Simple Fit

Daniel Leinonen, Duong Pham, Konsta Alajärvi,
Tiia Viitanen, Vincent Menke

December 2022

1 Description of the project

Monitoring body-measurements in a daily routine can be a huge deal to evaluate the body's physical health. During workouts or fitness routines it can help to reach higher goals or to push the body to its maximum potential.
Because of that we tried to develop an easy accessible fitness web application to keep track of these stats.

1.1 Goal

Our main goal was to create a web application for a browser (which supports Bluetooth Low Energy (BLE) connections eg. Chrome) that can display Polar Verity Sense sensor data and Polar H10 sensor data in real time. To achieve this we used the Web-Bluetooth API and React to create a seemingly smooth web page. From the end users point-of-view the final goal was to provide an uncomplicated application that is easy to understand and use. To assure that we took special care to evaluate complex data structures and display them in an appealing form.
The initial idea was, that end users of this application could vary from medical personnel, that could use the recorded data to shorten doctor appointments or to confirm short term body-measurements with long term recordings, to professional athletes and trainers, who want to push themselves or their trainee to the absolute limit while not risking injury. Even ordinary people could have an interest in this application to keep track of their body health.
Since this idea was very ambiguous we later scaled it down to a more manageable goal, focusing at especially athletes and trainers and ordinary people. This goal would still include the display of real time data in a user friendly way, which would be directly applicable while working out, but not the recording of it, making it difficult to find medical use-cases.

1.2 Project Partner

Partner of this project was Polar, a company that is mostly known for their extremely accurate Bluetooth body sensors and wearable technology. They specialize in fitness bracelets and smartwatches, and provide applications built around these devices.
They provided us with the necessary equipment (H10 and Verity Sense) and the initial idea for a web based application which supports these equipment. They also shown great interest in our project, motivating us to achieve set goals. Later in the project they would also help us in the development process.

1.3 The team

In this team we have Daniel Leinonen, Duong Pham, Konsta Alajärvi, Tiia Viitanen and Vincent Menke. Leinonen, Pham, Alajärvi and Viitanen are third

year Bachelor of Engineering, Information Technology students from Oulu University of Applied Sciences. Menke is a third year student in computer science with focus on media and communication from Darmstadt University of Applied Sciences (h_da). Since the team being international, communication was a delicate matter

At first we divided different tasks in data receiving and data process orientated tasks. Leinonen, Pham and Viitanen focused on the processing part, while Alajärvi and Menke focused more on the receiving part. Because this was not ideal and led to miscommunication, we later assigned tasks focusing on our individual strength.

While Viitanen took great care in creating different designs and layouts, that being her passion, Pham provided assistance and worked on the public availability of the project via the Google Cloud Platform. Alajärvi implemented many functional requirements, migrated the project to React and laid the foundation for the connectivity between the web-page and the sensors. Leinonen supported him on many occasions and prepared a database schema for future usage, see section 6.4. Menke was in charge of quality assurance and helped out in difficult parts of the programming. He did a major part in the data visualization and since his studies specializes in communication also was responsible for team management and communicating.

2 State of the Art

The idea to display sensor data is not innovative in any way. What makes this project somewhat unique is the usage of the Web-Bluetooth API, which is a fairly underused API to connect BLE-devices to web pages directly. This is currently only working with chromium based browsers. More information to the API in 4.1.
Since this project was done in cooperation with Polar, we also researched what products and applications they were providing. While doing so, we came to the conclusion, that Polar is not currently using this API, neither providing any web based application to monitor real time data.
Since this API is rather unknown we were also unable to find any comparable product on the market that provides this kind of service in a web based environment. In most cases the API was only used to read accelerometer data for rather small projects in the maker-community, often coupled with the usage of microcontrollers.
This lack of competition is a big deal, because it could provide Polar with some competitive advantages, should they decide to develop such product. This is especially true considering web applications being economically extremely beneficial and superior to standard mobile applications, because they are usually platform independent. This also makes a difference in maintenance. Secondly a web application can be shared easily by the publisher via QR-codes and are instantly available. This also correlates with another benefit: easy access. There is no need to download a whole app.
All these advantages could be important for the future development of Polar.

Change management principles	Service delivery principles
Start with what you do now	Focus on customer needs and expectations
Agree to pursue incremental, evolutionary change	Manage work, not workers
Encourage acts of leadership at all levels	Regularly review the network of services

Table 1: Kanban Principles [1]

3 Methodology

As our main development management technique we used Kanban as it is a commonly used technique even in commercial product development. To support this method visually we used Trello as a listing and management tool. Early in the project we also created a timeline to help manage scheduling and distributing tasks. We also used it to monitor how much time we spent on different tasks. Additionally we used standardized surveys and distributed them via Google Sheets to gather more insight of our users.

3.1 Kanban

Kanban is an agile development method to improve performance in software development. It requires the usage of lists to monitor tasks. These lists are visualized on a task-board, often containing at least three different lists: **To-Do**, **In Progress** and **Finished**. For this project we focused only on these three. Tasks can be claimed by any developer.
Furthermore Kanban determines three principles for management changes and three principles to ensure service delivery as you can see in table 1. While most of these are self-explanatory we especially focused on incremental change and customer needs. This is because in the beginning we had no specific vision of the product and not a clear understanding which requirements potential end users could have.
Instead of managing workers, we focused on managing work. This was done automatically due to freely assigning oneself to a specific task. This way every developer could claim tasks they felt proficient in.

3.1.1 Alternatives

Since this was an agile project we ignored methods like the traditional waterfall approaches and only considered two other, more scalable development methods; Scrum and Extreme programming.

1. **Scrum**
   We actively decided against the usage of Scrum since our Team-size was not big enough to support all the necessary roles required for this method,

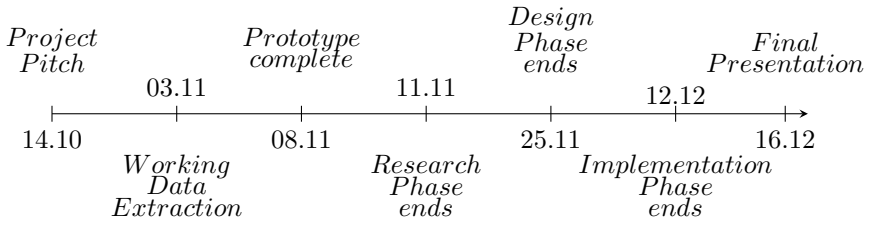

Figure 1: Project Timeline

although it would have structured our workflow and team management a lot better.

2. **Extreme Programming**
   Using extreme programming as our main development method was in our opinion also not the best choice since, like mentioned in section 3.1, we often had problems figuring out end user requirements and had to rewrite or even scrap existing functionalities. This issue would have made writing user-stories nearly impossible.

3.2 Timeline

In the project we used a timeline layout in Excel to monitor workloads. We later visualized the timeline with all different project phases and events like you see in figure 1. The timeline is essentially split into three phases, research, design and implementation, with two major events in the first phase.

3.2.1 Research phase

In this phase of the project we planned to research the new technologies that we had to work with. This phase was especially important in consideration of the Web-Bluetooth API, which none of us had experience working with. We also did a lot of research about the two different sensors we were using i.e. how we could access and read data collected with the sensors and what the received sample data looked like. Worth mentioning is that we split the team into two groups, one responsible for data extraction, the other one responsible for the visualization of this data. This would later become an issue due to miscommunication between the teams. To counteract that, we would later concentrate to not split us into groups but rather use the task-board to create groups dynamically.
The two important milestones in this phase were the working data extraction and a complete prototype which would help in later surveys to get a better understanding of the users needs. The first one of these was missed nearly completely with only a minor part of the data extraction working. Why this happened is explained in detail in section 5.2. For this reason all later milestones were prolonged by a few days, while the data extraction was developed parallel

with other tasks. With the basic data regrettably missing for the visualisation, stubs had to be implemented to visualize simulated data, delaying the surveys.

3.2.2 Design phase

In the design phase we firstly thought about potential technical designs. Because of that we decided to switch from plain HTML to React, which will be explained in section 4.2, and migrated the prototype accordingly. We also made advances on the appearance of our product and assigned design-exclusive roles, who created first design prototypes, which defined the final product appearance. Lastly we finished the creation of the surveys and began distributing them among developers and professional athletes. More in section 3.3. Because of the late distribution some functional requirements were still a bit unclear, which then took additional work in the next phase.

3.2.3 Implementation phase

During the implementation phase our original focus was only to implement all necessities and finishing up the product. Due to the delays in both previous phases some functional requirements like the ECG visualisation were still not even designed, see section 5.3. Because of that most of the work in this phase went into solving these issues, since most of the other requirements were fortunately mostly implemented already.
Lastly in this phase we made our product publicly available via the Google Could Platform, which will be discussed in section 4.4.

3.3 Surveys

For this project, we decided to do three different surveys. One inquiring medical professionals, second focused on athletes and third targeting developers. The reason for the surveys was to understand end user requirements and get their opinion about the project. In any project it is good to ask possible customer groups, what they think and what kind of application they would prefer use.
The distribution was relatively efficient due to the possibility of sharing them online. There were 8 answers from developers and 29 answers from athletes. Unfortunately the participation for the medical survey was too low to get a sufficient amount of information. While the amount of answers for both of the other surveys are not enough to make any definitive conclusion, they did give us some insight.
We came to the conclusion that most of the athletes (78,3%) preferred mobile applications over web applications. This indicated that even though web applications are easier to access, people still prefer mobile applications because most of the time they are smoother to use and provide a better user experience. On the other hand the information from the developers were very mixed. Some still preferred a mobile application for this use-case or recommended new independent frameworks like flutter, but nearly all saw the economical advantages of

web applications.

4 Software design

Software design is a very important part of software development, since mistakes here can lead to unnecessary work. More about that in section 7.2.
Due to the lack of alternatives we had to choose the Web-Bluetooth API for data transfer and paired it with React to create a smooth and responsive web application. For accessibility we chose to launch the application on Google Cloud Platform.

4.1 Web-Bluetooth API

In our project we used Web-Bluetooth API with Javascript, to transfer the sensor data from the Bluetooth devices to our web application. Web-Bluetooth API (Application Programming Interface) is a tool that allows web-browsers to communicate with devices through Bluetooth Low Energy personal area network over JavaScript. This is still an experimental technology and is not supported by all browsers, most of the features are supported on Google Chrome, Microsoft Edge and Opera. Web-Bluetooth API uses a protocol called GATT (Generic Attribute Profile) which defines the way that two Bluetooth Low Energy devices transfer data between each other using concepts called Services and Characteristics. Services can be imagined as a collection of a more specific data packets called Characteristics, the idea is that Characteristics containing similar kind of data are packed into the same Service. Characteristics are the lowest level concept of GATT, which encapsulate a single data point in time, so they are the main point of focus when interacting with BLE devices. Services and Characteristics both distinguish themselves with pre-defined 16-bit or 128-bit UUID (Universally Unique Identifier).

4.1.1 Alternatives

Currently there are not any other viable solutions to transfer data between web-browsers and BLE devices, at least with JavaScript.

4.2 React

React is a JavaScript library designed for building dynamic user interfaces. The main reasons why we decided to develop our application with React is, that it is a well known web application development tool in the IT industry and was the first option that came to our minds. We knew that with React it would be possible to make our applications user interface dynamic and highly responsive. in addition to that some members of our team had good previous experiences with React.

4.2.1 Advantages

React is known mostly for its speed, efficiency, flexibility and performance. The speed of React comes from many features, one of these is called virtual DOM (Document Object Model), which allows the server to update the UI of the application in small parts. The idea with React is that the UI is dissected into smaller entities called components. These components have their own logic, they can be re-used and update their own states. This adds efficiency when developing complex applications.

4.2.2 Alternatives

There are alternatives for React for example Angular JS, jQuery or Solid.js. These all could have been perfectly viable tools to use but none of our team members had any experience with these, so React was an easy choice.

4.3 Software architecture

In figure 2 you can see a diagram which represents our applications class hierarchy. There you can see index.js and App.test.js being the root components. While App.test.js is only for testing purposes, index.js renders the App.js component, which conditionally renders Connect.js, H10.js and VeritySense.js. These are our three main pages, representing the landing page, a specific page for the H10 sensor and one for the VeritySense sensor. Switching between these pages is done via routing to a different subdomain.
The three main components, while providing their own structure, still use other components, to better group the code and to avoid redundancy. This is beautifully done with the clock.js component, since on every pages there is a clock displayed somewhere. Isolating the clock functionality as a component made that code easily reusable.
Since all components need their own .css file, these are listed under modules for easy access. The two components in green, listed in the source folder are auto-generated React files.

4.4 Google Cloud Platform

Google Cloud Platform is a cloud hosting system by Google. We choose GCP to host our application because of 5 main advantages:

1. Easy deployment: We can spin up new instances or retire them in seconds, allowing us to accelerate development with quick deployments. Cloud computing supports new innovations by making it easy to test new ideas and design new applications without hardware limitations or slow procurement processes.

2. Scalability and flexibility: Cloud computing gives businesses more flexibility. The application can quickly scale resources and storage up to meet business demands without having to invest in physical infrastructure.

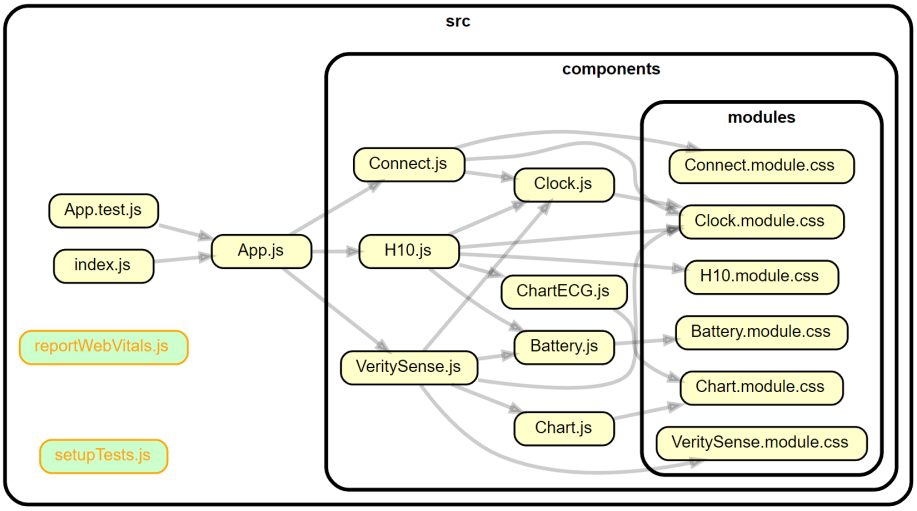

Figure 2: Class diagram

3. Data loss prevention: Cloud providers offer backup and disaster recovery features. Storing data in the cloud rather than locally can help prevent data loss in the event of an emergency, such as hardware malfunction, malicious threats, or even simple user error.

4. Cost saving: We do not have to pay for this application

5. Advanced security: Despite popular perceptions, cloud computing can actually strengthen your security posture because of the depth and breadth of security features, automatic maintenance, and centralized management. Reputable cloud providers also hire top security experts and employ the most advanced solutions, providing more robust protection.

4.4.1 Alternatives

Amazon Web Service and Microsoft Azure, the two main competitors, were not our choice mainly because their free trails are limited. With Google Cloud our application can run all time with zero cost. Furthermore we had some prior experience with Google Cloud, giving it an additional advantage.

5 Implementation

In this section we will talk about the final product. We will also take a look at important or especially difficult parts of the code.

5.1 Setting up the connection

First thing to do when starting our SimpleFit application is to find and connect to a Bluetooth device. This process starts with using the "requestDevice" method of the "bluetooth" interface provided by Web-Bluetooth API and iniatilizing "options" variable tells the server what kind of bluetooth devices and services we are looking for, this is explained in more detail further down below.

5.1.1 Options

In figure 3 we are looking into the "options" variable which we already briefly mentioned, this is required parameter for the "requestDevice" method and it is used to filter the devices we want to see and what kind of devices the application is searching for,it also has to contain all the service uuids referring to the services that we want to use after the connection is established. We also declared a company identifier in our options variable with a uuid referring to devices manufactured by Polar so only Polar devices are going to be visible with our application.

5.1.2 OnClick

In figure 4 we can see the what is happening when the onClickEvent triggers when clicking the "connect" button in our application. First the program requests devices from "bluetooth" interface with "options" parameter which has to be previously initialized, this returns a Promise to a BluetoothDevice object with the specified options. After this the program uses the "connect" method of the "gatt" interface referring to BluetoothRemoteGATTServer which tries to establish a connection between the two devices. When successful this again returns a Promise with the actual GATT server of the device, this is saved to state variable in this part of the program and is later used to access all the services and characteristics of the device.

5.2 Getting the sample-data

Like mentioned in section 3.2 we had a hard time getting the data extraction fully working. This was mainly due to scarce example code in the internet and to an insufficient documentation for the sensors.
The first major difficulty was reading the received data from the Polar custom service, which contained more complex data like ECG or Accelerometer-data. For this issue there were no examples in JavaScript at all and the ones in the polar-github were in written in Java and felt not applicable.
What we eventually settled on was an extra meeting with developers from polar

```
1
2     \centering
3     \begin{lstlisting}
4   let options = {
5       filters: [
6         {
7           // Filtering devices with company indentifier, showing only
                devices made by Polar
8           manufacturerData: [{ companyIdentifier: 0x006b }]
9         },
10        {
11          services: ["heart_rate"]
12        }
13      ],
14      acceptAllDevices: false,
15      optionalServices: [
16        "0000180a-0000-1000-8000-00805f9b34fb",
17        "0000180f-0000-1000-8000-00805f9b34fb",
18        "fb005c80-02e7-f387-1cad-8acd2d8df0c8"
19      ]
20    }
```

Figure 3: Options

```
1           const onClickEvent = () => {
2     navigator.bluetooth.requestDevice(options)
3       .then(device => device.gatt.connect())
4       .then(server => {
5         console.log(server);
6         props.checkConnection(true)
7         connectionEstablished(server.device.name);
8         props.func(server.device);
9         //return server.getPrimaryServices();
10      })
11      .catch(error => {
12            console.log('Argh! ' + error);
13      });
14  }
```

Figure 4: On click connect

responsible for the development of the polar SDK. This, while not solving the issue, at least gave us a hint on how to communicate with the sensors and how to process the data. With this new information we eventually wrote the very compact code in figure 5. This part of the code is split into two parts. The first, line 7 ff., is responsible for starting complex data-streams like ECG and can easily be expanded to setup additional data-streams as you can see in the commented part from line 16 till 18. Important to mention is, that starting the data stream is requested from the PMD-Controllpoint characteristic (in the code called Cntrl_char) but needs to be read out from the PMD-Data characteristic (data_char). This is the big difference to the second part of the code, line 23 ff., where simpler data is requested.
figure 6 shows the event-handler responsible for handling ECG data that is being received. The loop at line 8 begins at the tenth position of the received data array, because this is where the sample data begins. From here on out, every sample consists of three bytes in little endian notation. Since JavaScript does not provide any built-in functionality to convert data into 24 bit sized integer with little endianess, we created our own function for that in figure 7. Here we make sure that every one digit byte has a zero in front of it to represent correct hexadecimal notation, then concatenate every byte in order and parse them to an integer via the built-in parseInt()-Method. The last if-statement at line 14 is to support signed integer, since the ECG-data can also contain negative voltages. With all this, the data is processed correctly and ready to be visualized.

5.3 ECG-Visualization

This was probably the most complex part of the code, since JavaScript is quite unique with its asynchronous behaviour. To create a real time chart in JavaScript we decided on using the react-chartjs-2 library, because it felt like the easiest to understand.
In figure 8 you can see the part of the code where the sample-data is plotted into the chart. Unfortunately we were not able to create a perfectly stable implementation, so we decided on a workaround. ecg_array represents a FIFO-queue for all incoming ECG-data. In line 19 ff. you can see that every time this code is called, the value in ecg_array at the current position is being plotted as y value, with x as the current time. After that, the current position increases by one. Line 26 and 27 are setting the delay and refresh-rate to 0 so the function is called as often as possible. This opens up the probability for an out of range error, but due to the enormous amount of data the sensor is sending, and the chart being slower than the data is being received the real issue is that the chart is getting out of sync with the current values. With line 15 ff. we tried to counteract this issue, but this only "hard resets" the queue to a size of zero and starts this problematic cycle anew. A real fix is proposed in 6.2.

```
// Starts the data streams from polar device by first getting the "device" from props which is passed down from App.js
// then searching for the needed services from "device.gatt" meaning the device server
const startMeasurement = () => {
  props.device.gatt.getPrimaryServices()
  .then(services => {
    services.forEach(element => {
      if (element.uuid === PMD_Service) {
        element.getCharacteristic(Data_char).then(dataChar => {
          dataChar.startNotifications();
          dataChar.addEventListener("characteristicvaluechanged", handlePmdDataValueChanged);
        }).then(_ => {
          element.getCharacteristic(Cntrl_char)
          .then(controlChar => {
            console.log(controlChar.properties);
            controlChar.writeValueWithResponse(ECG_Array)
             /*.then(_ => {
              console.log(controlChar.properties);
              controlChar.writeValueWithResponse(ACC_Array);
            }); */
          })
        })

      } if (element.uuid === Heart_rate_Service) {
          element.getCharacteristic(Heart_rate_Char)
          .then(heartRateChar => {
            console.log(heartRateChar);
            heartRateChar.startNotifications();
            heartRateChar.addEventListener("characteristicvaluechanged", handleHRValueChanged);
          })
      }
      [...]
    })
  })
}
```

Figure 5: Getting the data

```
// Function for handeling the PMD-Data-Value change event
// checks which type of dataframe is recives (eg. ECG or ACC)
const handlePmdDataValueChanged = (event) => {
  // 0 means ECG
  if (event.target.value.getUint8(0) === 0) {
    let sample;
    let sample_array = new Array;
    for(let i = 10; i < event.target.value.byteLength; i=i+3){
      sample = parseToInt24(event.target.value.buffer.slice(i, i+3));
      sample_array.push(sample);
    }
    // Sent all samples of one dataframe to the chart
    setEcg(sample_array);
  }
  [...]
}
```

Figure 6: Reading the data

```
// Parsing the first 3 bytes of the argument array into an
    integervalue with little endianess
// used for ECG handling
function parseToInt24(byte_array){
  let m_array = new Uint8Array(byte_array)
  let value0 = m_array[0].toString(16)
  if (value0.length == 1) { value0 = 0 + value0 }
  let value1 = m_array[1].toString(16)
  if (value1.length == 1) { value1 = 0 + value1 }
  let value2 = m_array[2].toString(16)
  if (value2.length == 1) { value2 = 0 + value2 }
  let value = value2 + value1 + value0;
  value = parseInt(value, 16);

  if (value > 8388607) { value = value - 16777216 };
  return value;
}
```

Figure 7: Little endianness conversion

```
//Handle plotting via react-chartjs
 const options = {
    scales: {
      x: {
        type: "realtime",
        realtime: {
          onRefresh: function() {
            if (props.data === undefined) {
                data.current.datasets[0].data.push({
                  x: Date.now(),
                  y: 0,
                });
            }
            else {
              // If Dataarray gets to long, reset everything
              if(ecg.length > 50000){
                ecg_array.length = 0;
                i = 0;
              }
              data.current.datasets[0].data.push({
                x: Date.now(),
                y: ecg_array.at(i),
              });
              i++;
            }
          },
          delay: 0,
          refresh: 0,
        },
      },
    }
  }
```

Figure 8: Creating an ECG-Chart

Figure 9: Landing page

5.4 The Final Product

In figure 9 you can see the landing page of our project. From here the user can connect to a bluetooth device filtered by the options explained in 5.1.1. From here they can be rerouted accordingly, either to the H10 page, figure 10, or to the VeritySense page, figure 11. Both being very similar, displaying current beats per minute (BPM), their respective highest and lowest values, grouped on the left side for a better readability. Two buttons, one to change the color-theme and the other one to disconnect the device and reroute to the landing page are grouped on the right side as some sort of user interaction panel. The battery-level is also displayed here, since informing the user when to charge the device, which is also an interaction with the product. The only difference between these two pages are the diagrams. The VeritySense page is displaying the chronological sequence of the beats per minutes, while the H10 page displays the way more complex electrocardiogram (ECG).
For all three pages we went for a color pattern of mostly red, white and black, giving the application a dynamic and athletic look, further targeting sport enthusiasts.

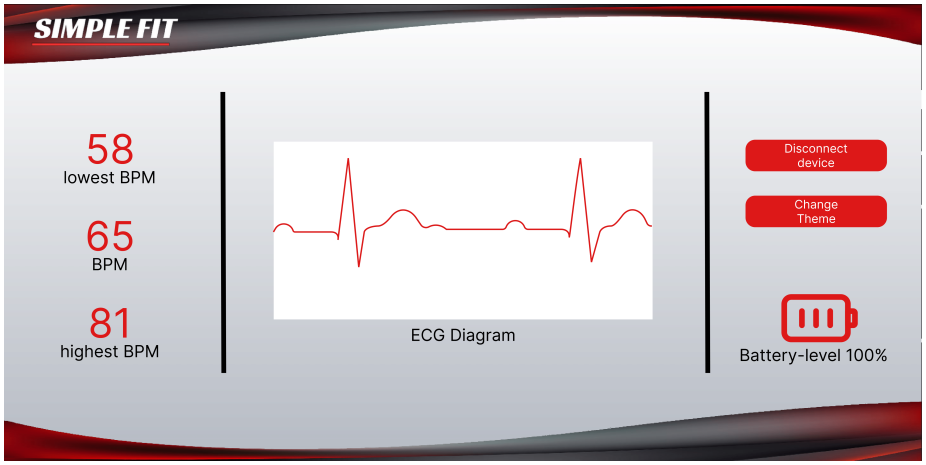

Figure 10: H10 page (concept art)

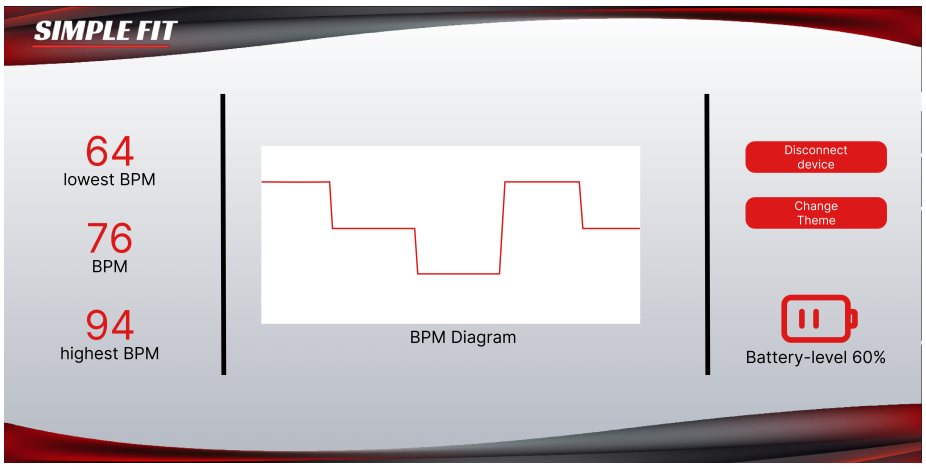

Figure 11: Verity Sense page (concept art)

6 Future Work

There are few things that we would have liked to implement, it there had been more time for development in the project.

6.1 DRY methodology implementation

DRY (Don't Repeat Yourself) methodology in software development is a principle that revolves around avoiding redundant code and repetition. When the DRY principle is well executed and implemented, modifying elements in an application does not require additional changes to be made elsewhere in the code, thus making it comprehensive and logically predictable. This in turn makes the maintenance and development of an application easier.
The DRY principle was a method that was utilized in our project albeit only later when the project had already advanced from the initial stages of design to implementation. This led to lots of repetition in the code, as well as creating redundant elements which could have been avoided had we had the DRY principle as our framework for the project. The creation of unnecessary elements or logic within them was noticed but pruning the code after everything was already somewhat successfully implemented turned out to be more time consuming than expected.
Dismissing the DRY principle in the early stages of the project was a lesson learned, unfortunately the hard way. Had the principle been well executed in our project, we would have saved time as well as made our code smarter, more efficient, and easier to maintain. Our unfortunate struggles trying to implement this methodology when it was already long overdue is a prime example why a well-designed and carefully premeditated blueprint, framework, for a project is essential and should be a major concern for any forthcoming projects we participate in.

6.2 Fixing the ECG plotting

The major issue with the ECG plotting is that the displayed data is out of sync with the currently recorded data. To fix that we would need to use the first ten bytes which are skipped in figure 6. These bytes contain a timestamp of the last ECG-packet sent in nanoseconds. Using this timestamp and the sample rate, which is 130 Hz for the ECG sample, it would be possible to calculate the exact amount of time between the first sent ECG-packet and the following sample.
Using Date.Now() the moment the first ECG-packet arrives and then adding the calculated elapsed time to the initial timestamp every sample could be placed at the correlating position.
This unfortunately does not fix the issue of the browser not being able to keep up with the vast amount of data. Luckily this can be dealt with by implementing a delta value which has to be lesser than the difference between sample A and its directly following sample B. Only if this applies sample B will be plotted.

6.3 Adding full accelorometer support

Basic functionalities for accelorometer are already implemented in the current version of the code. The data can be requested and is processed mostly correctly. The issue here is the seemingly random changes of the received data array size, correlating in data conversion errors. Another problem is the direction that the sensor is facing, because this changes the label of the axis.
Lastly the implementation for speed calculation is missing, but this can be done using the integral of different samples.

6.4 Database connection and persistent memory

Software that is targeted at the public is heavily dependent on insight gathered from the users. Therefore, databases are at the very heart of software development capabilities. An ably structured database brings order to the otherwise chaotic nature of data collection. An overwhelming amount of data is required to continue developing an application to meet the needs of customers. Instead of thinking of this as a burden a brief glance at the immense array of benefits that a database offers are enough to justify the work it requires to build a database, especially for projects aimed at a larger mass of users. Originally our plan was to create a functional database in conjunction with our web application to gather valuable data from our users. A database was in fact created with the necessary capabilities to collect and store all the data we had planned to gather. Unfortunately, this component of our project had to be scrapped due to time constraints. We do recognize the power and the advantages a database would have brought to our project. The data that users of our application would provide would allow us to analyze our user base and make necessary changes and further develop it to meet the needs of our clientele and reach out to a greater number of people. This would also serve as a terrific tool to even be a step ahead of users' needs.,

6.5 Automatic GCP deployment

At the moment, the deployment process on the cloud is not automatic. With every new version of the web page we will have to build and re-deploy it again on Google Cloud which makes the development of the application more time consuming and very hard to manage. In the future, an automation process will be implemented so for each time Github link updates, Google Cloud will compile, build and run the latest version of the application.

7 Lesson Learned

In any project it's crucial to look back and think what worked and what could have been done in a different way. Our three biggest struggles are explained here.

7.1 Project management

When looking back at this project, the first thing that should have been done differently was the choice of our management tools. As mentioned earlier, we used Trello as a list-making tool. This was because most of the team had used it in previous projects. Trello is quick to learn and rather easy to use, but there are many other project management tools that can do the same, while being more precise in logging. Instead of Trello, we would have liked to use Jira. Jira has a harder learning curve, but a better visual feedback, automatically creating status diagrams project and perhaps using it would have been more motivating than Trello.

7.2 Software design

Another lesson we had to learn was how we handled the software design. Designing proper software is a not an easy task and takes some experience. Every design choice that is made should be made with caution, and not depending on what feels convenient right now. The software we created, with multiple pages to navigate around is not incorrect, but a single page application would have been the cleaner and more professional way to do it. This would have made it far easier to utilize the DRY practice and would have made the code more maintainable.
Also the user could have benefited by a singe page app, trough a smoother navigation or fancy loading screens increasing the user-experience.
We recognized this issue in the last phase of the project, but were not able to fix it, since early mistakes like undesirable design choices take a lot of work, manpower or time to fix. Resources we did not had in the implementation phase due to the much more pressing matter, the missing implementation of functional necessities.

7.3 Overall issue

Although the previously mentioned mistakes are unfortunate, they are definitely fixable with more time or experience. What can not be fixed is the overall consent from end users who prefer mobile apps instead of web applications. Like we mentioned in 3.3 our surveys are not sufficient to come to any tangible conclusion, we still got the hint that such application is not desired, since nowadays mobile apps feel way more user friendly and consistent.
This consent is supported by our results developing this project for two month. The browser is just not capable to work with such amount of data in a short

time, since there are many other task running in the background. This made the application feel unresponsive sometimes, especially in consideration of the charts.
Our results are also supported by the unpleasant developing-experience we had while working with the Web-Bluetooth API and JavaScript. The API is definitely an interesting technology, it just felt very non-intuitive to use. JavaScript just does not feel designed to handle such low level data streams. In addition to that, asynchronous functions in JavaScript are often very unpredictable and not easy to work with.
Because of all these issues we came to the conclusion that we would not recommend Polar to further invest in projects regarding real time display of sample data in browser environment, since the demand is too niche. It was definitely an extremely interesting project, we had a lot of fun working and learning about so many new technologies, but we think there is evidence advising against such an investment. And in the end we can not confirm that we developed an application that is as user friendly as a mobile app, while being economically superior, since these major drawbacks make the development expense higher than expected.

References

[1] Sellgren, A. The official guide to the kanban method, Dec. 2022.