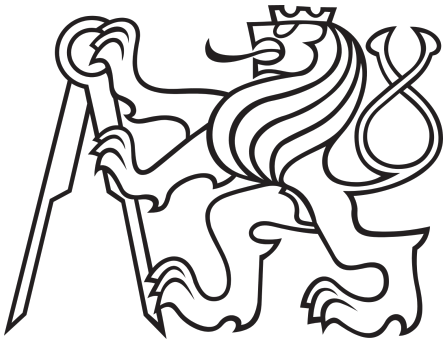
Platforma pro agregaci zdravotních a fitness dat z různých existujících platforem

Platform for agreagation of health and fitness data from different platforms

Bakalářská práce



ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická
Softwarové inženýrství a technologie
Katedra počítačů
21. května 2021

Duc Anh Le
Vedoucí práce: Ing. Martin Mudra

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Le** Jméno: **Duc Anh** Osobní číslo: **486997**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Platforma pro agregaci zdravotních a fitness dat z různých existujících platforem

Název bakalářské práce anglicky:

Platform for agreagation of health and fitness data from different platforms

Pokyny pro vypracování:

Analyzujte dostupné API či SDK pro následující platformy: Samsung Shealth,Apple Healthkit, Google Fit, Fitbit. Popište architekturu a dostupné možnosti synchronizace a získávání dat z těchto platforem. Analyzujte technologické prerekvizity pro vytvoření
aplikace pro synchronizaci těchto dat do vlastní serverové infrastruktury.
Navrhněte vhodnou rozšiřitelnou formu persistence dat podporující snadné
přidávání dostupných metrik. Navrhněte a implementujte serverovou aplikaci umožňující persistenci dostupných zdravotních a fitness metrik v rámci jedné databáze. Navrhněte systém pro ukládání obecných struktur dostupná data z výše zmíněných platforem. Do implementované platformy synchronizujte data z následujících platforem:
Samsung Shealth, Google Fit, Fitbit.
Pro synchronizaci dat z platformy SHealth vytvořte jednoduchou mobilní aplikaci pro posílání dat do implementované platformy.
Pro platformu vytvořte webovou aplikaci poskytující UI k dostupným uloženým datům. Visualizujte dostupná jednotlivých metrik pomocí grafů. Platformu otestujte proti reálnému napojení na výše zmíněné platformy. Pro účely testování umožněte také
zadávání jednotlivých dat napřímo z UI. Otestujte zejména chování platformy s větší množinout dat (řádově ~1000 záznamů pro každou zaznamenávanou metriku).

Seznam doporučené literatury:

[1] Oficiální dokumentace k Apple Healthkit
https://developer.apple.com/documentation/healthkit
[2] Oficiální dokumentace SamsungHealth,
https://developer.samsung.com/health/android/overview.html
[3] Google Fit SDK https://developers.google.com/fit/rest/v1/authorization
[4] Fitbit sdk https://dev.fitbit.com/build/reference/web-api/basics/
[5] Xamarin binding Apple healthkit https://docs.microsoft.com/cs-cz/xamarin/ios/platform/healthkit

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Martin Mudra, Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **11.02.2021** Termín odevzdání bakalářské práce: **21.05.2021**

Platnost zadání bakalářské práce: **30.09.2022**

Ing. Martin Mudra podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Acknowledgements

I would like to express my gratitude to my supervisor, Ing. Martin Mudra, for his valuable advice, patience, and guidaince.

Chtěl bych poděkovat svému vedoucímu práce Ing. Martinovi Mudrovi za cenné rady, trpělivost a odborné vedení.

Declaration

I declare that I carried out this bachelor thesis with the topic “Platform for agreagation of health and fitness data from different platforms” independently in accordance with the Methodical guideline No. 1/20, and only with the cited sources, literature and other professional sources.

Prohlašuji, že jsem práci na téma “Platforma pro agregaci zdravotních a fitness dat z různých existujících platforem” vypracoval samostatně v souladu s Metodickým pokynem č. 1/20 s použitím odborné literatury a pramenů.

Abstract

This bachelor thesis focuses on health and fitness aggregation from different existing platforms. This thesis aims to analyze available synchronization options, and design a suitable system for synchronizing data from these platforms into a server infrastructure.

Tato bakalářská práce se zabývá agregací zdravotních dat a fitness metrik z různých existujících platforem. Cílem této práce je analyzovat dostupné možnosti synchronizace dat a navrhnout vhodný systém pro synchronizaci těchto dat do vlastní serverové infrastruktury.

Keywords

Health data aggregation, Data synchronization, Data visualization, Extensible system

Agregace zdravotních dat, Synchronizace dat, Vizualizace dat, Rozšiřitelný systém

Contents

List of Figures

Chapter 1

Introduction

Smartwatches and fitness trackers are devices, which can measure and record a variety of metrics. These metrics include, for example, heart rate, blood pressure, water intake, sleep quality, or step count.

Smartwatches and fitness trackers can be used, for example, by athletes who want to monitor their body during training sessions, or by patients who suffer from low or high blood pressure, and need to constantly monitor their heart rate. The users can wear multiple devices from multiple manufacturers at the same time, for example, due to one device being more accurate for measuring a certain metric than the other, and vice versa.

These data are stored in the devices, or on external servers, and this thesis aims to analyze multiple platforms, look at how these data can be retrieved, and then design and implement an extensible system for storing these health and fitness data from multiple platforms.

Chapter 2

Analysis

2.1 Platform Analysis

The purpose of this analysis is to determine whether health data on specific platforms are accessible, and what methods can be used to access them.

Platforms to be analyzed are:

- Google Fit
- Fitbit
- Apple Health
- Samsung Health

Google Fit and Fitbit use the OAuth 2.0 protocol for authentication and access authorization [3, 5].

Samsung provides SDKs that allow data access using the Samsung Health application [11], and the Samsung Health Server [12].

Apple provides a framework using which data can be requested from a central repository using queries and methods provided by the framework [6].

2.1.1 Protocol OAuth 2.0

OAuth 2.0 is an authorization protocol that provides third-party application access to restricted resources, either on behalf of the resource owner or an application itself [1]. OAuth 2.0 an internet standard that is documented by the *RFC 6749 - The OAuth 2.0 Authorization Framework* [1].

Roles

The protocol defines four roles that participate in the authorization process [1]:

- The resource owner
- The resource server
- The client
- The authorization server

The resource owner is an entity that can allow a third-party application to access their protected resources [1]. The entity is in control of the resources [1], and represents, for example, a person or an organization.

The authorization server handles resource owner authentication and access token issuance to third-party applications [1].

The client is a third party that requests access to the resource owner's protected resources. Access must be granted by both the resource owner and the authorization server [1].

The resource server stores protected resources, and handles incoming authenticated requests [1].

Authorization Grant

An authorization grant is a credential that is used by the client to obtain an access token from the authorization server [1]. Access to resources is limited by the scope that was specified by the resource owner during the authorization [1]. The OAuth 2.0 protocol specifies four grant types [1]:

- Authorization code
- Implicit grant
- Password credentials
- Client credentials

This application will use authorization codes and client credentials. Client credentials will be needed for identifying requests coming from this application to Google and Fitbit servers, and authorization codes will be used for obtaining access tokens from users.

Client Registration

Before using the OAuth 2.0 protocol, the client must register with the authorization server in order to obtain a client identifier and a client secret by which the client can be identified [1].

In the case of using the authorization code grant, the authorization server requires the client to specify an endpoint, to which the resource owner will be redirected after specifying the scope and granting access to their protected resources [1]. The redirection endpoint is used by the authorization server to return a response containing authorization credentials [1].

Access Token

Access tokens are credentials used for direct access to protected resources, they have a limited lifetime and are restricted to the scope specified by the resource owner [1]. Valid access tokens are provided by the authorization server and should be stored in a secure storage by the client [1].

Protocol Flow

The interaction between the four roles, specified in the section 2.1.1, is illustrated in figure 2.1, and comprises of the following steps [1].

1. The client requests authorization from the resource owner.
2. The resource owner grants access and the client receives an authorization grant.
3. The client requests an access token by authenticating with the authorization server and presenting the authorization grant.
4. The authorization server provides the client with the access token that is limited to the scope specified by the resource owner.
5. The client requests access to the owner's protected resource by presenting the access token.
6. The resource server validates the access token, and grants access to the resources, if the token is valid.

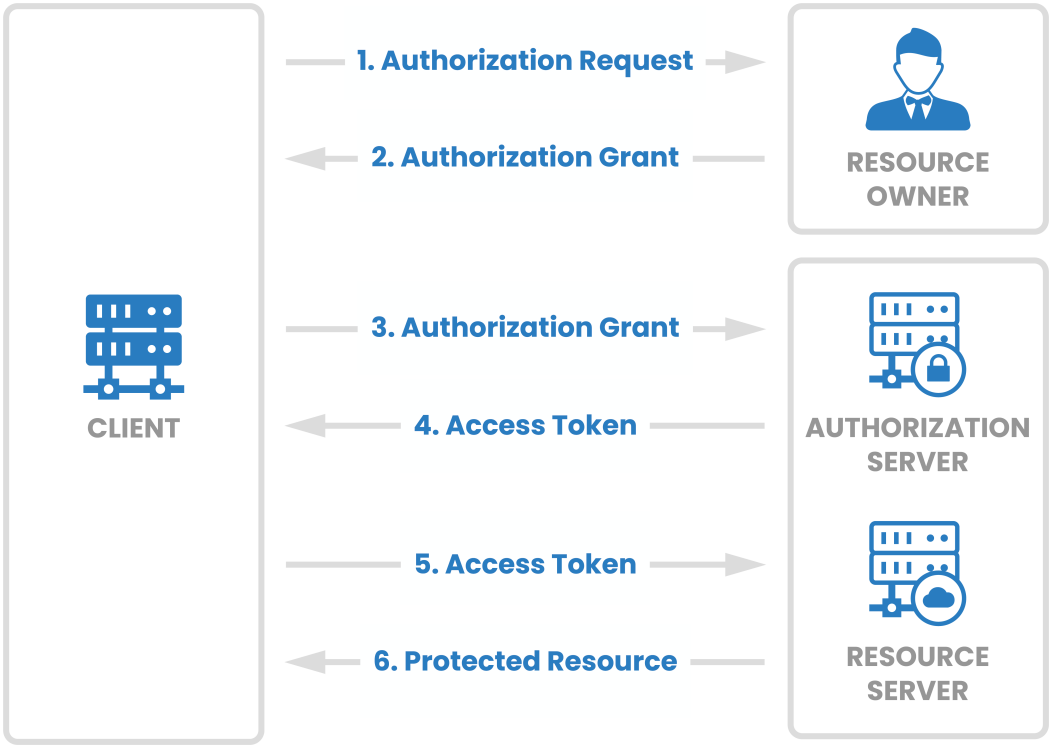


Figure 2.1: The OAuth 2.0 authorization process

2.1.2 Google Fit

Health data from a variety of devices and applications are stored in a central repository, called the fitness store [3]. The fitness store is a cloud service that persists health and wellness data using Google's infrastructure, which includes a set of Android and REST APIs that provide access to the fitness store [1]. The infrastructure of the fitness store is illustrated in the following figure 2.2.

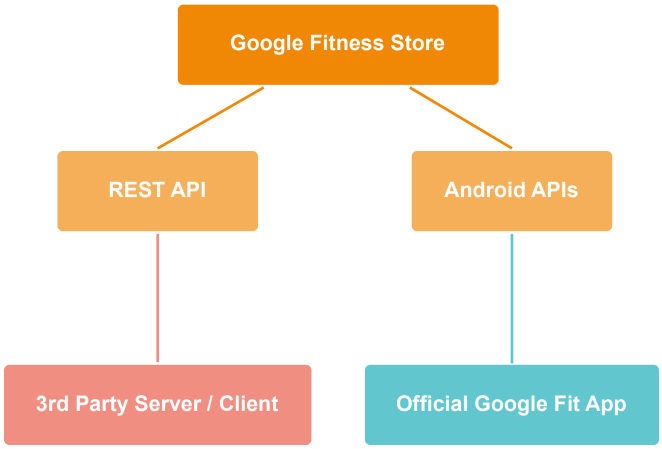


Figure 2.2: Google API architecture

Clients must be registered in order to have access to the Google Fit API. The registration consists of specifying authorized request origins and authorized redirect URIs.

Client registrations are managed by the Google Developers Console[1].

The result of the registration is a generated client identifier and a client secret, which are used for identifying incoming requests coming from the registered client.

Data Types

- Activity data type - captures physical activities that are performed in a certain time range. This data type measures metrics such as Basal Metabolic Rate, calories burned or step count [2].
- Body data type - captures body fat percentage, heart rate, height and weight [2].
- Sleep data type - captures sleep length, sleep type and sleep phases [2].
- Health data type - captures information regarding general health, which includes, e.g. blood glucose, blood pressure, body temperature or menstrual flow [2].

[1] Google Developers Console https://console.developers.google.com

Obtaining Access

The consent screen, shown in figure 2.3, is a web page that serves as an interface for users, through which they can grant access to their protected resources.

The screen displays information about the application and the requested scopes of access. The user can specify, to which scopes access will be granted, and to which denied.

After prompting the consent screen, the user is warned about an unverified application, as shown in figure 2.4. This warning occurs when an application is requesting access to sensitive scopes (activities, body measurement, heart rate etc. [4]), and has not yet been verified by Google. For the purpose of this application, the verified status is not necessary.

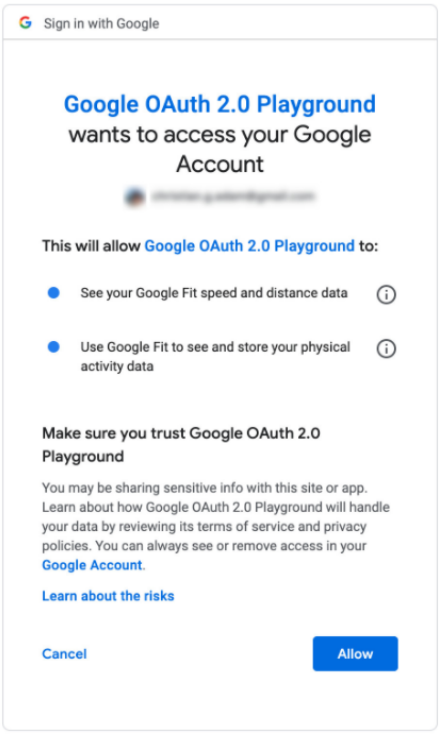


Figure 2.3: Consent screen

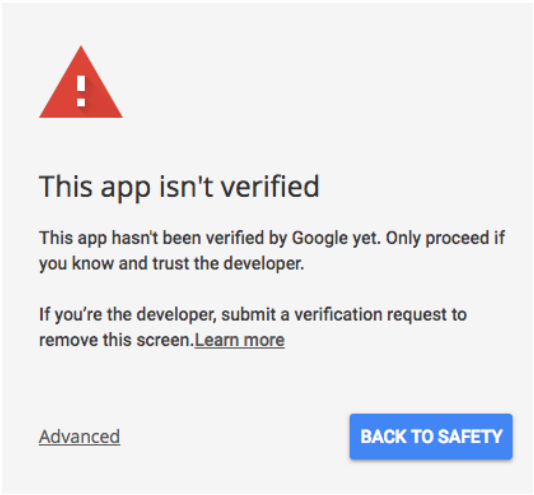


Figure 2.4: Unverified application warning

After a successful redirection to the consent screen, the user initializes the authorization process by logging into their Google account, and specifying the scope of access. If the authorization process finishes successfully, the user is redirected to one of the redirection endpoints specified on the client registration page. The result of the authorization process is an authorization code, which is used to obtain the user's access token.

The access token can be obtained by sending an HTTP request to the authorization server with the following structure:

```
POST https://oauth2.googleapis.com/token
Content-Type: application/x-www-form-urlencoded

client_id=[client_id]&client_secret=[client_secret]
&grant_type=authorization_code&redirect_uri[URI]&code=[code]
```

The parameters *client_id* and *client_secret* represent the client identifier and the client secret, *URI* is the path of one of the specified redirection endpoints, and *code* is the authorization code.

In case of a successful request, the authorization server returns a response that contains an access token, its lifetime, a refresh token and its type. The response has the following structure:

```
{
  "access_token": <access token>,
  "expires_in": 28800,
  "refresh_token": <refresh token>,
  "token_type": "Bearer"
  "user_id": <user id>
}
```

2.1.3 Fitbit

Health data and metrics are collected from Fitbit activity trackers, Aria scales and manually entered logs [5]. Logs can be manually added through an official Fitbit app for Android[2] or iOS[3], or by using the web API [5].

Registered clients can make a maximum of 150 API requests in an hour per each user [5]. An application can be registered on the official developer's website[4]. Once the registration is complete, the server generates a client identifier and a client secret, which are used to identify API requests from the registered application

Data Types

Body fat and weight logs can be manually logged by the user, or can be measured using the Aria scale [5].

Activity logs can be obtained using the Daily activity summary endpoint [5]. This endpoint returns a list of activities perfomed during the day, and additional data such as a summary of calories burned during the day, elevation, floors climbed, steps and activity minutes [5].

Daily step count can be extracted from activity logs, or obtained in a form of intraday time series, however, access to the intraday time series is limited, and granted only on a case-by-case basis [5].

Heart rate can be acquired in a form of heart rate zones, which are measured during activities, or as intraday time series [5]. Access to the heart rate intraday time series is also limited [5]. Heart rate zones can be divided into four categories [5]:

- Out of Range
- Fat Burn
- Cardio
- Peak

Each of the listed zones contain the amount of burned calories, maximum and minimum heart rate, and the duration of the zone in minutes [5]. Heart rate intraday time series contain the start time and the value of the heart rate at that specific time [5].

Sleep logs contain information regarding user's sleep, such as date, duration, efficiency and sleep levels [5]. Sleep levels can be of types [5]:

- Deep
- Light
- REM
- Wake
- Asleep
- Restless

[2]Fitbit Android app https://play.google.com/store/apps/details?id=com.fitbit.FitbitMobile
[3]Fitbit iOS app https://apps.apple.com/us/app/fitbit-health-fitness/id462638897
[4]Fitbit SDK https://dev.fitbit.com

- Awake

Each sleep level consists of the start date time, the level type, and the duration in seconds [5].

Obtaining Access

The authorization page, as shown in figure 2.5, will be used for gaining access to the user's data. On the authorization page, the user will be asked to log into their Fitbit account and specify the scope of access. For security reasons, the authorization occurs on the Fitbit server, and the authentication page must not be embedded in the application [5]. If the user consents to sharing their data by enabling at least one scope, the server redirects the user to the redirection endpoint that was specified on the registration page [5]. The URL of the endpoint contains a URL parameter with the authorization code [5], which will be used to obtain the access token.

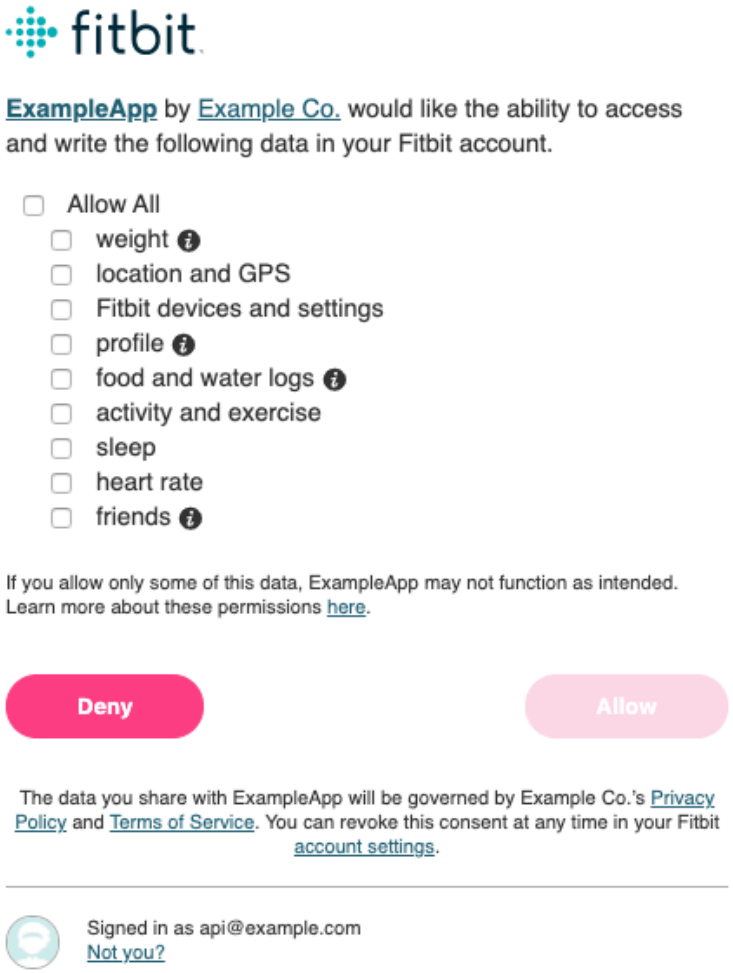


Figure 2.5: The Fitbit authentication page [5]

The access token can be obtained by sending an HTTP request with the authorization code to the authorization server [5] with the following structure:

```
POST https://api.fitbit.com/oauth2/token
Content-Type: application/x-www-form-urlencoded
Authorization: Basic [Base64_encoded_string]

client_id=[client_id]&grant_type=authorization_code
&redirect_uri[URI]&code=[code]
```

The parameter *Base64_encoded_string* represents the client identifier and the client secret separated by a colon, and Base64 encoded, which translates binary data into ASCII characters, and the parameter *URI* is the path of the redirect endpoint.

In case of a successful request, the authorization server returns a response with an access token, a refresh token and the token's lifetime in seconds.

```
{
  "access_token": <access tokenj>,
  "expires_in": 28800,
  "refresh_token": <refresh token>,
  "token_type": "Bearer"
  "user_id": <user id>
}
```

2.1.4 Apple HealthKit

The HealthKit provides a central repository containing health and fitness data from Apple devices [6]. Applications developed for this platform can, with the user's consent, access data stored in the repository [6].

Developers are not allowed to create new data types and units, however, the framework provides a wide variety of data types and classes that can be used [7].

Before using the HealthKit framework, the following steps must be performed [8].

1. Enabling the HealthKit in the application
2. Verifying that the HealthKit is available on the device
3. Creating the application's HealthKit store
4. Requesting permissions to read and share user's data

The HealthKit store is an access point for all data managed by the HealthKit [9]. Apple watches and iPhone devices have their own HealthKit stores that automatically perform synchronization between the two devices [7].

Data Types

The central repository stores the following data types in the HealthKit store [7]:

- Characteristic data. These records represent metrics, such as the user's birthdate, blood type, biological sex and skin type. The records are read-only and cannot be changed through a third-party application [7].

- Sample data. Most of the user's data is stored in samples that represent data at a particular point in time [7].
- Workout data. These records store information about exercise and fitness activities [7].
- Source data. Each sample, apart from storing corresponding data, also stores information about its source, such as information about the application or the device that saved the sample [7].
- Deleted objects. These objects are used to temporarily store the UUID of an item that has been deleted from the HealthKit store. The UUID can be used in a response when an object is deleted by the user [7].

Properties of Objects and Samples

All sample classes are subclasses of the *HKSample* class, which is a subclass of the *HKObject* class [7]. The *HKObject* subclasses are immutable and each object has the following properties [7]:

- UUID, a unique identifier for that particular entry [7].
- Metadata, data storing additional information about the entry [7].
- Source revision, the source of the entry can be, either a device that saved data into the HealthKit, or an application [7]. This property is recorded when the object is saved to the HealthKit store [7].
- Device, which is the hardware that generated the entry [7].

The *HKSample* classes, apart from inheriting properties from the *HKObject* class, also have additional properties: *type*, e.g. sleep analysis sample, a height sample, or a step count sample, *start date* and *end date* [7].

Samples are further divided into four subclasses: category samples, quantity samples, correlations, which composite data containing one or more samples, and workouts [7].

2.1.5 Samsung Health

The Samsung Health SDK for Android enables data sharing between the Samsung Health app and third-party applications [11]. The SDK for Android does not provide direct access to health data stored on the Samsung Health Server, but only to data stored in the Samsung Health app [11], therefore, the Samsung Health app installation is required for third-party applications to fucntion [11]. The relationship between a third-party application and Samsung Health app is shown in the following figure 2.6.

Figure 2.6: Data sharing between a third-party application and Samsung Health [11]

The restrictions for the latest SDK for Android are as follows[5] [11]:

- Samsung Health installation on the device (version 6.12 or above)
- Samsung Health runs on devices with Android 8.0 or above
- An applications's targetSdkVersion that uses Samsung Health SDK for Android must be 26 or above

From interacting with Samsung Health on different devices I found that health data is stored in the application, and is not automatically synchronized with data from other devices despite being signed in under the same account. The Samsung Health app does, however, provide an explicit option for users to synchronize their data with the Samsung Server across various devices.

[5] At the time of writing this thesis, the latest version of the SDK is 1.5.0

Access to user's data is permitted only after the user's explicit consent, and all permissions can be revoked by the user in the Samsung Health app at any time [11].

In comparison with the SDK for Android, the Server SDK provides a REST API for accessing data directly from the Samsung Health Server [12]. The REST API provided by the Server SDK uses the OAuth 2.0 protocol for authorization, and is not dependent on the Samsung Health app [12].

Both, the latest Server SDK and the Android SDK, are available only to Samsung Health partners who have a contract with the company [12, 11][6]. However, data can be accessed without an approved partnership through an older version of Samsung Health, which is compatible with the Android SDK version 1.4.0 [10].

The technological limitations for accessing data without a partnership are as follows:

- Using Samsung Android SDK version 1.4.0
- Installing a compatible Samsung Health app (6.2 or above and 6.11 or below) [10].
- Samsung Health app must have developer options enabled

Accessing data from the Samsung platform will require the development of a mobile application, which will acquire data from Samsung Health app.

2.1.6 Data Types

The API provides access to the following types of data [12]:

- Ambient temperature
- Blood glucose and Blood pressure
- Body temperature
- Caffeine intake
- Electrocardiogram data
- Exercise
- Food intake
- Floors climbed
- Glycated hemoglobin dat
- Heart rate
- Oxygen saturation
- Sleep and sleep stages
- Steps
- UV exposure
- Water intake
- Weight

[6] At the time of writing this thesis, the company is not accepting any new Partner Apps Program requests

2.2 Functional and Non-functional Requirements

2.2.1 Functional Requirements

- The system will allow user registration
- The system will allow logging in
- The system will allow users to log out
- The system will synchronize Fitbit health data on user's request
- The system will synchronize Google health data on user's request
- The system will synchronize Samsung health data on user's request
- The system will allow the user to display health data saved in the database
- The system will allow manual health data input into the system's database

2.2.2 Non-functional Requirements

- The system will periodically synchronize data from platforms
- The system will provide a user interface
- The system will only authorize access to data that belongs to the authenticated user
- The system will authenticate users based on their credentials
- The system will support synchronization from at least 3 platforms
- The system can be used by multiple users at the same time
- The system will visualize selected health data using graphs
- The system will be extensible
- The server will have a secured API
- The mobile application will be developed for Android devices

Chapter 3

Design

In this chapter, we will provide an overview of the design and technologies that will be used for the implementation of the system.

The system should be extensible, should automatically synchronize users' data, and should meet the system requirements specified in the section 2.2.

An extensible application is usually divided into separate components, that each handles different tasks and operations. Therefore, this system will be based on the client-server architecture in order to separate the back-end and front-end logic. The server will be responsible for storing and providing user's resources, and communicating with the client and the mobile application.

As mentioned in the section 2.1.5, a mobile application implementation is necessary for acquiring data from the Samsung platform.

3.1 Architecture

The server application will be responsible for storing user information along with their health data. Each user's data will be secured, and will only be accessible to the user to who the data belong. Only an authenticated user will be granted access to their data and synchronization options from 3rd party platforms.

The server architecture will be divided into three main layers.

The controller layer represents the REST API, which handles incoming requests. Its task is to convert data from the request into Java objects that are then passed down to the service layer.

The service layer handles authorization, object validation, object processing, and communicates with the data access objects (DAOs) in the persistence layer. This layer is also communicates with Google and Fitbit APIs through HTTP requests.

The persistence layer contains query methods that can be executed against the database. Objects in this layer are converted into database rows, or into objects in the case of performing a read operation.

The mobile application will communicate with the Samsung Health app through an SDK provided by Samsung. The acquired data will be sent to a corresponding server API endpoint, and then processed and persisted by the server application. The mobile application will only be usable by authenticated users.

The client web application will serve as an interface for users to display their data stored in the database. The web application will provide tools for users to manually synchronize data from different platforms, and will let users grant access to their data. Contents of the web application will be secured and will require an authorized user.

Both, the client application and the mobile application will communicate with the server by sending HTTP requests to the endpoints of the server's REST API.

Individual components of the system and their interfaces are illustrated in the following figure 3.1:

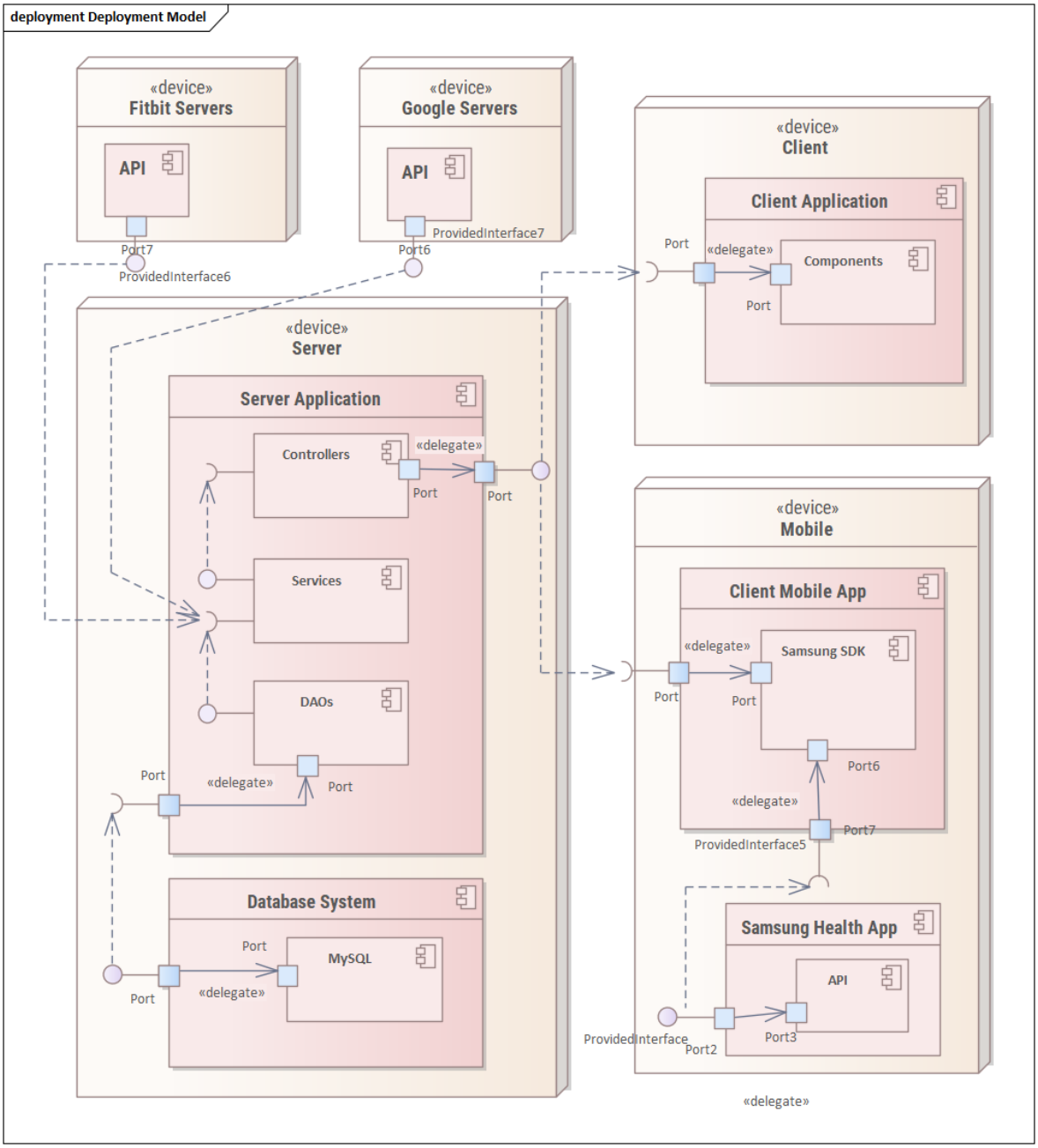


Figure 3.1: Proposed architecture

3.1.1 Server Application

The Spring Framework, specifically its extension, the Spring Boot, was selected for the development of the server application. It is an application framework for building enterprise Java applications [13]. Java is an object-oriented programming language that is fast, secure, reliable, and one of its main properties is that it can operate on multiple platforms[1].

The core of the Spring Framework is the utilization of the Inversion of Control (IoC) principle [14]. This principle transfers the responsibility of creating objects onto the framework.

In Spring, the Spring IoC container is responsible for the inversion of control, and the creation of beans, which are objects instantiated and managed by the IoC container [14]. The object dependencies are defined through arguments in constructors, arguments to a factory method, or properties in the object instance [14].

The extension of the Spring Framework, the Spring Boot, automatically configures the application based on dependencies that have been defined [16]. These dependencies can be defined in configuration files, or through using annotations.

The main advantage of the Spring Boot extension over the Spring Framework is the inclusion of an embedded server, and the automatic configuration [13].

The framework was chosen based on my experience with the Java language and the framework, that I have acquired during my studies and the fact that it allows the division into multiple application layers [15], which furthermore supports the extensibility of the system.

Task scheduling is also supported [13], which will be used for automating data synchronization. Moreover, the Spring Framework has well-written documentation and is under the *Apache License 2.0*.

Server API

The server application will provide a REST (Representational State Transfer) API, which is an architectural style that follows certain architectural principles [23]. Some of the principles are [23]:

- REST is a client-server architecture
- REST is stateless
- REST is a layered system

The REST API provided by the server will have endpoints for authentication, registration, health data access and manipulation, and health data synchronization from selected platforms. All the API routes will be secured except for routes that will be used for registration and user authentication.

[1] Introduction to Java Technology `https://www.oracle.com/java/technologies/introduction-to-java.html`

Authentication

This application will use a token-based authentication for user verification, and managing access to user's health data. Specifically, the system will use the JSON Web Token (JWT), which is an open standard that defines a way of transmitting information between independent parties in JSON format [24]. The information is digitally signed using a secret key, without which the information cannot be changed, and therefore can be trusted [24].

The structure of the JSON Web Token consists of three parts. Each part is Base64 encoded and is separated by a dot [24].

The first part is the header, which generally consists of two parts: the type of the token and the signing algorithm being used [24]. The token type is JWT, and the signing algorithm used in this application is SHA256.

The second part of the token is the payload, which contains the claims that hold information about an entity and additional data [24]. There are three types of claims: registered, public, and private claims [24].

The third part is the signature, which for its creation, requires the encoded header, the encoded payload and the secret [24]. The signature is created using one of the signing algorithms.

A JSON Web Token will be provided to users who successfully go through the authentication process by sending an HTTP request with valid credentials to the server's authentication endpoint. The token will have an expiration date, after which the user will be required to authenticate again.

Database

The server will store data in a MySQL database. The database was selected due to its integration with Spring boot, and my past experience of working with the system.

The database schema is based on the class diagram and will be attached in the appendices. The class diagram was designed with regard to the analysis that was conducted in chapter 2.

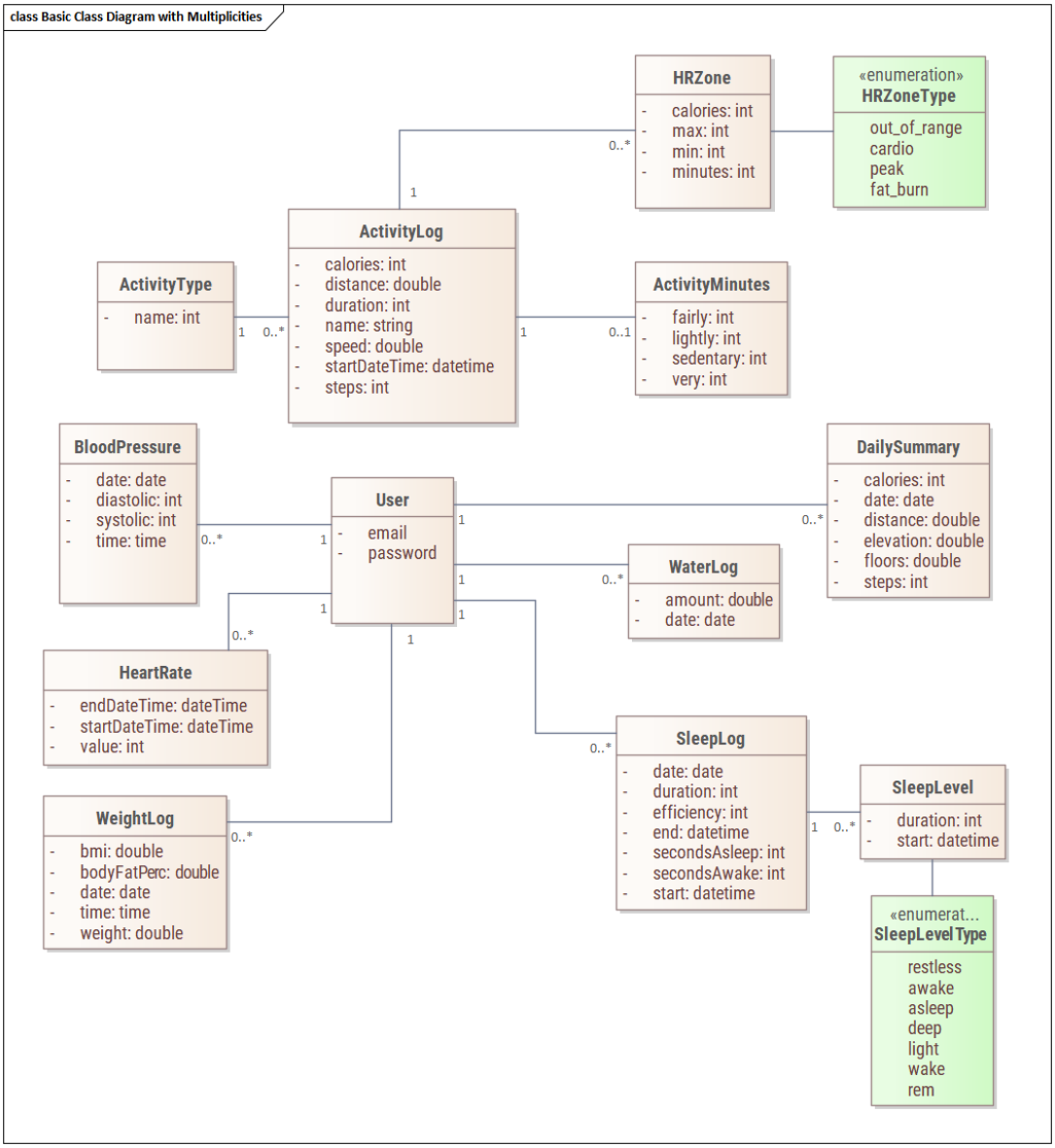


Figure 3.2: Class diagram

3.1.2 Client Application

For the development of the front-end, the React Framework was selected. React is an open source JavaScript based framework that is used for developing interactive and modern web user interfaces [22]. The framework encourages breaking down applications into smaller and simpler components [22], which results in components that are reusable

and easily maintainable.

One of the main objectives of the framework is to improve the rendering performance of the Document Object Model (DOM) [22]. The framework uses the concept of a Virtual DOM [22]. The virtual DOM is only a virtual representation of the real DOM and is updated each time the state of the application changes [22]. The virtual representation is compared with the real DOM, and the framework determines the most efficient way to update the DOM [22]. This concept results in better performance and faster rendering.

The React Framework was chosen based on its features and since it is based on the JavaScript language, loading new content onto the page does not require for the page to reload, which leads to an application that is more interactive, responsive, and user-friendly.

The client web application will provide a user interface that allows users to

- Register and log in
- Display health data according to a selected date
- Display charts visualising data by date range
- Manully add data logs
- Grant access to Fitbit and Google data
- Synchronize data from the platforms by date range

3.1.3 Mobile Application

The mobile application for retrieving data from Samsung Health app will be developed using React Native. React Native is an open source mobile application framework for developing Android and iOS applications. The framework was selected for its similarity with the React Framework. Similarly to React, React Native is based on working with components, state management, and styling using JavaScript [25].

The mobile application will be utilizing the npm package *rn-samsung-health*, which provides an API for accessing data from Samsung Health app.

The mobile application will have the following functionalities:

- Registration and logging in
- Synchronizing data Samsung Health by date range
- Periodically synchronizing data from Samsung Health

3.2 Used Libraries

Libraries were selected based on the license that they were under. All selected libraries are either under *Apache License 2.0*, *GNU General Public License v2.0*, *MIT License*, or under similar free licenses.

3.2.1 Server-side

Libraries used in the server-side application are managed by Maven, which is a tool that is used for building and managing Java projects. Maven aims to ease the building process, provide a uniform build system, provide quality project information and encourage good development practices [18]. Information about the project, configuration details and imported libraries are defined in an XML file, using which the Maven tool builds the project [17], and downloads specified libraries.

Spring Data JDBC

JDBC stands for Java Database Connectivity [19]. This dependency[2] provides an API that contains tools for connecting to the database, and executing queries against the database [19] (*Apache License 2.0*).

MYSQL Connector

This provides the application with a JDBC Driver, which is a component that allows java applications to interact with the database using JDBC [20] (*GNU General Public License v2.0*).

Spring Boot Starter Data JPA

This dependency allows the usage of JPA with Hibernate[3]. Hibernate is a Java framework that provides tools to store objects in relational database systems [21] (*Apache License 2.0*).

Spring Boot Starter Validation

Spring Boot Starter Validation serves as a tool for setting constraints, and validating objects using annotations[4] (*Apache License 2.0*).

Project Lombok

Project Lombok automates getter, setter and constuctor generation, by using annotations in class definitions. Getters, setters and constructors do not have to be explicitly declared, which leads to a cleaner and a shorter code[5] (*MIT License*).

[2]Spring Boot Starter Data JDBC
`https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-data-jdbc`

[3]Spring Boot Starter Data JPA
`https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-data-jpa`

[4]Spring Boot Starter Validation
`https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-validation`

[5]Project Lombok
`https://github.com/rzwitserloot/lombok`

Gson

The Gson library is used for converting Java objects into their JSON representation, and for converting JSON representations into Java objects[6] (*Apache License 2.0*).

Java JWT

The Java JWT library provides an interface for creating and verifying JSON Web Tokens used for user authentication[7] (*Apache License 2.0*).

3.2.2 Client-side

Packages and libraries in React are managed by npm, which is software registery that allows developers to share their software[8].

React Router

React Router is a routing library for React. It keeps the UI in sync with the URL[9] (*MIT License*).

React Nice Dates

React Nice Dates is a responsive and touch-friendly library for React that provides date picker components[10] (*MIT License*).

Recharts

Recharts is a library built with React and D3 library, which contains separated customizable components for creating charts and graphs[11] (*MIT License*).

Moment

Moment.js is a JavaScript library for parsing, validating and formatting dates[12] (*MIT License*).

Axios

Axios is a promise based JavaScript library used to make HTTP requests from node.js or the browser[13] (*MIT License*).

[6] Gson
https://github.com/google/gson

[7] Java JWT
https://github.com/jwtk/jjwt

[8] What is npm,
https://www.w3schools.com/whatis/whatis_npm.asp

[9] React Router
https://github.com/ReactTraining/react-router

[10] React Nice Dates
https://github.com/hernansartorio/react-nice-dates

[11] Recharts
https://github.com/recharts/recharts

[12] Moments
https://github.com/moment/moment

[13] Axios
https://github.com/axios/axios

3.2.3 Mobile Application

React Native Background Actions

React Native Background Actions is used for running background tasks in Android and iOS devices [14] (*MIT License*).

RN Samsung Health

RN Samsung Health is a React Native module for reading data from the Samsung Health app using the Samsung Android SDK [15] (*Unlicense License*).

RNSecureStorage

A secure storage for React Native applications. This storage will be used for storing user's authorization credentials [16] (*MIT License*).

[14] React Native Background Actions
https://github.com/Rapsssito/react-native-background-actions

[15] rn-samsung-healthhttps://github.com/GaneshSinghPapola/rn-samsung-health

[16] RNSecureStoragehttps://github.com/talut/rn-secure-storage

Chapter 4

Implementation

This section will describe certain parts of the implementation in more detail. The chapter will not cover the entire implementation process, but only some of the main parts.

4.1 Entity Structure

The following piece of code represents a User entity. This is the object representation of the database table. The Spring boot framework handles the conversion of the Java object to database data, and vice versa.

```
@Getter
@Setter
@Entity
public class User {
    @Id @GeneratedValue
    private Integer id;

    @NotNull
    @Email
    @Size(min = 5, max = 320)
    private String email;

    @NotNull
    @Size(min = 6, max = 128)
    private String password;

    @CreatedDate
    private Date createDate;

    @NotNull
    private Role role;
}
```

The frameworks performs the conversion with respect to the dependencies that were declared between the objects using annotations. The *@Entity* annotation lets the framework know that this class represents a database table, and annotations like *@Id*, *@NotNull*, *@Size* and *@Email* enforce constraints on the corresponding object fields - and database columns.

@Getter and *@Setter* annotations are part of the The Project Lombok library. They automate the generation of setter and getter methods, which leads to a shorter, cleaner, and more readable code.

4.2 Formatting and Persisting JSON Data

Data acquired from the body of HTTP responses from Fitbit and Google servers, and data that is sent from the mobile application, are all in JSON formats. In the Java language, the response body is represented by a string and needs to be converted into a JSON object.

Data in the String format is converted into a JsonNode object, which provides methods that can find a value by key, and methods that can parse the value, and return it in the desired data type, such as an integer, a double, or a boolean.

Values from the converted data are then mapped to the correlating entity fields and persisted into the database. The persistence is preceded by the removal of existing records. This step is necessary since the user could have, either, deleted or altered records in the Fitbit or Google apps, which without the removal, would stay in the database. This would mean that the data saved in the database would not correspond with the data saved in Fitbit and Google servers.

4.3 Data Validation

Data validation is handled by the Spring boot framework. It is performed in the service layer, where the object is validated in accordance with the constraints that were enforced with annotations, as shown in the section 4.1.

In the case of an object that violates any of the constraints, the server returns an HTTP response with status code 400 with a list of violated constraints. This response is further processed by the client application.

4.4 Periodic Data Synchronization

On the server, task scheduling is implemented with the usage of the *@Scheduled* annotation. The synchronization is performed through a scheduled task. The task is set to periodically synchronize data in a specified date range for each user who has a valid authorization token saved in the database.

The property *fixedRate* in the *@Scheduled* annotations contains a value in milliseconds that tells the framework, how frequently the scheduled task should be executed. The *initialDelay* property delays the first execution of the task by the number of specified milliseconds.

```
@Scheduled(fixedRate = 1000 * 60 * 60 * 1, initialDelay = 5000)
public void synchData() {
    userDao.findAll().forEach(user -> {
        //synchronize data
        fitbitSyncService.syncDataFromDateRange(user, endDate, startDate);
        googleSyncService.syncDataFromDateRange(user, endDate, startDate);
    });
}
```

The synchronization in the mobile application is implemented as a background task, which will run in the background, and will not interrupt the flow of the application. The background task is created using the *react-native-background-actions* library.

```
const backgroundTask = async (taskDataArguments) => {
    //asynchronous Promise
    await new Promise( async (resolve) => {
            //function handling data synchronization
            await syncHealthData();
            await sleep(delay);
    });
};
```

The task uses a Promise, which is an object that may sometime in the future return a value from an asynchronous operation[1]. The *await* keyword in the Promise makes the function behave synchronously, which means that the *sleep* function will be called only after the *syncHealthData* function has finished. The *sleep* function delays the execution of the background task by a number of specified milliseconds. In the mobile application, this value is set to 60 000 ms, therefore, the synchronization will be performed every 60 seconds.

4.5 REST API

In this section, we will provide an example of one of the REST API's endpoints and its implementation.

The following code snippet represents the implementation of the *ActivityController* class, which handles HTTP requests regarding activity health data.

```
@RestController
public class ActivityController {
    @Autowired ActivityService activityService;

    @GetMapping(value = "/activity/{platform}/{date}")
    public ResponseEntity<?> getDailyActivities(Request req, @PathVariable
        String date, @PathVariable String platform) {
        activityService.getActivityLogs(user, from, date);
    }
}
```

From the example, we can observe, that the controller does not handle any validation or data processing, but instead, calls a method in the service layer, that is responsible for these tasks. The field with the service object is annotated with *@Autowired*, which injects the dependency implicitly, without the need to instantiate the object.

The *@GetMapping* annotation means that the method will be called when there is an HTTP request with the method GET requesting this specific endpoint. The value of the annotation contains a path of the endpoint. The path can contain dynamic parameters, and in this case, contains *platform* and *date* parameters. The *platform* parameter represents the platform from which to select the data, and *date* represents the date that the data is from.

[1]Using Promises https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises

4.6 List of Synchronized Data

4.6.1 Fitbit

- Activities
- Water intake
- Sleep data and sleep stages
- Weight data
- Daily burned calories
- Daily step count
- Daily distance
- Daily floors climbed
- Daily elevation

Google

- Activities
- Daily step count
- Blood pressure
- Sleep data
- Weight data

4.6.2 Samsung

- Activities
- Daily steps
- Daily distance
- Daily calories
- Daily step count
- Sleep data and sleep stages
- Heart rate
- Water intake
- Blood pressure

Chapter 5

User Interface

This section will give an overview of the designed user interface, and break down certain parts of the UI.

5.1 Web Application

5.1.1 Sign In and Navigation

When first landing onto the page, the user is presented with a sign in screen, where they have to authenticate with their e-mail and password that they registered with.

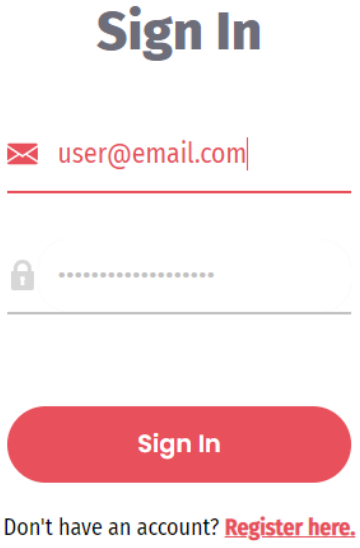


Figure 5.1: Sign in page

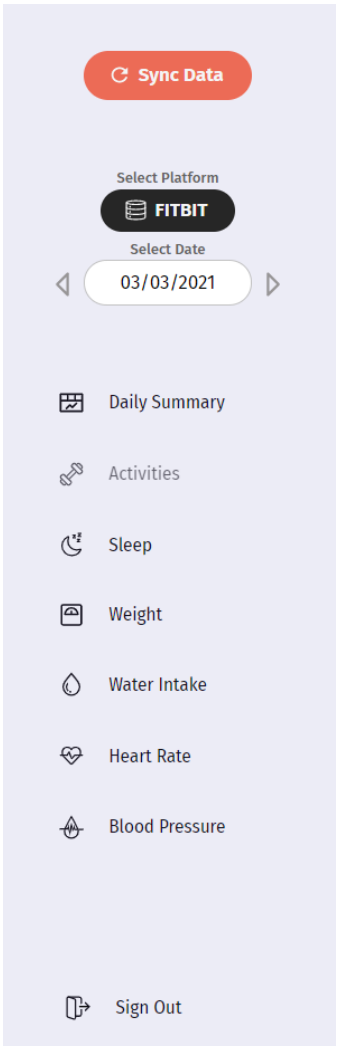


Figure 5.2: Navigation

The navigation bar contains the following elements:

- Button that opens a modal window where users can synchronize their data
- Button for selecting from which platform they want to display their data
- Navigation links, and a sign out option

5.1.2 Data Synchronization

The following figure 5.3 represents a modal window, which provides an interface for synchronizing Fitbit and Google data. The links at the bottom of the window open up a pop up window, where the user can grant necessary access permissions.

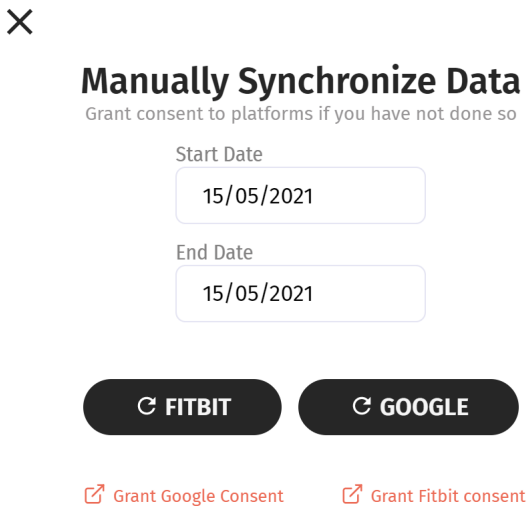


Figure 5.3: Modal window for manual data synchronization

The window provides date pickers, which let the user pick a date range from which to synchronize data.

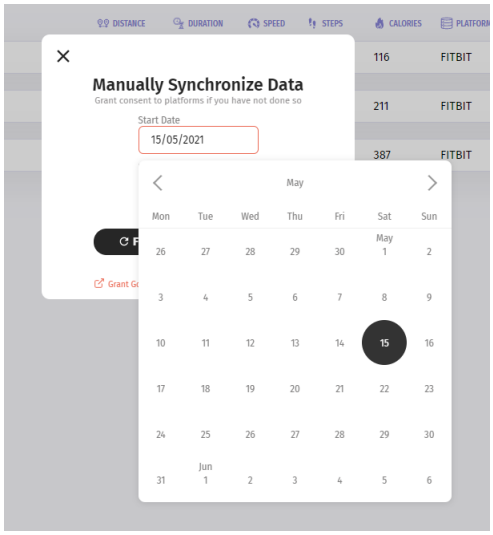


Figure 5.4: Sleep duration over a period of time

5.1.3 Data Visualization

Data are visualized with charts using the *Recharts* library.

The following figure 5.5 shows sleep duration over a period of time. The date range can be set using the date pickers that are located above the line chart.

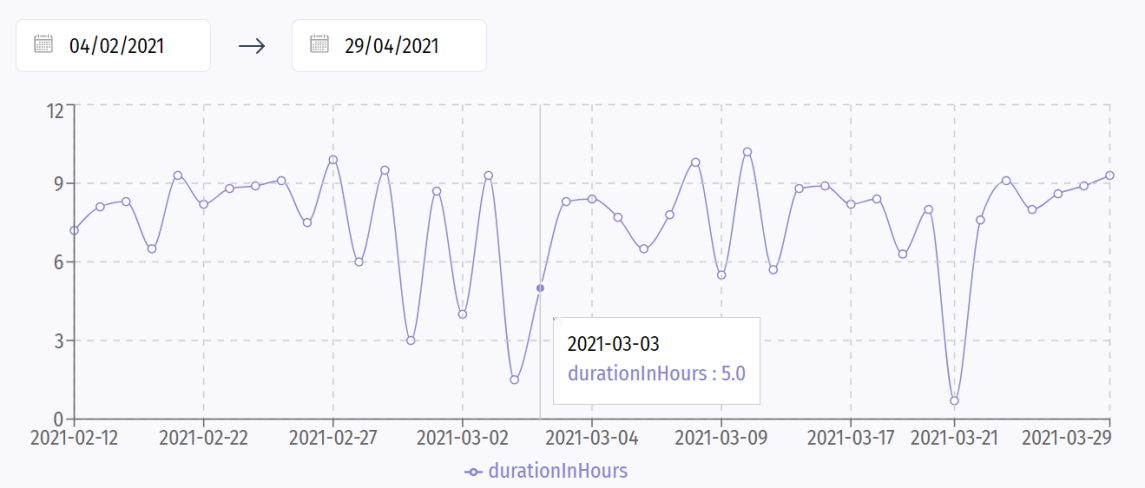


Figure 5.5: Sleep duration over a period of time

Data is also visualized simply as a list with columns. The following example is an activity log from 23.02.2012.

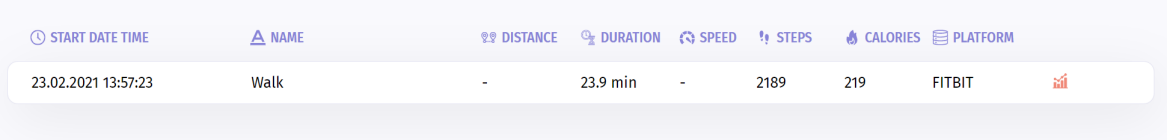


Figure 5.6: Activity log

The log contains a clickable icon at the end of the list item, which opens a modal window with bar charts that visualize active minutes and heart rate zones.

Heart rate zones, as shown in the bar chart in figure 5.7, are represented by four zones. Each one of these zones correspond to the heart rate intensity during the activity[1].

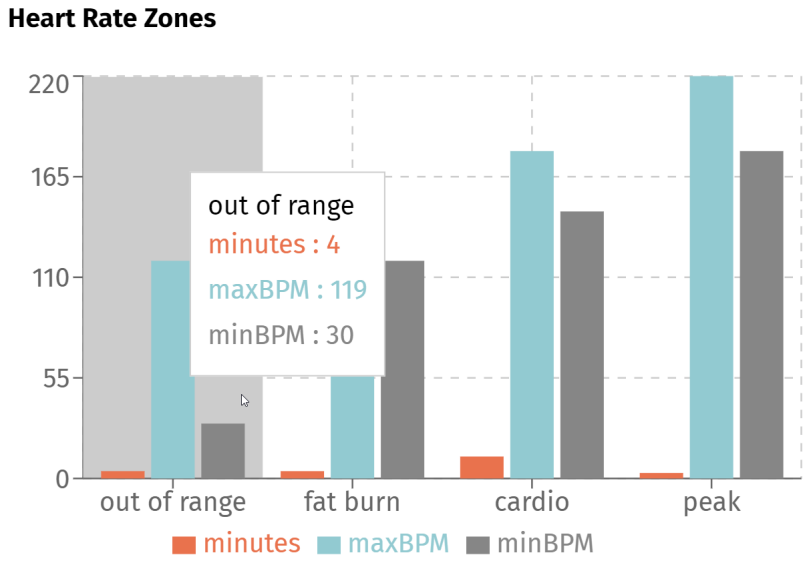


Figure 5.7: Heart rate zones

Active minutes tells us how active the person was during the activity.

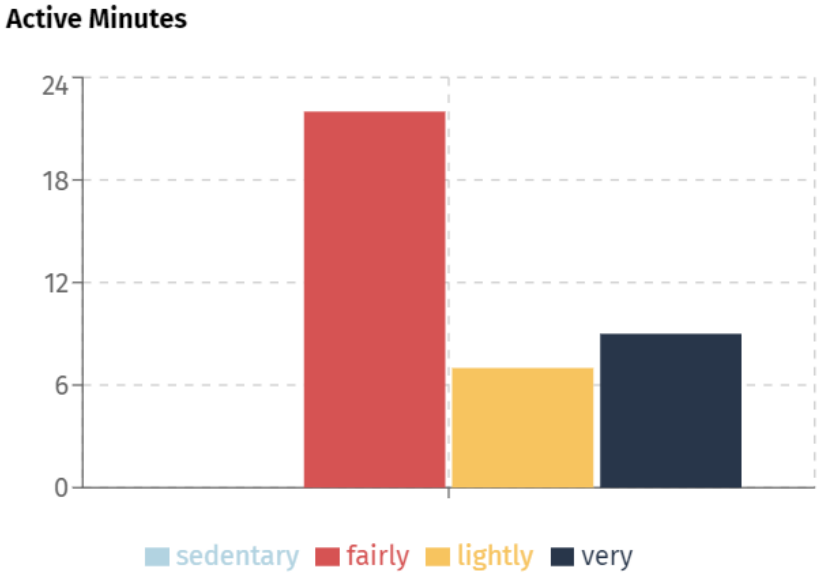


Figure 5.8: Active minutes

[1]Heart rate zones | the basicshttps://www.polar.com/blog/running-heart-rate-zones-basics

5.2 Mobile Application

The user interface of the mobile application is simplified in comparison with the web application. An implementation of a user interface of the mobile application was not listed as a requirement in the assignment of the bachelor thesis, however, I wanted to at least design a simple interface, that would display some of the data stored in the database.

After logging into the app, the user is asked to allow the app to read data from Samsung Health, as shown in figure 5.10.

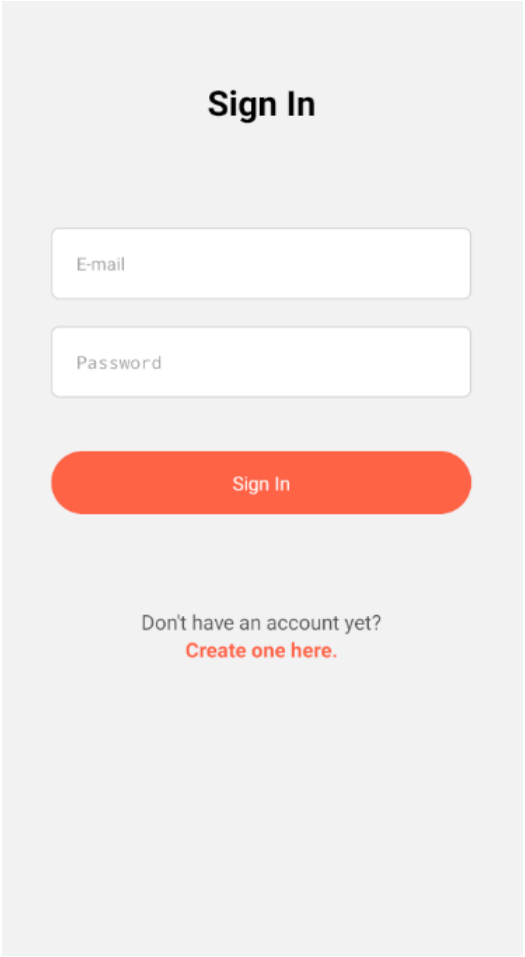


Figure 5.9: Sign in screen

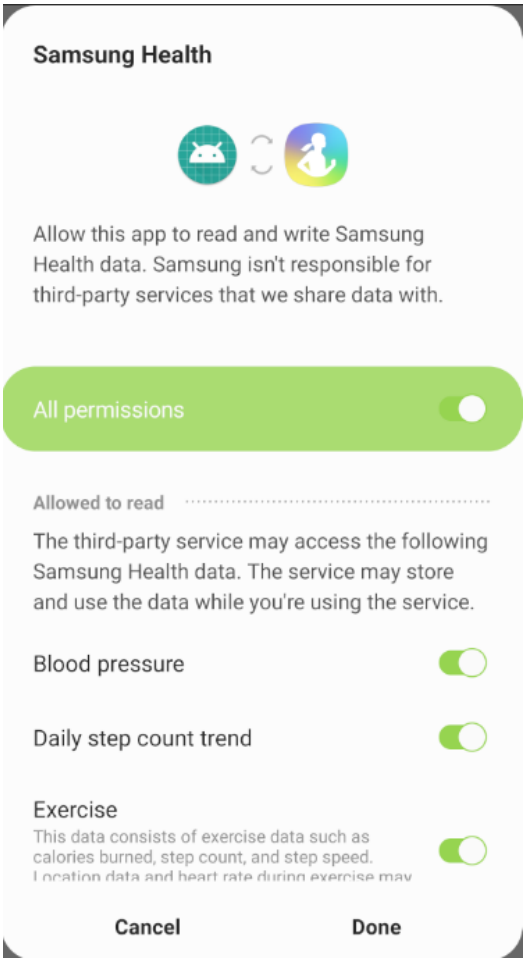


Figure 5.10: Permission screen

After granting necessary permissions, the user lands on a home screen, shown in figures 5.11 and 5.12, which displays a summary of data from a given day.

The home screen shows the following data:

- Calories burned, daily steps, and distance walked
- Amount of water drank
- Duration of sleep
- Weight
- Blood pressure

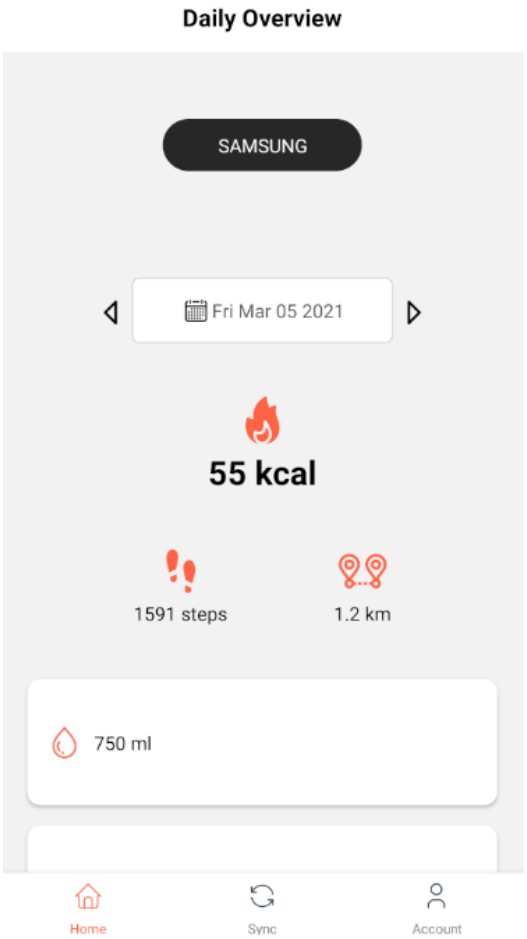


Figure 5.11: Home screen

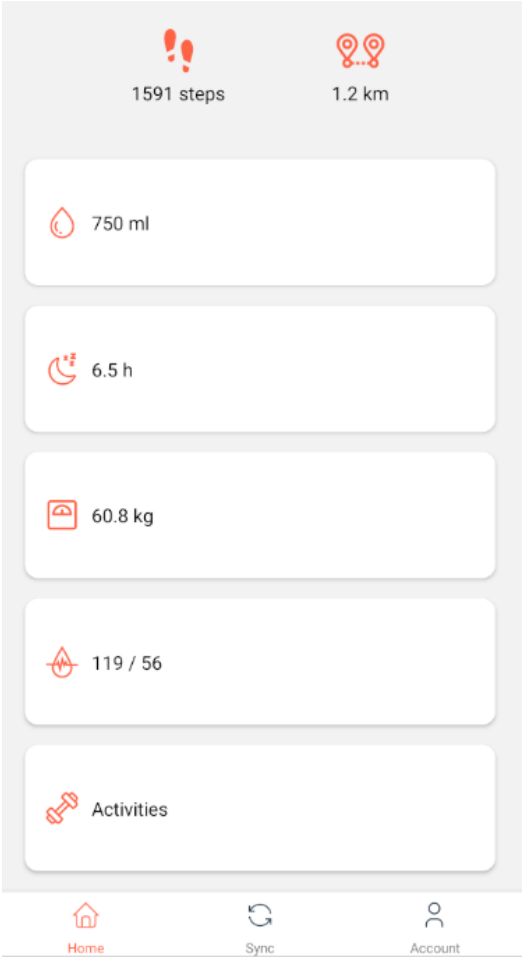


Figure 5.12: Home screen

Here is an example of a screen which displays all activities and some of the data related to them from a specific day. The screen shown in figure 5.14, lets the user to record the amount of water intake.

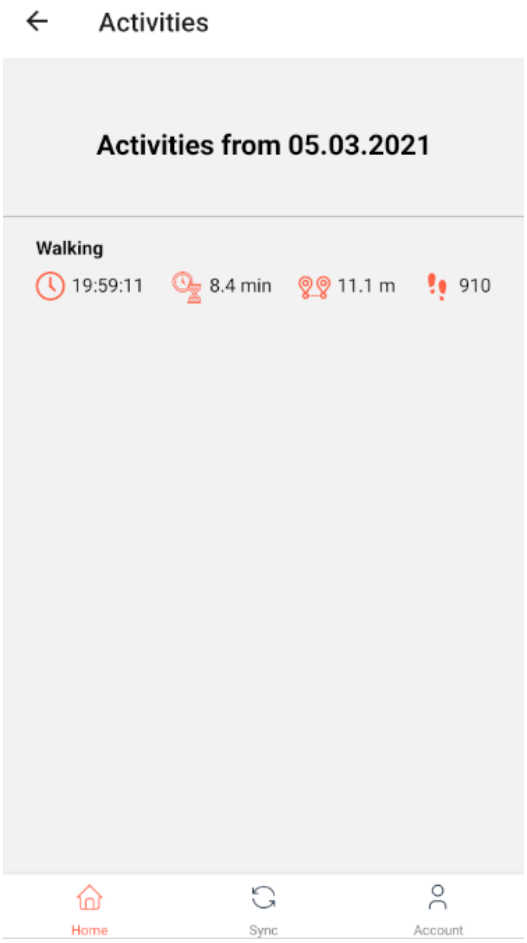


Figure 5.13: Activities screen

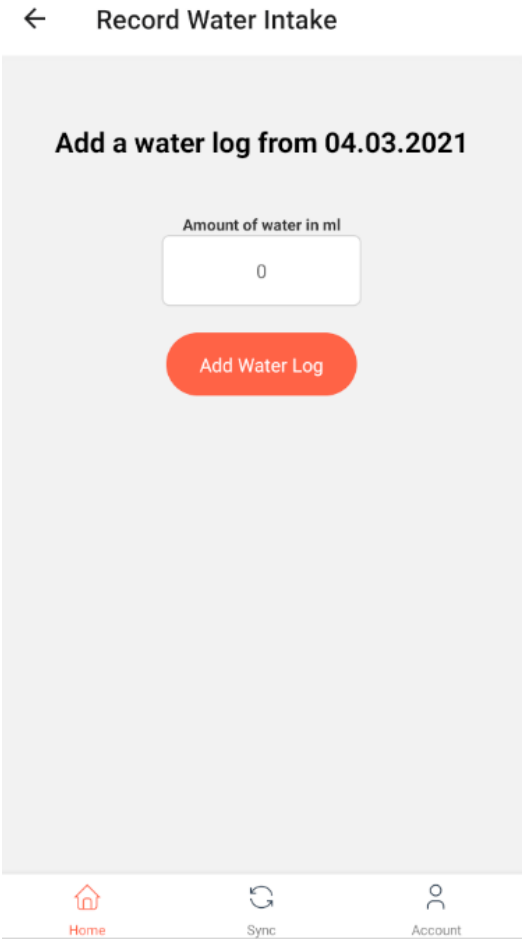


Figure 5.14: Record water screen

The screen shown in figure 5.15 serves as an interface for the user to manually pick a date range, and then proceed with the synchronization. The notification, shown in figure 5.16, gets updated each time the synchronization running in the background occurs.

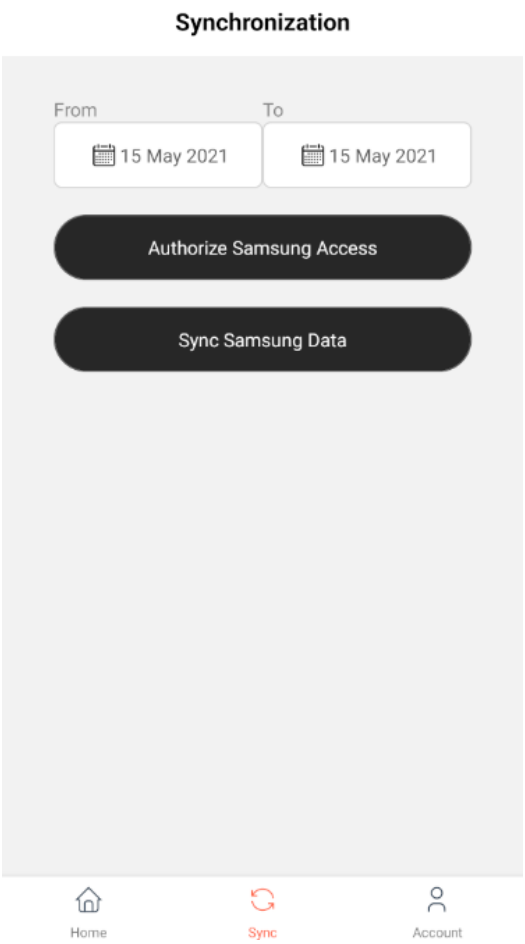


Figure 5.15: Synchronization screen

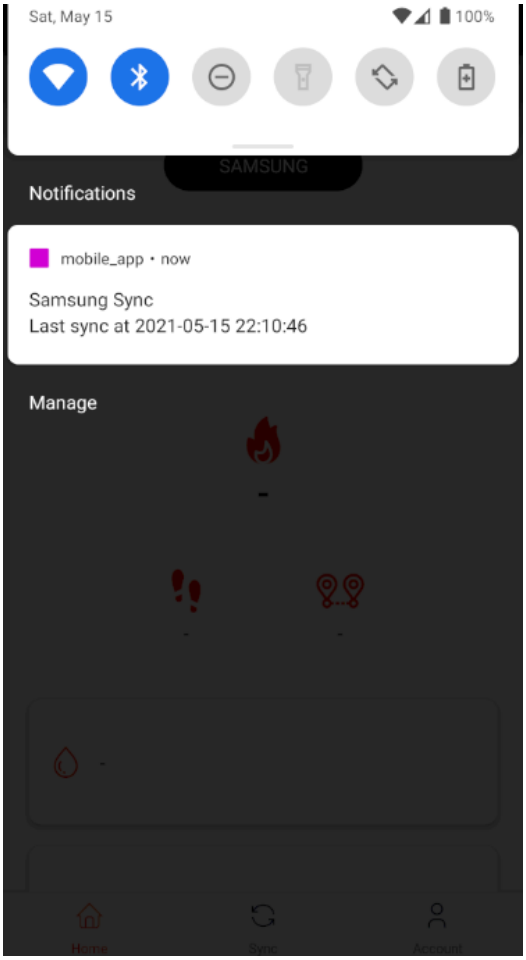


Figure 5.16: Record water screen

Chapter 6

Testing

The framework JUnit[1] was used for the testing of the application. The testing was divided into several phases, and was performed against an SQL in-memory database *H2 Database*[2].

6.1 Unit and Integration Testing

The tests were automated, and a *Generator* class provided data for the tests.

The following list shows some of the examples that the unit tests focused on:

- Testing the correction of queries executed against the database
- Testing object validation
- Testing methods tasked with creating objects
- Testing user authorization
- Testing the persistance of approximately 1000 records of each metric
- Testing that the user will find a the persisted data after adding a log

6.2 Testing Real Life Data

The application was used to retrieve data from Fitbit, Google and Samsung platforms in a date range from February 18th 2021 to April 18th 2021. Data from this data range was recorded with Samsung Galaxy Watch Active 2, Fitbit app, Samsung Health app, and Google Fit app. This date range was selected due to the fact that I was adding data, and wearing the smartwatch during this time period.

[1] JUnit `https://junit.org/junit5/`
[2] H2 Database Engine `https://www.h2database.com/html/main.html`

Chapter 7

Conclusion

The analysis of selected platforms on which health data are stored was conducted. The analysis provides an overview of the selected platforms, ways of how to access data on these platforms, and data types provided by them.

System requirements were determined by the thesis assignment and the conducted analysis. A system fulfilling the requirements was designed, and will be based on the client-server architecture. A system divided into a server application, client web application, and mobile application was proposed to store, display and acquire data from the selected platforms through APIs and a mobile application. The system was designed to be extensible to be easily extended by new platforms in the future.

The final implemented system is functional and meets the requirements of the assignment. Data synchronization is performed periodically on the server and in the mobile application. However, the synchronization can be manually triggered by the user.

The metrics are visualized in the web application using charts. A user interface for the visualization of the data on the mobile app was not in the requirements, however, the user interface was designed and implemented. The data in the mobile app are visualized in a simplified form, in comparison with the web application.

The application can be used by people, who for whatever reasons, use multiple tracker devices at a time, but also by people who use one platform and want to browse through their health data using this application.

Bibliography

[1] *The OAuth 2.0 Authorization Framework*. RFC 6749, DOI 10.17487/RFC6749, íjen 2012. https://www.rfc-editor.org/info/rfc6749. 4.12.2020.

[2] *Data types*. Google Fit https://developers.google.com/fit/datatypes. 4.12.2020.

[3] *Platform Overview*, https://developers.google.com/fit/overview. 9.12.2020.

[4] *Data Types, Authorizaztion scopes*, https://developers.google.com/fit/datatypes#authorization_scopes. 9.12.2020.

[5] *Fitbit Web API*, https://dev.fitbit.com/build/reference/web-api. 9.12.2020.

[6] *HealthKit*, Apple Developer Documentation. https://developer.apple.com/documentation/healthkit. 9.12.2020.

[7] *About the HealthKit Framework*, Apple Developer Documentation. https://developer.apple.com/documentation/healthkit/about_the_healthkit_framework. 9.12.2020.

[8] *Setting Up HealthKit*, Apple Developer Documentation. https://developer.apple.com/documentation/healthkit/setting_up_healthkit. 9.12.2020.

[9] *HKHealthStore*, Apple Developer Documentation. https://developer.apple.com/documentation/healthkit/hkhealthstore. 9.12.2020.

[10] *Samsung Health Android SDK - Data API Reference*, https://img-developer.samsung.com/onlinedocs/health/android/data/index.html. 9.12.2020.

[11] *Samsung Health SDK for Android*, Samsung Developers. https://developer.samsung.com/health/android/overview.html. 9.12.2020.

[12] *Samsung Health API Reference*, Samsung Developers. https://developer.samsung.com/health/server/partner-only/intro.html. 9.12.2020.

[13] *Spring Boot Reference Documentation*, https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/. 9.12.2020.

[14] *Spring Boot Reference Guide, "Core Technologies"*, https://docs.spring.io/spring-framework/docs/current/reference/html/core.html. 9.12.2020.

[15] *Spring Framework, "Application Layering"*, https://docs.spring.io/spring-roo/reference/html/base-layers.html. 9.12.2020.

[16] *Spring Boot Reference Guide, "III. Using Spring Boot"*, https://docs.spring.io/spring-boot/docs/current/reference/html/using-boot-auto-configuration.html#using-boot-auto-configuration. 9.12.2020.

[17] *Maven documentation, "Introduction to the POM"*, https://maven.apache.org/guides/introduction/introduction-to-the-pom.html. 9.12.2020.

[18] *About Maven, "What is Maven?"*, https://maven.apache.org/what-is-maven.html#mavens-objectives. 9.12.2020.

[19] *Java JDBC, javapoint.com*, https://www.javatpoint.com/java-jdbc. 9.12.2020.

[20] *Java JDBC Driver, javapoint.com*, https://www.javatpoint.com/jdbc-driver. 9.12.2020.

[21] *JPA vs Hibernate, javapoint.com*, https://www.javatpoint.com/jpa-vs-hibernate. 9.12.2020.

[22] *React Docs*, https://reactjs.org/docs/getting-started.html. 9.12.2020.

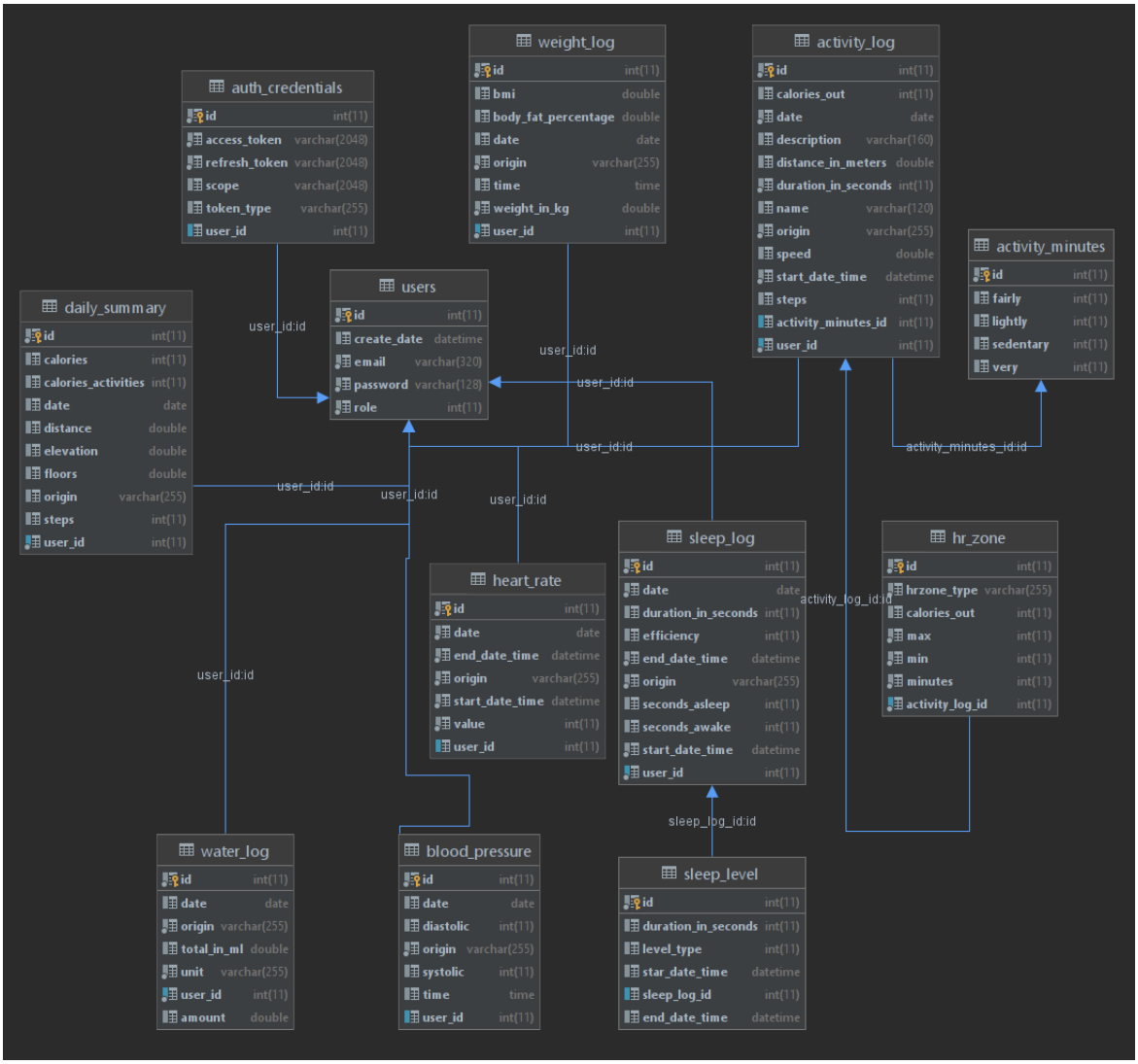
[23] *REST Principles*, https://ninenines.eu/docs/en/cowboy/2.6/guide/rest_principles/. 9.12.2020.

[24] *Introduction to JSON Web Tokens*, https://jwt.io/introduction. 9.12.2020.

[25] *React Native Docs*, https://reactnative.dev/docs/getting-started. 9.12.2020.

Appendices

Appendix A Database Schema



Appendix B Instalation Manual

Database Server

1. Set up a local MySQL server on your PC
2. Create a database called *fitnessapp*
3. Set database login credentials to username: *root* and password: *mysql* (this setting can be altered in the application.properties file in the folder containing the server application)

Server

The application server is located in the folder *healthdataapp*.

1. Install Java Runtime Environment 11
2. Run the Spring boot application using an IDE
3. Installation of a server is not required since Spring bood contains an embedded server
4. The database tables will be generated automatically upon running the server application

Web Application

The web application is located in the folder *web-app*.

1. Install the JavaScript runtime environment Node.js
2. Run the application using the command *npm run build*

Mobile Application

The mobile application is located in the folder *mobile_app*.

1. Install Android Studio, and configure a virtual android device
2. Run the command *npm run android* from the android folder in the root folder
3. The virtual device will be recognized and will automatically start, and then install the mobile application
4. Install Samsung Health app (version 6.2 or above and 6.11 or below) on the virtual device
5. Run the installed mobile application