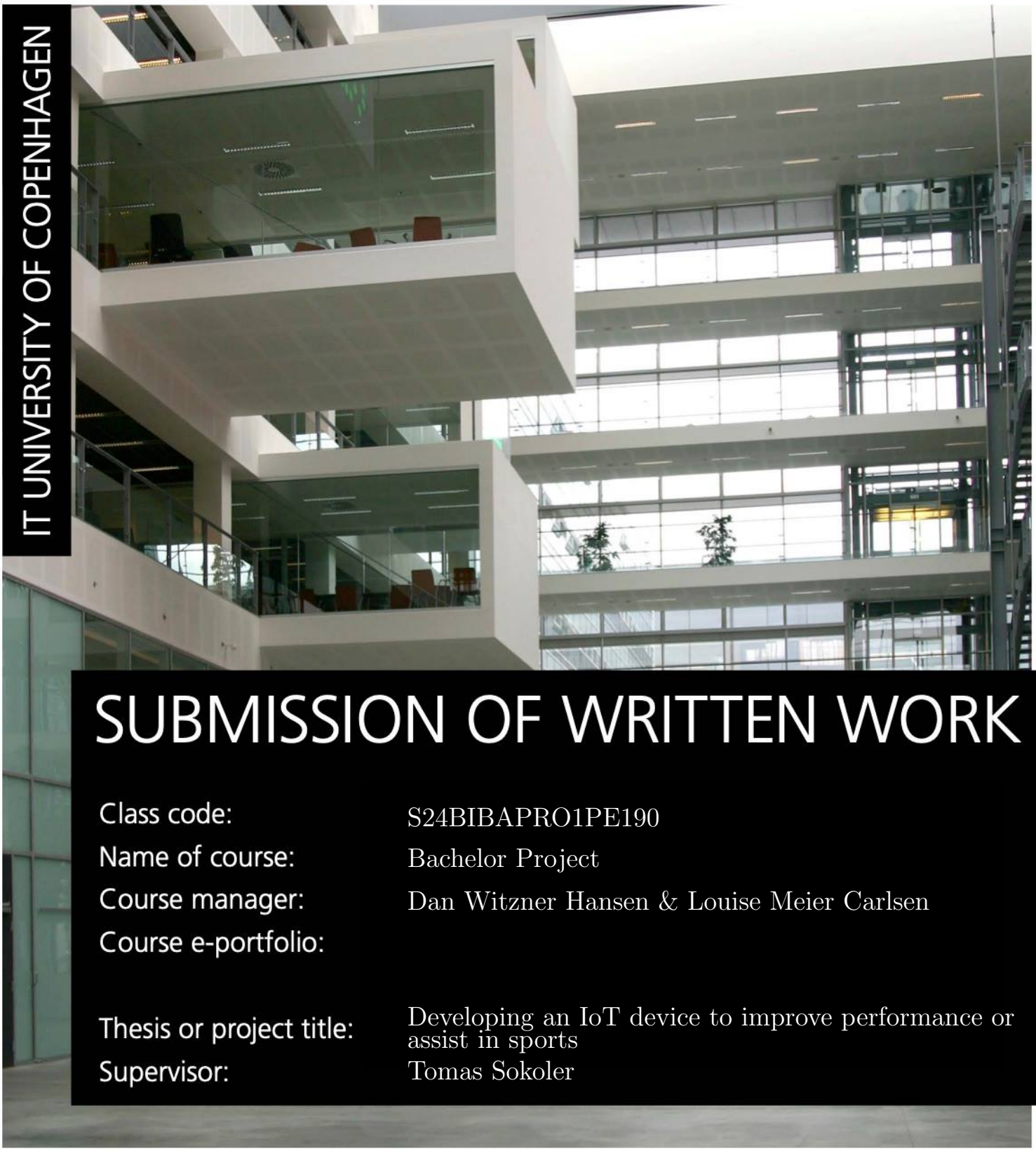


SUBMISSION OF WRITTEN WORK

Class code:	S24BIBAPRO1PE190
Name of course:	Bachelor Project
Course manager:	Dan Witzner Hansen & Louise Meier Carlsen
Course e-portfolio:	
Thesis or project title:	Developing an IoT device to improve performance or assist in sports
Supervisor:	Tomas Sokoler

	Full Name:	Birthdate (dd/mm-yyyy):	E-mail:
1.	Vitus Girelli Meiner	09/02-1999	vime@itu.dk
2.	Lucas Alexander Bjerre Fremming	13/06-2001	lufr@itu.dk
3.	Rasmus Nielsen	28/07-1999	rasni@itu.dk
4.			@itu.dk
5.			@itu.dk
6.			@itu.dk
7.			@itu.dk

Developing an IoT device to improve performance or assist in sports.

Vitus Girelli Meiner, Lucas Alexander Bjerre Fremming & Rasmus Nielsen

May 2024

Abstract

This thesis aims to explore the feasibility of developing an IoT system to be used in conjunction with a pin-loaded fitness machine, to provide immediate feedback, and analysis based on the principles of velocity based training. The system consists of a distance sensor mounted on the fitness machine and an application for user interaction. The distance sensor, connected to a Raspberry Pi, collects and computes movement data during an exercise. This data is transmitted to a cloud infrastructure, consisting of various Amazon Web Services, via the MQTT protocol, where it is further processed by applying velocity based training principles using AWS Lambda functions and stored in AWS DynamoDB. The application, developed using Next.js, provides an intuitive graphical user interface for users to monitor their workout performance, including an embedded QR code scanner to access the machine. The data displayed in the application offers insights into workout intensity and one-rep max estimations, as well as phase and repetition tracking, allowing users to optimize their training sessions. The project follows an Agile development framework with an iterative approach to help progress towards a final solution, where emphasis on testing and feedback has been central to refining the system's functionality and user experience. The implementation of this IoT solution demonstrates the potential of integrating IoT technology in fitness equipment to provide training feedback with the goal of improving overall workout efficiency. Future work could explore further automation of sensor tracking, scalability for commercial use, and enhanced user interface features to support larger user bases and deliver an even more integrated user experience. The final prototype developed in this project displays a step in the right direction for a final solution that integrates IoT technology, and fitness principles with fitness equipment to provide feedback and analysis for a user.

Contents

1 Introduction

In the current technology-driven world, the pursuit of health has become increasingly integrated into daily life, and the demand for sophisticated yet accessible tools to monitor workout progress has become more prevalent. Fitness goers constantly seek innovative and user-friendly solutions that not only enhance their training efficiency but also provide feedback to optimize performance. In response to this growing need, this thesis aims to explore the potential of an Internet of Things (IoT) based product, designed specifically around the principles of velocity based training (VBT). A method that uses the speed of a lift to optimize the training. This approach allows users to receive immediate feedback on their performance, enabling them to adjust their workout more effectively.

VBT is an exercise principle that uses technology to monitor the velocity of the user's movements during workouts. The basic idea behind VBT is to use the speed of a workout movement to calculate the user's intensity of an exercise based on a velocity threshold. Trainers and athletes can therefore maximize training effectiveness and efficiency by adjusting the load and speed according to the VBT feedback. Based on this, this thesis aims to answer the following problem:

How can IoT technologies be used to develop a product that integrates a device with an application for pin-loaded fitness machines, offering analysis of a user's workout based on the principles of velocity based training in a user-friendly way?

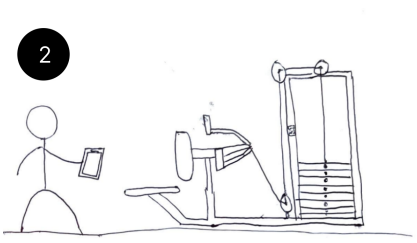
This project intends to create a solution to give consumers a simple way to monitor and analyze their workouts using the principles of VBT. With the use of a quick response (QR) scanner built into the application, users may scan a QR code placed on a workout machine to allow the application to recognize the machine and send apposite data. Once the application and machine are paired, a distance sensor mounted on the machine starts gathering data. This data is then processed and sent to keep in a database. In the end, the user receives the processed data on the application showing their workout metrics and analysis.

Following the data collection phase, the data undergoes an elaborate journey towards bettering user experience and usability. By using Next.js, a flexible framework for building React applications, we construct the frontend interface as a progressive web app (PWA). Next.js provides a foundation to create dynamic, client-side experiences that ensure that users can interact with the tracking device. Tailwind CSS is integrated into the frontend development process to make the user interface (UI) more aesthetic and streamline the styling. We also use shadcn/ui to speed up our development process, as it provides a comprehensive library of predesigned components that are easy to implement, enhancing both the functionality

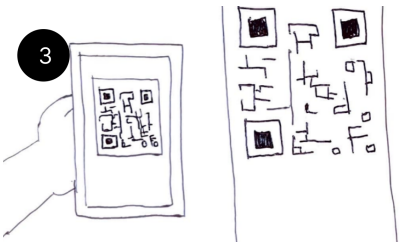
and design of our application. Transitioning to the backend infrastructure, we utilize the computational power of Amazon Web Services (AWS) to facilitate data processing and storage. AWS Lambda functions are created and deployed to handle the incoming data streams from the device. Simultaneously, AWS DynamoDB serves as the database solution, to securely store the processed workout data for seamless retrieval using a non-relational database. Moreover, the distance sensor, which is the main component in the data collection process, is set up with a Raspberry Pi. The IoT device uses the MQTT protocol by publishing and subscribing to topics, where the broker, being AWS IoT Core, acts as a conduit, facilitating the communication between the device and the cloud backend infrastructure. Through this setup, the product has the ability to perform real-time data collection and processing.

1.1 Storyboard

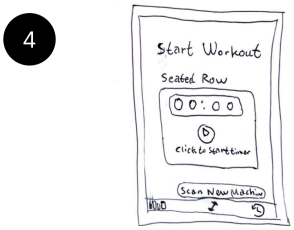
John Doe enters the fitness center to workout.



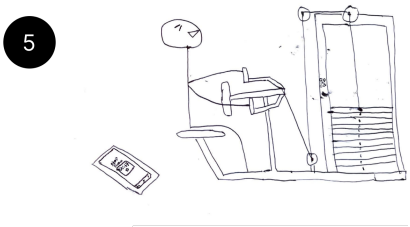
John Doe finds a pin-loaded fitness machine to workout on.



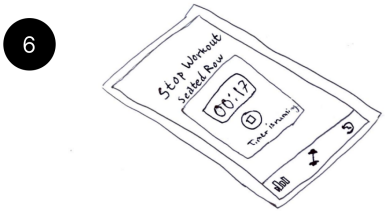
John Doe scans the QR code on the machine with his app.



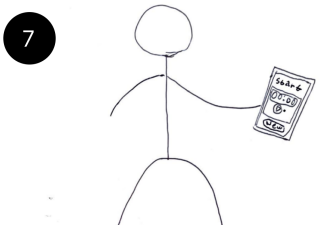
The app recognizes the specific machine and prompts John Doe to start his workout.



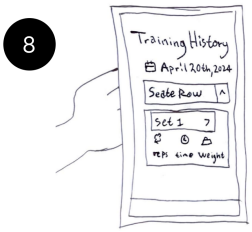
John Doe starts the app and uses the machine.



The application displays the elapsed time for the current set



John Doe ends his workout by pressing the stop button on the app.



John Doe sees how well he performed his workout, by looking at his training history which shows useful insights.

Figure 1: Overall caption for the storyboard

The storyboard displayed in figure 1 serves as a guideline of the features and functionality that the solution should contain, from a user's perspective.

2 Background

In this section, the background of the project will be discussed. The prototype is an IoT device that can collect and compute real-time data from an exercise, to then showcase it to the user on an application. There exists a broad range of ways to workout, and methods used to try and effectuate one's training. Without proper feedback, it is difficult to tell whether the work you put in has had any real effect or benefit.

2.1 Velocity Based Training

Velocity based training is most commonly used to calculate the intensity of the user's workout. Normally, the user would have to subjectively guess the intensity of their workout depending on how hard they perceived the set to be. In fitness terms, this is commonly referred to as the rate of perceived exertion (RPE)[34]. With VBT, athletes and other fitness goers can now have a way of measuring their intensity objectively to maximize their workout efficiency and improve their performance[12]. Incorporating VBT offers numerous benefits that make it easier to adjust weight and repetitions after each set during a workout session. This will encourage you to push harder and work smarter towards your goal. Visualizing the data makes it much easier to workout with intent. Being able to monitor and manage the load and intensity more precisely, helps prevent overtraining and gives you the advantage of reducing possible injuries[10].

2.2 Internet of Things

The Internet of Things, commonly referred to as IoT, is the connection of physical objects like gadgets, vehicles, and appliances to the internet. This is done by using various sensors, actuators, and software. IoT can range from small devices for personal use, at home or work, to large-scale and complex industry mechanisms. IoT technology also adds the ability to create a network of multiple smart devices allowing them to communicate with each other. This enables them to perform various tasks autonomously, like monitoring environmental conditions or keeping track of shipments and inventory. The capabilities are vast[44].

2.2.1 Industry Scale

IoT technology can be implemented in many different industries with very different purposes. Everything from warehouse logistics to the health care sector as well as farming and agriculture. IoT devices are used to monitor many different parameters, which can help to identify patterns and anomalies at the moment they occur, because of their real-time capabilities. This is advantageous for companies to spot errors and optimize their production, which in the long run can help cut costs and

maximize profit and efficiency[44]. Another example is the medical healthcare field, where IoT is used when patients are either monitored or tracked for diagnostics or surveillance[21].

2.2.2 Consumer Scale

IoT devices can be found in many people's homes and are used by most in their everyday life[13]. The term smart home is often used to describe the homes of people where IoT technology is used to enhance already existing items, like switching the light on and off or moving the blinds up and down. By incorporating IoT into these items they can be controlled from a remote device over the internet, usually being people's smartphones. Especially security in smart homes is prevalent, for instance, if someone is using your house doorbell or breaks into your house, you will receive an alert about these actions. These are just a few examples of the many possibilities of connecting your home to the internet to monitor, manage, and control various household items.

2.3 IoT Architecture

To fully grasp the capabilities and the innovative potential that come with IoT, it is crucial to understand its technical foundations. These foundations are built on a variety of technologies that allow one or more devices to autonomously collect, process, and transmit data.

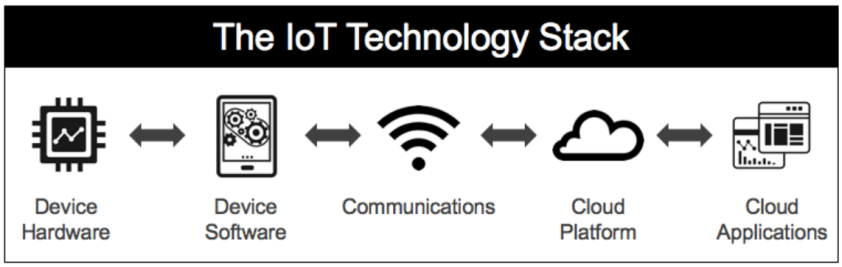


Figure 2: 5 Layer Diagram of The IoT Architecture[36].

.

The architecture of an IoT technology stack can be split into five layers as seen in figure 2. All of these layers play an essential role in enabling the integration and communication of devices, keeping them connected. This technology stack is found in all IoT technologies, no matter the scale of the system, and it is the combination of these layers that form the foundation upon which an IoT solution is built. They facilitate, capture, process, and distribute data across environments[36]. The following subsections will describe the different layers of the 5-layer technology stack.

2.3.1 Device Hardware

The first layer of the technology stack is device hardware, this is where the "Things", in the Internet of Things, are found. These devices are either products that are inherently connected or devices modified to make them connected. Depending on the solution and the needs of the project, the first part of the device hardware can be as small and simple as a System on a Chip (SoC), or if it is more demanding it can be a microcontroller (MC)[36]. An MC serves as the central processing unit and runs dedicated code that is programmed to manage inputs from sensors and return control outputs. Examples of popular MC's are[17]:

- **Raspberry Pi**, is a more complex MC, able to run an operating system (OS) offering a richer programming environment and greater computational power.
- **Arduino**, is more widespread offering larger documentation and more projects online, however, it does not offer the same computation power as its competitors, and is mostly used in smaller projects or schools.
- **ESP32**, is a very small budget-friendly MC offering built-in Wi-Fi, which makes it suited for applications requiring wireless internet connectivity for data transmission. However, they do tend to require higher technical knowledge than their competitors.

MCs are primarily used to perform specific tasks based on real-time data and are designed to be small and consume minimal power. This energy efficiency combined with the ability to handle and compute data from sensors is the reason that MCs have an essential role when designing IoT devices[46].

Sensors are the second part of the device hardware layer. Sensors can be added to an MC to gather data from the surrounding environment. These data can include measurements such as temperature, humidity, motion, light, and many others, depending on the specific sensor used. Sensors play an important role in capturing data, but to utilize the full potential of IoT these sensors should be connected to an actuator. An actuator is a component that is triggered based on the readings of a sensor. For example, the actuator could be a cooling fan that waits for a signal to start or stop from the controller. The controller will send a signal when a certain temperature from the sensor is reached[16].

2.3.2 Device Software

The device software layer is the layer that turns the hardware into a smart device. It defines how and what software you use on your device hardware, as well as how you communicate with the cloud layer or other devices. It is a crucial layer,

regarding cost. The creation of hardware is usually more costly than software since it can be very expensive to build hardware for specific purposes. It is therefore, easier to have a more generic piece of hardware that is multi-purposed, and then create specific software that makes you utilize the hardware in a specific way. This can help in cutting costs and making sure that if any customization is needed to be done in the future, it can be done through software instead of needing to rebuild the hardware. This is known as *software-defined hardware*, where the device software gives the hardware its purpose, and can make the hardware device serve multiple applications[36].

2.3.3 Communication

IoT communication is the connection of various devices over the internet. It is an important factor in IoT technology since communication enables the devices to gather and exchange data. All IoT devices need to have reliable, but also effective communication protocols to be implemented properly. This can be anything as simple as WiFi or Bluetooth, or something more specific like SigFox[42] or MQTT[32].

MQTT is an IoT communication protocol designed as a publish/subscribe messaging transport. MQTT today is used in all kinds of applications, both in consumer and industry scale systems[38].

2.3.4 Cloud Platform

In the fourth layer of the technology stack, we find the backbone of any IoT solution, the cloud platform. This layer encapsulates everything happening in the cloud, including computation and storage. There are a lot of cloud services available for an IoT system, e.g. AWS, Microsoft Azure, and Oracle, to mention a few.

Data Computation and Processing

To make sense of the data collected by the hardware there needs to be some kind of data processing present in an IoT solution. This processing is the collection and manipulation of data to produce meaningful information[30], which is then saved in a chosen storage solution.

Data Storage

There exist many different types of databases, and each has its benefits and drawbacks. Databases can mainly be categorized into two groups, relational and non-relational databases. The main difference between the two is that a relational database uses structured query language to process and store data. The way data is stored in these databases is with rows and columns in a tabular form. That way, the different data attributes are easily presented and relationships between them can be outlined and created[43]. Examples of such databases are MySQL

and PostgreSQL, where both update, delete, store, etc., are all done with SQL statements[45].

Non-relational databases do not use SQL to manage data, therefore they are also known as NoSQL databases. They instead use a variety of data models to access and manage information. This means that the implementation also differentiates depending on the specific data model, although many NoSQL databases use Javascript Object Notation (JSON), a lightweight data-interchange format that organizes data into key-value pairs. The benefit of using JSON is that it is not tied to any specific programming language or platform, making it scalable and flexible[41].

In an IoT solution, both types of databases can be applied, but for larger-scale solutions, NoSQL databases are more common. This is because traditional SQL databases are usually less well-suited for managing larger data sets, and are therefore not as ideal for higher-performing systems as a NoSQL database would be[29].

2.3.5 Cloud Applications

The fifth and final layer in the IoT technology stack is the cloud applications. These are the various applications that the end-user interacts with. Usually, it is implemented as a form of a web application, although there are situations where dedicated apps for desktop, mobile, and wearables might be needed. The cloud application can be everything from a consumer-directed application that controls the user's blinds, or an internal application that shows warehouse statistics for a specific firm[36].

2.4 User Friendliness

An important factor in all applications is the fact that it has to be user-friendly. If not, no user will be able to operate the application easily, reducing the chance of the application being a success. It is a priority that a user needs to press and do as few actions as possible, to get all the functionality out of the application. Developers can utilize principles like *Keep It Simple, Stupid* (KISS)[31], to achieve this. Phone applications in particular are mainly operated in one hand using a thumb, and therefore making sure that the most important parts of the application are reachable is very important. To achieve this the thumb-zone can be used to clarify where to place the various parts of the application.

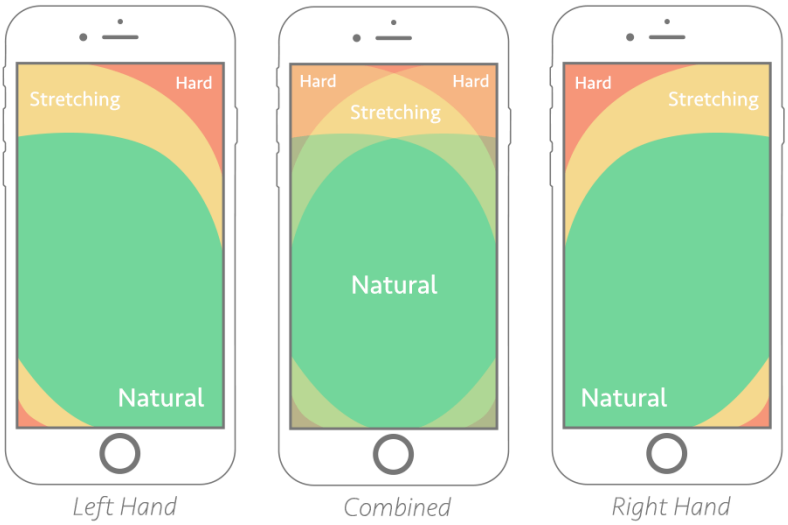


Figure 3: Thumb-zone mapping for left- and right-handed users. The "combined" zone shows the best possible placement areas for most users[3].

Based on this, a developer should aim to place the key features within the green zone as seen in figure 3. To prioritize simplicity and efficiency in the UI the principles of Gestalt psychology can also be used. These can be divided into 7 laws: The law of proximity, closure, similarity, common region, good continuation, symmetry, and the law of common fate[7]. Each of these laws offers valuable insights into how a user perceives and interacts with visual elements. By adhering to all of these principles it enables a developer to create an intuitive UI that makes the user effortlessly use the application.

3 Related Work

Creating IoT devices to assist users in achieving various goals including in the fitness industry is not a new phenomenon. Since the 1970s, when fitness gained popularity among mainstream audiences[4], the quest to find new ways to enhance training has only been expedited. And therefore it is only natural that modern technologies are now being incorporated into the realm of fitness. This section will describe other projects and theories that are either similar and use the same technologies as this project or the principles of VBT, as well as other cases where IoT devices are used in the realm of fitness and working out.

3.1 Academic Work

This section will look at three academic articles relevant to this project. Two of them regarding VBT in strength training and one regarding the architectural design of a system using IoT and AWS.

Article: Researched application of velocity based strength training

In the article "Researched application of velocity based strength training"[1], a study is conducted to provide an overview, and the benefits of monitoring movement velocity in strength training. The article shows that VBT can calculate a minimal velocity threshold (MVT) for a user on a given exercise. The calculated MVT will then remain stable even with changes in the user's maximum strength. During an exercise, this MVT can then be used to estimate the proximity to failure, and the exertion for the exercise, regardless of the weight being lifted. This article therefore shows both the accuracy and effect of applying the VBT principles in strength training and the procedure of how to implement them.

Article: Effects of velocity based training vs. traditional 1RM percentage-based training on improving strength, jump, linear sprint and change of direction speed performance: A Systematic review with meta-analysis

Another article "Effects of velocity based training vs. traditional 1RM percentage-based training on improving strength, jump, linear sprint and change of direction speed performance: A Systematic review with meta-analysis"[6], compares other studies that compare the training effects of VBT against the traditional 1RM percentage based training method. It shows that even though there are no significant differences between the two when comparing the improvement of outcome, VBT exhibits lower volume and less training stress. This means that by utilizing VBT the user can possibly prevent injuries and that it could be better suited for busier individuals.

Article: Design of Scalable IoT Architecture Based on AWS for Smart Live-stock

In the article "Design of Scalable IoT Architecture Based on AWS for Smart Livestock"[5], a cloud-based IoT system is developed to monitor real-time livestock. They use AWS such as IoT Core, Lambda, and DynamoDB for scalable data handling. This study emphasizes automated scaling and strong data processing to meet the needs of smart farming in the modern day. It showcases how cloud technologies can be applied in agriculture efficiently. This system is designed to specifically enhance the management of livestock through real-time data collection and analysis. This illustrates a use case of integration of IoT with cloud computing, in this instance for agricultural advancements.

3.2 Similar Tech Stacks

In a project described by Berat Dinçkan on DEV.to[14], an ESP32 microcontroller is employed to show how an IoT solution utilizing AWS IoT Core, DynamoDB, and Lambda can be done. The ESP32 connects and transmits data, collected by a sensor to monitor humidity and temperature to AWS IoT Core using the MQTT protocol. The data is then processed by Lambda functions, which also handle the storage management by placing the data into a DynamoDB table. Golang is used to write the scripts in the Lambda functions for backend processing. This integration showcases the functionality of AWS' serverless architecture in processing and storing sensor data gathered from IoT devices. This project gives insights into the practical deployment of real-time data handling and infrastructure scalability within the AWS ecosystem.

3.3 Commercial Products

There exist many fitness products that utilize VBT principles. The exact velocity at which an exercise is completed can be measured using tools like wearables, linear position transducers (LPT), laser optic devices, or any other distance sensor.

3.3.1 Linear Position Transducers

A commonly used VBT product in the gym are LPTs e.g. GymAware's product[26]. These transducers are essentially wires within a mechanism that runs along the equipment to which they are attached, commonly being barbells. It measures the displacement during lifts along this line. This allows for the calculation of velocity. They are relatively easy to set up and provide very accurate real-time data. This device is also generally more affordable but despite their easy setup they require regular calibrations to maintain the correct accuracy, and can easily be affected by external factors like how and where the device is placed on the barbell.

3.3.2 Wearables

Wearables are by far the most common IoT device for tracking fitness data. Many commercial products utilizing VBT either require attachments or have to be set up as a standalone device. This is combatted in wearable sensors. This makes it easier for individuals during workouts, whether it is a wearable device on the body or attached to a barbell or machine. These devices have sensors like gyroscopes and accelerometers to accurately track velocity and movement[2]. One of the most popular options for wearable devices on the body is the smartwatch. Almost 1 in 3 American citizens are using a smartwatch[11] and more than 9 out of 10 of these, use their smartwatches for fitness-related reasons[8]. It is small and convenient, and most people already wear it in their daily life. These factors are critical and are what have caused this type of IoT device to become the most popular one.

3.3.3 Distance Sensors

Distance sensors, utilizing laser, ultrasonic, or other distance technologies can be attached to the exercise equipment. For example, the company FLEX[39] uses a laser to measure distance. The sensor measures the minimum and the maximum distances while tracking the time to give the velocity of the lift. The placement of the device has to be in correlation with a specific reference point, often a mat on the floor or another flat consistent surface to ensure correct accuracy of the distance. This product showcases how to utilize distance sensors to obtain the velocity of a lift which can then be used for VBT analysis.

4 Methodology

This section outlines the various methodologies used in the project, providing details on the approach to system design, development processes, and testing protocols.

4.1 Development Framework

The choice to go with an Agile approach for the development of this project originates from how suitable it is for iterative design and how flexible it is in adapting to user feedback and test results. In the context of developing an IoT device where it is not certain which sensor would be the best option for VBT, whilst having to be used in conjunction with a Raspberry Pi and connected to AWS, it is advantageous to use the Agile development framework.

The Agile method emphasizes iterative development cycles, allowing for continuous improvement and refinement throughout the whole life cycle of the project. This also aligns with the evolving nature of IoT, where over time requirements might change and new technology may be introduced. The Agile framework promotes an environment where communication is key so there can be made the best use of each feedback loop.

4.2 Project Phases

To navigate through the whole development process of the project, we have outlined three phases that surround our approach to bring the project to fruition. The different phases represent a structure, from initial concept development to final implementation. Each phase embodies specific aspects of our project's life cycle that helped ensure that we met our objectives.

4.2.1 Conceptual Phase

In any iterative process, everything starts in the conceptual phase. This phase serves as the foundational stage of a project, where the focus is on defining overarching goals and objectives. Through the use of brainstorming sessions and discussions, various ideas and concepts within the scope of a project's theme are explored. Here the emphasis is placed on generating solutions within the realm of IoT technology. Following extensive reflection and evaluation, a clear scope of the project is then established, outlining the primary focus and objectives that need to be pursued. This phase lays the groundwork for the following phases by providing a direction and framework for development.

4.2.2 Prototype Phase

The prototype phase is the transition from conceptualization to practical implementation, where the ideas are transformed into prototypes. Building upon the outcomes of the conceptual phase, the primary objective is to develop a functional prototype that embodies key features and functionalities of the project's scope. Iterative design and an Agile framework for development cycles are employed to refine and further iterate upon prototyping, to ensure it aligns with the user requirements. While a prototype may lack full optimization and is not refined, a prototype serves as a crucial milestone for validating the concepts and gathering early feedback. This is also where testing, such as user tests and other in-field tests will be conducted to assess the usability and to identify areas that need improvement.

4.2.3 Final Implementation Phase

The final phase is the phase of implementation, which represents the culmination of the development journey, where the project is taken from prototyping to a fully realized solution. Depending on the scope of a project, reaching this phase is not always feasible, and will not always be done. This phase is about refinement and integration of all components, to make sure that the product is interoperable and fully functional across the system. In this phase, the key activities may include final thorough testing if any debugging needs to be done, and the last optimization to address remaining issues. The user feedback that would have been collected during the previous phase enables iterative improvements, as a guide for the final adjustments to the UI and functionality. Additionally, this is also where comprehensive documentation is prepared to support users in effectively using the product. The final implementation phase marks the official launch of the product or solution, which is the beginning of its operational deployment and ongoing maintenance and support.

4.3 Requirements Specification

The project entails a set of requirements. These requirements are categorized into functional and non-functional, outlining the operational functionalities and performance expectations of the system.

4.3.1 Functional Requirements

- The user can obtain feedback on their current workout, including data on their repetitions and phases.
- The user can see a history of previously performed workouts.

- The user can see their 1RM statistics for different movements performed.
- The user can easily get access to a chosen machine.
- The user can easily start and stop an exercise.
- The user can see the average intensity of their workouts.
- The user can easily navigate the application.

4.3.2 Nonfunctional Requirements

The non-functional requirements of the system are subdivided into usability, performance, and implementation criteria that it must adhere to.

Usability

- The system must be accessible from smartphones, tablets, and computers no matter what OS is used.
- The application must be able to immediately recognize a QR code.

Performance

- The system should be able to provide data right after a performed set on an exercise is done.
- The system must accurately and efficiently calculate phases and repetitions for a given set within 1 second.
- The application should load and display UI within 2.5 seconds.
- The system must be able to calculate its VBT data within 1 second.

Implementation

- The system must be provided through a progressive web application interface.
- The system must provide a graphical user interface (GUI), that a user can interact with.
- The system must be able to observe and monitor data produced by a user.
- The system must store data collected in a non-relational database.

4.4 Testing

The project is meant to be something that regular people who exercise can use in their workouts, therefore testing it is important. First and foremost we need to see whether the application can be used by people who workout. Is it user-friendly and can it be operated by the user? Secondly, does it add value to people's workouts, do they obtain information that they did not possess beforehand? Lastly, does the product perform as intended, are the different technologies working together? Are the data collected, will it be handled correctly and sent to the database, and does the database store the correct data, that can then be fetched elsewhere? To answer these questions, a set of tests have been put in place. Different tests have been conducted to give the best possible answer as to whether the product works or not. Mock tests, field tests, and application user tests are conducted to get the most accurate understanding of the project's capabilities at each step of the development.

4.4.1 Mock Testing

Mock tests are usually performed early in the project and periodically throughout. This method of testing involves using self-created simulated data instead of real-life data. The benefit of mock testing is that it allows for quick and easy testing without needing the complete setup of data-collecting hardware. This is particularly useful for testing different parts of the project independently, especially when the entire system is not fully integrated or operational[35].

4.4.2 Field Test

Field testing refers to the practice of testing the product in a real-life scenario, away from any controlled environment. The goal of this test is to identify any issues or problems that can occur when the product is tested in action before the final release is put out. It can be for validating the software performance in different environments or detecting any unforeseen bugs, to enhance the user's experience with the product[9].

4.4.3 User Test

User tests are conducted to see if a user can perform specific tasks on the product. The goal is to see if a user who is not familiar with the product can naturally figure out how to use it on their own. User tests are therefore testing the interface of the application, and it is important that the person that the user tests are performed on is neutral and does not have any experience with the product beforehand. There exist different types of user tests, they can either be done in person with the user or remotely. They can also be moderated or unmoderated, meaning you are asking

the user performing the test questions, whilst he is doing the test. These are great for gaining insights on what the user is thinking while using the application, but doing it unmoderated gives a more realistic view of how the person interacts with the application[27].

Evaluating User Tests

There are various ways one can evaluate a performed user test. One of these is using the system usability scale (SUS). The SUS has been used for over 30 years and provides a quick and easy way to evaluate almost all systems compared to industry standards. It is quick, cheap, and only consists of a 10-question survey, which the users of your system will answer. By then applying the scoring system on these answers, the system will get an overall score between 0 and 100. If the score is above 80.3 the system is almost perfect and is loved by the users, if the score is around 68 the system is alright, but could be better. Finally, if the score is 51 or below the system usability should be the number one priority moving forward[28].

5 Implementation of The Architecture

In this section, the technologies selected for each layer of the IoT technology stack used in our implementation will be outlined. The choices for the technologies on each of the layers are based on the requirements of the project.

5.1 Device Hardware

In this project, we have used an ultrasonic sensor. An ultrasonic sensor is a motion sensor that uses high-frequency sound waves to detect the distance between the sensor and its target. We chose to pair this sensor with a Raspberry Pi to obtain greater computational power and the possibility of freely choosing between what programming language we want to write in.

5.2 Device Software

On a Raspberry Pi different operating systems are available to choose between. We selected Raspberry Pi OS due to its ease of setup and its popularity among the available OSs.

Choosing the right programming language is crucial for the controlling software. Python was selected for the device software because that is also what we used in the AWS Lambda functions. This provides more consistency across the technology stack. Additionally, Python's ability to efficiently handle data simplifies the development and maintenance of the software.

A Python script is configured to run as a Linux service, which allows it to execute automatically once the Raspberry Pi is started and connected to WiFi. This setup ensures the script operates reliably and is always ready.

Initially, the script is waiting, subscribing to an MQTT topic that receives either a "start" or "stop" command. Upon receiving a "start" command, the script activates the sensor to collect raw data. Due to the nature of raw sensor data, which often includes a significant amount of noise and redundant values, not all collected data is immediately useful for fitness tracking or analysis.

To address this, another script was implemented to perform noise reduction and process the data. This script is executed with each update on the sensor, and by comparing distances, it processes the data into useful fitness data. By comparing the previous distance with the current one, the script calculates the current phase, that the user is performing, being either concentric or eccentric. By keeping track of this, the script can then calculate repetitions based on phase changes, and send a datapoint in JSON, for each phase to our cloud implementation via the MQTT protocol. A pseudo-code implementation outlining our logic is presented below:

```
    phase = eccentric if previousDistance < currentDistance else
       phase = Concentric

    if previousPhase != currentPhase:
        detectPhaseChange()
```

To reduce the amount of calculation, the script also contains a noise reduction threshold for the distance. This works by only processing data points from the sensor where the distance change is at least 3 centimeters. This removes unnecessary data processing and also ensures that a phase change is only recognized on bigger direction changes. This means that small direction changes, which might happen when the user is under heavy exertion, will not cause a phase change. The script continues until a "stop" command is received, at which point the data collection and processing will stop. To correctly track the last phase of the movement, a running average is introduced that constantly tracks the reset point of the eccentric phases. This reset point is calculated based on the maximum distance of all previous eccentric. The last phase is then calculated to be the last movement within a margin of this reset point. This ensures that the last phase will be tracked to the reset point's distance and time, and not the distance and time present when the "stop" command is received.

This collection of scripts on our hardware ensures that the cloud receives only the most relevant, significant, and clean fitness data, optimizing both the functionality and cost-effectiveness of the IoT solution, as well as ensuring that each script only has one purpose.

5.3 Data Management

Our data management journey starts with the collection of data. Our sensor collects raw data which is then immediately filtered and processed on our Raspberry Pi. Once the data has been computed from raw data into useful fitness data it will be published via MQTT to the topic "sensors/data". This processing is done to minimize the amount of data traffic coming from the Pi by only sending relevant fitness data to our cloud services. In AWS a trigger on the "sensors/data" topic will execute a Lambda function for a final cleaning of the data before it is sent to a table in our database. A trigger on this database will then execute an AWS Lambda function containing our VBT formulas on the latest inserted or modified database item, computing the intensity and one rep max (1RM) for the set performed, which is put into another table.

```
data_point = {
    "distance": 50,
    "time": 0.8169729709625244,
    "phaseTime": 0.8169729709625244,
    "set": 1,
    "rep": 1,
    "phase": "Concentric",
    "date": "25/04/2024",
    "machineId": "Machine1",
    "load": 0
  }
```

Figure 4: Example of a data point sent via the MQTT protocol.

5.4 Communication

Among the various communication protocols available, we have chosen MQTT to be used as the communication protocol for our project. This is due to its lightweight and efficient nature, making it highly suitable for limited bandwidth, which is typical of IoT environments[32]. In our project, MQTT facilitates communication between the IoT device and a specific topic in the MQTT broker, which is an AWS IoT core instance.

Communication Between The Frontend and Cloud

In the application, data for the statistics and history page is directly fetched from AWS DynamoDB. The statistics page queries the "VelocityData" table using sort keys to retrieve metrics such as average intensity and 1RM progress. The history page accesses the "WorkoutData" table, also by using sort keys to filter and organize the detailed workout data coming from specific dates and machines. Also, the device interaction like starting and stopping workouts is managed through a REST call that triggers an AWS Lambda function via AWS API Gateway. This function executes code that communicates with the IoT device to let it know whether to start or stop collecting raw sensor data.

5.5 Cloud Platform

Our entire cloud platform is hosted on AWS, consisting of AWS Lambda for data processing and AWS DynamoDB for data storage. We chose to use AWS since their IoT services are well known, and they have great free tier options. Also, AWS IoT Core played a role in this choice because of its built-in MQTT broker that provides an integrated solution for the communication between our IoT device and the cloud. Another reason for the choice is the coherent integration of services like AWS Lambda and AWS DynamoDB. This cloud setup allowed us to focus

more on developing the application and computation rather than managing the infrastructure.

Data Computation and Processing

AWS Lambda is a serverless computing service that executes code in response to certain events[19]. We have set up a trigger with a rule query statement: `SELECT * FROM "sensors/data"` which can be seen in figure 5, ensuring that every message published to this topic triggers a Lambda function that performs a smaller data cleaning, ensuring that the data is in the correct format, before sending it to our database table "WorkoutData".

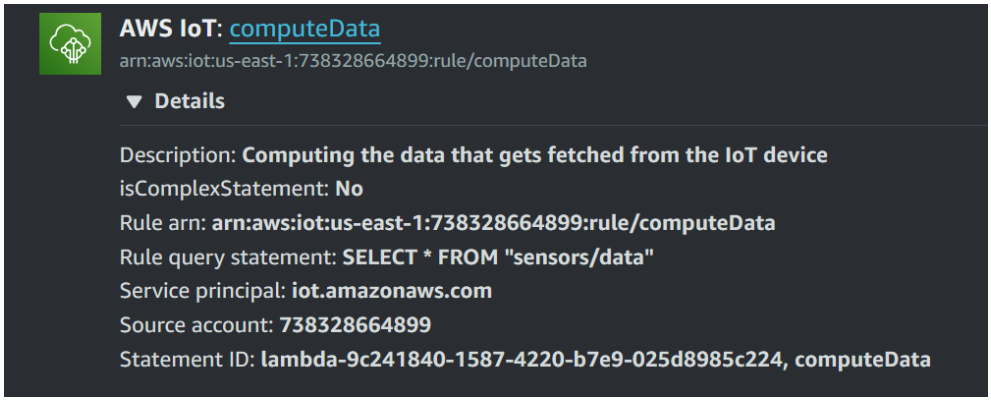


Figure 5: The Lambda trigger for our IoT topic.

In this table, we have set another trigger that runs our second Lambda function, the VBT computation. This trigger runs each time the table receives a change, and a rule has been set to only execute code when the change is an `INSERT` or `MODIFY` event.

```
    for record in event.get('Records'):
        if record.get('eventName') in ('INSERT') or
            record.get('eventName') in ('MODIFY'):
            data = record.get('dynamodb').get('NewImage')
```

If this rule is met the VBT Lambda function then calculates the intensity for the given set in the "WorkoutData" table. This means that each time a new phase is put in the table the VBT function will execute, applying the aforementioned VBT formulas to calculate the intensity and 1RM of the set. It works by keeping track of phases in a given set, and if more than 2 phases are present it will use these to perform its calculation. First, it uses the distance and time of all phases present to calculate an average velocity for the set.

```
    for distance, time in datapoints:
        totalDistance += distance
        totalTime += time
    velocity = (totalDistance / 100) / totalTime
```

This velocity will then be used on par with a 100% intensity velocity threshold, based on the unique `machineID`, to calculate how close the set was to 100% intensity for that specific machine. This threshold was set by calculating the mean velocity across all exercises targeting the same muscle group for novice lifters using the 1RM velocity table in[24].

```
    intensity = machineIdThresholdValue / velocity * 100
```

If the `Load` attribute is present in the set, the function will then use the calculated intensity to calculate a theoretical 1RM for the given set by multiplying the load by the distance to the 100% threshold for the machine.

```
    1RM = load * (100 / intensity)
```

This calculated VBT data is then put into our "VelocityData" table. This design allows for real-time processing of incoming data without the need for dedicated server management.

AWS DynamoDB is a non-relational database, which we chose for its high-performance capabilities. It offers not only the benefits of a typical NoSQL database but also runs serverless like AWS Lambda. This serverless architecture aligns with the agile nature of our project. Eliminating the need for manual provisioning and maintenance means that there is no downtime maintenance, no patching, and it scales to zero[18]. Additionally, the pay-as-you-go model helps to ensure cost efficiency.

In AWS DynamoDB we have two different tables, one for the sets performed by the user and one for the VBT data. We structured our "WorkoutData" table using a partition key, formatted as `USER#<UUID>`. This key ensures that data loads efficiently and supports our primary access pattern, which often involves retrieving all data related to a single user.

The sort key is composed of several attributes:

```
SESSION#<SessionDate>#MACHINE#<MachineID>#SET#<SetNumber>#REP
    #<RepNumber>#PHASE#<Phase>
```

This composite sort key organizes data within each partition, allowing us to store and query workout sessions efficiently. It supports precise queries at multiple levels of detail. From entire sessions down to specific repetitions and phases without the need for full table scans. For each repetition, we store items for both the eccentric and concentric phases in each repetition.

Name	Value	Type
MachineId	Machine1	String
SessionDate	23/04/2024	String
Phase	Concentric	String
PhaseTime	0.9485	Number
Distance	17.45	Number
Load	50	Number
Rep	3	Number
Set	2	Number

Table 1: A single random item in the "WorkoutData" table.

For the "VelocityData" table, we used the same partition key, formatted as `USER#<UUID>`, to ensure consistency in the database. The sort key is composed of several attributes to follow the same format as our "WorkoutData" table:

```
MACHINE#<MachineID>#SESSION#<SessionDate>#SET#<SetNumber>
```

For each set present in the "WorkoutData" table, we store a data point containing the calculated intensity and 1RM for the given set.

Name	Value	Type
MachineId	Machine1	String
SessionDate	23/04/2024	String
Intensity	100	Number
OneRepMax	100	Number
Set	2	Number

Table 2: Example of a single random item in the "VelocityData" table.

5.6 Cloud Applications

In this project the cloud application, being the user interface, was developed as a PWA. In this, the user can see insights based on their training. The chosen technology stack for the cloud application layer focuses on both performance and user experience. This ensures that the application is efficient and responsive across all devices. The application is built using Next.js, a React framework that enhances the user experience in various ways[33]. We chose to use this framework because we were already familiar with it. A PWA takes advantage of modern web capabilities to deliver an app-like experience as if it were written in a native mobile programming language[20]. Opting for a PWA over a traditional native application was a strategic

decision to avoid having to deploy the application through Google Play and Apple App Store. This allows for an easier and faster deployment and also lets us update the application directly via the web. While it is designed heavily with a mobile-first approach, the PWA still maintains all functionalities across all platforms, including desktops.

Typescript is the chosen programming language for developing the PWA. It significantly reduces the potential for runtime errors and bugs with its very strict syntax. With limited time for development, this assisted in minimizing the time spent debugging code.

For styling, Tailwind CSS is chosen for its utility-first approach, which is where you use predefined classes to write CSS directly in the HTML. This all leads to less code, resulting in faster development[15].

To further improve the development, a newly released, reusable component library called shadcn/ui was utilized[23]. Copying and pasting premade high-quality and tailored components made the development easier and faster while also improving the overall UI.

For the choice of hosting our PWA, we chose to use Vercel. Vercel is made by the same creators that are behind Next.js. And therefore has 100% support for everything Next.js offers, providing a flawless integration with our application[40].

5.7 Designing The Architecture

After thorough consideration of the possibilities within cloud infrastructure, we chose AWS due to its comprehensive list of services, which aligned perfectly with our needs for this project, and easy integration into the rest of our architecture.

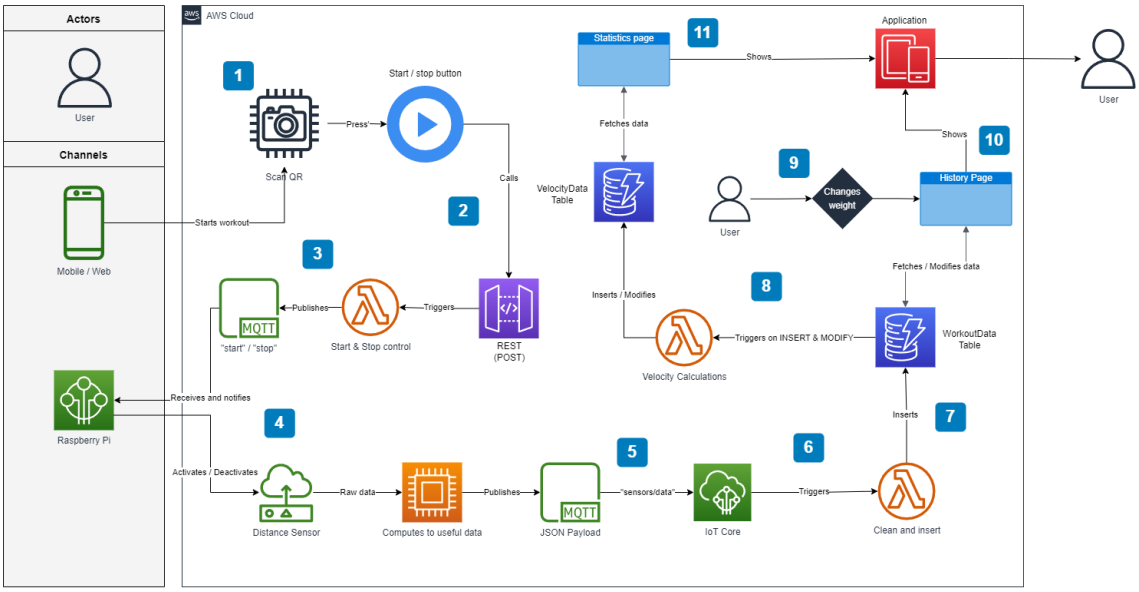


Figure 6: A diagram visualizing our system architecture.

The following list describes the flow of data and user interactions within the system architecture, corresponding to the numbers indicated in the architecture diagram:

1. Users initiate a workout session by scanning a QR code on a fitness machine using the application.
2. After scanning the QR code, the application displays a 'start' button. Pressing this button invokes an API Gateway endpoint, which triggers an AWS Lambda function.
3. The Lambda function sends a 'start' command via MQTT to the Raspberry Pi, signaling the start of data collection. Pressing 'stop' sends a 'stop' command to conclude data collection.
4. The Raspberry Pi collects raw sensor data and computes it into a structured JSON payload containing details on distance, time, phase, set number, repetition, and other workout parameters.
5. The structured data is sent from the Raspberry Pi to AWS IoT Core using the MQTT protocol.
6. An SQL rule in AWS IoT Core, `SELECT * FROM "sensors/data"`, triggers a Lambda function whenever a message is published to the "sensors/data" topic.
7. This Lambda function cleans the incoming data to fit the DynamoDB schema and inserts it into the table.

8. Upon insertion of new data into DynamoDB, the second Lambda function calculates intensity based on the distance and time values in the data.

9. If the user inputs or changes the weight used in a set, the corresponding data in DynamoDB is updated. This update triggers the second Lambda function to calculate the 1RM based on the weight input.

10. The history page of the application allows users to select a date from a calendar to fetch workout data. Detailed workout information, including set and individual repetition details containing phase and duration, is retrieved, cached, and displayed from AWS DynamoDB.

11. The statistics page of the application presents aggregate workout data. Users can view average intensity, total workouts, and 1RM for each machine, along with a graphical representation of their progress over time.

In figure 6 a diagram displaying our system architecture is shown. It illustrates all the different parts of our system and shows how they interact with each other.

6 Process

Throughout this project, our product has undergone an evolution that is driven by an iterative approach to accommodate changing requirements, different findings, and user feedback. This iterative process has allowed us to adapt and improve in each iteration of development. This section will give an overview of the development process, describing the milestones achieved in each iteration.

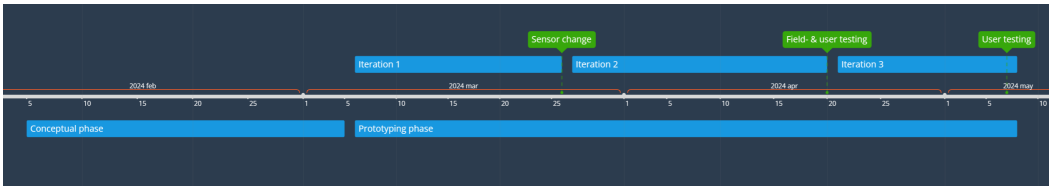


Figure 7: Project Timeline.

In the timeline of the project, we have undergone three iterations, each representing a distinct period in our journey toward a final prototype. In figure 7, it is depicted when each iteration, phase, and major testing took place. We will go more in-depth in the following subsections, and explore each iteration in detail, outlining what the product looked like at the time and what technologies it consisted of.

The project began with the conceptual phase. The decisions made here revolved primarily around what hardware and technology to use. In terms of hardware, we chose to use a Raspberry Pi, with Raspberry Pi OS, over its competitors the ESP32 and Arduino. This was because of its superior processing capabilities, which allowed us to compute the raw data before sending it to our cloud services. To get the desired data from a user performing an exercise we chose to use an accelerometer, based on a recommendation from our supervisor, which is a sensor that measures acceleration forces. For the cloud services needed, we decided to use AWS primarily due to their easy integration with IoT devices, and the allowance of good interoperability. AWS met all our criteria and provided a free tier, enabling us to start our project promptly and without cost. Of the available services we chose to use AWS IoT Core for communication with the hardware, AWS Lambda for our cloud computing, and AWS DynamoDB for our database. With these choices being made we were able to transition into the prototype phase and the first iteration.

6.1 Iteration 1

We started working on the product in the first iteration, right after the conceptual phase. This marked the beginning of the prototype phase. In this iteration's development, we ensured each layer in our technology stack worked by itself. This

means that we set up our Raspberry Pi to read sensor data from the accelerometer. AWS Lambda had some simple computation that could calculate phases and repetitions on mocked sensor data, and a single database table was set up with a relational structure, by nesting each item's attributes. Lastly, we had a simple frontend template. By ensuring that all layers were up and running by themselves, we had the necessary building blocks for the following iterations.

We also explored various options for enabling users to access workout equipment effortlessly. Among the considerations, we opted for an NFC tag solution due to its simplicity and widespread familiarity amongst users. The NFC technology, which is commonly used in platforms like Apple Wallet[37], would offer a compact and user-friendly method for users to interact with the fitness equipment.

For our frontend technology stack, we chose Next.js paired with Tailwind CSS and TypeScript due to their combined benefits in terms of performance and maintainability. These technologies work well together, enabling us to create a PWA. For the cloud computing, we chose to utilize Python. The reasoning for this was based on the fact that we did not expect any heavy computing, but rather multiple small scripts. Furthermore, we knew that we wanted real-time computation and therefore had to use a quick and efficient language.

The reason we chose to create a PWA instead of a normal website or a classic mobile application, was because we wanted the user to be able to access data from his workouts both on his phone and his computer. By creating a PWA it is possible to make the application look and feel like a native application for both mobile as well as for desktop. Even though we created a PWA, our intention was always for the user to interact with the IoT device exclusively through their smartphone.

Wireframing

Early in the development phase, we created a low-fidelity wireframe to use as a reference for the design and layout of our UI. The wireframe served as a blueprint for the frontend development to ensure that all the essential features and UI designs were considered going forward. As the first iteration progressed, new technical insights allowed us to refine the UI with small increases in the right direction. The wireframe can be seen in appendix C.

Figure 8: Screenshot of statistics page from iteration 1 of the application.

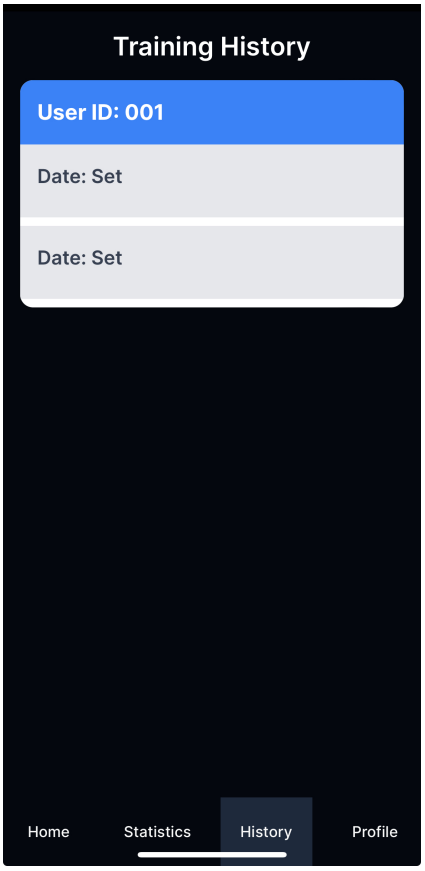


Figure 9: Screenshot of history page from iteration 1 of the application.

The frontend was very simple at this iteration, with 4 pages at the time. The page where the user starts a workout did not have a way of scanning and choosing a machine, and could not communicate with the hardware. The statistics page lacked functionality and design, only displaying hardcoded data instead of dynamic content, as depicted in figure 8. Similarly, the history page lacked usability and functionality, rendering it not user-friendly and without real data, as illustrated in figure 9. The profile page, accessible from the navigation bar, was just a blank page without any content.

6.1.1 Testing

For the first iteration, we only tested the AWS Lambda computation code. This was because the frontend, as well as the Raspberry Pi, were not developed enough to test, and the system as a whole was not connected rendering it impossible to perform system tests.

Mock Testing

We only conducted minimal mock testing on our AWS Lambda computation to ensure that it calculated somewhat correct phases and repetitions. We created mocked sensor data, simulating what we expected real-life sensor data to be. This mock data was then parsed into our function in various tests, ensuring the outputs of the various functions were equal to the expected ones.

```
[
  {
    y_axis: 0,
    x_axis: 0,
    time: 0.0,
  },
  {
    y_axis: 10,
    x_axis: 0,
    time: 2.0,
  },
  {
    y_axis: 20,
    x_axis: 0,
    time: 4.0,
  },
  {
    y_axis: 10,
    x_axis: 0,
    time: 6.0,
  },
  {
    y_axis: 0,
    x_axis: 0,
    time: 8.0,
  }
]
```

Figure 10: Mock sensor data.

In figure 10 an example of the used mock sensor data is depicted. We experienced that the overall intended functionality of the code worked.

6.2 Iteration 2

The second iteration was mostly focused on connecting the different layers and creating a working prototype, ready for user tests at the end of the iteration. We researched other suitable sensors to replace the accelerometer and ended up

choosing an ultrasonic sensor since its distance tracking range was in line with the distance the weight on a pin-loaded fitness machine travels.

Afterwards, we integrated this sensor onto our Raspberry Pi, making some improvements to our cloud code along the way to support the change. We had chosen to use NFC technology to be the way a user connects to the fitness machine, but we opted for a simpler solution which was a QR scanner instead.

The AWS Lambda code was refactored to solve some edge case issues that had been discovered, which allowed the code to provide a more precise measurement of the users' movement. In this iteration, our AWS DynamoDB table was also refactored to use a non-relational table structure, as it should be since we're using AWS DynamoDB.

In this iteration, we also implemented another AWS Lambda function to perform more computation, based on the VBT principles. This was done to provide more in-depth, and interesting feedback to the user, based on their current exercise. The provided feedback was the average calculated intensity of a set and a theoretical 1RM load for the specific exercise. To store this we created another table in our database, also structured in a non-relational style. However, this second table caused the overall structure of our database to be relational, since the two tables shared the same partition key that was a unique user identification.

We noticed some issues regarding imprecise AWS Lambda calculations, which we thought were due to the delay between the MC and the first Lambda function. To combat this, we choose to move the part of the code containing the movement calculation away from AWS Lambda and into the Raspberry Pi instead. This was also done to minimize the amount of communication needed and to utilize the computation power of the Raspberry Pi since it is the reason we chose to use this MC to begin with.

For the automatic execution of our AWS Lambda functions, we used various triggers. This was done to activate the correct function based on certain rules or events. We set up a trigger on one of the AWS Lambda functions that activated it when the Raspberry Pi sent data to a certain MQTT topic. For the VBT computation, another trigger was linked to the WorkoutData table and activated the function on all table events, for example when an item was inserted or modified. By doing this, the cloud computation was fully automatic.

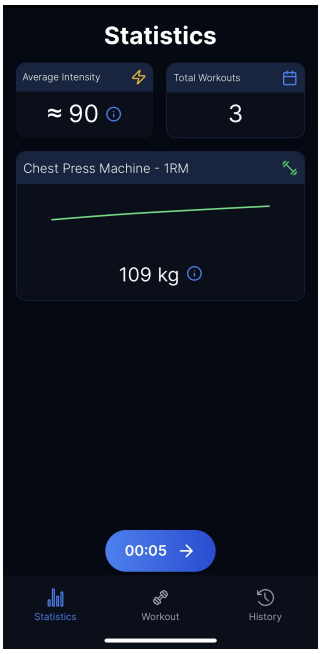


Figure 11: Screenshot of statistics page from iteration 2 of the application.

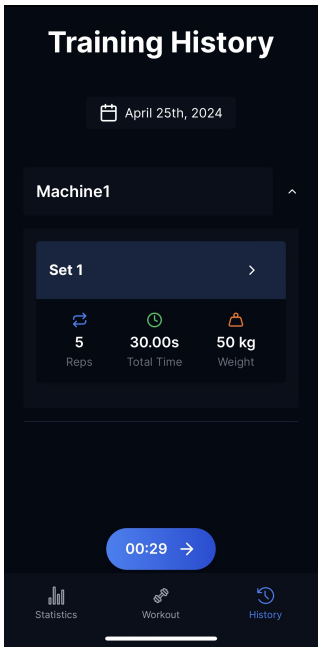


Figure 12: Screenshot of history page from iteration 2 of the application.

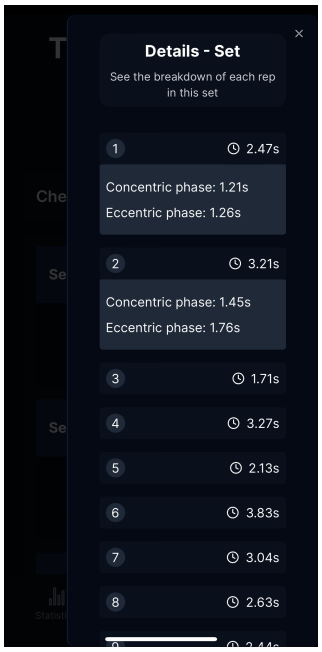


Figure 13: Screenshot of detail sheet on the history page from iteration 2 of the application.

The new computational features displayed in the application can be seen in figure 11. Not only does it display the average intensity, but also the total amount of workouts you have done. In addition to displaying the 1RM for the user, the user can see a graph of the progress in their 1RM for specific exercises by sliding since it is implemented as a carousel. The history page in figure 12 has been updated, so it gives a comprehensive overview of each set performed on a machine, including the number of repetitions, the total time it took to perform those repetitions and the load. As an additional feature that can be seen in figure 13, there is a sheet that shows additional content of the set. The content shown is a more detailed look at each repetition from the chosen set, providing the total time each repetition took along with each phase and phase time for each repetition.

6.2.1 Testing

In the second iteration, the big improvements to the system as a whole allowed us to begin testing more thoroughly on each layer in our system. As well as ending the iteration with multiple field tests and a user test on our application. This was done to obtain valuable insights and user feedback, on where our product was still lacking or behaving unexpectedly, so this could be addressed in the next iteration.

Mock Tests

We continued to use mock tests throughout the entirety of this iteration. To ensure that each layer in our system independently performed as expected. This included the computation on the Raspberry Pi, the VBT computation as well as the frontend.

```
[
{
    "is_first_entry": true,
    "distance": 0,
    "time": 0.0,
    "machineid": "machine1",
    "is_last_entry": false
},
{
    "is_first_entry": false,
    "distance": 25,
    "time": 1.0,
    "machineid": "machine1",
    "is_last_entry": false
},
{
    "is_first_entry": false,
    "distance": 50,
    "time": 2.0,
    "machineid": "machine1",
    "is_last_entry": false
},
{
    "is_first_entry": false,
    "distance": 25,
    "time": 3.0,
    "machineid": "machine1",
    "is_last_entry": false
},
{
    "is_first_entry": false,
    "distance": 0,
    "time": 4.0,
    "machineid": "machine1",
    "is_last_entry": true
},
]

```

Figure 14: Mock data used to test the Raspberry Pi computation.

In figure 14, the mock data we used to test our Raspberry Pi computation is shown.

The data simulates real-life data from our sensor and was used to ensure that our computation calculated phases and repetitions as intended. The data consist of multiple data points simulating sensor readings.

In appendix A figures 23, 24 & 25, the mock data used to test our VBT computation is depicted. This mock data was used to simulate the data streams coming from our database trigger, to test the functionality and calculation of our VBT computation.

```
{
    USER#<UUID>: USER#001
    MACHINE#<MachineID>#SESSION#<SessionDate>#SET#<SetNumber>#REP#<
    RepNumber>#PHASE#<Phase>: MACHINE#Machine1#SESSION#10/04/2024#SET
    #1#REP#1#PHASE#Concentric
    Distance: 33.91
    Load: 0
    MachineId: Machine1
    Phase: Concentric
    PhaseTime: 1.21
    Rep: 1
    SessionDate: 10/04/2024
    Set: 1
}
```

Figure 15: Mock database data for the frontend mock testing.

In Figure 15, an example of mock data simulating computed data in the database is depicted. This mock data was used to simulate real data that would be in the database that the application would fetch. Through mock testing, the code we developed was scrutinized to make sure its ability to display the right data was working correctly.

Field Testing

In this project, all field testing performed was done by ourselves and not a designated user. This was done to carry out the tests quickly and without the need for an external user. To ensure that no bias was present in these experiments, the tests only tested the tracking and calculation done by our prototype. This type of testing was done to guarantee the processed data were also correct in a real-life setting.

Field Test 1: Testing For Repetition Accuracy

For the first of our conducted field tests, we set up the sensor on a row machine. We then performed exactly 2 sets of 5 repetitions each on the machine. To ascertain that the phases, phase times, and repetitions are all calculated properly, an observer

will observe the test and time each phase with a stopwatch. The testing criteria for whether the test is successful or a failure is:

- The repetitions performed for each set correlate to the number of repetitions presented in the database and application.
- The phases tracked in the database correlate with the phases performed.
- The phase times in the database and application are within 0.25 seconds of the observer's manually tracked phase times.

Field Test 2: Testing Level of Intensity

For our second field test, we used the same setup as for the first. Here we will perform 1 set until failure, and 1 set until moderate fatigue on a machine. This means that the number of repetitions does not matter. The first set will stop when the subject physically can not perform more repetitions (RPE 10). The second set will stop when he experiences what he perceives as moderate fatigue (RPE 7.5)[34]. The testing criteria for whether this test is successful or a failure is that the level of intensity displayed on the application is within 10% of the perceived intensity when we performed the exercise.

Field Test Results

USER#...	MACHI...	Distance	Load	MachineId	Phase	PhaseTime	Rep	SessionDate	Set
USER#001	MACHINE...	5.03	0	Machine1	Concentric	2.5274514...	1	25/04/2024	1
USER#001	MACHINE...	82.02	0	Machine1	Eccentric	7.7813999...	1	25/04/2024	1
USER#001	MACHINE...	17.28	0	Machine1	Concentric	1.7136128...	2	25/04/2024	1
USER#001	MACHINE...	83.01	0	Machine1	Eccentric	1.3829918...	2	25/04/2024	1
USER#001	MACHINE...	19.64	0	Machine1	Concentric	1.8333993...	3	25/04/2024	1
USER#001	MACHINE...	96.02	0	Machine1	Eccentric	2.3415725...	3	25/04/2024	1
USER#001	MACHINE...	7.13	0	Machine1	Concentric	1.7354735...	4	25/04/2024	1
USER#001	MACHINE...	92.57	0	Machine1	Eccentric	1.6581126...	4	25/04/2024	1
USER#001	MACHINE...	10.9	0	Machine1	Concentric	6.9654940...	5	25/04/2024	1
USER#001	MACHINE...	96.03	0	Machine1	Eccentric	2.3166163...	5	25/04/2024	1
USER#001	MACHINE...	18.09	0	Machine1	Concentric	2.4115901...	1	25/04/2024	2
USER#001	MACHINE...	91.01	0	Machine1	Eccentric	8.6202301...	1	25/04/2024	2
USER#001	MACHINE...	13.19	0	Machine1	Concentric	2.3693437...	2	25/04/2024	2
USER#001	MACHINE...	94.48	0	Machine1	Eccentric	2.2499883...	2	25/04/2024	2
USER#001	MACHINE...	9.17	0	Machine1	Concentric	2.2403685...	3	25/04/2024	2
USER#001	MACHINE...	81.77	0	Machine1	Eccentric	2.0380872...	3	25/04/2024	2
USER#001	MACHINE...	12.31	0	Machine1	Concentric	2.4186913...	4	25/04/2024	2
USER#001	MACHINE...	81.49	0	Machine1	Eccentric	1.5468925...	4	25/04/2024	2
USER#001	MACHINE...	5.37	0	Machine1	Concentric	7.5764365...	5	25/04/2024	2
USER#001	MACHINE...	90.82	0	Machine1	Eccentric	2.2063914...	5	25/04/2024	2

Figure 16: Test 1 database data.

Set 1 phase	Set 1 time	Set 2 phase	Set 2 time
Concentric:	2.5	Concentric:	2.2
Eccentric:	2.6	Eccentric:	2.8
Concentric:	1.9	Concentric:	2.5
Eccentric:	1.5	Eccentric:	1.9
Concentric:	1.6	Concentric:	2.1
Eccentric:	2.3	Eccentric:	2.2
Concentric:	1.5	Concentric:	2.4
Eccentric:	1.9	Eccentric:	1.7
Concentric:	2.7	Concentric:	2.7
Eccentric:	2.5	Eccentric:	2.3

Table 3: Observed data for test 1.

For the results of the first field test, we expected to see exactly 2 sets of 5 repetitions in the database, as well as correctly ordered phases and phase times within 0.25 seconds of the actual phase time. When we compared the database data in figure 16, with the observed notations in table 3, we could see that the sets and repetitions were calculated correctly. However, the phases were inversed, meaning that both sets started with an eccentric phase instead of the observed concentric phase. Finally, almost all the phase times were also calculated correctly to be within the 0.25 second margin, except for the first and last phases of each set. From this test, we found that these were tracked from the time that the subject pressed the start and stop buttons in the application, and not from the beginning and end of the movement.

For the second test, we expected to see a calculated intensity within 10% of the subject's perceived intensity. After the first set was finished an RPE of 10 was described. In other words, a perceived intensity of around 100%. For the second set, an RPE of 6 was described, meaning an intensity of 60%.

USER#...	MACHINE#<MachineID>#SESS...	Intensity	MachineId	OneRep...	SessionDate	SetNumber
USER#001	USER#001#MACHINE#Machine1#...	156	Machine1	0	25/04/2024	1
USER#001	USER#001#MACHINE#Machine1#...	256	Machine1	0	25/04/2024	2

Figure 17: Test 2 database data.

As shown in the database in figure 17, the calculated intensity for the sets were 156% and 256%. Since the described intensity was around 100% and 60%, it was clear that our intensity calculation was incorrect. Both because the values did not correspond with the described values, and because the intensity should never be able to exceed 100%.

User Test: PWA

For the application user test, the user will be asked to open and navigate the application by himself without any help, whilst describing what he is doing and experiencing. The criteria for this test to pass is a positive response from the user, along with a SUS score of at least 68. Describing an intuitive design and easy navigation around our application and its features.

User Test Observations & Results

The full transcript in Danish of the user test can be seen in appendix E figure 59. Based on this it is obvious that the user could navigate the PWA by himself, and find most of the features. The only thing the user did not understand immediately was the intensity and 1RM fields in the statistics tab. However, the information buttons provided, helped him understand what these fields meant. There were still some features in the application that the user did not find by himself, those being the swipeable cards of 1RM's on the statistics page and the date picker and "set weight" feature on the history page. After the test was over, we also conducted a SUS survey on the user, this can be seen in appendix E figure 60. Here, the PWA scored 80 out of 100, which places it in the second highest percentile "excellent" 18, despite the small issues described in this test.

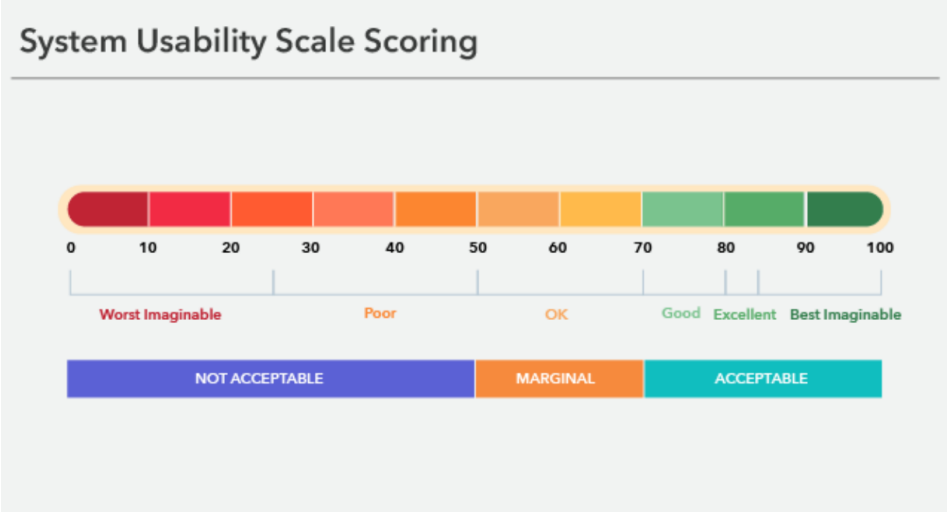


Figure 18: System usability scale scoring[22].

.

Getting these overall positive comments from the user, combined with the high SUS score. We can safely assume that the first user test passes the criteria being a SUS score of at least 68 along with positive feedback from the user.

6.3 Iteration 3

Based on the valuable insights gained from our field tests and user tests we initiated the third and final iteration of the project creating a final prototype. Here the focus was to correct the errors and bugs within our product, as well as to improve the UI based on the user feedback that we found from these tests.

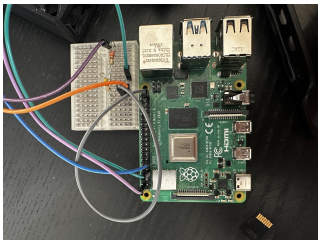


Figure 19: Iteration 2 Raspberry Pi.

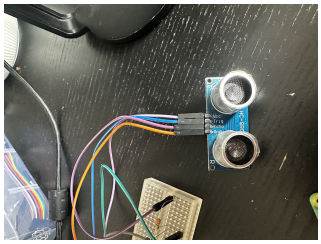


Figure 20: Iteration 2 ultrasonic sensor.

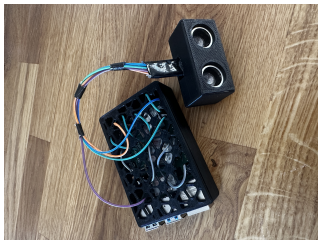


Figure 21: Iteration 3 all components.

Significant changes were made to the hardware setup to make it more compact and easy to configure. The Raspberry Pi and the ultrasonic sensor were previously connected via a breadboard, jumper wires, and resistors as seen in 19 & 20. In this iteration, the sensor and Raspberry Pi were encased in custom designed 3D-printed housings as seen in figure 21. The new case on the Raspberry Pi featured a honeycomb design at the top. This provided a more organized passage for the wires. Similarly, the case for the sensor was designed to protect the sensor and made sure that it could stay completely still as it was very sensitive. The use of a breadboard was phased out in favor of Wago connectors, which were placed inside the Raspberry Pi case. This change not only minimized the number of components we needed but also reduced the risk of loose connections that could have occurred with the breadboard.

For the data processing on the Raspberry Pi, there were two clear issues at hand. Those being that the first and last phases were calculated wrong, and all the phases were inverse. The latter of these issues originated from the way that we calculated phase changes. Since a phase is logged after a phase change is detected, the current phase theoretically did not belong to the logged phase. Luckily, this was a quick fix where we simply reversed all phases in the code, ensuring that they were logged as they were intended to be.

To handle the first and last phases of a set being tracked incorrectly, we implemented two changes. For the first phase, a timeout was implemented so the sensor did not start immediately after the user pressed the start button. The change introduced a countdown in the application. This provided a better synchronization between the sensor and the start of the exercise, resulting in a more precise

first phase tracking. For the last phase, we implemented a running average that tracked the average of all eccentric phases' maximum distance. This average was then used to temporarily log a phase each time the distance was within a margin of 5 centimeters of the average threshold. This meant that when the user pressed the stop button, instead of returning the distance and time of this action, the code returned the logged phase instead. This change enabled the code to correctly track the last phase.

As for the VBT formulas, we did not make any big changes to the calculations. Based on further mock testing we saw that they did calculate correctly, and the issue we saw in the second field test of iteration two stemmed from the way the trigger parsed database items to the AWS Lambda function. The code did receive a small quality change in this iteration. Instead of having a hardcoded velocity threshold, the refactoring enabled the code to recognize the machine identification of a set and use an appropriate velocity threshold based on this. This change provided a more precise intensity calculation for different machines since they now have various velocity thresholds instead of a singular threshold for all machines.

For the frontend, the UI only received some small changes. These changes were based on the user feedback gained from the user test done in the second iteration. We added arrows on the 1RM cards to indicate that they are swipeable. Highlighting was added for the dates in the calendar containing training data on the history page. The text for the changeable weight on each set was underlined to indicate the fact that it was pressable. Finally, we added a visual countdown when starting a set, to support proper first phase tracking on the Raspberry Pi. The whole application can be seen in appendix D.

6.3.1 Testing

At the end of this iteration in the prototype phase, we chose to replicate the first field test from the second iteration which yielded faulty results, as well as another user test on the application.

Mock Testing

In this iteration, we conducted minimal mock testing and focused on replicating the tests failing in iteration two. We conducted thorough mock testing on the intensity calculation which ultimately led us to the conclusion that the calculation itself was correct, but the way that the trigger parsed database items was the issue causing the wrong calculations. Based on this we chose not to reproduce the intensity field test from iteration two in this iteration.

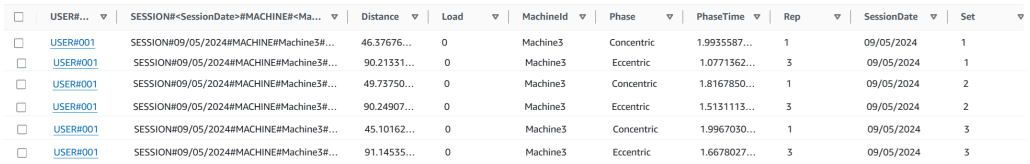
Field Test: Testing For Repetition Accuracy

In the second iteration, after performing the first field test, we found that we had an issue with the first and last phases of each set being tracked incorrectly.

As described the tracking computation was refactored in this iteration and we therefore replicated the same field test to see if these changes correctly fixed the issues. The setup for this test was as identical as possible to the field test in the second iteration, as seen in Appendix B.2, though this time the Raspberry Pi and the sensor were set up more securely on the row machine. Unlike the first test, we performed 3 sets of 3 repetitions on the machine. This was done to obtain more data to ascertain that the first and last phases of each set were correctly calculated. We used an observer with a stopwatch for this, as we did in the first test, however, he only tracked the first and last phase instead of all of the phases. The testing criteria became as such:

- The repetitions performed for each set correlate to the number of repetitions presented in the database and application.
- The phases tracked in the database correlate with the phases performed.
- The phase times of the first and last phase in the database and application are within 0.25 seconds of the observer's manually tracked phase times.

Field Test Result



USER#...	SESSION#<SessionDate>#MACHINE#<Ma...	Distance	Load	MachineId	Phase	PhaseTime	Rep	SessionDate	Set
USER#001	SESSION#09/05/2024#MACHINE#Machine3#...	46.37676...	0	Machine3	Concentric	1.9935587...	1	09/05/2024	1
USER#001	SESSION#09/05/2024#MACHINE#Machine3#...	90.21331...	0	Machine3	Eccentric	1.0771362...	3	09/05/2024	1
USER#001	SESSION#09/05/2024#MACHINE#Machine3#...	49.73750...	0	Machine3	Concentric	1.8167850...	1	09/05/2024	2
USER#001	SESSION#09/05/2024#MACHINE#Machine3#...	90.24907...	0	Machine3	Eccentric	1.5131113...	3	09/05/2024	2
USER#001	SESSION#09/05/2024#MACHINE#Machine3#...	45.10162...	0	Machine3	Concentric	1.9967030...	1	09/05/2024	3
USER#001	SESSION#09/05/2024#MACHINE#Machine3#...	91.14535...	0	Machine3	Eccentric	1.6678027...	3	09/05/2024	3

Figure 22: Field test data from the "WorkoutData" table.

First phase	First phase time	Last phase	Last phase time
Set 1 concentric:	2.1	Set 1 eccentric:	1.2
Set 2 concentric:	2.0	Set 2 eccentric:	1.5
Set 3 concentric:	1.8	Set 3 eccentric:	1.5

Table 4: Observed data for the field test.

By comparing the observation results in table 4 to the data present in the database seen in figure 22. It was clear that there was a big improvement from the test conducted in the second iteration. The results and database data were almost identical and certainly within the 0.25 second margin as specified in the criteria. Furthermore, both the repetitions and sets in the database were correlating with the ones specified for the test. This test passed all three test criteria successfully proving that the changes to the code were correct.

User Test: PWA

As in the second iteration, the PWA user test simply consisted of a user, navigating around our PWA with no help while he described what he was doing and what his experience was. Along with these descriptions and thoughts, a SUS survey was also conducted. The passing criteria for the test would be positive user feedback on the application like in the first application user test, along with a SUS score higher than the previous score, which was 80 out of 100.

User Test Observations & Results

The full transcript in Danish of the conducted test can be found in appendix E figure 61. Based on this transcript, it was obvious that the user could easily and intuitively navigate the PWA by himself as well as find and use the various features integrated, the most notable of these being the features that were not found in the previous user test. These features were the swipeable 1RM cards, the date picker, and the "set weight" feature. As in the test in the second iteration, a SUS survey was conducted again, which can be seen in appendix E figure 62. This test scored 87.5 out of 100, scoring this version of our PWA 7.5 points above the last one, placing it in the highest percentile being "best imaginable". The positive feedback from the user, paired with the higher SUS score, means that this test successfully fulfills all the criteria set.

7 Findings

After detailing each iteration of the development process in the project, we have gathered insights into the various challenges and successes as well as any necessary adaptations needed to refine our solution. Through iterative refinement, we have progressed towards a more robust and user-friendly prototype. The iterative approach enabled us to incorporate a design-test loop, where we used various testing methods over the entire project's lifecycle, to identify and solve current issues in our prototype. These findings serve as the bedrock on which we build our understanding of the evolution of the project, and will be described in this section.

7.1 Device Hard- and Software

In the first iteration, we explored the MPU6050 accelerometer as a potential solution for distance tracking. However, we quickly realized that an accelerometer was better at detecting orientation and acceleration rather than distance. We noticed this since sudden movements caused the data, which we thought to be distance data, to spike, and most of the time, it returned a constant number between -10 and 10 due to the force of gravity. We therefore started the second iteration by switching to the ultrasonic sensor that could properly measure distance.

During the second iteration of the project, we quickly realized that iPhones do not support the use of the NFC module on web applications but only with native ones. To make sure all platforms are supported, we changed the way a user would access a fitness machine to be via a QR code scanner instead. This change also proved to be simple to implement, still allowing the user to connect to a fitness machine in a simple manner through the application.

In our field tests, performed as the last thing in iteration two, we found that our data calculations were not performing as expected regarding the phase tracking. The first and last phases were incorrectly tracked from the moment that the user pressed the start and stop buttons on the app, resulting in wrong data for the first and last phases of each set. This could ultimately have impacted our intensity calculation as well. To combat this we changed the code to handle these edge cases correctly, resulting in correct data for the first and last phase of each set of the exercise as shown in the last iterations field test.

7.2 Cloud Services

In the second iteration, we found that we unpurposedly had structured our database table as a relational database, by having nested attributes for each item. This was the wrong practice for our chosen database, led to issues when fetching data in the frontend, and impacted the scalability and management of the database. This

prompted us to refactor to a non-relational table structure in the second iteration. While this refactor did make both the tables non-relational, our overall database structure was still relational, since both tables had the same partition key, being a `userID`, creating a relation between the two.

The way we set up our system architecture proved to be the correct way of achieving our goal. The only error with this architecture was the trigger for the VBT calculation. Based on the results of the second field test in the second iteration, we saw that this calculation was wrong. While the formulas used and the mathematics behind the calculation proved to be correct through mock testing, the way the data was parsed by the trigger was not. This resulted in incorrect intensity calculations, and since this was used to calculate the 1RM as well, this value was also wrong.

7.3 User Feedback

Based on the user tests we conducted in the second and third iteration, we found that our design for the application was user-friendly. The user feedback from both of these tests yielded positive results and they both had a high SUS score as well. There were however a few things that the user was confused about in the first test, and therefore we did make some small changes to the application in the third iteration. These changes led to the user having no confusion or questions in the last user test on the application along with a higher SUS score on our application.

8 Discussion

While certain elements of the project might have been approached differently with a broader scope, it is essential to recognize that the project's scope was defined by the objective outlined in the project statement. The developed system is meant to serve as a final prototype. In this section, we will explore potential enhancements and expansions of the product, and reflect on our approach and the methods used in development, based on our findings and future possibilities. We will also discuss some features that could contribute to a potential commercial scaling of the prototype.

8.1 Our Way of Working

The project never progressed into the final implementation phase. Due to the scope of the project, it was never a goal to reach this phase and create a fully-fledged solution, but rather to have a working prototype. Throughout the project, we employed an Agile framework and worked with an iterative development approach. The Agile framework provided flexibility to quickly adapt to any requirement changes and solve problems as they emerged. This was effective in making continuous improvements. This framework was complemented by our iterative approach, where the development was broken down into cycles, allowing for frequent reassessment and refinement based on user feedback and testing. These strategies helped us combat issues and challenges we came across, to help ensure that the system was operational and user-friendly. To get a deeper understanding of how the application performs at a usability level, it would have been better to collect more empirical evidence by testing it more thoroughly and on multiple subjects. Given more time, we could have implemented more iterations, allowing us to test the product more thoroughly, to help refine the prototype and come closer to a final solution. We should also have been consistent with system testing through iteration three instead of continuing isolated tests on each part of the system, to verify that they worked together. Specifically for the second iteration's field test, we identified issues with the intensity calculation. Instead of testing the new calculation on its own, we should have replicated the previously performed system tests that previously failed. This would have allowed us to identify the real issue faster, which was not the calculation, but instead the setup of our cloud services and triggers. This might have led us to solve this issue within the project's timespan.

8.2 Our Architecture

By developing an IoT device and application, we necessitated a reliable and scalable architecture, that was capable of real-time data processing and showing an analysis of an average person's performance. Since the velocity threshold values in the

calculation are based on a group of novice fitness lifters, the analysis of one's workout is not necessarily accurate for everyone, and would ultimately only be so, if the product calculated a personal velocity threshold for each user.

Other architectural decisions, such as the use of Raspberry Pi coupled with MQTT for reliable data transmission, and the integration of serverless computing and data storage with AWS Lambda and AWS DynamoDB, were important for minimizing latency and enhancing the responsiveness of our system. These choices reflect a deliberate focus on reliability and efficiency, which are both fundamental for IoT applications to succeed in dynamic environments like sports performance and fitness.

Despite this, the architecture of our cloud services also proved to be problematic based on the finding from the second field test from the second iteration, which tested the intensity calculation. Since we found from further mock testing that the calculations were correct, it relied on the way that data were parsed into the function. This meant that a set had to be parsed phase by phase, in chronological order, for the intensity and 1RM to be calculated correctly. The database trigger we used to run the lambda function containing the code, proved to parse the items in an incorrect order, leading to incorrect calculations.

To reflect on the scope of the project, a Raspberry Pi was theoretically the wrong choice of MC. An ESP32 would probably be able to handle the simple processing we are doing on the MC and is both smaller and cheaper than the Raspberry Pi. These factors would have made it cheaper for us to use and easier to install on a machine in the gym.

In reflection, while our choices helped us progress in the right direction, they also posed challenges in integration and initial setup, highlighting the importance of flexibility and adaptability in managing complex IoT systems. Challenges like incorrect data storage were quickly solved thanks to our choice of database, and addressing the incorrect sensor was straightforward by switching to an ultrasonic sensor. The Raspberry Pi's adaptability allowed for easy integration of the new sensor, simplifying adjustments in the device software layer. This examination not only underscores the successes but also the learning opportunities presented by the project.

8.3 Future Development

This section will discuss how and what we would develop and improve on the product if we were to make further development. This is based on both technical knowledge, the findings of our project as well as critical reflection.

In a commercial gym setting, scalability is a key consideration. Rather than requiring a Raspberry Pi for each machine, a more efficient approach would be to involve a central Raspberry Pi serving as a hub to manage multiple wireless sensors, distributed across various machines. This setup would be a superior approach for larger settings, as multiple sensors would send their data to a single hub, which would handle all data aggregation, processing, and communication with the cloud. Having multiple sensors sending data to a single Raspberry Pi, would also correctly utilize its computational power.

To enhance the precision of the VBT calculation, the various machine's velocity thresholds should be changed. A simple fix would be to have the application support a choice of whether the user is a novice, intermediate, or an experienced lifter. Yet to achieve the most correct and precise threshold, the product would have to create a personal velocity threshold for each machine based on a user's previous workouts.

An enhancement for the frontend could be the use of the WebSocket API. Unfortunately, due to time constraints and the scope of our project, we were unable to implement this feature. The WebSocket integration was intended to allow real-time database updates to be pushed directly to the frontend which would eliminate the need for the application to fetch data repeatedly. Reflecting on this aspect, it is clear that while the WebSocket integration remains an unfulfilled component and would have been beneficial, it is not critically necessary for our project's scope due to the amount of data involved. It would only be deemed relevant in a larger setting.
The integration of WebSockets would be a major feature to implement in future development. Implementing this would mean that the data fetching would only occur during the initial page load, thus reducing server load and data retrieval costs. The data fetched would then be cached, and future data would be sent to the application via the WebSocket connection. Each new data point would be added to the current cache and then the UI would immediately refresh. This leads to a more dynamic user experience, as the application receives updates immediately.

For the database structure, the correct implementation would be to merge the two tables into one. This is because, even though our tables are non-relational, the fact that we have two tables with the same partition key creates a relation between them. This would however make the current setup with the database trigger for the velocity calculation impossible since it would create an infinite loop with itself when updating the table. Therefore this computation trigger would need to be changed to instead be a trigger on the MQTT data topic that the Raspberry Pi is sending to. This would mean that the velocity calculation would be running on each data point in real-time, and then insert the combined workout and VBT data

in the table. By making this change, the incorrect intensity calculation would also be fixed, providing a correct analysis of the user's workout.

The biggest improvement possible for the sensor would be to both start and stop the tracking automatically. If this were to be implemented the sensor would be running continually, and then lock itself to a user when the QR code is scanned. It would start computing phases when a certain threshold for either distance or velocity is met, and then stop when the machine has not been moved for a certain amount of time. This would be a big improvement for the user since they would only have to scan a QR code and afterward leave the machine through the application.

An important factor that should be implemented if this product were to enter the fitness industry, would be authentication. To accommodate the GDPR laws present in Europe on data handling[25], if you were to implement authentication, you would have to also implement security measures. This is extremely important since you would have to handle users' personal information. Doing this would complement how the machine would be paired to a user and how the data for each user is stored. If authentication through login were implemented in the application a unique user identification would be present, which would then both be used as identification for the machine when the QR code is scanned and the partition key for the database.

9 Conclusion

How can IoT technologies be used to develop a product that integrates a device with an application for pin-loaded fitness machines, offering analysis of a user's workout based on the principles of velocity based training in a user-friendly way?

The integration of an IoT device with a pin-loaded fitness machine, utilizing the principles of VBT, has shown potential for providing an analysis of a user's workout. The project explored the feasibility of developing a system to collect, process, and display data, enabling performance tracking and feedback on the fitness machine. The result was a prototype that moved closer to a potential solution, offering users an intuitive way to analyze their performance.

By utilizing a Raspberry Pi for data collection and computation with an ultrasonic sensor for accurate measurements of exercise movements, the system tracked the phases, and repetitions in a set. The data was then transmitted to AWS for processing, utilizing AWS Lambda for real-time computation and AWS DynamoDB for scalable data storage. This architecture enabled the handling of workout data, providing users with feedback immediately after finishing a set on their exercise. This feedback contained performance metrics, such as intensity, 1RM, and phase timings.

Testing and iterative development played a bigger role in refining the system. Initial challenges, including inaccurate phase tracking and data handling issues, were addressed through agile development practices, leading to improvements in data accuracy and user experience. The positive feedback from users indicated that the final prototype is user-friendly and intuitive, confirming the project's success in meeting its usability goals.

This thesis describes how a prototype could be a step towards a possible solution by integrating an IoT technology stack with VBT principles to create a product that offers an analysis of a user's workout on a pin-loaded fitness machine. Overall, the project demonstrates the feasibility and potential benefits of integrating IoT technologies with fitness equipment.

Bibliography

[1] Eamonn Flanagan and Mladen Jovanović. "Researched Applications of Velocity Based Strength Training". In: *J. Australian Strength Cond.* 22 (Jan. 2014), pp. 58–69.

[2] Bryan J. Mann. "Velocity-Based Training in Football". In: *Strength and Conditioning Journal* 37 (6 2015), pp. 52–57. DOI: 10.1519/SSC.0000000000000177.

[3] Samantha Ingram. *The Thumb Zone: Designing For Mobile Users*. Sept. 2016. URL: https://www.smashingmagazine.com/2016/09/the-thumb-zone-designing-for-mobile-users/ (visited on 04/28/2024).

[4] Cedric X. Bryant. *The Evolution of Fitness Trends*. Dec. 2020. URL: https://health.usnews.com/health-news/blogs/eat-run/articles/the-evolution-of-fitness-trends (visited on 04/12/2024).

[5] Kristina Dineva and Tatiana Atanasova. "Design of Scalable IoT Architecture Based on AWS for Smart Livestock". In: *Animals (Basel)* (Sept. 2021). DOI: 10.3390/ani11092697.

[6] Liao Kaifang et al. "Effects of velocity based training vs. traditional 1RM percentage-based training on improving strength, jump, linear sprint and change of direction speed performance: A Systematic review with meta-analysis". In: *PLOS ONE* 16 (Nov. 2021), e0259790. DOI: 10.1371/journal.pone.0259790.

[7] Mariana WA. *Gestalt principles: How to apply them to a mobile app design*. July 2021. URL: https://bootcamp.uxdesign.cc/gestalt-principles-how-to-apply-them-to-a-mobile-app-design-f36dbaeb5372 (visited on 04/29/2024).

[8] Jacqueline DeMarco. *Nearly 70% of Americans Would Wear a Fitness Tracker/Smartwatch for Discounted Health Insurance*. Apr. 2022. URL: https://www.valuepenguin.com/fitness-tracker-smartwatch-health-survey (visited on 04/12/2024).

[9] Ritika Kumari. *What is Field Testing? Explained With Examples*. July 2023. URL: https://testsigma.com/blog/field-testing/ (visited on 04/29/2024).

[10] Brody Maxwell. *Velocity Based Training: A Beginner's Guide*. May 2023. URL: https://truerep.app/blog/vbt-a-beginners-guide (visited on 04/15/2024).

[11] NHLBI. *Study reveals wearable device trends among U.S. adults*. June 2023. URL: https://www.nhlbi.nih.gov/news/2023/study-reveals-wearable-device-trends-among-us-adults (visited on 04/12/2024).

[12] Owen Walker. *VELOCITY-BASED TRAINING*. Nov. 2023. URL: https://www.scienceforsport.com/velocity-based-training/ (visited on 04/15/2024).

[13] Carl Weinschenk. *Report: People Underestimate Number of IoT Devices in Their Homes*. Apr. 2023. URL: https://www.telecompetitor.com/report-people-underestimate-number-of-iot-devices-in-their-homes/ (visited on 04/20/2024).

[14] Berat Dinçkan. *ESP32 to AWS: Complete IoT Solution with IoT Core, DynamoDB, and Lambda Functions in Golang*. Jan. 2024. URL: https://dev.to/dinckan_berat/esp32-to-aws-complete-iot-solution-with-iot-core-dynamodb-and-lambda-functions-in-golang-5h2f (visited on 05/01/2024).

[15] *7 Reasons Why You Should Use Tailwind CSS Right Now*. URL: https://www.material-tailwind.com/blog/7-reasons-why-you-should-use-tailwind-css (visited on 01/05/2024).

[16] *Actuators in IoT*. URL: https://www.geeksforgeeks.org/actuators-in-iot/ (visited on 05/10/2024).

[17] JEAN-LUC AUFRANC. *Know the Differences between Raspberry Pi, Arduino, and ESP8266/ESP32*. URL: https://www.cnx-software.com/2020/03/24/know-the-differences-between-raspberry-pi-arduino-and-esp8266-esp32/ (visited on 04/20/2024).

[18] *AWS DynamoDB*. URL: https://aws.amazon.com/dynamodb/ (visited on 04/15/2024).

[19] *AWS Lambda*. URL: https://aws.amazon.com/lambda/ (visited on 04/15/2024).

[20] *Benefits of progressive web apps*. URL: https://www.divante.com/reports/pwabook/benefits-of-progressive-web-apps (visited on 01/05/2024).

[21] Richard Bevis. *7 Examples of IoT in Everyday Life*. URL: https://www.cbtnuggets.com/blog/technology/networking/seven-examples-of-iot-in-everyday-life (visited on 04/20/2024).

[22] Alana Chinn. *What's the System Usability Scale (SUS) & How Can You Use It?* URL: https://blog.hubspot.com/service/system-usability-scale-sus (visited on 05/08/2024).

[23] Nefe Emadamerho-Atori. *Introducing Shadcn UI: A reusable UI component collection*. URL: https://blog.logrocket.com/shadcn-ui-reusable-ui-component-collection/ (visited on 02/05/2024).

[24] *Estimate your one rep max with bar speed tracking data*. URL: https://www.vbtcoach.com/blog/1rm-and-velocity-based-training-vbt-a-complete-guide (visited on 04/20/2024).

[25] *General Data Protection Regulation*. URL: https://gdpr-info.eu/ (visited on 05/08/2024).

[26] *GymAware Homepage*. URL: https://gymaware.com/ (visited on 04/15/2024).

[27] Daniel Hindi. *What is Field Testing? Explained With Examples*. URL: https://buildfire.com/how-to-perform-user-testing-for-your-app/ (visited on 05/01/2024).

[28] *How To Use The System Usability Scale (SUS) To Evaluate The Usability Of Your Website*. URL: https://usabilitygeek.com/how-to-use-the-system-usability-scale-sus-to-evaluate-the-usability-of-your-website/ (visited on 05/06/2024).

[29] *IoT database*. URL: https://www.scylladb.com/glossary/iot-database/ (visited on 05/01/2024).

[30] Anni Junnila. *How IoT works - Part 3: data processing*. URL: https://trackinno.com/iot/how-iot-works-part-3-data-processing/ (visited on 05/01/2024).

[31] *Keep It Simple, Stupid (KISS)*. URL: https://www.interaction-design.org/literature/topics/keep-it-simple-stupid (visited on 04/28/2024).

[32] *MQTT: The Standard for IoT Messaging*. URL: https://mqtt.org (visited on 04/24/2024).

[33] Tharaka Romesh. *Why Choose Next.js - Top 5 Performance Benefits*. URL: https://cult.honeypot.io/reads/top-nextjs-performance-benefits/ (visited on 01/05/2024).

[34] *RPE in powerlifting: wha is RPE? + how to use rating of perceived exertion optimally*. URL: https://www.progressiverehabandstrength.com/articles/rpe-in-powerlifting-what-is-rpe (visited on 04/20/2024).

[35] *Software testing - mock testing*. URL: https://www.geeksforgeeks.org/software-testing-mock-testing/ (visited on 05/01/2024).

[36] *The 5 Layers of the IoT Technology Stack*. URL: https://danielelizalde.com/iot-primer/ (visited on 04/30/2024).

[37] *US Contactless Payment Statistics*. URL: https://finicalholdings.com/us-contactless-payment-statistics/ (visited on 04/19/2024).

[38] *Use Cases*. URL: https://mqtt.org/use-cases/ (visited on 04/24/2024).

[39] *VELOCITY BASED TRAINING – THE FUTURE OF STRENGTH TRAINING*. URL: https://www.flexstronger.com/velocity-based-training/ (visited on 04/15/2024).

[40] *Vercel Landingpage*. URL: https://vercel.com/ (visited on 05/10/2024).

[41] *What is NoSQL?* URL: https://aws.amazon.com/nosql/ (visited on 04/18/2024).

[42] *What is SIGFOX.* URL: https://www.electronics-notes.com/articles/connectivity/sigfox/what-is-sigfox-basics-m2m-iot.php (visited on 05/01/2024).

[43] *What is SQL?* URL: https://aws.amazon.com/what-is/sql/ (visited on 04/18/2024).

[44] *What is the IoT?* URL: https://www.ibm.com/topics/internet-of-things (visited on 04/15/2024).

[45] *What's the Difference Between MySQL and PostgreSQL?* URL: https://aws.amazon.com/compare/the-difference-between-mysql-vs-postgresql/ (visited on 04/18/2024).

[46] *Which MicroController is suitable for the Internet of Things?* URL: https://www.tutorialspoint.com/which-microcontroller-is-suitable-for-the-internet-of-things-iot (visited on 05/01/2024).

Appendices

A Mock Testing

```
{
  "Records": [
    {
      "eventID": "c4ca4238a0b923820dcc509a6f75849b",
      "eventName": "INSERT",
      "eventVersion": "1.1",
      "eventSource": "aws:dynamodb",
      "awsRegion": "us-east-1",
      "dynamodb": {
        "Keys": {
          "Id": {
            "N": "101"
          }
        },
        "NewImage": {
          "USER#<UUID>": {
            "S": "USER#001"
          },
          "MACHINE#<MachineID>#SESSION#<SessionDate>#SET#<SetNumber>#
    REP#<RepNumber>#PHASE#<Phase>": {
            "S": "MACHINE#machine1#SESSION#09/04/2024#SET#0#REP#1#
    PHASE#Concentric"
          },
          "Load": {
            "N": "0"
          },
          "Distance": {
            "N": "5.0"
          },
          "MachineId": {
            "S": "Machine1"
          },
          "Phase": {
            "S": "Concentric"
          },
          "Rep": {
            "N": "1"
          },
          "PhaseTime": {
            "N": "1"
          },
          "SessionDate": {
            "S": "09/04/2024"
          },
          "Set": {
            "N": "1"
          }
        },
```


Figure 23: Mock data for our VBT computation tests 1/3.

```
"ApproximateCreationDateTime": 1428537600,
        "SequenceNumber": "4421584500000000017450439091",
        "SizeBytes": 26,
        "StreamViewType": "NEW_AND_OLD_IMAGES"
      },
      "eventSourceARN": "arn:aws:dynamodb:us-east-1:123456789012:
    table/ExampleTableWithStream/stream/2015-06-27T00:48:05.899"
    },
    {
      "eventID": "c81e728d9d4c2f636f067f89cc14862c",
      "eventName": "REMOVE",
      "eventVersion": "1.1",
      "eventSource": "aws:dynamodb",
      "awsRegion": "us-east-1",
      "dynamodb": {
        "Keys": {
          "Id": {
            "N": "101"
          }
        },
        "NewImage": {
          "Message": {
            "S": "This item has changed"
          },
          "Id": {
            "N": "101"
          }
        },
        "OldImage": {
          "Message": {
            "S": "New item!"
          },
          "Id": {
            "N": "101"
          }
        },
```

Figure 24: Mock data for our VBT computation tests 2/3.

```
        "ApproximateCreationDateTime": 1428537600,
        "SequenceNumber": "44215845000000000017450439092",
        "SizeBytes": 59,
        "StreamViewType": "NEW_AND_OLD_IMAGES"
      },
      "eventSourceARN": "arn:aws:dynamodb:us-east-1:123456789012:
    table/ExampleTableWithStream/stream/2015-06-27T00:48:05.899"
    },
    {
      "eventID": "eccbc87e4b5ce2fe28308fd9f2a7baf3",
      "eventName": "REMOVE",
      "eventVersion": "1.1",
      "eventSource": "aws:dynamodb",
      "awsRegion": "us-east-1",
      "dynamodb": {
        "Keys": {
          "Id": {
            "N": "101"
          }
        },
        "OldImage": {
          "Message": {
            "S": "This item has changed"
          },
          "Id": {
            "N": "101"
          }
        },
        "ApproximateCreationDateTime": 1428537600,
        "SequenceNumber": "44215845000000000017450439093",
        "SizeBytes": 38,
        "StreamViewType": "NEW_AND_OLD_IMAGES"
      },
      "eventSourceARN": "arn:aws:dynamodb:us-east-1:123456789012:
    table/ExampleTableWithStream/stream/2015-06-27T00:48:05.899"
    }
  ]
}
```

Figure 25: Mock data for our VBT computation tests 3/3.

B Field Testing

B.1 Iteration 2

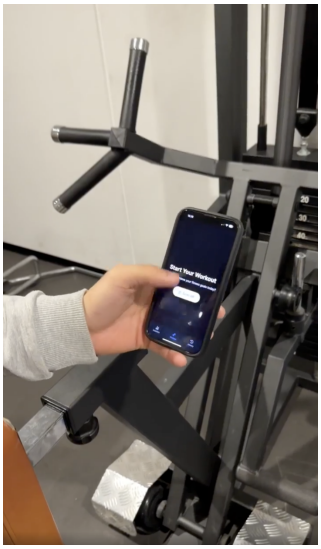


Figure 26: Starting the workout.

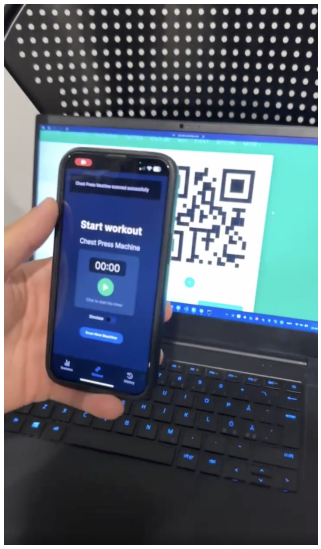


Figure 27: Scanning the machine.

Figure 28: Performing the exercise.

Figure 29: Stopping the workout.

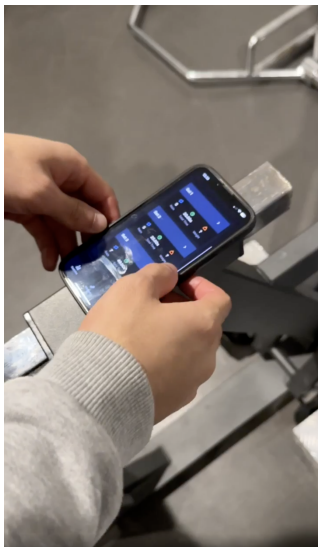


Figure 30: Showing the history page with the exercise performed.

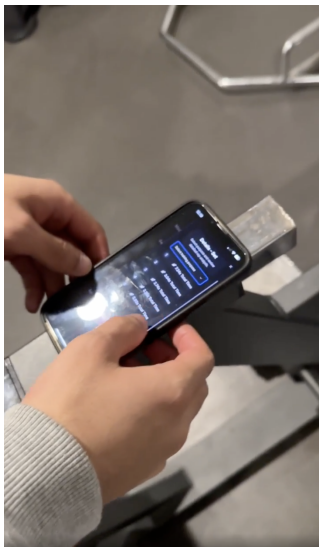


Figure 31: Showing the data for each repetition of the performed set.

B.2 Iteration 3

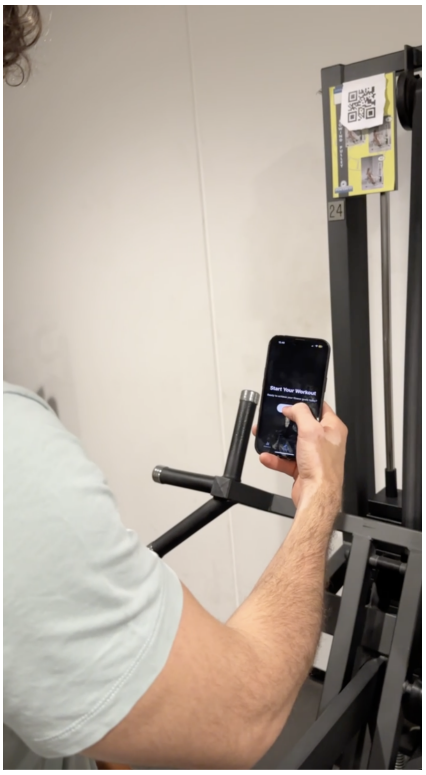


Figure 32: Starting the workout.

Figure 33: Scanning the machine.

Figure 34: Performing the exercise.

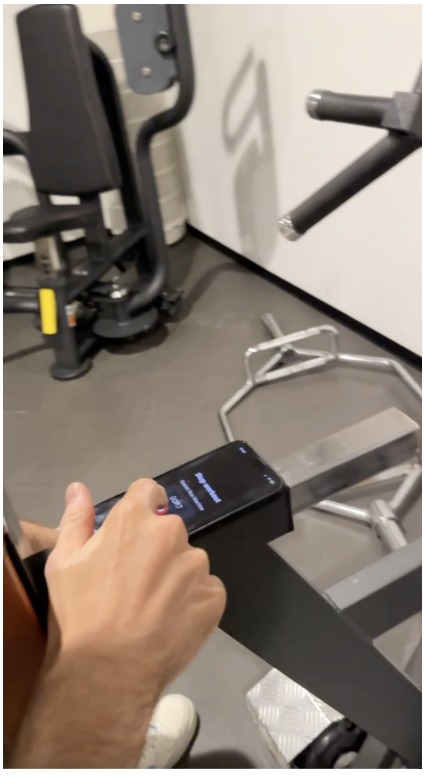


Figure 35: Stopping the workout.

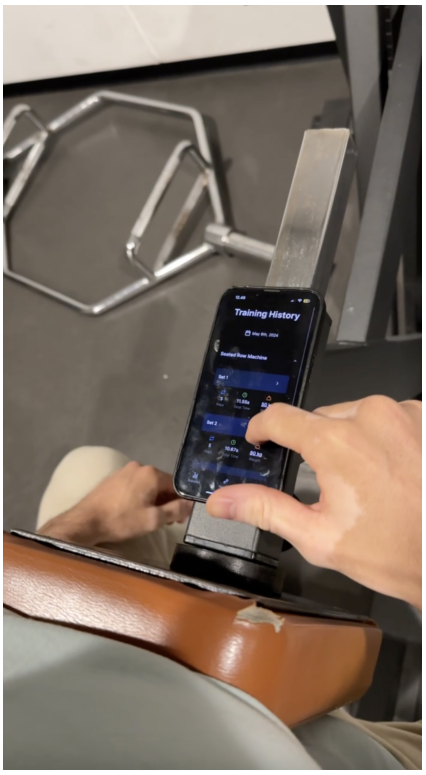


Figure 36: Showing the history page with the exercise performed.

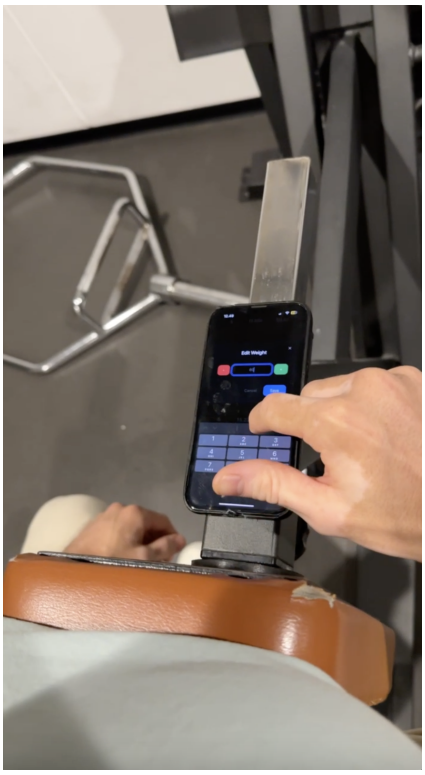


Figure 37: Adding the amount of weight lifted.

Figure 38: Showing the data for each repetition of the performed set.

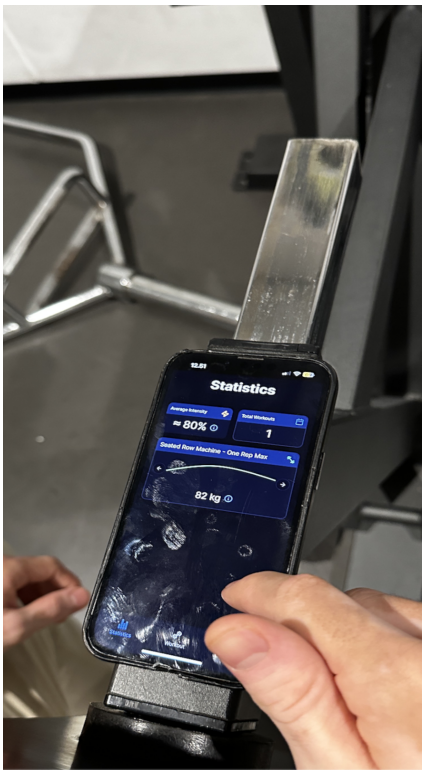


Figure 39: Showing the statistics page.

C Wireframe Images

Figure 40: Main page.

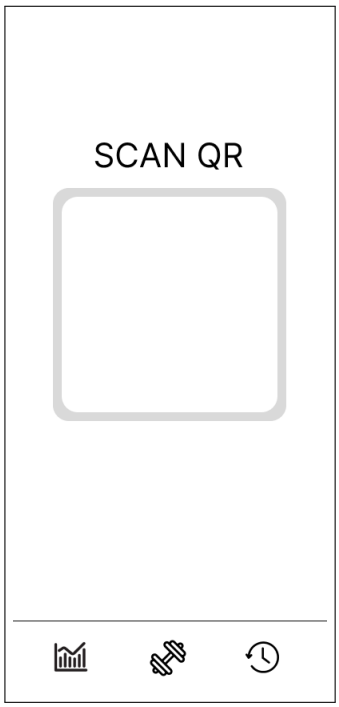


Figure 41: QR scanner page.

Figure 42: Start exercise page.

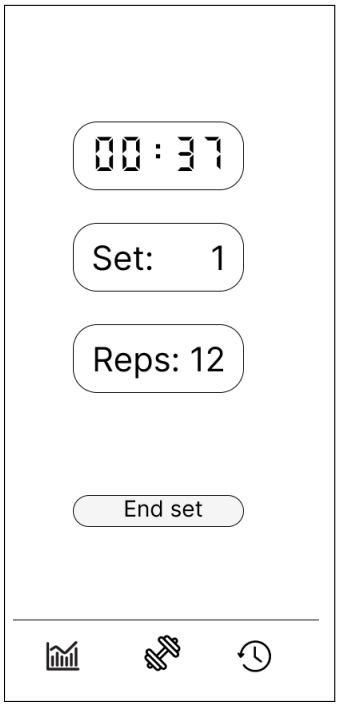


Figure 43: Active exercise page.

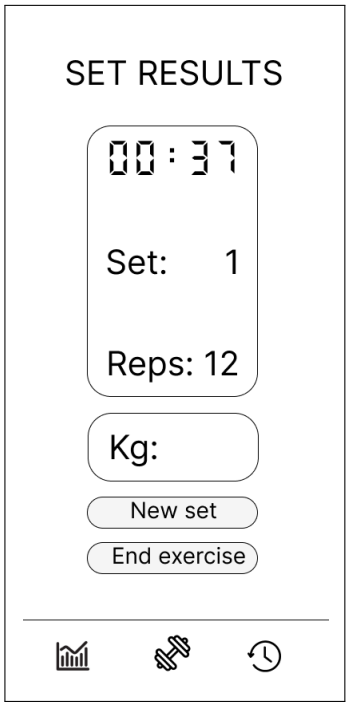


Figure 44: Exercise results page.

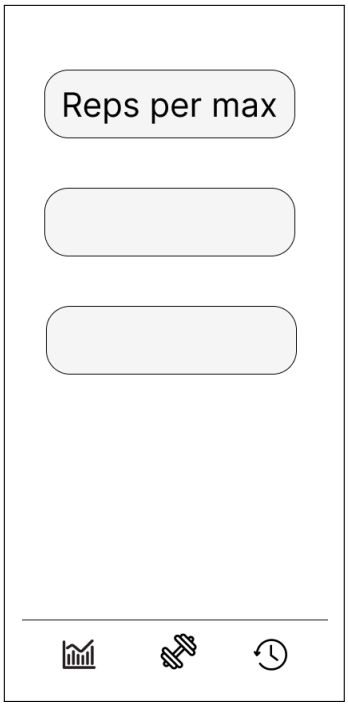


Figure 45: Specific machine for one rep max statistic page.

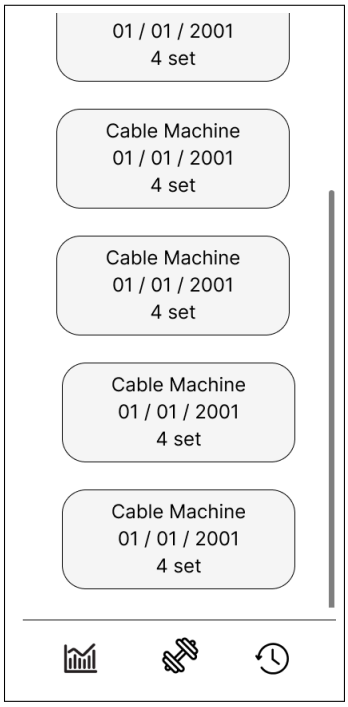


Figure 46: History page.

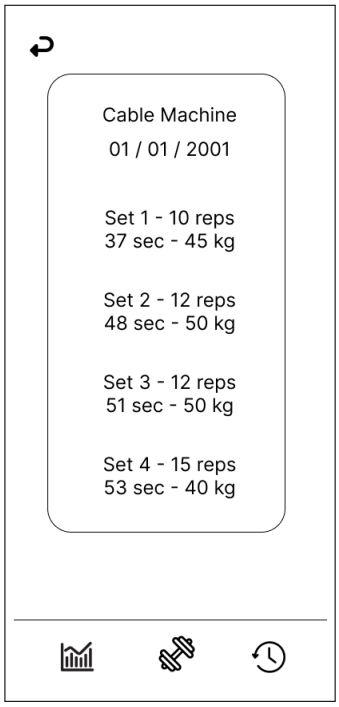


Figure 47: Specific machine history page.

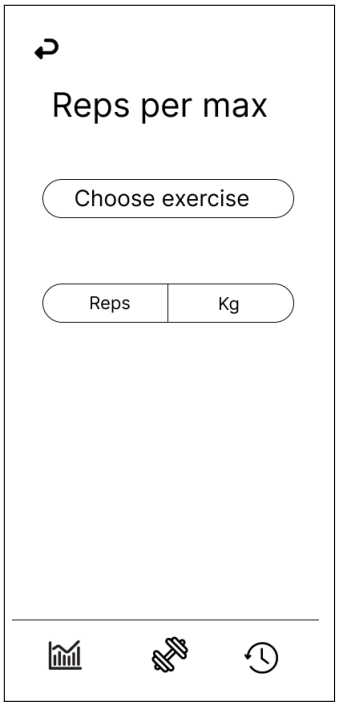


Figure 48: Choose the specific exercise for one rep max page.

D Final Application

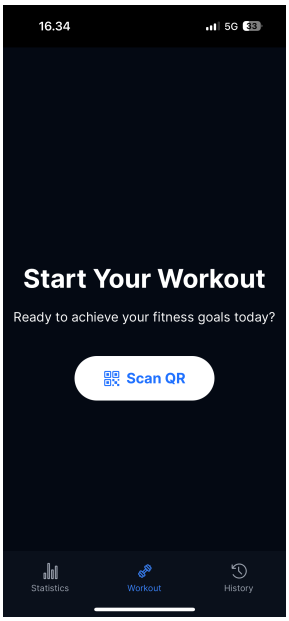


Figure 49: Main page - Workout page.

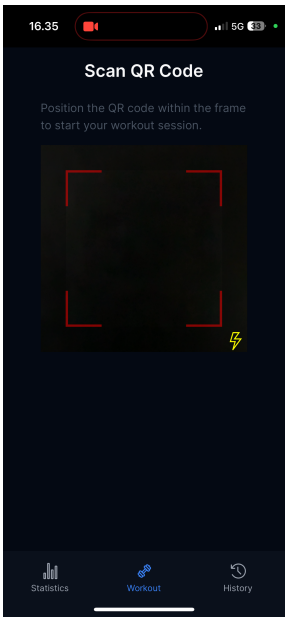


Figure 50: QR code scanner - Workout page.

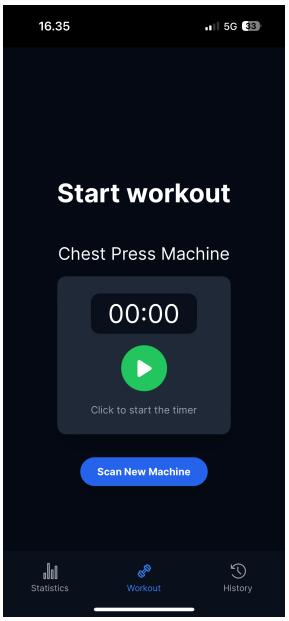


Figure 51: Start workout - Workout page.

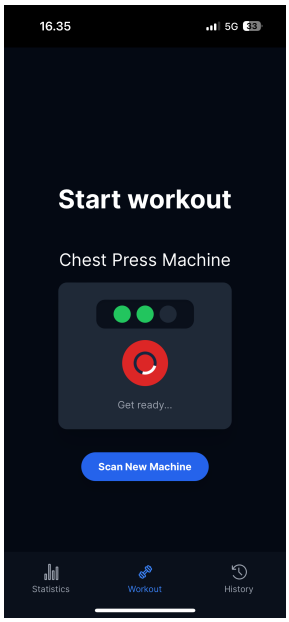


Figure 52: Countdown to begin the set - Workout page.

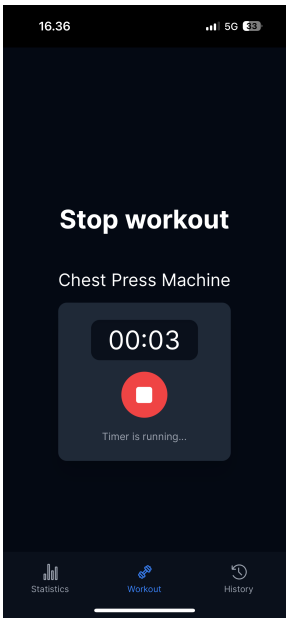


Figure 53: Stop workout - Workout page.

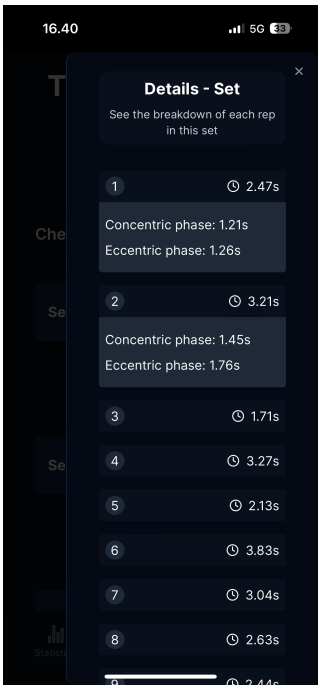


Figure 54: Set details - History page.

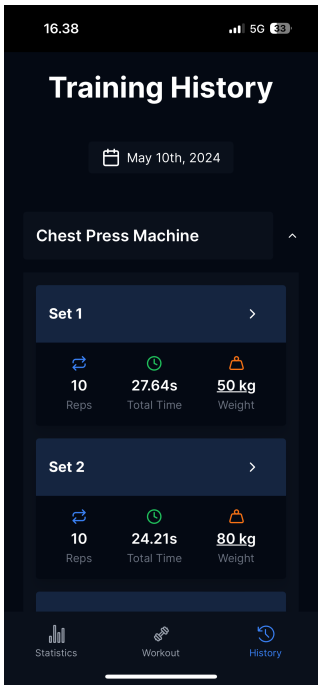


Figure 55: Workout details for one exercise - History page.

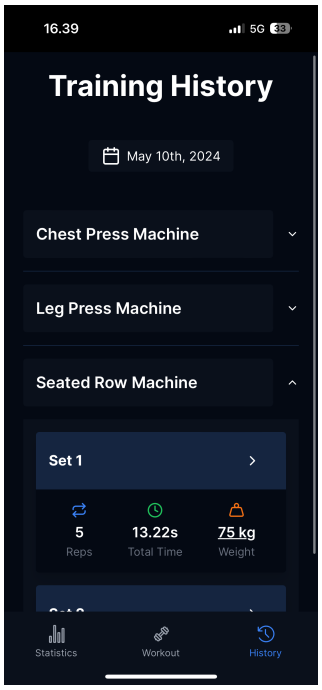


Figure 56: Multiple exercises - History page.

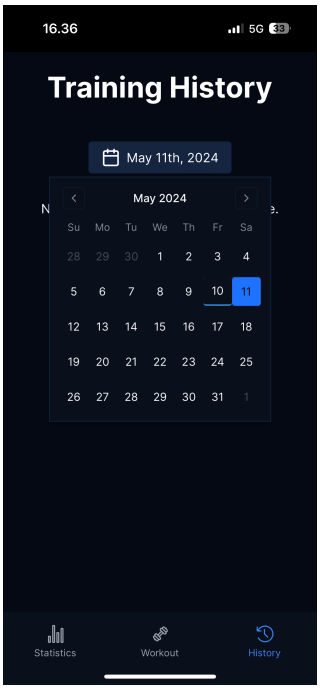


Figure 57: Date picker and calendar - History page.

Figure 58: Various statistics - Statistics page.

E User Testing

Jeg går ind på appen nu, og er i start tab der hedder workout hvor der står "start your workout" og jeg kan scanne en QR kode som jeg kunne forestille mig er til en vægt eller en maskine.

Jeg bevæger mig over i statistics tab hvor jeg kan se nogen tal, der står average intensity og så et tal som jeg ikke helt ved hvad betyder. Den måler en værdi som jeg ikke helt ved hvad er?

Jeg kan se der er en total workout som giver sig selv med et kalender tegn, der formoder jeg at man kan se hvor ofte man har trænet og hvilke dage man har trænet og hvor mange gange og sådan noget.

Så står der "chest press machine" 1RM, også står der kilo, RM ved jeg ikke helt hvad der betyder.. hov det står dernede "*læser op fra info tab"* Så det står hernede hvis man er i tvivl kan jeg se.

Så er der training history hvor jeg kan se dagens dato, så står der machine1 som jeg kunne forestille mig er en af de maskiner man har scannet med QR koden. Hvis jeg åbner den står der hvor mange reps jeg har lavet og hvor lang tid jeg har brugt pr set. Også vægten som man har løftet, også et andet set her nedenunder med nogen andre værdier som må være et andet set.

Figure 59: User test - application 1.

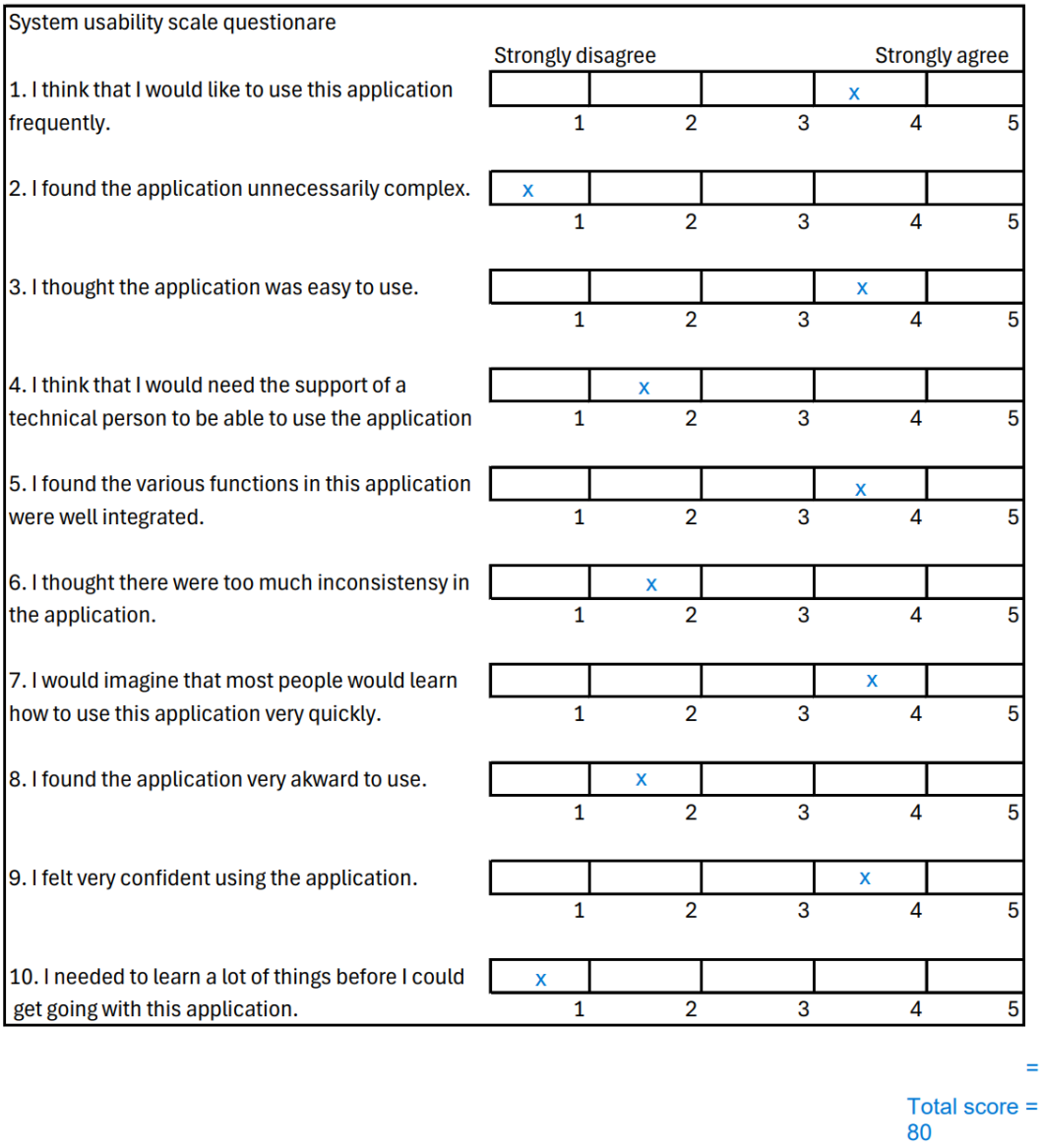


Figure 60: SUS user test 1.

Jeg går ind på workout appen, og er på en start side. Der står "start your workout" som jeg gerne vil så jeg trykker på scan QR code. Så bliver jeg bedt om at scanne en QR kode som jeg så gør på en den maskine jeg er på kunne jeg forestille mig. Efter det så ser jeg en start knap så jeg går ud fra jeg kan starte træningen her.

Jeg går nu over i statistiks siden hvor jeg kan se der står chest press one rep max som er på 85kg. Jeg kan se der er nogen pile i siderne så jeg prøvet at swipe, det virkede nu kan jeg istedet for chest press se seated row med en 1RM på 70kg. Jeg kan også se en blok hvor der står average intensitet som er på 90%, så jeg forestiller mig det er for alle de workouts jeg har lavet som jeg kan se ved siden af hvor der står total workouts hvilket er 3 workouts i alt så det giver jo meget god mening.

I den sidste side som hedder workout history kan jeg se jeg ikke har trænet idag da der er tomt. men jeg kan til gengæld klikke på datoen også er der nogle blå streger under nogen af datoerne så jeg går ud fra det er de dage jeg har trænet. Der er en blå streg under 3 maj så jeg prøver at trykke på den. Der står så jeg har lavet chest press og hvis jeg folder menuen ud kan jeg se jeg lavede 1 set på 3 reps som tog 10 sekunder med 80kg vægt, jeg lavede også 1 set på 3 reps på 12 sekunder hvor der er ikke er en vægt sat der er bare et spørgsmålstegn som er understreget så jeg prøver lige at trykke på den. Der kommer så en pop up som spørger mig om at indtaste en vægt så jeg skriver bare 50kg og trykker videre også opdaterer den på den anden side også.

Figure 61: User test - application 2.

System usability scale questionare

	Strongly disagree				Strongly agree	
1. I think that I would like to use this application frequently.				x		3
	1	2	3	4	5	
2. I found the application unnecessarily complex.	x					4
	1	2	3	4	5	
3. I thought the application was easy to use.					x	4
	1	2	3	4	5	
4. I think that I would need the support of a technical person to be able to use the application	x					4
	1	2	3	4	5	
5. I found the various functions in this application were well integrated.				x		3
	1	2	3	4	5	
6. I thought there were too much inconsistensy in the application.		x				3
	1	2	3	4	5	
7. I would imagine that most people would learn how to use this application very quickly.					x	4
	1	2	3	4	5	
8. I found the application very akward to use.	x					4
	1	2	3	4	5	
9. I felt very confident using the application.				x		3
	1	2	3	4	5	
10. I needed to learn a lot of things before I could get going with this application.		x				3
	1	2	3	4	5	

= 35

Total score = 35 * 2.5 =

87.5

Figure 62: SUS user test 2.