App for Increasing Patient Adherence To Medical Therapy

Jonathan Aanesen

Thesis submitted for the degree of
Master in Programming and System Architecture
60 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2021

App for Increasing Patient Adherence To Medical Therapy

Jonathan Aanesen

App for Increasing Patient Adherence To Medical Therapy

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

Adhering to medical therapy is a significant ongoing issue within the health sector. For patients with heart failure, it is vital for them to take their beta-blocker medication as it significantly reduces stress on their heart by lowering their heart rate. With the technological advances in recent years and the increased number of wearable smart devices which can measure resting heart rates, this thesis aims to create an application using Apple Watch smartwatches and iPhone smartphones to monitor and warn heart failure patients if they have forgotten to take their medicine.

In order to develop the application, we identified three functional requirements which the application would be dependent on: (i) fetching heart rate, (ii) dispatching notifications and (iii) running in the background. The requirements were researched with regards to their respective frameworks and limitations and then used during development to find an approach that would work for the application. The functionality was then combined to create a complete application, which was further configured and tuned for user-testing.

To verify if the application developed was working as intended, including if it had any potential for a real scenario, four participants tested the application for a duration between two and ten weeks. The participants participated in semi-formal interviews where they were asked questions about their experiences with the application and if the application fulfilled its purpose. The interviews indicated that the application worked as a minimum viable product by achieving the intended functionality and showed great promise for actual patients.

The results indicate that it is possible to develop mHealth applications for smartphones and smartwatches, which uses real-life, real-time resting heart rate data to increase patient adherence to medical therapy. Despite the application developed only was tested on healthy participants, the thesis lays the foundation for further development and research to tailor the app for actual patients.

Acknowledgements

I would like to thank my supervisor, Professor Jens Johan Kaasbøll, along with my co-supervisor Ole Jørgen Kaasbøll, for their invaluable help throughout the work on this thesis. I especially want to thank them for their engagement and motivation for the project and for allowing me to work on their idea. It has been inspiring and motivating working with two such competent supervisors. I wish them all the best in the future, with special regards to the continuation of this project.

I would also like to thank my father for always being there for me, especially for his unconditional support, encouragement and care during my years as a student. In addition I would like to thank my twin-brother for being my best friend through life and a person I always can lean on.

Lastly, I would like to thank the rest of my family and friends for their love and support.

Contents

List of Figures

List of Source Codes

Part I

Introduction and Background

Chapter 1

Introduction

1.1 Medical Background and Motivation

1.1.1 Heart Failure

In patients suffering from heart failure, the heart is unable to pump the blood sufficiently to support the needs of the body. Initially, this causes difficulties in breathing when patients exercise. However, the disease is progressive, symptoms gradually worsen, and the 5-year survival for patients suffering from heart failure is only 50-75%. As the incidence is as high as 200-250/100.000 (Dunlay, Roger, and Redfield 2017), it is clear that heart failure remains a major public health challenge.

1.1.2 Beta-blockers

One of the few effective treatments available that can slow down the progression of heart failure and reduce mortality is what is known as beta-blockers. Beta-blockers reduce the mortality of heart failure by 25-30% (Kotecha et al. 2017). Beta-blockers work by inhibiting the over-stimulation of adrenaline and adrenaline-like (adrenergic) substances present in patients suffering from heart failure. Adrenergic stimulation leads to activation of beta-receptors (hence the name beta-blockers is utilized for the class of medicines that antagonize the effect of adrenergic agonists at the beta-receptors). Adrenergic stimulation increases the heart rate, which is therefore often elevated in patients suffering from heart failure. As may be expected, beta-blocker therapy consequently affords a decrease in the heart rate (HR) of a magnitude of approximately 12 beats/minute (ibid.), (Kocianova et al. 2017). Interestingly, the mortality rate of patients suffering from heart failure is also closely correlated to the HR that the patients are able to achieve through the use of beta-blockers (hazard ratio of 1.16 per 10 beats/min increase)(Kotecha et al. 2017). Hence, HR is a measure of beta-blockers usage and an independent predictor of mortality in heart failure patients.

1.1.3 Adherence

Although beta-blockers are widely recognized to reduce mortality in patients suffering from heart failure, patient adherence (i.e. the level at which patients take medication as prescribed) is not 100%. In fact, a general rule-of-thumb is that up to 1/3 of patients take none of the prescribed medication or deviate a lot from the prescribed dose (Osterberg and Blaschke 2005). For instance, it is common for patients to take "drug holidays". In the case of heart failure therapy, this is supported by real-world data demonstrating that the ability to achieve the targeted dose of beta-blockers may not even exceed 50% (Corletto et al. 2018). Furthermore, a European multi-centre trial demonstrated that the probability of achieving the treatment target was highly dependent on the country the patient resided in (Ouwerkerk et al. 2017), thus indicating that adverse effects of the medication were not responsible for the poor adherence. Indeed, it was reported that the beta-blockers' adverse effects were the reason for not achieving the targeted dose for only 25% of the patients not achieving their target beta-blocker dose (ibid.). Thus, a large potential remains for extracting more benefit of beta-blocker therapy for patients suffering from heart failure by simply increasing adherence. As beta-blockers are now generic, i.e. no longer patent-protected and hence inexpensive, simple methods to increase adherence has the potential to translate into large public health gains in a cost-effective manner. Considering that the Norwegian list price for the latest approved, i.e. patent-protected, therapy for heart failure: sakubitril-valsarten - Entresto tm, is ≈17000 NOK/year and that the cost of a beta-blocker is ≈1700 NOK/year, the potential for getting more benefit out of a set healthcare budget is substantial.

1.1.4 Wearable Heart Rate Monitors

One way to increase patients' adherence to their prescribed therapy is by sending reminders by text messages to the patients' phones. However, this has only convincingly been shown to be effective if there is an element of interaction with the patient, i.e. the patients have to reply (Wald, Butt, and Bestwick 2015). Furthermore, text messages are general and not tailored to the individual person. Donevant et al. (2018) did a targeted exploration of literature related to outcomes in mHealth studies. The study involved comparing what kind of features the targeted studies had implemented to how effective they were. Comparing the different studies' results would tell if the interactive and passive features implemented caused statistically significant outcomes (SSO) and could therefore be considered a behaviour changing technique (BCT). Of the studies with SSO, 68.7% of these contained a combination of both passive and interactive features. This result signals that the most efficient way to make a mHealth app into a BCT is to use both passive and interactive features.

Recently, smartwatches with heart rate monitoring capability have started to become more widely available. The combined incorporation of accelerometers and gyroscopes in the watches enables accurate quantification of the resting heart rate (HR). Of the commercially available HR monitors, the Apple watch has been reported to be the most accurate (Wallen et al. 2016), (Wang et al. 2017), (Shcherbina et al. 2017). As beta-blockers directly affect the resting-HR, smartwatches allow for tailored interaction with the user based on the resting-HR at any given day. Furthermore, smartwatches are mass-produced and very

cheap compared to medical devices produced solely for clinical settings. Hence smartwatches may represent a very cost-effective means of a personally tailored interaction aimed at increasing patient adherence to beta-blocker therapy. Furthermore, by implementing commercially available smartwatches, one may avoid the social stigma otherwise associated with a visible device broadcasting one's poor health.

1.1.5 Motivation

The motivation for this thesis is to research, develop and test an minimum viable product (MVP) in the form of an app for iPhone and Apple Watch. The app is to monitor, interact with and warn patients whenever their resting heart rate indicates that they might not have taken their beta-blockers as prescribed. The app should have a simple user interface and primarily function as a background application. The app should be a low-maintenance application that the patient does not need to interact with, except for the warnings. The app should dispatch warnings while running in the background, making the process effortless for the patients. Concerning this thesis, the term "warning" will be used to describe the interactable notifications meant to warn or alert the patient.

1.2 Problem Statement

Considering how many people in the world suffer from heart failure and the public health burden it represents, the proportion of heart failure patients who do not adhere to their prescribed morbidity and mortality lowering therapy is alarming. The Apple Watch provides a cheap and ingenious way of constantly monitoring the patients' heart rates. Including the Apple platform's various frameworks makes it an untapped source of possible patient follow-up solutions from their medical care providers.

As for the current situation, there are no regulatory approved devices aimed at increasing heart failure patients' adherence based on individual, real-time sensor data. There exist applications to remind patients to take their medicines at given times daily. However, the application proposed in this thesis will focus on actively reminding them whenever they have not taken their medication based on real-time, real-world data. In the long run, the use of the suggested device may also entail improved adherence even in the absence of the device due to altered daily habits. In the end, the proposed application will hopefully strengthen the patients' adherence to medical therapy, reduce the public healthcare burden and prolong the patients' lives and their quality of life by doing so. (life: reduced mortality; quality of life: reduced morbidity, i.e. reduced fatigue, shortness of breath, hospital admissions (during exacerbations)).

The scope of the thesis will revolve around one common goal, which is to develop a working MVP of the proposed application. The usability of the MVP will be determined through experiments and user-testing.

With regards to architecture, primarily three functional requirements have been identified to achieve the desired functionality. During research and development of the application, these three requirements will be in focus:

1. **Fetching Heart Rate**
   The app must read the users' resting heart rate from the Apple Watch.

2. **Dispatching Notifications**
   The app must be able to dispatch notifications to the patient. The notifications will be used to warn the patient and request answers through some way of interaction.

3. **Running in Background**
   The app must be able to run in the background, such that the patient does not have to have it open at all times or open it often for monitoring. The application should be maintenance-free for the users.

In addition to these requirements, the application will need a simple and self-explanatory user interface where the user can configure and set up the application.

The thesis will discuss solutions and potential changes regarding the different requirements and discuss learnings from the user testing about possible improvements in future versions.

1.3 Research Method

The research method of this thesis can be characterized as Design-Based Research, which is described in (Ma and Harmon 2009). The method is derived from (Reeves 2000), who came up with Development Research. Development research is described by Reeves as a research process that consists of the following four steps:

1. Analysis of Practical Problems by Researchers and Practitioners
2. Development of Solutions with a Theoretical Framework
3. Evaluation and Testing of Solutions in Practice
4. Documentation and Reflection to Produce "Design Principles"

These steps are a recursive process, where the different steps are refined over time through problems, solutions and methods (Ma and Harmon 2009).

However, in design-based research, the process is iterative, allowing the development research to be carried out over one or more iterations. The method proposed by Ma and Harmon uses the same four-step framework as Reeves, but adds additional details or sub-tasks to each step. For the sake of this thesis, each step will be categorized into phases: *(i) analytical phase, (ii) development phase, (iii) testing phase, (iv) documentation phase.* Below, each phase will be described with a brief explanation of how this thesis's work fits the respective phase in Design-Based Research.

1.3.1 Analytical Phase

The analytical phase covers the first step: "analysis of a practical problem by researchers and practitioners". For this step, I, as the author and developer for this thesis, function both as a researcher and practitioner, in addition to my two supervisors who came up with the idea for the thesis. Ma and Harmon came up with two sub-tasks that would help the analytical phase:

1. Identifying a Practical Problem

2. Reviewing the Literature to Determine the Significance of the Problem

Both these sub-tasks have already been discussed and elaborated in section 1.1: Medical Background and Motivation. This phase was conducted at the very beginning of the work on this thesis. The phase laid the background for precisely what kind of application would need to be developed, the targeted problem, and how to approach it.

1.3.2 Development Phase

For the development phase, Ma and Harmon came up with five sub-tasks:

1. Conceptualize a solution within a theoretical framework

2. Determine the role of research in developing the solution

3. Identify the purpose and research questions for a development iteration.

4. Identify development methods

5. Develop a prototype that serves the research purpose

We go through technical background, analysis, design, and development for this phase of the research to fulfil all the sub-tasks. By the end of the phase, we will have a working prototype that serves the research purpose. We will have elaborated on possibilities, limitations, and external factors regarding the three functional requirements defined in section 1.2: Problem Statement.

1.3.3 Testing Phase

For this phase, Ma and Harmon identified three sub-tasks:

1. Identify research methods

2. Gather and analyze data to answer research questions

3. Draw conclusions and determine research findings

In this thesis, several experiments were conducted to ensure the functional requirements were upheld and that the application would serve the research purpose. The experiments can be divided into two types of evaluations: (i) software testing against functional requirements and (ii) user-testing. The data analyzed were qualitative data gathered from interviews conducted during the user-testing. At the end of the phase, the different results and feedback from the experiments are discussed as well as a conclusion for the thesis.

1.3.4 Documentation Phase

For the last phase, Ma and Harmon proposed two sub-tasks:

1. Synthesize design principles for developing the proposed solution

2. Synthesize guidance for conducting design-based research.

The last phase of the research will primarily consist of writing this thesis to document the various findings revolving around the concept, design, frameworks, and the proposed application. The thesis will also include a chapter outlining potential future work to guide any work based on this thesis.

1.4 Thesis Overview

Chapter 2: Technical Background gives a detailed background of everything related to the technical part of the thesis. The chapter gives an overview of the devices in focus and the various frameworks used, and how the application will be developed and distributed.

Chapter 3: Analysis and Design describe the process of how the application is to be developed with regards to design and architecture. The chapter discusses solutions to each functional requirement, as well as data structures and user interfaces.

Chapter 4: Development introduces diagrams with regards to flow, architecture and interface. The section describes how the applications work internally through descriptive class diagrams and externally by addressing the respective SwiftUI Views.

Chapter 5: Evaluation presents the experiments conducted, along with their results and observations. The chapter emphasizes the results, especially from the user-testing.

Chapter 6: Conclusion summarizes the research results in this thesis and presents a section describing the best way to use the proposed application. The chapter concludes the thesis by describing potential future work, which can be used for a continuation and proposes several solutions to problems observed during testing. It also proposes additional functionality which can further enhance the application and user experience.

Chapter 2

Technical Background

2.1 Apple's App Store Ecosystem

Apple is a widely known tech company that designs, develops and sells consumer electronics, computer software and online services. Mostly known for the Mac and iPhone, the company have a wide variety of products such as the iPhone, MacBook, iPad, iPod, AppleTV and AirPods.

An ecosystem is defined as a biological community of interacting organisms. In the digital world, this can be seen as actors that co-create value on a digital platform. (Hein et al. 2019)

Apple's developer documentation and the App Store Connect functions as an application programming interface (API) for developers to develop and distribute applications and features to the App Store. The App Store functions as an interface for consumers who take advantage of the increasing amount of applications and functionality on the platform. Apple as a company gets an advantage from the increasing popularity of their products, as well as a percentage of sales distributed through the App Store, a highly controversial and legally contested feature of the Apple ecosystem. The App Store also supports distribution to the various Apple platforms like iOS, iPadOS and MacOS, connecting them through the App Store. The *company* $\leftrightarrow$ *developer* $\leftrightarrow$ *consumer* relationship is what makes the Apple App Store ecosystem, where each actor benefit from the other.

The Apple platform revolves around everything that has to do with Apple and their products. Apple products have applications and functionality specifically developed to make their products work better together, making users stay within the platform. One example of how Apple benefits from their platform through sales of accessories is the sale of AirPods, which has almost doubled each year since its launch in late 2016. From 60 million AirPods sold in 2019 to 114 million in 2020, the AirPods contribute over $10 billion to Apple's revenue alone (Curry 2021). Because Apple removed the headphone jack on their new iPhones, users now also purchase the Apple AirPods to listen to music wirelessly. Even though newer generation iPhones still support headphones with an adapter through the Lightning port, Apple facilitates the shift towards wireless by removing the auxiliary port and offering their own wireless headphones. The Apple Watch creates additional functionality for the iPhone and is seen as a tool

to enhance the users' experience by performing simple tasks without touching their phone. Infrastructures such as iCloud, combined with AirDrop, AirPlay etc., which work interchangeably between different Apple devices, strengthen the platform. Apple's platform is strengthened because devices within the platform are designed to interact seamlessly straight out of the box. It provides a barrier for third parties and a competitive advantage for Apple's own products.

2.1.1 Apple Watch

The Apple Watch is a smartwatch developed by Apple and initially released in 2015. The Apple Watch has since then had 6 new versions released. Continuing to grow in popularity, "Apple Watch outsold the entire Swiss watch industry in 2019", having shipped over 30 million units (Kharpal 2020). The Apple Watch works as an accessory or companion to Apple's iPhones, allowing it to replace or accommodate applications on the iPhone and in the most recent eSIM versions replace the basic phone features of the iPhone. The watch has an integrated speaker, microphone, screen and multiple sensors making it also an independent device that can be used for various purposes such as a health monitor or a workout companion. It comes with different pre-installed features such as high and low heart rate notifications, irregular rhythm notifications and fall detection. It is also noteworthy that the ECG feature introduced with Apple Watch 4 has received FDA approval as a medical device thus decisively marking the entry of Apple into the heavily regulated healthcare industry. These health-related apps are powerful and great regarding emergencies and monitoring of a persons' health. The proposition of this thesis builds on top of these features and takes it one step further by directly integrating the use of the Apple Watch features with a medical intervention, not just a monitoring feature.

High and low heart rate notifications

A feature on the Apple Watch allows the user to set an upper and lower threshold for your heart rate when you have been inactive for the last 10 minutes. If the threshold is met, the user will receive a notification.

Irregular rhythm notifications

The watch can monitor and discover abnormal heart rhythms, including heart attacks. A case study was made with 419 297 participants over 8 months, where 2161 participants received notifications of an irregular pulse. It turned out that 84% of the notifications made by this feature, was concordant with Atrial Fibrillation (Perez et al. 2019).

2.1.2 iPhone

The iPhone is a smartphone developed by Apple. Steve Jobs introduced the first generation iPhone in 2007, and the latest generation (iPhone 12) was released in late 2020. The iPhone has in total sold over 2.2 billion devices and had a 23.4% market share in the last quarter of 2020, making it one of the most popular smartphones out there (Scarsella et al. 2021).

2.2 Frameworks

Apple frameworks function as building blocks in software development for the various Apple devices that exist. The frameworks provide different distinctive functionality, allowing applications to extend their functionality beyond the devices' default framework. Having separate frameworks for different functionalities allows the code to be modularized. Modularization allows an application to stay minimal, only being enlarged by the different frameworks it imports. This is a common front-end development architecture which is widely used and can for example be seen in JavaScript projects using Node package manager (NPM) such as React, Vue and Angular.

2.2.1 HealthKit

The HealthKit store is designed as a common repository between applications on the Apple platform, focusing on health-related data such as activities and sensor data. The HealthKit framework facilitates an extensive amount of data types and multiple ways for developers to use the data in their applications. (Apple Inc. 2020)

Permission

Many frameworks on the Apple platform requires permission by the user to be used. The HealthKit framework does not require just one permission but individual permissions for each type of health-related data being read or written. Apple has specific guidelines for how and when applications should ask and check for the users' permission. Usually, the permission-request screen is to be displayed the first time a user opens the application. The permission should be checked every time the application tries to access or store data using the framework. (Apple Inc. 2021b)

User Privacy

Health records stored on Apple devices such as the iPhone or Apple Watch are encrypted in the devices' HealthKit database, called the HealthKit store. If the user has chosen to synchronize the Health data between devices with iCloud, the data transfer has end-to-end encryption using two-factor authentication. The data is always transferred directly between the user device and iCloud, which means Apple does not have access to anyone's data unless they are enrolled in Apple's Improve Health Records program. This program allows the user to share anonymous health data with Apple to further develop, improve and understand the Health Records feature. The user is always in complete control of their own data, where they can choose what data to be stored, how it is stored, who to share it with, and they can delete it at any time. (Apple Inc. 2019). The only exception to this control is that the data is always stored on Apple's servers, and the user cannot decide the location where their data is stored as it's part of Apple's cloud architecture, meaning the data is most likely redundant, being backed up in multiple data centres on several continents.

It is difficult to know if the data stored on iCloud for the users' behalf is secure from Apple's access to it themselves. We can only trust that they keep

the data as confidential as they declare, considering that Apple would likely be liable for substantial financial penalties if the data was to leak.

Queries

We can use three different methods to access data from the HealthKit Store: direct method calls, queries, and long-running queries. Direct method calls are mainly for accessing characteristic data and does not give access to anything else. However, the queries and long-running queries are more advanced and have multiple types to return different data from the HealthKit store.

We have eight different query types, where three of them support long-running. Those, in particular, can be programmed to either return data once or keep running so our app does not need to request data every time it wants to update, but rather have an update handler for a long-running query which will automatically update the application every time the query has an update. There are four different long-running queries, where three of them can run as normal queries. The observer query is the last one that is only long-running and is highly relevant to the application developed in this thesis. The observer query monitors the HealthKit Store and alerts the application when specified data samples are added, updated or changed. This long-running query also supports alerting while the application runs in the background, providing the ability always to have up-to-date data in our application.

More details about the two relevant queries used in this thesis:

- **Statistical Collection Query**
  The statistical collection query is useful for performing multiple statistics queries over time intervals.

  This query can be initialized by providing a quantity type, quantity sample predicate, options, anchor date and interval components. The quantity type is what health data type the query will return. The quantity sample predicate allows you to limit the results returned by the query, but leaving it blank makes it return all possible results. The options allow us to define statistical calculations performed on the data and to define how the data is merged from different sources. Within the options we can for example, ask for the discrete average, which returns the calculated average quantity of the results. Anchor date defines when the time intervals begin. At last, there are interval components, which define how long an interval lasts. This can be a specific amount of minutes, hours, days or even weeks.

  After initializing the query, a result handler must be set. It is in the results handler where we handle any errors with the query or handle the resulting data sets. There can also be set a statistical update handler that allows the query to be long-running, calling the update handler every time there is a change in the defined data set. (Apple Inc. 2021j)

  An example of a simplified statistics collection query can be seen in Listing 1. Here the query and results handler is defined, then executed in the HealthKit store.

- **Observer Query**
  The observer query is useful for being notified when specific data types are added or changed in the HealthKit store.

```
func fetchRestingHeartRates() {
    let calendar = NSCalendar.current
    let anchorComponents = calendar.dateComponents(
        [.day, .month, .year, .weekday], from: NSDate() as Date)
    let anchorDate = Calendar.current.date(from: anchorComponents)
    let interval = NSDateComponents()
    interval.day = 1
    let endDate = Date()
    let startDate = calendar.date(byAdding: .day, value: -6, to: endDate)
    let quantityType =
        HKObjectType.quantityType(
            forIdentifier: HKQuantityTypeIdentifier.restingHeartRate)

    let query = HKStatisticsCollectionQuery(
        quantityType: quantityType,
        quantitySamplePredicate: nil,
        options: .discreteAverage,
        anchorDate: anchorDate,
        intervalComponents: interval as DateComponents)

    query.initialResultsHandler = {
        _, results, error in
        let statsCollection = results
        var values: Array<Double> = []
        var dates: Array<Date> = []
        statsCollection.enumerateStatistics(
            from: startDate, to: endDate) { statistics, _ in

            if let quantity = statistics.averageQuantity() {
                let date = statistics.startDate
                let value = quantity.doubleValue(for: HKUnit(from: "count/min"))
                values.append(value)
                dates.append(date)
            }
        }
        // Update app variables
    }
    healthStore.execute(query)
}
```

Listing 1: Statistical Collection Query

The observer query is instantiated with a quantity type, an optional predicate and an update handler. After the query has been instantiated, there is a possibility to enable background delivery for it. Background delivery can be enabled by calling the HealthKit store with ".enableBackgroundDelivery" and supplying the quantity type and a frequency to it. By enabling background delivery, the update handler will be called from the background every time the specified quantity type has a new or changed value

at the specified frequency. We can choose between immediate, hourly, daily and weekly for frequencies. Having frequency set to immediate does not guarantee that the update handler will be called immediately. Rather it means that the update handler will be called sometime after, but as soon as the device allocates the resources to do so. (Apple Inc. 2021h)

An example of a simplified observer query can be seen in Listing 2. Here the quantity type is supplied to the observer query. After the observer query has been instantiated, we enable background delivery and execute it in the HealthKit store.

```
func startObserver() {
    let quantityType = HKObjectType.quantityType(
        forIdentifier: HKQuantityTypeIdentifier.restingHeartRate)

    let query = HKObserverQuery(sampleType: quantityType, predicate: nil) {
        _, completionHandler, error in

        // Handle error

        // Update application here

        completionHandler()
    }
    healthStore.enableBackgroundDelivery(
        for: quantityType, frequency: .immediate) { _, error in

        // Handle error
    }
    healthStore.execute(query)
}
```

Listing 2: Observer Query

2.2.2 User Notification

User Notification is Apple's framework for handling and displaying different types of notifications. There are two main ways notifications can be generated: local notifications and remote notifications (push notifications). Local notifications are created locally on the users' device, while remote notifications are generated remotely and sent to the device through the Apple Push Notification service (APNs). Local notifications allow the developer to create the notification content and conditions locally to use information from the app. Push notifications enable the developer to push generic notifications to the user without having the app do any of the work.

Permission

Because notification-based interactions are considered disruptive, the app must obtain permission from the user to dispatch notifications. This is done by ac-

cessing the User Notification Center and requesting authorization for the notification options desired. The most common notification options are badge, sound, alert, carPlay, criticalAlert, provisional and announcement. Where carPlay is to deliver notifications to a carPlay system in a supported vehicle, and announcement is to allow Siri to read out messages over AirPods. A regular notification usually uses a badge, which updates the application badge by putting a red dot on it. It also uses sound to notify the user auditory and alert to display the standard notification pop-up, with or without actions.

If the user does not permit the app to deliver notifications, the last option is to use a provisional notification. This notification type is delivered silently to the user as a trial on how a notification from the app can be useful to the user. The provisional notification is also displayed with two actions allowing the user to give notification permissions to the app or keep them off.

When requesting authorization from the User Notification Center, the user notification framework will check if the application already has permission from the user. If not, it will prompt the user with a dialogue box where they can choose to allow it or not. It is recommended to make the authorization request in context with the functionality of the application. Meaning if the user does something on the app that schedules a notification, it is smarter to make the request at that time, so the user is prompted and gets a feeling of the notification's purpose. This is recommended rather than automatically requesting authorization on the first launch. (Apple Inc. 2021a)

Notification Structure

Notifications are represented as a notification request, which is an object made up of three components:

1. **Identifier**
   An unique string that allows the notification to be identified.

2. **Content**
   The notification content is an object which contains all visual information regarding the notification, such as title, subtitle, text, actions and badge number. The content is where the notification is designed and given shape.

   To add actions to a notification, the actions must be pre-determined and registered in the notification center as a notification type or category. When scheduling the notification, the category identifier is set in the content, which makes sure it will be delivered with the defined actions. To accommodate the notification actions, there must also be defined action selection handlers to perform the tasks associated with the actions.

3. **Trigger**
   The notification trigger is what decides when a notification is to be published on the device. Three main different triggers can be used: time interval, calendar or location where time interval allows for publishing a notification a set amount of time after being scheduled. Calendar trigger allows for a specific date and time the notification is to be published. Finally, the location trigger allows a notification to be triggered when the device enters or leaves a geographical region. In addition to taking time,

day or region, the trigger also has the option to repeat the trigger, meaning a notification can be delivered every day at a specified time by only scheduling it once.

```
func ScheduleNotification() {
    let content = UNMutableNotificationContent()
    content.title = "Notification title"
    content.body = "Notification body"
    content.sound = UNNotificationSound.defaultCritical

    // Create the trigger as a repeating event.
    let trigger = UNTimeIntervalNotificationTrigger(
        timeInterval: 1, repeats: false)

    // Create the request
    let identifier = UUID().uuidString
    let request = UNNotificationRequest(
        identifier, content, trigger)

    // Schedule the request with the system.
    let notificationCenter = UNUserNotificationCenter.current()
    notificationCenter.add(request) { error in
        if error != nil {
            // Handle any errors.
            NSLog("Error")
        } else {
            NSLog("Success")
        }
    }
    return
}
```

Listing 3: Function for scheduling notifications

In Listing 3 there is a code snippet of how basic notifications are created and scheduled using the User Notification framework. In this example, the notification content is defined first, containing a title, body and sound. Then the time interval trigger is defined with the value **Double: 1**, which is passed to the function. A universally unique identifier is then created along with the content and trigger, which are then passed into a **UNNotificationRequest**. The request is then scheduled through the User Notification Center, and the rest is handled there. (Apple Inc. 2021k)

Notification Types

There exist primarily three types of notifications: Normal notifications, actionable notifications and silent notifications. Normal notifications are the standard type of notification that displays a title and body while triggering a sound. These notifications can be interacted with by pressing on them to open the

associated application. Actionable notifications are very similar, but they provide "actions" or options for the user to interact with the application without opening it. This is done by swiping on the notification when displayed, then selecting one of the pre-defined actions. The silent push notifications are also called background push notifications. They work by having a push notification server send an empty or "silent" notification to the device, which wakes up the associated application and allows it to perform any non-UI related operations in the background.

Delivery

The notification center handles notifications and is the central object for managing notification-related activities for apps or app extensions. When scheduling a notification, the notification center makes sure the app has authorization from the user and will deliver the notification according to the trigger. The notification center also handles the notification when delivered and any actions the notifications may have. The notification center also gives the ability to cancel scheduled notifications using the identifier, which is useful if the user, for example, deletes a reminder or a variable that triggered the notification request changes before the notification is delivered.

2.2.3 WatchKit

WatchKit is Apple's framework providing the infrastructure for developing watchOS applications. The framework handles everything necessary for applications to run on Apple Watch devices and adds a seamless experience for Apple Watch development. (Apple Inc. 2021q)

2.2.4 EventKit

EventKit is Apple's framework providing access to calendar and reminder data. Similar to HealthKit, there is an EventKit Store which applications can access through this framework. This framework can create, retrieve, update and manage different types of events such as calendar events, reminders and alarms. (Apple Inc. 2021g)

Reminders app

In addition to the calendar, Apple has its own Reminders application, which the user can use to create reminders, sub-tasks and alerts. Through the EventKit framework API, applications can create these types of events in the user's reminders app. (Apple Inc. 2021e)

In Listing 4 there is an example of how a reminder for the Reminders app can be created from an application. The function takes an integer as a parameter to specify how many hours till the reminder is to be scheduled. The time interval can be set as a specific time or date, but in this example, it is set as the number of hours from the current date. The function then gets the users calendar for new reminders and sets up a new EventKit Reminder (EKReminder). The reminder is then provided with the calendar, a title and priority. A time interval is created with the hours parameter, and this is then used to set up an EventKit Alarm

```
func scheduleReminder(hours: Int) {
    guard let calendar =
        self.eventStore.defaultCalendarForNewReminders() else { return }
    let newReminder = EKReminder(eventStore: eventStore)

    newReminder.calendar = calendar
    newReminder.title = "Reminder title"
    newReminder.priority = 1

    let timeInterval = 60*60*hours

    let alarmTime = Date().addingTimeInterval(Double(timeInterval))
    let alarm = EKAlarm(absoluteDate: alarmTime)

    newReminder.addAlarm(alarm)

    let dueDate = Date().addingTimeInterval(Double(timeInterval))
    newReminder.dueDateComponents =
        Calendar.current.dateComponents(
            [.year, .month, .day, .hour, .minute, .second], from: dueDate)

    try! eventStore.save(newReminder, commit: true)
}
```

Listing 4: Function for scheduling reminders

(EKAlarm), which also is supplied to the reminder. A due date is also created from the time interval, which is supplied to the reminder before it is stored in the EventKit store.

2.2.5 Limitations

Before creating and developing a functional application performing the desired tasks, we must also identify some fundamental limitations of background processing on iOS and watchOS devices.

Background fetch

When applications are permitted to background fetch, it is the system that decides which applications get time and when. The applications are decided upon according to how much the applications are used and at what time they are usually used. For example, if you regularly read a news app during your lunch break, the system will most likely allow the news app to background fetch around half an hour before you would usually open it. This way, the application has fetched new content in preparation for the user to open it, such that the user will not have to wait for new content to be displayed or loaded in.

Battery state

The battery state of the device affects how much background processing is permitted. The battery has three different states: Sufficient battery, low power mode and time to recharge. When the phone is in time to recharge, it uses as little power as possible to ensure that the phone stays alive until it is being recharged. Only a handful of processes are allowed background execution time, and our app will not be able to check the HealthKit Store or publish notifications when in this mode. When the phone is in low power mode, the user has chosen to conserve battery and decide which applications should be prioritized. Here our application will function only if the user has decided it to be a priority. When the phone has sufficient battery, the applications run as normal, and the operating system decides the background execution allocation. Depending on how many applications want to use background execution and how extensive these executions are, it creates variability on how often our application can fetch data and schedule notifications.

HealthKit Access

A fundamental issue encountered while researching and prototyping is that the devices do not have access to the HealthKit store in a locked state. For example, the Apple Watch can write to the HealthKit Store while being worn, but the iPhone cannot detect a resting-HR change before the user unlocks the phone. Despite the app running in the background on the iPhone, it will only trigger notifications based on the resting-HR whenever the device becomes unlocked.

Notifications

If the application can only retrieve HealthKit Store data when the iPhone is unlocked, then the notifications will most likely only be delivered on the iPhone device unless the user manages to lock their iPhone before the notification is delivered. Only then will the notification be delivered on the Apple Watch.

Background Running

A smaller yet notable limitation to the functionality is that the users must open the app once before they can receive notifications. Of course, the application must have been opened initially to give permission for notifications, reminders and to read health-related data. Still, now and then, the system may terminate the application, making it useless. If the app is terminated by the user or system and not opened afterwards, it will not be able to run background tasks and, therefore, not detect changes in the users' resting heart rate. The iPhone keeps the application in the background after a restart, but updates may terminate all applications or require permissions to be reacquired.

Other Wearable Heart Rate Monitors

A not huge but noticeable limitation is that the Apple Watch is not the only device capable of determining resting-HR values and supplying them to the HealthKit store. This means that other health-related devices the user might have can store values to the HealthKit store, which the proposed application will

read. Because of the Apple Watches previously mentioned accuracy in section 1.1.4, it would be unlucky for a user to have another smart device distort the already accurate results of the Apple Watch. This limitation is directed to the users being limited to what smart devices they can use, and the recommended use-case will be discussed later in section 6.1.

2.3 Environment

When discussing an application's technical background, it is also important to mention the various tools used in a developing environment to create said application. This section will give a brief explanation of the different tools and languages used to develop the different applications proposed in this thesis for the Apple platform.

2.3.1 XCode

Xcode is an Integrated Development Environment(IDE) developed and distributed by Apple to develop applications for their various products. Xcode provides everything you need to create, test, deploy and distribute applications for the various operating systems used in Apple devices. The IDE is complete with everything a developer may need, such as full documentation of the various frameworks with examples, simulators, previews and much more (Apple Inc. 2021r). It is also said that "arguably, without such a powerful, refined, integrated development environment(IDE), the thriving ecosystem that is the App Store would not exist as it does today"(Knott 2014).

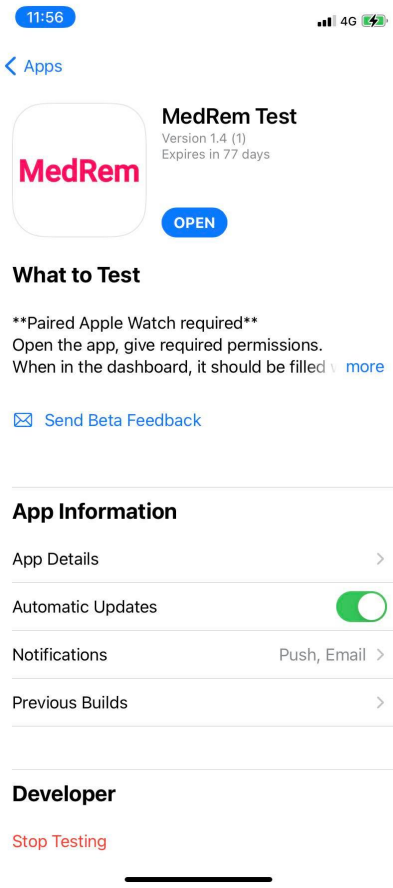

Figure 2.1: Screenshot of TestFlight app

2.3.2 TestFlight

When distributing applications in the App Store ecosystem, all applications are uploaded to the App Store Connect. This is where developers can manage their applications in the form of product pages, in-app purchases, test, submit and manage releases, and much more. Testflight is part of the App Store Connect, which concerns the testing of the application. It is a framework designed to help developers distribute test versions and obtain feedback to discover bugs and crashes. If an application distributed through Testflight crashes, the developer can check in Xcode for info about the crash occurrence. This makes bug fixing a lot easier as Xcode will specifically tell you what happened with which values for the crash to happen.

When the developer of an application adds testers to TestFlight, they get an invitation email guiding them to download the TestFlight app. In the TestFlight app, the testers have the possibility to download applications they are invited to test, as well as sending beta feedback to the developers. Figure 2.1 displays the MedRem Test application's page in the TestFlight app. (Apple Inc. 2021n)

2.3.3 Swift

Swift is the new main programming language used for developing applications in Xcode. It was first introduced in 2014 at the Apple Worldwide Developers Conference (WWDC14). Before Swift, another programming language called Objective-C was the primary language. It is a general-purpose, object-oriented language that mainly is an extension to the programming language C with messaging capabilities. Swift, on the other hand, is a general-purpose, multi-paradigm, compiled programming language. It is a modern programming language that requires less code to perform the same tasks as in Objective-C. The simple syntax and Python-like structure make it simple to pick up for beginners. Still, it is such a powerful programming language that experienced developers loves it.(Apple Inc. 2021l)

2.3.4 SwiftUI & App Delegate

SwiftUI is a user interface framework developed to create user interfaces across all Apple platforms easily. The framework is straightforward to use and has many built-in features making 5 lines of code into the most simplistic and beautiful user interface. The cross-platform compatibility with the bundle of features it handles for the developer automatically, like Dark Mode, makes it truly one of the most powerful UI frameworks available. In WWDC20, Apple introduced its own app lifecycle to SwiftUI. Previously the app lifecycle was managed through an AppDelegate in the UIKit framework and required some tedious configuration to perform default behaviours. Still wanting to simplify this process, Apple added a new type of lifecycle in SwiftUI. Developers still have the possibility to use the AppDelegate in projects with SwiftUI, as more complicated functionality requires more specific configuration, which the AppDelegate can provide. The new SwiftUI app lifecycle improves the development experience of simpler, straightforward applications and is very easy to understand. (Apple Inc. 2021m)

In Swift with SwiftUI, when a class is of the observable object type, it becomes an object with a publisher that emits before the object changes. This observable object can be instantiated in the App's environment as a StateObject. A structure of the View type can create an environment object variable that expects to find the given object and use its published variables. The environment object will then invalidate the View every time the object changes, which will refresh the view with the newly updated data. The variables which are to be read by the View has to be declared with the Published attribute for the observable object to publish its changes. An example of the minimum required code to make this work can be seen in Listing 5.

This way of state management is handy. It facilitates the communication between objects and views, allowing the data handling to be managed in the object and the projecting of said data in the views.

```swift
import SwiftUI

class DataModel: ObservableObject {
    // Publishes variable to the object
    @Published var seenVariable = "Hello, World!"
}

@main
struct App: App {
    // Instantiates the observable object in the environment
    @StateObject var dataModel: DataModel = DataModel()

    var body: some Scene {
        WindowGroup { TabView {
                ContentView().environmentObject(dataModel)
        } }
    }
}

struct ContentView: View {
    // Looks for a DataModel object in the environment
    @EnvironmentObject var dataModel: DataModel

    var body: some View {
        // Displays DataModel's seenVariable
        Text(model.seenVariable)
    }
}
```

Listing 5: Usage of an observable object

Part II

Design and Development

Chapter 3

Analysis and Design

The motivation of this thesis is to create an application that can supplement the prescribed medication beta-blocker. The application is to monitor and compare the recorded values to individual pre-defined set points, warn patients if the recorded value exceeds the set-point, and remind them to take their prescribed beta-blocker therapy. The final application developed in this thesis is called *MedRem*, an abbreviation of Medicine Reminder. MedRem monitors the patients resting-HR throughout the day with data from the Apple Watch and acts upon these findings between set intervals of the day.

The motive behind the application is to create an MVP, to prove the concept of an application supplementing a medicine to follow up the medical therapy adherence. The application provides an interface that displays the relevant data it uses, as well as multiple notifications used for communicating with the user.

In the introduction of this thesis, it was described three functional requirements which will be used to develop the application: (i) Fetching HR/Health-data, (ii) Dispatching notifications, and (iii) Background Application.

The reason the application is based on these three functional requirements is that:

- MedRem has to be able to check the users resting heart rate to determine if the user has taken their medicine or not.
- MedRem must deliver different types of notifications to the users to warn them if the recorded resting-HR values exceeds the pre-defined set-point and especially interact with the users.
- MedRem has to run in the background and do the two previous tasks while doing so. This is required such that MedRem is maintenance-free, and the users do not have to interact with it unless they receive a warning or are re-configuring the app.

In this chapter, we will get a detailed look at MedRem from a planning and architectural perspective. We will dig into how we separate the development for the functional requirements, recombine them in MedRem, and generally how the application is otherwise designed and structured.

3.1 Requirement Analysis

3.1.1 Stakeholders

There are four identified stakeholders for the MedRem application:

1. **Patients**
   The primary user of the application who interacts and benefit from it.

2. **Care providers**
   The secondary user might not use the application directly but benefit from their patient using the application. They are meant to help their users set up the application and follow up on them occasionally.

3. **Society**
   In a real-world setting, the care provider may be more or less incentivised to improve patient outcomes (e.g. compensated depending on patient outcomes or target achievements) and hence more or less of a stakeholder. While this is not currently the case in Norway, Norway represents a minute fraction of the total number of heart failure patients. For society as a whole, any method/intervention that reduces mortality and particularly morbidity is a benefit, thus making society an ultimate stakeholder.

4. **Developers**
   The developer is a contributing stakeholder who develops, maintain and distribute the application.

3.1.2 Mobile Platform

The various applications developed throughout this thesis are to be developed for mobile environments only. Some applications also need to support a wearable-mobile environment, working as standalone applications or as a companion for a mobile device. Because the applications are to be developed for mobile and possibly wearable platform, they must account for battery consumption, processing power and other limitations by which mobile devices are restricted.

3.1.3 Privacy

The application proposed in this thesis will read and use the users resting heart rate. Besides, the application will use these values to dispatch notifications and reminders to warn the user. As explained under section 2.2.1 considering the users' privacy and safety, the health data is not stored directly in the app but through Apple's HealthKit repository and fetched by the app whenever needed. The users must also give specific consent to HealthKit for the application to read their resting-HR. They must also give consent to the User Notification and EventKit frameworks for receiving notifications and reminders.

3.2 Modular Design

3.2.1 Functionality-Specific Applications

To split up the application functionality into sections that can be tested independently of each other and get familiar with the different frameworks' usage, a decision was made to divide the app into three smaller applications initially. These three applications concern their own batch of functionality and function as modules or building blocks for the final application: the MedRem app.

In the following modules, there will be references to both module and application. The application is the standalone app developed for testing the module functionality, and the module is referred to as everything regarding the desired functionality of that module. For instance, when recombining the modules into the MedRem app, the code regarding the module's functionality is copied from the different applications to the MedRem app.

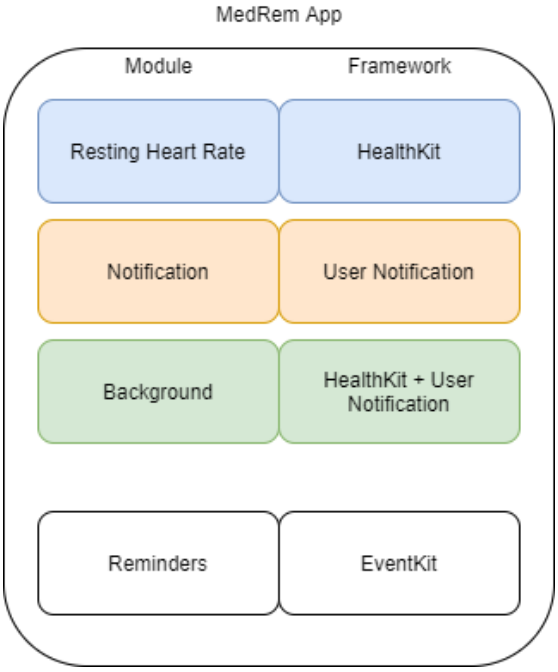

Figure 3.1: MedRem App modules

Heart Rate Module

This module's main purpose is to figure out the required steps to use the Apple HealthKit framework. Using this framework is not entirely straight forward because additional permissions are required from the app and physically from the user. Because of this, when trying to tap into the framework, the application must check the framework for available data, then it must check that the user has given consent for the type of data the application is trying to read/write.

To test applications using HealthKit, it is best to test with an actual device rather than the simulator. If using a simulator, the HealthKit Store must also

be filled with "fake" data, which the module can then retrieve through the framework. This makes it a bit more difficult to develop, considering the final application is not meant only to display data but react according to how the HealthKit data changes.

Figure 3.2: Heart Rate App

The heart rate app created was developed for WatchOS. However, there would be no difference between the application's functionality if it was developed solely for iOS. In fact, the code would be identical considering the functionality part, where only a tiny bit of the user interface styling would be different. This is because of the Apple HealthKit framework, which stores the users' health data in one common repository which applications can tap into given the users' consent. As long as a user has an Apple Watch paired with an iPhone, the application could function normally on the iPhone. The watch will provide the HealthKit repository with the heart rate data our application is interested in using. That way, the only use of the Apple Watch would be to read the users heart rate and supply it to the HealthKit store.

For this module, the main functionality was accomplished using an edited version of the example in Listing 1. The edited version used HealthKit's Sample Query instead of the Statistical Collection Query to get the most recent heart rate value from the HealthKit store.

Resting Heart Rate Module

Figure 3.3: Resting Heart Rate App

Using an Apple Watch device to test the application physically gave insight into how data is stored, how frequent and what values the application can fetch. After reviewing the health data stored in Apples' Health app, it turned out that the resting-HR data is set as one value for the entirety of the day, instead of values for different time points, which is how the regular heart rate is stored. The resting-HR value is updated throughout the day according to heart rate values that can be classified as sedentary or resting. Because of relevancy, the resting-HR value is what contains meaning for the end-product. Therefore, the application was changed so that it would display today's resting-HR level and the previous seven days to be easier to compare and find out if the HR is elevated. This can be seen in Figure 3.3 with data retrieved from the HealthKit store during testing. The module was mainly revised for the code to be relevant and re-usable for the final application.

For this version, the main functionality was accomplished using the example from Listing 1. The main difference between the two versions is that this improved version uses a statistics collection query to retrieve the resting-HR values for the last 7 days. This allows the application to display the different

days of data. Besides, the main reason for the different queries is how the HR and resting-HR data is stored. resting-HR is stored as one value for each day. While normal HR data is stored as one value for each time point it has a value registered, making the different queries fetch data according to these differences.

Notification Module

The notification app will be developed for an iPhone with a paired Apple Watch. The iPhone can be used as a notification controller, and the Apple Watch will be used to display notifications and testing. This way, the iPhone app can have multiple buttons for different notifications that we want to test, and the watch will hopefully display these correctly.

There are two main types of notifications: local notifications and remote notifications (also called push notifications). Local notifications create the notification content and condition such as time or location, which triggers the delivery to the user locally. Remote notifications are generated by a server through the Apple Push Notification service (APNs). This allows developers to push notifications to the user without having the app do any work. The applications in this thesis will focus on local notifications.

Figure 3.4: Normal Notification

Figure 3.5: Notification with actions

Figure 3.6: Actions of the notification

In the same way, HealthKit needs permission to store and read health-related data for the user. User notifications also require special permission prompted the first time the user opens the application. To keep delivering notifications without encountering errors, the app must therefore check if the user has given permission every time it starts up, as the user can go into settings and remove this permission at any time.

If the watchOS app has an iOS companion app like this example, the system will determine where the notification is to be delivered. This means that if the user is using their iPhone when the notification is delivered, the notification will be displayed on the iPhone. While when the iPhone is not in use, the notification will show up on the Apple watch instead. However, if the application is only for the Apple Watch or triggered by the Apple Watch, it will only be delivered on the watch. This must be accounted for further in the development process, considering how the application should communicate with the user.

You can see two different notification types tested in the three figures above, one regular notification, which only displays an alert with some text (Figure

3.4). Simultaneously, the other one has the same but includes three different action buttons that allow the user to interact with the application through the notification (Figure 3.5 & 3.6).

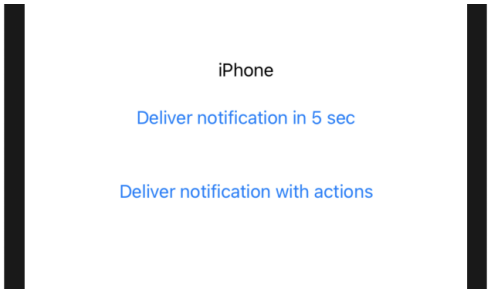

Figure 3.7: iOS App

Figure 3.8: WatchOS App

The last version of the app was developed with a user interface for both watch and phone (Figure 3.7 & 3.8), where each had a button that would trigger a normal notification. The phone application also had a button that would trigger a notification with actions. Because the notification is handled by the notification center, a default configuration of notifications is that they will not be delivered if the application that triggered them is in the foreground. This means that without a special configuration in our application, the notification will not show up by only pressing the button. Because our goal is to trigger notifications with an app running in the background, this special configuration will not be needed for testing purposes. The notifications will display in the simulator as long as the simulator changes the active app or turns off the simulator screen within 5 seconds of pressing the buttons. This is because the notifications in this example are programmed to be delivered after 5 seconds. By exiting the application, the notification can be delivered normally as the application is no longer in the foreground.

Background Module

The initial plan for the background module is to test the boundaries of background activity on both WatchOS and iOS to give a clear picture of what is possible when it comes to our desired app functionality. This is a difficult task because the devices decide what happens with the background applications and allocate resources unpredictably. It is important to explore different ways to achieve the desired background functionality to find the one best suited for the use case.

The first attempt of achieving the background functionality will be to use existing HealthKit queries, allowing us to run long-running queries in the background. These queries allow the application to update whenever a new set of a specified data type is stored or modified in the HealthKit store.

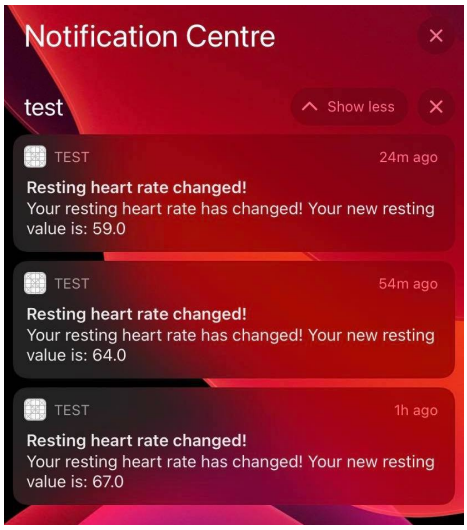

Figure 3.9: Notifications delivered from the Background App

Backlog

Due to the success in testing the background observer query and the time limitations, the following conceptual background functionalities were not developed, implemented or tested. During the research, a few different ways one could potentially acquire the desired background functionality were discovered. The most prominent solution was developed first to create an MVP in the shortest amount of time. Fortunately, the first solution developed was satisfactory, resulting in the following conceptual background functionalities not being developed during the work of this thesis.

1. **Background App Refresh/Background Fetch**
   Using background fetch to acquire background execution time has been partially implemented as it is what the observer query uses when enabled for background delivery. However, there was an idea to have the background fetch happen without a running query in the background. By doing this, the application could potentially check manually for new values every time it received a background fetch and then act upon those. (Apple Inc. 2021o)

2. **Background Push Notifications**
   The second idea was to have a remote push notification delivered to all devices with the app at regular intervals during the day. When an application receives a background push notification (an empty notification), it is granted background execution time to update or refresh its content. The app will stay up to date without the user having to update its content every time using it manually. The plan here was to have the application fetch the HealthKit data and act upon those by dispatching a local notification. This variant would work similarly to the background fetch, apart from forcing the background execution by receiving a background push

notification. This would be more reliable than "waiting" for a potential background fetch to occur. (Apple Inc. n.d.)

3. **WatchOS Complication**

Figure 3.10: Corner, circular and rectangular complications

The third idea would regard the Apple Watch, where the user can have complications on their watch face. The complications can display information or function as a quick way to open the desired app. The initial idea here was to display the resting-HR and show a warning sign with lots of color when the resting-HR was over the set boundary. This, however, could work in addition to the already developed application, as it is a small tool for the watch face and allows for an extra way of interacting with our application. (Apple Inc. 2021d) The different types of complications can be seen in Figure 3.10. In the applications WatchKit Extension, there can be found a file called "ComplicationController.swift". This file is where the developer can configure, design and implement the various complications. By having support for all the different complication types, the application allows the users to choose whichever watch face they like while still having the same functionality. Some complications are bigger and have the possibility to display more information to the user, but having the support for all variants would benefit the user in the best way as it leaves room for flexibility to their preferred watch face.

3.2.2 MedRem App

The MedRem application will be primarily built up by the existing functionality developed in the previous modules. Each module concerns one set of functionality from a given framework. The MedRem application combines these modules to achieve the desired MVP. The structure of the final application will be using the modules as displayed in Figure 3.1.

In addition to each of the three initial modules regarding one set of functionality, the MedRem application also includes a new module with functionality from the EventKit framework. This module allows the MedRem app to schedule reminders through Apple's Reminders app.

MedRem will be developed with a companion WatchKit app for the Apple Watch. This companion app will only copy the HealthKit functionality to display the same data on the iPhone app. The companion app will have no other function than providing the patient with the opportunity to view their data on the watch instead of the phone. We look away from the companion app for the rest of the thesis as it is only an additional user interface and does not contribute to the application otherwise. The flow of which data is stored, transmitted and then read by the MedRem application is as follows:

1. Apple Watch continuously reads HR values throughout the day. When it has read enough resting/sedentary values, it will create or update the resting HR for the user.
2. The values read by the Apple Watch is stored in the HealthKit store as long as the device is unlocked and the user has given permission for it to do so.
3. MedRem will read the resting HR values from the HealthKit store whenever it is notified of a change.

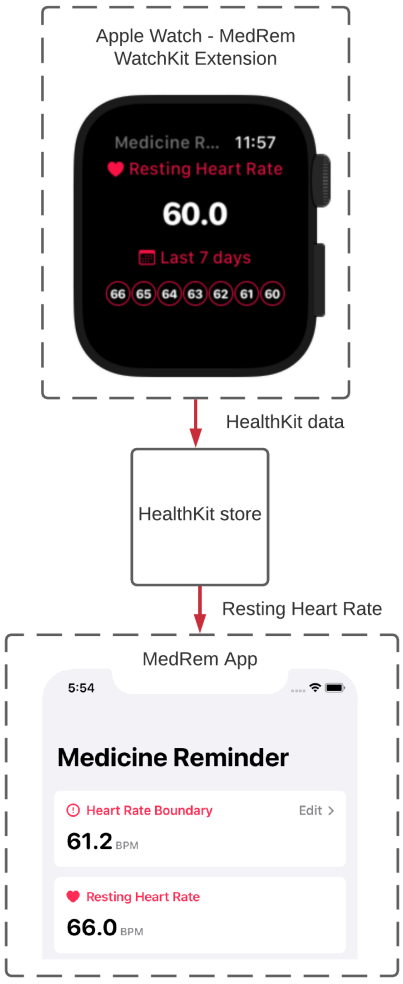

Figure 3.11: Data transfer in MedRem

Thus, the application's flow depends on the HealthKit store being updated and not on the Apple Watch directly, as displayed in Figure 3.11.

3.3 Data Structure

The different data used in MedRem are stored as variables in an observable object named UserData. This object is instantiated in the App's environment such that the different views have access to use and display the different variables. While the health-related data is always fetched from the HealthKit store, some settings data has to be stored on the device for the application to remember its state between sessions. The following variables are stored in the UserDefaults key store:

1. **Trigger boundary**
   Variable of the type Double. Used to check if the current resting-HR value is elevated or not.

2. **Dynamic boundary**
   Variable of the type Boolean. Used to determine if the trigger boundary should be dynamically adjusted when the user has a false alert.

3. **Dynamic boundary Gap**
   Variable of the type Double. The gap value is used to determine how much the trigger value should adjust dynamically when the user has a false alert.

4. **Warning dates**
   Array of Date objects. Implemented for testing and future use to show logs when the user has been alerted and is not currently used in the application.

3.3.1 Data management

The only personal data that is temporarily stored to display information in the application and notifications are the users' resting heart rate. The resting heart rate is stored as one value each day, where only the last 7 days' values are stored. There is no way to determine who this set of resting heart rates belongs to. The data is not to be considered as critical and in need of encryption because it is only temporarily stored while the application is open, and cannot be used to identify the user personally. The data is fetched from the HealthKit store, which is already encrypted for the users' privacy. The app also temporarily stores settings as variables that the user configures within the application. iOS apps use a class named UserDefaults to store data locally across launches of apps. This class will be used to store important data such as the settings and logging information. UserDefaults is an insecure repository stored in an open plist binary format, where you only need a key to retrieve a stored variable. Thus, only the settings and non-personal information is stored there, while the application will fetch the HR data from the HealthKit store every time it needs it.

3.4 User Interaction & Experience

Before developing a specific application, one should consider, plan and design the user interface beforehand such that the development phase becomes more of an implementation of the UI with minimal testing. When designing the user interface, one must also consider how the application is to be interacted with and experienced by the user. In this design phase, three different key elements are taken into consideration: how it's used, how it looks, and how it's experienced.

However, the user interface for the MedRem application is not as important because it is not the app itself that is the key element communicating with the user since the app is meant to run in the background. The user notifications are the key communicating elements of the application, which also requires interaction from the user through actions.

3.4.1 Apple UI Guidelines

In addition to documentation of the various frameworks and tools Apple has, they provide the "Human Interface Guidelines". The human interface guidelines is an extensive set of guidelines on how every framework to every component is recommended to be used. The guidelines provide different design principles on forming the user interface and recommended app architecture. For iOS, the guidelines provide 6 main design principles to maximize impact and reach:

1. **Aesthetic Integrity**
   The app's appearance and behaviour should integrate with its function.
2. **Consistency**
   The app should incorporate features and behaviours in ways people expect.
3. **Direct Manipulation**
   Items in the app should be affected immediately when the user interacts with them.
4. **Feedback**
   The apps interactive elements should provide feedback to the user, for example, making sure the user knows what element is selected by highlighting it.
5. **Metaphors**
   The app's behaviour should reflect on gestures that are familiar to the user. For example, changing a page should be done by a flick or swiping behaviour, mimicking the action of physically changing a paper page.
6. **User Control**
   Making the user feel like they are in control, but also warn about dangerous consequences. For example, confirming destructive actions and providing fallback possibilities.

(Apple Inc. 2021i)

3.4.2 User Notifications

The MedRem application uses primarily two different notifications. One for warning the user about their resting-HR, and the other to ask if they want a reminder for later if they have, in fact, forgotten their medicine.

For the warning notification, it is important to grab the users' attention and display important information to inform them about its occurrence. The notification is displayed in Figure 3.12. The text tells the user about their current resting-HR value and their boundary value and then asks them if they have remembered their medication. The notification has to be direct and make the user understand why they have been warned. This is achieved by telling the users about their current values.

The reminder notification asks if the user would like a reminder to take their medicine at a later occasion, perhaps because they are unable to take it at that current time. The key element of this notification is the actions that come with it, where the user is prompted to choose between receiving no reminder to how many hours until they would like to be reminded. The notification actions will then schedule a reminder through the EventKit framework.

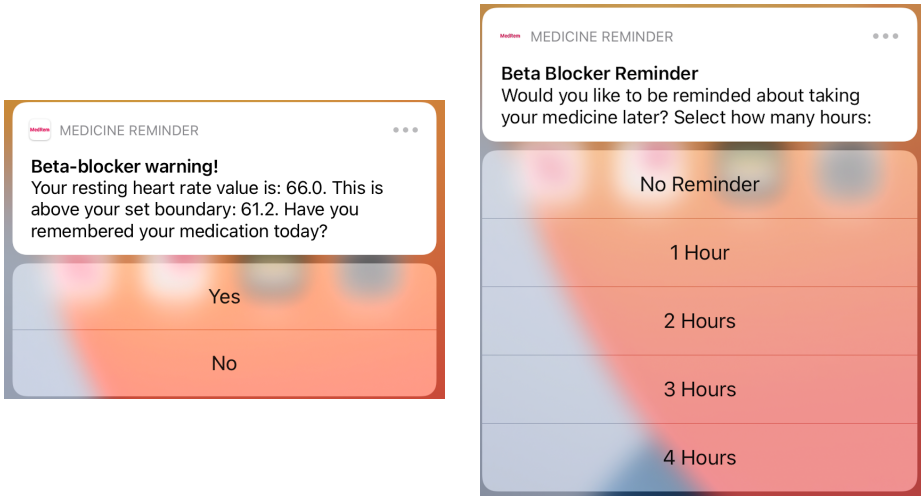

Figure 3.12: Warning and reminder notifications

3.4.3 User Interface

The user interface of the MedRem application is designed to reflect on the human interface guidelines. The final design of the implemented user interface can be seen in Figure 3.14. The screenshots of the application provided further in this thesis will be split 50/50 between dark and light mode to display the application supporting both appearances. Apple made Dark Mode available for iOS 13.0 and later, updating their UI guidelines to recommend supporting the new system-wide appearance. (Apple Inc. 2021f)

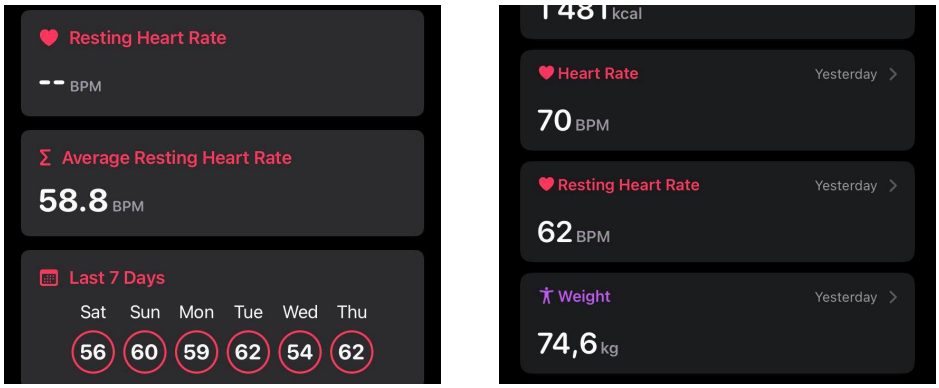

Figure 3.13: Comparison of MedRem (left) and Apple Health (right)

Since the application is meant for the background, there is no need for a unique and bespoke user interface. Apples Health app has been a major source of inspiration for how the user interface was designed, as shown in Figure 3.13. In the same way Apples Health app displays each health type and value in a SwiftUI GroupBox view, the MedRem app does the same with the boundary value, current resting-HR, weekly average resting-HR and the resting-HR values of the last 7 days. Having a similar interface as Apples Health app enforces the

second design principle in Section 3.4.1: Consistency, making the application more familiar and intuitive.

The settings page is found when pressing the boundary value GroupBox which has the "Edit" sign in the top right corner. This page also consists of group boxes that are slightly different, containing text inputs and a toggle. These text inputs and toggle are used as variables in the application. The users can change them, and when saving their changes, they are prompted with a warning, verifying if they are sure they would like to change their boundary settings. This is done to enforce the 6th design principle in Section 3.4.1: User Control.

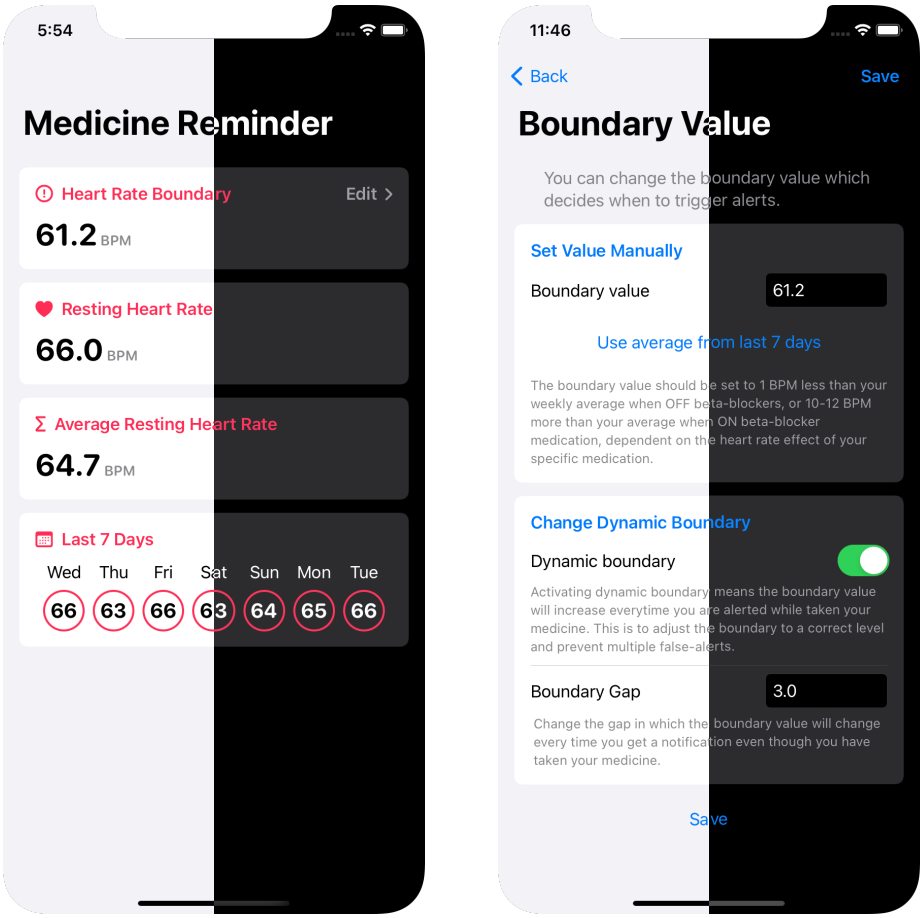

Figure 3.14: MedRem user interface

Chapter 4

Development

The background of app development is figuring out how to make the app look the same as the design and make it function in the way intended. This chapter will look at how the different applications use the various frameworks and data structures to create the desired functionality.

4.1 Resting Heart Rate App

4.1.1 Flow

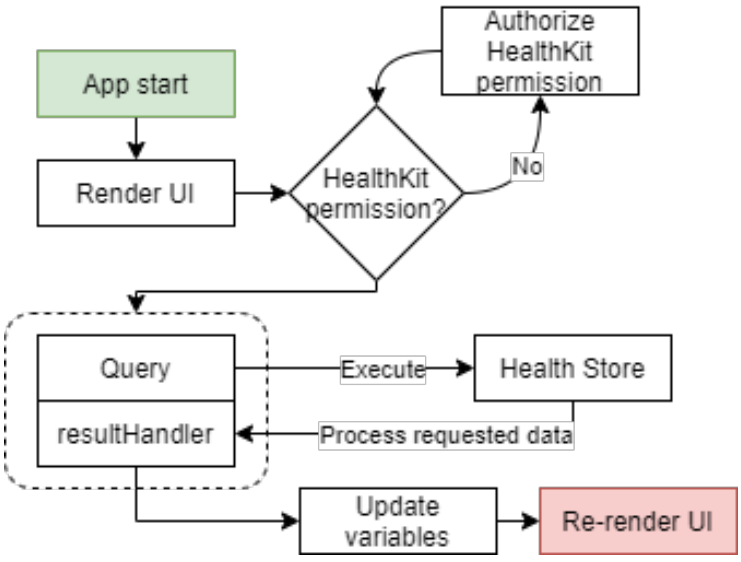

Figure 4.1: Flow-diagram of Heart Rate App

The heart rate app flow is simple, where most of the flow describes how health data is retrieved from the HealthKit Store using queries. First, to use the HealthKit framework, the app must check for permission from the user. If permission is given, the app can then execute a query in the HealthKit store. The store then returns the requested data in the query's result handler, where

the stored variables are updated to the new fetched ones. The user interface then updates the screen values because they have changed. The flow is identical to both versions of this app, where the only difference is what query is used.

4.1.2 Class Diagram and View

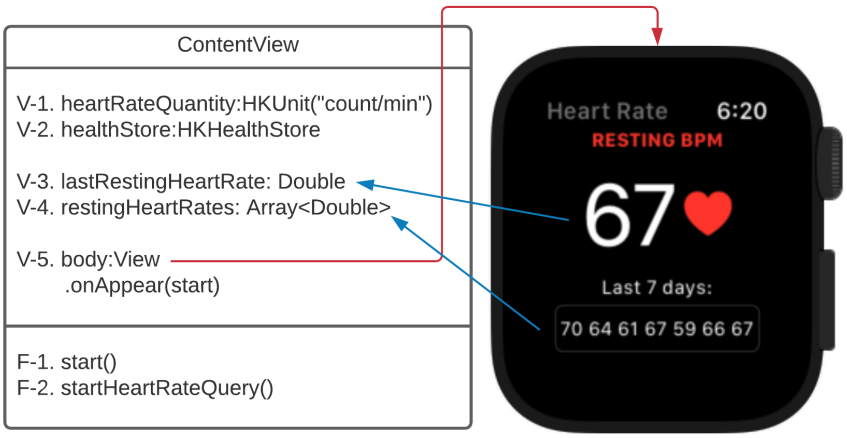

Figure 4.2: Class diagram and UI view of Heart Rate App

ContentView is not a class but a structure. In swift, the major difference between structs and classes is that structs do not support inheritance, type-casting and deinitializers. The ContentView struct is the main view responsible for initializing the content and layout in the user interface. For this task, the two needed functions are created as nested-function inside the ContentView for simplicity.

V-1 Constant used in the query to specify the quantity type returned.

V-2 Access point to the HealthKit store. Used to dispatch queries.

V-3 Variable to store the last resting heart rate.

V-4 List variable used to store the last seven days' resting heart rates.

V-5 View body of the app. This is where the interface is implemented. The blue arrow lines indicate text components in the interface using the appointed values. The onAppear(start) action calls the **F-1** (start) function upon the appearance of the view. Here used as a simple way of initializing the application.

F-1 Start function. Calls for authorization of HealthKit and calls **F-2** when the user gives permission.

F-2 Start heart rate query function. It is very similar to Listing 1 and includes a statistics update handler allowing the application to keep receiving updates from the HealthKit store every time a value in the scope is updated

or added. The function initializes a statistics collection query with initial results and statistics update handlers which updates **V-3** and **V-4**, then dispatches the query with **V-2** (the HealthKit store).

4.2 Notification App

4.2.1 Flow

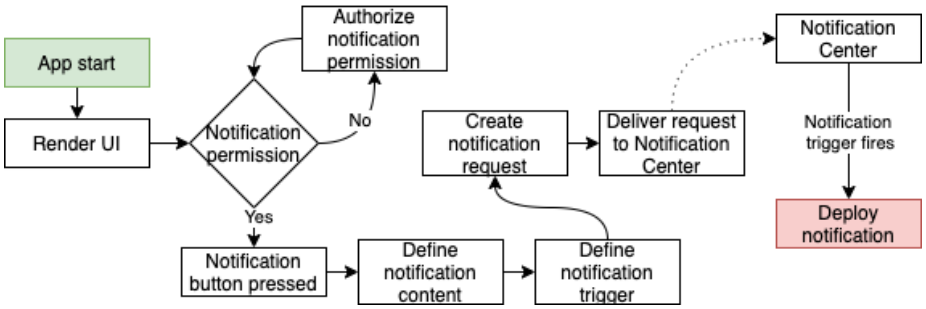

Figure 4.3: Flow-diagram of Notification App

The notifications app's flow is quite simple, as it mainly demonstrates the flow of any notification (Figure 4.3). In this example, the app requests authorization from the user when it initially opens instead of when the notification is scheduled. There is no particular reason for this other than that the final app will be using this way of requiring authorization from the user.

4.2.2 Class Diagram and View

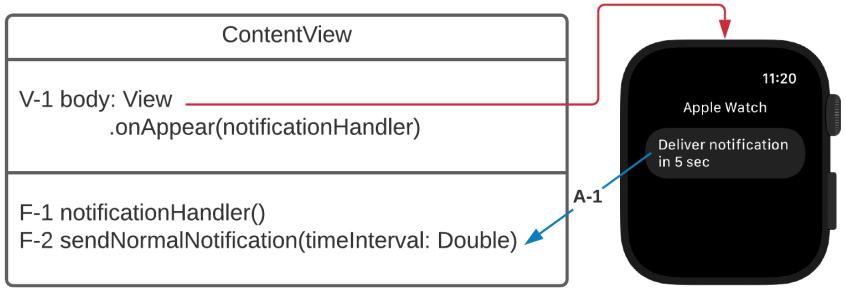

Figure 4.4: Class diagram and UI view of Notification App

The same as in section 4.1.2, due to the app's limited functionality, it uses nested functions because a proper class hierarchy is not needed for structuring.

V-1 View body of the app. This is where the interface is implemented. The *onAppear* action calls the **F-1** function upon the appearance of the view. Here used as a simple way of initializing the application.

F-1 Notification handler function is responsible for requiring the users' permission on startup to receive notifications from the app.

F-2 Notification dispatching function that takes a *timeInterval: Double* as a parameter. Similar to Listing 3, but with a *timeInterval* parameter added for flexibility in testing.

A-1 Button action. The Blue arrow indicates the **F-2** function being called when the button is triggered. As indicated in the text, the action calls the function with *timeInterval: 5.0* to have the notification delivered 5 seconds later.

4.3 Background App

4.3.1 Flow

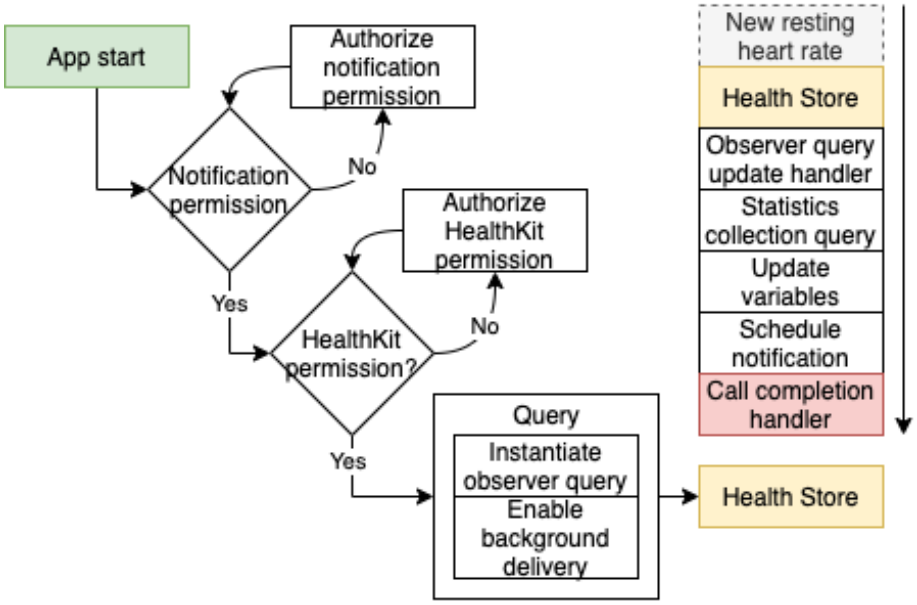

Figure 4.5: Flow-diagram of Observer query prototype

Simply implementing the background functionality would be difficult to test and verify. Therefore the background app uses functionality from the two previous applications. A flow diagram of this application can be seen in Figure 4.5. The application's main flow is the column on the right, which describes the events leading to a notification being dispatched. The left side concerns the actual application on the phone with setup, including authorizing User Notification- and HealthKit frameworks and the application instantiating the background observer query. This flow is very similar to the two Heart Rate apps' flow, except for using a different query.

Whenever a new resting heart rate value is stored or the existing value is updated, the HealthKit framework will trigger the observer query's update

handler. The update handler will then call for a statistics collection query that updates the app's existing variables. If the resting heart rate is, in fact, changed, it will schedule a notification in the notification center. The notification is then supposed to be dispatched, and it is possible to verify if the background functionality is working or not.

4.3.2 Class Diagram and View

This application has the same user interface as the Resting Heart Rate App and has some identical elements in the class diagram.

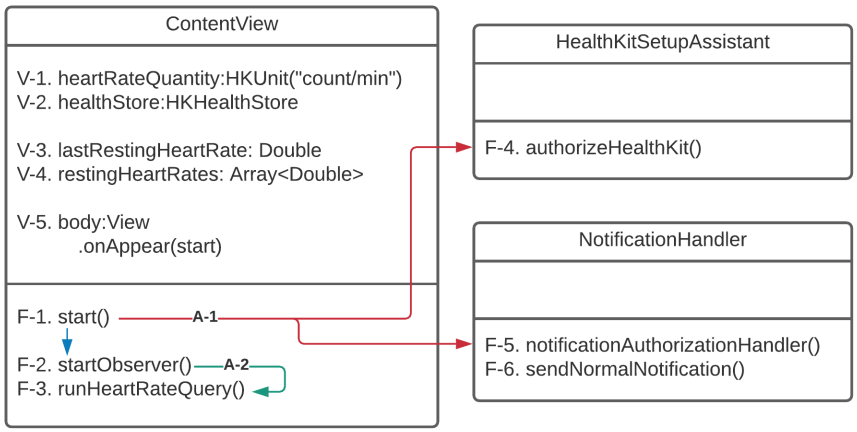

Figure 4.6: Class diagram of Background App

V-1,2,3,4,5 the same as in Figure 4.2.

F-1 Start function. Initializes the application by calling **F-4** to authorize the required HealthKit permissions, and **F-5** to authorize the use of User Notifications. When the required permissions are obtained, the function starts the observer query by calling **F-2**.

F-2 Start observer query function. This function initializes an observer query with a completion handler which calls the **F-3** function to fetch the observed change in resting heart rate.

F-3 Run heart rate query function. Almost identical to Listing 1, but also checks the fetched HR and calls **F-6** with the required parameters to dispatch a notification.

F-4 Handles permission to read/write health-related data, specifically in this example: resting heart rate.

F-5 Handles permission to display notifications.

F-6 Notification dispatching function. Similar to the one in section 4.2.2, but with two additional parameters: *title: String, body: String.*

A-1 Red line illustrates **F-1** calling **F-4** and **F-5**.

A-2 Green line illustrates **F-2** calling **F-3** every time the observer query is triggered.

4.4 MedRem

The MedRem application was, in the first instance, developed as a slim MVP to begin testing. It used the previously explained functionalities and combined them with a new and improved user interface. The demo was the first app to be tested on people other than the developer to discover bugs and gain feedback for further improvement.

After an initial round of feedback during the ongoing user testing, a second improved version was developed to address critical issues and add functionality that would cause major improvement to the user experience. The two major additional functionalities which were added to the second version will be explained later in sub-section 4.4.4 and 4.4.5.

4.4.1 Flow

The completed demo application flow is very similar to the Background App because it also uses the functionality from the Heart Rate app and the Notification app to test if it is working. However, in the MedRem application, the amount of notifications dispatched has to be limited by certain parameters. The current flow of how a notification ends up being dispatched is as follows:

1. User unlocks phone which allows the device to access the HealthKit Store
2. If there is a new resting heart rate value, the background observer query will be triggered and call its update handler.
3. The update handler calls for a statistical collection query to retrieve the new value(s).
4. The values in the app is then updated with the new ones. And the app checks:
    - If the most recent resting heart rate value is valid.
    - If the resting heart rate is higher than the set boundary.
    - If the time is between 14:00 and 20:00.
    - If it is at least 10 minutes since the last notification was dispatched.
5. If the previous parameters pass, the app schedules an actionable notification to warn about the possibility that the user has forgotten to take its medication, based on the resting heart rate value.

After the notification has been dispatched, it is up to the user to react to the notification's actions. If the user answers yes to whether they have taken their medication, the boundary value will be increased to adjust because of a "false" warning.

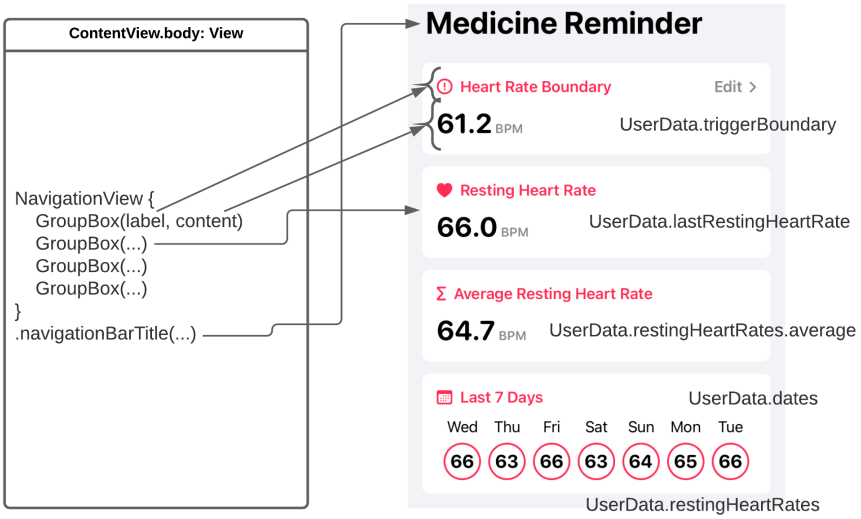

Figure 4.7: Swift UI View setup

4.4.2 SwiftUI Views

The UI of the MedRem app is created using the SwiftUI framework. As displayed in Figure 4.7, the main page's view is only built up by four GroupBox views inside a navigation view with an assigned title. Each of the GroupBox views is provided with a label and content. We have used Apples Label structure for the label, allowing us to create a label with an icon. The content section of the GroupBox allows for more flexibility, where it is possible to assign custom styles. In Figure 4.7 there is one style used for the first three GroupBoxes, which displays one HR value with a unit type, and one style for the final GroupBox, which displays 7 days and their respective HR values. On the right side of the screenshot, it is written the respective UserData values used by the different view components.

4.4.3 Internal Architecture

Even though the MedRem application has almost the identical flow of the Background App, MedRem is more complicated structured internally. The app is divided into 6 different main sections, as displayed in Figure 4.8:

1. **AppDelegate**
   The main controller of the application. Because of the background running query and authorization handling, the app uses an AppDelegate to handle the app lifecycle. The AppDelegate initializes the app by doing the authorization handling and starts the different HealthKit queries to fetch information to the app and begin the background functionality. The AppDelegate has instances of the Health, Notification and Event-handlers such that it can authorize the different frameworks and manage their behaviours. In addition to these responsibilities, the AppDelegate also

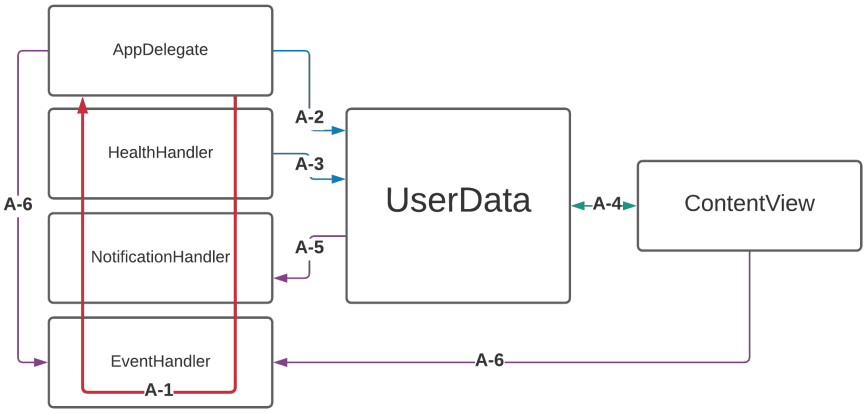

Figure 4.8: Class communication diagram

handles the different actions a user can perform through an actionable notification.

2. **HealthHandler**
   The health handler is the class where everything related to the HealthKit framework is handled. It consists of primarily three functions concerning authorizing HealthKit, executing observer- and statistical collection queries.

3. **NotificationHandler**
   The notification handler is the class where the notification related part of the application is, apart from handling the actions defined in the actionable notifications. This class has functions for dispatching the different notifications used by the application and authorizing the User Notification framework to be used by the application.

4. **EventHandler**
   The event handler handles authorization of EventKit and schedules reminders to the Reminders app.

5. **UserData**
   The brain of the application is the UserData class. This is where all app- and HR-related data is temporarily stored and handled. UserData is also an observed object which allows the various views and functions in the application to access the different user data. As well as being central storage of information, the class also decides to dispatch notifications based on the incoming data.

6. **ContentView**
   ContentView works as the face of the application. Because MedRem does not use the SwiftUI-lifecycle, it has a SceneDelegate which assigns ContentView as the root view. As the root view, ContentView has to handle navigation and the appearance of the app. Because the app is purposed for background use, the app has a simple user interface with very few

items and views to handle for navigation. The main navigation in the app is between the main page and the settings page.

How the different sections communicate is illustrated through the arrows A-1 to A-6 in Figure 4.8. The arrows indicate the following:

A-1 This communication line mainly regards the AppDelegate using its objects of the respective classes to authorize the different frameworks and functionality.

A-2 AppDelegate accesses various functions in UserData to reflect changes made from notification action answers.

A-3 HealthHandler communicates with UserData by supplying it with new resting-HR values every time the observer query triggers, as well as when the application is launched.

A-4 ContentView and the various sub-views has instances of the environment object of UserData, using it to display the different values throughout the application.

A-5 UserData checks the values it is supplied with from the health handler, and if the right parameters pass, it will schedule a notification through the notification handler.

A-6 Dependent on whether the user responds to the app through the notification actions or in-app actions, the AppDelegate or ContentView will use the event handler to schedule a reminder for later if the user asks for it.

4.4.4 In-app Notifications

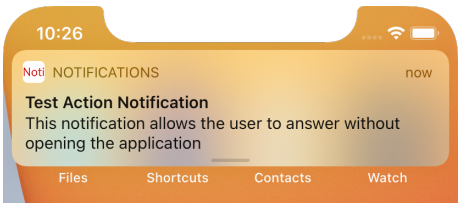

Figure 4.9: Action notification

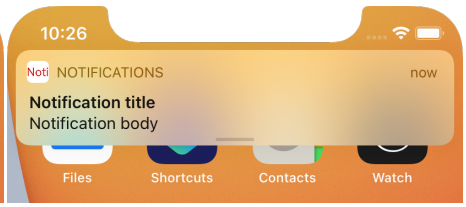

Figure 4.10: Normal notification

When a notification with actions is delivered to the user, it looks the same as a regular notification. This can be seen by the non-existent difference in Figure 4.9 and Figure 4.10. If the title and body of these two notifications were identical, the only way to tell them apart would be to swipe on them to display the actionable notification actions. Because it can be difficult to know when to swipe on a notification to see actions, most users only press the notification, which opens the associated application.

Therefore it was decided to implement a way for the application to prompt the user inside the application, and not just through a notification. This way, the user can interact with the alert even though the user might have dismissed the first notification or accidentally clicks the notification instead of swiping

on it. The implementation of this can be seen in Figure 4.11 where both the notification and in-app notification are displayed next to each other.

This functionality is achieved by setting a boolean value in UserData called *notifyQuestion*. In the apps' main view, an alert is added whenever the *notifyQuesiton* boolean is true. Whenever the app decides to dispatch a notification to alert the user, the *notifyQuestion* boolean is set to true so that the user will encounter the alert in-app if they accidentally click the notification instead of swiping it. If the user interacts with the notification, the *notifyQuestion* boolean is set to false such that the user will not encounter the same question twice.

Also, there is another boolean: *remindQuestion*, which is set to true whenever the user interacts with the notifyQuestion-alert. This is done so that an action sheet(right side of Figure 4.13) will pop up after the alert to ask if the user wants a reminder later.

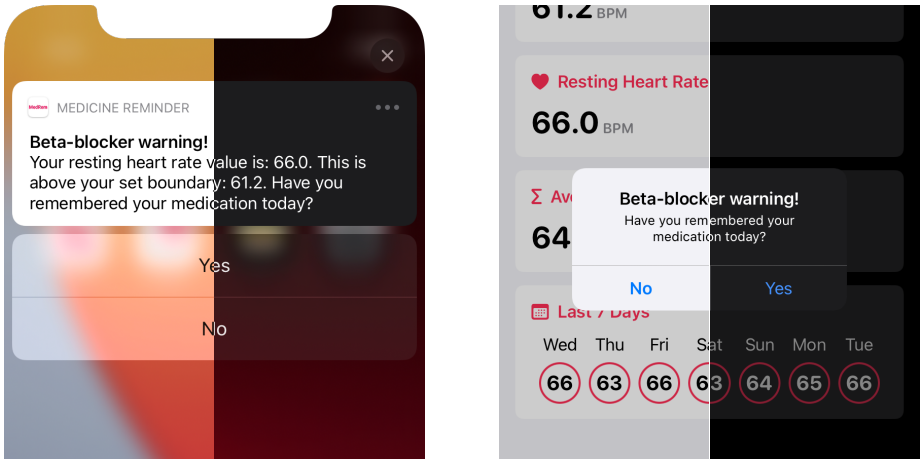

Figure 4.11: Notification actions and in-app alert

4.4.5 Reminders

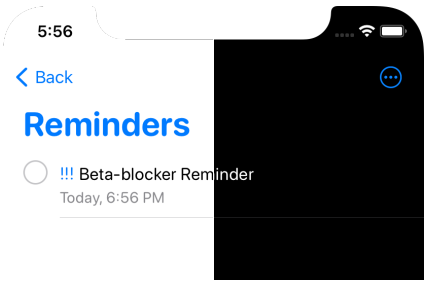

Figure 4.12: Reminder shown in Reminders app

Considering a scenario where the patients cannot immediately take the medication after being alerted, it was decided to implement a way for the application to again remind the patients after a set amount of time decided by the patient. It is possible to schedule a reminder in Apple's Reminders app through the EventKit framework. Using this framework, MedRem can schedule a reminder for the patient and give them options to choose between 1-4 hours to be reminded again to take their medicine. This way, when the patient is unable to take medicine after being warned, the patient can choose to be reminded a time later when they are hopefully back at home or in a situation better suited to take the missed medication dose. The reminder notification and the in-app notification can be

seen in Figure 4.13, while the actual reminder stored in the Reminders app can be seen in Figure 4.12.

This functionality is achieved by scheduling a second notification or in-app action sheet when the patient has interacted with the first notification/alert. Doing this, the patient will immediately, after interacting with the first notification/alert, be prompted with the new question, whose action will schedule a reminder through the EventKit framework.

The only catch about this functionality is that the patient must have the Reminders app installed on their device. The decision to use EventKit for this functionality instead of a normal notification was made because it is a reminder, and notifications from MedRem should be used for alerts and warnings only.

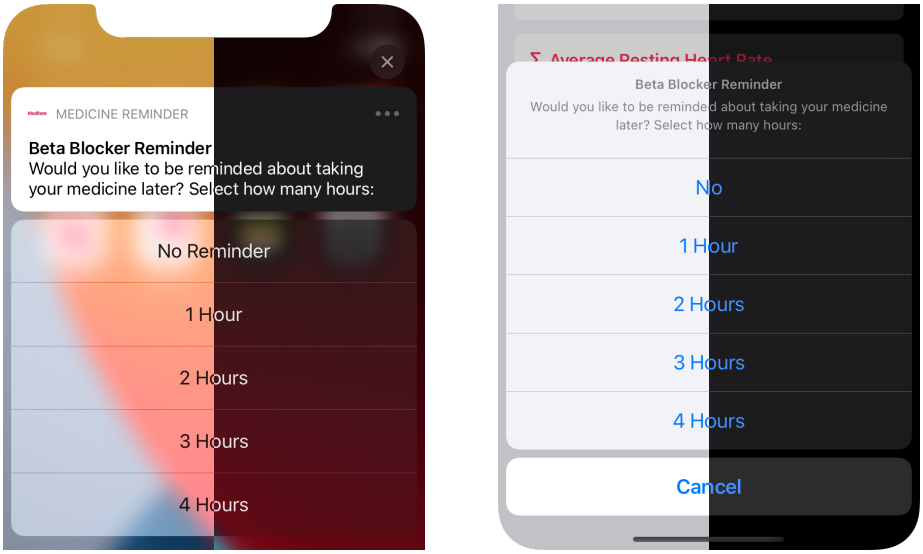

Figure 4.13: Reminder notification actions and in-app action sheet

Part III

Evaluation and Conclusion

Chapter 5

Evaluation

Experiments in this chapter are conducted to evaluate the final application's different modules and the final application in its entirety, and how users have perceived it. The test strategies explain how the modules are tested and how they are expected to perform.

5.1 Experiment A: Reading Heart Rates

This experiment was conducted on the resting heart rate app, testing the module to deliver the desired functionality of retrieving HR data from the HealthKit store. The goal of the experiment is to find a satisfactory way of reading and storing resting heart rate values in the application and discover different boundaries related to the frameworks needed to achieve this functionality.

5.1.1 Testing

The heart rate app and the resting heart rate app's core functionality were easy to test in the experiment as very little had to be done to see different results. The application was primarily developed with the example from Listing 1 and with certain changes to the quantity type and the query type to see how good results the app gets from the HealthKit store.

The first testing of this functionality was done with the heart rate app, using a sample query to fetch a snapshot of the latest heart rate values in the HealthKit store and displaying the latest. The app should then display the user's current heart rate at the time the user opened the application.

Further, the app was changed to use a statistical collection query with an update handler. Displaying the same value as before, but having the update handler to update the visual value in the user interface every time the value changes.

The next part of the testing was done by changing the statistical collection query to fetch the last 7 days worth of resting heart rates. The UI was changed to show the 7 values, including the one for the current day.

5.1.2 Results

Being a rather simple functionality, in the end, it is important to have the correct value displayed to use it as a parameter later. The first part of the test was done to figure out in the simplest way how to retrieve any value from the HealthKit store. Being successful at retrieving the heart rate, the app was then progressed to a more advanced query with more options for fetching more specific values, including an update handler allowing the value to be updated automatically. Again successfully retrieving the value and having it updated by itself, the query was changed to fetch resting heart rates. The resting heart rate value is the more interesting result of the test. Because regular heart rates are stored as values for each time point they are registered, it was easy to assume the resting heart rate would be the same, just at a rarer frequency. However, that was not the case. The testing showed that the resting heart rate was stored as one value for each day that had a resting heart rate stored. Meaning the resting heart rates are stored as a maximum of one value per day and not per time point where the value is registered. The value of the day is then updated, making it important for the application to check for updates continuously.

5.2 Experiment B: Dispatching Notifications

This experiment was conducted on the notification app. The goal of the experiment was to test the different notifications and options available. That includes figuring out how to create, dispatch, and manage notifications and how the notifications behave on iOS and WatchOS. The notification module is important to understand as it works as the communication part of the final application. Testing different types of notifications and understanding how these are configured simplifies adapting and customizing the notifications for the MedRem application.

5.2.1 Testing

The notification app is a minimalistic application for both iOS and WatchOS, where each version contains buttons to trigger a notification dispatch. On the iOS version, it was also added a button to trigger an actionable notification. The testing was done by pressing the button to trigger the notification delivery and then exiting the app or turning off the screen to have the notifications appear. The notifications were also configured with different payloads and trigger conditions to find the best-suited type for the MedRem app.

5.2.2 Results

This experiment gave insight into how notifications are dispatched and delivered. The normal and actionable notifications were delivered just as predicted. However, the actionable notifications required more configuration to handle the actions triggered by the user. This configuration was difficult to get right at first because it requires a notification delegate to intercept the action response and act on it accordingly. The actionable notifications came along successfully after some research and attempts. Adding the possibility for repeating notifications could be the solution of reminding the user to take their medicine at a later

time. Because it is possible to cancel a scheduled notification, it would be a viable approach as a reminder. However, this would require more UI configuration to allow the user to cancel their scheduled reminder notification, and it would require the app to keep track of notifications being scheduled, cancelled or delivered. Having tested this possibility, it was decided to keep notifications as on-time events with actions and rather implement a solution using EventKit for the reminder. Having tested and reconfigured the application to satisfaction marked the finish of this module.

5.3 Experiment C: Running Background-Tasks

This experiment is the last experiment focusing on one module/functionality particularly, and not a combination of modules. It is, however, tested through the use of other modules. It is difficult to test if an app or functionality is running correctly in the background. Therefore this module, as explained earlier, uses the two previous application to test if it works.

5.3.1 Testing

The testing of this module was done using a HealthKit observer query, as explained in Section 2.2.1 is a long-running query that can keep going in the background, even when the application is closed. This query is used to detect changes in HealthKit store data, specifically the resting heart rate of the user in this example. Whenever the resting heart rate value changes, the observer query should be triggered, which should trigger a statistical collection query to fetch the new resting heart rate. When the new resting heart rate value is fetched, the statistical collection query's completion handler should dispatch a notification stating that the user has a new or changed resting heart rate value.

This module can be tested in the XCode simulator by filling resting-HR values into the Health app. Whenever a new value is added, the applications observer query should be triggered as an observed value is changed or added. If the application is not immediately granted the background execution to dispatch the notification, it is possible to force it by using the "Simulate Background Fetch" tool in XCode. Whenever activating this option, the application will be granted background execution time. If the applications observer query has been triggered, it will then fetch the values and dispatch notifications.

To test the application for real-life purposes, it was also tested on a real iPhone device connected to an Apple Watch. This part of the test was the most important because the simulator testing would prove the concept to work, but how well the solution would apply to real-life had to be tested physically.

5.3.2 Results

The physical testing of the application gave insight into how rarely the resting heart rate value is updated and that there is no way to predict if the application is granted background execution when the observer query is triggered. Similarly to simulator testing, the Health app on the physical iPhone device can be used to enter manual data, which can trigger the observer query. Also, it was not guaranteed that the notification would show immediately, as the app running

in the background only was immediately granted background execution about half of the times. The other times when the app did not receive background execution time immediately, it could take between 5 minutes and a few hours before the application was finally granted the background execution time. This testing proved that the app would be able to deliver notifications eventually.

To test the background execution's viability, the app was tested over a longer period without altering the resting-HR data through the Health app. The testing was conducted by wearing the Apple Watch for at least 12 hours per day and checking if the application could deliver notifications at least once per day, preferably sooner in the day rather than later. Considering the resting-HR value only being updated a handful of times per day, the application needed to detect and notify about most of these changes in a reasonable amount of time. Despite the testing of manually adding resting-HR values to the Health app gave little motivation for the application to work in a real-life setting, the application did dispatch several notifications each day. The results were even without the interaction of the notifications or the application, proving the app could run in the background frequently enough for the module to work in the final application. The success of the background functionality marked the completion of this module.

5.4 Experiment D: The MedRem App

This experiment can be seen as an acceptance test before the user testing. The test was conducted to check if all the different modules worked with each other and that the application behaved as expected before the application was distributed to testers. Because the modules previously explained have been tested separately, the main focus of this experiment is to test the combination of the modules and any new additions to the app.

The experiment is the first to test an application with a particular user group, consisting of both a patient and their care provider. This means the testing has to discover if the app is perceived correctly in the way it is supposed to be used and not just test if the application works as intended.

5.4.1 Testing

To verify that the application is ready for user testing, the three previous modules' main functionality has to be tested through the MedRem app. In addition to these, the app must be tested more thoroughly, making sure that notifications are only dispatched when the right criteria are present.

The MedRem app is very similar to the previous experiment considering the main difference is a user interface, minor functionality, and parameters used to decide whether to warn the patient. Similar to the previous experiment testing the background execution, the MedRem app was tested first through the XCode simulator and later on a real device for a few days. The test covers two main areas: user experience and functionality. The user experience part of the testing is conducted on the user interface to see if it looks as it is supposed to and that the user interface's different elements are interactable in the way intended. This also includes interacting with the notifications and deciding how

well they are perceived, considering the text formulation and information put into the notification body.

The application's functionality testing should primarily focus on the main goal of the app; to warn the patient whenever their resting heart rate is above the set boundary/threshold. Also, considering other parameters such as the time of day and the time since the last notification was dispatched. Because of these parameters, the notifications are only supposed to be dispatched within a time frame, meaning the test must make sure the notifications can only dispatch within the time frame. Also, notifications should only be dispatched when the patient's resting-HR is above the set threshold.

5.4.2 Results

The user experience part of the application was simple to verify because it could be tested by the developer simultaneously as it was implemented. Also, the application's interactable parts were tested during the implementation, making sure the implemented code performed as intended. During testing, it was discovered that some items of the user interface did not scale properly on different iPhone devices, as the range of iPhone models uses very different screen resolutions. This was uncovered when testing the MedRem app on different simulated devices through XCode. The most probable cause for this first being uncovered during the acceptance testing was that the user interface was primarily focused on the developers iPhone 11. Instead of running the application on a simulator, the developer can load it instantly to their own device to test and interact with. Thus, some minor parts of the user interface were not tested to scale properly and later fixed after these results.

The MedRem app was in the same way as the background module, simple to test in the simulator. By providing the Health app fictive test data and changing the different settings in the MedRem app, the warning notifications would either be triggered by the resting-HR change, or the simulator could simulate a background fetch, enabling the application to do its checks and potentially trigger the notification. A limitation to the testing was discovered due to the time frame notifications can be dispatched. Because of this time frame, the testing must also be conducted within this time frame to yield the results. For testing purposes, the time frame was altered during the simulator testing to test the application at any given time of the day. Apart from these two minor issues, the simulator testing proved the application to work as intended.

The application was also tested on an actual device for a longer period of time. By setting the boundary value/threshold to an unrealistically low value, the app should dispatch warnings every time the resting-HR value is updated within the given time frame. This worked better than expected. Having the warning appear a few times each day also allowed for testing the dynamic boundary, increasing the boundary value/threshold if the patient claimed to have taken their medicine. Though after a random amount of hours, the application would crash. The application's crash reports in XCode can tell where in the code the application crashed. Apparently, sometimes the statistics collection query would return without a result. In the code of this first version of the application, the results handler of the statistics collection query was told to terminate the application if it returned with an error. Changing the application to ignore the error and not update anything if it occurred fixed this issue.

Overall the application seemed to work as intended, and after fixing the minor issues encountered, it was decided that the application was ready for user-testing.

5.5 Experiment E: Performing User-Tests

One crucial part of the application is that it should be simple to understand and easily adapted by patients and their care providers. Also, to test if the application works as an MVP, this experiment was conducted to determine how well the app is received by patients and external users. The application was distributed to four external testers, who wore Apple Watches and indirectly tested the app between two and ten weeks. The four testers were picked to receive helpful feedback quickly. As the purpose of the application is to work as an MVP, it was more important to test the application generally than specifically for patients with heart disease. For example, testing the application with users who understands it can give valuable feedback for the application in general, while later testing the application on patients would serve purposeful for the actual use-case of the application. This test is conducted to receive feedback which will prepare the app for testing real patients.

None of the participants in this experiment was sick, suffered from heart disease or were prescribed beta-blocker medicine. All testing was completed by having the app react to the user's normal heart rate and simulate a heightened heart rate by lowering the resting heart rate threshold/value for the application to dispatch warnings.

To get a broader understanding of how well the app was perceived, the four users can be divided into two different age groups, where two of the participants were in their 30s and the other two in their 60s. The importance here is to see if both age groups are able to understand the application and use it properly, even though it's a very hands-off application. It was also important to test the application on the elder age group as they are in the age range the application would target in a commercial setting. The participants have different professions; civil engineer, researcher, professor and adviser. Having different backgrounds helps to diversify how the application is tested and interacted with. Furthermore, two of the participants, who were in different age groups, were already acquainted with the concept because they were the ones who came up with the idea for this thesis. Using the concept creators of the application as testers, with two extra external testers who are unacquainted with the concept, gives a broader pool of observations because the observations from the concept creators will verify how the application relates to the concept and their projection of the solution. In contrast, the last two participants' observations will verify how externals perceive the application.

The four participants have different iPhone models, and the least experienced participant has used an iPhone device for 8 years. This means the participants are well known with the Apple platform and should navigate the application with little help.

The app was distributed through the TestFlight framework, as explained in section 2.3.2. The framework allows the application to be installed on the participants' devices normally as it would be through the App Store. Also, the framework gives the participants an option to send feedback to the developer,

enabling them to share crash data and provide screenshots with descriptions directly to the developer.

This experiment consists of two test phases and an interview part. The test phases are divided into preliminary testing and normal testing. The application was updated and improved between the two phases to encounter key issues discovered during the preliminary testing. This will be further discussed in the sections below. In the interviews, the participants were asked the same questions to get an overview of their overall experience of the application and how they tested it. The interviews gave valuable feedback, which will be discussed later.

5.5.1 Preliminary Testing, Early Results & Feedback

The preliminary testing was conducted on the two younger participants because only they had Apple Watches available at the time and could begin early testing. Later for the second phase, one of the preliminary participants' Apple Watch was lent to a third participant, and the developer's Apple Watch was lent to the final participant. This first test phase was primarily conducted to find out how others perceived the MedRem application, and if external participants could make the application crash and provide feedback on any "missing" functionality or something that was not working as well as it could have.

After two weeks, the two participants said in a short non-formal meeting that the application was working well but had a few areas they felt were lacking or missing. The two primary areas were receiving in-app notifications and the possibility to be reminded at a later time. Both of these functionalities were developed and added to the application and is discussed previously in section 4.4.4 & 4.4.5.

5.5.2 Testing

The application was distributed to the four participants through TestFlight. The two who already had it installed received an update through the framework such that everyone had the latest version of the app installed. Through the TestFlight app page seen in Figure 2.1, the participants were provided with the following description:

> **Paired Apple Watch required**
> Open the app, give the required permissions.
> When in the dashboard, it should be filled with resting heart rate data from the previous week. At the top, you should set a boundary that will decide whether notifications are delivered or not.
> Without using the app, see if you get notifications throughout the day.
> If something looks off or is wrong/unsatisfactory, you should take a screenshot and use the "Share Beta Feedback" when you try to share the screenshot. Also, you can use the "Send Beta Feedback" link in TestFlight to explain your troubles.

Also, the participants were instructed the following:

- Wear the Apple Watch at all times during the day and charge it while sleeping

- Test different threshold/boundary values to check if the application is dispatching notifications when it is supposed to.
- Try different answers to notifications, reminders and see how the app reacts accordingly.
- Try not to open the application except when changing settings, such that the background functionality can be tested over a longer period.

5.5.3 Interview

Each participant took part in a 10-minute interview consisting of a range of questions to get their thoughts on the application with regards to the applications: current state, usability, concept, and appearance. The participants were also asked questions about how they tested the application and about factors that could have impacted their test results. The interview guide can be seen in Appendix B. Because the interview followed a semi-formal structure, not all questions got a definitive answer, but having the participants talk around the different topics gave enough indication of the participants' impressions and thoughts. Following this section, each topic of questions will be clarified separately, along with their observations.

5.5.4 Observations

During the interviews, some observations were made on the different topics. These observations will be explained further in this section.

Testing Methods and External Factors

This question topic regarded how the participants tested the application, as in what features they tried, how did they adjust the boundary/threshold values and other factors such as how they wore the Apple Watch and if they had a tendency to take it off for different occasions. These questions were important because they set the background for why the participants may have had certain results.

The consensus between the participants for this topic was that everyone had adjusted the boundary value/threshold to some extent, receiving notification and testing the primary functionality of receiving warnings. The participants also charged the Apple Watch while sleeping and wore it from they woke up until they went to bed, apart from one exception.

All but one of the participants tried the new feature that lets the application make a reminder in Apple's Reminders app, which notifies the user after a set time. An issue encountered here was when one of the participants removed unused applications to free up space and deleted the Reminders app, making the missing feature crash the application.

A difference between some of the participants could be seen in who used watches daily before the experiment. Two of the participants did not daily wear watches, which resulted in them being more disturbed by the Apple Watches notifications and annoyance from wearing it in general. One of the participants ended up taking off the Apple Watch while working on computers, resulting in the resting-HR value not being updated before later in the afternoon. Other

participant said they would take the watch off while taking a shower or for a short period of time, but that would not have a great impact on the resting-HR value in the same way as leaving it off for hours while working at the computer.

Another issue encountered during testing by one participant is that the participant sometimes forgot to unlock the Apple Watch while wearing it. This resulted in the Apple Watch not being able to write data to the HealthKit store before it was unlocked, which could be sometime later in the day as the participant was not used to wearing it.

One of the participants experienced odd resting-HR readings with unnatural low values. The assumption made here was that this phenomenon could be caused if the user wore the Apple Watch too loosely, as the participant had improved readings after being instructed to tighten the watch band.

Current state

Questioning the participants about the current state of the application was achieved by asking questions such as how their overall impression of the application was, if it worked as expected and if there is anything they feel the application lacked.

For the applications current state, the consensus between the participants was that it is a simple application, which serves its purpose as an MVP. All the participants said the application works as they expected, especially considering that it is primarily meant to run in the background. When a participant was asked how usable the app is and if it covers a need, the participant said:

> *"I think it does. I think it covers a need. If you're not able to get this out of the Apple Watch itself, I think this works very well as a supplement for medicines."*

A few areas of improvement came up when the participants were asked if something was lacking with the application. These were the following:

- Bespoke motivational messages to engage the user and make it easier to grab their attention.
- Warnings to be clearer, with sound, vibration and alerts on both iPhone and Apple Watch devices.
- More information and a log stating when and why warnings had been dispatched.
- App on iPhone to remind the user/patient to wear the Apple Watch if the user/patient was not wearing one.
- A participant thought it could be an advantage to have the opportunity to have warnings delivered at one specific time during the day, such that the patient would know what was wrong immediately when receiving the warning.
- Application to check more thoroughly before dispatching a warning to limit the number of false warnings.

- History of how the patients resting-HR value has changed over time. Possible to check resting heart rates from larger periods of time and graph these to see the medicine's effect. This feature could be usable, especially for the patient's health care provider, but problematic in a data/GDPR sense.

Usability / User Experience

To get a grasp of how usable the application is for its purpose and if the participants have had a good experience testing the app, the participants were questioned about how simple it was to use the app, interact with different warnings and use the app in general. They were also asked if they thought the applications was usable in its current state and if they think it covers a need for patients with different heart conditions.

For the user experience, the consensus between the participants was that it is a very simple application to use due to its limited amount of functions and uses. Though for one of the participants who was unfamiliar with wearing a watch, it was more difficult to getting used to wearing the watch comfortably. The users also thought that the notifications were clear, well understood and was dispatched at reasonable times. Especially after the update that was published from the preliminary testing, the preliminary testers who then updated the app from the first version was pleased with how the application facilitated the notification actions in the app if they only pressed the notification. All participants said the app worked as they expected and dispatched warnings when it should have, proving its usability.

One of the participants also said:

> *"I really like how the threshold value is changed when I answer that I have taken my medicine, and if I haven't, the app provides me with new options to help me with reminders. I think the app serves its purpose if the purpose is to make patients take their medicine."*

Only one of the participants ended up not trying to have a reminder set for later. However, the other participants who used this feature were pleased and thought it worked well.

Appearance / User Interface

Because the application is an MVP, the user interface is not as important as the functionality. Therefore the participants were questioned on what they think of its appearance and how important they think the application's appearance is. A consensus between the participants was that they did not really pay attention to the user interface. They thought it was simple and fit the application's needs in a good way. Two of the participants pointed out that the user interface was good because it did not interrupt the application's usability and that the less they need to scroll, the better. Another participant said it looks good, but there is no need for it to look good because the functionality and interaction are in the notification warnings.

The participants were also asked about how important the looks of an application is for them. Two of the participants answered that they thought it was important that applications look simple and are readable. Another participant

said the MedRem app should look more clinical to tell the user it's a medical supplement, but the looks of applications were not that important otherwise.

5.5.5 Discussion

One of the issues encountered during the testing was that the app would crash if Apples Reminders app were uninstalled from the device. The downside with this functionality is that the user needs two applications installed for it to work properly. The upside is that the MedRem app uses existing frameworks on the Apple platform and that the Reminders app handles the reminders. This makes the MedRem app more lightweight and simplified the development needed for the functionality. The fix for this issue would be to implement the app to check for the Reminders permission to be given and that it's able to use the EventKit framework without any errors. If the framework does not throw any errors, it can provide the reminder option to the user.

As mentioned in section 5.5.4, two of the users were not used to wearing watches daily. When it comes to the core need of the MedRem application, the Apple Watch is only used for reading the patients resting-HR. If the user has no other reason than the MedRem app to wear the Apple Watch, it can be easily seen as more of an annoyance than a tool, as the application passively uses it. Therefore, the MedRem application is more suited to people who are used to wearing a watch or who has other benefits from wearing a Apple Watch device on a daily basis.

Another Apple Watch issue was that the users could forget to unlock it when initially putting it on in the morning or that the user forgot to wear it in the first place. A solution for implementing the functionality required to notify the patients that they are not wearing the Apple Watch will be described in Future Work section 6.2.2: Reminder to Wear Apple Watch. In the same way, MedRem will remind the patient to wear the Apple Watch, and it will also trigger a reminder if the patient has forgotten to unlock the Apple Watch. This is because the solution is based on checking if there is existing HealthKit data stored by the Apple Watch on the same day, and if the Apple Watch is locked, it will not have had the permission to write new data to the HealthKit store.

A participant came up with the idea to provide motivational messages in the warnings, meaning if the patient were to receive a warning, it would tell them a fact or a motivational message to grab their attention and motivate them to remember to take their medicine. Having bespoke, informative and motivational messages is an ingenious way of playing mind tricks on patients, almost guilt-tripping them into remembering to take their medicine. This proposition will be described in Future Work section: 6.2.7: Motivational Messages.

Another participant wanted the warnings to be more profound or easier to notice than a regular notification. The User Notification framework only allows the application to define the sound of its notifications. The MedRem app is set to **defaultCritical**, which is one of the most recognisable notification sounds available. However, if the patient has their iPhone in silent mode, the notification will not be permitted to make any sounds. Unfortunately, there is no way to bypass this. A solution could be to recommend the patient only to allow notifications from a handful of applications, including MedRem, such that the patient is more alerted on all notifications rather than none.

Having warnings dispatched only at a specific time came up during the

interviews. Of course, this is both a viable and doable solution, embracing the fact that patients would probably know the nature of the notification before opening it, considering the time it is dispatched. In the current version of MedRem, applications are not allowed to be dispatch earlier than 13:00 because Apple HealthKit uses some hours before it provides a resting-HR value close to its daily average. The resting-HR values always begin as higher values and are later updated and calibrated correctly. This solution makes it so that users are likely to be warned by the application not so long after 13:00 if they have a higher resting-HR that day. Thus, there might not be a need to implement this functionality because MedRem is already likely to dispatch warnings in the same time span.

As mentioned in section 3.4.3, the design of the application is inspired by Apples Health app. The decision was made to give the MedRem app more of a familiar feeling as a health application and to use less time to figure out a bespoke user interface. In the interviews, the participants were very consistent in saying they did not pay much attention to the user interface, indicating that the user interface was simple and well understood. Often a good design is what people do not tend to notice. The feedback from the interviews pointed out that none of the participants had really thought about the interface, proving the interface works well for the application.

5.5.6 Conclusion

To summarize, the experiment was conducted on four participants by supplying them with the application through Testflight. Also, through TestFlight, the users were provided with a short description of what they should do, and they were instructed on how they should test the application verbally. The test was designed to evaluate the application from a user perspective. By exposing the application to a broader user base, the test would give greater insight into how it works in a real scenario.

In conclusion, the participants' answers in the interviews gave a lot of insight into how the MedRem app is used and gave valuable suggestions for improvements. Overall the participants thought the app worked as an MVP, and the functionality worked as expected. There were no big incidents or fatal bugs discovered during the testing, and the areas of improvements have been discussed above. Some of these improvements will be discussed for future work.

5.6 Summary of Results

Various experiments were conducted in this chapter. The experiments consisted of testing each module or batch of functionality separate in standalone applications in the same way the MedRem app was developed. Each module's experiment would then verify the integrity of the functionality it intended to contribute with for the MedRem app. The MedRem app was also tested through an experiment, essentially being an acceptance test before the user-testing. In the final experiment, the application was distributed to four users who thoroughly tested and used it for a few weeks and then gave feedback through a semi-structured interview. In this section, we will go through the main findings of the experiments.

Both experiment A and B, two rather simple experiments testing very straight forward functionality based on parts of the code that were difficult to figure out and get right. The problematic part of the development was using the associated frameworks correctly to get the right outcome. These experiments were important to verify that the code delivered the desired functionality. The experiments successfully retrieved the resting-HR from the HealthKit store and delivered local notifications through the User Notification framework. They then proved that the modular codes were ready to be re-used for the MedRem app.

Experiment C could be seen as testing the baby steps of the MedRem app. Since the functionality that was to be tested in the module was difficult to test, it used functionality from the previous experiments to verify its own functionality. The experiment gave a lot of insight into how the MedRem app would have to be tested further down the line, switching between testing on simulators and real-life devices. This experiment was the most important experiment conducted during development, as the results could have changed the outcome of the MedRem app in terms of background functionality. Thankfully the developed functionality was satisfactory and was re-used in the MedRem app.

Experiment D worked as an acceptance test, checking all functionality and discovering all possible bugs before the application was considered ready for external user-testing. Minor bugs were discovered during this experiment which was later corrected before the app was considered ready for external testing. The experiment relied heavily on insight gained from Experiment C, which allowed the application to be tested in simulators and in real life.

The final experiment E: Performing User-Tests, was conducted over 3 months with four participants. A preliminary testing phase with two participants was conducted to gain quick feedback and add important functionality, which would have otherwise been difficult to test for the users. An example of this was the actionable notifications being difficult to interact with unless the user knew specifically to swipe on the notification. The main phase of the testing was conducted on all four participants for a duration between two and ten weeks, ending in an interview where the developer could question them regarding usability, functionality, user experience and the concept. The interviews resulted in suggestions on additional functionality that could have been developed for the application. Still, as an MVP, the application was working as intended, where all of the participants would have recommended it to someone suffering from heart disease.

Chapter 6

Conclusion

In this thesis, we have addressed an ongoing issue with heart failure patients not adhering to their medical therapy. This thesis's motivation is to create an application that would supplement the patients' medical therapy. The application should remind the patient to take their medicine when forgotten, based on their resting heart rate, which the application monitors. To approach the issue on hand, we came up with three functional requirements that would set the application's basic functionality. In addition to these requirements, the thesis's goal was to develop an MVP of the application, which would serve its purpose to see if it is possible to create an application that can remind patients to take their medicine based on their resting heart rate.

Thorough research was concluded to determine how to develop iOS and Apple Watch applications, structure the application and what frameworks to use. During the development phase, it was decided to split the functional requirements into separate modules to develop the functionality in a sandbox environment, meaning no other part of the application would interfere with the functionality in focus. After successfully developing the three modules, they were combined and configured with a user interface to become the MedRem application.

Evaluation of MedRem was achieved through five experiments; (A) Reading Heart Rates, (B) Dispatching Notifications, (C) Running Background-Tasks, (D) The MedRem App, and (E) Performing User-Tests. Experiment A, B, and C were experiments conducted to verify the functional requirements separately. At the same time, D tested all the functional requirements focusing on the application working as a whole. Experiment E was conducted on four participants for two to ten weeks and was the most crucial experiment because of the feedback received during its interviews.

The results from the experiments can conclude that the MedRem application developed in this thesis does indeed work as intended. Considering the many obstacles and limitations discovered during the research and development of the MedRem app, its outcome is better than expected. The background functionality almost sat a significant stop for the whole application. The feedback received from the user testing, in particular, can verify that the application shows promise for its usability and how patients with actual heart conditions could adapt it.

As a mHealth application aimed at increasing medical adherence for heart

disease patients, the application incorporates one behaviour changing feature of interactively warning patients through notifications. Considering the study mentioned in section 1.1.4 by Donevant et al. (2018), the applications with the most percentage of statistically significant outcomes were the ones that combined both passive and interactive features. As the MedRem application has not been tested on actual patients, we can only assume it will have a similar effect as other studies using the same type of interactive features. The upside of MedRem is that it uses real-time, real-life resting-HR data, which can be used to further enhance the user experience by creating bespoke interactions using the patients' health data. Following the trend of also implementing passive features in MedRem can provide a higher potential of success, particularly for making the app change behaviours and increase medication adherence.

To further guide any research related to this thesis, we have laid the groundwork explaining how the application is structured and developed from a design and architectural standpoint. We have discussed multiple solutions to each functional requirement, emphasizing the background functionality as it is the most challenging requirement to meet. The results from the experiments verify the goal and functional requirements of the thesis and bring valuable insight into further improvements and functionalities for the application and concept. For the final part of the thesis, we will dive into future work by explaining different findings and proposing conceptual solutions to them, respectively.

6.0.1 SPARK Application

In Appendix C: Spark Application, we submitted an application for the potential of future funding to continue to develop and research the MVP created in this thesis. Some of the comments/concerns/questions raised in the feedback were introduced because at the time of application, the app was still in the early stages of development. Some of these comments can now be addressed by the results obtained from the experiments.

The feedback states that HR measurements are noisy, and thus a clear and robust strategy is needed to avoid false positives. For the MVP developed in this thesis, the app uses the patient's resting HR value determined by the HealthKit framework on the Apple Watch. The resting-HR has shown to be relatively stable with little variance as long as the patient/user has worn the Apple Watch for the same amount of time throughout the day. By providing a dynamic boundary, the app can find a threshold at which the resting-HR should not trigger false positives. This is considering that most have a 5 BPM resting-HR variation, where some deviations are not problematic, but demonstrates that long-term measurement combined with accelerometers to decide resting-HR is in effect quite a powerful technique to personalize the feedback, and likely much better than rare in-office measurements during a doctor visit.

In addition, the feedback stated that there is very limited proof of the technology. The applications developed in this thesis should be enough proof that it is possible to create a valid application, which is usable and covers an unmet need. Though the MedRem app is still limited and needs further development to be relevant for actual patients, emphasising how the patients' resting-HR differs if they do not take their medicine in 24 hours. This is further discussed in Section 6.2: Future Work.

For the last point regarding data-collection, we have in this thesis discussed

how to retrieve health-related data from the HealthKit store through various queries. For the current application, privacy and GDPR are handled by Apple, as the health data is only used and displayed by the app but not distributed. If the data is to be collected in the future in a health information system or a database, the privacy aspects would need to be re-visited.

6.1 The Perfect Scenario

This section is about what scenario the application developed in this thesis would be best used and adapted. Due to the many limitations discovered during the work of this thesis, the way the application is used should be limited by certain factors to keep it working as intended. Also, because people are different and may adapt technology and habits in particular ways, it is essential to define a recommended use case for the application. Therefore comes the following recommendations:

6.1.1 Nightstand Charging

The most basic requirement for the application to function is that both the Apple Watch and iPhone are charged. It is recommended to have the chargers on the nightstand so that the user can charge while sleeping every night, which will result in the users always having enough power during the day.

The second reason for this is to make the user have a routine, where the first thing they do in the morning is to put on the Apple Watch, and the last thing they do is put it to charging. Because the resting-HR is updated throughout the day and the initial value is often much higher than the last, it is important to be consistent when the user wears the Apple Watch. Being consistent allows the data being compared to a boundary to be as accurate as possible, minimizing the chance of giving false alerts. This recommendation should be simple to adapt as most people are used to wearing watches, and by charging on the nightstand, the Apple Watch will be able to last the entire duration of the day.

6.1.2 Boundary Value Set By Healthcare Provider

To set the best calibrated boundary value for the user, it is recommended that the user has used an Apple Watch device regularly for at least the last 7-14 days while consistently either on or off their beta-blocker medication for that duration. By wearing the Apple Watch device consistently for 7-14 days, it is easier to determine what boundary value the user should have, which determines when they should be alerted.

If the user has just been prescribed beta-blockers but has been wearing an Apple Watch device consistently, the user, with the collaboration of their healthcare provider, can set a boundary between 4-7 BPM lower than their 7-14 day average.

If the user has been on a beta-blocker medication for the period while wearing an Apple Watch device consistently, the user, with the collaboration of their healthcare provider, can set a boundary between 4-7 BPM higher than their 7-14 days average.

The boundary will then be somewhere between the user's HR value when on and off the medicine, making sure the application will trigger an alert when the user has forgotten their medicine. When setting a calibrated boundary value with a healthcare provider, it can be wise to decrease the dynamic boundary gap value to a small decimal number or turn it off. This is a precaution for the boundary not to be dynamically over-compensated and end up outside the patient's HR scope, resulting in no alerts being triggered even when the user is on their medicine. Alternatively the app my have a toggle for turning off the dynamic boundary after a set amount of changes.

If the user sets the boundary value up without the help of a healthcare provider, it can be recommended to set it to the average resting-HR value while having a higher dynamic boundary gap. Every time the user receives a false alert, the boundary value will adjust until the user no longer receives a false alert. If the boundary value is 10-12 BPM above than their 7-14 day average, the value is outside of the scope and should be reset to be re-calibrated.

6.1.3 Apple Watch Set-Up

The Apple Watch comes without special configuration, loaded with multiple applications and features trying to interact with the users through notifications. Because the application proposed in this thesis is not meant to be a distraction for the user in their daily life, the change to an Apple Watch can be sudden for someone who is not used to wearing a smart device. Various applications on the Apple Watch dispatch multiple notifications during the day, which can from time to time be very distracting. For the perfect scenario, it is recommended to disable all types of notifications that the user is not interested in. Thus the Apple Watch will not be a distraction to wear and the MedRem warnings will be more noticeable.

The second reason it is recommended to make the Apple Watch less of a distraction is that the Apple Watch functions as an HR and motion sensor, providing resting-HR for the proposed application. The Apple Watch should not be an obstacle for the user as the main purpose is to warn the user when needed and not be a distraction otherwise.

6.1.4 iPhone Set-Up

It can be recommended to limit the number of applications allowed to dispatch notifications to the user on the iPhone. The iPhone is the device that receives notifications from the MedRem application. Having too many notifications appear constantly can make it difficult to notice the MedRem warning notification if buried between other notifications. This limitation should be considered because the MedRem notifications need to grab the users attention.

In addition to limiting notifications, for the MedRem app to function properly, the user is required to have Apple's Reminder app installed. This app is installed on all iPhone devices by default, but it is also one of the default applications that the user can delete. Therefore it is important that the users have not deleted the Reminder app, such that the MedRem app can schedule reminders. These setup requirements imply that the device (app + watch + phone) ideally should be set up together with a healthcare provider knowledgeable of these aspects.

6.1.5 Keeping the Application in the Background

Because of a previously mentioned limitation in section 2.2.5, it is important for the user to be aware that the application must be kept in the background for it to work. Because the system may terminate the application, or if an update terminates it or requires new permissions, the user will every now and then be required to reopen the application to reauthorize permissions or run it in the background. One can imagine that in a commercial setting, a reminder to restart the app after a software update pushed by Apple will be sent to every user from a central location.

6.1.6 Not Using Other Smart-Watches

Another previously mentioned limitation in section 2.2.5 is that multiple devices can access the HealthKit store. If the user has multiple smart-watches or, for example, has a sports watch used for running, golfing, hiking etc., the other watch is also able to write resting heart rate values to the HealthKit store. The result of having multiple data in the store is that the value read by the application will be the average of those. For example, if the user has a Garmin golf watch, which the user uses from time to time while playing golf, the resting heart rate value will be altered every time the user synchronizes the Garmin watch with their iPhone, as most of the Garmin watches do not synchronize continuously. Also, if the user only wears the other device while performing sports activities, the value is bound to be much higher than it is supposed to because it has not had a chance to measure a proper resting-HR value. Only devices with accelerometers which are able to determine resting-HR will affect the application.

In other words, the use of another device in addition to wearing the Apple Watch daily will confuse the proposed application in two particular ways:

- Extra resting-HR values will distort the Apple Watches value at arbitrary days where the user synchronizes other values. This will result in highly volatile day-to-day readings, making it difficult to determine a proper average to base the alert boundary on.

- High resting-HR values from sports activities will "sabotage" specific days resting-HR values and possibly trigger the alert because of high values.

Therefore it is recommended when using the proposed app not to use any other smart-watch. If the user has to use another watch that cannot be subsidised with an Apple Watch application, the user must make sure the smart-watch does not have access to write resting-HR values to the HealthKit store.

6.2 Future work

Throughout this thesis, there have been several observations to areas of improvement that can be made to the MedRem application. Even though the application fulfils the set requirements, there are potential possibilities for further developing this prototype into a finished product. A further developed prototype could be used to perform a clinical study, which will be useful for

commercializing the application in the future. The sections below are propositions of future work which can aid in the continued development of this concept.

6.2.1 Beta-blocker Research

The next step to further improve the MedRem application would be to develop it into a version aimed at actual patients. This will require further research regarding the beta-blocker medication on how it affects the patients resting-HR. The study should also find out specific differences to commonly prescribed beta-blockers in the form of dosages, frequency of dosages and how long the effect of a dosage lasts. This information will be valuable in the further configuration of a threshold/boundary value or algorithm which can accommodate differences in the beta-blockers and discover abnormalities.

6.2.2 Reminder to Wear Apple Watch

During user testing, a proposition was to have the possibility to be reminded by the phone through a notification if the user was not wearing their Apple Watch. Also, because the Apple Watch requires a PIN-code to be unlocked when initially put on, the users would, from time to time, put it on but not unlock it for several hours resulting in it not having access to write data to the HealthKit store. If no data is written to the HealthKit store, the application cannot detect changes in resting-HR and will be useless.

Unfortunately, Apple's wrist detection API is not available to the public, making it a bit more difficult than just requesting the API to get an answer on whether the Apple Watch is currently being worn or not. Because the iPhone alone can determine steps or movement, a proposed way to implement this feature would be to check after a given point in time if the user has moved but not registered any HR values. If the user has current HR values in the HealthKit store, they are wearing the watch properly and have not gotten a resting-HR value yet. However, if there are no current HR values from the watch, but there are movement data on the phone, it can be interpreted as the user is out and about but not wearing their watch or wearing it properly. This can be achieved by registering a background observer query for steps, and every time it fires to check if:

- The time of the day is reasonable to assume the user is awake.
- There is no resting-HR value stored for the day.
- The amount of steps has been updated/increased in the last hour.
- There is no current HR value stored for the last hour.

If these checks are fulfilled, the application should notify the user to remind them to put on or wear their Apple Watch properly.

Considering geo-location data may also aid in assessing whether the user is likely to be home or at work or another location where the user spends significant amount of time and is likely to have the watch available, but not on. This may also be timed to, e.g. 30 min after initiation of movement (as detected by the phone) following night-time to ensure that the reminder is likely to be sent to the user before the user leaves his/her place of residence.

6.2.3 CareKit & ResearchKit

To further develop this prototype into a fully commercial product for the health sector, I, as the prototype developer, recommend implementing the app using the CareKit and ResearchKit frameworks. CareKit is an open-source framework developed by Apple to help to develop health-related apps. This framework makes it simple to create custom health apps for specific needs of treatment, progress, medicines, connect users with their care providers, etc. ResearchKit is another Apple framework used to perform medical studies, simplifying the distribution of applications and surveys to clinical subjects.

If the MedRem application was to be integrated with CareKit, it could simplify connecting the patients with their care providers. Creating one application for the user, which is to monitor, remind and alert them while integrating functionality for the user to send their data directly to their care provider. The care provider would then have the data needed to give their patients a proper assessment without being there physically, also being aware of how often the patients receive alerts.

Integrating ResearchKit into the application or creating a sibling application with ResearchKit would give a potential benefit with regards to performing clinical studies, surveys, retrieving data and receiving proper feedback on the application. (Apple Inc. 2021c)

Looking back to Section 1.1.4: Wearable Heart Rate Monitors, we mentioned a study that compared different features of mHealth applications aimed at changing a patients behaviour towards medical adherence (Donevant et al. 2018). Even though the study emphasized the differences between passive and interactive features, the consensus of the article would tell that the applications with the highest rate of significant outcomes were the ones who used a good combination of both types of features. One example in the study was an application that served as a mobile diary where the patients could log when they took their medicine. This application also included SMS-reminders, though the reminders were generic and not personalized to the patients real life data.

The CareKit framework provides many possibilities for various features to be seamlessly implemented in the MedRem app. Seeing the outcomes of Donevant et al.'s study, it can be recommended to further implement both passive and interactive features by combining MedRem with CareKit functionality. The combination would work as a specialized patient follow-up application that can interact with the patient using customized and bespoke notifications regarding their own real-life data while also providing other features such as an electronic diary and uploading data to their healthcare provider.

6.2.4 Privacy and Security

If the application is to be developed and perhaps integrated with various healthcare systems, several factors must be accounted for regarding security. For the MedRem application, the app only uses the users' data that they willingly store on the device. If a future version of the app were to relay the data or information to different healthcare systems, it must comply with local data protection regulations. These regulations differ between countries, meaning the application must be developed for a set country specifically. For example, all Norwegian patient data must be stored on servers in Norway and not in other countries. If

the app were to be developed for several countries, the app must be developed differently security-wise to comply with the various regulations caused by the geographical differences. Security is a costly process for medical research as it is expensive to stay compliant with different laws.

6.2.5 Custom Watch Face or Complication

As mentioned in section 3.2.1, a possible functionality improvement was to implement the use of complications. However, if we were to take it a step further, we would create our own bespoke watch face for the users, making the application communicate with the user more clearly through the Apple Watch instead of the iPhone. This can be achieved by using Apple's Watch Connectivity framework, which is a framework allowing applications on iPhone and Apple Watch to communicate with each other (Apple Inc. 2021p). This framework would allow the MedRem application to update a complication on the Apple Watch.

Considering a scenario where the user wears the Apple Watch primarily for the MedRem application, a bespoke watch face/complication would be more useful for communication. Besides, it would declutter the Apple Watch interface so that it is not excessive to the user. Therefore, using the Watch Connectivity framework, the application can update the watch face/complication to display certain data next to time. For example, having the Apple Watch display a normal watch when the data is normal and giving it a look which grabs the users attention if the data is above the boundary.

6.2.6 Onboarding

Because MedRem is a specialized application with a particular purpose, it is important that it is used and set up properly. The patient should be going through the onboarding process with a healthcare provider, allowing the patient to understand how the application works and making the healthcare provider decide on the different variables used in the application. An onboarding process is a great way to help the patient and healthcare provider discover the application and set it up correctly the first time the application is launched. This will happen simultaneously as they are prompted with the different authorization windows and alerts, making it a natural part of the first time set-up.

The onboarding screens can be a series of navigation views, having each view link to the next, or be a flat list of pages with page control, allowing the patient or healthcare provider to swipe between the pages. An example of the pages can be the following:

1. **Greeting page**
   A page showing a greeting that is simultaneous to the various authorization pop-ups. This page should also tell the patient that they should be in the presence of a healthcare provider when setting up the application and not proceed without one.

2. **Information page**
   A page briefly explaining who the app is designed for and what it does.

3. **Set-up page**
   A page allowing the healthcare provider to plot in the patients' bound-

ary threshold and configuration for the app such that the application is finished set-up before the patient enters the actual application.

4. **Disclaimer page**
   A page containing a simple disclaimer explaining that there are limitations to the application and it does not guarantee correct warnings 100% of the time.

After the patient and care provider has gone through these pages, they should be directed to the main application. The onboarding process should not be displayed again unless the user somehow manages to delete the applications stored data. This can be implemented by storing a boolean variable in the UserDefaults key store, telling if the application has gone through the onboarding process before or not. Every time on launch, the application will display the onboarding navigation views if the variable is not set.

Another addition to the onboarding can be an update-onboarding. It is similar to the first onboarding, but only a simple page view tells the patient what is new with the application after an update. This can be implemented by storing the last version the user had an update-onboarding page shown, in the UserDefaults repository. If the application suddenly has had an update, the correct update-onboarding page should be displayed. The variable should be updated when the patient dismisses the update-onboarding page.

6.2.7 UI Improvements

There have been discovered multiple areas of improvement regarding the user interface of the MedRem app. This section will explain and propose solutions or improvements to new and existing components making the user interface.

Logging

A partially implemented feature in the code is the ability to log when warnings have been dispatched. It is implemented to the extent where it saves the warning date objects to the UserDefaults key store under the key *warningDates*, but there are no UI elements that use the data. To take full advantage of this data, a page or component in the UI of the MedRem app should be made to display the different occurrences of the warnings. The ability to see a more detailed history of warnings can help care providers follow up on their patients.

Graphs

Currently, the UI only shows the last 7 days of resting-HR values. An improvement to this would be to create a horizontally scrollable graph that allows the user to see their data way past the seven days it currently displays. The statistical collection query used can easily be modified to fetch all stored resting-HR values, making the implementation of this feature only dependent on a View component which can display the values in a user friendly and intuitive way. It is secure to fetch and graph the resting-HR data in the app because we can directly plot the data from the query and do not need to permanently store it.

Motivational Messages

The warning notifications dispatched by the application are very generic. Therefore it was proposed to have the notification shuffle between motivating messages to engage the user. For example, whenever a notification is dispatched, it can say *Adhering to your medical therapy will prolong your life! Your resting heart rate is above your set threshold, have you remembered your medicine today?*. Different messages can attract the user's attention more effectively and motivate the user to take their medicine so that they are less likely to forget.

6.2.8 Family alert

The age group for the MedRem app is in the higher age span of adults. Therefore in most cases, the users/patients have a caring family who would like to have an insight into whether their parent is adhering to their medical therapy. It could be beneficial to implement functionality for alerting close family or household members whenever the app thinks the users have not taken their medicines. This way, not only the application and a care provider can follow up with the patient, but also the family close to the patient. On the other hand, this feature would need extensive security measures and a thorough disclaimer considering the increased risk of data leaks and confidentiality with regards to the patient. For now, this proposed feature is only conceptual, as it would require more research to determine the viability of it.

Bibliography

Apple Inc. (Feb. 2019). *About the privacy and security of your health records*. URL: https://support.apple.com/en-us/HT209519 (visited on 10/20/2020).

— (2020). *About the HealthKit Framework*. URL: https://developer.apple.com/documentation/healthkit/about_the_healthkit_framework (visited on 09/29/2020).

— (2021a). *Asking Permission to Use Notifications*. URL: https://developer.apple.com/documentation/usernotifications/asking_permission_to_use_notifications (visited on 01/12/2021).

— (2021b). *Authorizing Access to Health Data*. URL: https://developer.apple.com/documentation/healthkit/authorizing_access_to_health_data (visited on 01/12/2021).

— (2021c). *CareKit*. URL: https://www.researchandcare.org/carekit/ (visited on 02/04/2021).

— (2021d). *Complications*. URL: https://developer.apple.com/design/human-interface-guidelines/watchos/overview/complications/ (visited on 01/11/2021).

— (2021e). *Creating Events and Reminders*. URL: https://developer.apple.com/documentation/eventkit/creating_events_and_reminders (visited on 01/22/2021).

— (2021f). *Dark Mode*. URL: https://developer.apple.com/design/human-interface-guidelines/ios/visual-design/dark-mode (visited on 02/23/2021).

— (2021g). *EventKit*. URL: https://developer.apple.com/documentation/eventkit (visited on 01/22/2021).

— (2021h). *Executing Observer Queries*. URL: https://developer.apple.com/documentation/healthkit/hkobserverquery/executing_observer_queries (visited on 01/12/2021).

— (2021i). *Human Interface Guidelines*. URL: https://developer.apple.com/design/human-interface-guidelines/ (visited on 02/23/2021).

— (n.d.). *Pushing Background Updates to Your App*. URL: https://developer.apple.com/documentation/usernotifications/setting_up_a_remote_notification_server/pushing_background_updates_to_your_app.

— (2021j). *Reading Data from HealthKit*. URL: https://developer.apple.com/documentation/healthkit/reading_data_from_healthkit (visited on 01/11/2021).

— (2021k). *Scheduling a Notification Locally from Your App*. URL: https://developer.apple.com/documentation/usernotifications/scheduling_a_notification_locally_from_your_app (visited on 01/13/2021).

Apple Inc. (2021l). *Swift*. URL: https://developer.apple.com/swift/ (visited on 02/04/2021).
— (2021m). *SwiftUI*. URL: https://developer.apple.com/xcode/swiftui/ (visited on 02/05/2021).
— (2021n). *TestFlight*. URL: https://developer.apple.com/testflight/ (visited on 02/08/2021).
— (2021o). *Updating Your App with Background App Refresh*. URL: https://developer.apple.com/documentation/uikit/app_and_environment/scenes/preparing_your_ui_to_run_in_the_background/updating_your_app_with_background_app_refresh (visited on 01/11/2021).
— (2021p). *Watch Connectivity*. URL: https://developer.apple.com/documentation/watchconnectivity (visited on 02/04/2021).
— (2021q). *WatchKit*. URL: https://developer.apple.com/documentation/watchkit (visited on 04/05/2021).
— (2021r). *Xcode*. URL: https://developer.apple.com/xcode/ (visited on 02/04/2021).
Corletto, Anna et al. (2018). "Beta blockers and chronic heart failure patients: prognostic impact of a dose targeted beta blocker therapy vs. heart rate targeted strategy". In: *Clinical Research in Cardiology* 107.11, pp. 1040–1049. DOI: 10.1007/s00392-018-1277-4.
Curry, David (Mar. 2021). *Apple Statistics (2021)*. URL: https://www.businessofapps.com/data/apple-statistics/ (visited on 04/22/2021).
Donevant, Sara Belle et al. (2018). "Exploring app features with outcomes in mHealth studies involving chronic respiratory diseases, diabetes, and hypertension: a targeted exploration of the literature". In: *Journal of the American Medical Informatics Association* 25.10, pp. 1407–1418. DOI: 10.1093/jamia/ocy104.
Dunlay, Shannon M, Veronique Lee Roger, and Margaret May Redfield (Oct. 2017). *Epidemiology of heart failure with preserved ejection fraction*. URL: https://mayoclinic.pure.elsevier.com/en/publications/epidemiology-of-heart-failure-with-preserved-ejection-fraction.
Hein, Andreas et al. (2019). "Digital platform ecosystems". In: *Electronic Markets* 30.1, pp. 87–98. DOI: 10.1007/s12525-019-00377-4.
Kharpal, Arjun (Feb. 2020). *Apple Watch outsold the entire Swiss watch industry in 2019*. URL: https://www.cnbc.com/2020/02/06/apple-watch-outsold-the-entire-swiss-watch-industry-in-2019.html#.
Knott, Matthew (2014). *Beginning Xcode*. Apress.
Kocianova, Eva et al. (Dec. 2017). "Heart rate is a useful marker of adherence to beta-blocker treatment in hypertension". In: *Blood Pressure* 26.5, pp. 311–318. DOI: 10.1080/08037051.2017.1346458.
Kotecha, Dipak et al. (June 2017). *Heart Rate and Rhythm and the Benefit of Beta-Blockers in Patients With Heart Failure*. URL: https://www.ncbi.nlm.nih.gov/pubmed/28467883.
Ma, Yuxin and Stephen W. Harmon (2009). "A Case Study of Design-Based Research for Creating a Vision Prototype of a Technology-Based Innovative Learning Environment". English. In: *Journal of Interactive Learning Research* 20.1. Copyright - Copyright Association for the Advancement of Computing in Education 2009; Document feature - Diagrams; ; Last updated - 2010-06-08, pp. 75–93. URL: https://www-proquest-com.ezproxy.

uio.no/scholarly-journals/case-study-design-based-research-creating-vision/docview/211211726/se-2?accountid=14699.

Osterberg, Lars and Terrence Blaschke (2005). "Adherence to Medication". In: *New England Journal of Medicine* 353.5. PMID: 16079372, pp. 487–497. DOI: 10.1056/NEJMra050100. eprint: https://doi.org/10.1056/NEJMra050100. URL: https://doi.org/10.1056/NEJMra050100.

Ouwerkerk, W et al. (June 2017). *Determinants and clinical outcome of up-titration of ACE-inhibitors and beta-blockers in patients with heart failure: a prospective European study.* URL: https://www.ncbi.nlm.nih.gov/pubmed/28329163.

Perez, Marco V. et al. (2019). "Large-Scale Assessment of a Smartwatch to Identify Atrial Fibrillation". In: *New England Journal of Medicine* 381.20, pp. 1909–1917. DOI: 10.1056/nejmoa1901183.

Reeves, Thomas C. (2000). "Enhancing the Worth of Instructional Technology Research through 'Design Experiments' and Other Development Research Strategies". In: *Annual Meeting of the American Educational Research Association*. URL: http://it.coe.uga.edu/~treeves/AERA2000Reeves.pdf.

Scarsella, Anothony et al. (Jan. 2021). *Smartphone Shipments Return to Positive Growth in the Fourth Quarter Driven by Record Performance by Apple, According to IDC.* URL: https://www.idc.com/getdoc.jsp?containerId=prUS47410621.

Shcherbina, Anna et al. (May 2017). *Accuracy in Wrist-Worn, Sensor-Based Measurements of Heart Rate and Energy Expenditure in a Diverse Cohort.* URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5491979/.

Wald, David S., Shahena Butt, and Jonathan P. Bestwick (2015). "One-way Versus Two-way Text Messaging on Improving Medication Adherence: Meta-analysis of Randomized Trials". In: *The American Journal of Medicine* 128.10. DOI: 10.1016/j.amjmed.2015.05.035.

Wallen, Matthew P. et al. (2016). "Accuracy of Heart Rate Watches: Implications for Weight Management". In: *Plos One* 11.5. DOI: 10.1371/journal.pone.0154420.

Wang, Robert et al. (Jan. 2017). "Accuracy of Wrist-Worn Heart Rate Monitors". In: *JAMA Cardiology* 2.1, p. 104. DOI: 10.1001/jamacardio.2016.3340.

Appendices

Appendix A

Source Codes

The complete source codes with XCode project files have been uploaded to different repositories on GitHub, including a brief README taken from the development chapter.

A.1 MedRem

`https://github.com/Jaanesen/Medicine-Reminder`

A.2 Resting Heart Rate App

`https://github.com/Jaanesen/Resting-Heart-Rate-App`

A.3 Notification App

`https://github.com/Jaanesen/Notification-App`

A.4 Background App

`https://github.com/Jaanesen/Background-App`

Appendix B

Interview Guide

Interview Guide - MedRem app

Leading questions

1. How old are you?
2. What is your profession/work?
3. Which iPhone version are you using?
4. For how many years have you used iPhone?
5. Have you before this user-test used an Apple Watch?

Main questions

1. For how long did you test the application?
2. At what time of day do you start and stop wearing the Apple Watch?

   From you wake up till you go to bed?
3. How did you test the application? How did you make the application give different results?

   Changing boundary value?, dynamic value, answering warnings differently?
4. What do you think of the application as a whole?
5. Is the application working as you would have expected?

   Is it something not working as you expected?

   Is there any functionality you think is missing?
6. Do you think the application was easy to use? Why/why not?
7. What do you think about the applications appearance?

   How important is an apps appearance for you? Why?

8. Have you had a tendency to stop wearing the Apple Watch for a larger amount of time during the day?

   Why? Annoying or uncomfortable?

   If so: Do you think it has had an impact on the results?

9. Have you received warnings when you are supposed to?

   Do you notice the warnings immediately, or after some time?

10. Have you experienced receiving warnings when you shouldn't have?

11. Have you asked to receive later reminders?

    Have this worked as expected?

Finishing questions

1. How usable do you think the application is? Does it cover a need?
2. If you were in the applications user-group and had forgotten to take your medicine, do you think the application would have helped reminding you? (Not in a test situation, but real life)
3. What do you think is best about the application?
4. Do you think there is something not working as good as it should have in the application? What do you think can be done to improve it?
5. Do you have any improvements you want to bring to the application?
6. How likely would you be to recommend the application to someone who is in the user-group, if this was not a prototype?

Appendix C

SPARK Application

To potentially receive funding for further development of the application in this thesis, an application was submitted to SPARK Norway, a two-year innovation programme. Unfortunately, the project was not selected. However, we received valuable feedback considering that the application was submitted during the app development and not after evaluation. Following is the written feedback received.

UiO : Life Science
University of Oslo

Jens Kaasbøll

Date: March 18th 2021

Your application for SPARK Norway

We refer to your application for SPARK Norway program Round 4, 2021.
As previously informed your project, WatchBeta-Blockers: Smart watch app for increasing beta-blocker adherence, was not selected in this round, but we want to give you a feedback from the evaluation that we hope will be valuable for you to develop your project further.

Feedback from the evaluation:

- The project is addressing an unmet need within health-related life sciences
- It is a strong team and an interesting project with a novel approach that may have potential

However, it is at a very early stage and the following comments/concerns/questions were raised in the evaluation process:

- Medication adherence is a major problem. Simple solutions like this could improve treatment benefit at low cost. Other smart phone-based adherence systems use sensors to detect pill taking motion/box opening or proximity to dispenser unit - not seen HR based approach elsewhere.
- Good idea, but unsure about ability of technology to detect actual medication use (beta blocker response is individual and dose-dependent
- Most medicines are slow release and have relatively long half-life - when is detection possible (time frame)?
- HF patients are slowly titrated - what is the baseline?
- Norwegian registry data shows that patients with high HR often use higher doses of betablockers - ie. not a direct HR/BB relationship)
- IP not really discussed. The idea is relatively simple to replicate unless dose detection algorithm is unique and with high accuracy
- Team is good but small - seems to need more medical analytics, signal analysis, and software development resources. Detection of med use needs more proof
- The project has a useful and ambitious goal, at the same time a lot of aspects are not considered in the project: HR measurements are usually quite noisy and, thus, a clear and robust strategy on how to avoid false positives is required. Moreover, the project aims to perform several clinical trials over a two-year period, there is a very limited proof of concept of the technology
- The feasibility of the project is questionable as it requires first and foremost a robust study on methods and hypothesis
- There might be problems in collecting data from Apple devices and thus the aspects of privacy and GDPR should be also studied in detail

LIVSVITENSKAP LIFE SCIENCE

UiO:Life Science
Postal addr.: PO Box 1078 Blindern, 0316 OSLO
Visiting addr.: Sogn Arena, Klaus Torgårds vei 3, first floor, 0372 Oslo

postmottak@lifescience.uio.no
www.uio.no/life-science
Org. no.: 971 035 854

We want to thank you again for your interest in SPARK Norway. There will be a new call from SPARK Norway announced in the fall. We welcome you to participate in our monthly educational meetings and hope that they will be useful for you in your future work. We announce all our activities on our web page www.uio.no/life-science/spark.

We wish you good luck in your continuous innovation endeavor!

Sincerely yours,

Morten Egeberg
Administrative leader UiO:Life Science
& leader of SPARK Norway