1 Introduction

This thesis is a 15 HEC Master's thesis in Software Engineering. It explores the domain of mobile application development, wearable health technology, and data normalization. As healthcare becomes increasingly more digital, wearable devices like smartwatches and fitness trackers are gaining traction as tools for collecting physiological and behavioral data.

The widespread adoption and acceptance of such devices offers insightful possibilities for health monitoring, especially for conditions like migraine that could benefit from health monitoring. However, as of today that landscape is flawed due to data formats and APIs beeing differently interpreted depending on the different providers, making it difficult to integrate health data across platforms or use it effectively in research and machine learning. [1]

This thesis aims to investigate this challenge and contributes by exploring how a mobile software framework solution might help unify and normalize health data from various sources. The aim is to simplify data handling for researchers and developers while also hopefully enabling more effective use of health data in fields such as machine learning.

1.1 Background

As digital health technologies advance, the integration of wearable devices into everyday life has become increasingly common. The market for devices such as smartwatches, fitness trackers, and other biometric sensors is something that has seen rapid growth in recent years, with big stakeholders such as Alphabet Inc/Google, Apple inc, Garmin Ltd and Samsung Electronics Co. Ltd being some of the most prominent players. [2]. This has in turn led to a surge in the amount of possible health data being collected which has unlocked new potential within health research, disease identification and also potential treatment. [3]

One medical area where this type of data collection posibillities can have a high impact in is migraine detection and prevention. Migraines are complex neurological conditions often triggered by combinations of both physiological and environmental factors [4]. The ability to monitor physiological signals and important metrics that can potentially indiciate an impending migraine attack could be a game changer. Wearable devices can provide possibilities for monitoring these signals, allowing for early detection and prevention [5]. This would in turn lead to a road of benefits for both patients and healthcare providers, including improved quality of life, reduced healthcare costs, and more effective treatment plans.

However, while the capabilities of individual wearable devices are improving, the broader ecosystem remains limited. Each device manufacturer typically provides its own proprietary API and data format. This inconsistency makes it difficult for software developers and researchers to combine data from multiple sources, interpret it uniformly, and make something meaningful out of it.

One major challenge in this area is the lack of a unified framework for integrating wearable health data in a standardized way, especially within mobile environments. There are some existing solutions closely related to these issues but none of those provides a full solution. For example solutions such as the flutter health package and React Native Health [6], [7] are both libraries enabling access to health data, but none of the data is normalized or standardized. This means that the data collected is not in a format that can be easily used for machine learning or other data analysis tasks, where it is perfered to have structured data [8]. This lack of standardization poses a significant barrier to the effective use of wearable health data in both research and development.

1.2 Related work

In recent years, as the market for wearable devices has grown, so has the amount of research and development in this area. The current approaches to wearable data intergration similar to what is proposed in this thesis can be divided into three categories: web-based solutions, data processing frameworks and data extracting solutions. One noteable example is WearMerge [9], which was presented at IEEE International Conference on Pervasive Computing and Communications Workshops in 2022. WearMerge is a web-based approach which converts wearable data into Open Mhealth schemas. This solution offers similar data aggregation capabilities to our work but lacks the ability and support for mobile usage, which is highly relevant for potential wearable health monitoring.

Along similar research and works there is also Tasrif [10]. Tasrif is a Python based preprocessing framework for wearable data. It is designed to effectively handle and process data from widely used platforms such as Apple Health. More than just targeting bigger platforms, it also directly supports integration with machine learning libraries and focuses on offline data processing. Tasrif is a promising solution for data preprocessing purposes but just like WearMerge, it is not designed for mobile integration.

In addition to web and data processing frameworks, there are also several libraries for extracting data from wearable devices which is closely related to the work in this thesis. The solutions that exist are platform specific and are designed to work for their respective platforms. Some examples for these solutions for this are the flutter health package, [6], React Native Health [7] and React Native Health Connect [11]. These libraries are designed to work with specific platforms and they also lack the ability to normalize and standardize the data.

1.3 Problem formulation

While wearable devices have become increasingly capable of capturing health related metrics, there are still room for growth and gaps that can be filled. This is primarily due to the fragmented landscape of health data platforms, where most providers offer its own proprietary data format and API. As discussed in the previous sections, although some server based frameworks like Shimmer support multi-provider integration, they are not designed for mobile environments or real time use.

Existing solutions like WearMerge and Tasrif focus on data aggregation and preprocessing but either lack mobile compatibility or do not address data normalization. Furthermore, platform specific libraries such as React Native Health and Health Connect provide access to raw data but without standardization. This leaves a significant technical gap, which can be summarized as the lack of a unified, mobile based framework capable of collecting and normalizing wearable health data from multiple providers. This gap between single platform solutions and server based frameworks has been identified as a significant barrier in the context of big data healthcare solutions. [12].

To explore this gap, the thesis is structured by the following research questions:

1. How can wearable health data from different platforms be effectively normalized into a unified format suitable for migraine-focused machine learning applications?
2. What are the key requirements and challenges in implementing real-time data normalization for migraine-relevant wearable data in a mobile environment?
3. How effective is the Open mHealth schema as a standardized format for representing migraine-relevant physiological data from diverse wearable devices?

1.4 Motivation

From a scientific perspective, by unifying heterogeneous data from wearable devices into a common format, the thesis contributes to ongoing research in health data integration and machine learning. The findings could potentially support and enable effective data analysis and machine learning applications in the different fields of health research. By proposing a mobile based approach that supports real time data collection and normalization, the thesis aims to address the limitations of existing solutions and provide a more practical framework that can be used for more efficent research.

Additionally from a societal perspective, by enabling and effectively using wearable health data, wearables can be used to improve health monitoring and also be a tool for early detection and prevention of possible health issues. One of the areas that the thesis aims to focus on is migraine, which is a condition that can possibly be predicted and prevented due to the fact that migraine can be triggered by physiological and environmental factors [4]. By enabling these insights through wearable devices, the project can potentially support better health outcomes and quality of life for individuals suffering from migraines or any other predictable health conditions.

Furthermore, from an industry standpoint, the proposed framework provides a robust and reusable solution for developers and potential healthtech companies by building a framework which has built in support for multiplatform integration along with data normalization. This greatly contributes to effective development in the field of health data management and machine learning. The framework can be used as a foundation for future applications, enabling developers to focus on building their solutions rather than dealing with the complexities of data integration and normalization.

1.5 Results

None obtained

1.6 Scope/Limitation

Not finished

1.7 Target group

The primary target group are software developers and researchers working within the field of digital health, particularly those focusing on mobile application developments and wearable data integration. A challenge these stakeholders potentially face is working around the fragmented health data formats when developing solutions or preparing datasets for machine learning solutions.

Furthermore the framework can also be of interest for healthcare focused research teams aiming to collect and analyze physiological datapoints for conditions like migraine. By offering easier ways to access and normalize data across different platforms, the framework can support both development solutions but also research studies.

1.8 Outline

Not finished

2. Method

The applied method for this thesis is Design Science Research (DSR). DSR was chosen due to its natural fit for projects involved in creation of a software artifact. It is described by vom Brocke et al [13] as a paradigm that seeks to enhance knowledge through the creation of an innovative artifacts. They further explain that the aim of DSR is to generate knowledge about how things can and should be constructed, knowledge that is referred to as design knowledge within the DSR paradigm. Within the scope of this thesis, several research methods are used in combination. These include literature review to understand the theoretical background, artifact construction through iterative development and experimentation to evaluate the artifacts ability to meet the defined objectives. Together these methods form a multimethod approach aligned with the DSR process model.

2.1 Research project

The company Neurawave [14] presented a need for collecting health metrics that could be used for migraine prediction with machine learning. In this work, we follow the DSR methodology outlined by Peffers et al [15].

Phase 1 - Problem identification and insight gathering

We began our research project by conducting open-ended interviews with the founders of neurawave [14]. This approach helps us build a deeper understanding of both stakeholder needs and the problem domain. One of the key requirements was that the data collection mechanism had to be integrated into their existing cross-platform mobile application. The main reason for this requirement is the ability for the existing application to act upon realtime health data. Additionally, we conducted literature review to gather insights regarding the state of the art available solutions, and which health data is commonly used in migraine prediction.

Phase 2 - Objective formulation

Based on the analysis of identified requirements gathered through open-ended interviews and the research gap identified in the literature review, we defined the project's objectives. This involved a systematic consolidation of findings and stakeholder discussions to establish clear, actionable goals that would guide the subsequent design and development of the software artifact.

Phase 3 - Design and implementation

Guided by the formulated objectives, the software artifact, a data collection framework for mobile platforms, was designed and developed. The development process was iterative, characterized by cycles of design, prototyping, implementation, and stakeholder feedback. This approach allowed for progressive refinement of the framework's functionality and architecture, ensuring alignment with the project goals and evolving insights.

Phase 4 - Demonstrating efficacy

Upon completion of the data collection framework, its implemented functionality was subjected to a series of tests to demonstrate its capability in addressing the defined problem. These evaluations included functional, integration, and system testing to verify the core features and overall system performance. Subsequently, a collaborative assessment was conducted with the stakeholders. In this phase, the framework's achieved functionality and performance were compared against their initially defined goals and requirements to determine how well the objectives were met.

Phase 5 - Critical evaluation and feedback integration

The critical evaluation of the data collection framework was conducted through quantitative assessments using experiments. These experiments were designed to meticulously measure the framework's accuracy and reliability across various scenarios and use-cases. Specifically, the evaluations focused on the system's capability to effectively fetch and convert data, alongside its performance concerning execution time. To ensure a comprehensive assessment, these metrics were tested under different operational conditions, including scenarios for historical data retrieval and real-time data processing. The findings from this evaluation phase were then analyzed to identify potential areas for refinement.

Phase 6 - Dissemination and community engagement

This thesis serves as the primary vehicle for disseminating our research findings and their implications, aiming to engage with and contribute to the knowledge base of AI researchers and developers.

2.2 Research methods

We began our research project by conducting open-ended interviews with the founders of Neurawave [14]. Open-ended interviewing allows for a flexible, free-flowing conversation and is considered a qualitative method of data collection [16]. This method was chosen because the stakeholders possessed significantly more domain knowledge than us, making them essential in framing the problem accurately. As described by Alshenqeeti [16], the purpose of open-ended interviews is to "broaden the scope of understanding of investigated phenomena". We believed this approach would help us build a deeper understanding of both stakeholder needs and the problem domain.

Open-ended interviews fall within the broader category of qualitative methods, which are generally holistic and aimed at answering "what" questions. In contrast, structured interviews are more quantitative, relying on closed-ended questions, such as those with yes/no responses [16]. As Lakshman et al. [17] explain, quantitative methods focus on examining the effect of an independent variable on a dependent variable in a measurable, numerical way. However, at this early stage in the project, a qualitative approach was better suited to developing our understanding and framing our objectives.

Following the interviews, we conducted litterature research to expand our contextual understanding of the problem area. Literature research involves exploring a broad body of work related to a topic and is typically less rigid than a systematic literature review [18]. A systematic review, by contrast, targets a specific research question and is intended to gather empirical evidence to support conclusions. We chose a narrative-style literature review to ground our work theoretically and better understand the current landscape of data collection for health-related machine learning applications.

Alternative approaches, such as expert interviews could also have been employed to gather real-world insights. As described by Döringer [19], expert interviews are valuable for problem-centered exploration and knowledge gathering. However, due to the time and resource constraints, this method was not feasible within the scope of this thesis. Nonetheless, it represents a promising avenue for future research and validation.

During the design and development phase, we followed the artifact creation process outlined in the Design Science Research methodology. While this is not an empirical method in the classical sense, it is a core research activity in DSR, where existing knowledge and stakeholder input are synthesized into a functional solution. We began by defining application requirements that map directly to stakeholder requirements and then iteratively developed the plugin aimed at fulfilling those needs. No specific structured development framework as followed, as our team had prior experience in collaborative software development.

To evaluate the developed solution, we will use experimentation as the primary method of validating functionality. According to Basili et al. [20], experimentation is an iterative process of hypothesizing and testing. In our case, this involves defining (or redefining) software requirements, implementing them (the hypothesis), and verifying wether those requirements are fulfilled (the test).

Additionally, we will perform validation to ensure tha the plugin's software requirements align with the original stakeholder expectation [21]. As an alternative to experimentation, we considered survey-based research to gather stakeholder opinions on desired functionality. However, we ultimately decided against this due to our limited timeframe. We also believed that the iterative loop supported by experimentation would be hindered by the time required to create, distribute and analyze a survey. In our context, experimentation offers more immediate feedback and supports rapid iteration, which we viewed as essential for effective development.

2.3 Data for migraine prediction using machine learning

During the literature review of previous studies investigating migraine prediction using machine learning, we found only two studies [5], [22]. The health metrics used in both studies are outlined below. It is worth noting that several health metrics was included in either study, such as hours of working [22] out and step count [5]. We also found some studies [23], [24], [25] that examined different triggers (internal and external factors) such as weather, diet, hormonal changes that could be valuable in predicting migraine episodes, but none of those were included in the studies that combined migraine prediction with machine learning.

2.4 Wearables and datastores

We decided that the framework should target the native health data stores due to the fact that they provide a unified API for extracting health data that is not affected by the third party wearable. Since Android has 71.9% of the global market share and IoS having 27.68% [26], thus making Google Health Connect and Apple Health Kit the native data stores we mainly target. During the validation of the framework we will perform experiments using Apple Watch, Google Fitbit and Garmin smart watches as providers of health data to the native datastores. The selection of wearables are based on available resources and a reasonable coverage of the most common wearables.

Health metric	**Description**	**Refs**
Heart rate	The rate of heart beats per minute.	[5], [22]
Heart rate variability	Variation in time between heartbeats	[5]
Skin temperature	The temperature of the skin, preferably measured at finger or wrist	[5], [22]
Skin conductance	Measurement of how good the skin conducts electricity. Has been shown to increase due to sweating, as a response to stress	[5]
Respiratory rate	Measurement of the rate of breathing. Measured as breaths per minute.	[5]
Sleep time	Measurement the amount of hours slept	[5], [22]

We have decided to include heart rate and skin temperature in our data processing. The selection is based on a combination on what data is available from regular wearables and what health metrics have been shown in previous studies to be of high value.

2.5 Reliability and validity

Validity

One limitation related to validity stems from the scope of the evaluation. The developed component will be tested by two individuals in controlled, but limited, settings. As such, the generalizability of the findings to a broader user base or different use cases is uncertain. Furthermore, the component is validated exclusively on migraine related health data. While it may be adaptable to other domains within health monitoring, such applicability remains untested and therefore unknown.

Another validity concern involves the integration of the component with the datastores and/or wearable devices. The component will initially be tested using two or three specific health data providers. Its performance and compatibility with other, non integrated wearables remain unverified, and thus it cannot be assumed that the framework will function the equally well across other health data providers.

Additionally the component rely on third party API's for data collection. Any inaccuracies or failure in those APIs could directly compromise the integrity of the data and therefore the validity of the results. This is particularly concerning given that the quality and frequency of data vary significantly between high cost devices (e.g, Apple Watch, Fitbit) and low cost alternatives (e.g, Bangle.js, EmotiBit). The component might perform well in high quality data environments but underperform in cases where the input data is sparse, noisy or unreliable.

Reliability

A significant threat to reliability is the dynamic nature of third party wearable health data provider APIs. These APIs are frequently updated, and changes in their structure or functionality may break the integration with out framework. This implies that future researchers attempting to replicate this study might need to adapt the codebase to updated APIs. To mitigate this risk, we will document the integration process thoroughly and provide implementation guides to facilitate replication.

On the other hand, the controlled experiments conducted as a part of this project will be carefully designed and well documented. This will support the reproducibility of results and allow others to validate the findings under similar conditions.

Risks and mitigations strategies

A number of general risks may impact both the reliability and validity of the project outcomes:

- Lack of access to wearable data may hinder testing or validation of the component under real world conditions. To address this, we prioritize using devices with public or open APIs and ensure local caching of test data were possible.
- Complexity of data normalization and integration may result in inconsistent behavior across devices. This will be mitigated by adopting standardized data formats and implementing preprocessing checks.
- Variability in data completeness, or encountering unexpected data structures from providers (especially low-cost or experimental devices), could impact the reliability of data retrieval or the accuracy of data normalization processes. To address this, the framework implements robust error handling for data parsing and transformation, focusing on structural correctness during format conversion (e.g., to Open mHealth) and transparently reporting any encountered issues.
- Time management and scope creep pose risks to project completion. A well defined timeline and iterative planning approach are used to ensure focus and manage scope.
- Limited expertise in wearable APIs and data pipelines may slow progress. To mitigate this, we will rely on documentation and existing libraries where possible.
- Framework data outputs, though correctly formatted, generally require further user-side processing and feature engineering for optimal machine learning application. To address this, the framework's defined scope in data retrieval and format normalization, alongside its documented output data structures, enables users to effectively plan this subsequent data preparation.
- API rate limits and access restrictions may impact data collection throughput. Mitigation strategies include caching responses, adhering to API usage guidelines and contacting vendors if necessary.

By identifying these threats and planning accordingly, this project strives to maintain both reliability and validity, despite the challenges inherent in working with third-party hardware and health data.

2.6 Ethical considerations

While this project doest not involve direct interaction with human participants outside of the stakeholders in the initial interviews, it does involve collecting and processing of health related data via wearable devices. This introduces several ethical considerations, particular around privacy, data confidentiality and data ownership.

Confidentiality

The component is designed to collect health metrics that are sensitive by nature, such as heart rate and heart rate variability. Even though no personally identifiable information is intended to be handled, care must be taken to ensure that the processing of data is in accordance with privacy regulations such as GDPR.

To address this, all test data used during development and experimentation will be anonymized. No user names, contact information, or device IDs linked to individuals will be stored or processed in any way that could allow for re-identification. No data will be stored within the component, only processing of the data. No data will be shared externally or stored in the cloud.

Sampling bias

The testing of the component will be done with using a limited number of wearable devices and datasets related to migraine prediction. As a result, the dataset is narrow in scope which may introduce selection bias. This poses a limitation upon the generalizability of the results to other conditions such as other devices or populations. Since the primary objective is to develop and evaluate a working prototype in a focused context, this tradeoff is considered acceptable for the scope of the project.

Participation and consent

If any real user data is collected (for example, if Neurawave employees or external users voluntarily contribute data for evaluation), explicit informed consent will be obtained in written form. Participants will be made aware of what data is being collected for processing and how it will be used. Participation will be strictly voluntary, and participants will have the right to withdraw at any time without any consequence.

Risk of harm

This project poses minimal risk of harm. as it does not involve any physical or psychological intervention. However, improper handling of sensitive data could lead to privacy breaches. To minimize this risk, the component itself will not store any data.

3. Theoretical background

The current landscape of health data has seen a significant transformation due to the proliferation of wearable devices. Devices such as smartwatches, smart rings and fitness trackers have advanced rapidly in recent years, now offering high-quality, clinically certified data collection capabilities [1]. This longitudinal and increasingly accurate health monitoring opens up new possibilities in health research, disease detection and personalized treatment strategies [3]. The global wearable device market continues to grow at a substantial rate, with an estimated compound annual growth rate of 14.6% between 2023 and 2030 [2]. Leading industry actors include Alphabet Inc./Google, Apple Inc., Garmin Ltd., and Samsung Electronics Co., Ltd. [2].

While the growth of wearable technologies presents valuable opportunities, it also introduces significant technical and interoperability challenges. Each provider typically offers its own platforms for accessing and modifying health data, with unique APIs, data models and permission systems. This fragmentation complicates efforts to aggregate data from multiple providers. For instance, Apple's HealthKit provides a generalized abstraction for measurements via types such as HKQuantityType [27], representing quantities like step counts. In contrast, Google Health structures similar metrics as domain-specific records such as StepRecord [28], including associated metadata like start and end times. These inconsistensies in data modeling extend beyond simple metrics such as step counts to more complex physiological measurements such as heart rate variability, sleep stages and stress levels. The absence of standardized approaches for normalizing and integrating such heterogenous health data represents a significant obstacle for developers and researchers building cross platform applications, particulary for those leveraging machine learning techniques.

Existing frameworks for collecting health data

The existing frameworks for collecting health data are outlined below:

Framework	**Features**	**Missing features**
Health 12.0.1, Flutter package	Enables reading and writing health data to and from Apple health and Google Health Connect.	• Lacks support for several providers (Fitbit, garmin etc) • No support for Open mHealth data format.
React Native Health, React native package	Package for interacting with Apple HealthKit for iOS.	• Lacks support for several providers (e.g., Google Health connect). • No support for Open mHealth.
React native health connect, React native package	Package for interacting with Health Connect for Android.	• Lacks support for several providers (e.g Apple HealthKit for iOS) • No support for Open mHealth.
Shimmer, web platform	Application for extracting health data from multiple providers into Open mHealth data format.	• Is not natively supported on mobile.
Tasrif, python application	Application for extracting health data from multiple providers. Integrates with existing python ML libraries.	• Is not natively supported on mobile. • Does not support Open mHealth.

While each of these frameworks fulfills part of the requirements for a multi-provider health data integration, none currently provide a complete, mobile-native, cross-platform solution that supports standardized output such as Open mHealth format.

Health data standards

The current state of e-health data standards are outlined below:

Standard	Description	Reference
Fast Health Interoperability Resources - FHIR	Data standard for exchaning health care information digitally. Modular specification with focus on health care, with modules such as medications, diagnostics, etc.	[29]
Open mHealth	Data standard for mobile health data. Provides schemas for creating a uniform data structure for health data recorded by wearable devices.	[30]
IEEE P1752 - Standard for mobile health data working group	Provides data standard for representing physical activity, sleep and metadata.	[31]

Efforts toward standardizing mobile health data are ongoing. Babu et al. [3] highlight the importance of cross-organizational collaboration to enhance data quality, consistency and interoperability.

Health data stores and API

The plugin developed in this thesis targets two major health data stores: Apple HealthKit [32] on iOS and Google Health Connect [33] (formerly Google Fit) on Android. While both APIs offer similar capabilities for reading and writing data, they differ significantly in internal structure and terminology.

Access to these data stores requires explicit user permission, granted per-app and per-data-type, ensuring user privacy and conrol. Once permissions are granted, both platforms expose APIs for querying health data.

Google Health Connect uses specific record classes, such as Steps [34] or HeartRate [35], each containing metadata like startTime, endTime and a value field. In contrast, Apple HealthKit uses types such as HKQuantityType [36] or HKWorkoutType [37]. When requesting step data, for example, a HKQuantitySample [38] is returned containing the step count as an HKQuantity along with metadata like the measurement time window and data source.

Selected software

One requirement for this framework is that it must be cross-platform and mobile-native. Several development frameworks support this, including React Native [39], Flutter [40], LynxJS [41] and Kotlin Multiplatform [42].

Flutter was selected for this project due to existing infrastructure and developer experience within the organization (Neurawave). Flutter uses the Dart programming language, with native platform functionality implemented in Swift (iOS) and Kotlin (Android). Platform-specific functionality is accessed via MethodChannels, which allow Flutter code to call native APIs directly.

No additional third-party plugins will be used beyond what is required to interface with HealthKit and Health Connect.

Research gap and problem formulation

Despite the growing availability of health data APIs and frameworks, there is no existing mobile-native, cross-platform plugin capable of aggregating health data from multiple providers and exporting it in a standardized format such as Open mHealth. While it is theoretically possible to combine mutiple existing tools to achieve similar results, this approach is highly impractical and prone to compability issues, platform-specific limitations, and increased development complexity.

This fragmentation presents a significant barrier for developers and researchers who wish to build cross-platform health solutions or apply machine learning techniques to unified health datasets. The framework developed in this thesis aims to address this gap by offering a native, extensible solution for standardized mobile health data integration.

4 Research project - Implementation

4.1 Design and technique

We designed a Flutter plugin using the Dart programming language, with platform-specific implementations written in Swift for iOS and Kotlin for Android. The plugin exposes a public API that enables communication with the native health data stores. When a user invokes a method from this API, a MethodChannel is used internally to bridge the Dart code and the native platform code. The Dart layer is responsible for defining the public API, making method calls to the platform specific implementations, and performing data transformation. The native code handles permission requests and performs the actual data extraction.

The high level data flow is illustrated below:

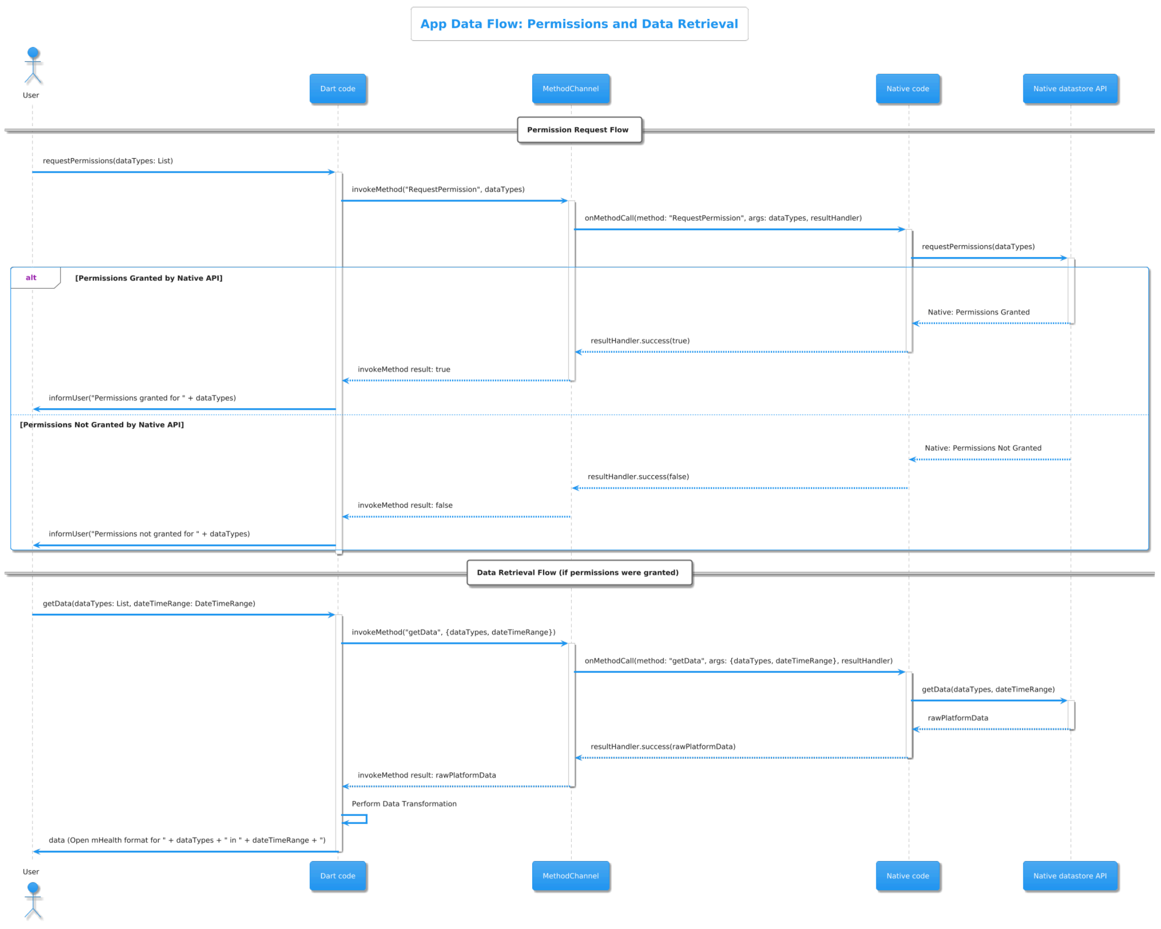
App Data Flow: Permissions and Data Retrieval
User
Dart code
MethodChannel
Native code
Native datastore API
Permission Request Flow
requestPermissions(dataTypes: List)
invokeMethod("RequestPermission", dataTypes)
onMethodCall(method: "RequestPermission", args: dataTypes, resultHandler)
requestPermissions(dataTypes)
alt
[Permissions Granted by Native API]
Native: Permissions Granted
resultHandler.success(true)
invokeMethod result: true
informUser("Permissions granted for " + dataTypes)
[Permissions Not Granted by Native API]
Native: Permissions Not Granted
resultHandler.success(false)
invokeMethod result: false
informUser("Permissions not granted for " + dataTypes)
Data Retrieval Flow (if permissions were granted)
getData(dataTypes: List, dateTimeRange: DateTimeRange)
invokeMethod("getData", {dataTypes, dateTimeRange})
onMethodCall(method: "getData", args: {dataTypes, dateTimeRange}, resultHandler)
getData(dataTypes, dateTimeRange)
rawPlatformData
resultHandler.success(rawPlatformData)
invokeMethod result: rawPlatformData
Perform Data Transformation
data (Open mHealth format for " + dataTypes + " in " + dateTimeRange + ")
User
Dart code
MethodChannel
Native code
Native datastore API

Figure 1: Data flow of requesting permissions and extracting health data

Several standardization models exist for structuring health data, including Open mHealth, FHIR and IEEE P1752. We chose Open mHealth due to its lightweight nature, JSON-compatibility and its specific focus on mobile health data. While FHIR is a powerful and widely adopted clinical standard, it introduces significant overhead for mobile use cases, requiring modeling of patients, medications and other elements irrelevant to this project [29]. Additionally, Open mHealth has begun incorporating elements from IEEE P1752, further reinforcing its suitability for mobile-focused applications.

This framework targets heart rate and heart rate variability data. On Android, it extracts HeartRate [35] and HeartRateVariabilityRmssd [43] records from Google Health Connect. On iOS, it extracts HKQuantitySample objects with type heartRate [44] and heartRateVariabilitySDNN [45]. The extracted data is transformed into the Open mHealth Heart Rate schema [46] and a custom heart rate variability schema modeled in accordance with the Open mHealth schema design principles [47] and template guidelines [48].

4.2 Implementation concerns

A primary objective of the plugin is to enhance interoperability across mobile health platforms. To achieve this, it provides functionality to normalize all extracted data into the Open mHealth format, regardless of the source platform or its underlying API. Metadata not explicitly defined within the Open mHealth schema, such as device-specific identifiers or detailed hardware information, is intentionally omitted during this process. This approach ensures data consistency and reduces variability when aggregating data from diverse sources.

Ensuring correct and user-transparent permission handling was a central, yet challenging, aspect of the implementation. While the plugin offers a unified API for requesting and managing data permissions, the native mechanisms provided by Apple's HealthKit and Google's Health Connect differ substantially. These differences presented considerable implementation hurdles. A significant divergence is HealthKit's privacy-centric design, which, unlike Health Connect, does not allow an application to programmatically determine the current status of read permissions for specific data types once a user has interacted with the initial permission prompt. Furthermore, to address operating system limitations that restrict applications from repeatedly requesting permissions, and to empower users with direct control, the plugin incorporates functionality to guide users to the relevant system settings page where they can manually adjust health data permissions for the application.

Data granularity and sampling resolution are determined by the user through a specified time window for each data query. The plugin respects the data resolution and any rate limits imposed by the underlying native health data store (HealthKit or Health Connect), retrieving data strictly within the user-defined timeframe.

The plugin also addresses scenarios where necessary platform components might be missing or prerequisites are not met. If a user attempts to utilize the plugin on an unsupported operating system version, they are provided with clear, user-facing feedback regarding the compatibility issue. Similarly, if a required health data store (like Health Connect) is not installed or accessible, the plugin detects this and informs the user of the situation, guiding them on the necessary prerequisites rather than attempting automated redirections.

Finally, the plugin is designed with extensibility as a core principle. Its modular architecture facilitates the straightforward integration of additional health data types, new data providers (e.g., specific third-party wearable APIs beyond native platform stores), and expanded health metric support in future development iterations.

4.3 Demonstration

The plugin will be demonstrated through a dedicated test application built in Flutter. This application implements the full data flow, including permission requests and data extraction. In addition, the application includes automated tests that validate the correctness of the data transformation and adherence to the Open mHealth schema.

The test application will be evaluated using real devices including Apple Watch, Fitbit and Garmin Venu to simulate real-world usage and ensure compatibility across a range of wearables. The demonstration will showcase both real-time and historical data extraction, within the constraints of each data stores retention policy and access limitations.

The plugin will be considered succesful if it consistently provides accurate and correctly formatted Open mHealth data on both iOS and Android platforms, across a diverse set of wearable devices.

5 Results

5.1 Resulting artifact

This section presents the 'wearable_health' Flutter plugin, the software artifact resulting from the design choices and implementation techniques detailed in Section 4.1. The developed plugin offers a Dart API designed to abstract the complexities of native platform interactions for managing health data permissions, extracting data from Apple HealthKit and Google Health Connect, and subsequently normalizing this data into the Open mHealth format.

Interaction with the plugin typically involves first obtaining a platform-specific interface object. For instance, methods on the main plugin instance allow developers to retrieve either an AppleHealthKit interface or a GoogleHealthConnect interface. These platform-specific interface objects then expose a largely consistent set of methods for data operations. The core functionalities available through these interfaces are detailed in Table 5.1.

API	Description	Platform(s)
checkPermissions	Retrieves a list of the health metrics that the user has given permissions to be read. Not implemented on HealthKit due to Apple's privacy policy.	HealthConnect
requestPermissions	Requests read permissions of provided data types.	HealthConnect & HealthKit
checkDataStoreAvailability	Checks the availability of underlying data store.	HealthConnect & HealthKit
redirectToPermissionsSettings	Provides a stable fallback for the request permissions functionality. Useful when the underlying os blocks requests due to several factors.	HealthConnect & HealthKit
getRawData	Retrieves serialized native data records of specified types and within specified time frame.	HealthConnect & HealthKit
getData	Retrieves data of specified types and within specified timeframe	HealthConnect & HealthKit

API	Description	Platform(s)
	as dart objects that are structurally equivalent to their native counterpart.	
getPlatformVersion	Retrieves the platform version of the device running the plugin.	HealthConnect & HealthKit

Table 5.1 - Description of the public API.

In addition to the direct data retrieval methods, the plugin includes Dart extension methods that facilitate the conversion of the retrieved Dart objects into the standardized Open mHealth format.

Internally, the plugin's software architecture employs a Model-Controller-Service pattern implemented independently within both the Dart layer and each native platform-specific layer. This modular design separates concerns for API interaction, business logic, and direct data source communication within their respective environments.

5.2 Experiment results

This section presents the empirical data obtained from the experiments conducted to evaluate the 'wearable_health' plugin. The primary aim of these experiments was to assess the plugin's core functionalities, data transformation accuracy, and operational performance.

All experiments were executed using the dedicated Flutter-based experiment application, as outlined in the demonstration plan in Section 4.3, and followed the evaluation approach described within the Design Science Research methodology.

The tests were performed across a variety of configurations, encompassing different mobile operating systems (iOS and Android), physical wearable devices (as specified in Section 4.3), target health data types (Heart Rate and Heart Rate Variability), for both historical data extraction and real time data extraction. The data presented in the following subsections are key findings derived from experimental logs and automatically generated test reports. These subsections will detail the results pertaining to data retrieval completeness, accuracy of data value conversion through the normalization process, and execution time for data retrieval and transformation.

5.2.1 Data retrieval completeness

=> Samlad tabell med experimentens resultat

5.2.2 Conversion accuracy

=> Samlad tabell med experimentens resultat

5.2.3 Execution time

=> Samlad tabell med experimentens resultat

6 Analysis

6.1 Analysis introduktion

This chapter presents a critical analysis of the empirical results from Chapter 5, which detailed the capabilities and performance of the 'wearable_health' plugin. The primary purpose of this analysis is to interpret these findings, thereby transforming the collected data into evidence and arguments that directly address the research questions formulated in Section 1.3 of this thesis.

The subsequent analysis is structured to systematically examine the results in relation to each research question. Key areas of focus will include the plugin's effectiveness in data normalization, the identified requirements and challenges pertinent to its implementation in a mobile environment, and the suitability of the Open mHealth schema for representing the targeted health data. Throughout this chapter, all interpretations will be rigorously grounded in the data presented in Chapter 5 to ensure objectivity and a clear line of reasoning.

6.2 Analysis of the artifacts core functionality and correctness

This analysis section evaluates the 'wearable_health' plugin's core capabilities—specifically, data extraction completeness and the accuracy of its data normalization processes. The interpretations are grounded in the experimental results presented for data retrieval (Section 5.2.1) and conversion accuracy (Section 5.2.2), aiming to provide insights relevant to research questions RQ1 (effective normalization) and RQ3 (Open mHealth effectiveness).

While the experimental data regarding data retrieval indicated that the plugin, within the tested scenarios, consistently fetched the expected volume of records, this analysis is confined to the specific devices, data types, and conditions evaluated. The observed reliability under these parameters provides positive initial evidence concerning the fundamental prerequisite of data availability for effective normalization (RQ1). However, the plugin's performance with untested data providers or under more adverse conditions remains unverified by this data, representing a limitation in generalizing this reliability.

Concerning data transformation and normalization accuracy, the experimental results indicated a high success rate in the correct conversion of data. This suggests the technical viability of the implemented strategies for the data encountered in testing. While this points to robust core logic, the process of defining and maintaining these transformations, particularly if extended to more complex or less consistently structured native data types beyond the scope of this study, could present ongoing challenges. The validated preservation of data values and structural conformance to the Open mHealth schema do, however, suggest that the implemented normalization strategies are operational as intended within the tested parameters.

The application of the Open mHealth schema to represent the targeted physiological data from the selected wearable platforms offers specific insights for RQ3. The observed accurate mapping of the tested native data structures to a unified Open mHealth representation, with no critical information loss detected during validation, suggests its applicability as a standard for these data types within this project's scope. It is important to note, however, that the necessity for a custom HRV schema, albeit based on Open mHealth principles, indicates that the core Open mHealth standard may not natively encompass all desired physiological metrics with sufficient out-of-the-box granularity. This implies a potential trade-off between adherence to a base standard and the need for domain-specific extensions. Nevertheless, achieving multi-platform normalization using this combined approach provides a tangible example pertinent to RQ1, illustrating one pathway by which such health data can be brought into a unified format.

In summary, while the experimental results within the tested configurations indicate dependable data extraction and accurate normalization capabilities (relevant to RQ1 and RQ3), this analysis also acknowledges that these findings are context-bound to the evaluated scenarios and chosen schemas. The observations provide foundational evidence for the plugin's core effectiveness, while concurrently highlighting areas of inherent complexity and potential limitations in achieving universal cross-platform data unification and standardization.

6.3 Analysis of the artifacts performance characteristics

This section analyzes the operational performance of the 'wearable_health' plugin, focusing on the execution times for data extraction and subsequent data conversion processes. The analysis is based on the quantitative performance metrics presented in Section 5.2.3 and is particularly important for understanding practical considerations pertinent to RQ2 (requirements and challenges for implementing near real-time data normalization).

The experimental results for data extraction (Section 5.2.3) indicated that for datasets comprising approximately !![1000 to 5000 records]!!, execution times were consistently observed in the vicinity of !![300 milliseconds across the tested platforms and devices]!!. The absence of significant outliers suggests a predictable performance profile within this operational range. While these figures may be considered acceptable for use cases involving single, discrete requests or periodic data synchronization, their implications for applications demanding true real-time continuous monitoring or very frequent polling (relevant to RQ2) warrant careful consideration. Such latency, though modest, could become a cumulative factor in resource consumption (e.g., battery) or perceived application responsiveness if invoked excessively. The predictability of this performance is, however, a positive aspect for developers anticipating system behavior for similar data loads.

Regarding the performance of the data conversion phase (transforming raw serialized data to Open mHealth), the measured execution times were consistently minimal, typically ranging from 0 to 3 milliseconds per dataset. This negligible processing overhead underscores the computational efficiency of the current normalization algorithms. This efficiency is a strong positive indicator for near real-time processing aspirations (RQ2), suggesting the normalization step itself adds minimal latency for the currently implemented transformations. It is acknowledged, however, that this high efficiency pertains to the specific data types and transformation logic of this project; more complex future normalizations could potentially introduce greater overhead.

In summary, the performance analysis provides a nuanced perspective. The data conversion step is demonstrably highly efficient for the current scope. The data extraction times, while consistent and predictable for the tested loads, present characteristics that necessitate careful evaluation by developers against the specific responsiveness and resource constraints of their target mobile applications, particularly those with stringent near real-time requirements (RQ2). These findings offer quantitative insights into the plugin's operational behavior within its evaluated parameters.

6.4 Analysis of Implemented Strategies for Design and Operational Concerns

This section analyzes how the 'wearable_health' plugin's final design and implemented features (Section 5.1), along with its observed functionalities (Section 5.2), relate to the key implementation concerns previously identified in Section 4.2. The aim is to critically examine the nature of the implemented strategies and their inherent trade-offs, offering insights relevant to RQ2.

A central concern involved permission handling, given the noted divergences between HealthKit and Health Connect. The resulting API addresses these platform-specific constraints, for example, by limiting the checkPermissions method to Health Connect due to HealthKit's privacy architecture. While the redirectToPermissionsSettings method provides a consistent fallback mechanism across platforms, this design inherently transfers a degree of responsibility for detailed permission verification and ongoing management to the end-user or the integrating developer. This outcome underscores a persistent challenge in achieving full operational parity and abstraction when developing cross-platform solutions for data access (RQ2).

Regarding interoperability and data standardization, the ‘wearable_health’ plugin provides developers with flexible data access options. As detailed in Section 5.1 and validated for its accuracy, the plugin can deliver data as raw native records, as Dart objects structurally mirroring the native data, or as data normalized to the Open mHealth format. The successful technical implementation of the Open mHealth normalization path demonstrates one viable strategy for achieving a standardized data output for the tested data types. When developers choose to utilize the Open mHealth formatted output, the plugin indeed applies the strategy of omitting non-standardized metadata to ensure a consistent, uniform data structure suitable for cross-platform aggregation. This specific transformation path thus offers a clear trade-off, available at the developer’s discretion: prioritizing the broad interoperability benefits of the Open mHealth standard, which may involve a reduction in granular, platform-unique metadata within that particular standardized output. Crucially, however, the plugin’s parallel capability to provide less abstracted data formats (raw or native-equivalent Dart objects) ensures that richer, device-specific contextual information remains accessible to the developer should their specific use case demand it. This multi-format approach is analyzed as a strength, offering a balance between strict standardization and access to detailed source data.

The approach to handling unavailable data stores or unsupported OS versions was implemented by informing the user of the specific issue. While this provides essential feedback and prevents unexpected application failures, the current implementation does not extend to offering alternative data sources or more advanced error recovery strategies. This level of error handling is analyzed as a foundational measure for stability, with more sophisticated responses remaining outside the defined scope of this project.

Finally, while extensibility was a stated design goal, the analysis of the implemented MCS architecture primarily indicates its theoretical suitability for future expansion due to its inherent modularity. The practical ease and efficiency of integrating entirely new data types or third-party providers were not empirically validated within the project’s timeframe. Thus, while the chosen architecture aligns with design principles that generally support extensibility, this attribute is more accurately characterized as a potential for future development rather than a demonstrated outcome of the current artifact.

In summary, this analysis of the implemented strategies for predefined concerns reveals that the ‘wearable_health’ plugin embodies a series of pragmatic technical decisions and inherent trade-offs. While core functionalities were realized, the approaches taken also serve to highlight the persistent complexities and limitations faced when creating standardized abstraction layers over disparate native systems. These observations contribute to a nuanced understanding of the requirements and challenges (RQ2) in developing such cross-platform mobile health data frameworks.

6.5 Summary

The analysis concerning RQ1 (effective data normalization) suggests that the plugin, through its implemented strategies, offers a feasible pathway to normalizing health data from the tested disparate mobile platforms into a unified format (Open mHealth). While dependable data extraction and accurate normalization were observed within the defined experimental scope, the identified need for project-specific schema extensions (e.g., for HRV) indicates that achieving a truly “unified” representation often necessitates careful adaptation and extension of existing standards to capture specific data nuances.

Regarding RQ3 (effectiveness of Open mHealth), the analysis indicates that the Open mHealth schema, when appropriately complemented, demonstrated practical applicability for representing the targeted physiological data from the evaluated sources. Its adoption facilitated a consistent data structure, a key aspect of its effectiveness. However, this specific standardization path, if exclusively used, involves a trade-off regarding the richness of available metadata, although the plugin mitigates this by also offering access to less abstracted data formats.

The investigation pertinent to RQ2 (requirements, challenges, and performance) revealed a multifaceted picture. While data conversion processes proved highly efficient, the observed data extraction times, though consistent for the tested loads, call for careful consideration in contexts with stringent real-time or high-frequency polling demands. The implemented solutions to various design and operational concerns (such as permission handling and error management) were found to be functional responses to complex cross-platform realities, often embodying pragmatic trade-offs rather than absolute solutions. Furthermore, attributes like architectural extensibility, while supported by design principles, remain largely theoretical pending further empirical validation.

These arguments and evidence highlights both the functional capabilities of the developed 'wearable_health' plugin within its specific operational context and the inherent complexities, limitations, and design trade-offs encountered.

7 Discussion

The overarching purpose of this thesis, as outlined in Section 1.3, was to explore key aspects of designing and implementing a cross-platform framework for mobile health data collection and normalization, exemplified by the 'wearable_health' plugin. The preceding chapters have detailed the methodology employed (Chapter 2), the design and implementation of the artifact (Chapter 4), the empirical results obtained from its evaluation (Chapter 5), and a focused analysis of these results in relation to the initial research questions (Chapter 6).

Regarding RQ1 – How can wearable health data from different platforms be effectively normalized into a unified format suitable for migraine-focused machine learning applications? – the development and subsequent analysis of the 'wearable_health' plugin demonstrated a viable pathway. The study indicated that a Flutter-based plugin, employing platform-specific native modules and a clearly defined data transformation pipeline, can effectively abstract complexities of disparate platform APIs for the purpose of data extraction. Furthermore, the implemented strategies for normalizing this data to the Open mHealth standard, including the judicious use of custom schema extensions where the base standard was found insufficient, proved to be technically sound for the tested data types. This suggests that achieving "effective normalization" for such specialized applications often involves a pragmatic combination of leveraging existing standards while allowing for tailored adaptations to capture necessary data nuances, thereby providing a structured, though not universally applicable without consideration for such extensions, input for potential ML pipelines.

The investigation into RQ2 – What are the key requirements and challenges in implementing real-time data normalization for migraine-relevant wearable data in a mobile environment? – illuminated several critical aspects based on the analyses in Sections 6.3 and 6.4. Key requirements identified through this DSR project include robust mechanisms for managing diverse and restrictive platform-specific permission models, strategies for handling API limitations and potential data inconsistencies from sources, and computationally efficient data processing logic suitable for mobile resource constraints. The primary challenges analyzed encompass the inherent difficulties in creating a truly seamless abstraction layer over fundamentally different native systems, the trade-offs between achieving a high degree of standardization versus retaining rich, platform-specific metadata, and ensuring predictable performance. While the plugin's data conversion stage was found to be highly efficient, the data extraction times (approximately !![300 milliseconds for 1000-5000 records]!!) suggest that while suitable for many intermittent mobile use cases, careful architectural consideration regarding data polling frequency and volume is essential for applications with stringent near real-time demands.

In addressing RQ3 – How effective is the Open mHealth schema as a standardized format for representing migraine-relevant physiological data from diverse wearable devices? – the analysis suggests that the Open mHealth schema, when augmented by necessary, context-specific extensions (such as the custom HRV schema developed in this project), can serve as an effective foundation for representing the targeted physiological data from the evaluated sources. Its application facilitated the creation of a consistent data structure, which is vital for unified datasets. However, the identified need for such an extension for HRV also implies that the base Open mHealth standard, in its current iteration, may not comprehensively cater to all granularities or types of migraine-relevant physiological data without such adaptation. Thus, its "effectiveness" is determined to be context-dependent, often relying on the schema's inherent flexibility to be extended in accordance with its design principles, while users of the plugin retain access to less abstracted data formats if the Open mHealth trade-offs are unsuitable for their specific needs.

The initial problem formulation highlighted a significant knowledge gap: the lack of a unified, mobile-based framework capable of both collecting and subsequently normalizing wearable health data from diverse providers for further use, particularly in contexts like machine learning. While existing mobile libraries such as Flutter Health and React Native Health offer valuable foundational access to native health data stores, they typically do not provide built-in data normalization or standardization capabilities. The analysis of the 'wearable_health' plugin's core functionality indicated its technical feasibility in not only extracting data from both Apple HealthKit and Google Health Connect but also in transforming this data into the Open mHealth standard directly within the mobile environment. This positions the developed plugin as a targeted response to the identified gap, offering a more integrated solution by embedding the crucial standardization step, which is often a prerequisite for interoperable health data analysis, within the mobile application layer. Although the current implementation of 'wearable_health' is focused on specific data types, its successful operation provides an empirical demonstration of this mobile-centric, normalizing approach, differentiating it from libraries that solely focus on raw data extraction.

The review of related work identified frameworks such as WearMerge and Tasrif, which provide sophisticated data aggregation and preprocessing functionalities but are primarily designed for web-based or offline, non-mobile environments. WearMerge, for example, shares the use of Open mHealth schemas with this project but lacks direct mobile integration. The 'wearable_health' plugin, while perhaps not currently possessing the same breadth of data aggregation features as a mature solution like WearMerge, distinguishes itself through its native mobile architecture, thereby addressing the specific limitation of "lack of mobile usage support" noted for such systems. This makes it potentially more suitable for applications requiring on-device processing or more frequent, near real-time data interactions.

Similarly, Tasrif offers a robust Python-based environment for extensive offline data preprocessing and integration with machine learning libraries. The 'wearable_health' plugin does not seek to replicate these advanced analytical capabilities. Instead, its contribution is positioned upstream, within the mobile data capture phase. By collecting and standardizing data directly on the mobile device, 'wearable_health' can provide a prepared data output that could subsequently be ingested by more specialized backend frameworks like Tasrif or utilized in other data analysis pipelines. The plugin's provision of multiple output formats (raw, native-equivalent Dart objects, and Open mHealth) offers developers the flexibility to select the appropriate level of data abstraction for their specific needs, potentially bridging the gap between mobile data collection and more intensive offline analysis. Consequently, 'wearable_health' can be viewed as a complementary tool that addresses a distinct niche not directly covered by these existing, predominantly server-side or desktop-oriented frameworks.

The decision to adopt the Open mHealth standard was informed by its documented suitability for mobile health data. The analysis of its effectiveness within this project indicated its general applicability for representing the targeted physiological data. A key finding, however, was the necessity of developing a custom Heart Rate Variability schema, albeit one designed in accordance with Open mHealth principles. This experience suggests that while Open mHealth provides a valuable and lightweight foundational standard for mobile contexts, its base schemas may not always offer sufficient granularity or specific structures for all types of specialized physiological data without requiring such extensions. This observation aligns with the broader discourse in health informatics regarding the inherent tension between universal standardization and the need for domain-specific detail, and it contributes a practical data point to the understanding of Open mHealth's application in real-world mobile projects.

The development journey of 'wearable_health' empirically confirmed several challenges inherent in cross-platform mobile access to health data, a theme reflected in the initial background discussion on data fragmentation and further detailed in the analysis of implementation concerns. The marked differences in permission models, API functionalities, and data structures between Apple's HealthKit and Google's Health Connect necessitated significant platform-specific logic within the plugin's native layers. Although the plugin's Dart API aims to provide a unified interface, the analysis acknowledged that certain platform-specific constraints (such as HealthKit's opacity regarding read permission status) inevitably impact the level of achievable abstraction and may require developers using the plugin to implement platform-aware strategies. This project's experiences therefore provide a concrete illustration of the design trade-offs and implementation complexities involved in striving for seamless cross-platform functionality in the sensitive and technically diverse domain of mobile health data.

To accurately contextualize the findings and conclusions of this project, it is essential to acknowledge its inherent limitations. Firstly, the scope of the evaluation for the 'wearable_health' plugin was intentionally focused on a core set of health data types. Testing was conducted using a specific selection of wearable devices under controlled conditions, primarily by the development team. Consequently, the generalizability of the plugin's observed performance, reliability, and robustness to other data types, a broader array of wearable hardware, or diverse end-user populations and real-world conditions remains largely unverified by this study.

Secondly, the plugin's functionality is intrinsically dependent on the Application Programming Interfaces (APIs) of third-party native health data stores. These APIs and their underlying platforms are dynamic ecosystems subject to updates and modifications by their respective proprietors. Such changes could potentially impact the plugin's future compatibility and long-term operational stability, possibly necessitating code adaptations to maintain functionality. While the plugin demonstrated compatibility with the API versions available during the project's timeframe, this reliance on external entities introduces an element of dependency and potential future maintenance overhead.

Finally, the developed 'wearable_health' plugin should be regarded as a research prototype. Although it successfully demonstrates the core functionalities of cross-platform data extraction and normalization as designed, it has not undergone the extensive, rigorous testing, comprehensive error handling development, or performance hardening typically required for a production-grade software component intended for deployment in critical or large-scale applications.

The findings and identified limitations of this study naturally suggest several promising avenues for future work and research. A primary direction involves the continued enhancement and maturation of the ‘wearable_health’ plugin itself. This could entail expanding its support to a wider range of health data types pertinent to migraine research or other conditions, integrating direct API support for additional third-party wearable ecosystems (beyond relying solely on native platform data stores), and further refining its error handling, data validation protocols, and security considerations. Empirically validating the plugin’s designed extensibility by systematically incorporating new data types or provider integrations would also constitute a valuable subsequent investigation.

Beyond direct artifact development, further research should focus on evaluating the practical impact and utility of the standardized data produced by the plugin within real-world application scenarios. Integrating ‘wearable_health’ into a full-scale data collection pipeline for a clinical or applied research study, such as Neurawave’s proposed migraine prediction initiative, would provide crucial evidence of its efficacy in preparing data for complex machine learning models and its overall value in the data-to-insight lifecycle. Comparative studies could also be undertaken to analyze the trade-offs between the Open mHealth normalization pathway facilitated by this plugin and alternative raw data or different standardization approaches, particularly concerning their influence on subsequent analytical outcomes or ML model performance.

Finally, the broader challenges encountered in this project highlight persistent areas for ongoing research in the field of mobile health informatics. This includes developing more adaptive and resilient strategies for managing evolving cross-platform permission models and data access restrictions, creating robust methods for handling highly asynchronous, sparse, or intermittently available data streams from diverse wearables, and contributing to the continued evolution and adoption of interoperable data standards like Open mHealth to better accommodate emerging data types and complex use cases in personalized and preventative health.

8 Conclusion

The primary outcome, the ‘wearable_health’ plugin, demonstrated a feasible approach to RQ1 (effective normalization). It successfully unified Heart Rate and HRV data from HealthKit and Health Connect into the Open mHealth standard, although this highlighted that achieving optimal data representation often requires context-specific schema adaptations, such as the custom HRV extension developed herein. Regarding RQ2 (requirements and challenges), the project underscored critical needs for managing disparate platform APIs and permission models. While data conversion proved highly efficient (0-3ms), extraction times (approximately !![300 milliseconds for 1000-5000 records]!!) were identified as a factor requiring careful consideration for applications with stringent (near) real-time demands in mobile environments. For RQ3 (Open mHealth effectiveness), the schema, when appropriately augmented, was found applicable for the targeted data. However, this analysis also confirmed a trade-off between the consistency afforded by standardization and the richness of platform-specific metadata, a choice the plugin accommodates by offering multiple data output formats.

This work contributes to bridging the identified knowledge gap for a unified, mobile-centric normalization framework, with relevance for both Design Science Research (through the artifact and generated design knowledge) and industry (providing a practical tool for developers like Neurawave). While the plugin’s architectural principles may offer some generalizability, its specific performance and API effectiveness are necessarily context-bound to the tested mobile platforms and evolving third-party APIs.

Critically reflecting on the project, a broader evaluation scope, encompassing more diverse wearable devices and more rigorously defined real-time scenarios, alongside direct usability testing of the plugin's API, could have yielded more comprehensive insights. Furthermore, an even more exhaustive exploration of standard Open mHealth modeling capabilities before creating custom extensions might have revealed alternative representation strategies.

These reflections and the project's inherent scope limitations naturally point to several avenues for future work. Key directions include enhancing the 'wearable_health' plugin with support for additional data types and direct third-party API integrations, and empirically validating its designed extensibility. Further research into resilient abstraction techniques against dynamic native APIs, and the impact of various data normalization levels on machine learning model performance, also presents as valuable. Crucially, integrating and evaluating the plugin within a real-world study, such as Neurawave's migraine prediction initiative, is essential for definitive validation of its practical utility.

In conclusion, this thesis has delivered a functional software artifact, 'wearable_health,' and has generated significant design knowledge regarding the complexities of cross-platform mobile health data integration. Despite acknowledged limitations, the plugin and the insights derived from its development represent a tangible contribution, offering a foundation for future research and development aimed at fostering more interoperable and intelligent mobile health solutions.

Bibliography

[1] Sophie Huhn *et al.*, "The Impact of Wearable Technologies in Health Research: Scoping Review," *JMIR Mhealth Uhealth*, vol. 10, no. 1, Jan. 2022.

[2] Grand View Research, "Wearable Technology Market Size, Share & Trends Analysis Report By Product (Head & Eyewear, Wristwear), By Application (Consumer Electronics, Healthcare), By Region (Asia Pacific, Europe), And Segment Forecasts, 2023 - 2030," 978–1–68038–165–8.

[3] Mohan Babu, Ziv Lautman, Xiangping Lin, Milan H B Sobota, and Michael P Snyder, "Wearable Devices: Implications for Precision Medicine and the Future of Health Care," *Annual Review of Medicine*, vol. 1, no. 1, Nov. 2023.

[4] Hassan Kesserwani, "Migraine Triggers: An Overview of the Pharmacology, Biochemistry, Atmospherics, and Their Effects on Neural Networks," *Cureus*, vol. 13, no. 4, Apr. 2021.

[5] Viroslava Kapustynska, Vytautas Abromavičius, Artūras Serackis, Šarūnas Paulikas, Kristina Ryliškienė, and Saulius Andruškevičius, "Machine Learning and Wearable Technology: Monitoring Changes in Biomedical Signal Patterns during Pre-Migraine Nights," *Healthcare (Basel)*, vol. 12, no. 17, Aug. 2024, doi: https://doi.org/10.3390/healthcare12171701.

[6] "Health Package." [Online]. Available: https://pub.dev/packages/health

[7] "React Native Health." [Online]. Available: https://www.npmjs.com/package/react-native-health

[8] Maryam Tayefi *et al.*, "Challenges and opportunities beyond structured data in analysis of electronic health records," *WIREs - Wiley Interdisciplinary Reviews*, vol. 13, no. 6, Feb. 2021, doi: 10.1002/wics.1549.

[9] Dimitrios Panteleimon Giakatos, Sofia Yfantidou, Stefanos Efstathiou, and Athena Vakali, "WearMerge: An Interoperable Framework for Self-tracking Data Integration and Standardization," May 06, 2022, *IEEE*. doi: 10.1109/PerComWorkshops53856.2022.9767462.

[10] Abdulaziz Al Homaid, Syed Hashim, Fadhil Abubaker, Ummar Abbas, Faisal Farooq, and Joao Palotti, "Tasrif: processing wearable data in Python," May 06, 2022, *IEEE*. doi: 10.1109/PerComWorkshops53856.2022.9767286.

[11] "React Native Health Connect." [Online]. Available: https://www.npmjs.com/package/react-native-health-connect

[12] Sumarsono, Muhammad Anshari, and Mohammad Nabil Almunawar, "Big Data in Healthcare for Personalization & Customization of Healthcare Services," Sep. 19, 2019, *IEEE*. doi: 10.1109/ICIMTech.2019.8843822.

[13] in *Design Science Research Cases*, 2020, pp. 1–13.

[14] Neurawave, "Neurawave's website." [Online]. Available: https://www.neurawave.se/

[15] Ken Peffers, Tuure Tuunanen, Marcus A. Rothenberger, and Samir Chatterjee, "A Design Science Research Methodology for Information Systems Research," *Journal of Management Information Systems*, vol. 24, no. 3, 2007, doi: 10.2753/MIS0742-1222240302.

[16] Hamza Alshenqeeti, *English Linguistics Research*, vol. 3, no. 1, Mar. 2014, doi: 10.5430/elr.v3n1p39.

[17] M. Lakshman, Leena Sinha, Moumita Biswas, Maryann Charles, and N.K. Arora, "Quantitative vs Qualitative research methods," *The Indian Journal of Pediatrics*, vol. 67, no. 1, May 2000, doi: https://doi.org/10.1007/BF02820690.

[18] Priscilla Robinson and John Lowe, "Literature reviews vs systematic reviews," *Australian and New Zealand Journal of Public Health*, vol. 39, no. 2, Apr. 2015, doi: 10.1111/1753-6405.12393.

[19] Stefanie Döringer, "The problem-centered expert interview. Combining qualitative interviewing approaches for investigating implicit expert knowledge," *International Journal of Social Research Methodology*, vol. 24, no. 3, May 2020, doi: 10.1080/13645579.2020.1766777.

[20] Victor R. Basili, Richard W. Selby, and David H. Hutchens, "Experimentation in software engineering," *IEEE Transactions on Software Engineering*, vol. 12, no. 7, Sep. 2012, doi: 10.1109/TSE.1986.6312975.

[21] P. Sujatha, G. V. Sankar, A. S. Rao, and T. Satyanarayana, "The Role of Software Verification and Validation in Software Development Process," *IETE Technical Review*, vol. 18, no. 1, Mar. 2015, doi: 10.1080/02564602.2001.11416938.

[22] Anker Stubberud *et al.*, "Forecasting migraine with machine learning based on mobile phone diary and wearable data," *Cephalgia*, vol. 43, no. 5, May 2023, doi: https://doi.org/10.1177/03331024231169244.

[23] Antonio L Aguilar-Shea, Javier A Membrilla Md, and Javier Diaz-de-Teran, "Migraine review for general practice," *Aten Primaria*, vol. 54, no. 2, Nov. 2021, doi: 10.1016/j.aprim.2021.102208.

[24] Michael J Marmura, "Triggers, Protectors, and Predictors in Episodic Migraine," *Current pain and headache reports*, vol. 12, no. 81, Oct. 2018, doi: 10.1007/s11916-018-0734-0.

[25] Dana P Turner, Adriana D Lebowitz, Ivana Chtay, and Timothy T Houle, "Forecasting Migraine Attacks and the Utility of Identifying Triggers," *Current pain and headache reports*, vol. 22, no. 9, Jul. 2018, doi: 10.1007/s11916-018-0715-3.

[26] Statcounter, "Mobile os market share." [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide

[27] [Online]. Available: https://developer.apple.com/documentation/healthkit/hkquantitytype

[28] [Online]. Available: https://developer.android.com/reference/kotlin/androidx/health/connect/client/records/StepsRecord

[29] [Online]. Available: https://hl7.org/fhir/

[30] [Online]. Available: https://www.openmhealth.org/

[31] [Online]. Available: https://standards.ieee.org/ieee/1752.1/6982/

[32] [Online]. Available: https://developer.apple.com/documentation/healthkit

[33] [Online]. Available: https://developer.android.com/health-and-fitness/guides/health-connect

[34] [Online]. Available: https://developer.android.com/reference/kotlin/androidx/health/connect/client/records/StepsRecord

[35] [Online]. Available: https://developer.android.com/reference/kotlin/androidx/health/connect/client/records/HeartRateRecord

[36] [Online]. Available: https://developer.apple.com/documentation/healthkit/hkquantitytype

[37] [Online]. Available: https://developer.apple.com/documentation/healthkit/hkworkouttype

[38] [Online]. Available: https://developer.apple.com/documentation/healthkit/hkquantitysample

[39] [Online]. Available: https://reactnative.dev/

[40] [Online]. Available: https://flutter.dev/

[41] [Online]. Available: https://lynxjs.org/

[42] [Online]. Available: https://kotlinlang.org/docs/multiplatform.html

[43] [Online]. Available: https://developer.android.com/reference/kotlin/androidx/health/connect/client/records/HeartRateVariabilityRmssdRecord

[44] [Online]. Available: https://developer.apple.com/documentation/healthkit/hkquantitytypeidentifier/heartrate

[45] [Online]. Available: https://developer.apple.com/documentation/healthkit/hkquantitytypeidentifier/2881127-heartratevariabilitysdnn

[46] [Online]. Available: https://www.openmhealth.org/documentation/#/schema-docs/schema-library/schemas/omh_heart-rate

[47] [Online]. Available: https://www.openmhealth.org/documentation/#/schema-docs/schema-design-principles

[48] [Online]. Available: https://www.openmhealth.org/documentation/#/schema-docs/write-a-schema