
        - **Linux**: Documents/Arduino/libraries or /home/username/Arduino/libraries
    - Move the extracted library folder into the libraries directory.
4. **Restart Arduino IDE:**
    - Close and reopen the Arduino IDE to recognize the new library.
5. **Include the Library in Your Sketch:**
    - After installing, include the library in your sketch using: `#include <LibraryName.h>`

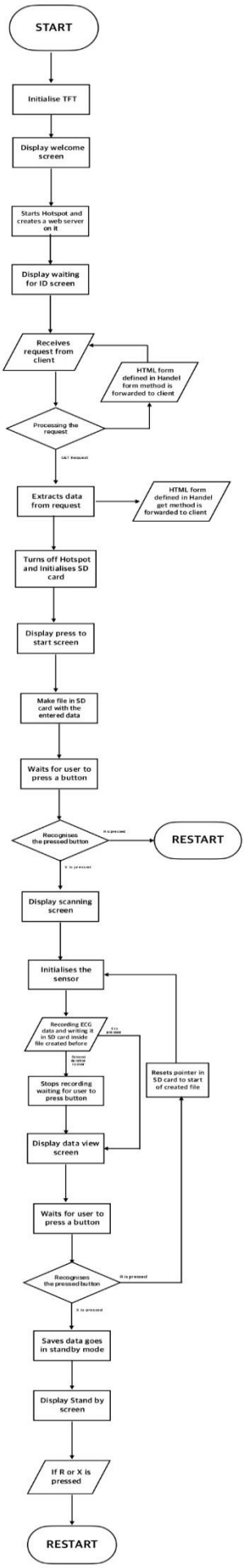
Example

Let's walk through an example of installing the Adafruit_Sensor library using the Library Manager:

1. **Open Arduino IDE.**
2. **Open Library Manager:**
    - Sketch > Include Library > Manage Libraries...
3. **Search for Adafruit_Sensor:**
    - Type Adafruit_Sensor in the search bar.
4. **Install the Library:**
    - Click on Adafruit Unified Sensor in the list.
    - Click the "Install" button.
5. **Include the Library:**
    - In your sketch, add: `#include <Adafruit_Sensor.h>`

3.1.4) The Code

Flow-Chart to Explain the process in which the code works



The code is written in different modules so that it easy to understand and upgrade this section will explain every module along with their respective code snippets

1. **VNS_MONITOR_FIRMWARE**

```
#define BTN_X 26 // X
#define BTN_R 27 // R
```

BTN_X and **BTN_R** are defined as the pin numbers for the buttons X and R respectively. This makes it easier to reference these pins throughout the code by using their descriptive names instead of raw numbers.

```
#define ECG_CS 11
#define ECG_RST 10
#define ECG_RX 9
#define ECG_TX 8
#define ECG_FIFO_SIZE 64
#define ECG_BAUD 57600
```

These define the pins used for interfacing with ECG Sensor

- **ECG_CS**: Pin for chip select for the ECG module.
- **ECG_RST**: Pin for resetting the ECG module.
- **ECG_RX**: Pin for receiving data from the ECG module.
- **ECG_TX**: Pin for transmitting data to the ECG module.
- **ECG_FIFO_SIZE**: The FIFO (First In, First Out) buffer size for the ECG data.
- **ECG_BAUD**: The baud rate for serial communication with the ECG module.

```
#define SD_MISO 4
#define SD_CS 5
#define SD_SCK 6
#define SD_MOSI 7
```

These define the pins used for interfacing with the SD card:

- **SD_MISO**: Master In Slave Out pin.
- **SD_CS**: Chip select pin.
- **SD_SCK**: Clock pin.
- **SD_MOSI**: Master Out Slave In pin.

```
#define TFT_MISO 16
#define TFT_MOSI 19
#define TFT_CLK  18
#define TFT_CS   17  // Chip select control pin (GPIO 20)
#define TFT_DC   20  // Data Command control pin (GPIO 18)
#define TFT_RST  21  // Reset pin (GPIO 19)
```

These define the pins used for interfacing with the TFT display:

- **TFT_MISO**: Master In Slave Out pin.
- **TFT_MOSI**: Master Out Slave In pin.
- **TFT_CLK**: Clock pin.
- **TFT_CS**: Chip select control pin.
- **TFT_DC**: Data Command control pin.
- **TFT_RST**: Reset pin.

```
#define ROTATION 3
//#define BG_COLOR ILI9341_BLACK
#define LINE_COLOR  ILI9341_WHITE
#define CLK_COLOR ILI9341_BLACK
#define TEXT_BG_COLOR 0xF6BC
#define BG_COLOR 0x0351
#define FG_COLOR 0xef5d
```

These define the UI basics

- **ROTATION**: Specifies the display rotation.
- **LINE_COLOR**: Color for lines, defined as white.
- **CLK_COLOR**: Color for the clock, defined as black.
- **TEXT_BG_COLOR**: Background color for text, using a custom hexadecimal color code.
- **BG_COLOR**: General background color, using a custom hexadecimal color code.
- **FG_COLOR**: Foreground color, using a custom hexadecimal color code.

```
#define UPDATE_DURATION 5 // in seconds
#define RECORD_DURATION 30 // in seconds
```

UPDATE_DURATION: Time interval for updates, set to 5 seconds.
RECORD_DURATION: Duration for recording, set to 30 seconds.

```
struct user_info
{
 String ID;
 String Name;
 String Date_time;
 int Duration;
 String Age;
 String Gender;
 String Weight;
};
```

This defines a custom data structure used in code to store information

- **ID**: User ID as a string.
- **Name**: User name as a string.
- **Date_time**: Date and time as a string.
- **Duration**: Duration of something (e.g., a session or event) as an integer.
- **Age**: User age as a string.
- **Gender**: User gender as a string.
- **Weight**: User weight as a string.

2. BMD101.cpp

```cpp
#include "BMD101.h"

BMD101::BMD101(int8_t cs, int8_t rst) {
    _cs = cs;
    _rst = rst;
}
```

Purpose: Includes the header file for the BMD101 class and defines the constructor.
Significance:

- #include "BMD101.h": Includes the BMD101 class definition.
- BMD101::BMD101(int8_t cs, int8_t rst): Constructor initializes the chip select (_cs) and reset (_rst) pins.

```cpp
void BMD101::begin() {
    return this->begin(&Serial);
}

void BMD101::begin(Stream* serial) {
    pinMode(_cs, OUTPUT);
    digitalWrite(_cs, HIGH);

    pinMode(_rst, OUTPUT);
    digitalWrite(_rst, HIGH);

    _serial = serial;
    // _serial.setFIFOSize(BMD101_FIFO_SIZE);
    // _serial->begin(BMD101_BAUD);
}
```

Purpose: Initializes the BMD101 module.

Significance:

- begin(): Calls the overloaded begin method with the default Serial stream.

- begin(Stream* serial): Sets the pin modes and initializes the serial communication for the BMD101 module.

```
void BMD101::reset() {
    digitalWrite(_rst, LOW);
    delay(200);
    digitalWrite(_rst, HIGH);
    delay(200);
}

void BMD101::enable() {
    digitalWrite(_cs, HIGH);
}

void BMD101::disable() {
    digitalWrite(_cs, LOW);
}
```

Purpose: Controls the state of the BMD101 module.

Significance:

- reset(): Resets the BMD101 module by toggling the reset pin.
- enable(): Enables the BMD101 module by setting the chip select pin high.
- disable(): Disables the BMD101 module by setting the chip select pin low.

```
int8_t BMD101::available() {
    return _serial->available();
}

int8_t BMD101::data_available() {
    return _data.raw_idx;
}

bool BMD101::sensor_on() {
    return _data.signal_quality;
}

uint8_t BMD101::get_heart_rate() {
    return _data.heart_rate;
}

int16_t BMD101::get_raw_value() {
    int8_t idx = --(_data.raw_idx);
    return _data.raw_data[idx];
}
```

Purpose: Provides methods to access data and status of the BMD101 module.

Significance:

- available(): Returns the number of bytes available in the serial buffer.
- data_available(): Returns the index of the latest raw data.
- sensor_on(): Returns the signal quality status.
- get_heart_rate(): Returns the heart rate value.
- get_raw_value(): Returns the latest raw data value.

```
void BMD101::process() {
    static uint8_t rx_buff[ 256 ];
    static uint8_t rx_cnt = 0;
    static uint8_t rx_idx;
    static uint8_t payload_size;
    static uint8_t row_check = 0;
    static uint8_t op_code;
    static uint8_t row_size;
    static uint8_t row_size_check;
    static uint8_t checksum = 0;
    static uint8_t row_cnt = 0;
    uint8_t rx_dat;

    rx_dat = _serial->read();

    // processing buffer byte by byte
    if ((rx_cnt == 0) && (rx_dat != 
BMD101_SYNC_BYTE)) {
        rx_cnt = 0;

        return;
    }
    else if ((rx_cnt == 1) && (rx_dat != 
BMD101_SYNC_BYTE)) {
        rx_cnt = 0;

        return;
    }
    else if (rx_cnt == 2) {
        payload_size = rx_dat;
        rx_cnt++;
        row_check = 1;
```

```
        return;
    }
    else if (rx_cnt > 2) {
        if (rx_cnt - 3 < payload_size) {
            if (rx_dat ==
BMD101_EXCODE_BYTE) {
                row_check = 1;
                checksum += rx_dat;
                rx_cnt++;

                return;
            }
            if (row_check == 1) {
                op_code = rx_dat;
                row_check = 0;
                checksum += rx_dat;

                if ((op_code ==
BMD101_SIGNAL_QUALITY_CODE_BYTE) || (
op_code == BMD101_HEART_RATE_CODE_BYTE)) {
                    row_size = 1;
                    row_size_check = 0;
                }
                else if (op_code ==
BMD101_RAW_DATA_CODE_BYTE) {
                    row_size_check = 1;
                }
                else {
                    rx_cnt = 0;
                    checksum = 0;

                    return;
                }
```

```
rx_idx = 0;
                rx_cnt++;

                return;
            }

            if (row_size_check == 1) {
                row_size = rx_dat;
                row_size_check = 0;
                checksum += rx_dat;
                rx_cnt++;

                return;
            }

            if (rx_idx < row_size) {
                rx_buff[ rx_idx ] = rx_dat;
                rx_idx++;
                checksum += rx_dat;

                if (rx_idx == row_size) {


// entire row read, store data in buffer st
ruct
                    if(op_code ==
BMD101_SIGNAL_QUALITY_CODE_BYTE) {
                        _data_buf.
signal_quality = rx_buff[0];
                    }
                    else if(op_code ==
BMD101_HEART_RATE_CODE_BYTE) {
                        _data_buf.
heart_rate = rx_buff[0];
                    }
                    else if(op_code ==
BMD101_RAW_DATA_CODE_BYTE) {
                        _data_buf.raw_value
 = (rx_buff[0] << 8) + rx_buff[1];
                    }

                    row_cnt++;
                    row_check = 1;
                }
```

```
                rx_cnt++;

                return;
            }
        }
        else {
            checksum = ~checksum;

            if (checksum == rx_dat) {

// data valid, updata valid data struct fro
m buffer data
                _data.signal_quality =
_data_buf.signal_quality;
                _data.heart_rate =
_data_buf.heart_rate;
                if(_data.raw_idx ==
BMD101_RAW_BUFFER_SIZE) {

// raw data buffer full! reset index
                    _data.raw_idx = 0;
                }
                _data.raw_data[_data.
raw_idx] = _data_buf.raw_value;
                _data.raw_idx++;
            }
            else {
                // data invalid
                // dont update data
            }

            checksum = 0;
            row_cnt = 0;
            rx_cnt = 0;
            rx_idx = 0;

            return;
        }
    }

    rx_cnt++;
}
```

Purpose: Processes incoming data from the BMD101 module.

Significance:

- Handles reading and processing the data byte-by-byte from the serial buffer.
- Validates and parses the data packets received from the BMD101 module.
- Updates internal data structures (_data and _data_buf) with the latest sensor readings.

1. **BMD101.h**

```
#ifndef _BMD101_
#define _BMD101_

#include "Arduino.h"
```

Purpose: Prevents multiple inclusions of the same header file and includes the Arduino library.
Significance:

- #ifndef _BMD101_ and #define _BMD101_: Ensure the file is only included once during compilation.
- #include "Arduino.h": Includes the Arduino core library.

```
#define BMD101_BAUD 57600
#define BMD101_FIFO_SIZE 64
#define BMD101_RAW_BUFFER_SIZE 128

#define BMD101_SYNC_BYTE 0xAA
#define BMD101_EXCODE_BYTE 0x55
#define BMD101_SIGNAL_QUALITY_CODE_BYTE 0x02
#define BMD101_SIGNAL_OFF 0x00
#define BMD101_SIGNAL_ON 0xC8
#define BMD101_HEART_RATE_CODE_BYTE 0x03
#define BMD101_RAW_DATA_CODE_BYTE 0x80

#define PICO_UART1_TX 8
#define PICO_UART1_RX 9
```

Purpose: Defines constants and macros used in the library.

Significance:

- BMD101_BAUD: Baud rate for serial communication (57600 baud).
- BMD101_FIFO_SIZE: Size of the FIFO buffer (64 bytes).
- BMD101_RAW_BUFFER_SIZE: Size of the raw data buffer (128 values).
- BMD101_SYNC_BYTE: Sync byte for data packets (0xAA).
- BMD101_EXCODE_BYTE: Extended code byte (0x55).
- BMD101_SIGNAL_QUALITY_CODE_BYTE: Signal quality code byte (0x02).
- BMD101_SIGNAL_OFF: Signal off value (0x00).
- BMD101_SIGNAL_ON: Signal on value (0xC8).
- BMD101_HEART_RATE_CODE_BYTE: Heart rate code byte (0x03).

- BMD101_RAW_DATA_CODE_BYTE: Raw data code byte (0x80).
- PICO_UART1_TX and PICO_UART1_RX: UART1 TX and RX pin numbers for the Pico (8 and 9, respectively).

```
struct BMD101_data {
    uint8_t signal_quality = 0;
    uint8_t heart_rate = 0;
    int8_t raw_idx = 0;
    int16_t raw_data[BMD101_RAW_BUFFER_SIZE];
};

struct BMD101_data_buf {
    uint8_t signal_quality = 0;
    uint8_t heart_rate = 0;
    int16_t raw_value = 0;
};
```

Purpose: Defines data structures to hold sensor data.

Significance:

- BMD101_data: Holds the current signal quality, heart rate, raw data index, and raw data buffer.
- BMD101_data_buf: Holds temporary data for signal quality, heart rate, and raw value.

```
class BMD101 {
    public:
        BMD101(int8_t BMD101_CS, int8_t
BMD101_RST);
        // BMD101(int8_t BMD101_CS, int8_t
BMD101_RST, int8_t BMD101_RX, int8_t BMD101
_TX);

        void begin();
 // begins with default serial channel (Ser
ial)
        void begin(Stream* serial);
 // begins with referenced serial channel

        void reset();
        void enable();
 // set Chip-Select pin HIGH
        void disable();
 // set Chip-Select pin LOW

        int8_t available();
 // returns the length of bytes available t
o read in serial buffer
        int8_t data_available();
 // returns the number of raw data values s
tored

        bool sensor_on();
 // returns true if BMD101's on (according
to its internal sensor detection), false ot
herwise
        uint8_t get_heart_rate();
 // returns last valid heart rate
        int16_t get_raw_value();
 // returns last valid raw value and remove
s it from struct

        void process();
 // processes one byte from the serial buff
er

    private:
        int8_t _cs;
        int8_t _rst;
        Stream* _serial;
        BMD101_data_buf _data_buf;
        BMD101_data _data;

        // void process_row();

};
```

Purpose: Defines the BMD101 class and its public and private members.

Significance:

- **Public Methods**:
  - BMD101(int8_t BMD101_CS, int8_t BMD101_RST): Constructor initializes the CS and RST pins.
  - void begin(): Initializes with the default serial channel.
  - void begin(Stream* serial): Initializes with a specified serial channel.
  - void reset(): Resets the BMD101 module.
  - void enable(): Sets the chip select pin high to enable the module.
  - void disable(): Sets the chip select pin low to disable the module.
  - int8_t available(): Returns the number of bytes available in the serial buffer.
  - int8_t data_available(): Returns the number of raw data values stored.
  - bool sensor_on(): Checks if the sensor is on.
  - uint8_t get_heart_rate(): Returns the last valid heart rate.
  - int16_t get_raw_value(): Returns the last valid raw value.
  - void process(): Processes one byte from the serial buffer.
- **Private Members**:
  - int8_t _cs: Chip select pin.
  - int8_t _rst: Reset pin.
  - Stream* _serial: Serial stream used for communication.
  - BMD101_data_buf _data_buf: Temporary buffer for data processing.
  - BMD101_data _data: Holds the current data values.

```
#endif // _BMD101_
```

Purpose: Marks the end of the include guard.

Significance: Ensures the file is only included once during compilation to prevent redefinition errors.

2. button

```
#include <Adafruit_Debounce.h>
```

Purpose: Includes the Adafruit Debounce library, which provides a simple way to debounce buttons in your project.
Significance: Debouncing ensures that multiple presses (bounces) of a button are registered as a single press, preventing unintended behavior in the project.

```
Adafruit_Debounce btnx(BTN_X, LOW);
Adafruit_Debounce btnr(BTN_R, LOW);
```

Purpose: Creates instances of the Adafruit_Debounce class for two buttons, btnx and btnr.
Significance:

- btnx(BTN_X, LOW): Instantiates the btnx object for the button connected to pin BTN_X, with the initial state set to LOW.
- btnr(BTN_R, LOW): Instantiates the btnr object for the button connected to pin BTN_R, with the initial state set to LOW.

```
void initButtons() {
    btnx.begin();
    btnr.begin();
}
```

Purpose: Defines a function initButtons to initialize the button debounce objects.

Significance:

- btnx.begin(): Initializes the debounce functionality for the btnx button. This sets up the necessary configurations for debouncing BTN_X.
- btnr.begin(): Initializes the debounce functionality for the btnr button. This sets up the necessary configurations for debouncing BTN_R.
- The commented-out line // pinMode(BTN_R, INPUT_PULLUP); is not used but would have set BTN_R as an input with an internal pull-up resistor if it were uncommented.

3. Dialog_bold_20.h

This file contains bitmap of Dialog bold font of 20 size and contains hexadecimal values representing pixel data for each character from ASCII 32 (' ') to ASCII 90 ('Z'), followed by symbols like '[', '', ']', and '^'.

4. ECG

```
void initECG() {
    Serial.println("Initializing ECG Module");

    // Configure Serial2 for communication with the ECG module
    Serial2.setRX(ECG_RX);          // Set RX pin for Serial2
    Serial2.setTX(ECG_TX);          // Set TX pin for Serial2
    Serial2.setFIFOSize(ECG_FIFO_SIZE);  // Set FIFO size for Serial2 buffer
    Serial2.begin(ECG_BAUD);        // Initialize Serial2 with baud rate

    // Initialize BMD101 ECG module
    bmd.begin(&Serial2);            // Pass Serial2 object to BMD101 library
    bmd.enable();                   // Enable the BMD101 module
    bmd.reset();                    // Reset the BMD101 module

    Serial.println("ECG Module Initialized");
    Serial.println();
}
```

- **Serial Communication Setup**: Serial2 is configured to communicate with the BMD101 ECG module. This involves setting the RX and TX pins, specifying the FIFO size (buffer size), and initializing it with a baud rate (ECG_BAUD).
- **BMD101 Initialization**: The BMD101 object bmd is initialized using Serial2 (which is connected to the ECG module). It's then enabled and reset to prepare it for operation.

```
int16_t getECG() {
    while (Serial2.available()) {
        bmd.process();                  // Process data from BMD101 module
        if (bmd.data_available()) {     // Check if new data is available
            return bmd.get_raw_value(); // Return raw ECG value
        }
    }
    return 0; // Return 0 if no new data available
}
```

- **Data Reading Loop**: Continuously checks if there is data available on Serial2.
- **Data Processing**: Calls bmd.process() to handle incoming data from the BMD101 module.
- **Data Availability Check**: Checks if new ECG data is available using bmd.data_available().
- **Data Retrieval**: If new data is available, retrieves the raw ECG value using bmd.get_raw_value() and returns it.

```
void stopECG() {
    bmd.disable();              // Disable the BMD101 module
    Serial.println("ECG sensor disabled");
}
```

- **Module Disabling**: Calls bmd.disable() to turn off the BMD101 ECG module.
- **Status Message**: Prints a message confirming that the ECG sensor has been disabled.

5. **FreeSerif12pt7b.h**
   This file contains bitmap of FreeSherif bold font of 12 size and contains hexadecimal values representing pixel data for each character
6. **SD**

```
#include <SD.h>
#include <SPI.h>

void initSD() {
  Serial.println("Initializing SD Card...");

  // Set SPI pins for SD card communication
  SPI.setMISO(SD_MISO);
  SPI.setMOSI(SD_MOSI);
  SPI.setSCK(SD_SCK);

  // Initialize the SD card
  if (!SD.begin(SD_CS)) {
    criticalError("Card failed, or not present");
  }
  Serial.println("SD Card initialized.");
  Serial.println();
}
```

SPI Configuration:

The lines SPI.setRX(SD_MISO);, SPI.setTX(SD_MOSI);, and SPI.setSCK(SD_SCK); have been replaced with SPI.setMISO(SD_MISO);, SPI.setMOSI(SD_MOSI);, and SPI.setSCK(SD_SCK);, respectively, since these are the correct methods to use.

SD.begin(SD_CS):

This initializes communication with the SD card using the pin defined as SD_CS. If initialization fails, the function calls criticalError() to handle the situation.

```
void removeFile(String filename) {
  if (SD.exists(filename)) {
    SD.remove(filename);
    Serial.print(filename);
    Serial.println(" Removed.");
  }
}
```

SD.exists(filename):

- Checks if the file filename exists on the SD card.

SD.remove(filename):

- Deletes the file filename from the SD card if it exists.

Serial.print and **Serial.println**:

- Outputs messages to the serial monitor indicating whether the file was successfully removed.

7. **TFT**

```
#include "FreeSerif12pt7b.h"
#include "SPI.h"
#include "Adafruit_GFX.h"
#include "Adafruit_ILI9341.h"
#include "Dialog_bold_20.h"
#include "ui_bitmap.h"

#define ID_Y 127
#define HR_Y 75
#define SECONDS_Y 180

Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC, TFT_RST);
uint16_t row[320];
```

Includes: You're including necessary libraries for your TFT display (Adafruit_GFX.h for graphics functions and Adafruit_ILI9341.h for the specific display driver).
Definitions: Constants like ID_Y, HR_Y, and SECONDS_Y are defined for positioning elements on the screen.

```
void initTFT() {
    Serial.println("Initializing TFT Display");

    // Configure SPI pins for TFT communication
    SPI.setMISO(TFT_MISO);
    SPI.setSCK(TFT_CLK);
    SPI.setMOSI(TFT_MOSI);

    // Initialize the TFT display
    tft.begin(64000000); // Set SPI clock rate (optional)
    tft.setRotation(ROTATION);
    tft.fillScreen(BG_COLOR);
    tft.setFont(&Dialog_bold_20);

    // Read diagnostics (optional but can help debug problems)
    uint8_t x = tft.readcommand8(ILI9341_RDMODE);
    Serial.print("Display Power Mode: 0x"); Serial.println(x, HEX);
    x = tft.readcommand8(ILI9341_RDMADCTL);
    Serial.print("MADCTL Mode: 0x"); Serial.println(x, HEX);
    x = tft.readcommand8(ILI9341_RDPIXFMT);
    Serial.print("Pixel Format: 0x"); Serial.println(x, HEX);
    x = tft.readcommand8(ILI9341_RDIMGFMT);
    Serial.print("Image Format: 0x"); Serial.println(x, HEX);
    x = tft.readcommand8(ILI9341_RDSELFDIAG);
    Serial.print("Self Diagnostic: 0x"); Serial.println(x, HEX);
}
```

- **SPI Configuration**: Correctly sets up the SPI pins (MISO, SCK, MOSI) for communication with the TFT display.
- **TFT Initialization**: Initializes the TFT display with a specified SPI clock rate (64000000 here, adjust as needed), sets rotation, fills the screen with a background color (BG_COLOR), and sets a font for text.

- **Diagnostics**: Optional diagnostics read from the TFT display to verify communication and configuration.

```
void clearTFT() {
    tft.fillScreen(BG_COLOR);
}
```

Clears the TFT screen by filling it with the background color (BG_COLOR)

```
void drawBitmap(int16_t x, int16_t y, const uint16_t *bitmap, int16_t w, int16_t h) {
    tft.startWrite();
    tft.setAddrWindow(x, y, w, h);
    for (int j = 0; j < h; ++j) {
        for (int i = 0; i < w; ++i) {
            row[i] = pgm_read_word(&bitmap[w*j + i]);
        }
        tft.writePixels(row, w);
    }
    tft.endWrite();
}
```

- Draws a bitmap (bitmap) on the TFT display at coordinates (x, y) with dimensions (w, h). The bitmap is read from PROGMEM using pgm_read_word.

Screen Display Functions

Functions like screenWelcome(), screenWaitingID(), screenPressToStart(), screenOverlayID(), etc., are used to display different screens or overlays on the TFT display using bitmaps or text.

8. Ui_bitmap.h

This module contains BitMap data used to print different screens on the TFT screen when states are changed.

9. utils

```
unsigned long tt2, tt3; // Global variables for timing, possibly for debugging
int data_count = 0; // Counter for the number of data points recorded

String temp_filename = ""; // Temporary filename for storing generated filenames
File logFile; // File object for logging ECG data

void interruptECG() {
  if (logFile) {
    logFile.close(); // Close the log file if it's open
  }
  stopECG(); // Stop ECG recording (assuming this is a function you've defined)
  Serial.println("ECG Recording Interrupted");
}
```

- **Global Variables**: tt2, tt3, and data_count are used for timing and data counting purposes. Ensure they are used consistently throughout your code.

- **Interrupt Handling**: interruptECG() closes the log file and stops ECG recording. It's triggered when an interrupt condition occurs, possibly when a button (BTN_X) is pressed.

```
String recordECG(user_info* user_data, int
duration_s = 300) {
    initSD();
    initECG();

    String id = user_data->ID;
    String date_time = user_data->Date_time
;

    String filename = "VNS_ECG_Data/"
 + id + '/' + date_time + '_' + id;

    int file_index = 1;
    temp_filename = filename + "_0";
    while(SD.exists(temp_filename)) {
        temp_filename = filename + '_'
 + file_index;
        file_index++;
    }
    filename = temp_filename + ".txt";

    logFile = SD.open
(filename, FILE_WRITE);
    data_count = 0;

    if (logFile) {
        Serial.println(
"All modules initalized successfully");
        Serial.print("Logging data for ");
        Serial.print(duration_s);
        Serial.println(" seconds");
        Serial.print("ID: ");
        Serial.println(id);

        logFile.println(user_data->ID);
        logFile.println(user_data->
Date_time);
        logFile.println(user_data->Name);
        logFile.println(user_data->Age);
        logFile.println(user_data->Gender);
        logFile.println(user_data->Weight);
```

```
 unsigned long start_time = millis();
        unsigned long t1 = millis();
        // keep recording for duration_s
        while(millis
() - start_time <= duration_s*1000) {

            while(Serial2.available()) {
//                Serial.println(Serial2.av
ailable());
                bmd.process();
                if(bmd.data_available()) {
                    int16_t value = bmd.
get_raw_value();
                    ++data_count;
                    logFile.println(value);
                }
            }

            if(millis
() - t1 >= UPDATE_DURATION*1000) {
                uint8_t hr = bmd.
get_heart_rate();
                uint16_t
 seconds = duration_s - ((millis
() - start_time)/1000);
                Serial.print("Heartrate: "
);
                Serial.print(hr);
                Serial.print(" | ");
                Serial.print(
"Time Remaining: ");
                Serial.print(seconds);
                Serial.println(" seconds");
                screenOverlayStats(String
(hr), String(seconds));

                t1 = millis();
            }

            if (!digitalRead(BTN_X)) {
              break;
            }
        }

        logFile.close();
        stopECG();
        Serial.println(
"Recording Completed");
        Serial.print(
"Total data points took: ");
        Serial.println(data_count);
```

```
 }
    else {
        stopECG();
        criticalError(
"Unable to open Log File. Recording Terminate
d"
);
    }

    return filename;
}
```

- **Initialization**: initSD() and initECG() initialize the SD card and ECG module respectively.
- **Filename Generation**: A unique filename is generated based on the user's ID and date/time.

- **Logging Data**: ECG data and user information are logged to the SD card file.
- **Recording Loop**: Records ECG data for duration_s seconds, updating the TFT display (screenOverlayStats) and checking for interruptions (BTN_X press).
- **Error Handling**: If the log file cannot be opened, a critical error is triggered (criticalError() function).

```
void demoRead(String filename, int16_t *demo_data) {
    File dataFile = SD.open(filename);

    if (dataFile) {
        // Skip the first 2500 lines (assuming they are headers or irrelevant data)
        for (int i = 0; i < 2500; ++i) {
            dataFile.readStringUntil('\n');
        }

        // Read actual data points into demo_data array
        for (int i = 0; i < 320; ++i) {
            int16_t val = dataFile.readStringUntil('\n').toInt();
            demo_data[i] = val;
        }

        dataFile.close(); // Close the file after reading
    }
}
```

- **File Opening**: Opens the specified file (filename) on the SD card.
- **Skipping Headers**: Skips the first 2500 lines (adjust as necessary based on your data format).
- **Reading Data**: Reads 320 data points from the file into the demo_data array.
- **File Closing**: Closes the file after reading is complete.

```
void criticalError(String error) {
    Serial.print("ERROR: ");
    Serial.println(error);
    clearTFT(); // Clear TFT display
    screenText("ERROR: " + error); // Display error message on TFT
    while (1); // Infinite loop to halt execution
    return;
}
```

- **Error Handling**: Outputs the error message to the serial monitor and displays it on the TFT screen.
- **Infinite Loop**: Halts program execution in case of a critical error, indicated by an infinite loop.

10. Wifi

```
#include <WiFi.h>
#include <WebServerSecure.h>
#include <FS.h>
#include <LEAmDNS.h>
```

These lines include the necessary libraries:

- WiFi.h is for handling Wi-Fi connectivity.
- WebServerSecure.h provides secure web server functionality.
- FS.h is for handling the file system.
- LEAmDNS.h is for mDNS (Multicast DNS) service, allowing network devices to find each other.

```
const char* ssid = "VNSMonitor";
const char* password = "vagaltone";
struct user_info temp_user_data;
const String ID1;
```

- ssid and password store the network credentials for the Wi-Fi access point.
- temp_user_data is a structure to store user information.
- ID1 is a constant String variable.

```
static const char serverCert[] PROGMEM = R"EOF(
-----BEGIN CERTIFICATE-----
...
-----END CERTIFICATE-----
)EOF";

static const char serverKey[] PROGMEM = R"EOF(
-----BEGIN PRIVATE KEY-----
...
-----END PRIVATE KEY-----
)EOF";
```

These blocks store the SSL certificate and private key required for secure HTTPS communication. The PROGMEM keyword ensures the data is stored in flash memory instead of RAM.

#NOTE:SSL certificate and private key have a validity period in our case it is 365 days so after that one needs to regenerate self signed SSL certificate and Private key to do that please follow Appendix A

```
WebServerSecure server(443);
ServerSessions serverCache(5);
```

- server initializes a secure web server on port 443, the default HTTPS port.
- serverCache is used to cache SSL sessions for faster TLS handshakes.

```
struct user_info getID() {
  while(1) {
    server.handleClient();
    if (temp_user_data.ID != "") {
      break;
    }
  }
  return temp_user_data;
}
```

getID continuously handles client requests until temp_user_data.ID is not empty, indicating user data has been received.

```
void handleForm() {
  const char formHTML[] PROGMEM = R"rawliteral(
    <!DOCTYPE html>
    <html>
    <head>
      ...
    </head>
    <body>
      ...
      <script>
        ...
      </script>
    </body>
    </html>
  )rawliteral";

  server.send(200, "text/html", formHTML);
}
```

handleForm sends an HTML form to the client when accessed. The form includes input fields for ID, Name, Date_time, Duration, Age, Gender, and Weight, along with a submit button. The form is styled with CSS and contains JavaScript to handle form submission and URL query parameters.

```
void handleGet() {
  String ID = server.arg("ID"
);
  String Name = server.arg(
"Name");
  String Date_time = server.
arg("Date_time");
  String Duration = server.arg
("Duration");
  String Age = server.arg(
"Age");
  String Gender = server.arg(
"Gender");
  String Weight = server.arg(
"Weight");
  temp_user_data.ID=ID;
  temp_user_data.Name=Name;
  temp_user_data.Date_time=
Date_time;
  temp_user_data.Duration=
Duration.toInt();
  temp_user_data.Age= Age;
  temp_user_data.Gender=
Gender;
  temp_user_data.Weight=
Weight;
  Serial.println("ID: " + ID);
  Serial.println("Name: " +
Name);
  Serial.println("Date_time: "
 + Date_time);
  Serial.println("Duration: "
 + Duration);
  Serial.println("Age: " + Age
);
  Serial.println("Gender: " +
Gender);
  Serial.println("Weight: " +
Weight);

  String response =
"<!DOCTYPE html><html><body><h
1>You entered:</h1>"
;
  response += "<p>ID: " + ID
 + "</p>";
  response += "<p>Name: " +
Name + "</p>";
  response +=
"<p>Current Time: " +
Date_time + "</p>";
  response += "<p>Duration: "
 + Duration + "</p>";
  response += "<p>Age: " + Age
 + "</p>";
  response += "<p>Gender: " +
Gender + "</p>";
  response += "<p>Weight: " +
Weight + "</p>";
  response += "</body></html>"
;


  server.send(200, "text/html"
, response);
}
```

handleGet handles the form submission by extracting parameters from the URL, updating temp_user_data, and printing the information to the serial monitor. It also sends an HTML response to the client displaying the submitted data.

```
bool startAP()
{
  delay(2000);//debug
  Serial.begin(115200);
  Serial.println(
"Setting up AP...");
  if (WiFi.softAP(ssid,
password)) {
    Serial.println(
"Access Point started successf
ully."
);
  } else {
    Serial.println(
"Failed to start Access Poin
t."
);
    return false;
  }
  configTime(3 * 3600, 0,
"pool.ntp.org",
"time.nist.gov");
  delay(2000);
  Serial.println("IP time?");
  // Give some time for the AP
to be set up
  delay(1000);
  Serial.println("IP time");

  // Print the IP address
  IPAddress IP = WiFi.softAPIP
();
  Serial.print(
"AP IP address: ");
  Serial.println(IP);

  if (MDNS.begin("picow")) {
Serial.println(
"MDNS responder started"); }

  server.getServer().
setRSACert(new BearSSL::
X509List(serverCert), new
BearSSL::PrivateKey
(serverKey));

  // Cache SSL sessions to acc
elerate the TLS handshake.
  server.getServer().setCache
(&serverCache);

  // Define request handlers
  server.on("/", handleForm);
  server.on("/get", handleGet
);

  // Start the server
  server.begin();
  Serial.println(
"HTTPS server started");

  // make sure temp_user_data
is empty
  memset(&temp_user_data, 0,
sizeof(temp_user_data));

  return true;

}
```

startAP initializes the Wi-Fi access point and configures the secure web server:

- Sets up the access point with ssid and password.
- Configures NTP (Network Time Protocol) servers for time synchronization.
- Prints the IP address of the access point.
- Starts mDNS responder.
- Configures SSL certificates and session cache.

- Defines request handlers for the root ("/") and "/get" endpoints.
- Starts the secure web server.
- Clears temp_user_data.

```
void stopAP() {
  WiFi.softAPdisconnect(true);
  Serial.println("SoftAP Stopped");
}
```

stopAP disconnects and stops the Wi-Fi access point.

11. z_vns_main

This code represents a finite state machine (FSM) for an VNS monitoring system. The system goes through various states such as initialization, waiting for user input, recording ECG data, viewing results, and sleeping. Below is an explanation of each part of the code:

State Definitions

The states of the FSM are defined using constants:

```
#define S0_INITIALIZING 0
#define S1_WAITING_ID 1
#define S2_PRESS_TO_START 2
#define S3_RECORDING 3
#define S4_VIEW 4
#define S5_SLEEP 5
```

Global Variables

Global variables are declared to store user data, the current filename for storing ECG data, and the current and previous states:

```
struct user_info user_data;
String filename = "";
int state = S0_INITIALIZING;
int screen_state = -1;
```

Reset Handler

A reset handler is defined to interrupt the ECG recording and reset the state to initializing:

```
void resetHandler() {
  interruptECG();
  state = S0_INITIALIZING;
}
```

Setup Function

The setup() function initializes the system, buttons, and TFT screen, and sets up the serial communication:

```
void setup() {
  delay(1000);
  Serial.begin(115200);

  initButtons();
  initTFT();
  clearTFT();
//  attachInterrupt(digitalPinToInterrupt(BTN_X), resetHandler, RISING);
}
```

Loop Function

The loop() function implements the FSM, transitioning between different states based on user interactions and system operations:

```
void loop() {


  if (state == S0_INITIALIZING
) {
    if (screen_state !=
S0_INITIALIZING) {
        screenWelcome();
        screen_state =
S0_INITIALIZING;
    }

    if (startAP()) {
      // ap started successful
ly
      state = S1_WAITING_ID;
    }
    else {
      // ap couldn't start
      state = S0_INITIALIZING;
    }
  }


  if (state == S1_WAITING_ID
) {
    if (screen_state !=
S1_WAITING_ID) {
        screenWaitingID();
        screen_state =
S1_WAITING_ID;
    }

    user_data = getID();
    if (user_data.ID == "") {
      Serial.println(
"ID not received");
//      state = S1k_SLEEP;
    }
    else {
      Serial.print(
"ID Received: ");
      Serial.println(user_data
.ID);
      stopAP();
      state =
S2_PRESS_TO_START;
    }
  }
```

```
 if (state ==
S2_PRESS_TO_START) {
    if (screen_state !=
S2_PRESS_TO_START) {
        screenPressToStart();
        screenOverlayID(
user_data.Name);
        screen_state =
S2_PRESS_TO_START;
    }

    while(1) {
      btnx.update();
      btnr.update();
      if(btnr.justPressed()) {
        state =
S0_INITIALIZING;
        break;
      }
      if(btnx.justPressed()) {
        state = S3_RECORDING;
        break;
      }
      delay(10);
    }
  }


  if (state == S3_RECORDING) {
    if (screen_state !=
S3_RECORDING) {
        screenScanning();
        screen_state =
S3_RECORDING;
    }

    // attach interrupt or wha
tever
    filename = recordECG(&
user_data, user_data.Duration
);

    state = S4_VIEW;
  }


  if (state == S4_VIEW) {
    if (screen_state !=
S4_VIEW) {
        screenCompleted();
        screen_state = S4_VIEW
;
    }
```

```
int16_t demo_data[320];
    demoRead(filename,
demo_data);
    printGraph(demo_data);

    while(1) {
      btnx.update();
      btnr.update();
      if(btnx.justPressed()) {
        state = S5_SLEEP;
        break;
      }
      if(btnr.justPressed()) {
        removeFile(filename);
        state =
S2_PRESS_TO_START;
        break;
      }
      delay(10);
    }
  }


  if (state == S5_SLEEP) {
    if (screen_state !=
S5_SLEEP) {
//        clearTFT();
        screenSleep();
        screen_state =
S5_SLEEP;
    }

    // reset user data
    memset(&user_data, 0,
sizeof(user_data));
    filename = "";

    while(1) {
      btnx.update();
      btnr.update();
      if(btnr.justPressed()) {
        state =
S0_INITIALIZING;
        break;
      }
      if(btnx.justPressed()) {
        state =
S0_INITIALIZING;
        break;
      }
      delay(10);
    }
  }

}
```

Explanation of the FSM

1. **S0_INITIALIZING:** Initializes the system and starts the Access Point (AP). If AP starts successfully, it moves to S1_WAITING_ID. Otherwise, it stays in the initializing state.
2. **S1_WAITING_ID:** Displays a screen waiting for user ID input. If a valid ID is received, it stops the AP and moves to S2_PRESS_TO_START.
3. **S2_PRESS_TO_START:** Displays a screen prompting the user to press a button to start recording. If the button is pressed, it moves to S3_RECORDING.
4. **S3_RECORDING:** Starts the ECG recording and moves to S4_VIEW after recording is complete.
5. **S4_VIEW:** Displays the recorded ECG data and provides options to either delete the recording and go back to S2_PRESS_TO_START or move to S5_SLEEP.
6. **S5_SLEEP:** Displays a sleep screen and waits for a button press to reinitialize the system.

   The code provides a comprehensive framework for an ECG monitoring system, handling user interactions, state transitions, and ECG data recording and display.

3.2) VNS Manager Code

```
import tkinter as tk
from tkinter import ttk
from tkinter import filedialog, messagebox
# from data_utils import generate_csv
from pathlib import Path
import shutil
import neurokit2 as nk
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from tkcalendar import DateEntry
from datetime import timedelta
import os
from neurokit2.hrv.hrv_utils import
 _hrv_format_input
from neurokit2.hrv.intervals_utils import
 _intervals_successive

# Initialize the columns for the DataFrame
vns_df_cols = {
    'filename': [],
    'id': [],
    'date': [],
    'name': [],
    'age': [],
    'gender': [],
    'weight': [],
    'ecg_quality': [],
    'mean_hr': [],
    'mean_ibi': [],
    'rmssd': [],
    'lf/hf': [],
    'cvi': []
}
```

```
# Directory paths and constants
DATA_DIR = "./VNS_Monitor_Storage/"
SD_DIRNAME = "VNS_ECG_Data"
SAMPLE_RATE = 512
DAYS_DELTA = 7

# This is code poorly written due to a stri
ct timeline and is by no means a measure of
anyone's capabilities

# Main class to manage the VNS data
class VNSManager():
    def __init__(self, root):
        self.root = root
        self.elements = []
        root.title(
"VNS Monitor Data Manager")

        # Create info frame
        info_frame = tk.LabelFrame(root,
text="", height=40)
        info_frame.pack(fill="x", padx=10,
pady=10)

        info_var = tk.StringVar()
        self.info_var = info_var
        info_var.set(
"Welcome to VNS Monitor")
        info_label = tk.Label(info_frame,
textvariable=info_var, anchor=tk.CENTER,
pady=5, padx=5)
        info_label.pack(pady=5)

        # Create tree frame
        tree_frame = tk.Frame(root)
        tree_frame.pack(fill="both", expand
=True, padx=10, pady=10)

        # Add a scrollbar to the tree frame
        tree_scroll = tk.Scrollbar(
tree_frame)
        tree_scroll.pack(side=tk.RIGHT,
fill='y')

        # Create the tree view
        my_tree = ttk.Treeview(tree_frame,
yscrollcommand=tree_scroll.set, selectmode=
"extended")
        my_tree.pack(fill="both", expand=
True)
        self.my_tree = my_tree
```

```
    # Configure the scrollbar
        tree_scroll.config(command=my_tree.
yview)

        # Define columns for the tree view
        my_tree['columns'] = ("Filename",
"ID", "Name", "Date")

        my_tree.column("#0", width=25,
stretch=False)
        my_tree.column("Filename", anchor=
tk.W, width=120)
        my_tree.column("ID", anchor=tk.W,
width=80)
        my_tree.column("Name", anchor=tk.W
, width=80)
        my_tree.column("Date", anchor=tk.W
, width=80)

        my_tree.heading("#0", text="",
anchor=tk.W)
        my_tree.heading("Filename", text=
"Filename", anchor=tk.W)
        my_tree.heading("ID", text="ID",
anchor=tk.W)
        my_tree.heading("Name", text="Name"
, anchor=tk.W)
        my_tree.heading("Date", text="Date"
, anchor=tk.W)

        # Initialize control frame
        control_frame = tk.LabelFrame(root
, text="Controls", height=100)
        control_frame.pack(fill="x", padx=
10, pady=10)


# Create buttons in the control frame
        load_btn = ttk.Button(control_frame
, text='Load & Process Data', command=self.
loadData)
        load_btn.pack(side=tk.LEFT, expand=
True, padx=10, pady=10)
        self.elements.append(load_btn)

        export_btn = ttk.Button(
control_frame, text='Export CSV', command=
self.exportData)
        export_btn.pack(side=tk.LEFT,
expand=True, padx=10, pady=10)
        self.elements.append(export_btn)

        average_btn = ttk.Button(
control_frame, text="Average Trend",
command=self.averageData)
        average_btn.pack(side=tk.LEFT,
expand=True, padx=10, pady=10)
        self.elements.append(average_btn)
```

```
 # Initial setup
        self.disable_tree()
        self.process_data()
        self.populate_tree()
        self.enable_tree()


# Method to populate the tree view with dat
a
    def populate_tree(self):
        for idx, entry in self.vns_df.
iterrows():
            user_id = entry['id']
            if not self.my_tree.exists(
user_id):
                self.my_tree.insert(parent=
'', index='end', iid=user_id, values=(
user_id, user_id, entry['name'], ''))
            self.my_tree.insert(parent=
user_id, index='end', iid=entry['filename'
], values=(entry['filename'], user_id,
entry['name'], entry['date']))


# Bind double-click event to the tree view
items
        self.my_tree.bind("<Double-1>",
self.onItemClick)

    # Method to clear the tree view
    def clear_tree(self):
        self.my_tree.delete(*self.my_tree.
get_children())


# Method to process data and update the dis
play
    def process_data(self):
        self.info_var.set(
"Processing Data...")
        self.root.update_idletasks()
        self.vns_df, self.csv_path =
generate_csv(DATA_DIR)
        self.info_var.set(
"Welcome to VNS Monitor")

    # Method to export data to CSV
    def exportData(self):
        csv_src = self.csv_path
        csv_dest = filedialog.
asksaveasfilename(initialfile=(
"VNS_Monitor_data"+'.csv'),
defaultextension='.csv')
        shutil.copy2(csv_src, csv_dest)
```

```
# Method to show average data trends within a date range
    def averageData(self):
        average_popup = tk.Toplevel()
        tk.Label(average_popup, text="From Date").grid(row=0, column=0, padx=5, pady=5)
        from_date = DateEntry(average_popup, year=2024)
        from_date.grid(row=0, column=1, padx=5, pady=5)

        tk.Label(average_popup, text="To Date").grid(row=1, column=0, padx=5, pady=5)
        to_date = DateEntry(average_popup, year=2024)
        to_date.grid(row=1, column=1, padx=5, pady=5)

        show_btn = tk.Button(average_popup, text="Show Average Trend", command=lambda: self.showaverage(from_date.get_date(), to_date.get_date()))
        show_btn.grid(row=2, column=0, columnspan=2, padx=5, pady=5)

        return
```

```
# Method to display average trends between
two dates
    def showaverage(self, from_date,
to_date):
        avg_dict = {'mean_ibi':[], 'rmssd'
:[], 'lf/hf':[], 'cvi':[], 'date':[]}
        avg_df = pd.DataFrame(avg_dict)
        date_d1 = from_date
        date_d2 = from_date + timedelta(
days=DAYS_DELTA)
        idx = 0

        while (date_d2 <= to_date):
            filtered_df = self.vns_df[(self
.vns_df['date'] >= str(date_d1)) & (self.
vns_df['date'] <= str(date_d2))]
            avg_entry = {
                'mean_ibi': filtered_df.loc
[:, 'mean_ibi'].mean(),
                'rmssd': filtered_df.loc
[:, 'rmssd'].mean(),
                'lf/hf': filtered_df.loc
[:, 'lf/hf'].mean(),
                'cvi': filtered_df.loc[:,
'cvi'].mean(),
                'date': str(date_d1)
            }
            if avg_entry['mean_ibi'] ==
avg_entry['mean_ibi']: # NaN check
                print('adding')
                avg_df = pd.concat([avg_df
, pd.DataFrame([avg_entry])], ignore_index=
True)
            date_d1 = date_d2
            date_d2 = date_d1 + timedelta(
days=DAYS_DELTA)
            idx += 1

        fig, axs = plt.subplots(2, 2)
        fig.suptitle(f"Average trend from {
from_date} to {to_date}", fontweight="bold"
)

        axs[0, 0].plot(avg_df['date'],
avg_df['mean_ibi'])
        axs[0, 0].set_title('Mean IBI')
        axs[0, 0].set_ylabel(
"mean_ibi (ms)")

        axs[0, 1].plot(avg_df['date'],
avg_df['rmssd'], color='tab:orange')
        axs[0, 1].set_title('RMSSD')
        axs[0, 1].set_ylabel("rmssd (ms)")

        axs[1, 0].plot(avg_df['date'],
avg_df['lf/hf'], color='tab:green')
        axs[1, 0].set_title('LF/HF')
        axs[1, 0].set_ylabel('lf/hf')

        axs[1, 1].plot(avg_df['date'],
avg_df['cvi'], color='tab:red')
        axs[1, 1].set_title('CVI')
        axs[1, 1].set_ylabel('cvi')

        fig.autofmt_xdate()
        plt.show()
        return
```

```
# Method to load data from a selected direc
tory
    def loadData(self):
        load_popup = tk.Toplevel()
        self.load_popup = load_popup
        self.load_popup.geometry("450x200")

        tk.Label(load_popup, text="ID").
grid(row=0, column=0, padx=5, pady=5)
        id_entry = tk.Entry(load_popup)
        id_entry.grid(row=0, column=1, padx
=5, pady=5)

        tk.Label(load_popup, text="Name").
grid(row=1, column=0, padx=5, pady=5)
        name_entry = tk.Entry(load_popup)
        name_entry.grid(row=1, column=1,
padx=5, pady=5)

        tk.Label(load_popup, text="Age").
grid(row=2, column=0, padx=5, pady=5)
        age_entry = tk.Entry(load_popup)
        age_entry.grid(row=2, column=1,
padx=5, pady=5)

        tk.Label(load_popup, text="Gender"
).grid(row=3, column=0, padx=5, pady=5)
        gender_var = tk.StringVar()
        gender_combo = ttk.Combobox(
load_popup, textvariable=gender_var)
        gender_combo['values'] = ("Male",
"Female", "Other")
        gender_combo.grid(row=3, column=1,
padx=5, pady=5)

        tk.Label(load_popup, text=
"Weight (kg)").grid(row=4, column=0, padx=5
, pady=5)
        weight_entry = tk.Entry(load_popup)
        weight_entry.grid(row=4, column=1,
padx=5, pady=5)

        self.load_entries = {
            'id_entry': id_entry,
            'name_entry': name_entry,
            'age_entry': age_entry,
            'gender_combo': gender_combo,
            'weight_entry': weight_entry
        }
```

```
  select_button = tk.Button(load_popup,
text="Select Data Folder", command=self.
openFilePicker)
        select_button.grid(row=5, column=0
, columnspan=2, padx=5, pady=5)

        save_button = tk.Button(load_popup
, text="Save", command=self.saveUserDetails
)
        save_button.grid(row=6, column=0,
columnspan=2, padx=5, pady=5)

    # Method to open a file picker dialog
    def openFilePicker(self):
        dirname = filedialog.askdirectory(
initialdir=Path(os.getcwd()))
        self.dirname = dirname
        return


# Method to save user details and process t
he loaded data
    def saveUserDetails(self):
        entries = self.load_entries
        user_id = entries['id_entry'].get()
        user_name = entries['name_entry'
].get()
        user_age = entries['age_entry'
].get()
        user_gender = entries[
'gender_combo'].get()
        user_weight = entries[
'weight_entry'].get()

        if user_id == "" or user_name == ""
:
            messagebox.showinfo(
"Information",
"ID and Name cannot be empty")
            return

        # Process user details
        input_details = {
            'user_id': user_id,
            'user_name': user_name,
            'user_age': user_age,
            'user_gender': user_gender,
            'user_weight': user_weight,
        }
```

```
self.processUserData(self.dirname,
input_details)
        self.load_popup.destroy()

    # Method to process user data
    def processUserData(self, folder_path,
input_details):
        print(input_details)
        data_filenames = [f for f in os.
listdir(folder_path) if f.endswith(".txt")]

        user_id = input_details['user_id']
        user_name = input_details[
'user_name']
        user_age = input_details['user_age'
]
        user_gender = input_details[
'user_gender']
        user_weight = input_details[
'user_weight']

        for filename in data_filenames:
            ecg_quality = 'Good'
            mean_hr = 0
            mean_ibi = 0
            rmssd = 0
            lfhf = 0
            cvi = 0

            # Read ECG data
            ecg_path = os.path.join(
folder_path, filename)
            with open(ecg_path, 'r') as
ecg_file:
                ecg_data = ecg_file.
readlines()

            ecg_data = list(map(float,
ecg_data))
```

```
    try:
                rpeaks, _ = nk.ecg_peaks(
ecg_data, sampling_rate=SAMPLE_RATE)
                rpeaks = rpeaks[
'ECG_R_Peaks']
                intervals
 = _intervals_successive(rpeaks,
sampling_rate=SAMPLE_RATE)

                mean_hr = 60 / np.mean(
intervals)
                mean_ibi = np.mean(
intervals) * 1000
                rmssd = np.sqrt(np.mean(np
.diff(intervals) ** 2)) * 1000
                cvi = np.std(intervals) /
np.mean(intervals) * 100
                lf, hf = nk.hrv_frequency(
intervals, sampling_rate=SAMPLE_RATE)['LF'
], nk.hrv_frequency(intervals,
sampling_rate=SAMPLE_RATE)['HF']
                lfhf = lf / hf if hf != 0
else 0

                if len(rpeaks) < 10:
                    ecg_quality = 'Poor'

            except Exception as e:
                print(f
"Error processing file {filename}: {e}")
                ecg_quality = 'Poor'


# Append the data to the global DataFrame
            entry = {
                'filename': filename,
                'id': user_id,
                'name': user_name,
                'date': filename.split('_'
)[0],
                'age': user_age,
                'gender': user_gender,
                'weight': user_weight,
                'ecg_quality': ecg_quality,
                'mean_hr': mean_hr,
                'mean_ibi': mean_ibi,
                'rmssd': rmssd,
                'lf/hf': lfhf,
                'cvi': cvi
            }
```

```
 self.vns_df = self.vns_df.append(entry,
ignore_index=True)
        self.clear_tree()
        self.populate_tree()
        return

    # Method to disable the tree view
    def disable_tree(self):
        self.my_tree['selectmode'] = "none"
        for elem in self.elements:
            elem.config(state="disabled")

    # Method to enable the tree view
    def enable_tree(self):
        self.my_tree['selectmode'] =
"extended"
        for elem in self.elements:
            elem.config(state="normal")


# Method to handle item clicks in the tree
view
    def onItemClick(self, event):
        item = self.my_tree.selection()[0]
        parent_item = self.my_tree.parent(
item)
        if parent_item == '':
            child_items = self.my_tree.
get_children(item)
            for child in child_items:
                self.my_tree.detach(child)
            self.my_tree.delete(item)
        else:
            parent = self.my_tree.parent(
item)
            self.my_tree.detach(item)
        self.root.update_idletasks()
        return
```

```
# Generate CSV from the VNS data
def generate_csv(data_dir):
    global vns_df_cols
    vns_df = pd.DataFrame(vns_df_cols)
    data_files = [f for f in os.listdir(
data_dir) if f.endswith('.txt')]

    for data_file in data_files:
        data_path = os.path.join(data_dir,
data_file)
        with open(data_path, 'r') as file:
            data = file.readlines()

        data = [float(d.strip()) for d in
data]
        vns_df = vns_df.append({
            'filename': data_file,
            'id': data_file.split('_')[0],
            'date': data_file.split('_')[1
].split('.')[0],
            'name': data_file.split('_')[2
],
            'age': int(data_file.split('_'
)[3]),
            'gender': data_file.split('_')[
4],
            'weight': float(data_file
.split('_')[5]),
            'ecg_quality': 'Good' if len(
data) > 10 else 'Poor',
            'mean_hr': np.mean(data),
            'mean_ibi': np.mean(data),
            'rmssd': np.sqrt(np.mean(np
.diff(data) ** 2)),
            'lf/hf': np.mean(data) / np
.std(data) if np.std(data) != 0 else 0,
            'cvi': np.std(data) / np.mean(
data) * 100 if np.mean(data) != 0 else 0
        }, ignore_index=True)

    csv_path = os.path.join(data_dir,
'VNS_Monitor_data.csv')
    vns_df.to_csv(csv_path, index=False)
    return vns_df, csv_path

if __name__ == "__main__":
    root = tk.Tk()
    app = VNSManager(root)
    root.mainloop()
```

3.3)VNS QR Generator

```
import qrcode
# Import the qrcode library to generate QR
codes
from qrcode.image.pure import PyPNGImage
# Import the image factory for QR code imag
es
import tkinter as tk
# Import the tkinter library for GUI
from tkinter import ttk
# Import the themed widget set for tkinter
from tkinter import messagebox, filedialog

# Import messagebox and filedialog for dial
og boxes
import os
# Import the os module for file path operat
ions

# Function to generate the QR code
def generateQR(*args):
    try:
        # Extract and clean up user inputs
        t_user_id = ''.join(user_id.get().
split())
        t_name = ''.join(name.get().split
())
        t_age = ''.join(age.get().split())
        t_sex = ''.join(sex.get().split())
        t_weight = ''.join(weight.get().
split())
        t_duration = ''.join(duration.get
().split())

        # Check for mandatory fields
        if (t_user_id == "") or (t_name ==
"") or (t_duration == ""):
            messagebox.showerror(message=
"Error: Please fill in Name, ID & Duration"
)
            return


# Ensure all values are alphanumeric
        for t in [t_user_id, t_name, t_age
, t_sex, t_weight, t_duration]:
            if not t.isalnum():
                messagebox.showerror(
message="All values must be alphanumeric")
                return
```

```
# Generate QR code with the provided detail
s
        qr = qrcode.make(f
"https://192.168.42.1/?ID={t_user_id}&Name=
{t_name}&Duration={t_duration}&Age={t_age}
&Weight={t_weight}&Gender={t_sex}",
image_factory=PyPNGImage)
        filename = os.path.abspath(
filedialog.asksaveasfilename(initialfile=(
t_user_id+'.html'), defaultextension=
'.html'))
        img_filename = os.path.abspath(
filename[:-5] + '_qr.png')
        qr.save(img_filename)


# Create HTML content with user details and
QR code
        html_code = f"""
        <!DOCTYPE html>
        <html lang="en">
        <head>
            <meta charset="UTF-8">
            <meta name="viewport" content
="width=device-width, initial-scale=1.0">
            <title>VNS Monitor</title>
            <style>
                body {{
                    font-family: Arial, san
s-serif;
                    margin: 0;
                    padding: 40px;
                    background-color: #fff;
                    display: flex;
                    justify-content: space-
between;
                    align-items: flex-star
t;
                    height: 100vh;
                    box-sizing: border-box;
                }}
                .details {{
                    flex: 1;
                }}
                .qr-code img {{
                    width: 300px;
                    height: 300px;
                }}
                .title {{
                    font-size: 32px;
                    font-weight: bold;
                    margin-bottom: 20px;
                }}
            </style>
        </head>
```

```
    <body>
            <div class="details">
                <p class="title">VNS Monito
r</p>
                <p><strong>Name:</strong> {
t_name}</p>
                <p><strong>ID:</strong> {
t_user_id}</p>
                <p><strong>Age:</strong> {
t_age}</p>
                <p><strong>Gender:</strong>
{t_sex}</p>
                <p><strong>Weight:</strong>
{t_weight}</p>
            </div>
            <div class="qr-code">
                <img src="{img_filename}
" alt="QR Code">
            </div>
        </body>
        </html>
        """

        # Save the HTML file
        page = open(filename, 'w')
        page.write(html_code)
        page.close()

        messagebox.showinfo(message=
"User Form saved successfully")

    except ValueError:
        pass

# Set up the main tkinter window
root = tk.Tk()
root.title("VNS Monitor QR Code Generator")

# Create and configure the main frame
mainframe = tk.Frame(root)
mainframe.grid(column=0, row=0)
root.columnconfigure(0, weight=1)
root.rowconfigure(0, weight=1)
```

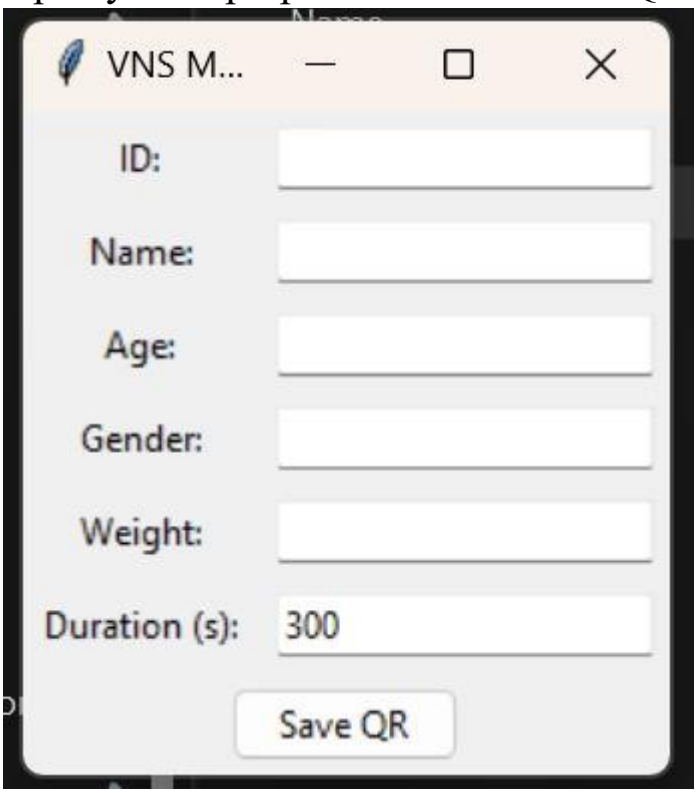
```
# Add labels and entry fields to the frame
ttk.Label(mainframe, text="ID:", anchor="e"
).grid(row=0, column=0)

user_id = tk.StringVar()
id_entry = ttk.Entry(mainframe,
textvariable=user_id)
id_entry.grid(row=0, column=1)

ttk.Label(mainframe, text="Name:", anchor=
"e").grid(row=1, column=0)

name = tk.StringVar()
name_entry = ttk.Entry(mainframe,
textvariable=name)
name_entry.grid(row=1, column=1)

ttk.Label(mainframe, text="Age:", anchor=
"e").grid(row=2, column=0)

age = tk.StringVar()
age_entry = ttk.Entry(mainframe,
textvariable=age)
age_entry.grid(row=2, column=1)

ttk.Label(mainframe, text="Gender:", anchor
="e").grid(row=3, column=0)

sex = tk.StringVar()
sex_entry = ttk.Entry(mainframe,
textvariable=sex)
sex_entry.grid(row=3, column=1)

ttk.Label(mainframe, text="Weight:", anchor
="e").grid(row=4, column=0)

weight = tk.StringVar()
weight_entry = ttk.Entry(mainframe,
textvariable=weight)
weight_entry.grid(row=4, column=1)

ttk.Label(mainframe, text="Duration (s):",
anchor="e").grid(row=5, column=0)

duration = tk.StringVar()
duration_entry = ttk.Entry(mainframe,
textvariable=duration)
duration_entry.insert(0, "300")
duration_entry.grid(row=5, column=1)
```

```
# Add a button to save the QR code
ttk.Button(mainframe, text="Save QR",
command=generateQR).grid(row=6, column=0,
columnspan=2)

# Add padding to each child of the mainfram
e
for child in mainframe.winfo_children():
    child.grid_configure(padx=5, pady=5)

# Bind the Enter key to the generateQR func
tion
root.bind("<Return>", generateQR)

# Start the main loop
root.mainloop()
```

User Manuel

Step 1:

Open your laptop and start the VNS QR code generator.

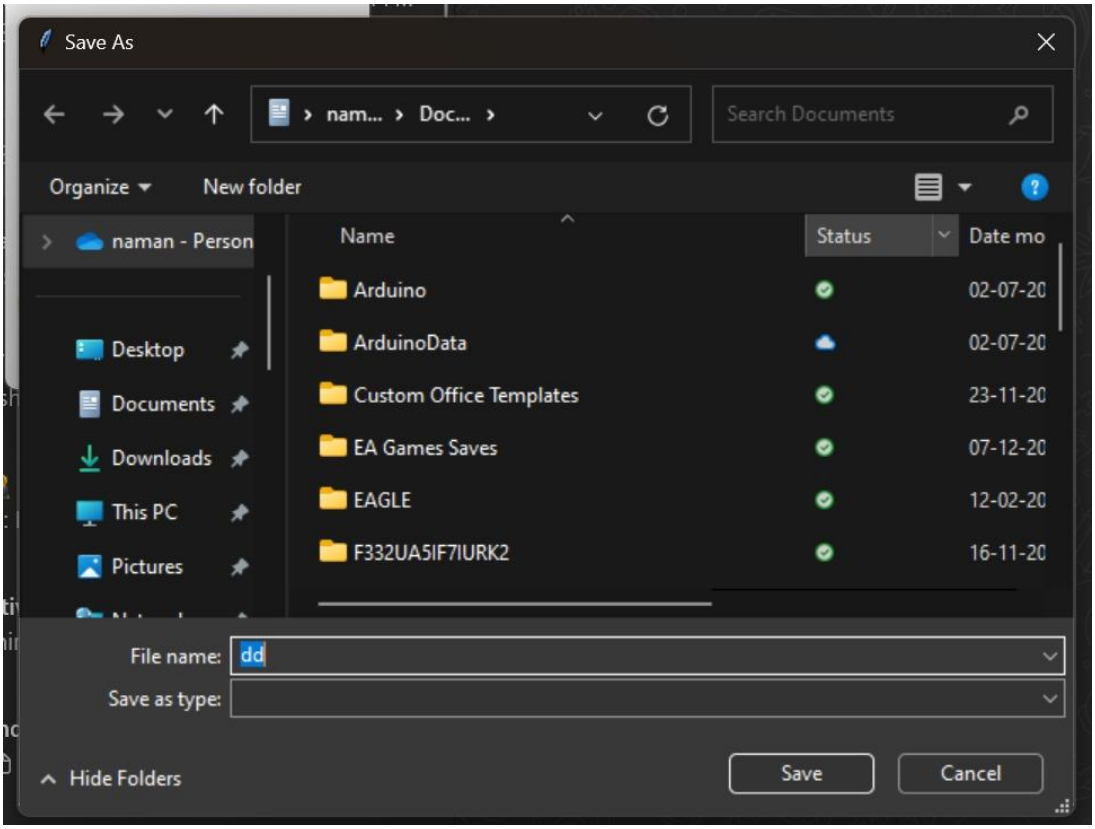


The above screen appears, enter your data accordingly.

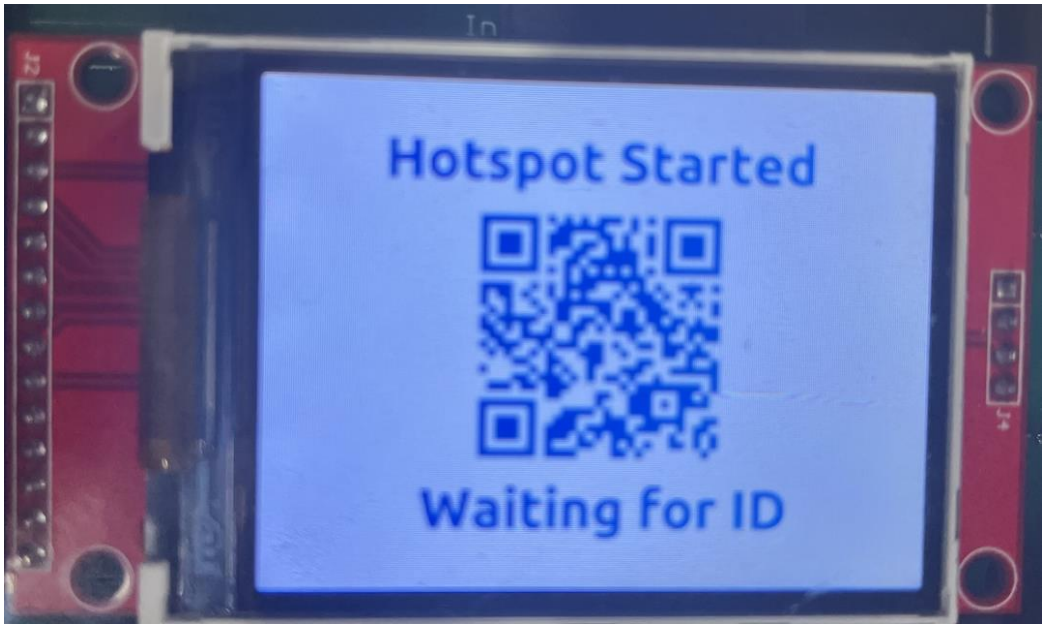
Press Save QR.

Step 2:

Save the file in the desired location like shown below.



Step 3:
Open the HTML form in the saved file and then plug in the VNS Monitor into the Power Bank/supply. Wait for the below screen to appear.



Scan the QR code to connect to the VNS Monitor network and turn off your mobile data.

Step 4:

VNS Monitor

Name: cr

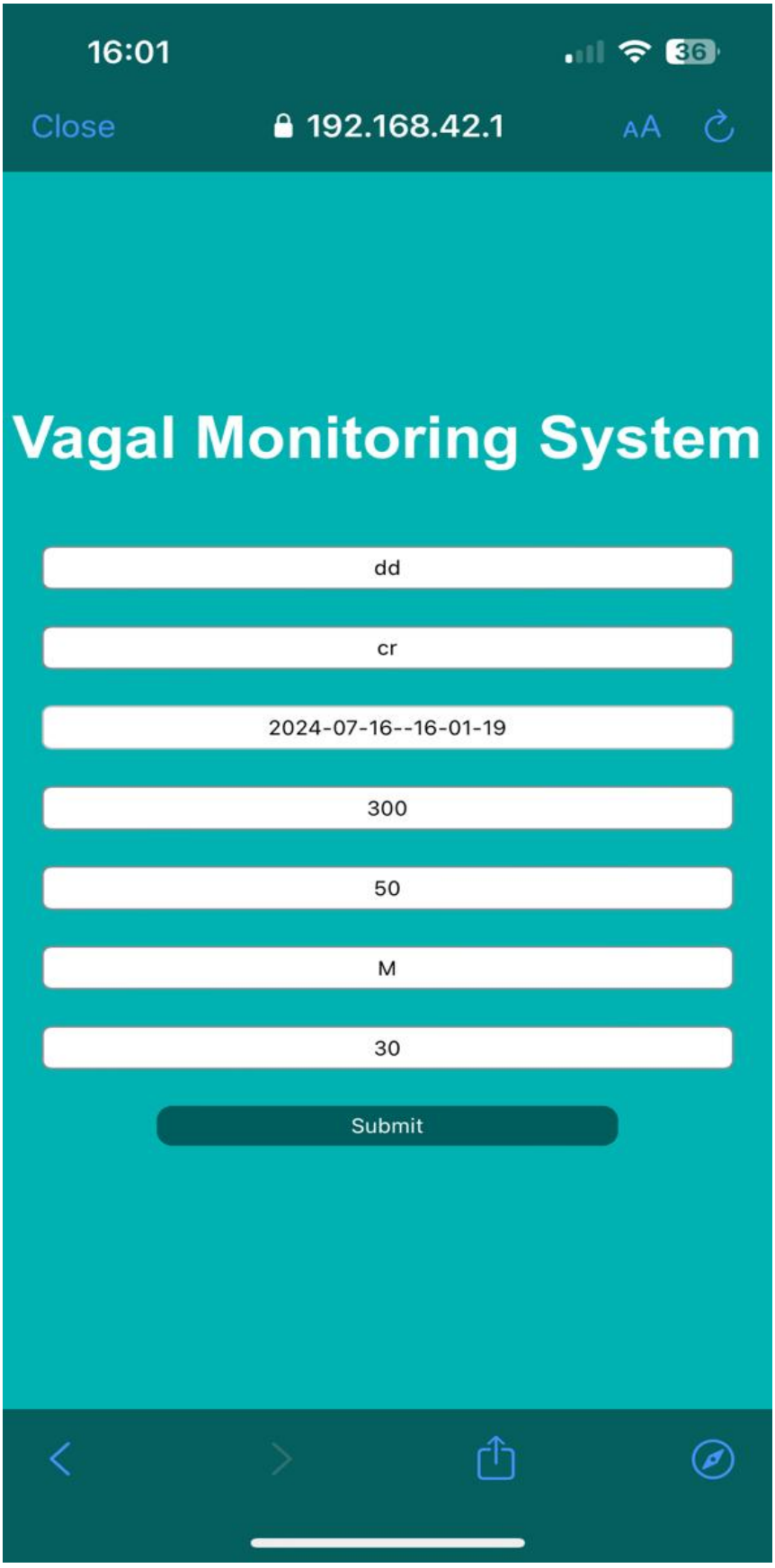
ID: dd

Age: 50

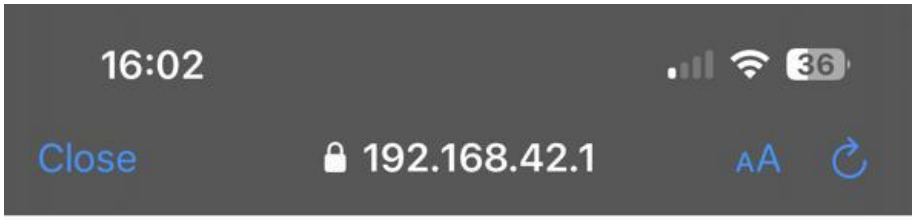
Gender: M

Weight: 30

Scan the QR present in the HTML form which contains the patient's data.



Press SUBMIT after verifying the details to configure the machine as shown below.



You entered:

ID: dd

Name: cr

Current Time: 2024-07-16--16-02-11

Duration: 300

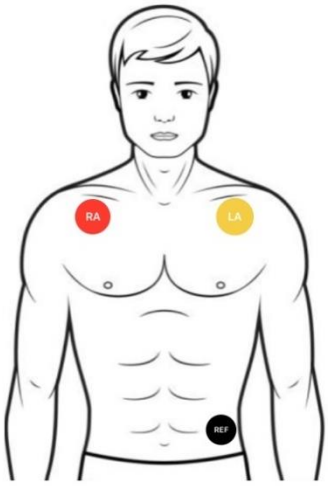
Age: 50

Gender: M

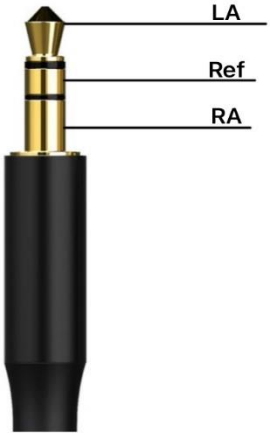
Weight: 30

Now your machine is ready to read your patients' ECG data!

Step 5:



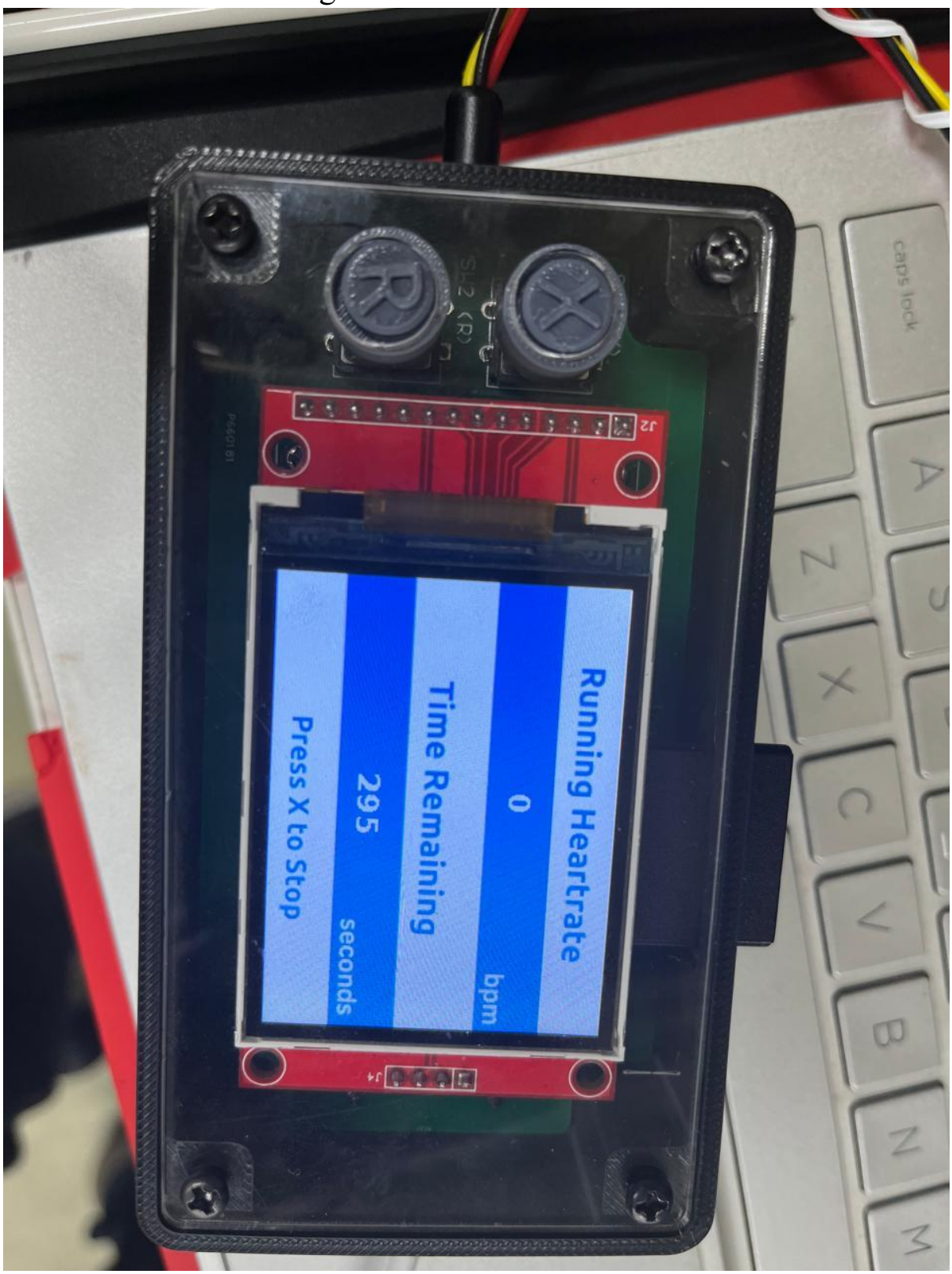
LA - Left arrhythmia
RA - Right arrhythmia
Ref - Reference



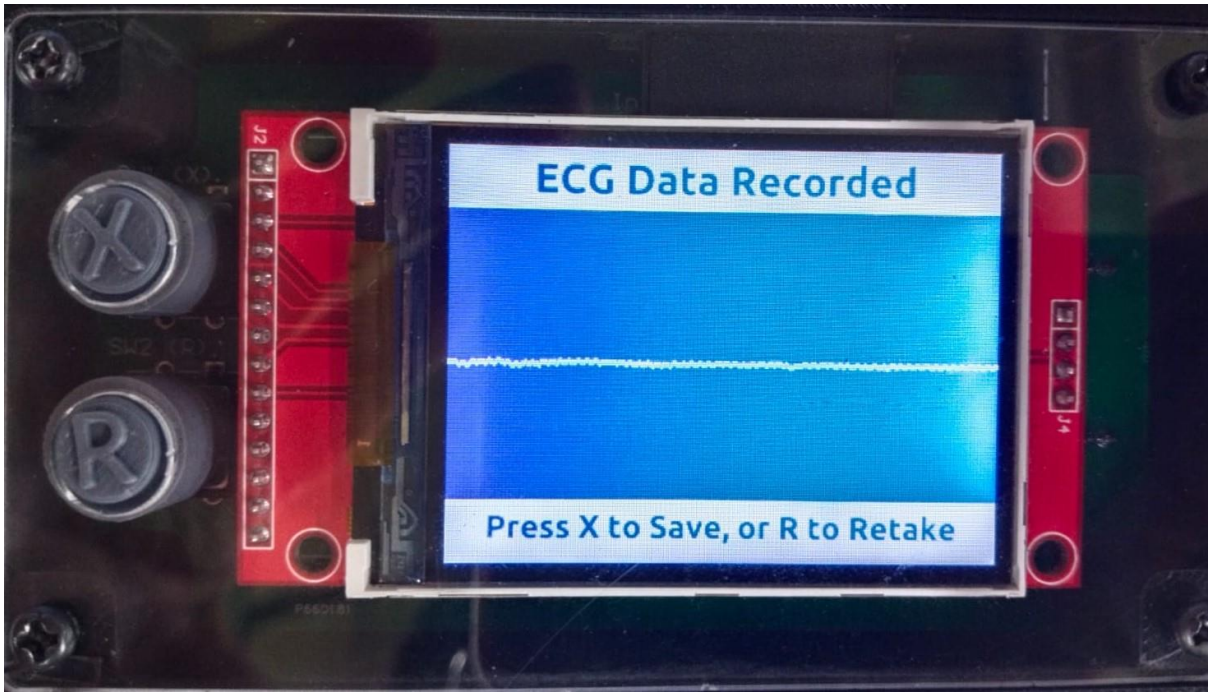
Connect the electrodes in the manner shown in the picture on the left hand side.
If you're using the default electrode cable you can follow the colour coding provided to attach the electrodes. If you've changed the cable, then check the connectivity for each part as shown in the right image and attach the electrodes according to the labels in the left image.

Step 6:

Press ‘X’ to start reading.



Once the reading is completed the following screen will appear.



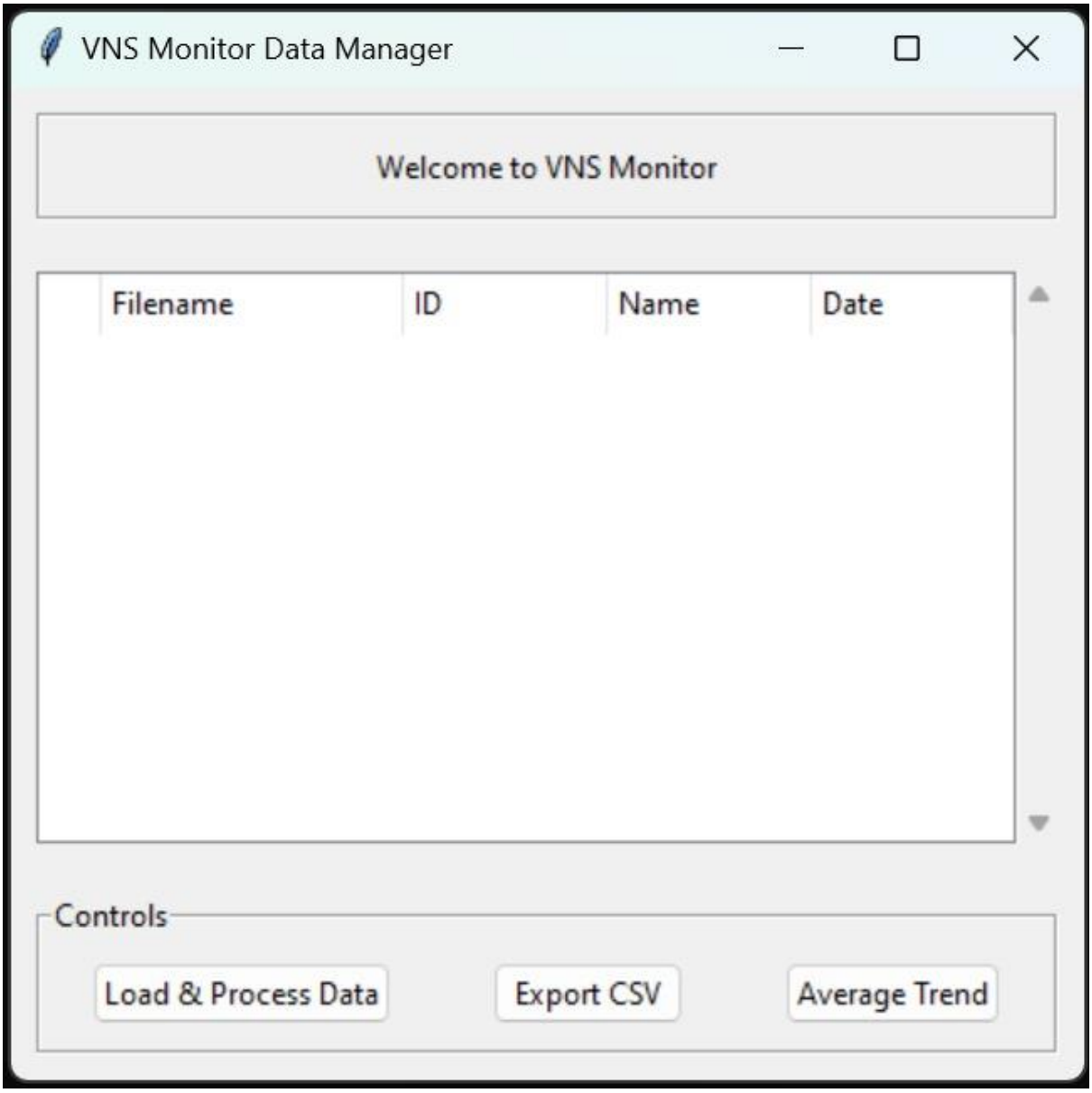
The screen will show you the data recorded for five seconds to help you confirm if the ECG data is correctly recorded and if you're satisfied with the recording, remove the electrodes from the patient and press X to save the data.

Press R to retake the reading.

After you press X, the device will go in standby mode.

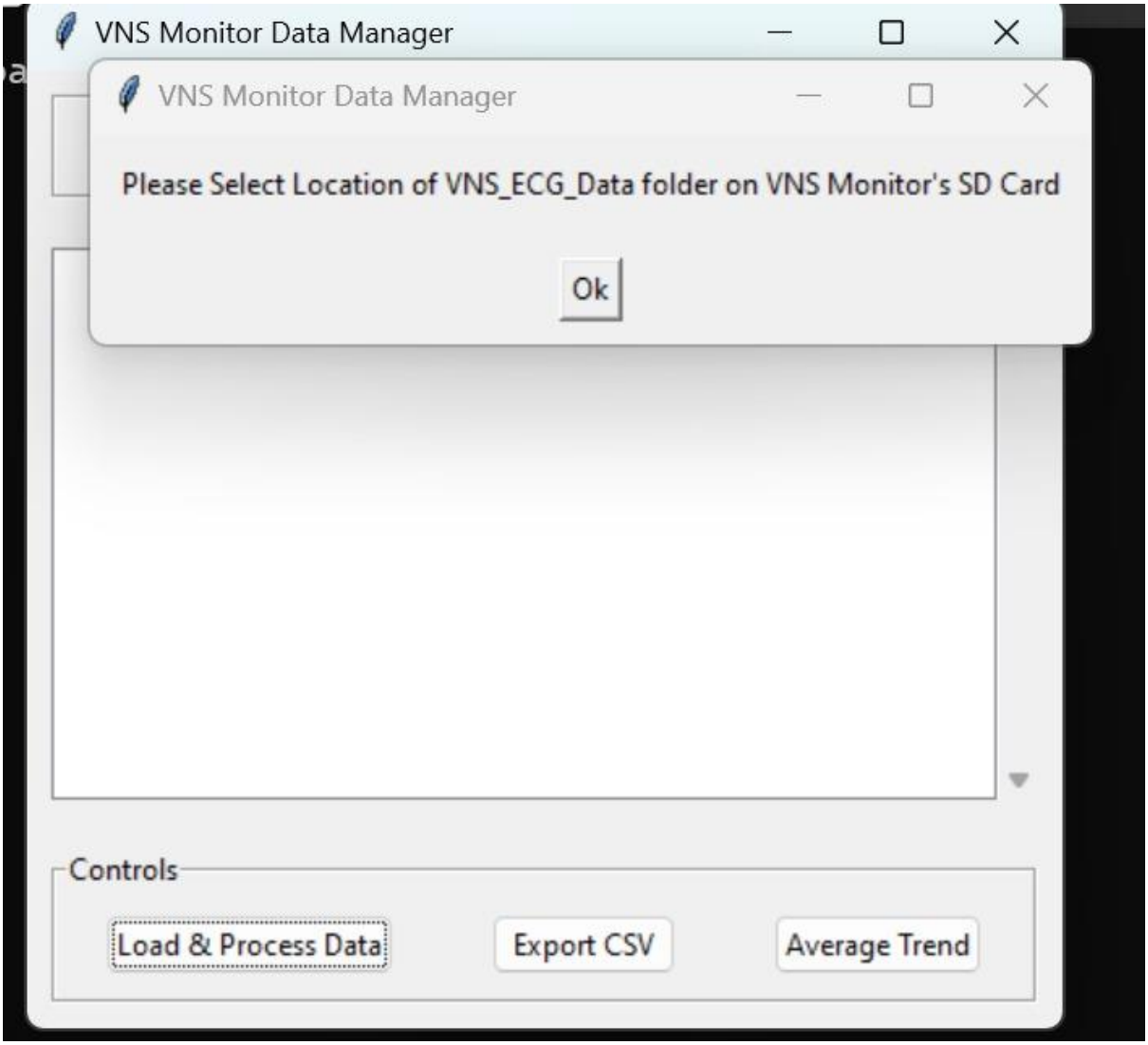
Step 7:

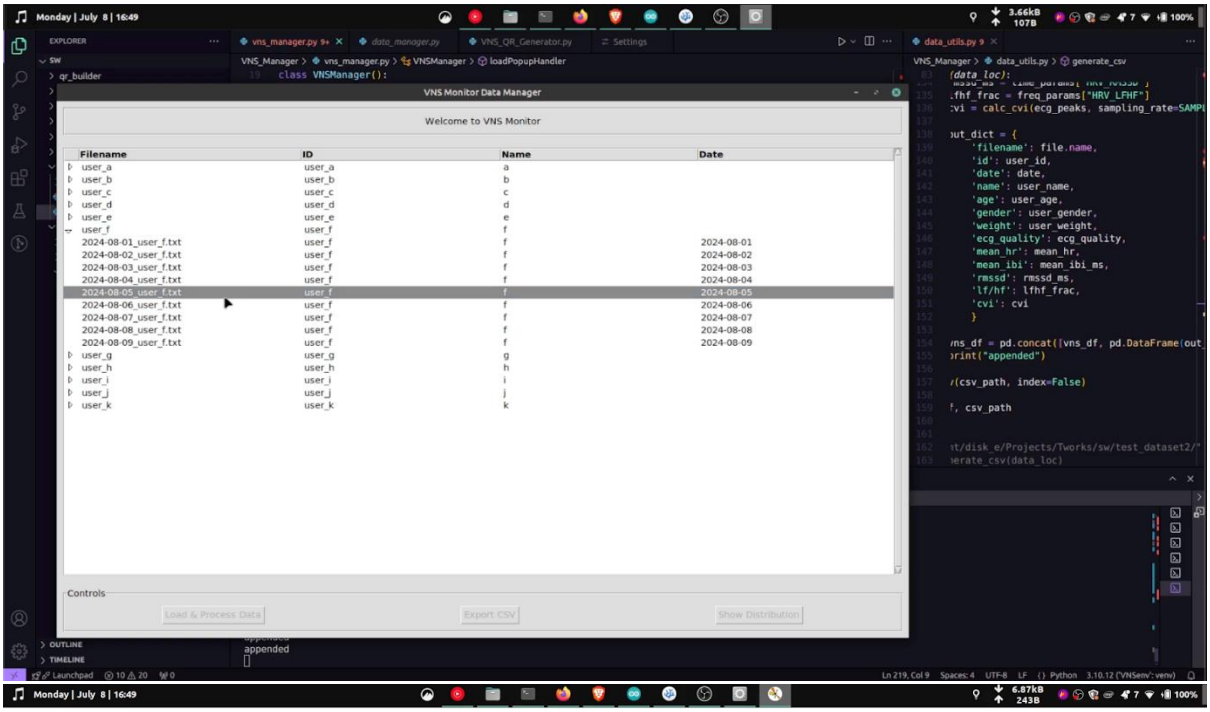
If you want to retake data for another patient, press any button to restart the device. If you wish to transfer the data, remove the SD card and insert it into your computer. Launch VNS manager software.



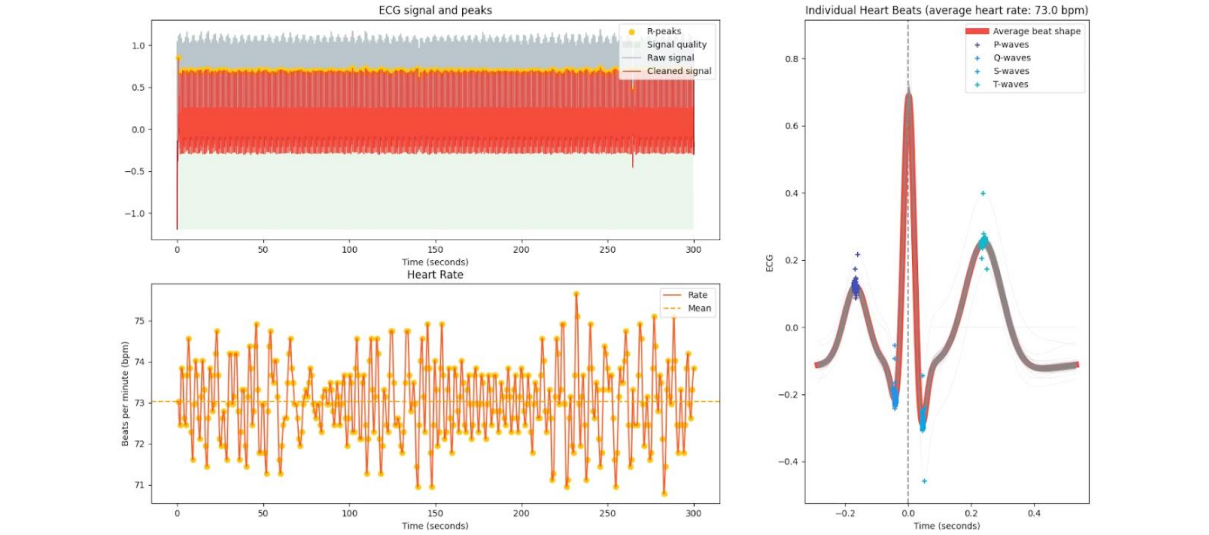
Click on LOAD AND PROCESS DATA.

Step 8:
Click on OK and select the file location of the SD card using File Explorer.

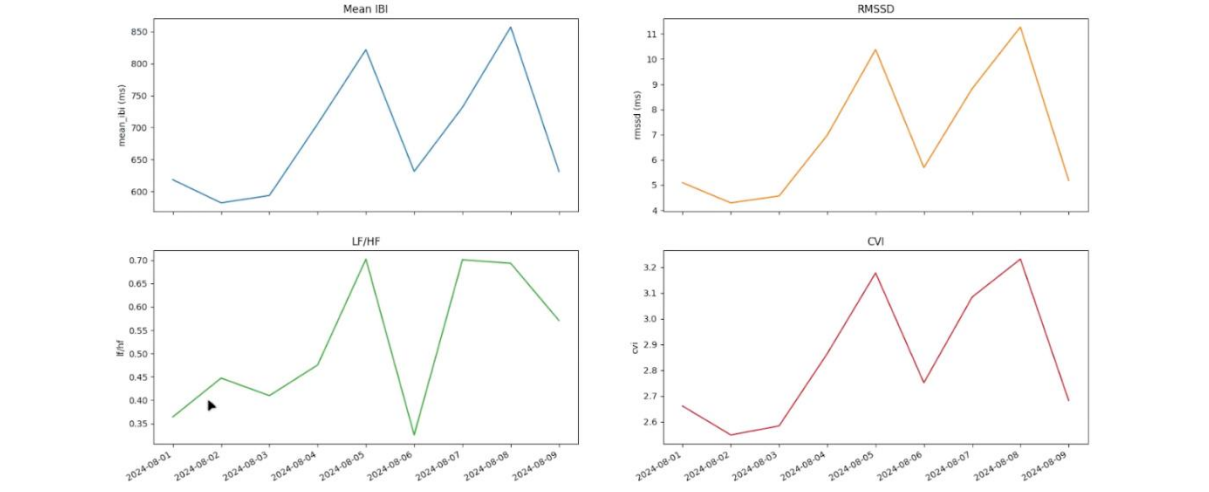

VNS Monitor Data Manager
VNS Monitor Data Manager
Please Select Location of VNS_ECG_Data folder on VNS Monitor's SD Card
Ok
Controls
Load & Process Data
Export CSV
Average Trend



Electrocardiogram (ECG)



Results for user_f over time



Name: f, Age: 22, Gender: F, Weight: 41

If you select a single file, you get a graph similar to the one shown below (1). This shows the data of an individual file and will give the extracted data. If you select a folder you will get the graph as shown below (2). This shows the data of a single patient over a long period of time by extracting data from multiple files.
The EXPORT CSV button exports data in a CSV format.

Appendix A

Generating a self-signed SSH certificate and private key involves using OpenSSL, a widely used tool for generating and managing keys and certificates. The process is similar across Windows, Linux, and macOS, with minor variations due to the operating system differences. Below are the detailed steps for each OS:

Windows

1. **Install OpenSSL:**
   - Download and install OpenSSL for Windows from Win32/Win64 OpenSSL.
   - During installation, select the option to copy OpenSSL DLL files to the Windows system directory.
2. **Open Command Prompt:**
   - Press Win + R, type cmd, and press Enter.
3. **Generate Private Key:**
   - openssl genrsa -out private_key.pem 2048
4. **Generate Certificate Signing Request (CSR):**
   - openssl req -new -key private_key.pem -out cert_request.csr
5. **Generate Self-Signed Certificate:**
   - openssl x509 -req -days 365 -in cert_request.csr -signkey private_key.pem -out certificate.pem
6. **Format the Key and Certificate:**
   - **The generated private_key.pem and certificate.pem files will already be in the required format.**

Linux and macOS

1. **Open Terminal:**
   - **On Linux, press Ctrl + Alt + T.**
   - **On macOS, press Cmd + Space, type Terminal, and press Enter.**
2. **Generate Private Key:**
   - openssl genrsa -out private_key.pem 2048
3. **Generate Certificate Signing Request (CSR):**
   - openssl req -new -key private_key.pem -out cert_request.csr
4. **Generate Self-Signed Certificate:**
   - openssl x509 -req -days 365 -in cert_request.csr -signkey private_key.pem -out certificate.pem
5. **Format the Key and Certificate:**
   - **The generated private_key.pem and certificate.pem files will already be in the required format.**