Creating a GDPR compliant app for research data collection

Dominik Koller

June 2022

Abstract

Collecting user data for research must be done transparently and with complete user control over their data, for practical, ethical and legal reasons. This project takes the EU General Data Protection Regulation (GDPR) as a guideline for creating a data collection app for research, describes possible ways of implementing features required by the GDPR in a user-friendly way, including a fully working app implementing these suggestions. The app collects iPhone and Apple Watch Health data as well as self-reported mental wellbeing data on a server with full transparency and user control. The process of data collection is demonstrated and fully implemented in an iPhone app and AWS server. The type of collected data is selected to facilitate follow-up work, in which I want to investigate correlations between self-reported mood data and automatically collected health data. The pipeline and data collected in this project is ready to be analysed, with a fully functioning and tested connection in python directly to the server API.

Contents

1 Motivation

The primary objective of this project is to create an application and data pipeline for collecting user data for research purposes.

Health-related data gathered from individuals is increasingly important in many areas of research [Jim+20]. Collecting such data comes with significant difficulties. Privacy of participants and transparency must be preserved (compare [Ost+17]); data collection, transmission and storage must be secured; data pipelines should be automated and scalable. In this project, I established design principles to meet these challenges. These principles focus on the user experience from the perspective of the individual whose data is collected, as well as the researcher who is to use the collected data. Legal requirements for data collection differ widely in different jurisdictions; however, the European GDPR is often seen as pioneering data protection law globally (eg [Dix18]) and is particularly restrictive for data collectors and protective of individual's rights on privacy and transparency [Gre18]. I will therefore use the GDPR as a guideline for designing an ethically responsible data collection application. I will especially focus on designing a user experience that does not only legally follow the requirements of the GDPR, but also follows the spirit of its design principles around transparency and privacy every step of the way. The data gathered is physical health data from a smartwatch and self-reported mental wellbeing data, thus highly sensitive data, highlighting the importance of the established design principles. It can be all collected on user's devices and therefore showcase a highly automated data pipeline. Using data collected by end-user devices such as smartphones and smartwatches is a relatively new and promising field of research in psychology [RD16]. While I believe we must be very cautious about generating and using sensitive personal data - as reflected in the focus on transparency and privacy in this project - I also believe there is potential to innovate on prevalent methodology in quantitative psychology research using potentially very large datasets of human physiology and behaviour (eg see [SAL20]). Predicting mental wellbeing is a step towards more comprehensive approaches of diagnosis and perhaps even help in treatment of mental health. With this project, I wish to provide a start in this direction, by building a data pipeline and establishing ethical design principles for data collection.

2 User data collection in academic research

Collecting sensitive user data for academic research is often valuable and necessary. Ethical, technological and legal challenges as well as user experience design must all be taken carefully into consideration.

3 Design Challenges

Designing the User Experience for data collection apps means taking responsibility of the relevant ethical and practical challenges *from the user's perspective.* We must design for privacy, transparency, and user control, all while making using the application as easy and enjoyable as possible. In order to avoid miscommunication and for ease of use, the user interface must be consistent, important information always shown prominently, and pages with information and controls must be easily reachable via clearly communicated buttons and navigation.

3.1 Designing for Privacy

Respecting users' privacy and data collection for research are by no means mutually exclusive. I suggest three principles to maximise users' privacy in this project:

- *Collect only the data you need.* Every piece of data collected must be actually needed for analysis. In the application for this project, where I ask which kind of data is predictive of mental wellbeing, this means collecting data strictly only in the timespan necessary for this analysis.

- *Anonymize data where possible.* Anonymization, however, is challenging and often reversible [Ola+]. In this project, I will not collect any directly identifying data, and will not attempt to actively anonymize any collected data.
- *Do not publish sensitive user data*, especially if not anonymized or de-anonymizable. When necessary, for example for journal submissions, data may be made available to reviewers only - for example, see the Nature editorial *'Finding a sensible approach to sensitive data'* [18]. In this project, no data will be made public.

3.2 Designing for Transparency

Transparency is the most important aspect of designing any digital services that collect data: users have a right to know what they are signing up for when using a product. At any point in the user experience, it is important to communicate clearly what data is being collected and why. Also, users must have access to data collected on them at all times. Designing for transparency also requires following well-established user interface rules: text must be clearly visible, readable fonts must be chosen, and navigation to the relevant information must be made clear and recognisable.

3.3 Designing for User Control

Users must have control over their data. Besides transparency, this includes giving users as much choice as possibly in *which data is collected*, and giving users the option to delete all their data. Deletion is crucial, also legally, as discussed in section (4). Designing for control equally requires following well-established user interface design rules: buttons for user actions must be clearly visible as such, user interface elements must be consistent across the application, all controls available to the user must also be clearly recognizable to the user. Companies often use 'deceptive design' patterns, designed to trick users into doing things, such as giving consent to data collection [Yae16]. Such practices have been partially banned, for example in California [Vin21]. In order to design for user control, such dark patterns have to be avoided. When designing an interface, the exact opposite must be the goal - to make sure that the user does exactly what she intends to do at any point when using the application.

4 Legal Requirements

In this section, I will outline the legal requirements for a data collection application when distributing to users, according to the GDPR[1]. I will not attempt to summarize the GDPR in any way; instead, I will point out the parts relevant

[1]Note that the EU GDPR, which I reference here, is an EU regulation and does no longer apply in the UK. In the UK, there is now the UK GDPR, which is in all aspects relevant to this project identical to the EU GDPR.

to research data collection and this project in particular. The legislation is not specifically aimed at research, but rather towards any actor collecting data in the EU, such as private companies, government agencies as well as researchers. Importantly, a University as a public body is required by the GDPR to have a Data Protection Officer [16f]. Researchers may seek advice from their Data Protection Officer, including which measures they are required to take in order to fulfil their legal requirements. Article 5 of the GDPR states personal data must be

> *processed lawfully, fairly and in a transparent manner in relation to the data subject ('lawfulness, fairness and transparency')* [16a]

Performing a 'task in the public interest' for public research or 'legitimate interests' for non-public authorities, charities and commercial organisations provides a lawful basis for research (compare [20]).

For this project, 'lawfully, fairly and in a transparent manner' is relevant in three ways: regarding the *right to be informed*, the obligation for *data minimization*, and the *right to be forgotten.* Recital (39) of the GDPR states

> *It should be transparent to natural persons that personal data concerning them are collected, used, consulted or otherwise processed and to what extent the personal data are or will be processed. The principle of transparency requires that any information and communication relating to the processing of those personal data be easily accessible and easy to understand, and that clear and plain language be used.* [16d]

The legal instructions here are quite clear: users must understand at each point which data is collected and why. This applies to any research project and must be followed for ethical and legal reasons. Regarding *data minimization*, the GDPR states that data must be

> *adequate, relevant and limited to what is necessary in relation to the purposes for which they are processed ('data minimisation')* [16c]

and

> *collected for specified, explicit and legitimate purposes and not further processed in a manner that is incompatible with those purposes; further processing for archiving purposes in the public interest, scientific or historical research purposes or statistical purposes shall, in accordance with Article 89(1), not be considered to be incompatible with the initial purposes ('purpose limitation').* [16b]

In the context of research data collection, and this project in particular, this means that we must *only collect the data necessary to answer the research question* and communicate this requirement clearly to users. The third point in the GDPR relevant to this project is the *right to be forgotten.* The GDPR states that

> *a data subject should have the right to have his or her personal data erased and no longer processed where [...] a data subject has withdrawn his or her consent or objects to the processing of personal data concerning him or her* [16e]

To put this into context, this 'right to be forgotten' has had a significant impact on the global tech industry; for example, this recital was central in a lawsuit of CNIL, the french data-protection regulator, against Google which CNIL won [Eco19]. Notably however, the 'right to be forgotten' does *not* apply to arguably any data collection in research. The GDPR further states that even if this right is exercised,

> *further retention of the personal data should be lawful where it is necessary, [...] on the grounds of public interest in the area of public health, for archiving purposes in the public interest, scientific or historical research purposes or statistical purposes.* [16e]

Specifically, this means that for scientific research purposes, the data controller (your organisation, the University, represented by the Data Protection Officer) typically does *not* have the obligation to delete personal data on user request. However, I believe this 'right to be forgotten' is still relevant to scientific research and this project in particular. It highlights that *if possible*, we should respect an individuals wish to delete their personal data. Even if this is in our case and many others not legally binding, I suggest researchers should follow in the spirit of the GDPR and *make deletion possible* whenever they can. For example, when designing user interfaces, there should be a clear and accessible way for users to request deletion; data pipelines should be built so that deletion is automated and as complete as possible; and every case where data will not be able to be deleted (such as when data is published, or after results depend on all data being available for reproducibility) should be clearly communicated to users beforehand. I will later suggest and implement an automation mechanism in the data pipeline for deleting data which, to my knowledge, is novel in data collection for scientific research; this is inspired (though as we have seen not required) by the 'right to be forgotten' in the GDPR.

5 Examples

In this section, I present other works relevant to this project, both regarding ethical data collection and mood prediction.

5.1 Success Examples

5.1.1 PPD ACT

PPD ACT is an *'an app-based genetic study of postpartum depression'* which sought to research post-partum depression (PPD) [Gui+18]. Researchers released an app in the US and Australian Apple App Stores to identify 7344 cases of PPD and collect 2946 samples of DNA biokit data. Participants were asked questions in order to identify PPD, and could then choose to be sent a DNA spit testing kit which they would in return send to a lab. Researchers went to great lengths in order to ensure transparency, consent and data security. Before storing any data, users were informed of which data was collected and why; the reading and understanding of this information was then checked by asking questions the participants had to answer correctly to proceed. Participants then 'finger-signed' an electronic consent document. Only then would the app collect any data from participants. These steps were first taken when identifying PPD, then repeated when participants were asked to order and submit the DNA sample. It was made clear at any point that participants could withdraw in the app or via email. When withdrawn, a participant's data including DNA data would be deleted from all repositories. Data security was ensured by designing a custom database and access interface; data was encrypted on participants' devices with the researchers' public key, sent to the server and stored encrypted, and researchers could access the data via the custom interface and their private key.

This study offers a great example of how to collect data from participants via automated pipelines while upholding ethical and legal standards regarding transparency, consent and data security. This study was conducted in cooperation with Apple.

5.1.2 Large-scale Wearable Data

In the paper *'Learning Generalizable Physiological Representations from Large-scale Wearable Data'*, researchers collect a large dataset of wrist accelerometer and wearable ECG data (over 280,000 hours of data) and propose a novel self-supervised and transfer learning technique to predict other variables associated with an individual's health, such as the maximal oxygen consumption (VO2max), one of the best indicators for cardiovascular fitness [Spa+20]. The method to inform participants and confirm their consent was done traditionally, that is in person and not via a distributed application. Participants were given specialized devices and not using their own peripherals. The study is relevant for this paper in two ways: firstly, how this very large dataset was handled when publishing the paper in a journal; secondly, the method of predicting seemingly unrelated health variables from accelerometer and ECG data. Journals usually ask researchers to publish data when anonymization is possible. In a talk I attended, author Cecilia Mascolo stated that for this particular paper, the journal initially asked for publication of the full dataset. However, the researchers insisted that the data could be de-anonymized and therefore should not be published. Instead, they set up a procedure for reviewers to access their data and other researchers to do so upon request. This is a great example of *researcher's integrity* where the customary protocol set out by a major scientific journal failed to protect participant's privacy. Methodologically, this paper presents a pioneering approach to working with large scale datasets of continuously measured health indicators. In a first step, a sequential model is trained to predict heart rate using accelerometer and past heart rate data. This data is relatively easy to collect in very large quantities. Researchers postulate that in order to predict a quantity as essential and interconnected with other health related variables as the heart rate, an algorithm must essentially learn to represent the *general physical health state* of an individual, at least to some degree. Thus, they postulate that the last layer in the sequential model should contain information about many *other, seemingly unrelated* health variables. This gives rise to the next step, transfer learning from the last layer in the sequential model to other variables, such as VO2max. Importantly, these are clinically significant variables that are otherwise *difficult to measure*, especially in large quantities. The transfer learning approach allows for learning the function from easy to collect data to difficult to collect data while not requiring large datasets of the latter. I suggest this is to be seen as an algorithm that learns a representation of the general physical health state of an individual, which is an excitingly new and extremely promising approach to health research. While the data I collected in the course of this project is not nearly enough to use these models, my data collection methods through their automation and scalability are suitable for collecting and working with similarly large datasets. In followup work, I would like to establish whether a similar transfer learning technique could be used to predict not only indicators of physical health as researchers did in this work, but also to predict indicators of mental health as I did in the present paper. This is significant, especially given the symmetry of *easy to collect physical health data* and *difficult to collect mental health data*.

5.1.3 Predicting Mental Illness Onset

In the study 'Predictive Modeling of Mental Illness Onset Using Wearable Devices and Medical Examination Data: Machine Learning Approach', researchers used sleep and other wearable data to predict mental illness onset [SSK22]. The data for this study was obtained from society-managed health insurance members in Japan, thus not specifically collected for this study. Data included large sets of wearable data from Fitbit devices, as well as insurance claims regarding mental illness. Research methods were not notably innovative, standard models for time series data were used. The relevant and novel parts of this study were its size (37,856 time series and 4,612 individuals, however only 24 of which exhibited mental illness onset) and the results. The large size of wearable data of this study is one instance of a change in research methodology in psychology, conducting quantitative research in areas where previously only qualitative research was viable. The results are also intriguing: while not highly accurate, the models showed moderate predictive power of mental illness onset. Importantly, wearable data showed higher predictive power than medical examination data in the same time spans. This is an important validation for the usage of wearable data in mental health prediction, diagnosis and eventually treatment.

5.2 Failure Example

The most prominent example of misused research data was the Facebook - Cambridge Analytica scandal [Mer18]. Researchers misled participants (and, in fact, non-participants) by violating all principles I established above: transparency was not given as many people were not informed that their data was being collected at all; people whose data was being collected did not have any control over which data was being collected nor were they given a possibility to opt out or have their data deleted; data was being collected under a false pretext of academic research when in reality it was used for political advertisement. These are serious legal and ethical shortcomings which must be avoided in any research setting (or in fact any setting at all).

6 Reasearch App 'Anima': Mental Wellbeing Prediction

6.1 Motivation

This app is intended to showcase the design guidelines discovered above. The full data pipeline is operational, tested and actively employed. All design principles are followed: users are informed on what data is collected and the reasons for its collection (designed for transparency); they can always see all their collected data and completely delete all data at any time (designed for user control). No directly identifying information is collected or stored and only necessary data is collected (designed for privacy), but also no active anonymization is performed; users are made aware of this fact. Measuring and quantifying variables is traditionally very difficult in mental health research. No consistent biomarkers exist for mental wellbeing in general, though recent work has made advances in that direction [Gar+20]. Diagnosis of mental illness is often biased, eg by race or gender [Gar21]. I suggest that we might be able to make diagnosis more reliable by using behavioural data such as sleep and activity data, as well as continuously measured physical health data such as heart rate and O2 saturation, and learning a function from these time series data to variables related to mental health. The example in section (5.1.3) suggests as much. It is therefore a valuable contribution to suggest a method for the relevant data collection, and to find whether I can predict my own mental wellbeing given data I collected on myself.

6.2 Research Goals

To demonstrate the full data pipeline from collection of user data to analysis to results of a research project, this app aims to predict a one-dimensional measure of mental wellbeing from data automatically collected by an iPhone and an Apple Watch. The app is built to fulfil all requirements in order to be published on the Apple Store (although I did not actually publish it). The main goal was to create a fully functional data collection application with a user interface that follows all established design principles; the secondary goal was to analyse this data in order to predict mental wellbeing from physical health and behaviour data.

6.3 Collected Data

The automatically collected data includes time-series data of Heart Rate, Exercise Time, Headphone Audio Exposure, Environment Audio Exposure, Stair Ascent Speed, Step Count, Walking Double Support, Walking Heart Rate, Walking Speed, Step Length, and total Walking Distance. Those categories of data were chosen as a subset of available data in the Apple HealthKit datastore; the choice to include these was made somewhat arbitrarily; I found all of these could potentially be interesting when related to mental wellbeing. This data is collected automatically by an Apple Watch and an Apple iPhone and accessed via the Apple HealthKit API. The app is run on an iPhone and accesses all data in the same manner, whether collected on the phone or on the watch.

Wellbeing data is user-reported and collected on a one-dimensional scale. It has been shown that one-dimensional measures of self-reported mental health is associated with multi-dimensional measures of mental health and health problems ([Ahm+14]), suggesting that predicting a one-dimensional measure of mental wellbeing is a valuable contribution and a starting point for more detailed analysis.

HealthKit data is collected in a restricted time span only. This time span ranges from one week prior to the first user-reported wellbeing measure up to the last such user-reported measure. This is to reduce the amount of data collected to the subset necessary for analysis.

Data is collected in combination with a unique anonymous user identifier. No user email address, name, location, or any directly identifying information is collected.

6.4 Deleting Data

In general, when users turn off collection of a specific type of data, all corresponding entries are deleted on the device and on the server. No entries are deleted from the Apple HealthKit storage, which is separate to this application. Deletion of all data concerning a particular user is also possible, following a more comprehensive approach. A clearly visible button in the user interface for deleting all data opens an information page that informs the user they are about to delete all their previously collected data. If they then choose to delete, three things happen:

- all datapoints with that specific user identifier are deleted, on the server and on the device
- a new user identifier is created for the device; the old one is deleted

- the old user identifier on the server gets added to a list of deleted user identifiers

This approach ensures that the user choice to delete their data is stored via their anonymous user id. Should there be any copy of the database in the pipeline for any reason - a backup, cache, redundancy copy - it is always easy to ensure that requested deletion is carried out throughout the whole data pipeline. I built this mechanism, but did not have to use it in this project due to the simple data pipeline and lack of copies or backups of my database. I have not seen other research projects use similar mechanisms; however, I suggest when building scalable automated data pipelines important user choices such as a request for deletion should be both carried out immediately and stored for later correction of potential mistakes (such as not deleting the corresponding entries in a copy).

If there was a specific date fixed after which data cannot be deleted for valid reasons, for example for publication of a paper, I suggest there should be a notice included in the application stating that after a certain date, user data cannot be deleted anymore in order to ensure reproducibility of results. This is not the case for the current application; users can always choose to delete any part or all of their data.

6.5 Inspecting Data

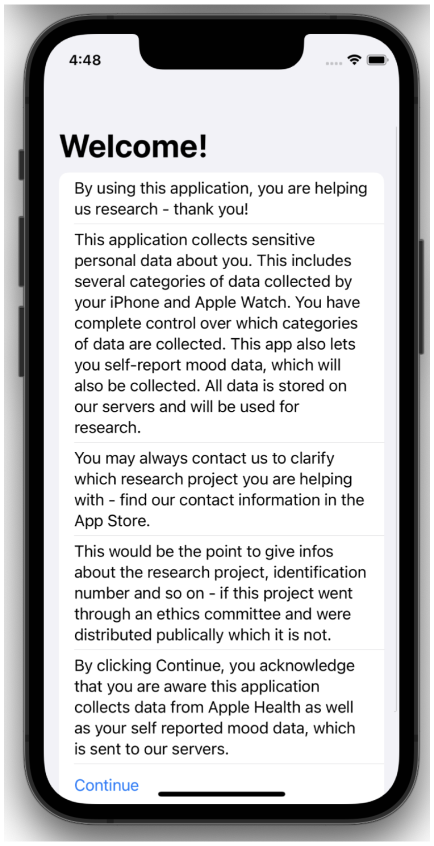
Users may inspect, change and delete any part of their self-reported mood data. Any part of any record may be edited: date and time, valence, and additionally a note (a data field I added just because I found it convenient to use as a journal when recording my own mood data).

6.6 User Experience

The User Experience for this app focuses on delivering the established design principles in an intuitive manner. At each point, users must understand at every point when using the app

- which data is collected
- why data is collected
- which control they have over the data (including deletion).

In detail, when opening the app for the first time, a message screen explains why this app collects data, what data it can collect, and how the data is used. See figure (1). Then, users are presented with three tabs. The default tab enables users to record their current mental wellbeing; when first opening this tab, a helpful message tells the user what she can do in this tab. See figures (2) and (3). In the recorded data tab, users can see, modify and delete all their recorded mental wellbeing data. See figures (4), (5), and (6). Recorded mental wellbeing includes the one-dimensional wellbeing measure, date and time



4:48

Welcome!

By using this application, you are helping us research - thank you!

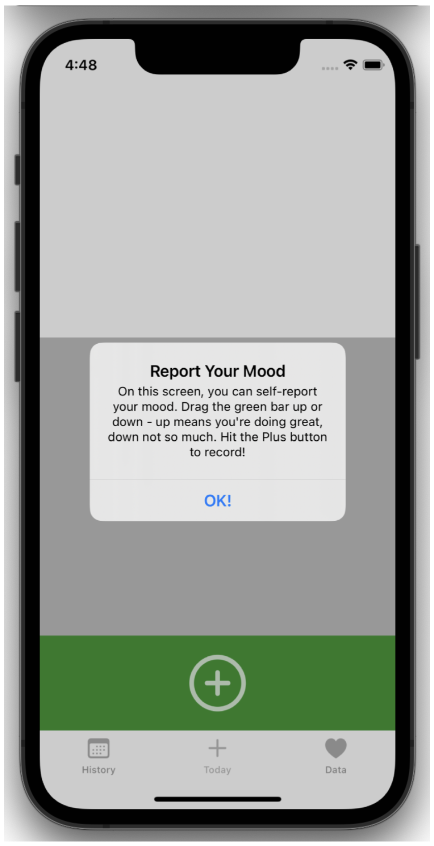
This application collects sensitive personal data about you. This includes several categories of data collected by your iPhone and Apple Watch. You have complete control over which categories of data are collected. This app also lets you self-report mood data, which will also be collected. All data is stored on our servers and will be used for research.

You may always contact us to clarify which research project you are helping with - find our contact information in the App Store.

This would be the point to give infos about the research project, identification number and so on - if this project went through an ethics committee and were distributed publically which it is not.

By clicking Continue, you acknowledge that you are aware this application collects data from Apple Health as well as your self reported mood data, which is sent to our servers.

Figure 1: The welcome screen shows information and asks for consent to collect data. Since this project does not have a site or reference number to refer to, part of this text remains a placeholder text.



Report Your Mood

On this screen, you can self-report your mood. Drag the green bar up or down - up means you're doing great, down not so much. Hit the Plus button to record!

Figure 2: Upon opening the mood recording tab for the first time, a message is shown explaining what the user can do here.

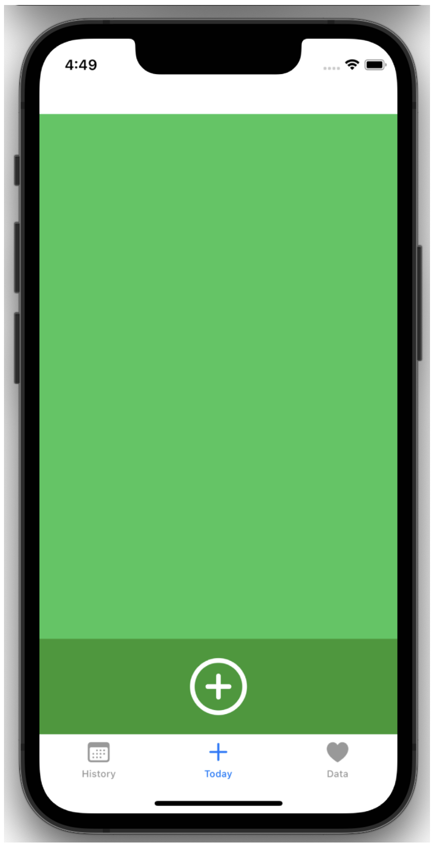
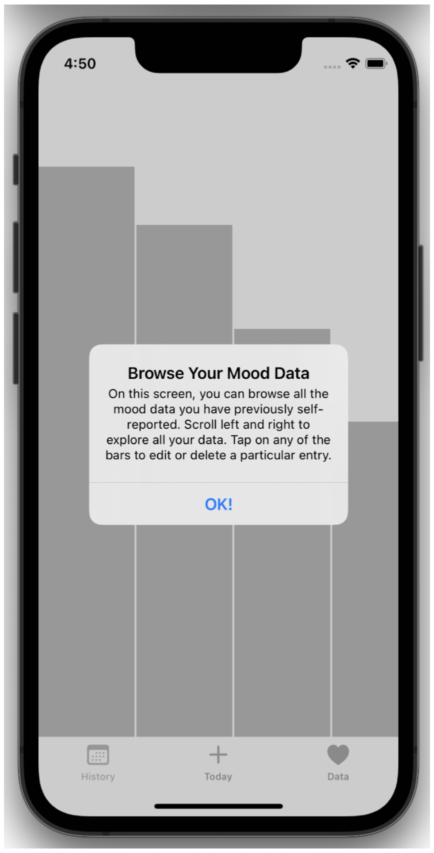


Figure 3: The user then lands on the page where she can self-report mood data.



Browse Your Mood Data

On this screen, you can browse all the mood data you have previously self-reported. Scroll left and right to explore all your data. Tap on any of the bars to edit or delete a particular entry.

OK!

Figure 4: Upon opening the mood data inspection tab for the first time, a message is shown explaining what the user can do here.

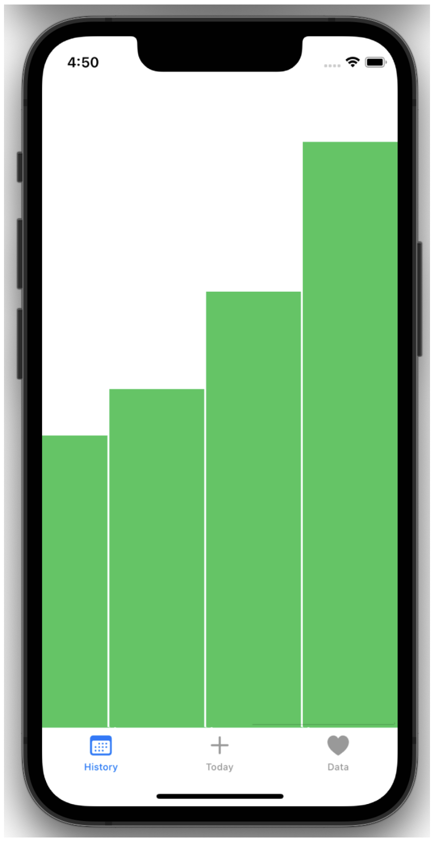
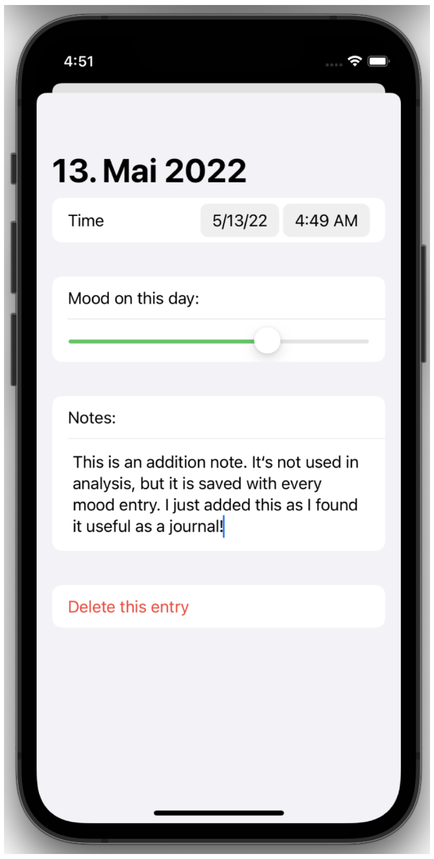


Figure 5: In the mood data inspection tab, the user can view all mood data and select individual records to edit or delete.

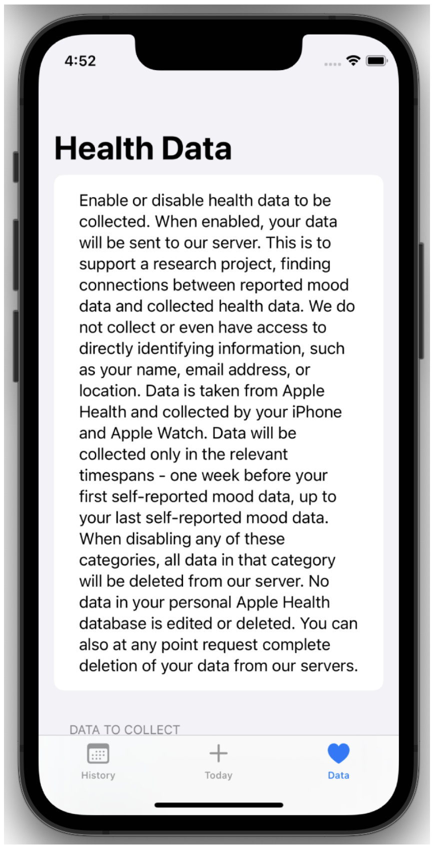


This is an addition note. It's not used in analysis, but it is saved with every mood entry. I just added this as I found it useful as a journal!

Figure 6: Having tapped a particular mood datapoint, the user may edit all its fields. This includes the date and time, the valence, and an optional note.

of recording, and additionally a note which is stored with this particular data point. The note feature was added simply as I found it convenient when using the app for monitoring myself; I do not use it in any data analysis. Datapoints can effortlessly be edited; their representation changes dynamically - a lot of effort in this project went into making these interactions effortless.

The data collection tab gives the user information and full control over their collected health data. First, a text explains why data is collected, which data can be collected, how it is used, where it will be stored and which control the user has over this data. Note that much of this information has already been given on the first start of the app; the text here is more comprehensive and is permanently displayed. See figures (7) and (8). It is explained that data is


4:52

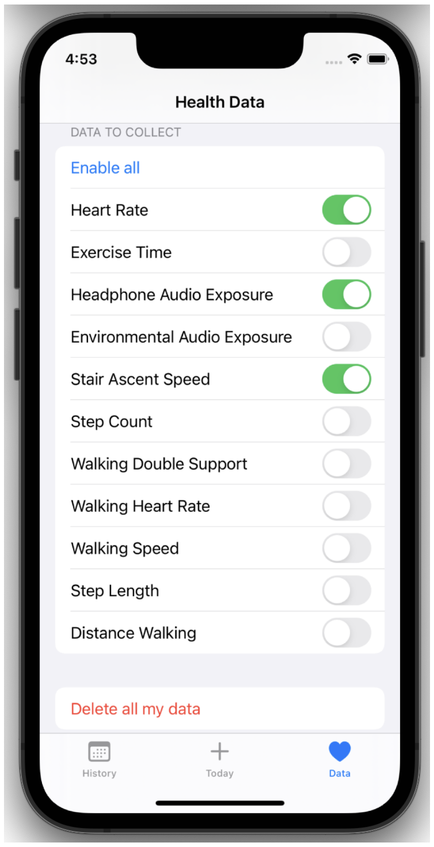
Health Data

Enable or disable health data to be collected. When enabled, your data will be sent to our server. This is to support a research project, finding connections between reported mood data and collected health data. We do not collect or even have access to directly identifying information, such as your name, email address, or location. Data is taken from Apple Health and collected by your iPhone and Apple Watch. Data will be collected only in the relevant timespans - one week before your first self-reported mood data, up to your last self-reported mood data. When disabling any of these categories, all data in that category will be deleted from our server. No data in your personal Apple Health database is edited or deleted. You can also at any point request complete deletion of your data from our servers.

DATA TO COLLECT

Figure 7: In the Health Data tab, there is a detailed explanation for which data is being collected and which control the user has over the data.

collected via the Apple HealthKit API. Users can select which type of data is collected with granular control. Upon turning on access to a new type of data in HealthKit, users are taken to the iPhone's permission screen to allow access to this particular type of data, as outlined in the Apple Developer Documentation


4:53

Health Data

DATA TO COLLECT

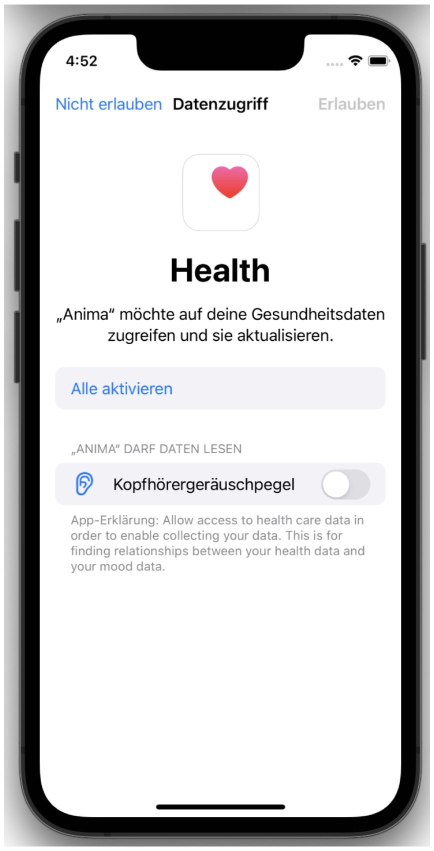
Enable all

Heart Rate	✓
Exercise Time	
Headphone Audio Exposure	✓
Environmental Audio Exposure	
Stair Ascent Speed	✓
Step Count	
Walking Double Support	
Walking Heart Rate	
Walking Speed	
Step Length	
Distance Walking	

Delete all my data

Figure 8: The user may select or deselect any category of data to be collected. Upon deselection, data collection is stopped and existing data on the server is deleted.

[App]. Users can also choose to enable all data to be collected, in which case the authorization screen will appear only once. See figure (9). Disabling any of



Health

„Anima" möchte auf deine Gesundheitsdaten zugreifen und sie aktualisieren.

„ANIMA" DARF DATEN LESEN

Kopfhörergeräuschpegel

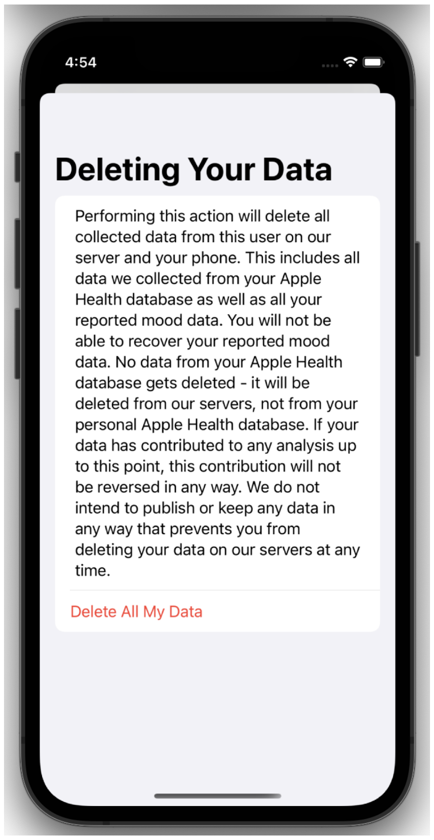
App-Erklärung: Allow access to health care data in order to enable collecting your data. This is for finding relationships between your health data and your mood data.

Figure 9: Upon enabling collection of a particular data type, the user is sent to the Apple HealthKit permission page where she can allow access to this particular type of data.

these data types will do two things: it will stop sending new datapoints to be stored on the server, and it will also delete any previously stored datapoints on the server. This ensures that what the user sees is exactly which kind of data exists on the server. Finally, on this tab users can delete all their data. This action will not be done immediately. Instead, a sheet will open to explain what deleting your data means. See figure (10).

6.7 Code Architecture

The application is built entirely in Swift. The architecture of the code follows a strict Model-View-ViewModel pattern (MVVM). This ties in nicely with the modern SwiftUI framework, which comes with multiple helpers in the form of



4:54

Deleting Your Data

Performing this action will delete all collected data from this user on our server and your phone. This includes all data we collected from your Apple Health database as well as all your reported mood data. You will not be able to recover your reported mood data. No data from your Apple Health database gets deleted - it will be deleted from our servers, not from your personal Apple Health database. If your data has contributed to any analysis up to this point, this contribution will not be reversed in any way. We do not intend to publish or keep any data in any way that prevents you from deleting your data on our servers at any time.

Figure 10: Before deleting, it is explained to the user what deleting their data will do and what it will not do. Specifically, it will delete all their data in the application and the server; it will not, however, delete any data in Apple Health.

property wrappers built specifically for this architecture pattern. The model is defined in 'AnimaModel.swift', and defines all possibilities to get and set data. The viewmodel is defined in 'ViewModel.swift'. It references a model with the @Published keyword; this takes care that changes to the model get propagated to the UI automatically. The main views are defined in 'AnimaApp.swift' and 'ContentView.swift'. The views in these reference a viewmodel with the @ObservedObject keyword; this again makes sure that any changes get propagated to the UI immediately. It would be out of scope for this project to explain these patterns. I followed the patterns described in the Stanford course CS193p (Developing Applications for iOS using SwiftUI) meticulously [Heg21]. Importantly, these patterns enabled me to handle all slow tasks asynchronously, such as connecting to the server and synchronizing data, while the UI would be updated automatically when data changes. This gives rise to a very smooth user experience overall. Besides the Apple libraries SwiftUI and HealthKit, I used the third party library and backend service Realm to synchronize data with a server. I chose this over Apple's own Core Data framework, as it proved much more flexible; for example, Core Data only allows fully public or fully private data which is not acceptable for this project. Realm provides automatic database synchronization and full flexibility over permissions.

6.8 Data Storage and Security

Realm provides a fully functional server backend with a web interface. I used their free tier MongoDB Atlas cloud service. The server location can be set to be in the European Union, using Amazon AWS infrastructure. The server was configured so that anonymous users may access only their own data. The non-relational database is configured via a flexible schema; the schema for the main type QuantitySample was this:

```
{
  "title": "QuantitySample",
  "bsonType": "object",
  "required": [
    "_id",
    "_partition",
    "quantityType",
    "quantity",
    "unit",
    "startDate",
    "endDate"
  ],
  "properties": {
    "_id": {
      "bsonType": "string"
    },
    "_partition": {
      "bsonType": "string"
    },
    "quantityType": {
      "bsonType": "string"
    },
    "quantity": {
      "bsonType": "double"
    },
    "unit": {
      "bsonType": "string"
    },
    "startDate": {
      "bsonType": "date"
    },
    "endDate": {
      "bsonType": "date"
    },
    "productType": {
      "bsonType": "string"
    },
    "sourceVersion": {
      "bsonType": "string"
    },
    "sourceOperatingSystemVersion": {
      "bsonType": "string"
    },
    "deviceName": {
      "bsonType": "string"
    },
    "deviceModel": {
      "bsonType": "string"
    },
    "motionContext": {
      "bsonType": "int"
    }
  }
}
```

This defines all fields required of an Apple HealthKit quantity. Data was not encrypted on the server; thus, trust is extended to AWS and Realm MongoDB, not only the researchers.

6.9 Data Pipeline

Data is generated on the iPhone or Apple Watch, then stored in the Apple HealthKit database on the iPhone. This is accessed by the application developed for the iPhone using the HealthKit API. Data is then sent to a AWS server run by MongoDB Atlas; Realm and its Swift library RealmSwift manage the data synchronization. Data is then accessed by a python script via the Atlas Python API. Data is never encrypted. It is stored in plain text on a AWS server in the European Union. Users can always delete all their data in the pipeline; this is fully automated, no action by a server administrator has to be taken. This automation is important for the scalability of GDPR compliance: data of many users may be collected without additional effort (except probably upgrading from the MongoDB free tier, as requests are limited).

6.10 Proof-of-concept Data Analysis

Data can be accessed via the MongoDB python API:

```
import pymongo
from pymongo.server_api import ServerApi
from pprint import pprint

user = "********"
password = "********"
client = pymongo.MongoClient("mongodb+srv://"+user+":"+password+"
    ↪ @cluster0.sq3ax.mongodb.net/?retryWrites=true&w=majority",
    ↪  server_api=ServerApi('1'))
db = client['AnimaDB']
pprint(db.list_collection_names())
```

```
['QuantitySample', 'MoodData', 'DiscardedUserID']
```

These results are a fully functioning proof-of-concept for follow-up work, in which I plan to carry out more sophisticated analysis. Here, I find a positive correlation between my average walking speed and self-reported mood data:

```
from datetime import datetime, timedelta
import numpy as np

moodData = db['MoodData']
quantitySamples = db['QuantitySample']

partition = "6201d3b322ecfa351e498dbf" # the anonymous user id
    ↪ for my user
quantityType = "HKQuantityTypeIdentifierWalkingSpeed" # the
    ↪ quantitytype I am interested in

delta = timedelta(days = 1.0) # the duration of physical health
    ↪ data I am interested per sample

X = []
```

```
Y = []

for y in moodData.find({"_partition": partition}):
    d = y["timestamp"]
    start = d - delta
    end = d

    # this pipeline will first filter for Walking Speed values in
        ↪ the relevant time span
    # then, it will create a group object with a singular entry of
        ↪  the average walking speed in that time span
    pipeline = [
        {"$match": {"_partition": partition,
                    "startDate": {'$lt': end, '$gte': start},
                    "quantityType": quantityType} },
         {"$group": {
             "_id": 0,
             "averageWalkingSpeed": { "$avg": "$quantity" }
         }
        }
    ]
    averageWalkingSpeed = quantitySamples.aggregate(pipeline).next
        ↪ ()["averageWalkingSpeed"] # this is a singleton group,
        ↪ thus next() will get us the one grouped element, and ["
        ↪ averageWalkingSpeed"] will return a float

    X.append(averageWalkingSpeed)
    Y.append(y["valence"])

npY = np.array(Y)
npX = np.array(X)

np.corrcoef(npY, npX)[0, 1] # correlation coeffifient
```

```
0.08949506568189569
```

7 Follow-up work

In follow-up work, I want to use this application and data pipeline to predict a one-dimensional measure of mental wellbeing from automatically collected physical health data. People generate data about themselves in abundance, yet this data is mainly used for targeted advertising and selling products. I believe that using personal wearable and health data for research and eventually to improve mental health treatment is valuable and promising if done transparently and ethically.

8 Conclusion

In this project, I set out to establish design principles for a data collection application for research, to be published to individual study participants. I established these principles based on the EU GDPR, and built an application based on these principles, taking great care in the design of the user experience and user interface to carefully consider the principles of transparency, privacy, and user control. The data collection application is fully functional with an operational backend server based on a non-relational database run on MongoDB Atlas using a European AWS cloud server. I used the application to gather data about myself, including Apple Health data from an iPhone and an Apple Watch. Following my data pipeline, I imported data to python using the backend server's API. In a proof-of-concept data analysis, I found a positive correlation between my average walking speed the day before self-reporting mood data and the valence of the mood data.

References

[Ahm+14] Farah Ahmad et al. "Single item measures of self-rated mental health: a scoping review". In: *BMC Health Services Research* 14.1 (Sept. 2014). DOI: `10.1186/1472-6963-14-398`. URL: `https://doi.org/10.1186/1472-6963-14-398`.

[16a] *Article 5(1)(a) of the General Data Protection Regulation.* European Commission. Apr. 1, 2016. URL: `https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679&from=EN#d1e3732-1-1` (visited on 06/17/2022).

[16b] *Article 5(1)(b) of the General Data Protection Regulation.* European Commission. Apr. 1, 2016. URL: `https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679&from=EN#d1e3732-1-1` (visited on 06/17/2022).

[16c] *Article 5(1)(c) of the General Data Protection Regulation.* European Commission. Apr. 1, 2016. URL: `https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679&from=EN#d1e3732-1-1` (visited on 06/17/2022).

[16d] *Recital (39) of the General Data Protection Regulation.* European Commission. Apr. 1, 2016. URL: `https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679&from=EN#d1e3732-1-1` (visited on 06/17/2022).

[16e] *Recital (65) of the General Data Protection Regulation.* European Commission. Apr. 1, 2016. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679&from=EN#d1e3732-1-1 (visited on 06/17/2022).

[16f] *Recital (97) of the General Data Protection Regulation.* European Commission. Apr. 1, 2016. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679&from=EN#d1e3732-1-1 (visited on 06/17/2022).

[RD16] Blaine Reeder and Alexandria David. "Health at hand: A systematic review of smart watch uses for health and wellness". In: *Journal of Biomedical Informatics* 63 (2016), pp. 269–276. ISSN: 1532-0464. DOI: https://doi.org/10.1016/j.jbi.2016.09.001. URL: https://www.sciencedirect.com/science/article/pii/S1532046416301137.

[Yae16] 2016 12:57 pm UTC Yael Grauer - Jul 28. *Dark patterns are designed to trick you (and they're all over the web).* July 2016. URL: https://arstechnica.com/information-technology/2016/07/dark-patterns-are-designed-to-trick-you-and-theyre-all-over-the-web/.

[Ost+17] Kirsten Ostherr et al. "Trust and privacy in the context of user-generated health data". In: *Big Data & Society* 4.1 (2017), p. 2053951717704673. DOI: 10.1177/2053951717704673. eprint: https://doi.org/10.1177/2053951717704673. URL: https://doi.org/10.1177/2053951717704673.

[Dix18] Helen Dixon. "Regulate to Liberate: Can Europe Save the Internet?" In: *Foreign Affairs* 97.5 (2018), pp. 28–32. ISSN: 00157120. URL: http://www.jstor.org/stable/44823910 (visited on 06/27/2022).

[18] "Finding a sensible approach to sensitive data". In: *Scientific Data* 5.1 (Nov. 2018). DOI: 10.1038/sdata.2018.253. URL: https://doi.org/10.1038/sdata.2018.253.

[Gre18] Samuel Greengard. "Weighing the impact of GDPR". In: *Communications of the ACM* 61.11 (2018), pp. 16–18.

[Gui+18] Jerry Guintivano et al. "PPD ACT: an app-based genetic study of postpartum depression". In: *Translational Psychiatry* 8.1 (Nov. 2018). DOI: 10.1038/s41398-018-0305-5. URL: https://doi.org/10.1038/s41398-018-0305-5.

[Mer18] Sam Meredith. *Facebook-Cambridge Analytica: A timeline of the data hijacking scandal.* 2018. URL: https://www.cnbc.com/2018/04/10/facebook-cambridge-analytica-a-timeline-of-the-data-hijacking-scandal.html (visited on 03/12/2022).

[Eco19] The Economist. *The French fine against Google is the start of a war.* 2019. URL: https://www.economist.com/business/2019/01/24/the-french-fine-against-google-is-the-start-of-a-war (visited on 06/12/2022).

[Gar+20] Maria Salud Garcia-Gutierrez et al. “Biomarkers in Psychiatry: Concept, Definition, Types and Relevance to the Clinical Reality”. In: *Frontiers in Psychiatry* 11 (May 2020). DOI: 10.3389/fpsyt.2020.00432. URL: https://doi.org/10.3389/fpsyt.2020.00432.

[20] *GDPR and research – an overview for researchers.* 2020. URL: https://www.ukri.org/about-us/policies-standards-and-data/gdpr-and-research-an-overview-for-researchers/ (visited on 06/12/2022).

[Jim+20] Heather S. L. Jim et al. “Innovations in research and clinical care using patient-generated health data”. In: *CA: A Cancer Journal for Clinicians* 70.3 (Apr. 2020), pp. 182–199. DOI: 10.3322/caac.21608. URL: https://doi.org/10.3322/caac.21608.

[Spa+20] Dimitris Spathis et al. *Learning Generalizable Physiological Representations from Large-scale Wearable Data.* 2020. DOI: 10.48550/ARXIV.2011.04601. URL: https://arxiv.org/abs/2011.04601.

[SAL20] Madeena Sultana, Majed Al-Jefri, and Joon Lee. “Using Machine Learning and Smartphone and Smartwatch Data to Detect Emotional States and Transitions: Exploratory Study”. In: *JMIR mHealth and uHealth* 8.9 (Sept. 2020), e17818. DOI: 10.2196/17818. URL: https://doi.org/10.2196/17818.

[Gar21] Howard N. Garb. “Race bias and gender bias in the diagnosis of psychological disorders”. In: *Clinical Psychology Review* 90 (2021), p. 102087. ISSN: 0272-7358. DOI: https://doi.org/10.1016/j.cpr.2021.102087. URL: https://www.sciencedirect.com/science/article/pii/S0272735821001306.

[Heg21] Paul Hegarty. *CS193p (Developing Applications for iOS using SwiftUI).* 2021. URL: https://cs193p.sites.stanford.edu (visited on 03/12/2022).

[Vin21] James Vincent. *California bans 'dark patterns' that trick users into giving away their personal data.* Mar. 2021. URL: https://www.theverge.com/2021/3/16/22333506/california-bans-dark-patterns-opt-out-selling-data.

[SSK22] Tomoki Saito, Hikaru Suzuki, and Akifumi Kishi. “Predictive Modeling of Mental Illness Onset Using Wearable Devices and Medical Examination Data: Machine Learning Approach”. In: *Frontiers in Digital Health* 4 (2022). ISSN: 2673-253X. DOI: 10.3389/fdgth.2022.861808. URL: https://www.frontiersin.org/article/10.3389/fdgth.2022.861808.

[App] Apple. *Authorizing Access to Health Data*. URL: https://developer.apple.com/documentation/healthkit/authorizing_access_to_health_data (visited on 02/13/2022).

[Ola+] Iyiola E. Olatunji et al. "A Review of Anonymization for Healthcare Data". In: *Big Data* 0.0 (0). PMID: 35271377, null. DOI: 10.1089/big.2021.0169. eprint: https://doi.org/10.1089/big.2021.0169. URL: https://doi.org/10.1089/big.2021.0169.

9 Appendix

9.1 AnimaApp.swift

```
import SwiftUI
import RealmSwift

let appid = "anima-owkag"
let app = App(id: appid)

@main
struct AnimaApp: SwiftUI.App {
    let viewModel: ViewModel? = nil

    var body: some Scene {
        WindowGroup {
            SyncContentView(app: app)
        }
    }
}

struct SyncContentView: View {
    // Observe the Realm app object in order to react to login
        ↪ state changes.
    @ObservedObject var app: RealmSwift.App

    var body: some View {
        if let user = app.currentUser {
            // If there is a logged in user, pass the user ID as
                ↪ the
            // partitionValue to the view that opens a realm.
            OpenSyncedRealmView().environment(\.partitionValue,
                ↪ user.id)
        } else {
            // If there is no user logged in, show the login view.
                ↪  This effectively means the user has not seen
                ↪ the initial greeting and introductino message.
                ↪ Login in anonymous and handled by realm
            LoginView()
        }
    }
}

struct LoginView: View {
    // Hold an error if one occurs so we can display it.
    @State var error: Error?

    // Keep track of whether login is in progress.
    @State var isLoggingIn = false
    var body: some View {
        VStack {
            if isLoggingIn {
                ProgressView()
            }
            if let error = error {
                Text("Error:␣\(error.localizedDescription)")
            }
            NavigationView {
                Form {
                    Section {
                        Text("By␣using␣this␣application,␣you␣are␣
                            ↪ helping␣us␣research␣-␣thank␣you!")
                        Text("This␣application␣collects␣sensitive␣
                            ↪ personal␣data␣about␣you.␣This␣
                            ↪ includes␣several␣categories␣of␣data␣
                            ↪ collected␣by␣your␣iPhone␣and␣Apple␣
                            ↪ Watch.␣You␣have␣complete␣control␣over
                            ↪ ␣which␣categories␣of␣data␣are␣
                            ↪ collected.␣This␣app␣also␣lets␣you␣
                            ↪ self-report␣mood␣data,␣which␣will␣
                            ↪ also␣be␣collected.␣All␣data␣is␣stored
                            ↪ ␣on␣our␣servers␣and␣will␣be␣used␣for␣
                            ↪ research.")
                        Text("You␣may␣always␣contact␣us␣to␣clarify␣
                            ↪ which␣research␣project␣you␣are␣
                            ↪ helping␣with␣-␣find␣our␣contact␣
                            ↪ information␣in␣the␣App␣Store.")
                        Text("This␣would␣be␣the␣point␣to␣give␣infos␣
                            ↪ about␣the␣research␣project,␣
                            ↪ identification␣number␣and␣so␣on␣-␣if␣
                            ↪ this␣project␣went␣through␣an␣ethics␣
                            ↪ committee␣and␣were␣distributed␣
                            ↪ publically␣which␣it␣is␣not.")
                        Text("By␣clicking␣Continue,␣you␣acknowledge␣
                            ↪ that␣you␣are␣aware␣this␣application␣
                            ↪ collects␣data␣from␣Apple␣Health␣as␣
                            ↪ well␣as␣your␣self␣reported␣mood␣data,
                            ↪ ␣which␣is␣sent␣to␣our␣servers.")
                        Button("Continue") {
                            // Button pressed, so log in
                           isLoggingIn = true
                           app.login(credentials: .anonymous) {
                               ↪ result in
                               isLoggingIn = false
                               if case let .failure(error) = result
                                   ↪ {
                                   print("Failed␣to␣log␣in:␣\(error.
                                       ↪ localizedDescription)")
                                   // Set error to observed property
                                       ↪ so it can be displayed
                                   self.error = error
                                   return
                               }
                               // Other views are observing the app
                                   ↪ and will detect
                               // that the currentUser has changed.
                                   ↪ Nothing more to do here.
                               print("Logged␣in")
                           }
                        }.disabled(isLoggingIn)
                    }.navigationTitle(Text("Welcome!"))
                }
            }
         }
     }
  }

  /// This view opens a synced realm.
  struct OpenSyncedRealmView: View {
         // Use AsyncOpen to download the latest changes from
         // your Realm app before opening the realm.
         // Leave the `partitionValue` an empty string to get this
         // value from the environment object passed in above.
         @AsyncOpen(appId: appid, partitionValue: "", timeout: 10000)
          ↪ var asyncOpen

         var body: some View {
         switch asyncOpen {
         // Starting the Realm.asyncOpen process.
         // Show a progress view.
         case .connecting:
            VStack {
                Text("Connecting")
                ProgressView()
            }
         // Waiting for a user to be logged in before executing
         // Realm.asyncOpen.
         case .waitingForUser:
            ProgressView("Waiting␣for␣user␣to␣log␣in...")
         // The realm has been opened and is ready for use.
         // Show the content view.
         case .open(let realm):
            let viewModel = ViewModel(realm: realm)
            ContentView(viewModel: viewModel)
            // The realm is currently being downloaded from the
                ↪ server.
            // Show a progress view.
            case .progress(let progress):
                ProgressView(progress)
            // Opening the Realm failed.
            // Show an error view.
            case .error(let error):
                ErrorView(error: error)
         }
         }
  }
  struct ErrorView: View {
         var error: Error

         var body: some View {
         VStack {
            Text("Error␣opening␣the␣realm:␣\(error.
                ↪ localizedDescription)")
         }
         }
  }

  enum Errors: Error {
         case runtimeError(String)
  }

  struct SyncedContentView_Previews: PreviewProvider {
         static var previews: some View {
        // SyncContentView(app: app)
        // .preferredColorScheme(.light)
         LoginView()
    }
}
```

9.2 AnimaModel.swift

```
import Foundation
import SwiftUI
import Realm
import RealmSwift
import HealthKit

struct AnimaModel {

    let realm: RealmS

    // Device-specific preferences:
    let userDefaults = UserDefaults.standard

    var isCollectingSampleDataDict: [String:Bool]
    var isCollectingSampleDataDictKey = "
        ↪ anima_collectingSampleData"

    // in order to show error messages in any tab. Quite
        ↪ simplistically, this will only allow a single error
        ↪ message at a time, no queueing
    var errorMessage: String?
    var errorMessageIsPresented = false

    init(realm: Realm?) {
        if let realm = realm {
            self.realm = realm
        }
        else {
            self.realm = try! Realm()
        }

        isCollectingSampleDataDict = userDefaults.object(forKey:
            ↪ isCollectingSampleDataDictKey) as? [String:Bool] ??
            ↪  [:]
    }

    mutating func addMoodData(_ data: MoodData) {
```

```
    try! realm.write {
        realm.add(data)
    }
}

mutating func changeMoodData(_ moodData: MoodData, with f: (_
    ↪ moodData : MoodData) -> Void) {
    let thawedRealm = realm.thaw()

    try! thawedRealm.write {
        if let data = moodData.thaw() {
            f(data)
        }
    }
}

mutating func removeMoodData(_ data: MoodData) {
    try! realm.write {
        if let data = data.thaw() {
            realm.delete(data)
        }
    }
}

mutating func addQuantitySamples(_ samples: [QuantitySample])
    ↪ {
    let thawedRealm = realm.thaw()
    try! thawedRealm.write {
        thawedRealm.add(samples, update: .modified)
    }
}

mutating func removeQuantitySamples(ofType: HKQuantityType) {
    let thawedRealm = realm.thaw()
    try! thawedRealm.write {
        let quantityType = ofType.identifier
        // construct query to delete all objects of that
            ↪ quantity type
        thawedRealm.delete(realm.objects(QuantitySample.self).
            ↪ filter("quantityType␣==␣'"+quantityType+"'"))
    }
}

mutating func removeAllData(){
    let thawedRealm = realm.thaw()
    try! thawedRealm.write {
            thawedRealm.deleteAll()
        }
    }

    mutating func discardUser() {
        try! realm.write {
            let d = DiscardedUserID(value: app.currentUser?.id)
            realm.add(d) // adds the current user as discarded
        }
        if let user = app.currentUser {
            user.logOut()
        }
    }

    func isCollectingSampleData(ofType sampleType: HKSampleType)
        ↪ -> Bool {
        return isCollectingSampleDataDict[sampleType.identifier]
            ↪ ?? false
    }

    mutating func setCollectingSampleData(ofType sampleType:
        ↪ HKSampleType, to newValue: Bool) {
        isCollectingSampleDataDict[sampleType.identifier] =
            ↪ newValue
        userDefaults.set(isCollectingSampleDataDict, forKey:
            ↪ isCollectingSampleDataDictKey)
    }

// func allQuantitySamples(ofType: HKQuantityType) -> Array<
    ↪ QuantitySample> {
// // note the filtering could be done faster using type unsafe
    ↪ filtering using a string predecate (look at how realm
    ↪ recommends to filter)
// Array(realm.objects(QuantitySample.self).filter {
    ↪ quantitySample in
// quantitySample.quantityType == ofType.identifier
// })
// }

    var allMoodData: Array<MoodData> {
        Array(realm.objects(MoodData.self).sorted(byKeyPath: "
            ↪ timestamp", ascending: true))
    }
}
```

```
final class MoodData: Object, Identifiable, NSCopying {

    @Persisted(primaryKey: true) var _id: ObjectId = ObjectId.
        ↪ generate()
    @Persisted var valence: Double
    @Persisted var notes: String = ""
    @Persisted var timestamp: Date = Date()

    func copy(with zone: NSZone? = nil) -> Any {
        let copy = MoodData()
        copy.valence = valence
        copy.notes = notes
        copy.timestamp = timestamp
        return copy
    }
}

final class DiscardedUserID: Object {
    @Persisted(primaryKey: true) var _id: ObjectId = ObjectId.
        ↪ generate()
    @Persisted var userId: String
}

struct AnimaModel_Previews: PreviewProvider {
    static var previews: some View {
        SyncContentView(app: app)
            .preferredColorScheme(.light)
    }
}
```

9.3 ContentView.swift

```
import SwiftUI
import RealmSwift

struct ContentView: View {
    @ObservedObject var viewModel: ViewModel
    @State var selection: String = "reporter"

    var body: some View {
        TabView(selection: $selection) {
            HistoryView(viewModel: viewModel, editMoodData:
                ↪ MoodData())
                .tabItem {
                    Label("History", systemImage: "calendar")
                }.tag("history")
            MoodReporterView(addMoodData: viewModel.addMoodData)
                .tabItem {
                    Label("Today", systemImage: "plus")
                }.tag("reporter")
            HealthDataView(viewModel: viewModel)
                .tabItem {
                    Label("Data", systemImage: "heart")
                }.tag("health␣data")
        }
        .alert("Something␣went␣wrong!", isPresented: $viewModel.
            ↪ errorMessageIsPresented) {
            if let errorMessage = viewModel.errorMessage {
                Text(errorMessage)
            }
            Button("OK", role: .cancel) { }
        }
    }
}

struct HistoryView: View {
    @ObservedObject var viewModel: ViewModel

    @State var editingMoodData: MoodData? = nil
    @State private var showingEditingSheet = false
    @State private var showingFirstInfo = true

    /// Selected task for updating status.
    @StateRealmObject var editMoodData: MoodData

    var body: some View {
        ScrollView(.horizontal) {
            HStack(spacing: 2) {
                ForEach(viewModel.moodData) { data in
                    MoodDataView(moodData: data)
                        .accentColor(.green)
                        .onTapGesture {
                            editMoodData = data
                            showingEditingSheet = true
                        }
                }
            }
        }
        .sheet(isPresented: $showingEditingSheet) {
            EditMoodDataView(data: editMoodData, viewModel:
                ↪ viewModel, isShowing: $showingEditingSheet)
        }
        .alert("Browse␣Your␣Mood␣Data", isPresented:
            ↪ $showingFirstInfo) {
            Button("OK!", role: .cancel) { }
        } message: {
            Text("On␣this␣screen,␣you␣can␣browse␣all␣the␣mood␣data
                ↪ ␣you␣have␣previously␣self-reported.␣Scroll␣left␣
                ↪ and␣right␣to␣explore␣all␣your␣data.␣Tap␣on␣any␣
                ↪ of␣the␣bars␣to␣edit␣or␣delete␣a␣particular␣entry
                ↪ .")
        }
    }
}

struct MoodDataView: View {

    @ObservedRealmObject var moodData: MoodData

    var body: some View {
        GeometryReader { geo in
            VStack {
                Spacer()
                ZStack {
                    Color.accentColor
                    if (moodData.notes != "") {
                        ZStack(alignment: .bottomTrailing) {
                            Color.clear
                            Image(systemName: "rectangle.and.pencil.
                                ↪ and.ellipsis")
                                .padding(5)
                        }
                    }
                }.frame(height: geo.size.height*moodData.valence)
            }
        }.frame(width: 100)
         .contentShape(Rectangle())
    }
}

struct MoodReporterView: View {
    var addMoodData: (Double) -> Void

    @State var sliderValue: Double = 0.5
    @State var showingFirstInfo: Bool = true

    var body: some View {
        VStack(spacing: 0) {
            CustomSlider(sliderValue: $sliderValue)
                .accentColor(Color.green)
            Button(action: {
                addMoodData(sliderValue)
            }, label: {
                ZStack {
                    Color(hue: 0.3339, saturation: 0.7, brightness:
                        ↪  0.6)
                    Image(systemName: "plus.circle")
                        .accentColor(Color.white)
                        .font(.system(size: 60))
                }
            }).frame(height: 100)
        }
        .ignoresSafeArea(edges: .top)
        .onAppear {
            sliderValue = 0.5
        }
        .alert("Report␣Your␣Mood", isPresented: $showingFirstInfo)
            ↪  {
            Button("OK!", role: .cancel) { }
        } message: {
            Text("On␣this␣screen,␣you␣can␣self-report␣your␣mood.␣
                ↪ Drag␣the␣green␣bar␣up␣or␣down␣-␣up␣means␣you’re␣
                ↪ doing␣great,␣down␣not␣so␣much.␣Hit␣the␣Plus␣
                ↪ button␣to␣record!")
        }
    }
}

struct EditMoodDataView: View {

    @ObservedRealmObject var data: MoodData
    @ObservedObject var viewModel: ViewModel
    @Binding var isShowing: Bool

// @State var valenceEditingValue: Double = 0.0
// @State var notesEditingValue: String = ""
    @State private var showingAlert = false

    var body: some View {

        NavigationView {
            VStack(alignment: .leading) {
                Form {
                    Section {
                        // Text(data.timestamp, style: .time)
                        DatePicker(
                                "Time",
                                selection: $data.timestamp,
                                displayedComponents: [.date, .
                                    ↪ hourAndMinute]
                            )
                    }
                    Section {
                        Text("Mood␣on␣this␣day:␣")
                        Slider(value: $data.valence)
                            .tint(.green)
// .onSubmit {
// viewModel.changeMoodData(data) { moodData in
// moodData.valence = valenceEditingValue
// }
// }
                    }
                    Section {
                        Text("Notes:")
                        TextEditor(text: $data.notes)
// .onSubmit {
// viewModel.changeMoodData(data) { moodData in
// moodData.notes = notesEditingValue
// }
// }
                    }
                    Section {
                        Button("Delete␣this␣entry", role: .
                            ↪ destructive) {
                            showingAlert = true
                        }
                        .alert(isPresented:$showingAlert) {
                            Alert(
                                title: Text("Are␣you␣sure␣you␣want␣
                                    ↪ to␣delete␣this␣entry?"),
                                primaryButton: .destructive(Text("
                                    ↪ Delete")) {
                                    viewModel.removeMoodData(data)
                                    isShowing = false
                                },
                                secondaryButton: .cancel()
                            )
                        }
                    }
```

```
            }
        }.navigationTitle(Text(data.timestamp, style: .date))
    }
  }
}

struct ContentView_Previews: PreviewProvider {
    static var viewModel = ViewModel(realm: nil)

    static var previews: some View {
// SyncContentView(app: app)
// .preferredColorScheme(.light)
        Group {
            ContentView(viewModel: viewModel)
                .preferredColorScheme(/*@START_MENU_TOKEN@*/.dark/*
                    ↪ @END_MENU_TOKEN@*/)
        }
    }
}
```

9.4 HealthDataModel.swift

```
import Foundation
import HealthKit
import RealmSwift

class QuantitySample: Object, Identifiable {

    @Persisted(primaryKey: true) var _id: String
    @Persisted var quantityType: String
    @Persisted var quantity: Double
    @Persisted var unit: String
    @Persisted var startDate: Date
    @Persisted var endDate: Date
    @Persisted var productType: String?
    @Persisted var sourceVersion: String?
    @Persisted var sourceOperatingSystemVersion: String?
    @Persisted var deviceName: String?
    @Persisted var deviceModel: String?

    @Persisted var motionContext: Int?

    convenience init(from quantitySample: HKQuantitySample,
        ↪ quantityUnit: HKUnit) {
        self.init()
        self._id = quantitySample.uuid.uuidString
        self.quantityType = quantitySample.quantityType.identifier
        self.unit = quantityUnit.unitString
        self.quantity = quantitySample.quantity.doubleValue(for:
            ↪ quantityUnit)
        self.startDate = quantitySample.startDate
        self.endDate = quantitySample.endDate
        self.productType = quantitySample.sourceRevision.
            ↪ productType
        self.sourceVersion = quantitySample.sourceRevision.version
        self.sourceOperatingSystemVersion = String(quantitySample.
            ↪ sourceRevision.operatingSystemVersion.majorVersion)
            ↪  + "." +
                                        String(quantitySample.
                                            ↪ sourceRevision.
                                            ↪ operatingSystemVersion
                                            ↪ .minorVersion) + ".
                                            ↪ " +
                                        String(quantitySample.
                                            ↪ sourceRevision.
                                            ↪ operatingSystemVersion
                                            ↪ .patchVersion)
        self.deviceName = quantitySample.device?.name
        self.deviceModel = quantitySample.device?.model

        self.motionContext = quantitySample.metadata?[
            ↪ HKMetadataKeyHeartRateMotionContext] as? Int
    }
}
```

9.5 HealthDataView.swift

```
import SwiftUI
import HealthKit

struct HealthDataView: View {

    @ObservedObject var viewModel: ViewModel
    @State var showDeleteAlert = false

    var body: some View {


        NavigationView {
```

```
Form {
    Text("Enable␣or␣disable␣health␣data␣to␣be␣collected
        ↪ .␣When␣enabled,␣your␣data␣will␣be␣sent␣to␣
        ↪ our␣server.␣This␣is␣to␣support␣a␣research␣
        ↪ project,␣finding␣connections␣between␣
        ↪ reported␣mood␣data␣and␣collected␣health␣data
        ↪ .␣We␣do␣not␣collect␣or␣even␣have␣access␣to␣
        ↪ directly␣identifying␣information,␣such␣as␣
        ↪ your␣name,␣email␣address,␣or␣location.␣Data␣
        ↪ is␣taken␣from␣Apple␣Health␣and␣collected␣by␣
        ↪ your␣iPhone␣and␣Apple␣Watch.␣Data␣will␣be␣
        ↪ collected␣only␣in␣the␣relevant␣timespans␣-␣
        ↪ one␣week␣before␣your␣first␣self-reported␣
        ↪ mood␣data,␣up␣to␣your␣last␣self-reported␣
        ↪ mood␣data.␣When␣disabling␣any␣of␣these␣
        ↪ categories,␣all␣data␣in␣that␣category␣will␣
        ↪ be␣deleted␣from␣our␣server.␣No␣data␣in␣your␣
        ↪ personal␣Apple␣Health␣database␣is␣edited␣or␣
        ↪ deleted.␣You␣can␣also␣at␣any␣point␣request␣
        ↪ complete␣deletion␣of␣your␣data␣from␣our␣
        ↪ servers.")
        .fixedSize(horizontal: false, vertical: true)
        .padding(10)

    Section(header: Text("Data␣to␣collect")) {
        if(!viewModel.allQueryManagersEnabled) {
            Button("Enable␣all") {
                viewModel.enableAllQueryManagers()
            }
        } else {
            Button("Disable␣all") {
                viewModel.disableAllQueryManagers()
            }
        }

        ForEach(viewModel.queryManagers) { queryManager
            ↪  in
            QueryToggle(query: queryManager)
        }
    }
    Section {
        Button("Delete␣all␣my␣data", role: .destructive
            ↪ ) {
            showDeleteAlert = true
        }
    }
```

```
            }.navigationTitle("Health␣Data")
        }.sheet(isPresented: $showDeleteAlert) {

            NavigationView {
                Form {
                    Text("Performing␣this␣action␣will␣delete␣all␣
                        ↪ collected␣data␣from␣this␣user␣on␣our␣
                        ↪ server␣and␣your␣phone.␣This␣includes␣all
                        ↪ ␣data␣we␣collected␣from␣your␣Apple␣
                        ↪ Health␣database␣as␣well␣as␣all␣your␣
                        ↪ reported␣mood␣data.␣You␣will␣not␣be␣able
                        ↪ ␣to␣recover␣your␣reported␣mood␣data.␣No␣
                        ↪ data␣from␣your␣Apple␣Health␣database␣
                        ↪ gets␣deleted␣-␣it␣will␣be␣deleted␣from␣
                        ↪ our␣servers,␣not␣from␣your␣personal␣
                        ↪ Apple␣Health␣database.␣If␣your␣data␣has␣
                        ↪ contributed␣to␣any␣analysis␣up␣to␣this␣
                        ↪ point,␣this␣contribution␣will␣not␣be␣
                        ↪ reversed␣in␣any␣way.␣We␣do␣not␣intend␣to
                        ↪ ␣publish␣or␣keep␣any␣data␣in␣any␣way␣
                        ↪ that␣prevents␣you␣from␣deleting␣your␣
                        ↪ data␣on␣our␣servers␣at␣any␣time.")
                        .textSelection(.enabled).padding(5)
                    Button("Delete␣All␣My␣Data", role: .destructive
                        ↪ ) {
                        viewModel.removeAllData()
                        viewModel.discardUser() // will add the user
                            ↪  ID to the list of deleted users and
                            ↪ log out!
                        showDeleteAlert = false }
                }.navigationTitle("Deleting␣Your␣Data")
            }
        }
    }
}

struct QueryToggle: View {
    @ObservedObject var query: ViewModel.QueryManager

    var body: some View {
        Toggle(query.descriptiveName, isOn: $query.enabled)
    }
}

struct HealthDataView_Previews: PreviewProvider {
    static var viewModel = ViewModel(realm: nil)
    static var previews: some View {
        HealthDataView(viewModel: viewModel)
    }
}
```

9.6 ViewModel.swift

```
import Foundation
import RealmSwift
import HealthKit

let healthStore = HKHealthStore()

class ViewModel: ObservableObject {

    @Published private var model: AnimaModel

    var queryManagers: [QueryManager] = []

    init(realm: Realm?) {
        model = AnimaModel(realm: realm)

        queryManagers = possibleSampleTypes.map { (quantityType,
            ↪ descriptiveName, unit) in
            return QueryManager(sampleType: quantityType,
                ↪ displayName: descriptiveName, unit: unit,
                ↪ viewModel: self)
        }
    }

    // Just a list of quantities that could be potentially
        ↪ interesting. Hand picked, could be any values.
    // TODO low prio I guess this should go into some config or
        ↪ constants file
    var possibleSampleTypes: [(HKQuantityType, String, HKUnit)] =
    [(HKQuantityType(.heartRate), "Heart␣Rate", HKUnit(from: "
        ↪ count/min")),
     (HKQuantityType(.appleExerciseTime), "Exercise␣Time", HKUnit.
         ↪ hour()),
     (HKQuantityType(.headphoneAudioExposure), "Headphone␣Audio␣
         ↪ Exposure", (HKUnit.decibelAWeightedSoundPressureLevel
         ↪ ())),
     (HKQuantityType(.environmentalAudioExposure), "Environmental␣
         ↪ Audio␣Exposure", (HKUnit.
         ↪ decibelAWeightedSoundPressureLevel())),
     (HKQuantityType(.stairAscentSpeed), "Stair␣Ascent␣Speed",
         ↪ HKUnit(from: "m/s")),
     (HKQuantityType(.stepCount), "Step␣Count", HKUnit.count()),
     (HKQuantityType(.walkingDoubleSupportPercentage), "Walking␣
         ↪ Double␣Support", (HKUnit.percent())),
     (HKQuantityType(.walkingHeartRateAverage), "Walking␣Heart␣
         ↪ Rate", HKUnit(from: ("count/min"))),
     (HKQuantityType(.walkingSpeed), "Walking␣Speed", HKUnit(from:
         ↪  "m/s")),
     (HKQuantityType(.walkingStepLength), "Step␣Length", HKUnit.
         ↪ meter()),
     (HKQuantityType(.distanceWalkingRunning), "Distance␣Walking",
         ↪  HKUnit.meter())]

    func isCollectingSampleData(ofType sampleType: HKSampleType)
        ↪ -> Bool {
        let manager = queryManagers.first { m in m.sampleType ==
            ↪ sampleType }
        return manager?.enabled ?? false
    }

    func setCollectingSampleData(ofType sampleType: HKSampleType,
        ↪ to newValue: Bool) {
        if let manager = queryManagers.first(where: { m in m.
            ↪ sampleType == sampleType }) {
            manager.enabled = newValue
        }
    }

    var moodData: Array<MoodData> {
        model.allMoodData
    }

    public func addMoodData(valence: Double) {
        let moodData = MoodData()
        moodData.valence = min(max(valence, 0.0), 1.0)
        moodData.timestamp = Date()
        model.addMoodData(moodData)
    }

    public func changeMoodData(_ moodData: MoodData, with f: (_
        ↪ moodData : MoodData) -> Void) {
        model.changeMoodData(moodData, with: f)
    }
```

```
public func removeMoodData(_ moodData: MoodData) {
    model.removeMoodData(moodData)
}

public func removeAllData() {
    model.removeAllData()
}

public func discardUser() {
    model.discardUser()
}

var errorMessageIsPresented: Bool {
    get { return model.errorMessageIsPresented }
    set { model.errorMessageIsPresented = newValue }
}

var errorMessage: String? {
    get { return model.errorMessage }
    set { model.errorMessage = newValue }
}

var allQueryManagersEnabled: Bool {
    get { queryManagers.allSatisfy { m in m.enabled } }
}

func enableAllQueryManagers() {
    healthStore.requestAuthorization(toShare: nil, read: Set(
        ↪ possibleSampleTypes.map{ quantityType, _, _ in
        ↪ quantityType })) { didAsk, error in
        if let error = error {
            print("oh␣no") // TODO handle error
        } else {
            for manager in self.queryManagers {
                manager.enabled = true
            }
        }
    }
}

func disableAllQueryManagers() {
    self.queryManagers.forEach { queryManager in
        queryManager.enabled = false
    }
}
```

```
// TODO low prio think about the public interface of this
// -> which parts if any should be visible to the View?
// if none, should we have another type to interface with the
    ↪ view?

// Public fields of this type are accessed by the view
class QueryManager: Identifiable, ObservableObject {

    private var viewModel: ViewModel

    var sampleType: HKSampleType
    var descriptiveName: String
    var unit: HKUnit

    private var query: HKQuery?

    init(sampleType: HKQuantityType, displayName: String, unit
        ↪ : HKUnit, viewModel: ViewModel) {
        self.sampleType = sampleType
        self.viewModel = viewModel
        self.descriptiveName = displayName
        self.unit = unit
        self.enabled = self.enabled // running the setter
    }

    var enabled: Bool {
        get {
            return viewModel.model.isCollectingSampleData(
                ↪ ofType: sampleType) }
        set {
            // This will publish to the UI, thus do it on the
                ↪ main queue
            DispatchQueue.main.async {
                self.viewModel.model.setCollectingSampleData(
                    ↪ ofType: self.sampleType, to: newValue)
            }
            if(newValue && query == nil) {
                let newQuery = createQuery(type: sampleType)
                healthStore.requestAuthorization(toShare: nil,
                    ↪ read: [sampleType]) { (didAsk, error) in
                    if let error = error {
                        print("oh␣no") // TODO handle error
                    } else {
                        if (self.query == nil) {
                            self.query = newQuery
                            healthStore.execute(newQuery)
                    }
                }
            }
        }
        else if (!newValue && query != nil) {
            healthStore.stop(query!)
            self.viewModel.model.removeQuantitySamples(
                ↪ ofType: self.sampleType)
            query = nil
        }
    }
}

private func queryResultsHandler(query:
    ↪ HKAnchoredObjectQuery, samplesOrNil: [HKSample]?,
    ↪ deletedObjectsOrNil: [HKDeletedObject]?, newAnchor:
    ↪  HKQueryAnchor?, errorOrNid: Error?) {
    guard let samples = samplesOrNil, let deletedObjects =
        ↪  deletedObjectsOrNil else {
        // TODO Properly handle the error.
        return
    }

    DispatchQueue.main.async {
        self.viewModel.model.addQuantitySamples(
            samples
                .compactMap { $0 as? HKQuantitySample }
                .map { QuantitySample(from: $0, quantityUnit
                    ↪ : self.unit) }
        )
    }
}

private func createQuery(type: HKSampleType, from fromDate
    ↪ : Date, to endDate: Date) -> HKQuery {
    let predicate = HKQuery.predicateForSamples(withStart:
        ↪  fromDate, end: endDate, options: .
        ↪ strictStartDate)
    let query = HKAnchoredObjectQuery(type: type,
                                      predicate: predicate,
                                      anchor: nil, // on app
                                          ↪ startup, get all
                                          ↪ data again. TODO
                                          ↪ for performance
                                          ↪ use an anchor
```

```
                                            limit:
                                                ↪ HKObjectQueryNoLimit
                                                ↪ ,
                                            resultsHandler:
                                                ↪ queryResultsHandler
                                                ↪ )
            query.updateHandler = queryResultsHandler

            return query
        }
    }
}
```

9.7 CustomSlider.swift

```
import SwiftUI

struct CustomSlider: View {

    @Binding var sliderValue: Double
    @State private var offset: Double = 0

    var body: some View {
        GeometryReader { geo in
            ZStack(alignment: .bottom){
                Color.clear
                Rectangle()
                    .foregroundColor(.accentColor)
                    .frame(height: geo.size.height * sliderValue)
            }
            .contentShape(Rectangle())
            .gesture(DragGesture(minimumDistance: .zero)
                        .onChanged({ value in
                let diff = value.translation.height - offset
                sliderValue -= (diff / geo.size.height)
                sliderValue = min(max(sliderValue, 0), 1)
                offset = value.translation.height
            })
                        .onEnded({ _ in
                offset = 0
            }))
            .animation(.easeInOut(duration: 0.1), value:
                ↪ sliderValue)
        }
    }
}

struct CustomSlider_PreviewWrapper: View {
    @State var value: Double
    var body: some View {
        CustomSlider(sliderValue: $value)
    }
}

struct CustomSlider_Previews: PreviewProvider {
    static var previews: some View {
        CustomSlider_PreviewWrapper(value: 0.3)
    }
}
```