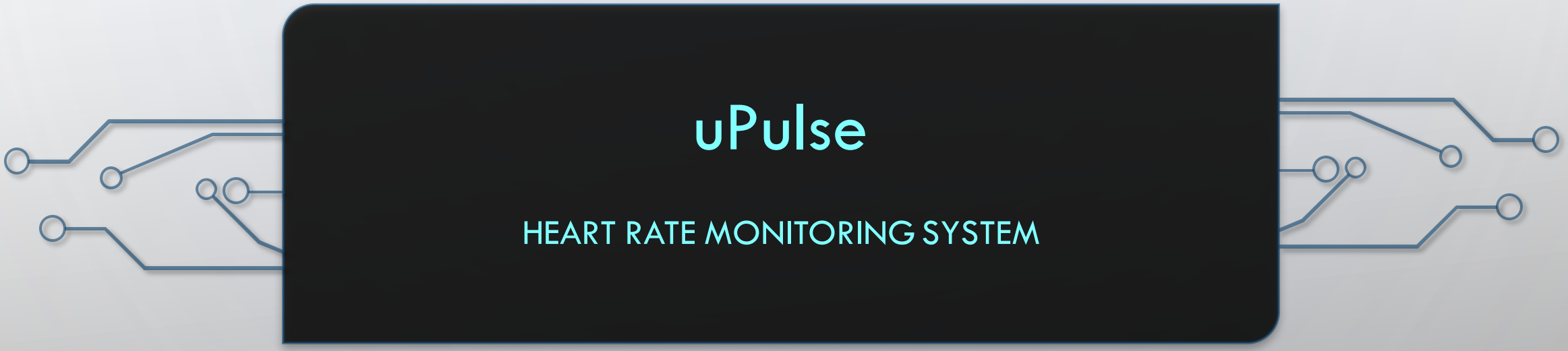

uPulse
HEART RATE MONITORING SYSTEM

Vision and Goals

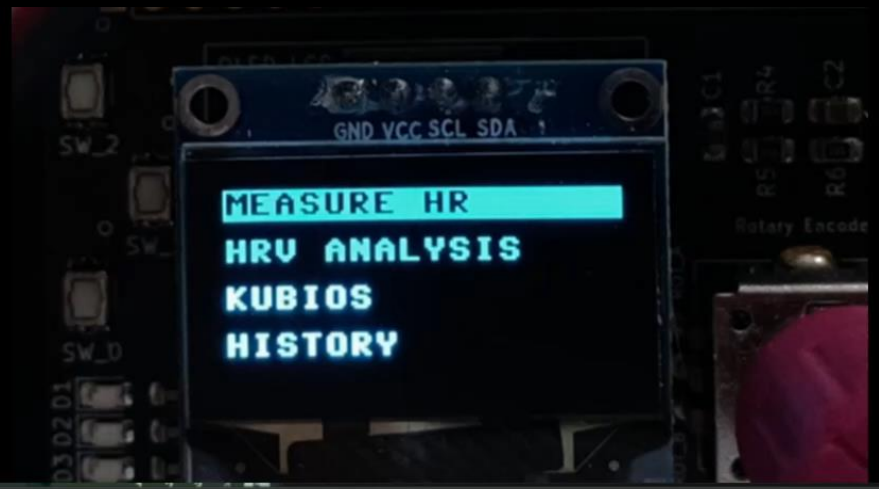
Our aim for this project was to create a heart rate monitoring device that would be useful and easy to use, that also has all the features named in the highest level (Grade 5) evaluation criteria (Backend running on Raspberry Pi, Kubios Cloud, History, OLED Menu ,animated PPG signal, basic HRV analysis, +-10% max difference when compared to pulse oximeter, start and stop options for everything, must be standalone system).

Our personal primary goal was to learn about the integration between hardware and software and putting our skills to use.

uPulse Usage and Features

- uPulse is a heart rate monitoring device that uses an optical sensor to provide real-time analysis of heart rate variability (HRV), which can be used to monitor cardiovascular health. uPulse can be used by anyone to get details about their health and help improve it.
- Key features:
  - **Optical Heart Rate Detection:** uPulse uses photoplethysmography (PPG) technology to precisely measure heartbeat data.
  - **HRV Analysis on Device:** uPulse calculates HRV measures like RMSSD and SDNN locally on the Pico to monitor stress levels and cardiovascular health.
  - **Kubios HRV Analysis:** Measured data is sent to the Kubios API for advanced HRV analysis, providing detailed insights into cardiovascular health and stress levels.
  - **Simple and easy UI:** uPulse has a simple OLED display interface with a rotary encoder for easy navigation through menu options.

DEMO




```
vladi — vladi@raspberrypi: ~/uPulse — ssh vladi@10.0.01.33 — 121×38
vladi@raspberrypi:~/uPulse $ python backend4.py
Connecting to broker: 10.0.01.33
Subscribing to topic: pico/output
Server is running...
```

```
>>> %Run -c $EDITOR_CONTENT

  MPY: soft reboot
```

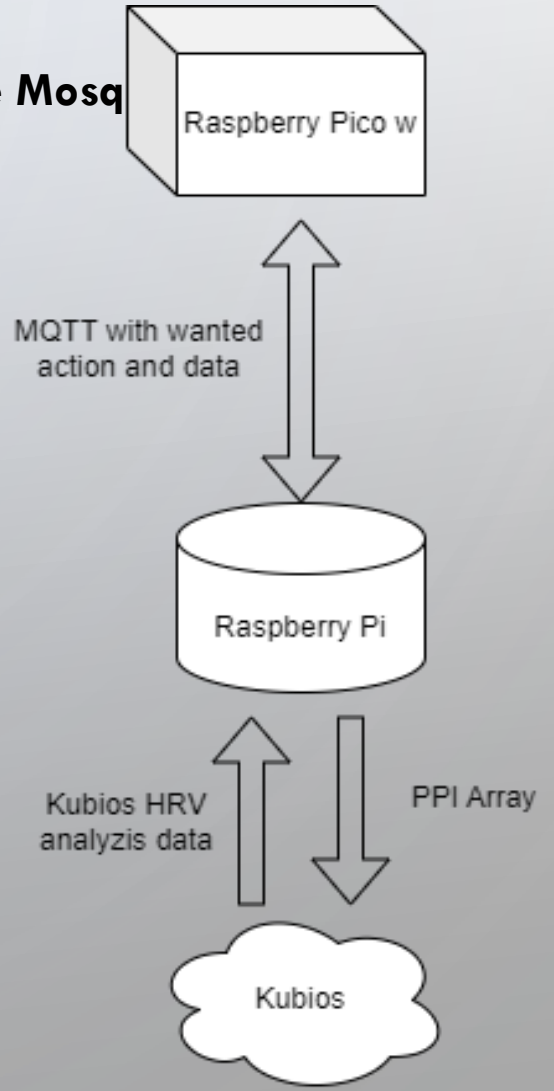
Frontend Code Structure

- Main
  - Main menu rendering.
  - Handles menu item selection, Monitor and History function calling.
- Error Handling
  - Analysis Error
  - Connection failed
  - Request timeout
- Monitor
  - Rendering, animating and processing logic for the HR, HRV analysis and Kubios selections.
  - Monitor function accepts two parameters, "type" and "duration"
    example Monitor("hr"), Monitor("hrv", 30) and Monitor("kubios", 30),
    type kubios takes up different logic path where it makes a "request" instead of calculating values.
- History
  - Menu of timestamps and History item rendering.
  - Timestamps, and History request handling.
- Calculate
  - Functions for calculating mean PPI, mean HR, SDNN etc.
- Functions
  - Functions that are used in different scenarios and different places.
  - Aka "Global utility functions"

Backend

uPulse's backend is running on a Raspberry Pi, which uses Python. The backend system's purpose is to send the data gathered by the heart rate sensor to the Kubios HRV analysis API and then store the results in JSON database. These results are shown in the history.

Data is transmitted between the Raspberry Pico w and the Raspberry Pi using the Mosq broker with these three calls:



"kubios": Contains a PPI(Pulse-to-Pulse Interval) Array for Kubios to analyze.
MQTT payload: { *"type": "**kubios**", "data": PPI_array* }
Returns Kubios results to frontend, so the user can see the results. Additionally, the results are stored in the history JSON file in Raspberry Pi.

"*timestamps*": Contains nothing except the call type.
MQTT payload: { *"type": "**timestamps**"*}
Returns list of all the timestamps of Kubios HRV analyzes to frontend.

"*history*": Contains the index of the selected timestamp.
MQTT payload: { *"type": "**history**", "data": current_selection -1* }
Returns the analyzed Kubios data of the given index to frontend.

What We Learned

- We learned valuable about teamwork and communication.
- We improved our knowledge about software, hardware and how they work together.
- We learned that the sensor cannot detect heartrate from finger-tip when pressing the finger against the sensor like a hydraulic press.
- We learned that it's possible to make this project from scratch in 3 days.
- ~~Learned that maybe it's not the best idea to implement the whole backend two days before project presentation.~~

```
1:  procedure triangular(in: S, out: t)
2:      t := λℓ.⊥
3:      let {a⃗_1, ... a⃗_s} = S
4:      for i := 1 to s do
5:          ℓ := leading(a⃗_i)
6:          while (ℓ > 0 ∧ ℓ ∈ dom(t))
7:              a⃗' := t(ℓ)
8:              p := power(π_ℓ(a⃗_i))
9:              p' := power(π_ℓ(a⃗'))
10:             if p ≥ p' then
11:                 a⃗_i := (π_ℓ(a⃗')/2^p')a⃗_i − 2^(p−p')
12:             else
13:                 t := t[ℓ ↦ a⃗_i]
14:                 a⃗_i := (π_ℓ(a⃗_i)/2^p)a⃗' − 2^(p'−p)a
15:             endif
16:             ℓ := leading(a⃗_i)
17:         endwhile
18:         if ℓ > 0 then t := t[ℓ ↦ a⃗_i]
19:     endfor
20: endprocedure
```

QUESTIONS?