



Institutt for Informasjonsteknologi

Postadresse: Postboks 4 St. Olavs plass, 0130 Oslo

Besøksadresse: Holbergs plass, Oslo

PROSJEKT NR.

2022-64

TILGJENGELIGHET

Public report, public source-code.

Telefon: 22 45 32 00

BACHELORPROSJEKT

HOVEDPROSJEKTETS TITTEL Inquiry, Documentation and Evaluation of Activity-Wristbands for DIGI-EL, Digital Applications for handling early labour	DATO 25.04.2022
	ANTALL SIDER / BILAG 98
PROSJEKTDeltakere Hassan M Sheikh Mohamed Abdullahi Nojus Budreika Rosa Berg	INTERN VEILEDER Kristian Wold
OPPDRAGSGIVER Karlstad University	KONTAKTPERSON Lothar Fritsch

SAMMENDRAG

In collaboration with Karlstad Universities' DIGI-EL project, our group has evaluated activity-wristband Fitbit Charge 4. Assessments for accuracy, safety and privacy were performed. Additionally, as a part of the project, the team has developed a software solution to view and interact with the data collected throughout the project.

3 STIKKORD

Early Labour

Fitness-tracker

React

Abstract

Since digital tools have become synonymous with productivity and convenience, questions have risen on whether these tools can assist people in more intimate situations. DIGI-EL at Karlstad University wants to find out if activity-wristbands can be used to reduce the stress associated with childbirth. To investigate this hypothesis DIGI-EL wants to know that the tools they use are safe, accurate and legal. In this report we evaluate the Fitbit Charge 4 according to criteria set by Karlstad University. We measure the accuracy of the device's functionalities and we investigate its compliance with GDPR privacy laws and data-safety laws in the context of sensitive research. We find the properties of the wristband unsatisfactory for research purposes but good enough for a slight increase in quality of life. We also provide a react-application that demonstrates how data can be extracted from the Fitbit API.

Table of Contents

Main Title	1
Abstract	3
Table of Contents	4
Glossary	7
1. Introduction	8
1.1 Preface	8
1.2 Project Background	8
1.3 Research and Project-Drafting	9
1.4 Report Structure and Chapter Overview	11
2. Theoretical framework	12
2.1 React.js	12
2.2 Fitbit Web API	13
2.2.1 Background	13
2.2.2 Prerequisites	13
2.2.3 Authorization	14
2.2.4 API endpoints	15
2.2.5 Making requests	15
2.2.6 Scopes	17
2.2.7 Intraday	19
2.3 Data export	20
2.3.1 Preface	20
2.3.2 Web API	20
2.3.3 Fitbit official desktop app	20
2.3.4 Third-party applications	22
3. Product development	24
3.1 Aims	24
3.2 Work methodology	25
3.3 Development phases	26
3.3.1 Research phase	26
3.3.2 Design phase	27

3.3.3 Implementation phase	30
3.3.4 Limitations	32
4. Product Specification	33
4.1 Initial requirements	33
4.3 Changes in the requirements	34
5: Product Documentation	36
5.1 Preface	36
5.2 Third-party libraries	36
5.3 Running the application	37
5.3 React concepts	37
5.3.1 Components & Props	37
5.3.1 Hooks	39
5.4 Authorization	41
5.5 Context	42
5.6 Fetching data from the API	43
5.7 Handling data	47
5.8 Dashboard components	48
5.9 Selecting date period	51
6: User Manual	54
6.1 Prerequisites	54
6.2 Logging in to the system	55
6.3 User-interface overview	56
6.4 Add to home screen	58
7. Fitbit, GDPR and Privacy	60
7.1 Background	60
7.2 The implications	60
7.3 A look at the GDPR	61
7.4 Fitbit privacy policy and Terms of use	62
7.5 Data	63
7.6 Discussion	65
8. Experiments and Methodology	66
8.1 Technical specifications	66
8.2 Heart-rate Monitor	67
8.3 Sleep	68

8.4 Water resistance	69
8.5 Results	69
8.5.1 Heart-Rate: Resting	71
8.5.2 Heart-Rate: Stairs	72
8.5.3 Heart-Rate: Walking	74
8.5.4 Heart-Rate: Discussion	77
8.5.5 Sleep: Findings and Discussion	77
8.5.6 Water Resistance: Findings and Discussion	78
8.5.7 Battery life: Findings and Discussion	79
8.5.8 Wristband fit & comfort: Findings and Discussion	80
9. Conclusion	81
9.1 Future Work	82
Appendices	84
A) Excess Experiment Data	84
B) Notes from the sleep experiments	95
C) Project Timeline	97
D) Project Repository	97
Sources	98

Glossary

API-call - A request for information sent to a server by an application.

Database - A computer-system built around storing, categorizing and accessing data usually through a database-management system such as mySQL or MongoDB.

Front-end - Part of the application that the user sees and interacts with.

HTTP - Hypertext Transfer Protocol is a protocol for sending resources over the World Wide Web.

Endpoint - An “API endpoint” defines a path to the resource that the developer wants to access. They’re appended onto the URL to point to the specific subcategory of data the developer wants.

URI - Uniform Resource Identifier is a String of characters that refer to the resource on the internet. Uniform Resource Locator (URL) is a subset of URI that describes how to reach the resource (Javatpoint, n.d.).

Token - A unique “key” composed of random letters and symbols that is exchanged between systems for verification and identification.

JSON - JavaScript Object Notation is a lightweight data-interchange format (Json.org, n.d.).

HTML - HyperText Markup Language is language to build structure for web documents that are displayed on the internet.

CSS - Cascading Style Sheets is a language that is used to style the appearance of HTML elements.

MIT license - A software license that permits commercial use, distribution, modification and private use (Choose a License, n.d.).

BPM - The pulse is measured in beats per minute, which is the number of beats in a minute.

1. Introduction

1.1 Preface

In this paper we will describe the process of testing and assessing the Fitbit Charge 4. Familiarity with smart-devices is assumed for our assessment of the Fitbit. However, the section where we describe our react-application might prove challenging to someone without experience with modern web-development. We will try to use clear language and layman terms to make this paper accessible for readers without a background in development. If this cannot be done without ruining a paragraph we will include the term or concept in the appendix where it will be fully explained.

1.2 Project Background

Our team was introduced to the project through an administrator at Oslomet. Karlstad University needed a follow-up project to their pilot study in 2020 and we were added to the roster one by one until we had our full team.

Research shows that over 75% of pregnant women make use of digital aid in connection with their pregnancy (Lupton, D. and Pedersen, S., 2016), yet there have been very few investigations into the best practices and applications of these digital tools.

DIGI-EL is a project by Karlstad university that studies the effects digital applications can have for the well-being of women in early labor. The goal of this project is to evaluate if activity-wristbands can help alleviate stress a woman can experience when pregnant.

As Karlstad University does not have the resources to manufacture their own activity-wristbands for research-use, they are testing the viability of commercial fitness-trackers instead. This comes with its own set of challenges, such as potential inaccuracies with the sensors and concerns about data-protection and privacy as data collected by the devices are sent abroad to American servers, something that is in conflict with European data-safety and Swedish research-laws.

As DIGI-EL does not possess the means to build their own wristbands, their research into this field depends on wristbands made for personal use by corporations. Relying on commercial wristbands poses its own set of challenges that need to be understood and that's why our group is part of a second bachelor-project evaluating the safety and usability of these wristbands.

What differentiates our project from the previous one (Tunc, J., Bahmiary D., et. al, 2020) is our focus on only one fitness-tracker, the Fitbit Charge 4, whereas the 2020 study investigated the strengths and weaknesses of different brands. In spite of these differences we hope to find that a focused study on a singular activity-wristband provides useful and unique insight into the benefits and challenges these devices can provide in the research being conducted by DIGI-EL.

1.3 Research and Project-Drafting

Our initial research and planning started through a series of group meetings where we talked among ourselves and with representatives from Karlstad University in order to determine the scope and requirements of the project. The talks with the representatives made it clear that they expected a process similar

to the one that had been submitted in 2020 and that there were very few restrictions to the project as long as we fulfilled the base requirements of testing the accuracy of the Fitbit Charge 4. We were asked to provide DIGI-EL with an assessment of its relevance in their studies and an assessment of its relationship with European data-protection laws. The application-part of the project had to interact with the data collected by the fitbit, so we made it a goal to ensure that the data is represented in a manner that is comprehensive and accessible to the imagined end-user, which are nurses and medical researchers associated with DIGI-EL.

As emphasis was placed on the 2020 report we used it as the outline for our own testing. In the following meetings we broke down their methodology, we figured out what worked and we started drawing up a plan for our own testing. This planning became the pre-project report.

In these meetings we also agreed that the application should be built in React as the team had previous experience with Javascript and it would be a useful learning experience to build a react-web-application from scratch, especially because the project involved interacting with a foreign database¹ with user-information through API-calls².

The first thing we drafted was the form and schedule of the testing. We agreed on three primary intensities of activity we would be gathering data from and we made a spreadsheet outlining the timeline of the overall project. Since the entire project relied on data we would gather through field testing, we made it a priority to finish testing before we started development on the application that would

¹ [Glossary - Database](#)

² [Glossary - API-call](#)

handle the data we could gather through the testing, as the Fitbit would be saving the measurements we made.

1.4 Report Structure and Chapter Overview

In this section we will introduce the structure of our report and a brief overview of each chapter.

1. Introduction explains our relationship to the project, our workflow and the draft.
2. Theoretical Framework explains the necessary theory around our programming solution.
3. Product Development elaborates on the practical process of developing the application.
4. Product Specification describes the requirements we set for the application.
5. Product Documentation is a guide for those interested in developing and maintaining the app. It covers the most relevant parts of the code, how it works, and why it was implemented the way it is.
6. User Manual is a user-oriented manual to the application.
7. Fitbit, GDPR and Privacy investigates Fitbit's compliance with data-protection laws and privacy concerns in the context of DIGI-EL's research.
8. Experiments and Methodology explains our testing, our approach to testing and our results.
9. Conclusions describe our findings and reflections.

2. Theoretical framework

This chapter explains the necessary theory around our programming solution, primarily the usage of Fitbit Web API. Inner workings of the API will be discussed in this chapter together with other relevant theory background.

2.1 React.js

We chose to use a JavaScript library called React.js for this project (React, n.d.a). React is a front-end³ library designed to build user-interfaces. Compared to vanilla JavaScript, it offers some benefits that make web-applications easier to develop. One of the main benefits of React is reusable components (React, n.d.a). Components are building blocks in React that encapsulate state and logic into reusable user-interfaces. These components can then be reused throughout the application. This brings more flexibility and robustness to the development.

React would allow us to build a stable and comprehensive web-application that interacts smoothly with tasks like accessing data provided by a foreign server and funneling data into charts. This was thanks to the many existing solutions and libraries that exist in the React ecosystem. React also gave us flexibility as it could be modified to suit any request our clients might have had down the line, which allowed us to future-proof our project as we were helping the clients make specific demands as to what they would want this application to do.

³ [Glossary - front-end](#)

2.2 Fitbit Web API

2.2.1 Background

Our application required us to interact with the Fitbit Web API (Fitbit Developer, n.d., a) which is provided for free by the Fitbit corporation. An API stands for Application Programming Interface and is somewhat like a strict receptionist for a company that develops interactive digital software (Red Hat, 2017). It's a way for somebody outside of the company to be given directions to specific data or features that the software company wants other companies or independent developers to make use of. This allows external developers to build their own software out of data obtained from the company's software without the company exposing any vulnerabilities or copyrighted code by giving foreign developers direct access to the source code of the software.

Fitbit's Web API communicates with the external system through HTTP⁴ requests. HTTP requests are methods that indicate what kind of operation is going to be performed for a given resource (MDN Web Docs, 2021). GET, POST, UPDATE and DELETE are the most commonly used HTTP request methods. In our case, to obtain the data from the Fitbit server we will primarily use GET requests.

2.2.2 Prerequisites

To interact with Fitbit's Web API, the first step is to create a Fitbit developer account. The developer is then required to register an application using this link: <https://dev.fitbit.com/apps>.

Registered application is then given a client ID and a secret-key. These keys are later needed for the authorization process (Fitbit Developer, n.d., b).

⁴ [Glossary - HTTP](#)

2.2.3 Authorization

Interacting with the Fitbit API is not frictionless. To access data for our application we first need to Authorize the users. Authorization is a security concept that providers ensure that those who need to access data on their servers or API get the proper permissions to do so while preventing the data from being intercepted or accessed by anyone without permission. OAuth2 (<https://oauth.net/2/>) is the industry-standard authorization protocol and it was the one we interacted with making our application. OAuth2 verifies the authorization request sent by the provider's system and then returns a unique token which gives access to the data for the user. Sequence diagram of how our application authorizes users through OAuth2 is shown in figure 2.1.

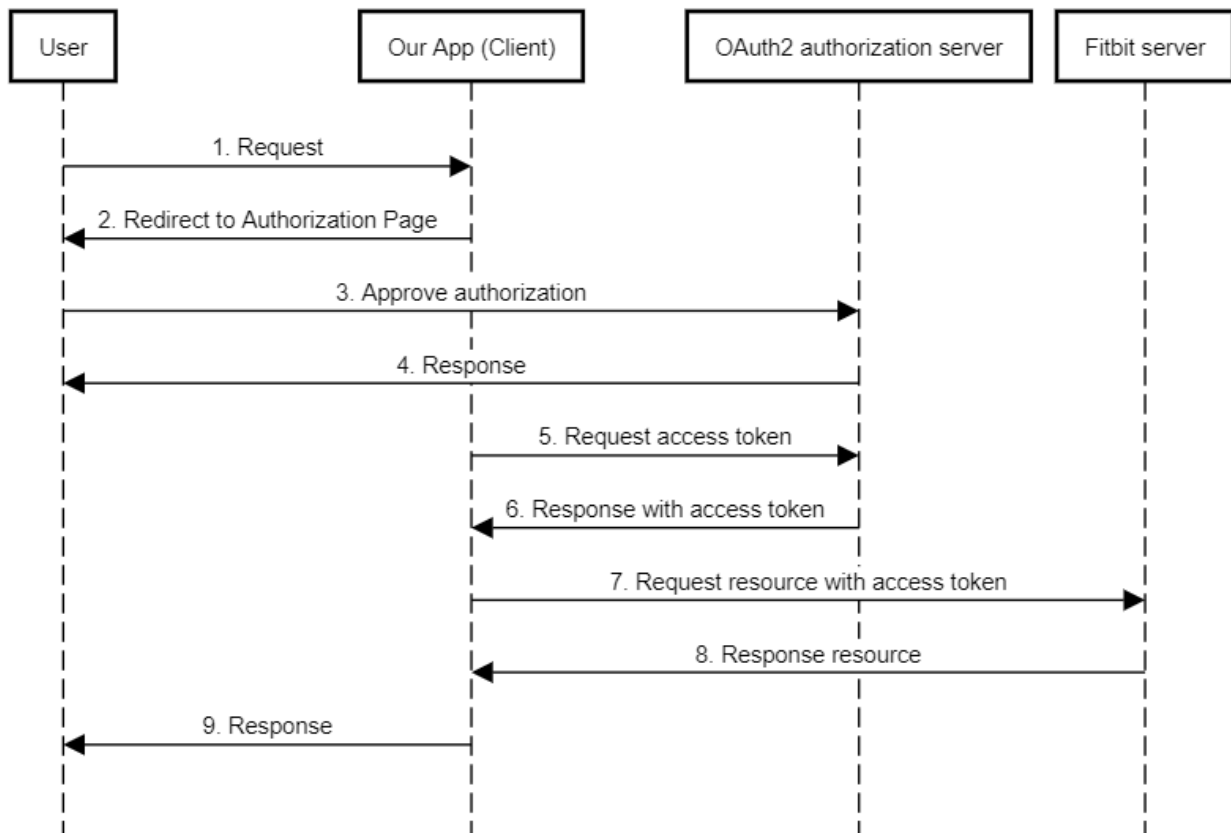


Figure 2.1: Sequence diagram shows how the user is authorized to our application

2.2.4 API endpoints

Our interaction with the Web API centered around the endpoints⁵ that corresponded with the data we were measuring and testing. These endpoints gave us access to the data the Fitbit Charge 4 had collected about our heart rates, steps taken, sleep and calories that we've noted in the application or on the wristband. We used five primary API endpoints for our project:

1. **/1/user/[user-id]/activities/heart/date/[start-date]/[end-date].json** is a GET request that retrieves heart rate time series data over a period of time by specifying a date range. The response will include only the daily summary values.
2. **/1/user/[user-id]/activities/heart/date/[date]/1d/1min.json** is a GET request that retrieves the heart rate intraday time series data on a specific date in 1 minute intervals.
3. **/1.2/user/[user-id]/sleep/date/[startDate]/[endDate].json** is a GET request that retrieves a list of a user's sleep log entries for a date range.
4. **/1/user/[user-id]/activities/steps/date/[start-date]/[end-date].json** is a GET request that retrieves the steps data with specified date range.
5. **/1/user/[user-id]/activities/calories/date/[start-date]/[end-date].json** is a GET request that retrieves the calories burned data with specified date range.

2.2.5 Making requests

To make requests, we need to specify the user-id as an URI⁶ argument. Each user will have a unique user-id which helps to identify whose data the API is going to return. User-id can also be substituted with “-” (dash) for currently

⁵ [Glossary - Endpoints](#)

⁶ [Glossary - URI](#)

logged in users. As a part of URI arguments, it is also necessary to define date or date-range that the API uses to return desired results. Based on the specific endpoints there may be additional URI arguments that need to be filled out. For example, if the developer wants to retrieve the amount of steps taken, the resource URI argument is also required, as illustrated in figure 2.2.

URI Arguments

user-id	required	The encoded ID of the user. Use "-" (dash) for current logged-in user.
resource	required	The resource of the data to be returned. See supported values in the Resource Options section.
start-date	required	The start date specified in the format <code>yyyy-MM-dd</code> or <code>today</code> .
end-date	required	The end date specified in the format <code>yyyy-MM-dd</code> or <code>today</code> .

Figure 2.2: Shows URI arguments that are required to make an API call to fetch the step count. Taken from Fitbit Web API documentation:

<https://dev.fitbit.com/build/reference/web-api/activity-timeseries/get-activity-timeseries-by-date-range/>

Furthermore, we require an authorization header to be filled in with the user's access token⁷, as seen in the figure 2.3 located below.

The screenshot shows the Postman interface for an API call. The method is GET and the URL is `https://api.fitbit.com/1/user/-/activities/steps/date/2022-03-01/2022-05-16.json`. The Headers tab is selected, showing 6 headers. One header is visible: Authorization: Bearer eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiIyMzg5OU0iLC...

KEY	VALUE	DESCRIPTION	...	Bulk Edit	Presets
<input checked="" type="checkbox"/> Authorization	Bearer eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiIyMzg5OU0iLC...				
Key	Value	Description			

Figure 2.3: Shows the setup of making an API-call to retrieve step count using Postman as the tool. To make a valid request the Authorization Header is filled out, as well as URL with required arguments such as dates, user and resource.

⁷ [Glossary - Token](#)

GET requests send back a JSON⁸ response after a successful request, which is a clean and efficient way to organize data with key-value pairs. The basic structure of a JSON response can look something like this as shown in the figure 2.4:

```
{
  "activities-steps": [
    {
      "dateTime": "2019-01-01",
      "value": "0"
    }
  ],
  ...
}
```

Figure 2.4: Example JSON response, after a valid GET request.

In this example “activities-steps” is a key that holds an array as a value. Array itself contains key-value pairs that point us, and our program, to the data that is relevant for our purposes. Predictable and standardized nature of JSON responses makes it easy to convert them to the native JavaScript objects.

2.2.6 Scopes

Developers are not allowed to access user’s information without the user consent. The consent is controlled through a list of scopes, where each scope is responsible for a specific category of data collection, as seen in figure 2.5.

⁸ [Glossary - JSON](#)

activity	Includes activity data and exercise log related features, such as steps, distance, calories burned, and active minutes
heartrate	Includes the continuous heart rate data and related analysis
location	Includes the GPS and other location data
nutrition	includes calorie consumption and nutrition related features, such as food/water logging, goals, and plans
profile	includes basic user information
settings	includes user account and device settings, such as alarms
sleep	includes sleep logs and related sleep analysis
social	includes friend-related features, such as friend list, invitations, and leaderboard
weight	includes weight and body fat information, such as body mass index, body fat percentage, and goals

Figure 2.5: Table of all available scopes. Taken from Fitbit Web API documentation:

<https://dev.fitbit.com/build/reference/web-api/developer-guide/application-design/#Scopes>

Each API-call depends on some sort of scope. That means if the user hasn't consented to a specific scope that the API-call depends on, the data won't be retrieved and the request will return an error message. Users can allow/decline scopes in the Authorization panel, as shown in figure 2.6.

fitbit

ExampleApp by **Example Co.** would like the ability to access and write the following data in your Fitbit account.

- Allow All
 - weight ⓘ
 - location and GPS
 - Fitbit devices and settings
 - profile ⓘ
 - food and water logs ⓘ
 - activity and exercise
 - sleep
 - heart rate
 - friends ⓘ

If you allow only some of this data, ExampleApp may not function as intended. Learn more about these permissions [here](#).

Deny **Allow**

The data you share with ExampleApp will be governed by Example Co.'s [Privacy Policy](#) and [Terms of Service](#). You can revoke this consent at any time in your Fitbit [account settings](#).

Signed in as [api@example.com](#)
[Not you?](#)

Figure 2.6: Users can accept / decline specific scopes of data when logging in to the application. Taken from Fitbit Web API documentation: <https://dev.fitbit.com/build/reference/web-api/developer-guide/application-design/#Scopes>

2.2.7 Intraday

The Web API also provides a feature called Intraday, which allows a developer to access the minute-by-minute and second-to-second heart-monitor or activity readings from the wristband. This allows very close scrutiny of the data provided by the wristband, but due to the sensitive nature of this comprehensive insight into a person’s body and life Intraday data is only provided to developers that fill out a special form. Our team had to fill out this form as a minute-by-minute overview of heart rate data had been requested by our clients in our last meeting with them.

2.3 Data export

2.3.1 Preface

Our client at Karlstad University wanted to know what options are there to export the data collected by the Fitbit tracker. We explored different ways we can export the data and will describe them here.

2.3.2 Web API

The public Web API exposes virtually all data collected by the tracker. The data is made available by making API calls - HTTP requests that return a JSON response body containing the extracted data. This approach is suited most when the data needs to be implemented into the application. A downside of this approach is that it requires programming knowledge. At the same time, JSON-object would need to be reformatted to make the data more readable. Web API also has a specific endpoint that returns user's collected activities in TCX file format instead of JSON.

2.3.3 Fitbit official desktop app

Users' collected data can be exported directly from the Fitbit's official web application. In the desktop version the data can be exported by first navigating to the Settings (Figure 2.8) and then clicking on the Data Export option.

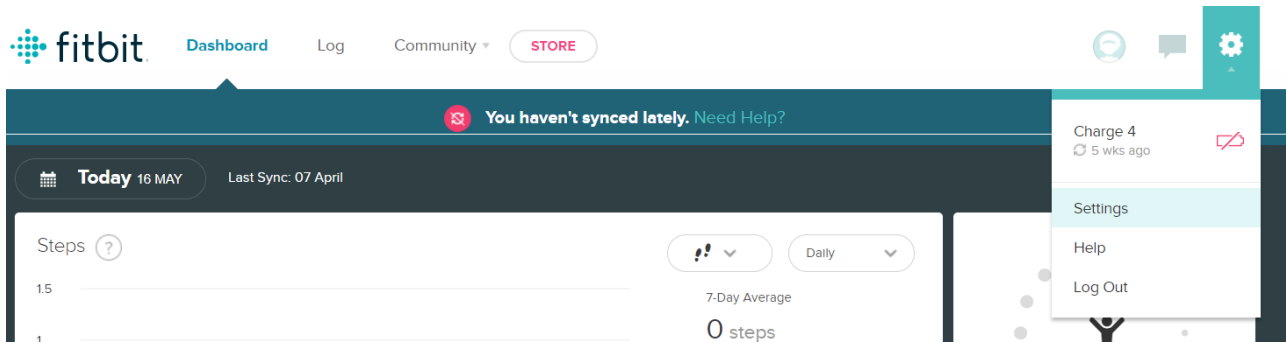


Figure 2.8: In the Fitbit Web application Data Export function can be found in Settings.

Here, the user has two options for data export. User can either choose to export the data by specifying the time period or export the entire account archive. Supported file format for download is CSV or Excel file format. The export options are shown in the figure 2.9, located below.

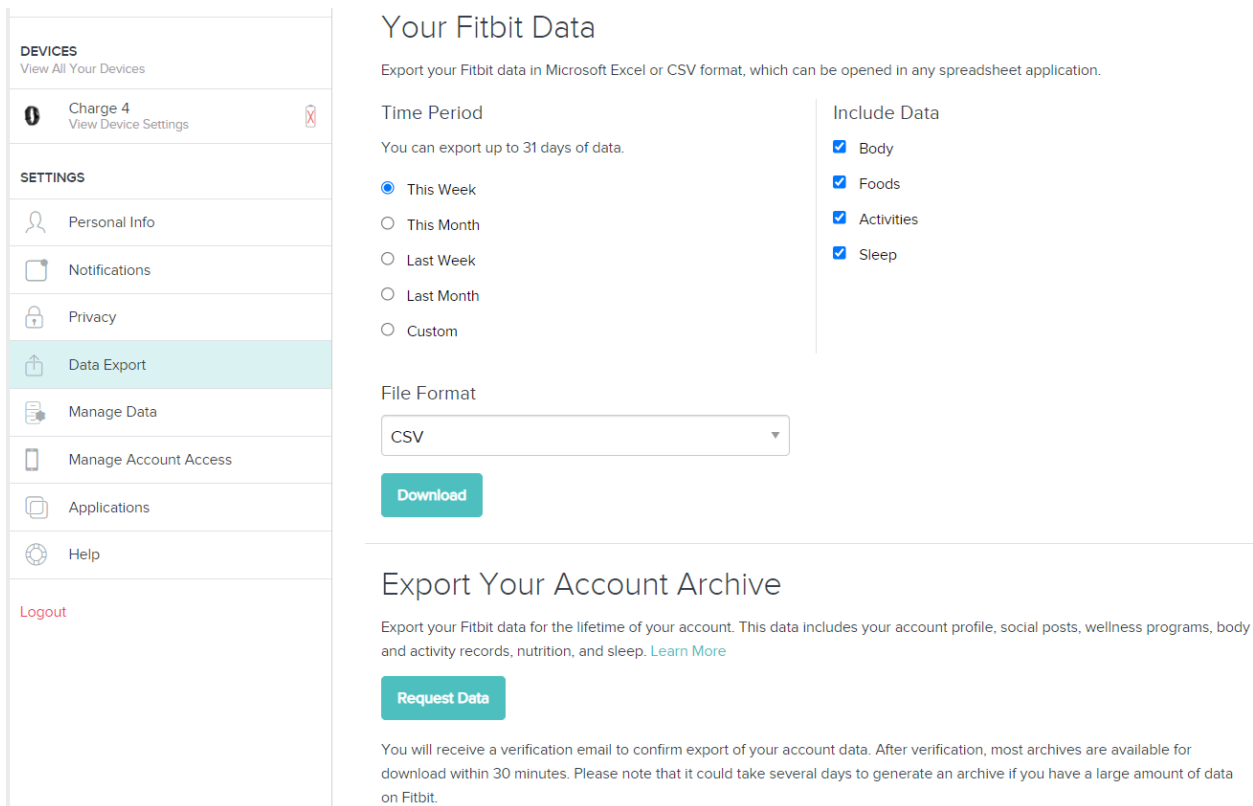


Figure 2.9: From the Fitbit Web application the user can export the entire account archive, or export data by specifying the scopes and time period.

Additionally, we can export data collected during activities. This can be achieved by first navigating to the “Log”, found in the navigation bar at the top of the window, and then clicking on the Activities tab. Activities will be listed under “Activity History”. User can click “View details” on a desired activity to reveal additional information about it. To export the data, user has to click on the three-dot menu located at the upper right of the panel, and then to click on “Export as a TCX File”. Steps are illustrated in figure 2.10.




Figure 2.10: Activities can be downloaded as TCX files.

2.3.4 Third-party applications

When our team tried to export activities containing the test data that we performed, we encountered some problems. The downloaded file contained an empty dataset. After some research we found out that other people had a similar problem. This is possibly a bug and might be fixed in the future. This problem led us to explore other third-party options. The application that we found is called FitToStrava (<https://www.fittostrava.com/>) and it worked well for us for exporting activity data (Figure 2.11).

Choose a date and then click on "Sync with Fitbit" or "Sync with Fitbit for more days"

2022-02-09 

[Sync with Fitbit for more days](#) [Sync with Fitbit](#)

Activities Summary

List of all activities - February 9, 2022


Date	Activity	Duration	Total time	Calories	Steps	Distance	Heart Avg	
February 9, 2022 14:45-15:15	Walk	0 h 30 m 13 s	30 min	231 cal	2818	2.068 Km	119 bpm	Details Export to Strava 

Figure 2.11: Our team used the app FitToStrava to export the data from the tests that we performed.

To use the application we had to authorize with the Fitbit account. Additionally, we had to create a Strava account. Afterwards, the activity data could be accessed by the application and therefore exported as a TCX file.

3. Product development

In this chapter, the process about our product development will be discussed. Chapter will define aims and goals for the product and at the same time will give more insight into the several development phases that took place while working on the product.

3.1 Aims

Karlstad University did not give us any distinct requirements for the utility of the application we would develop in conjunction with our research and testing. Therefore we set our own requirements for the project. We wanted to build a light-weight web-application that could demonstrate the capacity of the Fitbit API in the context of health-overview and research. We figured this was a good primary focus because the only technical requirement we were given was for the application to interact with the data collected by the Fitbit Charge 4 in some way.

This culminated in a lot of freedom to do our own thing and we used this freedom to build the application around a simple interface that would make the data collected by the Fitbit easily accessible on a dashboard. We would later revise this design and make additions based on feedback by our clients in Sweden, but the core idea remained the same for the entire development process. As we were dealing with very rudimentary data we felt it would benefit the project if we presented our solution in a clear way. We didn't want the application to grow too complex as too much functionality could distract from the data the hypothetical end-user would attempt to access.

Here are the goals that we set ourselves for this product:

- The application utilizes Web API to get the data from the fitness tracker. We will focus on the data from heart rate, sleep, steps taken and calories burned.
- The application will display the collected data in an easy to visualize manner. App will use charts to help with that.
- The application will provide summaries of the different data that has been collected for a given period of time.

3.2 Work methodology

Due to the more scientific nature of this project, we did not adapt any specific software development workflows. With that said, our development methodology was leaning towards an agile approach. The reasons for this approach were changing product specification and not-clearly defined end-product.

Agile is an iterative approach to software development that focuses on delivering the product in small increments. Requirements and documentation are continuously evaluated alongside the development (Atlassian, n.d.). This approach is therefore well suited when a product is going to change or evolve over time, which is a case for our product.

To organize development of the product the version control was needed. Git and GitHub were our choices. Having the product uploaded to the Git repository allowed our team to work on implementing features in parallel. We could also track changes made to the code, which would allow us to backtrack if any problems occurred with the code.

3.3 Development phases

The development consisted of the following phases:

- Research phase
- Design phase
- Implementation phase

3.3.1 Research phase

Before we could work on implementing the code, there were several areas that needed to be researched first. One of the first things that needed figuring out was what kind of application we were going to create. Since the project did not provide many specific requirements regarding the application development, the type of application that would be developed was chosen by the team. With several of the group members having experience with web applications, we decided it would be most appropriate to build our product as such.

Next, it was important to research what kind of technologies and libraries would be most suited for the size and scope of our product. In one of the team meetings, we agreed that the application should be built in React.js. The team had previous experience with JavaScript and it would be a useful learning experience to build a web-application with an in-demand JavaScript library. The other reason why we chose React.js, is that it provides an easier and more convenient way to build web-applications by using reusable components. Our team began learning React by getting familiar with its main concepts, in this phase.

Additionally, we have decided to develop our product as a progressive web app (PWA). The idea behind PWAs is that they use web technologies to provide the

experience of native apps. PWAs are just websites behind the scenes that can be installed (Freecodecamp, 2021). This allows our product to behave both like a website and a mobile app at the same time.

3.3.2 Design phase

In the design phase, we brainstormed on how the application should look. We used prototyping tool Figma to create a low-fidelity prototype to make a basic layout of the user interface (Figure 3.1). We then incrementally made improvements to the sketch and created a medium-fidelity prototype (Figure 3.2). The medium-fidelity prototype is a basis for our design of user-interface.

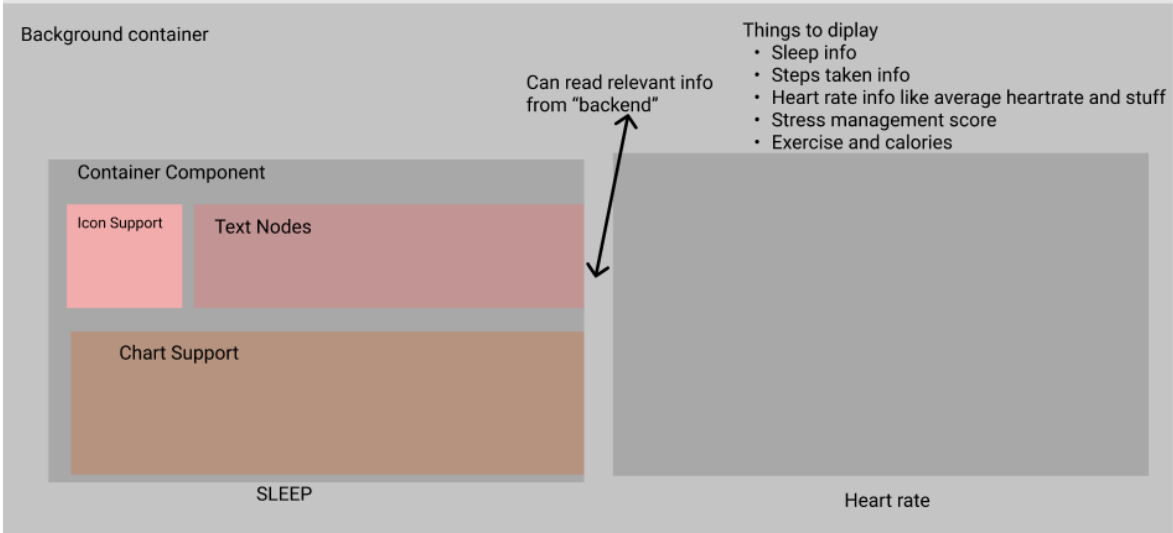


Figure 3.1: Low-fidelity prototype of the application

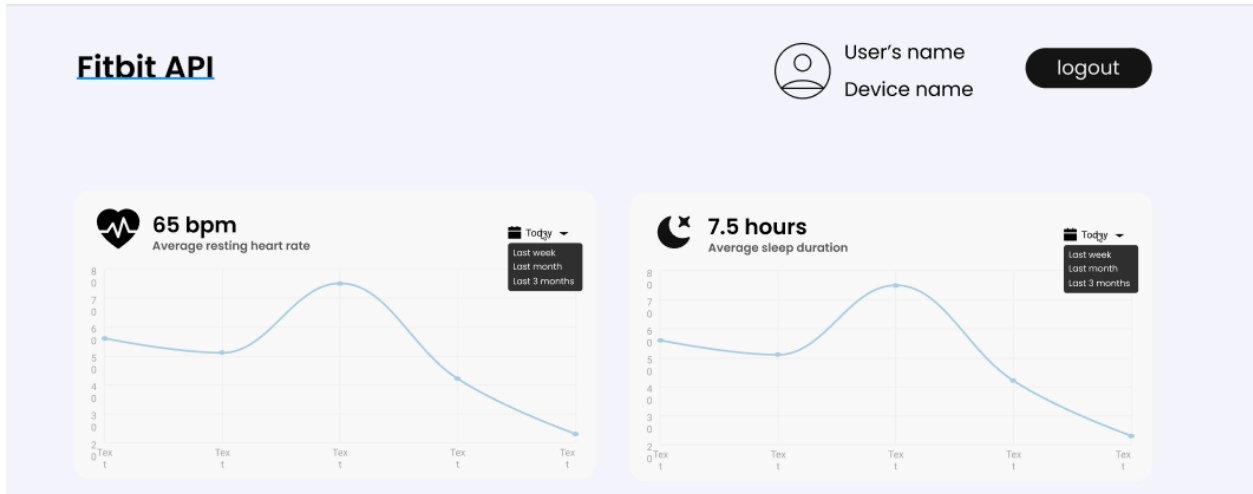


Figure 3.2: Mid-fidelity prototype of the application

While iterating on design of the user-interface our goal was to make it user-friendly. To accomplish that, we relied on a certain set of rules called Gestalt Principles. Gestalt Principles describe how humans perceive visual objects (Bradley, 2014). By understanding these principles, one can design the product in a way that is easy to understand for the human perception.

Gestalt's Principle of proximity was used throughout the user-interface. This principle indicates that objects that are grouped more closely with one another are perceived as more related, while objects that are placed further from one another are perceived as less related (Bradley, 2014). We applied this rule in our user-interface to indicate that some elements belong to a group. For example, we can see it in the finished product, shown in figure 3.3. Each dashboard component has several interface elements grouped together, such as a header containing an icon with summary text as well as a date selector and a chart. These elements differ in their composition and appearance, nonetheless they look like they represent one combined element instead of several individual ones.

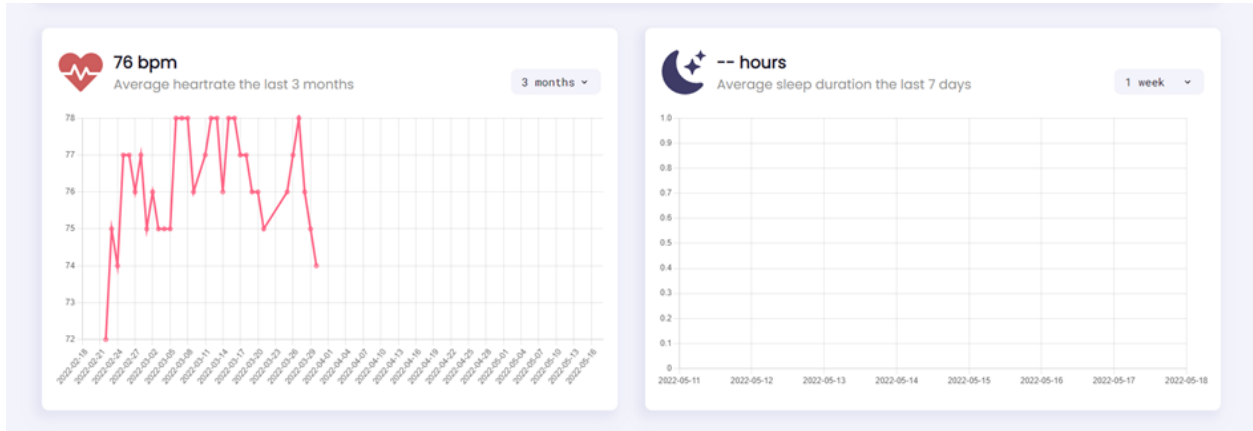


Figure 3.3: Dashboard component user-interface

The figure 3.4 illustrates that even further. In this case the background of dashboard components is removed, yet these elements still appear to be related to one another.

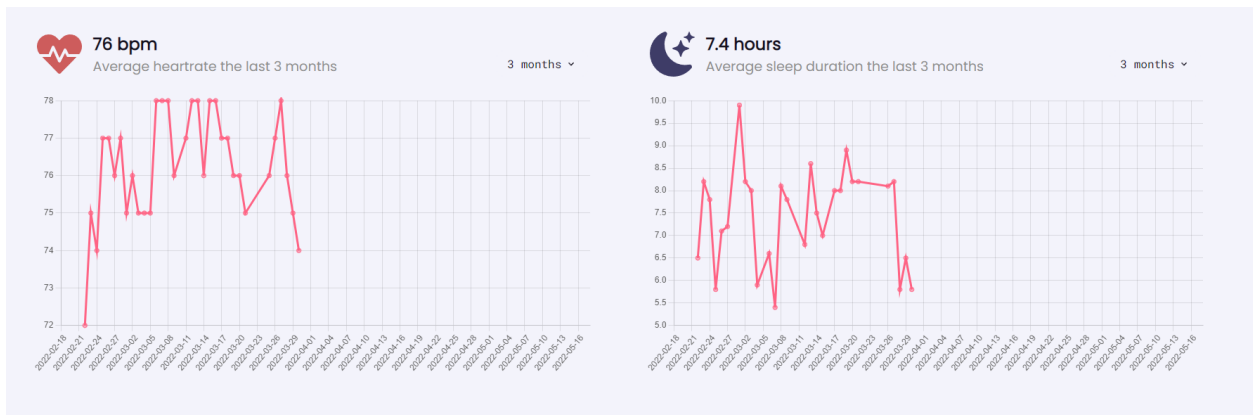


Figure 3.4: Dashboard component user-interface without the white background

Common Region is the other principle that we applied in our design. In our case it goes hand in hand with the law of proximity. Principle of Common Region describes that we perceive objects as related when they are placed in the same closed region (Bradley, 2014). Figure 3.3 shows the dashboard components enclosed in a common region while figure 3.4 shows the components without common region applied. As we can see from the figures, it is arguably easier to differentiate between heart rate and sleep component in figure 3.3.

Additionally, our design incorporated the principle of Figure / Ground. This principle states that: “Elements are perceived as either figure (the element in focus) or ground (the background on which the figure rests)” (Bradley, 2014). In our case, the principle is being utilized in the dashboard components, as shown in figure 3.3. Common Region and Figure/Ground principles work together in this case. The area of Common Region provides a shape while the contrast of colors between background and a dashboard component defines a relationship for Figure/Ground. Therefore, dashboard components appear to stand out from a background and are perceived as elements in focus. As a comparison, figure 3.5 contains the principle of Common Region without Figure/Ground. In this case, the components don’t stand out from the background as much, and as a result are not perceived as elements in focus to a degree that figure 3.3 illustrates.

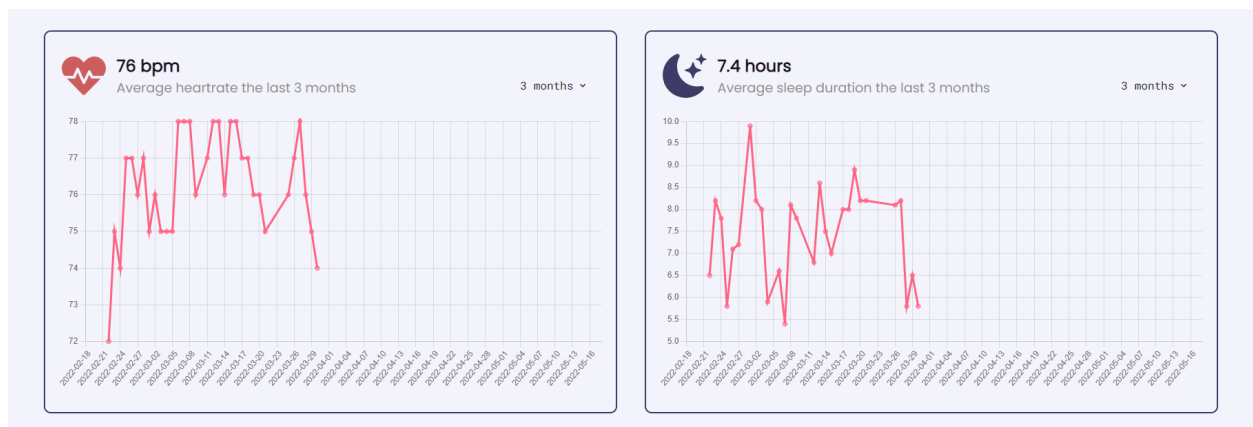


Figure 3.5: Dashboard component user-interface without Figure/ground principle

3.3.3 Implementation phase

Implementation phase revolved around coding the application in React. The source code was stored in Github. Our team would create a new branch in Github each time a new feature was being implemented. The code then would be reviewed before merging the newly added feature to the Main branch. This way

of working allowed for parallel implementation of features. At the same time, having multiple branches proved to be an effective way of not breaking the application completely, since the Main branch would only contain the reviewed features.

Certain prerequisite knowledge was required, in order to develop the application effectively. Foremost, it required us to have a decent understanding of JavaScript, since React uses this programming language. Furthermore, as our product is a web application, understanding of HTML⁹/CSS¹⁰ was also needed.

In this phase, we began by exploring Fitbit Web API documentation to determine what exact endpoints we would need to use in order to get the data that we wanted to display in the application. The endpoints that we ended up using are discussed in [Chapter 2.2.4](#).

While developing the React app, our team thoroughly referenced the official React documentation. This was done for a couple of reasons. First of all, since our team members had no prior knowledge of React, it was necessary for us to gain understanding of how this technology works. Additionally, as a library React is quite flexible. Meaning, that the same code logic can be achieved in multiple ways. At the same time, it is constantly evolving by having new features added. Therefore, we tried to follow best practices and methods to the best of our ability that were suggested by the React documentation.

For example, React has two approaches to creating components. Class-based components that are using classes and Functional components that are just functions (React, n.d., b). Up until version 16.8, the Class components were

⁹ [Glossary - HTML](#)

¹⁰ [Glossary - CSS](#)

primarily used by developers since they allowed for logic and state implementation while functional components mainly were responsible for just rendering UI. However, since version 16.8 came out, React introduced the concept of Hooks. They let developers use state and other React features without writing a class (React, n.d., c). As of 2022, functional components with hooks are the preferred method to create components. That's because hooks were introduced to address the problems that developers encountered while working with class components (React, n.d., c).

3.3.4 Limitations

Adding a testing phase could've been beneficial to find any potential bugs and to validate if the software performs as expected. The same could've been said about user testing. Which is a type of testing where users would evaluate design and usability of the user-interface. On the other hand, it would have been quite difficult to find the users for this kind of test since the application requires them to have a Fitbit account and a wristband. In the end we decided not to include these tests because a good portion of the available time on the project was already taken up by the armband evaluation and testing.

4. Product Specification

4.1 Initial requirements

Since requirements for the application from the client were minimal, our team had to come up with our own requirements for this application. We decided that initial requirements should be minimal, that is to cover the basic minimal viable functionality. Our initial requirements focused on displaying the collected data into the charts as well as providing summaries in form of average values based on a time period. If most of the initial requirements are to be completed, we then will iterate with new requirements and functionalities.

Tables below show initial requirements that have been decided before the implementation phase.

#	Functional requirements
1	The system displays user's name
2	The system displays wristband model
3	The user must be able to login to the system with Fitbit account
4	The user must be able to log out of the system
5	The system implements Fitbit Web API
6	The system shows summary of the average heart rate for the given period
7	The system shows summary of the average sleep duration for the given period
8	The system shows summary of the average step count for the given period
9	The system shows summary of the total calories burned for the given period

10	The user can filter heart rate, sleep, step count and calories burned data by date periods
11	The system must be able to provide charts for the given data

#	Non-functional requirements
1	The system has a responsive design
2	The system provides feedback for the user if elements on the page are loading
3	The system provides feedback for the user if error has occurred
4	The system has PWA support
5	The data takes less than 3 seconds to load
6	The system uses icons to help differentiate the data
7	The system uses HTTPS

#	Organizational requirements
1	The system is a web application
2	The system is built using React.js

4.2 Changes in the requirements

Nearing the end of the development our team had a meeting with a client from Karlstad University. In the meeting they gave us additional requirements for the application. One of the things that they wished for to be in application, was some sort of calendar window that displays summary for the given day. That is, it would show heart rate, sleep and steps data. Besides that, they wished for some

functionality that would let the pregnant woman's partner to interact with the application in some way. This was brought up by the client because according to them, the partner often feels left out and wants to help the significant other during pregnancy.

Tables below show requirements that have been added during the development.

#	Functional requirements
1	The system should have a calendar view that gives summaries for each day with heart rate, sleep and step count info.
2	The system should provide minute-by-minute heart rate overview for the given day

#	Non-functional requirements
1	The system should be accessible by the pregnant women's partner

5: Product Documentation

5.1 Preface

Product documentation covers the most important parts of the code for those interested in further developing or maintaining the application. This chapter attempts to explain how the code works and why it is implemented the way it is.

Source code (Zip):

drive.google.com/file/d/1G0az8shskdrMG9G_5Bjg7-Wr90e7iNN9

Github:

github.com/cosmosgirlandcrows/Bachelor2022_FitbitreactJS

5.2 Third-party libraries

To make the development easier for us, the React project implemented several third-party libraries. This allowed our team to focus on the core aims of the product instead of spending too much time coding everything from scratch. All third-party libraries are open-source and fall under MIT license¹¹. Here's the list of the libraries that were used in this project:

- Charts.js - <https://www.chartjs.org/>
- Moment.js. - <https://momentjs.com/>
- Html-react-parser - <https://github.com/remarkablemark/html-react-parser>
- React-Calendar - <https://projects.wojtekmaj.pl/react-calendar/>
- React-Icons - <https://react-icons.github.io/react-icons/>
- React-Outside-Click-Handler - <https://github.com/airbnb/react-outside-click-handler>

¹¹ [Glossary - MIT license](#)

5.3 Running the application

To run the development version of the program, while having root folder of the project as current directory, first type the following in terminal:

```
Budre@DESKTOP-UCJU21H MINGW64 ~/Desktop/Bachelor2022_FitbitreactJS (main)
$ npm install
```

This will install all of the needed dependencies for the project. This command is only required on initial start-up. After it has finished installing dependencies, type the following to start the app:

```
Budre@DESKTOP-UCJU21H MINGW64 ~/Desktop/Bachelor2022_FitbitreactJS (main)
$ npm start
```

Browser will open the web-application in a new tab/window with the URL of `localhost:3000`.

5.4 React concepts

In order to understand most of this product documentation, one should be familiar with some of the core concepts of React. Therefore, some of them will be discussed here.

5.4.1 Components & Props

Our application is built from functional components. Functional components look just like normal JavaScript functions (React, n.d., b). Each functional component has a return statement that renders the UI. Additionally, components accept

arguments called `props`. They are useful when the data needs to be passed from one component to another. In the code snippet shown below, `title` and `onClick` are `prop` values. These values are used in the `return` statement. We can render multiple instances of this component with different properties. This results in a dynamic and reusable component.

```
1  const Button = ({ title, onClick }) => {  
2    return (  
3      <button className="btn" onClick={onClick}>  
4        {title}  
5      </button>  
6    );  
7  };  
8  
9  export default Button;
```

In order to incorporate this component in the application, it must be exported as shown in the snippet above. The component must then be imported by the file that requires it. The following code below shows how to accomplish it.

```
1  import Button from "./Button";
```

As illustrated in the following code, to render a component an HTML syntax is used: `<Button/>`. The components should always start with a capital letter to help differentiate between HTML-tags.

```

1  return (
2    <header className="mainHeader">
3      <h1>DIGI-EL Dashboard</h1>
4      <Profile
5        access_token={access_token}
6        apiProfile={apiProfile}
7        apiDeviceInfo={apiDeviceInfo}
8      />
9      <Button onClick={() => logout()} title="Logout" />
10   </header>
11 );

```

5.4.1 Hooks

To incorporate state and logic into components, we can use Hooks. According to React documentation hooks are functions that let functional components “hook into” React state and other features (React, n.d., d). Every hook in React starts with prefix *use*. Some of the most common hooks are `useState` and `useEffect`.

`useState` lets developers add React state to the functional component. The hook returns a pair of values: the current state and a function to update the state. What makes the state differ from normal variables is that the state persists between renders and does not get reset. Below is an example of the `useState` hook inside our app.

```

1  const [title, setTitle] = useState("Average heartrate on");

```

`useEffect` is a hook that executes a block of code when something causes the component to re-render. By default it gets called on every re-render, although by passing a dependency array as a second argument we can control when the `useEffect` will run. For example, in the code snippet below, the code inside the hook will only run when the variable `selected` changes value.

```
1  useEffect(() => {
2    handleOnChange(selected);
3  }, [selected]);
```

React also allows you to build your own Hooks. Custom Hooks can be used when some logic needs to be shared between several components. In order to create a custom hook, the function must start with prefix `use` and the function should call other hooks inside it. The following code is a custom hook that we made in order to save values to `localStorage`.

```
1  const useLocalStorage = (key, defaultValue = "") => {
2    const [value, setValue] = useState(() => {
3      return checkForValueInStorage(key, defaultValue);
4    });
5
6    useEffect(() => {
7      localStorage.setItem(key, JSON.stringify(value));
8    }, [value]);
9
10   return [value, setValue];
11  };
```


5.5 Authorization

Authorization in the app works by checking if the access token is present in the URL bar. Authorization logic takes place in `App.js`, which is a root component of the app. First when the code inside `App.js` gets executed, the URL of the site is saved to a variable – `window.location.href` returns the URL of the current page. The `url` variable is then used as an argument for functions to extract `userId` and `access_token` if they are present in the URL. By default, when a user enters the website for the first time, `access_token` and `userId` will be empty. This code is showcased below:

```
1 function App() {
2   const url = window.location.href;
3   const access_token = getAccessToken(url);
4   const userId = getUserId(url);
5   const BASE_URL = "https://api.fitbit.com/1/user/";
```

The code then has an *if-statement* that checks for `access_token` as shown in the code snippet below. If `access_token` is an empty *String*, that means that the user has not authorized yet and as a result the code inside *if-statement* will get executed. We use the authorization endpoint provided by the Fitbit Web API and assign it to `window.location.href`. This redirects the user to the Authorization panel. After successful authorization, the user is redirected back to the main URL of the application. The URL now contains `access_token` and `userId` as URI arguments.

```

1  if (access_token == "") {
2      window.location.href =
3          "https://www.fitbit.com/oauth2/authorize?response_type
=token&client_id=██████████&redirect_uri=https://fitbit-bach202
2.herokuapp.com/&scope=activity heartrate location nutrition
profile settings sleep social weight&expires_in=604800";
4  }

```

5.6 Context

The variables `access_token`, `userId` and `BASE_URL` are required for making API requests. As a result those variables need to be passed down to components that do the data-fetching. One way to do it is to pass them as `props`. This will work great if the nested component is a direct child of `App.js`. However, if the data-fetching component is nested in any other components, the `props` will have to be passed through all of those nested components even though they do not require these values. The better way to do it, is to use `Context` API. By using `Context` the data can be shared between components without passing `props` through every level of the tree (React, n.d., e). Instead, the components can access the data directly from the `Context` scope. The code below shows how the `Context` is implemented inside the `App.js` component.

```

1  return (
2      //Values inside AuthenticationContext can be accessed by any child component
3      <>
4          <AuthenticationContext.Provider value={{ userId, access_token, BASE_URL }}>
5              <Header />
6              <Grid />
7          </AuthenticationContext.Provider>
8      </>
9  );

```

Every child component inside `<AuthenticationContext.Provider/>` has direct access to `access_token`, `userId` and `BASE_URL`. For example in our app, the `GridItemContainer` component needs `access_token` in order to make an API-call. It can get the `Context` data like this:

```
1 const { access_token } = useContext(AuthenticationContext);
```

We use `useContext` hook and pass the created `Context` as an argument. The hook returns values stored in `AuthenticationContext`. In this case, only the `access_token` is needed by the component.

5.7 Fetching data from the API

We used JavaScript's Fetch API to make HTTP requests to the Web API. To send a request, function `fetch()` is used. `fetch()` takes in a path to the resource as a first argument and an init object that can set custom settings to the request, as a second argument. Below is the `fetch()` implementation that our app used to make requests.

```

1  fetch(apiUrl, { ...headers, signal: controller.signal })
2    .then((response) => {
3      if (response.ok) return response.json();
4      throw `Error ${response.status}`;
5    })
6    .then((response) => {
7      setData(response);
8      setLoading(false);
9    })
10   .catch((error) => {
11     setError(error);
12   })

```

When a HTTP request sends back a response, the code inside first `then()` will get executed. On the successful request, the response will get converted to JSON and on failed request an error will be thrown. The second `then()` takes in the returned JSON-object and saves it to a state. If the error was thrown, `catch()` block will set the error object to a state.

In our app there are multiple components that need to fetch the data. Most of the data-fetching components require the same `fetch()` implementation. The component first needs to define `data`, `loading` and `error` states using `useState` hook. Component then needs to implement a `useEffect` hook with URL as a dependency and have `fetch()` logic inside it. The `useEffect` will ensure that `fetch()` will only send requests when the url changes. We would have to repeat these steps for every component that makes use of `fetch()`. This results in code duplication. Because of that, the better way to do it, would be to extract this logic to a custom hook. `useFetch` is a custom made hook that handles data-fetching logic in our application, as shown below.

```

1  const useFetch = (
2    access_token,
3    url,
4    headers = {
5      headers: {
6        authorization: "Bearer " + access_token,
7        accept: "application/json",
8      },
9    }
10 ) => {
11   const [data, setData] = useState(null);
12   const [loading, setLoading] = useState(false);
13   const [error, setError] = useState(null);
14   const [apiUrl, setApiUrl] = useState(url);

```

useFetch takes in access_token, url and headers as arguments. These values are then used in fetch() later on. Firstly, the hook has data, loading, error and url states defined. It then has a useEffect with fetch() logic inside it. The following snippet illustrates that.

```

1  useEffect(() => {
2    const controller = new AbortController();
3
4    if (apiUrl) {
5      setLoading(true);
6      setError(null);
7    }
8
9    fetch(apiUrl, { ...headers, signal: controller.signal })
10   .then((response) => {
11     if (response.ok) return response.json();
12     throw `Error ${response.status}`;
13   })
14   .then((response) => {
15     setData(response);
16     setLoading(false);
17   })
18   .catch((error) => {
19     setError(error);
20   })
21   .finally();
22
23   return () => {
24     setLoading(false);
25     controller.abort();
26   };
27 }, [apiUrl]);

```

Lastly, the hook returns `data`, `loading` and `error` states together with `setApiUrl`, which is a function to set the URL path, as demonstrated in the code below.

```

1  return [data, loading, error, setApiUrl];

```

useFetch can then be used inside different components like this:

```
const [profile, profileLoading, profileError] = useFetch(access_token, apiProfile);
```

5.8 Handling data

The data that gets returned from the Web API, can't be used in the UI right away. That's because most of the time our application requires only specific values from the API responses. Thus, the data needs to be handled in some way before applying it to the user-interface. For the majority of data, we use built-in array methods such as `map()`, `filter()` and `reduce()` to extract the wanted data.

For example in our app, `HeartrateComponent.js` requires heart rate data from the Web API. HTTP request that fetches heart rate returns a JSON response body containing all of the relevant values as shown in the snippet to the right.

We are only interested in `restingHeartRate` value from this JSON response as it will later be used for calculating average heart rate for the given date range period. In

```
1  {
2    "activities-heart": [
3      {
4        "dateTime": "2022-03-01",
5        "value": {
6          "customHeartRateZones": [],
7          "heartRateZones": [
8            {
9              "caloriesOut": 2316.51702,
10             "max": 124,
11             "min": 30,
12             "minutes": 1439,
13             "name": "Out of Range"
14           },
15           {
16             "caloriesOut": 5.96592,
17             "max": 148,
18             "min": 124,
19             "minutes": 1,
20             "name": "Fat Burn"
21           },
22           {
23             "caloriesOut": 0,
24             "max": 179,
25             "min": 148,
26             "minutes": 0,
27             "name": "Cardio"
28           },
29           {
30             "caloriesOut": 0,
31             "max": 220,
32             "min": 179,
33             "minutes": 0,
34             "name": "Peak"
35           }
36         ],
37         "restingHeartRate": 75
38       }
39     ],

```

`HeartrateComponent.js` there is a function called `handleData()` that gets called when the heart rate data is retrieved from the Web API. The function looks like this:

```
1  const handleData = (data) => {
2    const dataset = getDataset(data, "activities-heart");
3    const valuesArray = dataset((array) => {
4      return array
5        .map((el) => el.value.restingHeartRate)
6        .filter((el) => !isNaN(el));
7    });
8    const avg = getAverage(valuesArray);
```

First, inside the function, the array is saved to a variable `dataset`. Then, the `dataset` has array methods `map()` and `filter()` applied. `map()` method, for each element in the array, selects the `restingHeartRate` value and returns it to the new array containing only that value. The returned array then has a `filter()` method applied. `filter()` method checks if each element inside the array is a number and then returns a new array with elements that matches the expression. The array is saved to `valuesArray` variable and can now be used to calculate the average value.

5.9 Dashboard components

Each dashboard element consists of several nested components. This was done in conjunction with React documentation (React, n.d., f), which states that components should ideally be responsible for doing one thing or otherwise be split up into smaller subcomponents if the component logic grows out of proportion. Firstly, each dashboard item has `gridItemContainer.js`

implemented at the top level. This component handles states such as API state, header state and chart state whose values are later passed down to child components as `props`. These states are shown in the following code snippet.

```
1  const GridItemContainer = ( { baseProps, handleData, handleChartData, children } ) => {
2    const { base_title, base_url, base_value, base_icon, classes } = baseProps;
3    const { access_token } = useContext(AuthenticationContext);
4
5    //API state
6    const [data, loading, error, setFullUrl] = useFetch(access_token, null);
7    //Header state
8    const [title, setTitle] = useState(base_title);
9    const [value, setValue] = useState(base_value);
10   //Chart state
11   const [labels, setLabels] = useState([]);
12   const [dataset, setDataset] = useState([]);
```

`GridItemContainer` renders `gridItemHeader` and `gridItemContent` as its child components. These components don't have any state by themselves, and they just render the UI together with values that they received as props, as it is shown below.

```
1  const GridItemHeader = ({ titletext, value, icon, children }) => {
2    return (
3      <div className="gridItemHeader">
4        <div className="gridHeaderWrapper">
5          <div className="gridHeaderIcon">{icon}</div>
6          <div>
7            <h1>{value}</h1>
8            <h2>{titletext}</h2>
9          </div>
10         </div>
11
12         <div className="dateContainer">{children}</div>
13       </div>
14     );
15   };
```

Each dashboard item then has a specialized component that wraps around `gridItemContainer`. It does not render any additional UI, but has logic that only applies to a specific dashboard item. For example in the following code, `HeartrateComponent.js` handles logic specifically for heart rate data. As this component would not work for the dashboard item that displays sleep info since the sleep data would require to be handled differently.

```
1  const HeartrateComponent = ({ base_url }) => {
2    const base_title = "Average heartrate";
3    const base_value = "-- bpm";
4    const base_icon = <FaHeartbeat className="heartIcon" />;
5    const baseProps = { base_title, base_url, base_value, base_icon };
6
7    const handleData = (data) => {...};
8    const handleChartData = (labels, data) => {...};
9
10   return (
11     <>
12       <GridItemContainer
13         baseProps={baseProps}
14         handleData={handleData}
15         handleChartData={handleChartData}
16       />
17     </>
18   );
19   };
```

Lastly, these specialized components are then placed inside `Grid.js` component, as demonstrated in the code below. `Grid.js` defines a layout and acts as a container for the dashboard items. `Grid.js` uses the CSS Grid layout system, hence the name of the component. CSS Grid lets the layout be composed of rows and columns which can in return make the layout quite flexible and customizable.

```

1  const Grid = () => {
2    const { userId, BASE_URL } = useContext(AuthenticationContext);
3    //api endpoints
4    const apiHeartrate = `${BASE_URL}${userId}/activities/heart/date/`;
5    const apiSleep = `${BASE_URL}${userId}/sleep/date/`;
6    const apiSteps = `${BASE_URL}${userId}/activities/steps/date/`;
7    const apiCalories = `${BASE_URL}${userId}/activities/calories/date/`;
8
9    return (
10     <div className="container">
11       <HeartrateCalendarComponent base_url={apiHeartrate} />
12       <HeartrateComponent base_url={apiHeartrate} />
13       <SleepComponent base_url={apiSleep} />
14       <StepsComponent base_url={apiSteps} />
15       <CaloriesComponent base_url={apiCalories} />
16     </div>
17   );
18 };

```

5.10 Selecting date period

Individual items in the dashboard have the ability to select the date period. Once a user changes the date, the UI reactively updates and at the same time a new API-call is made to retrieve the data with updated date arguments. Inside our app code, we have a `DateSelector.js` component that is responsible for handling date changes. The component can be seen in the code below. This component renders a dropdown menu that allows the user to select between date periods.

```

1  const DateSelector = ({ values, handleOnChange }) => {
2    const [isToggled, setIsToggled] = useState(false);
3    const [selected, setSelected] = useState(values[0]);
4    const [insideDropdown, setInsideDropdown] = useState(false);
5
6    useEffect(() => {
7      //this function gets passed as a prop from GridItemHeader
8      handleOnChange(selected);
9    }, [selected]);

```

First, the component has several states defined. `isToggled` handles the state for opening and closing the dropdown, `selected` is a state for the currently selected date while `insideDropdown` is a state that keeps track if the mouse cursor is inside the dropdown menu. Further, `useEffect` is defined. Once the `selected` variable changes state, a function `handleOnChange()` will get called inside the hook. This function is defined in `GridItemContainer` component but gets called from `DateSelector` as a callback. Function can be seen in the code snippet below. Function takes in the selected date as an argument and then further passes it down to other functions that set the item's title or update the url with new date arguments.

```
1  const handleDateChange = (selectedDate) => {
2    const [startDate, endDate] = getDates(selectedDate);
3    setLabels(enumerateDates(startDate, endDate));
4    setTitle(generateTitleText(base_title, selectedDate));
5    setFullUrl(composeUrl(base_url, startDate, endDate));
6  };
```

This callback function is then passed down to the `DateSelector` component as a prop which can be seen in the following snippet.

```
1  <DateSelector handleOnChange={handleDateChange} />
```

Furthermore, as demonstrated in the code below, there are several functions implemented inside `DateSelector` that handle the logic for closing/opening and for selecting the date.

```

1  const toggleDropdown = () => {
2      setToggled(!isToggled);
3      setInsideDropdown(false);
4  };
5  const handleMouseEnter = () => setInsideDropdown(true);
6  const handleMouseLeave = () => setInsideDropdown(false);
7  const handleClick = (index) => {
8      setToggled(!isToggled);
9      setSelected(values[index]);
10 };
11 const handleBlur = () => {
12     if (!insideDropdown) setToggled(!setToggled);
13 };

```

As illustrated in the snippet below, once the user clicks on the `<input/>` tag, `toggleDropdown()` is called and as a result `isToggled` state is flipped to the opposite of the current value. If `isToggled` equals to `true`, then the `` tag will be rendered. Thus, the dropdown menu will appear on the UI. When the user clicks on the dropdown item, the function `handleClick()` will get executed. Function updates the `selected` state with a value from the dropdown item. Additionally, `` tag listens for when a mouse enters and leaves the dropdown. Therefore, If the user clicks outside the dropdown, the dropdown will close.

```

1  return (
2      <div className="select">
3          <div className="inputWrapper">
4              <input type="text" readOnly className="selectInput" value={selected} onClick={toggleDropdown} onBlur={handleBlur} />
5              <MdKeyboardArrowDown className="arrow" />
6              {isToggled && (
7                  <ul className="dropdown" onMouseEnter={handleMouseEnter} onMouseLeave={handleMouseLeave} >
8                      {values.map((value, index) => (
9                          <li key={index} onClick={() => handleClick(index)}>
10                             {value}
11                         </li>
12                     ))}
13                 </ul>
14             )}
15          </div>
16      </div>
17  );

```

6: User Manual

This user manual covers the base functionality of the React web application. It contains description of the application's functionality as well as instructions on how potential users can utilize the functionality. No prior computer knowledge is required to be able to use this program.

Application URL: <https://fitbit-bach2022.herokuapp.com/>

6.1 Prerequisites

To use this application, it is required that the web browser is present on the user's device. The application will work on all modern-day browsers such as Chrome, Safari, Firefox, Edge, or Opera. For the user to be able to install the app, the user's browser must support PWA functionality. The figure 6.1 shows browsers that support PWAs as of May 2022.

	IE	Edge	Firefox	Chrome	Safari	Opera	Safari on iOS	Opera Mini	Android Browser	Opera Mobile	Chrome for Android	Firefox for Android	UC Browser for Android	Samsung Internet	QQ Browser	Baidu Browser	KaiOS Browser
		12-16	2-75														
		17-18	76-85	4-38			3.2-11.2										
	6-10	79-100	86-99	39-100	3.1-15.3	10-85	1.3-15.3		2.1-4.4.4	12-12.1				4-15.0			
	11	101	100	101	15.4	86	15.4	all	101	64	101	100	12.12	16.0	10.4	7.12	2.5
			101-102	102-104	TP	87											

Figure 6.1: Browsers that support PWAs. Green color indicates full support. Taken from <https://caniuse.com/?search=PWA>.

The application requires that the user has created a Fitbit account. Similarly, for the application to display the data the user must have a Fitbit tracking device.

6.2 Logging in to the system

Upon entering the application, users need to log in with their Fitbit account. This step is necessary, so that the user data collected by the wristband could be provided. After opening the application, the user will be greeted by the login view as shown in the figure 6.2. After entering account details, the user will be prompted to select the scopes that the API will be able to access. Also, by clicking the area marked in red rectangle, as shown in the Figure 6.3, the user can set the duration for how long the app can have access to the Fitbit data. Last step is to click on the “Allow” button.

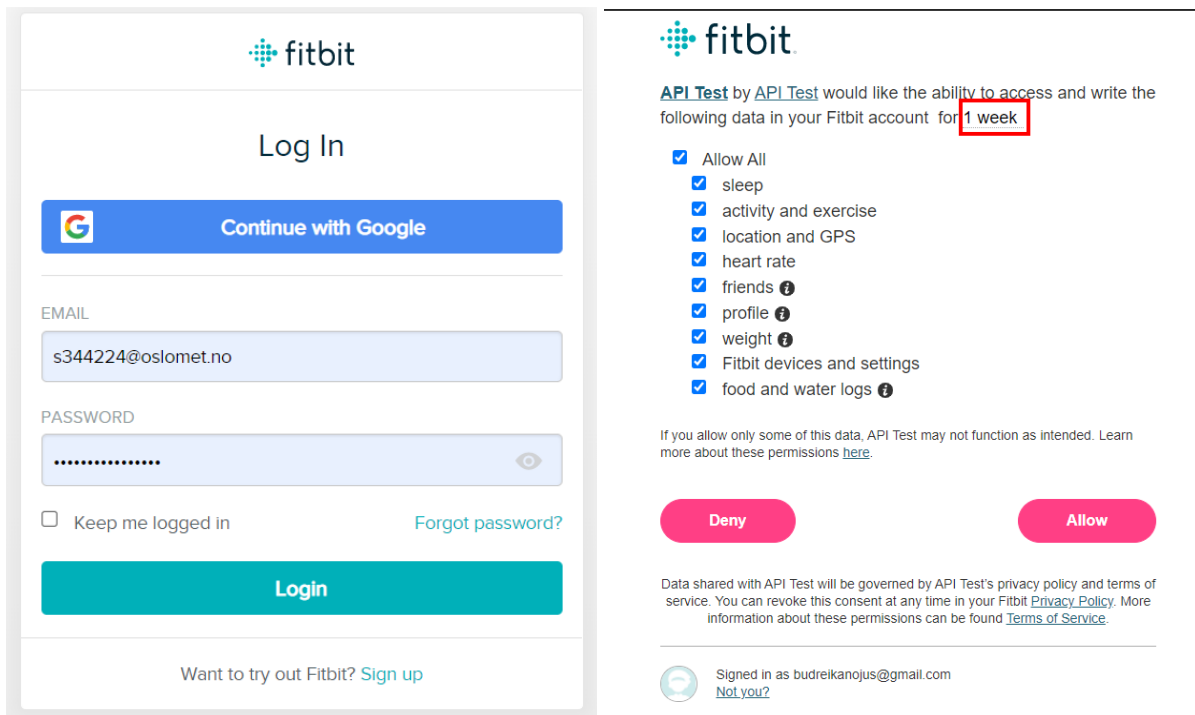


Figure 6.2 & 6.3: User is required to fill out login credentials before proceeding forward. Afterwards the user can allow / decline scopes that the app can access.

To log out of the system, the user has to click “Log out” button located at the top-right of the application.

6.3 User-interface overview

The user-interface primarily contains a Dashboard view. Profile information is also displayed at the top of the screen together with the “logout” button.

The dashboard shows summaries of heart rate, sleep, step count and total calories. Each data scope is contained in its own component, as illustrated in figure 6.4. Each of these components have the following:

1. Header – shows summary for the selected date period.
2. Date selector – let's user select date range / day by clicking the button.
3. Chart – provides more detailed visualization of the measurements for the selected date period.

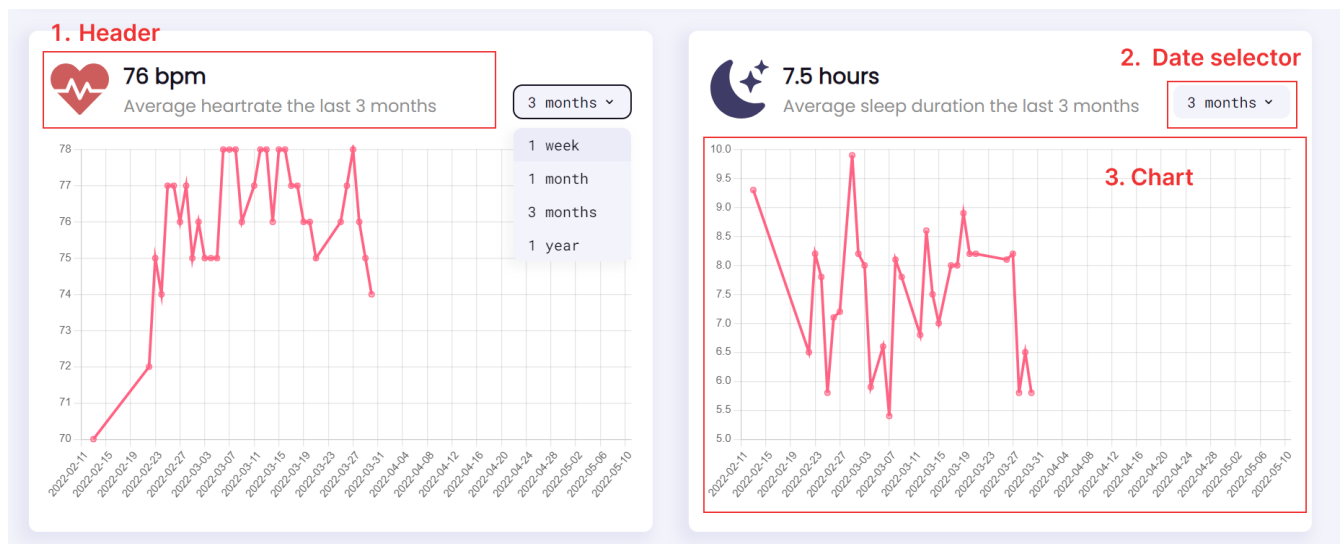


Figure 6.4: Each dashboard component has a header, date selector and a chart.

For example on the heart rate component, the user can choose a date period that the data points will be limited to. Users can choose to display data for the past week, past month, past 3 months or a past year. In the UI this can be achieved by clicking on the dropdown menu, located in the top-right of the dashboard component (Figure 6.5).

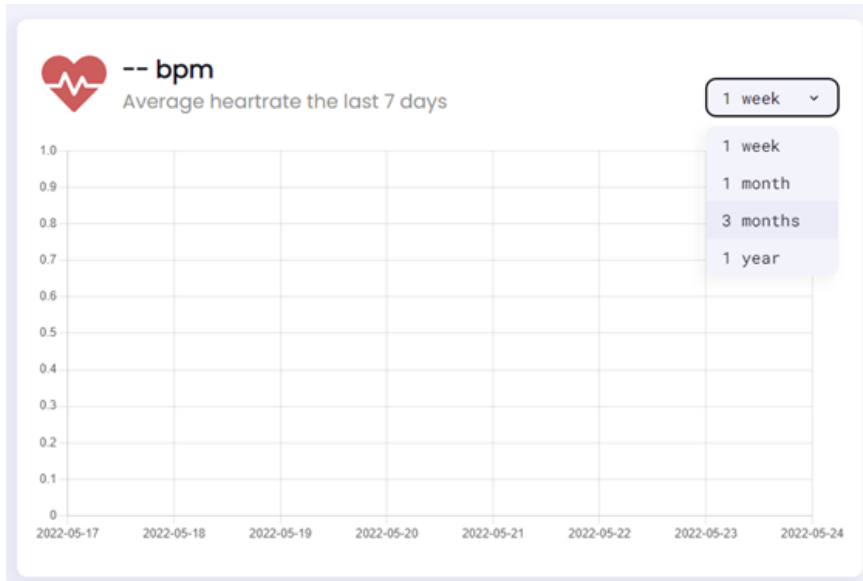


Figure 6.5: Each component inside the dashboard can set a date period.

The UI will then update accordingly to match the date specification, as seen in figure 6.6.

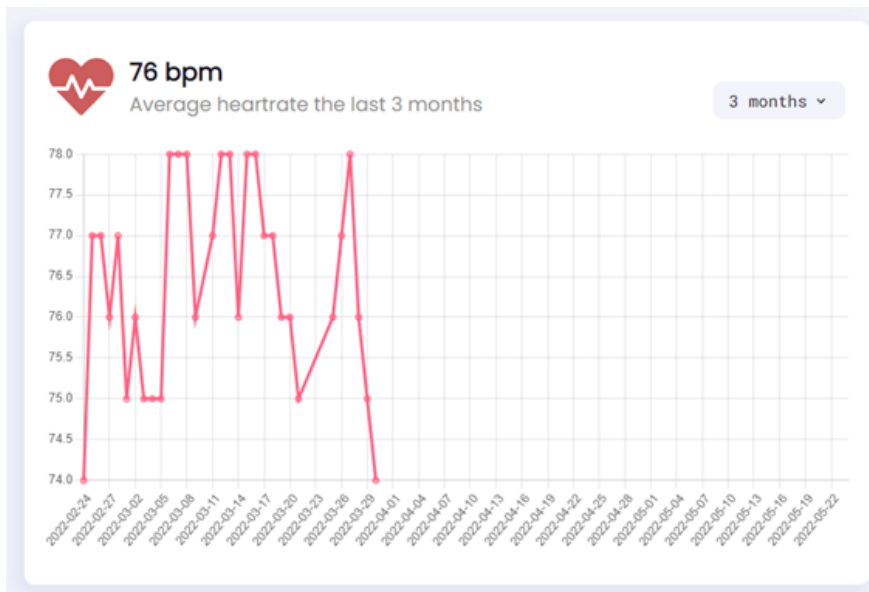


Figure 6.6: The average bpm and chart updates when the user changes date.

Same implementation can be seen in other dashboard components (Figure 6.7 & 6.8).



Figure 6.7: Component that displays sleep duration.

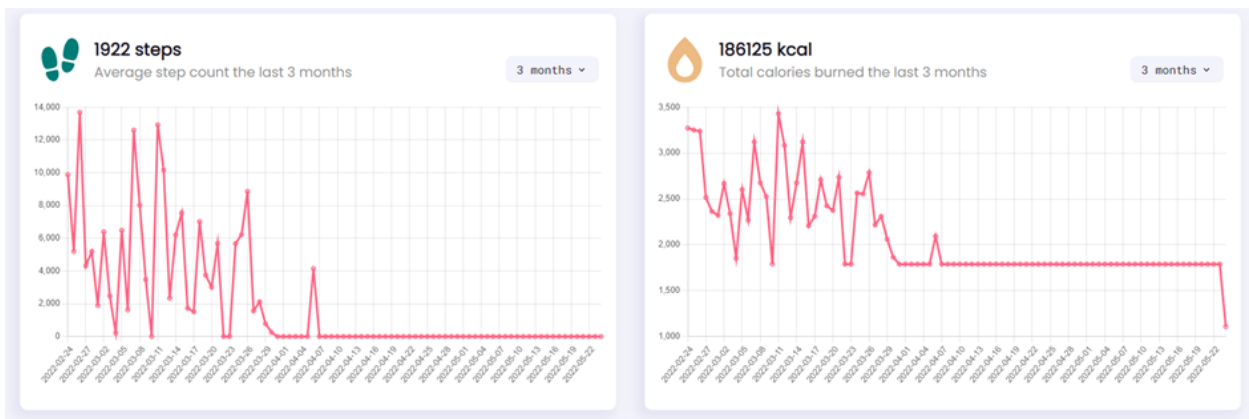


Figure 6.8: Step counter and calories-burned components.

6.4 Add to home screen

If the browser supports progressive web app functionality, the app should be easily installable. For example, there are two ways to install the application in the desktop version of Chrome browser. The first method is to install it by clicking on the “Install icon” located in the url bar. The second method is to install it by clicking on the three-dot menu as shown below (Figure 6.9). For mobile devices that use Chrome browsers, the similar process applies (Figure 6.10). After installation, the application should appear on the device's home screen. It can be assumed that the installation process is similar on other web browsers.

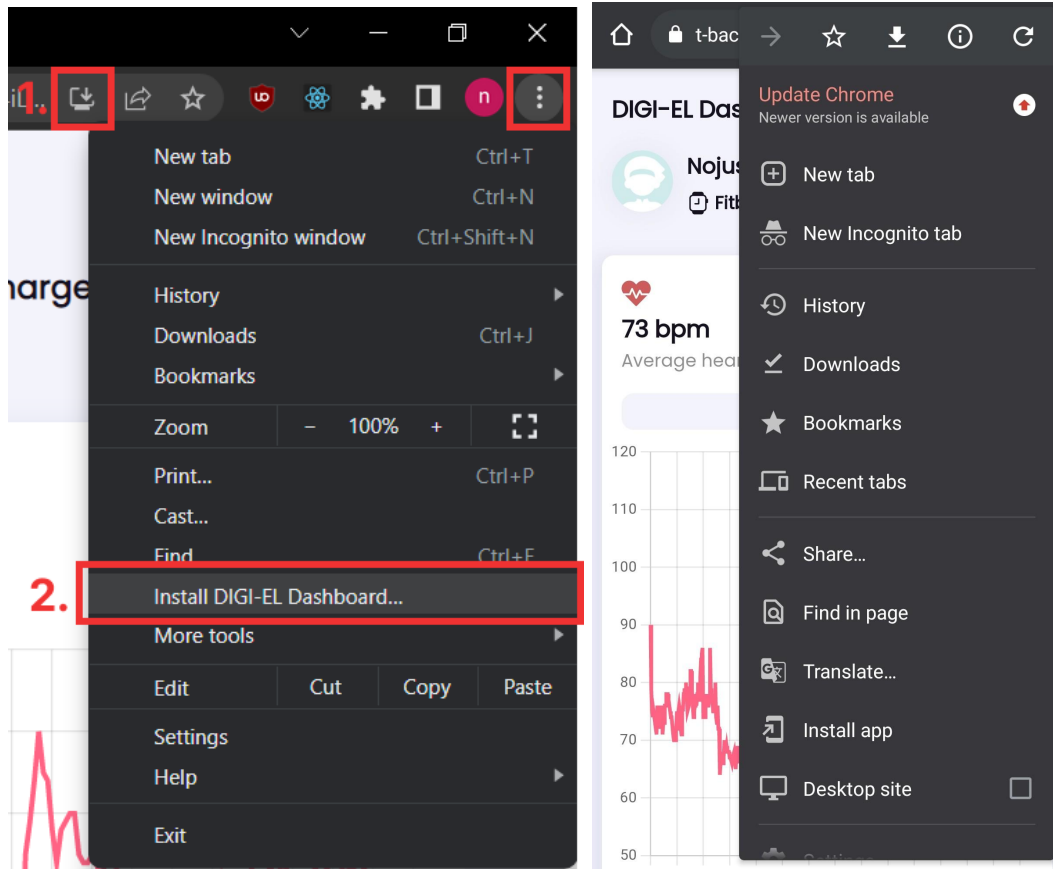


Figure 6.9 & 6.10: Shows installation of the application for the desktop and mobile devices accordingly.

7. Fitbit, GDPR and Privacy

7.1 Background

We live in the age of information technology where a high amount of data is transported through the internet like we have never seen before, there is even a word for it, Big data. A lot of the data is sensitive personal data that can easily be misused. Therefore privacy and protection of data has never been more important than it is today. Fitbit is an electronic device that requires personal data from the consumer. It's part of this project's objective to assess the privacy terms and data security of Fitbit. We also want to see if what said in the privacy terms is inline with real experience. Unfortunately it's not in our capacity to test all kinds of scenarios when it comes to this subject due to time constraints and limited resources. Therefore our findings on the privacy matter are mainly based on research papers done by others and company policy such as the Fitbit privacy policy and rights of users in Europe according to GDPR. A large portion of our findings are based on papers (Hilts et al., 2016: Every Step You Fake: A Comparative Analysis of Fitness Tracker Privacy and Security). This is one of the most comprehensive litteratures we found on this topic. And although the literature is from 2016, many of the concerns still hold true.

7.2 The implications

In recent years we have all heard about tech companies leaking personal data to third parties without the consent of its users or hackers getting hold of data in unlawful ways. With wearable fitness devices the implications of such an event are severe, as information collected by these devices are not only personal data like e-mail, but one could also draw a conclusion on a person's fitness or health based on this data. For example, the Fitbit wearables can also record time and

location of an activity. Such data could be then used in a criminal court case as evidence. Also insurance companies could use it to set premium prices based on a person's fitness or a user may manipulate this data in order to affect an outcome. Therefore it is important that users are aware of these implications, their rights and how the data is used.

7.3 A look at the GDPR

The "General Data Protection Regulation" or GDPR as it is commonly referred to is a detailed regulation implemented by the European Union in 2018. The goal of the GDPR is to protect the personal data of EU citizens across member states and simplify the process to request and supervise how your personal data is processed and distributed so you could meaningfully respond to the company if there is a violation of your privacy rights. Personal data is any information that can identify a person directly or indirectly for example name, an identification number, location data, an online identifier and more.

Breach of these regulations can lead to high fines, up to €20 million or up to 4% - whichever is higher - in the worldwide annual revenue of the entity controlling the data, this is stated in Art. 83(5) GDPR.

The regulation has 7 important principles:

- "Lawfulness, fairness and transparency", "Purpose limitation"
- "Data minimisation"
- "Accuracy"
- "Storage limitation"
- "Integrity and confidentiality (security)"
- "Accountability"

In brief, these principles ensure that personal data are collected in a lawful way, with the consent of the consumer and not used or processed in another form than its intended purpose. It limits data collection, in that no entity shall collect any more personal data than is necessary for an explicit functionality of the application. Data shall be accurate and up to date at all times. Personal data shall be identifiable only for a necessary time period. And the data controller or collector shall be accountable and be able to show compliance with the regulation.

7.4 Fitbit privacy policy and Terms of use

The privacy policy and Term of use can be found on the Fitbit website, ([Fitbit Legal: Privacy Policy](#)) and (<https://www.fitbit.com/global/us/legal/terms-of-service>) respectively. These documents outline the requirements to use a wristband and the policy of the company.

According to both the Privacy policy and Terms of use the Fitbit needs personal data like name, email and in some cases phone numbers in order to offer you the service. The privacy policy says it requires this information in order to create a user account and populate the dashboard with relevant data and do research to develop new services. If the user grants access, it can also collect and store the location an activity has taken place in.

Furthermore, Fitbit can store additional personal data if one connects the application with social media or an email such as Gmail.

According to the company, the data will be used to personalize and improve their service, furthermore data collected is used for data analysis and research says

FitBit in its privacy policy. Data like height, weight, gender is used to calculate calories burned and keep track of progress. The policy also says it does not share Personal data except for “de-identified” non-personal information with third parties. And users have the freedom to change their personal data as they see fit to reflect the privacy laws of the country they live in. Moreover, data will be stored as long as the account is active according to the policy. Consumers should be aware that Fitbit in their Term of Use claims the right to use and share any information given to them.

In its privacy policy the company says it complies with the GDPR regulation by asking for consent when a user registers with their product and every time the user opts to add additional functionality or service. It also gives users the right to control their data , withdraw consent any time or stop data from being processed as per GDPR regulation.

In comparing GDPR and Fitbit privacy policy, we find that Fitbit seems to agree with what GDPR considered to be personal data.

7.5 Data

Wearable fitness devices such as FitBit use bluetooth to communicate between the app on the phone and the wristband. The goal of the application, amongst others, is to send the data collected from the wristband to the Fitbit servers. When the wristband device uses bluetooth to connect to the mobile, it must make itself discoverable, by “advertising” through bluetooth technology. This can potentially expose data and mac addresses to a “man in the middle ” attack. An exposed mac address can for example be tracked by a third party to locate the device, meaning for example some one or a company could track your

movement. The research paper by Open effect, did an extensive testing to address these threats. The tests captured and read the data transmitted via bluetooth and the data sent from the mobile app to the Fitbit servers. It was confirmed that the FitBit app sends general personal data like name, height, e-mail and reproductive health info. But also phone serial number, IMEI number and Bluetooth Media Access Control (MAC), which is the identifier of the wristband. The mobile app sends this information for every activity that's completed, it acts only as an intermediary, as the data itself is not stored in the app but fetched from the servers every time. This process is called synchronization.

This personal data was transferred securely to the Fitbit server. Since data transferred from the mobile application to the cloud was secured with the HTTPS protocol and data was encrypted as tests done by *Open effect* has shown in their article, (Hilts et al., 2016, p.33: Every Step You Fake: A Comparative Analysis of Fitness Tracker Privacy and Security).

As we mentioned earlier all data from the wristband is sent via bluetooth to the mobile app. Fitbit states on their website the wristband uses Bluetooth Low Energy (BLE) technology to sync the data. This version of bluetooth technology has less security. There have been concerns previously regarding this bluetooth technology. For example, if a user turns off the bluetooth on their phone the wristband will continue to advertise the mac address and data via Bluetooth. With BLE the mac address of the wristband is static and which would allow a malicious attacker to connect to the and that way track the user. But this has been addressed and fixed by Fitbit, (Hilts et al., 2016, p.36: Every Step You Fake: A Comparative Analysis of Fitness Tracker Privacy and Security). The newer wristband can randomize the mac address now and data on the wristband is encrypted.

7.6 Discussion

Our findings based on the Fitbit documentation and other literature regarding fitbit privacy have been positive for the most part. General security and privacy on the devices are good. They are also inline with the GDPR regulations. Fitbit employs HTTPS for security and AES encryption to encrypt data. Fitbit has learned from the past and updated most of their code to improve security.

One potential security threat is that Fitbit still uses the outdated Bluetooth technology (Bluetooth low energy) version in their new devices instead of Bluetooth 4.0 implementation. This technology is prone to man-in-the-middle attack. Data between the wristband and mobile app can be sniffed and captured, however this is only the fitness data and not other personal data like name or e-mail.

8. Experiments and Methodology

The testing of the Fitbit Charge 4 was built on two primary inspirations: the requirements directly outlined by Karlstad University and the tests that had been conducted by the previous groups. As we were testing the same attributes as the previous group for the same purposes we saw it fitting to base our own tests on what worked for the previous group.

8.1 Technical specifications

	Fitbit Charge 4	Fitbit Charge 5
Date released	March, 2020	September, 2021
Display	Grayscale OLED touchscreen	Colored AMOLED touchscreen
Screen size	1.57 inches	1.04 inches
Battery Life	Up to 7 days	Up to 7 days
Weight	27 grams	28 grams
Sensors	Optical heart rate monitor, gps, accelerometer, oxygen saturation (SpO2)*, device temperature sensor	Optical heart rate monitor, gps, accelerometer, oxygen saturation (SpO2)*, device temperature sensor, ambient light sensor, ECG sensor**, EDA stress sensor
Water resistance	Up to 50m in depth	Up to 50m in depth

* Not available in all markets

** Only available in select countries

Specifications obtained from:

<https://www.fitbit.com/global/fi/products/trackers/charge4>

<https://www.fitbit.com/global/us/products/trackers/charge5>

8.2 Heart-rate Monitor

The accuracy of the sensors within the activity-wristband was of interest to Karlstad and therefore the bulk of our physical tests were focused around measuring the accuracy of the Fitbit's heart-rate monitor feature. In order to measure the accuracy of this feature we were in need of a baseline to compare the measurements of the Fitbit to. Thanks to Professor Gjøvaag at Oslomet we were allowed to borrow the Polar H7. The Polar H7 is a specialized tool for measuring heart-rate which could export data much like the Fitbit could. This made it a viable tool for checking the accuracy of the Fitbit because we could directly compare the measurements between the two devices on a second-by-second basis.

Our tests involved three different levels of physical activity.

These levels were resting, walking and stairs. The previous group had used resting, walking and an exercise bike but we did not have access to an exercise bike during our testing so we emulated the same level of physicality using stairs.

Resting - The subject would be physically still, keeping their heart rate low. We measured the resting heart-rate for 3 minutes giving us 60 data points per test.

Walking - The subject would walk at a leisurely pace outside to emulate a medium level of physical activity. We measured the walking heart-rate for 30 minutes giving us 600 data points per test.

Stairs - The subject would jog up and down stairs a set amount of times to emulate a high level of physical activity and a high heart-rate. This test was built on how long it took to complete 5 sets up and down the stairs and not a set time, so we measured an average 1:33 per test which is 31 data points per test.

For each test we would test the Fitbit on:

- Both wrists
- Two levels of tightness (loose and tight)
- Two positions, on top of the wrist and directly facing the pulse

Each test would effectively be done six times. The activity would be done for a set time frame or a set amount of rounds while the subject wore a Fitbit on one wrist and a Polar H7 wrapped around their chest with the sensor in the middle of the chest. The position of the H7 is arbitrary, but we wanted an easy position to replicate visually over several tests in order to keep things consistent.

Our heart rate measuring tests involved a second-to-second tracking of the heart rate using the Polar H7 and the Fitbit Charge 4. The Polar H7 recorded the heart rate at every second but the Fitbit had a less consistent interval. To keep our comparison accurate we compared the Polar H7 only on the points that were recorded by the Fitbit, which causes the one to three second intervals in our data sets. Implications of this are discussed in a later section.

8.3 Sleep

Professional sleep measuring and analysis is beyond the reach of our project as there are no consumer products that can match the efficacy of sleep research tools used by somnologists. To compensate for this we used consumer sleep-evaluation apps available on the Google Play market to compare feedback from the app with feedback from the Fitbit. The app we chose for this purpose was Sleep for Android. (Nálevka P, (2016), *Sleep as Android: Smart alarm*)

This test was done by putting on the fitbit before going to bed. The data was collected from the fitbit app such as hours slept, the resting bpm and duration of the sleep. The app also provides the person with information such as how many times and how long the person was restless and got awake.

Due to miscommunications within the group we were not able to make everyone who participated in the sleep-testing use the same application to measure the fitbit data against. Therefore we can not draw any conclusions from this part of the test. However, an accurate assessment of this was already considered a far shot due to the lack of standardization and research techniques available without a lab.

8.4 Water resistance

To test water resistance we wore the activity-wristband while doing various water-related activities, such as being in the shower, doing the dishes and washing hands. We did not see the need to do extensive testing as this was a binary answer as to whether or not the activity wristband worked normally after contact with water.

8.5 Results

As our data is not normally distributed because it is a comparison between two measures, a normal t-test and analysis of results will not help us measure how much the Fitbit's data deviates from the measurements by the Polar H7. Our alternative hypothesis would be proven easily and we'd gain no further insight into the matter. Instead we will use a Tukey Plot, also known as a Bland-Altman plot, to display the degree of deviation between the Fitbit and the Polar H7.

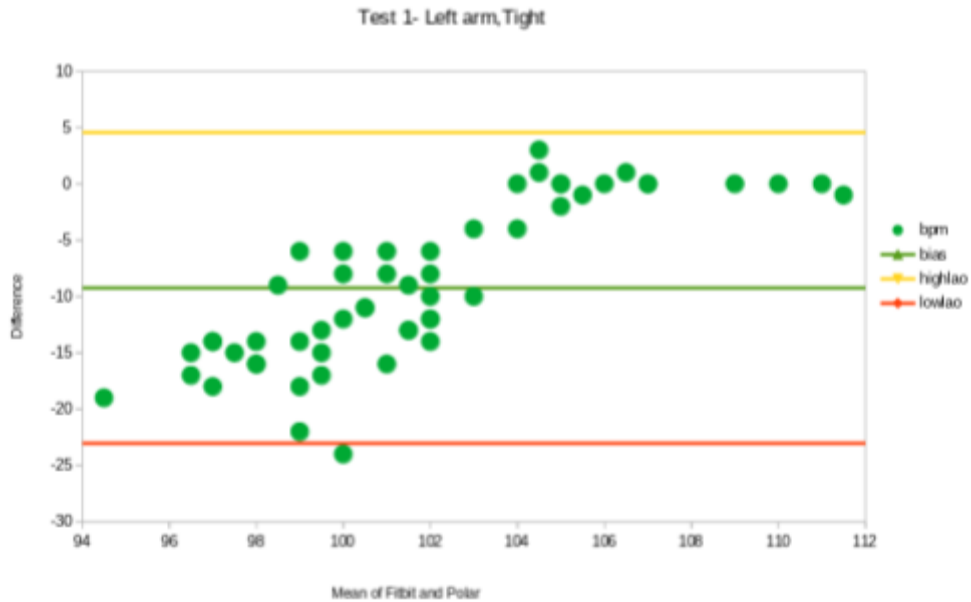


Fig 8.1 - Test 1, Left arm, Tight strap

Let us take a moment to understand the format of the data we will be using. The Bland-Altman plot is used to compare data derived from measurements. Each point represents a differential value on the Y axis, which is the difference in measurements between the Fitbit Charge 4 and the Polar H7. These points are plotted along the X axis, which represents the value of the mean of each data-point. As explained by Giavarina D.: “(...)the difference of the two paired measurements is plotted against the mean of the two measurements“ (G.D, 2015). The mean is the average value between the BPM measured by the Fitbit and the Polar. The bias is the average of the difference between every point,

allowing us to spot inconsistencies in more volatile data whose mean does not float around 0. Highlao and lowlao stand for High and Low Limit of Agreement respectively and values that fall outside of these borders are more than 1.96s (Standard deviation) from the mean difference, implying the devices deviate from each-other significantly. High/Low limits of agreement are not important to these tests as we're testing to what degree the Fitbit matches the readings of the Polar H7, however it's still useful to be able to spot data which would indicate a massive disagreement between our measurements.

8.5.1 Heart-Rate: Resting

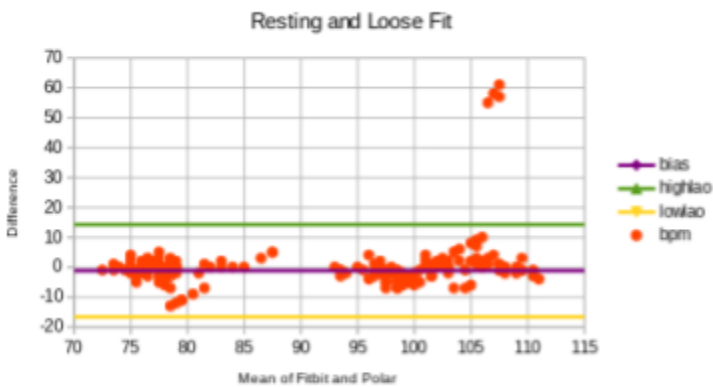


Fig 8.2 - Resting and Loose fit

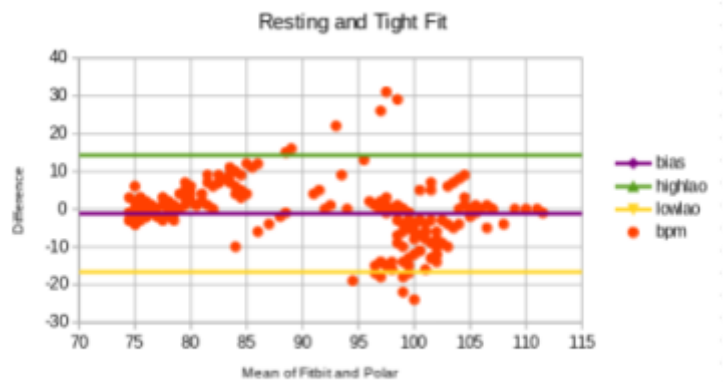


Fig 8.3 - Resting and Tight Fit

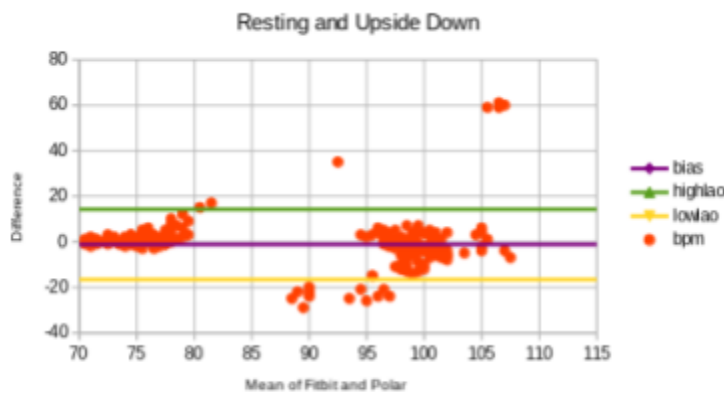


Fig 8.4 - Resting and Upside Down

The impression we get from the resting tests is that in low-BPM circumstances the measurements between the Fitbit Charge 4 and the Polar H7 stay mostly consistent with most measures falling within the limit of agreement. However, we can also see that there are consistent outliers indicating the Fitbit Charge 4 frequently makes at least one handful of deviations for each test.

8.5.2 Heart-Rate: Stairs

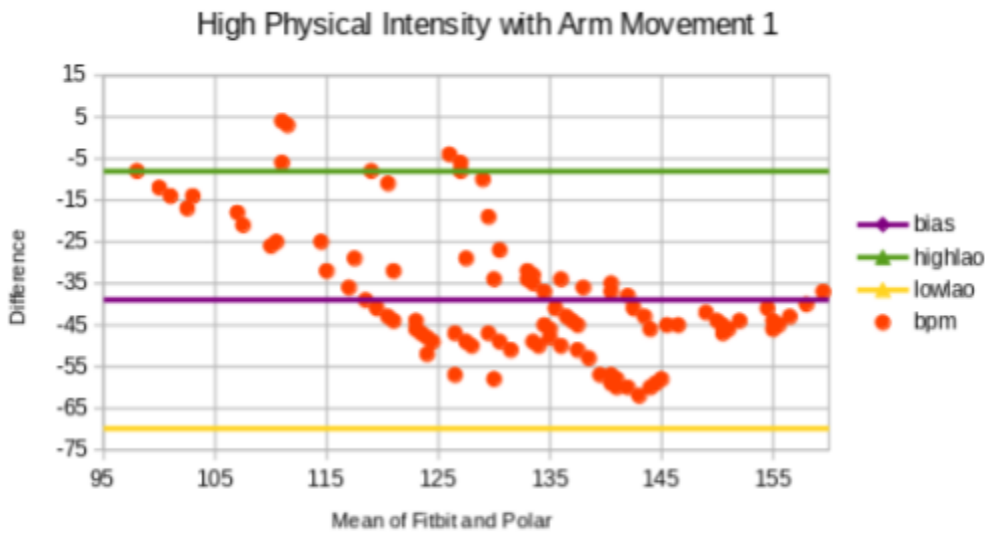


Fig 8.5 High physical intensity and arm movement

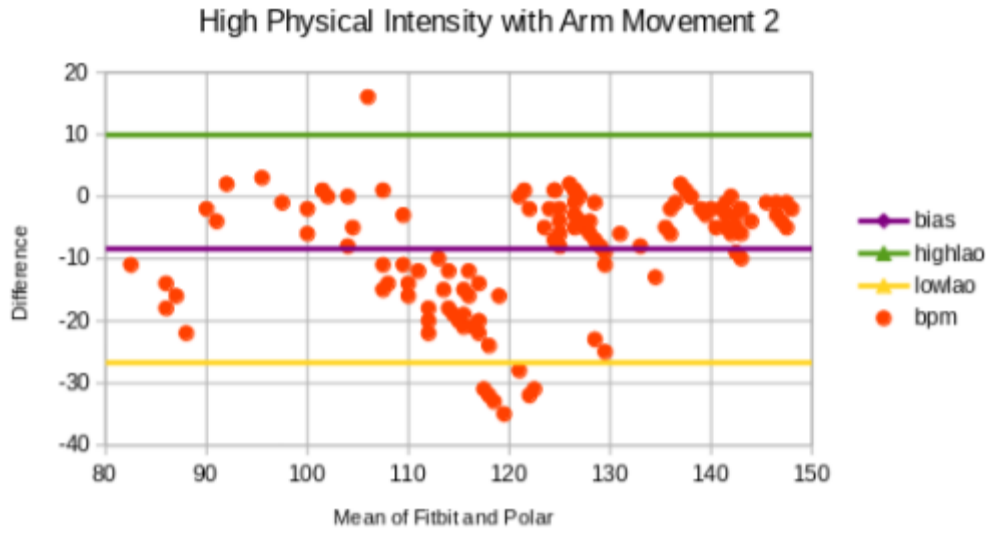


Fig 8.6 - High Physical Intensity with Arm Movement 2

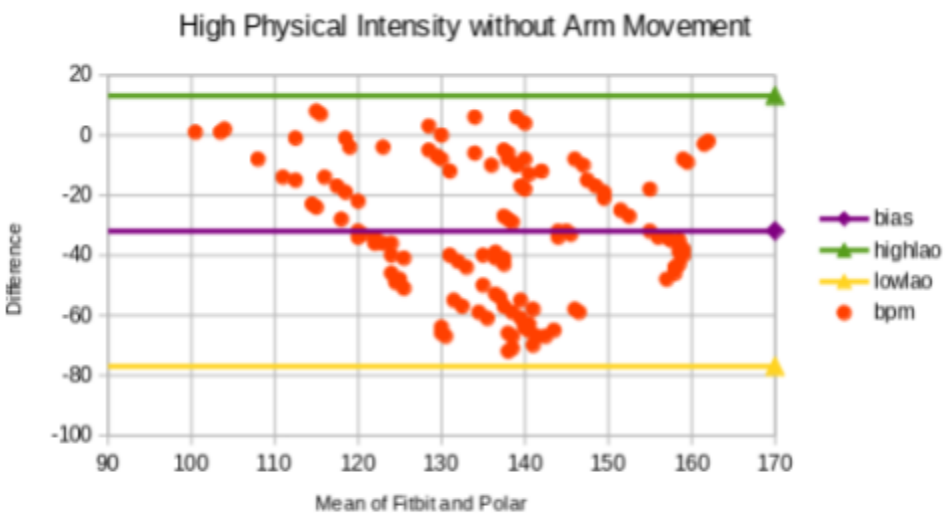
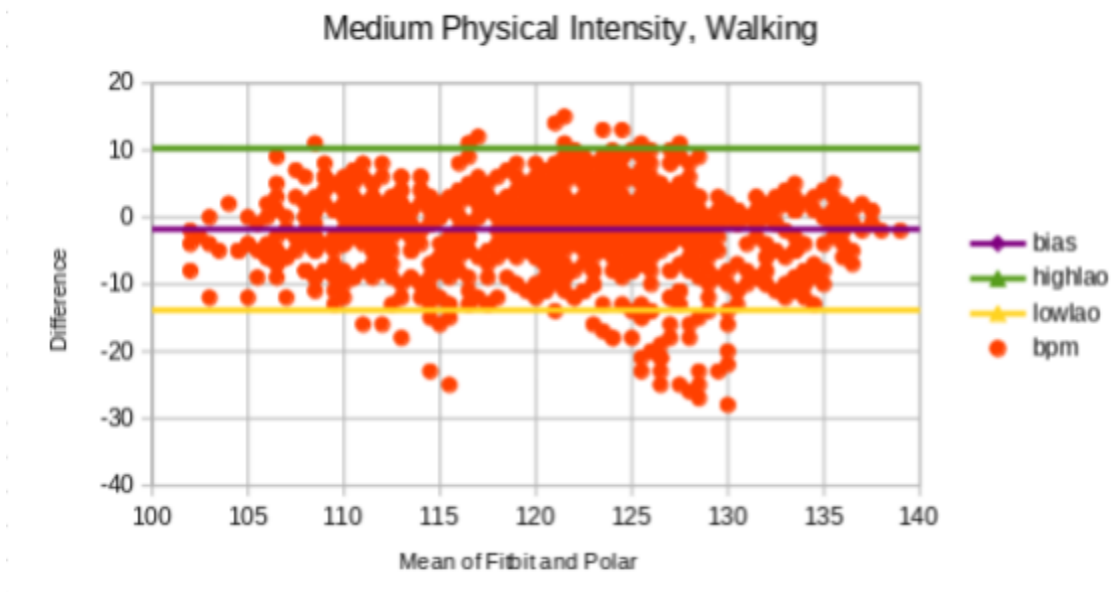


Fig 8.7 - High Physical Intensity, No Arm Movement

When a high level of physical intensity was used to test the heart-monitor we see a much more scattered and uneven result. While our data stays within the limit of agreement we can see that a majority of the data points go from barely within the

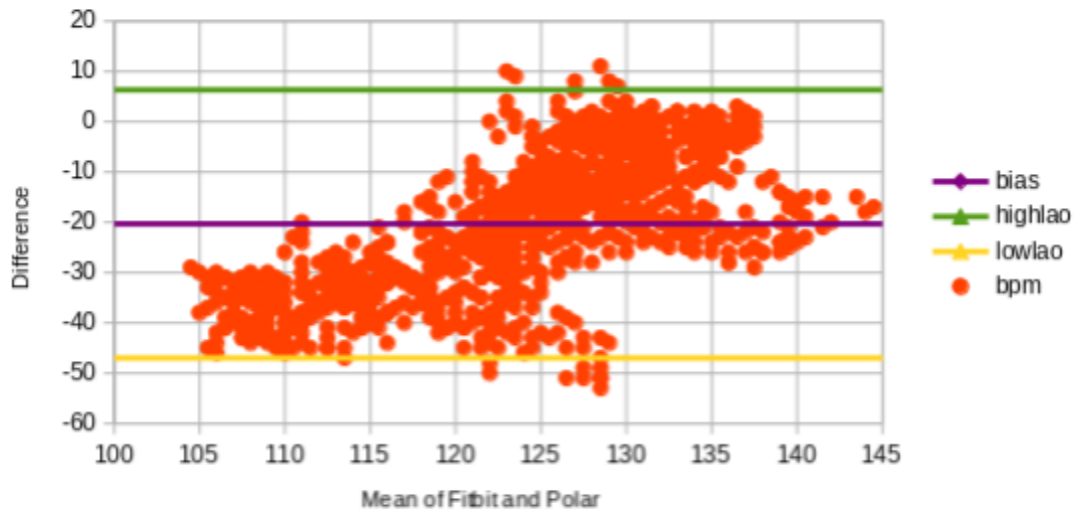
range of 0 to -80. In two of the tests we see the bias drawing itself around -35, meaning the average of the difference between the Polar H7 and the Fitbit lies at -35. If the purpose of the Fitbit was to give accurate assessments of the subjects pulse during tests that involve a high level of physical movement this data could be used to make a strong argument against the Fitbit. However, in the context of health-research around pregnant people we have to keep in mind that the subjects of DIGI-EL's research do not see frequent bursts of physical activity due to the taxing nature of pregnancy. Therefore the inaccuracies in this data may not have as much implication as it may seem at first.

8.5.3 Heart-Rate: Walking



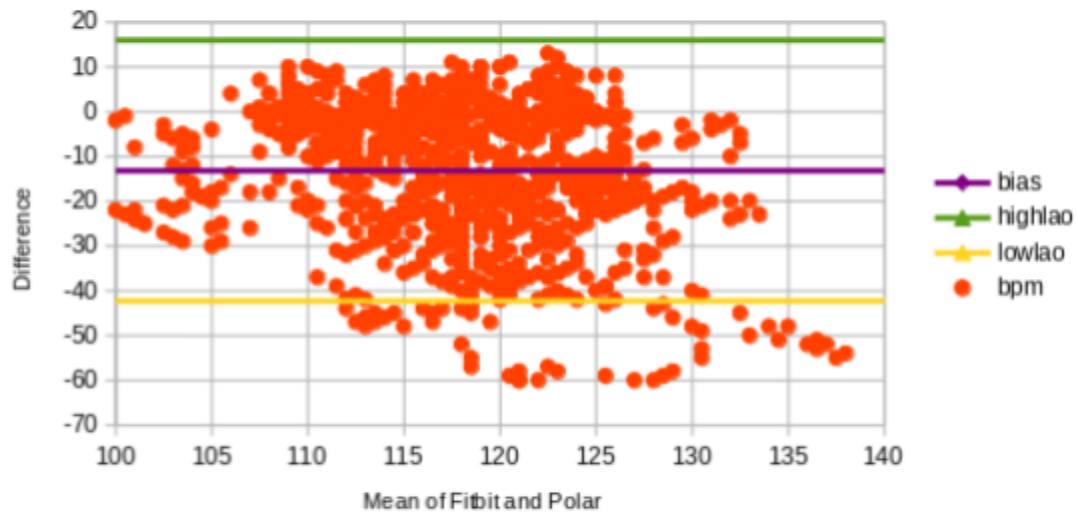
8.8 - Walking, Set 1

Medium Physical Intensity, Walking 2

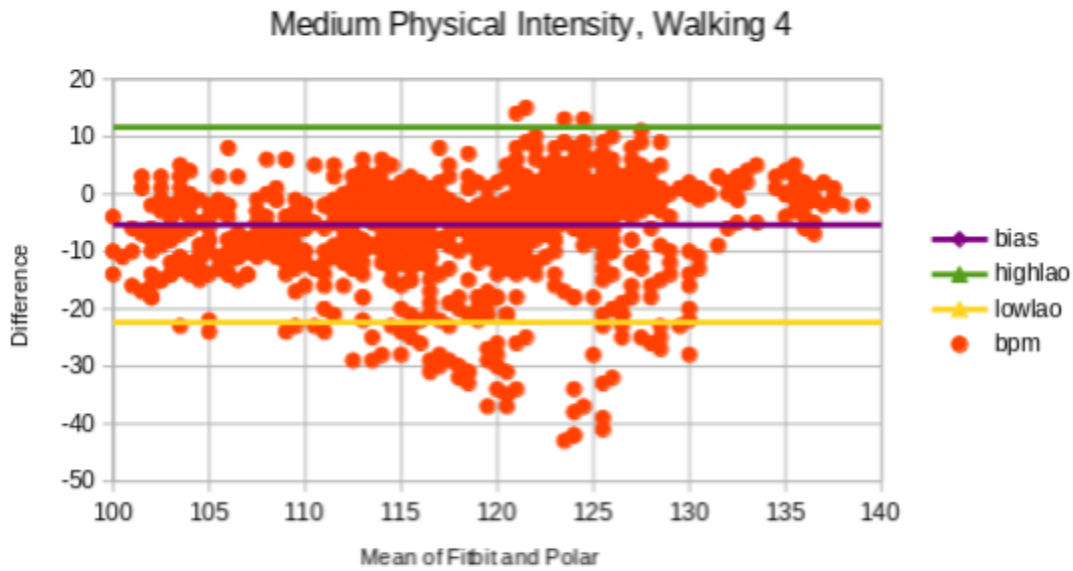


8.9 - Walking, Set 2

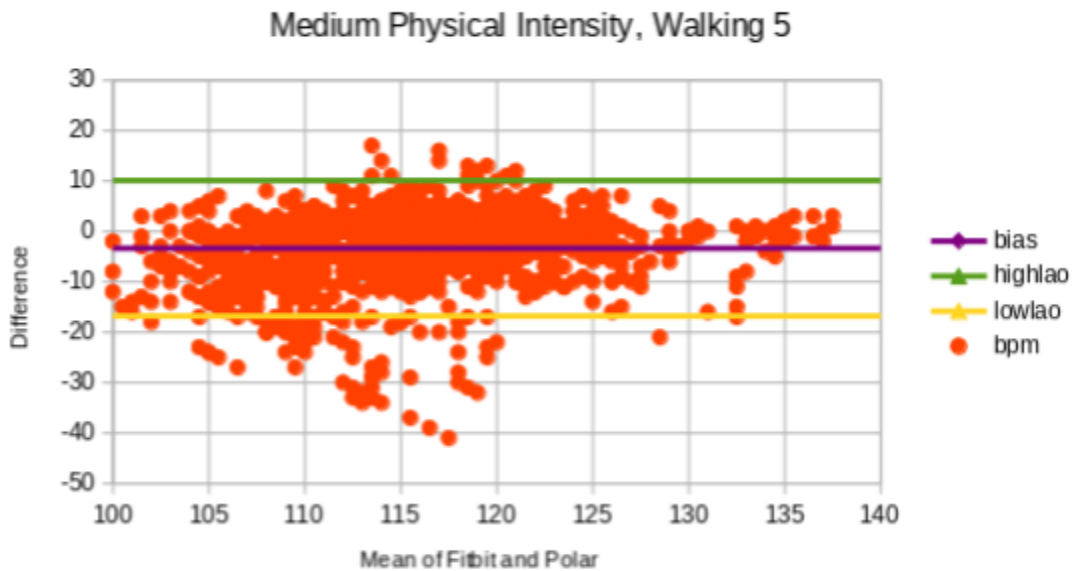
Medium Physical Intensity, Walking 3



8.10 - Walking, Set 3



8.11 - Walking, Set 4



8.12 - Walking, Set 5

In the tests involving a level of physical activity that is more similar to the levels a hypothetical pregnant person would encounter day-to-day with larger amounts of data points we can discern that the Fitbit has an issue with consistency. In some

tests the bias is fairly close to 0 which indicates a low difference between measurements, but even here we get sets like 8.9 and 8.10 where the average difference is greater than -10 indicating measurements that drift apart frequently on scales that disrupt the ability for accurate measurements to be made.

8.5.4 Heart-Rate: Discussion

The tests consistently show a concerning amount of difference between the measurements between the two devices. The severity of this disparity depends on the need for accuracy, which would be determined by the conditions of future experiments. While we would not recommend using the Fitbit Charge 4 for tests that require very precise and consistent measurements it can still be useful in research that interprets the data more comparatively. However this would only be feasible in a testing environment with little to no physical strain as physical activity introduces inconsistencies that pollute the data and makes comparative analysis difficult.

8.5.5 Sleep: Findings and Discussion

Due to the aforementioned communication errors the results are limited as only the tests where we know for sure the appropriate application was used can be used to evaluate the fitbit.

Sleep test 1			Sleep test 2		
	App	Fitbit		App	Fitbit
Duration (h)	08:23	07:50	Duration	07:21	07:14
Deep Sleep (%)	45 %	16.6 %	Deep Sleep (%)	46 %	12.90%
Deep Sleep (h)	03:44	01:18	Deep Sleep (h)	03:24	00:56
Time awake (h)	00:00	01:12	Time awake (h)	00:09	00:39
Sleep start	22:28	23:02	Sleep start	22:57	23:14
Sleep end	06:52	06:52	Sleep end	06:27	06:28
Resting heart rate (bpm)		67	Resting heart rate (bpm)		70

Fig 8.13 - Sleep tests 1 and 2

Sleep test 3	App	Fitbit		Sleep test 4	App	Fitbit
Duration	05:12	06:28		Duration	07:39	09:57
Deep Sleep (%)	51 %	10.80%		Deep Sleep (%)	44 %	11.70%
Deep Sleep (h)	02:39	00:42		Deep Sleep (h)	03:21	01:10
Time awake (h)	00:00	00:42		Time awake (h)	00:11	01:20
Sleep start	01:03	01:15		Sleep start	00:56	01:06
Sleep end	06:15	07:43		Sleep end	08:47	11:03
Resting heart rate (bpm)		68		Resting heart rate (bpm)		69

Fig 8.14 - Sleep test 3 and 4

As we can see from Fig 8.13 and 8.14 the app and the Fitbit seem to be in disagreement in most areas. According to the Sleep Foundation (Pacheco, D., 2022) deep sleep should make up 13%-23% of your sleep at night, which means in this regard the Fitbit was much more accurate than the app.

8.5.6 Water Resistance: Findings and Discussion

According to the specification the armband is 5atm certified. 5atm means that a device is able to withstand pressure that is equal to a depth of 50 meters or 5 bars. This makes the tracker suitable for activities such as washing dishes, showering, and swimming (Garmin, n.d.).

During the assessment, the tracker was being worn by our group in different conditions. Tracker continued to function as expected when worn in rain, when taking a shower or when washing dishes. The team did not notice any changes to the functioning of the tracker. With that being said, according to Fitbit, it is recommended to take off the armband completely when taking a shower. As not doing so, over a period of time could increase the risk of damage to the tracker due to the chemical exposure of soaps and shampoos. However, it is safe to conclude that the tracker can be worn in circumstances that may get the armband wet. This would therefore be suitable for pregnant women, as there are

many daily activities such as washing hands or dishes that are commonly done during the period of pregnancy.

8.5.7 Battery life: Findings and Discussion

Fitbit claims Charge 4 will last up to 7 days with normal use or up to 5 hours with continuous GPS use. Depending on various settings and configurations of the tracker the battery life can vary between users. Fitbit does not disclose battery capacity used in Charge 4 in the product [page](#).

To assess the life of the battery the tracker was used continuously for the duration that took the tracker to deplete its battery capacity from full charge. The battery assessment was simulated to represent the normal usage of the armband. Tracker remained on the wrist throughout the assessment and was not taken off. In this period the tracker was primarily monitoring heart rate, step count and sleep. Additionally, on some days the armband was logging 30 minute activity/exercise which caused GPS to turn on by default. The brightness of the display was set to Auto, meaning it would adjust brightness accordingly to the light conditions.

With this configuration the Charge 4 managed to last around 5 days and 10 hours. It can be assumed that the armband could last closer to 7 days if the GPS had not been used at all. It took around 2 hours to recharge the battery from 0 to 100% which matched the Fitbit's specification. Overall, the life of the battery was observed to be sufficient for normal use and not an area of concern. Having to recharge the device every 5-6 days for 2 hours doesn't seem to be that inconvenient, although Fitbit recommends recharging every few days to ensure the armband is always tracking.

8.5.8 Wristband fit & comfort: Findings and Discussion

According to product specification of [Charge 4](#), the tracker comes with two wristband sizes. Small band fits the wrist from 140mm to 180mm in circumference. Large band fits the wrist from 180mm to 220mm in circumference. Bands are made of flexible rubber-like material and are detachable. Fitbit recommends wearing an armband loosely enough so that it can move back and forth the wrist.

Some of the group members observed that device could be difficult to put on by yourself. Furthermore, it was also observed that the band was a little irritating after prolonged use. According to Fitbit if a user is experiencing irritation or redness, the wristband should be loosened, or the device should be taken off completely until the irritation disappears (Fitbit, n.d.). Furthermore, they recommend removing the device regularly for about an hour if the user is wearing the device for extended periods (Fitbit, n.d.). We observed that irritation was considerably reduced when the band was loosened up.

If the irritation persists however, the problem could potentially be solved by swapping to different material bands that Fitbit sells separately. Leather or woven bands that Fitbit sells as separate accessories however are not water resistant neither are they intended for high-intensity exercise. Another negative is that these wristbands can cost from \$35 to \$50. This makes it a not cost-effective solution. Overall, despite some of the group members experiencing discomfort from the wristband, we observed that discomfort can be managed by loosening the strap or taking the device off completely. Taking breaks from the device matched well with battery life, that is every 5 days when the tracker needs charging. Therefore, the band should be suitable for pregnant women in most

cases. In cases where irritation does not go away, accessory wristbands in different materials could be considered.

9. Conclusion

In this project we have tested the technical capacities and accuracy of the Fitbit Charge 4 through a series of experiments in the context of DIGI-EL's research on digital tools during labor. We have also created a React application that demonstrates how data can be fetched from the Fitbit API and plotted on charts.

From our research we found that the Fitbit Charge 4's heart-rate monitor struggles with inaccuracies and random spikes polluting research data, excluding certain forms of analysis to be performed with it. If the interest of the research is to identify irregularities under high physical activity or other tasks that would require a high degree of consistency and precision the Fitbit would not be a viable option due to its frequent misreadings. The Fitbit is much more suited for gathering data used in comparative analysis in research that does not anticipate any high levels of physical activity in the test-subjects. This applies in the context of pregnancy and therefore we can approve of the Fitbit's use as long as these inaccuracies aren't detrimental to the research being done.

From our testing of the Fitbit's physical traits we found that it is waterproof in every context a pregnant person would be wearing a Fitbit. In this part it lives up to all expectations and requirements.

The Sleep-functionality of the Fitbit provided accurate data on the sleep of our test-subjects, but the limited information this data contains makes it difficult to build any conclusions off it alone.

The Fitbit has rather sharp edges and we would not recommend handling a child while wearing it as the concerns are sharp enough to damage the skin of a newborn.

9.1 Future Work

Our evaluation of the Fitbit Charge 4 only touches on aspects of the wristband that were available for us to test within our capacities as students. Whether this research is partially or fully applicable within future studies run by DIGI-EL depends on the conditions of those experiments. There are many areas that could benefit from reexamination and re-testing. The sleep testing would benefit from a more in-depth study with more variables to compare the application and Fitbit between. Future studies on the accuracy of the heart-rate monitor could include a wider variety of physical activities more directly related to the day-to-day life of a pregnant woman.

The application we've developed did not accomplish some of the later requirements that we've acquired from our client, which were defined in [Chapter 4.2](#). Because of that, the future work on the app should revolve around implementing the remaining requirements first. When it comes to the requirements, we did not manage to implement the Calendar view functionality as well as functionality for pregnant woman's partner to interact with the application in some way. Moreover, the application would benefit greatly from software testing. As of now, it's hard to determine how reliable and how performant the app is.

Additionally, for a more complete solution, the application could incorporate a back-end system (server) together with database support. Currently, our solution

is purely front-end oriented and depends on Web API to fetch the data. Back-end system would benefit clients at Karlstad University greatly, because the data that would be collected by the fitness tracker could then be saved to the universities' own database. However, the app would still be required to communicate with the Web API. At the same time, it would be possible for nurses to have access to the patient's armband data through the database, if need be. That's unlike the current version of our app, where only the user of the tracker can access the data collected.

Appendices

A) Excess Experiment Data

Spreadsheet:

https://docs.google.com/spreadsheets/d/1ap5LWyzZ1rxWCp9MaEUtrrM0tdvG_9N53Lja4S1vy54/edit?fbclid=IwAR0eObDVMinDtwzDAEjFyfsVLqDv7k3zW_k4lvLhhFxlVCo44p5KKKUH2QM#gid=1884432564

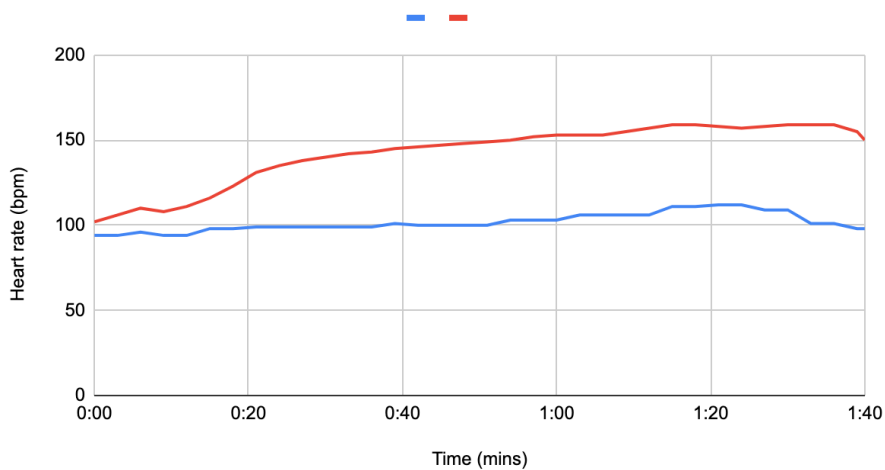
Tables of stairs data

The tables show the maximum, minimum and average heart rate during this activity. Graphs show heart rate deviation between devices during stairs exercise.

Test participant 1: 1st round

Test 1		
Max HR	112	159
Min HR	94	102
Avg HR	102	142
	Fitbit (bpm)	Polar (bpm)

Test 1 Stairs

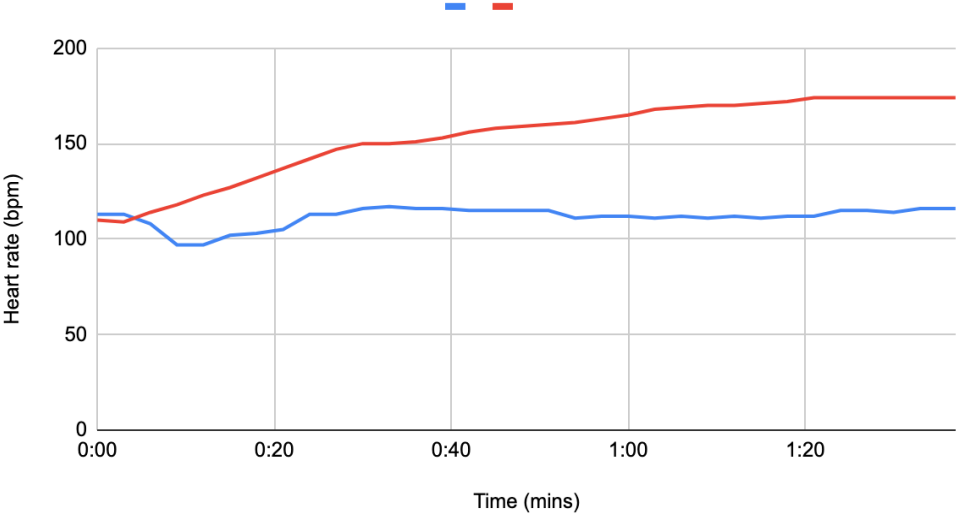


Blue line = Fitbit Charge 4, Red line = Polar H7

Test participant 1: 2nd round

Test 2		
Max HR	117	174
Min HR	97	109
Avg HR	112	154
	Fitbit (bpm)	Polar (bpm)

Test 2 Stairs

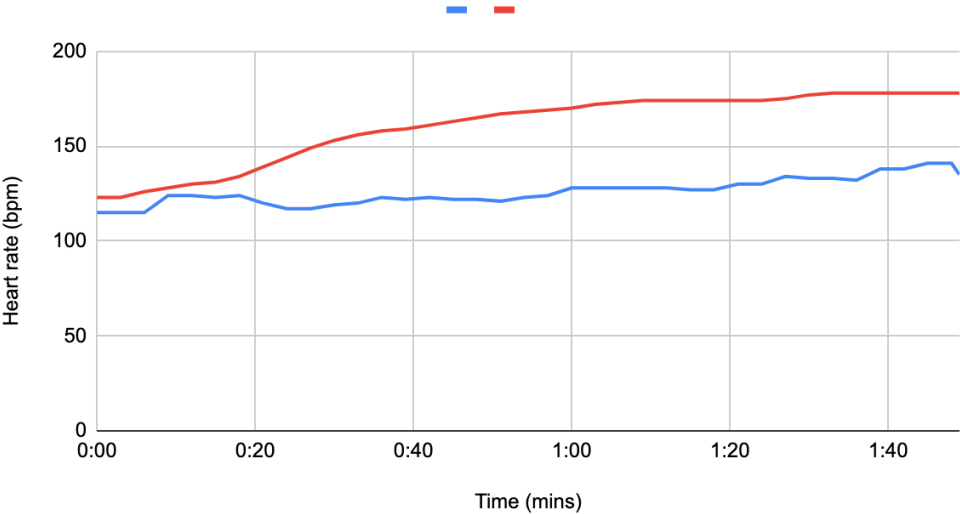


Blue line = Fitbit Charge 4, Red line = Polar H7

Test participant 1: 3rd round

Test 3		
Max HR	141	178
Min HR	115	123
Avg HR	126	161
	Fitbit (bpm)	Polar (bpm)

Test 3 Stairs exercise

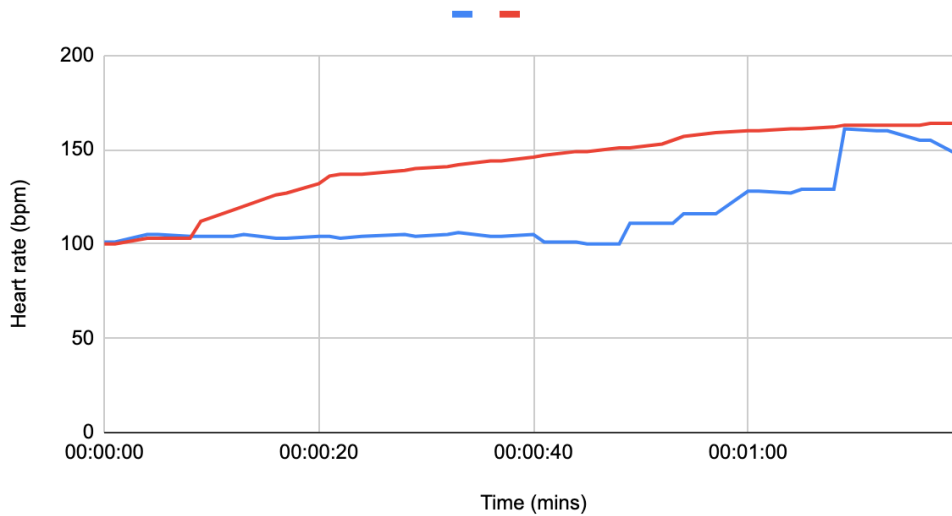


Test participant 1: 4th round - without arm movement

Testing whether limited arm movement when jogging up the stairs will give better results.

Test 1 - without arm movement		
Max HR	129	162
Min HR	100	100
Avg HR	108	138
	Fitbit (bpm)	Polar (bpm)

Test 1 : Without arm movement

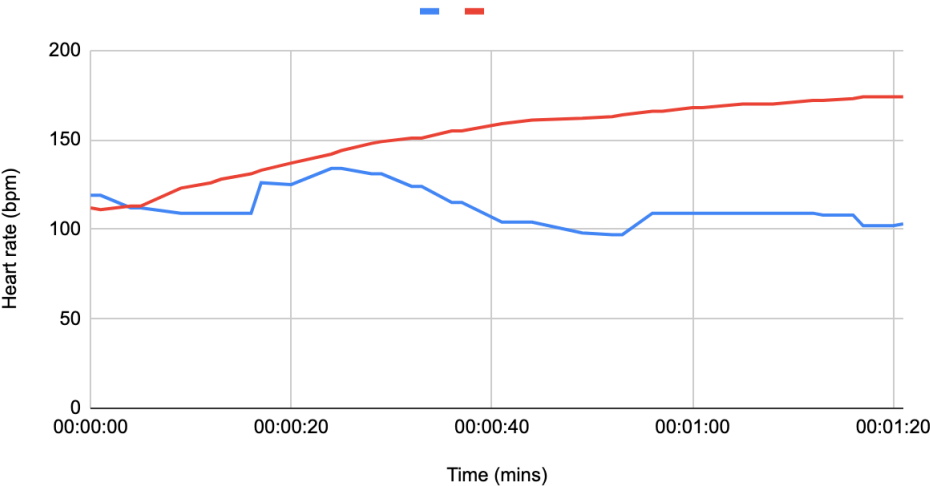


Blue line = Fitbit Charge 4, Red line = Polar H7

Test participant 1: 5th round - without arm movement

Test 2 - without arm movement		
Max HR	134	174
Min HR	97	111
Avg HR	113	151
	Fitbit (bpm)	Polar (bpm)

Test 2: Without arm movement

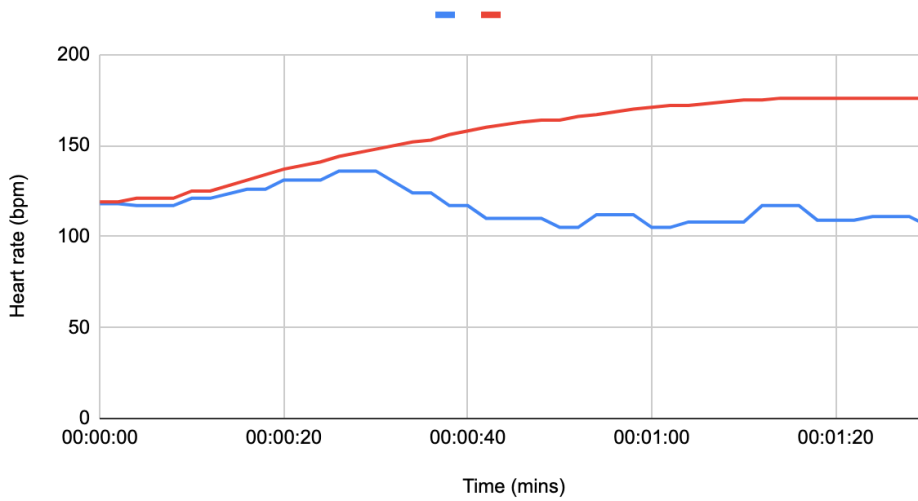


Blue line = Fitbit Charge 4, Red line = Polar H7

Test participant 1: 6th round - without arm movement

Test 3 - without arm movement		
Max HR	136	176
Min HR	105	119
Avg HR	116	155
	Fitbit (bpm)	Polar (bpm)

Test 3: Without arm movement

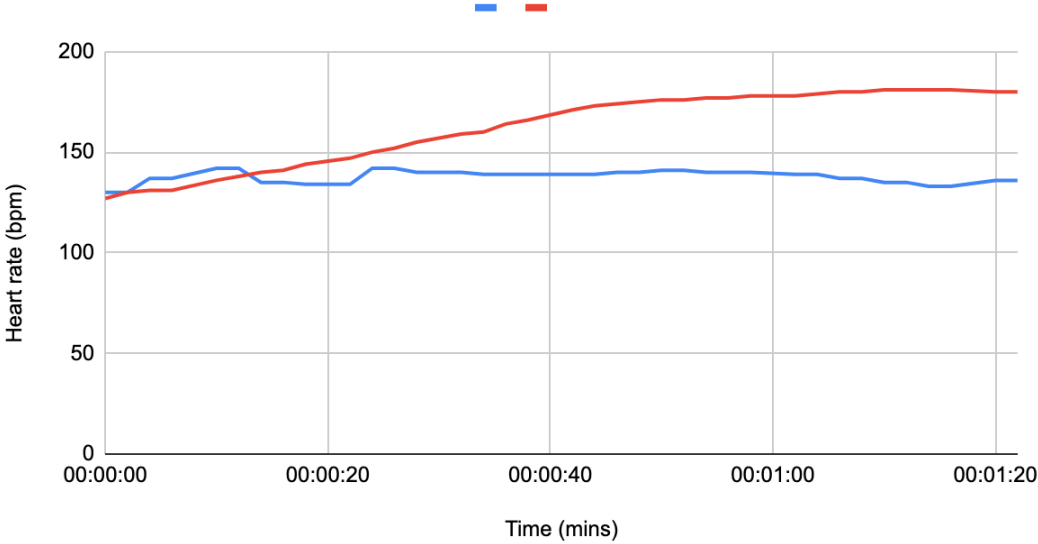


Blue line = Fitbit Charge 4, Red line = Polar H7

Test participant 1: 7th round - without arm movement

Test 4 - without arm movement		
Max HR	142	181
Min HR	130	127
Avg HR	138	161
	Fitbit (bpm)	Polar (bpm)

Test 4: Without arm movement

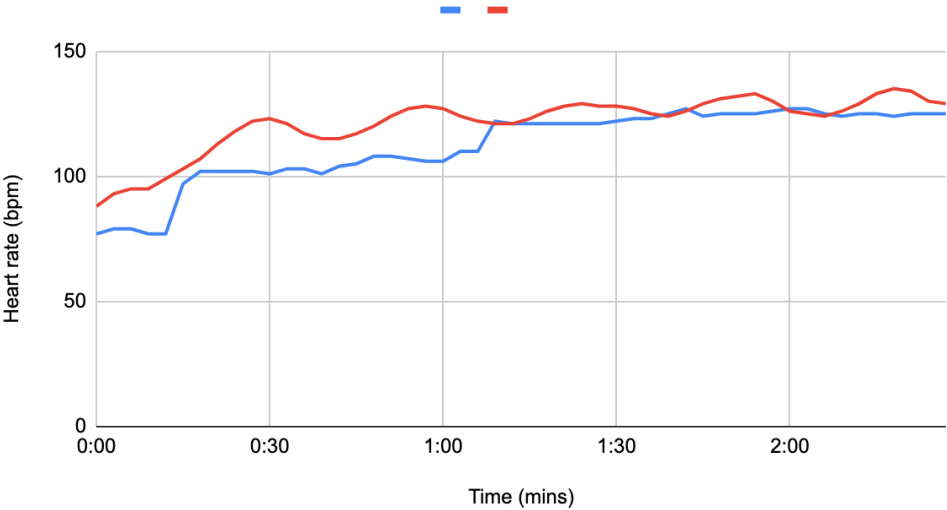


Blue line = Fitbit Charge 4, Red line = Polar H7

Test participant 2: 1st round

Test 1		
Max HR	127	135
Min HR	77	88
Avg HR	112	121
	Fitbit (bpm)	Polar (bpm)

Test 1 : Stairs exercise

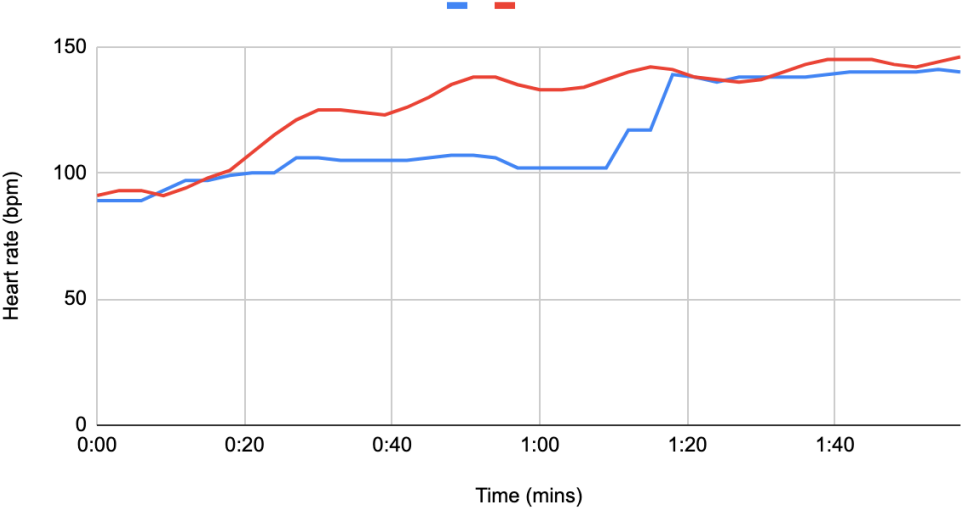


Blue line = Fitbit Charge 4, Red line = Polar H7

Test participant 2: 2nd round

Test 2		
Max HR	141	146
Min HR	89	91
Avg HR	115	128
	Fitbit (bpm)	Polar (bpm)

Test 2: Stairs exercise

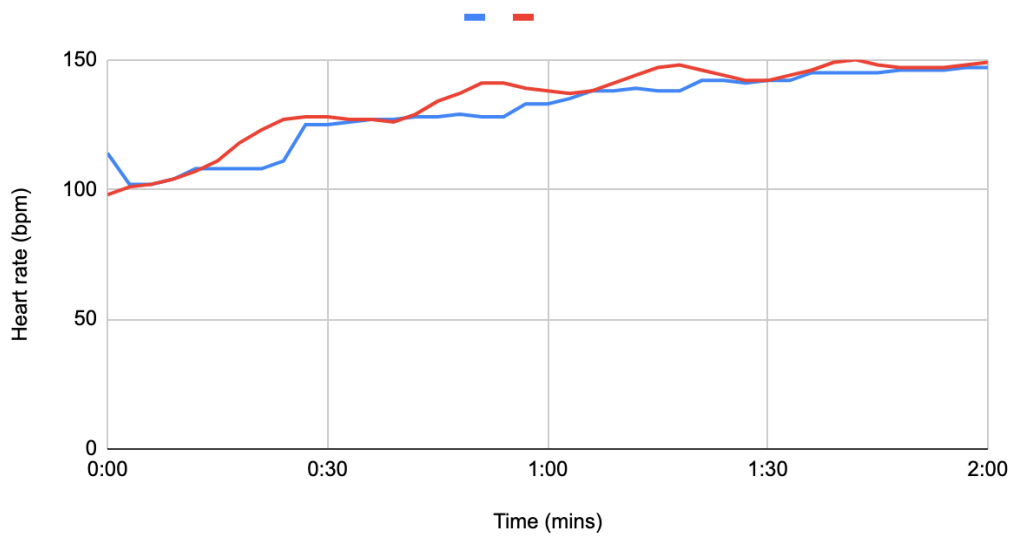


Blue line = Fitbit Charge 4, Red line = Polar H7

Test participant 2: 3rd round

Test 3		
Max HR	147	150
Min HR	102	98
Avg HR	130	134
	Fitbit (bpm)	Polar (bpm)

Test 3: Stairs exercise



Blue line = Fitbit Charge 4, Red line = Polar H7

Tables of walking data

Tables below show the maximum, minimum and average heart rate during 30 minute walk activity and compare these values between devices.

Test 1		
Max HR	130	128
Min HR	104	99
Avg HR	119	118
	Fitbit (bpm)	Polar (bpm)

Test 2		
Max HR	135	141
Min HR	97	102
Avg HR	119	120
	Fitbit (bpm)	Polar (bpm)

Test 3		
Max HR	138	147
Min HR	89	118
Avg HR	119	134
Time	Fitbit (bpm)	Polar (bpm)

Test 4		
Max HR	136	155
Min HR	83	119
Avg HR	109	135
Time	Fitbit (bpm)	Polar (bpm)

B) Notes from the sleep experiments

This test was done by putting on the Fitbit before going to bed. The data was collected from the fitbit app such as hours slept, the resting bpm and duration of the sleep. The app also provides the person with information such as how many times and how long the person was restless and got awake.

Test participant 1

Night 1: Friday 4th February

Sleep start: 3:10 am

Sleep end: 8:14 am

I went to bed at 3am so the fitbit is accurate about when my sleep started. I woke up at 8:15 am which is accurate as well.

The fitbit recorded the time asleep as 4 hours and 29 minutes. It took 13 minutes to fall asleep. The resting heart rate was 51bpm. The highest bpm recorded was 73 while the lowest being 44.

Night 2: Wednesday 9th February

Sleep start: 3:07 am

Sleep end: 6:59 am

I went to bed at 3am and had an alarm set for 7 am. The wristband is accurate with when I went to bed and when I woke up. It also shows that I had woken for about 15 minutes which is when I woke up in the middle of my sleep.

The fitbit recorded the time asleep as 3 hours and 35 minutes. It took 9 minutes to fall asleep. The resting heart rate was 55 bpm. The highest bpm recorded was 58 while the lowest being 44.

Test Participant 2

Night 1: Tuesday 8th February

Sleep start: 10:28 pm

Sleep end: 6:52 am

I began sleeping at 10:28 but the fitbit app started recording my sleep from 11:02. The fitbit records a total of 7 hours and 50 minutes. Fitbit shows I woke at 6:52am which is accurate. It also shows I had woken up for 1 hour and 12 minutes and that is correct. The resting heart rate was 67 bpm. Total deep sleep was 1 hour 18 minutes which is 16.6%.

Night 2: Wednesday 9th February

Sleep start: 10:57 pm

Sleep end: 6:27 am

I began to sleep at 10:57pm but the fitbit app shows that I started sleeping at 11:14 pm. I woke up at 6:27am which corresponds with the fitbit app. I woke up in the middle of night and stayed awake for a bit which fitbit shows was 39 minutes.

The resting heart rate was 70 bpm. The fitbit shows a total of 56 minutes (12.90%) deep sleep.

Night 3: Tuesday 22nd February

Sleep start: 01:15

Sleep end: 07:43

I went to bed at around 1am which Fitbit is accurate with. I had an alarm for 7:40 and it shows I woke up at 7:43 which is accurate.

The resting heart rate was 68 bpm. The fitbit shows a total of 42 minutes (10.80%) deep sleep.

Night 4: Tuesday 1st March

Sleep start: 01:06

Sleep end: 11:03

The fitbit is accurate about when I went to sleep as I went to bed at 1am. I set my alarm for 11 am so the Fitbit is accurate about what time I woke up.

The resting heart rate was 69 bpm. The fitbit shows a total of 1 hour and 10 minutes (11.70%) deep sleep.

C) Project Timeline

The image of our project timeline is too big for the paper and would be unreadable if copied in, so here's a link to the [full image](#).

D) Project Repository

Github:

https://github.com/cosmosgirlandcrows/Bachelor2022_FitbitreactJS

Source code:

https://drive.google.com/file/d/1G0az8shskdrMG9G_5Bjq7-Wr90e7iNN9/view?usp=sharing

Sources

Atlassian. (n.d.). *What is Agile?* <https://www.atlassian.com/agile>

Benedetto, S., Caldato, C., Bazzan, E., Greenwood, D. C., Pensabene, V. & Actis, P. (2018). Assessment of the Fitbit Charge 2 for monitoring heart rate. *PLOS ONE* 13(2): e0192691. <https://doi.org/10.1371/journal.pone.0192691>

Bland JM & Altman DG. (1986). *Statistical methods for assessing agreement between two methods of clinical measurement, Lancet.*
<https://www-users.york.ac.uk/~mb55/meas/ba.htm>

Bradley, S. (2014, 29. March). *Design Principles: Visual Perception And The Principles Of Gestalt.*
<https://www.smashingmagazine.com/2014/03/design-principles-visual-perception-and-the-principles-of-gestalt/>

Choose a License. (n.d.). *MIT license.* <https://choosealicense.com/licenses/mit/>

European Commission. (2022, 25. March). *European Commission and United States Joint Statement on Trans-Atlantic Data Privacy Framework.*
https://ec.europa.eu/commission/presscorner/detail/en/ip_22_2087

European Commission. (n.d.). *EU-US data transfers.*
https://ec.europa.eu/info/law/law-topic/data-protection/international-dimension-data-protection/eu-us-data-transfers_en

European Data Protection Supervisor. (n.d.). *International transfers.*
https://edps.europa.eu/data-protection/data-protection/reference-library/international-transfers_en

Fitbit. (2021, 16. August). *Fitbit Privacy Policy*.

<https://www.fitbit.com/global/us/legal/privacy-policy>

Fitbit. (n.d.). *Wear & Care*. <https://www.fitbit.com/global/fi/product-care>

Fitbit Developer. (n.d., a). *Getting started with the Fitbit APIs*.

<https://dev.fitbit.com/build/reference/web-api/developer-guide/getting-started/>

Fitbit Developer. (n.d., b). *Authorization*.

<https://dev.fitbit.com/build/reference/web-api/developer-guide/authorization/>

Freecodecamp. (2021, 6. April). *What is a PWA? Progressive Web Apps for Beginners*. <https://www.freecodecamp.org/news/what-are-progressive-web-apps/>

Garmin. (n.d.). *Water Rating Definitions*.

<https://www.garmin.com/en-US/legal/waterrating-definitions/>

Giavarina D. (2015). Understanding Bland Altman analysis. *Biochemia Medica* 2015;25(2):141–51. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4470095/>

Hilts, A., Parsons, C. & Knockel, J. (2016). *Every Step You Fake: A Comparative Analysis of Fitness Tracker Privacy and Security*. (Open Effect Report 2016).

Open Effect. https://openeffect.ca/reports/Every_Step_You_Fake.pdf

Integritetsskydds myndigheten. (2021, 13. September). *Schrems II-domen och överföringar till tredje land*.

<https://www.imy.se/verksamhet/dataskydd/det-har-galler-enligt-gdpr/overforing-till-tredje-land/schrems-ii-domen-overforingar-till-tredje-land/>

Javatpoint. (n.d.). *URI vs URL | Difference between URI and URL*.

<https://www.javatpoint.com/uri-vs-url>

Jay Summer (2022, 25. March). *What Is a Normal Sleeping Heart Rate?*
<https://www.sleepfoundation.org/physical-health/sleeping-heart-rate> (Accessed 6.04.2022)

Jonas Emre Tunc, Daniel Bahmiary & Suleiman Zaman (2020). WRISTBAND EVALUATION, Karlstad University.
<http://kau.diva-portal.org/smash/record.jsf?pid=diva2%3A1536589&dswid=2308>

Json.org. (n.d.). *Introducing JSON*. <https://www.json.org/json-en.html>

Lupton, D. & Pedersen, S. (2016). An Australian survey of women's use of pregnancy and parenting apps. *Women and birth [online]*, 29(4), pages 368-375.
<https://dx.doi.org/10.1016/j.wombi.2016.01.008>

MDN Web Docs. (2021, 3. October). *HTTP request methods*.
<https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>

Nálevka P, (2016), *Sleep as Android: Smart alarm* [Mobile App], Google Play Store, <https://play.google.com/store/apps/details?id=com.urbandroid.sleep>

Pacheco, D. (2022, 11. March). *Deep Sleep: How Much Do You Need?*
<https://www.sleepfoundation.org/stages-of-sleep/deep-sleep> (Accessed 23.05.2022)

Privacy not included. (2020, 2. November). *Fitbit Charge 4*.
<https://foundation.mozilla.org/en/privacynotincluded/fitbit-charge-4/>

Privacy not included. (n.d.). *Our methodology*.
<https://foundation.mozilla.org/en/privacynotincluded/about/methodology/#minimum-security-standards>

React. (n.d., a). *React - A JavaScript library for building user interfaces.*

<https://reactjs.org/>

React. (n.d., b). *Components and Props.*

<https://reactjs.org/docs/components-and-props.html>

React. (n.d., c). *Introducing Hooks.* <https://reactjs.org/docs/hooks-intro.html>

React. (n.d., d). *Hooks at a Glance.* <https://reactjs.org/docs/hooks-overview.html>

React. (n.d., e). *Context.* <https://reactjs.org/docs/context.html>

React. (n.d., f). *Thinking in React.* <https://reactjs.org/docs/thinking-in-react.html>

Red Hat. (2017, 31. October). *What is an API?*

<https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces>

Icon used in our application:

Smart Watch icon, created by Blak1ta - Flaticon.

https://www.flaticon.com/premium-icon/smart-watch_2976731?term=smart%20watch&page=1&position=10&page=1&position=10&related_id=2976731&origin=search