food4future  
StudyCompanion

**Android App & Backend Server**

**Specification Document**

App Structure fundamentals  
Frontend/Backend communication protocol  
Data model

*Version: 0.7   
2024-02-22*

**IMPORTANT NOTE as of February 2024:**

**This document is no longer up-to-date!!!!!!**

All communication interface specification was replaced by the OpenAPI interface / **Swagger UI**, which you can access on a running backend instance at:

**https://{your-backend-instance-URL}/docs**

or directly via the OpenAPI interface:

**https://{your-backend-instance-URL}/openapi.json**

Both the f4f Study Management Portal Web App and the Android App communicate with the backend server through the endpoints defined in that API.

This document was created in an early project phase, were the software was not yet meant to be generic solution for self-customizable studies but rather specified to be used in our first “SEriNa“ study only. This was also the phase in which most effort was put into the development of the Android App.  
The structure and documentation of the API was heavily improved later to be conform with the newly developed *f4f Study Management Portal* Web Application and allow to create and manage custom study contents, but we kept the Android App mostly untouched concerning the communication interface. For the schema information (food selections, enums, images, data types, device config, etc.), even though everything is now dynamically customizable, the server provides specific legacy endpoints (in the OpenAPI spec categorized as “Internal – Schema Access”) specifically for the Android App, where it finds all schema information as static JSON documents. In the background the server generates those documents dynamically depending on which user authentication information the App sends with the request header.

We keep this document as a reference since it mostly shows how the Android App uses the interface, although the schema has changed in many aspects and there are more endpoints in the API than described here.

Also notice that support of wearable Garmin Devices as still described in this document was **removed**.

# User Roles

|  |  |  |
| --- | --- | --- |
| **#** | **Role** | **Tasks** |
| **A** | Administrator | * Creation and Deletion of users * Assign any role to any user * inherits all abilities from roles:  Supervisor and Nurse * manage Bluetooth connections to sensor devices on the smartphone |
| **S** | Supervisor | * access to pseudonymized participant list (usernames are removed) through Rest API * access to all study data acquired from participants through Rest API |
| **N** | Nurse | * creation/deletion of users with role participant * modification of static data for participants * Generation of QR codes which allow participants to download the app and connect it to f4f * send registration e-mails to participants * Modification of username and password for participants |
| **P** | Participant | * answering of daily questions during course of study (user data acquisition) * receive notifications about synchronization and device connection status * manage Bluetooth connections to sensor devices on the smartphone (pair, unpair, connect to devices) * synchronize/read data from sensor devices via Bluetooth (also in background) * synchronize study data with server back-end  (also in background) |

# Back-End REST API endpoints

## General notes

The f4f server backend API is accessed through sending GET, PUT, POST or DELETE requests to one of the following base URLs:

During test & development phase:  
  **https://f4f.tm.th-wildau.de:8443/**{endpoint}[**/{**url-params...}]

During active phase:  
  **https://f4f.tm.th-wildau.de[:443]/**{endpoint}[**/{**url-params...}]

**Input parameters**

The tables in the following sections describe various input parameters.

For POST and PUT requests, the input parameters must be formatted as JSON Strings as body of the request. The requests must have „*Content-Type: application/json*“ header defined.

Input parameters, which are used to uniquely identify a corresponding dataset and which are an indisensible part of the request, are directly passed as part of the endpoint url. This is the case for example with /user/{id} or /sync/{SyncProcessID}.

For GET requests, the HTTP specification does not allow a body. Hence for these requests, additional input parameters are to be passed as parameters in the HTTP header. Boolean values in HTTP header are String-represented with „true“ resp. “false“.

The „/token“ POST request is exceptional, since it follows OAuth 2.0 conventions. Details are found in the following section.

**Output Data**

The returned output data of all requests is formatted as JSON string in the body of the returned server response.

**Errors / Invalid requests**

Invalid requests or requests, which caused server errors, return a response with an appropriate HTTP Request Code inequal to 200 contining a JSON Object with one string field named “details” containing an error description string .

## Authorization process (OAuth 2.0)

To use requests which require user authorization (which regards to almost all requests), a valid **access token** must first be acquired from the server.

This can be achieved through a POST-Request on the /token endpoint. Note that the following **HTTP header** lines are required for this request:

Content-Type: application/x-www-form-urlencoded; charset=UTF-8  
 username: {username of user to sign in}  
 password: {password of user to sign in}

This must be done only **once**. The „*access\_token*“ entry of the returned JSON object contains the authentication token, which can be used for all following restricted requests.**Requests, which require authorization**, must contain the following HTTP header line:

Authorization: Bearer {access token}

In case the used access token is invalid or has been invalidated, the server responds with a Unauthorized response (HTTP Code: 401)**.** If this happens, a new access\_token must be acquired.

To manually **destroy** the currently used token, a GET request must be sent to the /logout endpoint. Future requests using the same access token as used in the logout request will then fail.

**Access Token Expiration**

Currently an access\_token has no timed automatic timed invalidation, but this might be changed in future.

Only three active tokens are allowed per user! If attempting to acquire a fourth token, the oldest one gets invalidated. (TODO: Yet to implement!)  
We are doing that to avoid bunches of „dead“ sessions in the database.

The following table shows an overview of all endpoints which regard to authorization.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Function** | **Role(s)** | **Method** | **Input Params** | **Output Data** | **endpoint** |
| Login (Create Session) | {all} | POST | *OAuth2 conform username + password MIME Type:  application/x-www-form-urlencoded* | {"access\_token": Token , "token\_type": "bearer"} | /token |
| Generate Auth Token for another user (for QR Code generation) | A, N | GET |  |  | /token/ {user\_id} |
| Logout (Destroy Session) | {all} | GET | *--* |  | /logout |
| Get currently authenticated user  (e.g. for checking login state) | A,N,S,P | GET |  | JSON Object of Type : User or *Null* when not authenticated | /me |
| Re-send registration email containing QR-Code and username [and password\*] | A, N | POST | email\* : String  (The user’s email address)  reset\_password\*\* : Boolean   (Default: False)  \* If an email address is not specified as input parameter, it will try to take it from the user dataset. If there is no ‘email’ field specified for the given user, server will respond with error 400. \*\* If reset\_password is True, a new password will be random-generated and shown to user in the mail.  If reset\_password is False, the user will not see his/her current password in the mail text, but an “(wie von Ihnen ... festgelegt)” note instead. | success = True,  if mail could be sent successfully,  success = False otherwise | /sendmail/{user\_id} |

## User management endpoints

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Function** | **Role(s)** | **Method** | **Input Params** | **Output Data** | **endpoint** |
| Create User/ Participant | A, N\* | POST | User: User *(with empty ‘uuid’)* username *or* new\_password *can be left empty for auto-generation*  *Optional extra field:* new\_password – *manually sets the password*  *Optional extra field:* email - if specified, an registration email is sent to this address,  containing username, password and permanent QR-Code | id : String  username: String | /user |
| Update User/Participant | A, N\* | PUT |  | success = True : Boolean | /user |
| Delete User/Participant | A, N\* | DELETE |  | success = True : Boolean | /user/{id} |
| Show User/Participant | A, N\*, S\*, P\*\* | GET |  | user : User | /user/{id} |
| List Users/Participants | A, N\*, S\* | GET |  | users : List<User> | /users |

\* Only query parcitipants returned

Note: If a user is created without specifying a password (so it will be auto-generated), and no e-mail specified or send\_registration\_mail set to False, the generated clear-text password will be **dismissed.** In this case Log-in is only possible by generated QR codes or after manually changing password for the user afterwards.

## Server-side data schemas and configuration endpoints

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Function** | **Role(s)** | **Method** | **Input Params** | **Output Data** | **endpoint** |
| Get field schema definitions | {all} | GET |  | schemas : List<Field > | /schemas |
| Get Enum definitions | {all} | GET |  | enums : List<Enum> | /enums |
| Get ADT definitions | {all} | GET |  | JSON Object with ADT id’s as keys and List<Field> as values. | /adts |
| Get Device Configuration data | {all} | GET |  | JSON Object with configuration key-value sets | /config |
| Get API Info  Can be used to check if this is a valid f4f server endpoint and if the API version is compatible with the client. | {all} | GET |  | {  "api\_name": "**f4f-server**" "api\_version": API\_VERSION  }  API\_VERSION is an integer value representing the API version number  (currently always 1). | /info |

## Data synchronization endpoints (Participant <-> Server)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Function** | **Role(s)** | **Method** | **Input Params** | **Output Data** | **endpoint** |
| Request new SyncProcessID | P | GET |  | sync\_id : String | /sync |
| Request Data | P | GET | *datatype : DataType (must be LabData or UserData!)*  *all : boolean*  *(optional, if true, request all data available on server, except for these, which were already requested within this sync process)*  limit : Int *(optional, limit the number of requested data sets)* | data : List<*as defined in datatype* > containing all added or modified data since last Sync, or empty Objects with ID field only (for deleted datasets) | /sync/{SyncProcessID} |
| Send Data | P | POST | *datatype : DataType (must be* SensorData *or* UserData*!)*  data : List<(as defined in datatype)> containing all added or modified data since last Sync, or empty Objects with ID field only (for deleted datasets) | identifiers : List<String>  *A list of identifiers, corresponding to each entry of dataset. (null for deleted datasets)* | /sync/{SyncProcessID} |
| Confirm finished Sync process | P | GET | *—* | success = True : Bool | /sync/{SyncProcessID}/finish |

## Data access endpoints (Server -> Supervisor)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Function** | **Role(s)** | **Method** | **Input Params** | **Output Data** | **endpoint** |
| Get information about what data is available for the specified user | S | GET |  | stats : List<Stats> | /data/stats/{user\_id} |
| ~~Request number of available datasets for the criteria specified as input params~~ | ~~S~~ | ~~GET~~ | *~~user\_id : string, datatype : RequestableDataType timestamp\_start : long, timestamp\_end : long~~*  *Note:* replaced by using *count\_only* field of *DataRequest* | ~~num\_data : long~~ | ~~/data/count~~ |
| Request data specified by the request input param | S | POST | *DataRequest* | data : List<RequestedData> *[Optional]*,  count : Int | /data/request |

Tips for Supervisors:

* Use GET /users to request a list of available user ID‘s
* User GET /user/{user\_id} to request the anonymized static data for one user

## Other Endpoints

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Function** | **Role(s)** | **Method** | **Input Params** | **Output Data** | **endpoint** |
| Post server log message | {all] | POST | *{*  *msg : String*  *anon\_key\* : String =   ‘SQD3ib67ttxvkSpln2K7cw’*  *Optional\*\*:   client\_version\*\*: Integer*  *}* | success = True : Bool | /log |

\* the *anon\_key* field must contain the specified hard-coded key String, when no authorization token is set (for anonymous logging)  
\*\* Must contain the VERSION\_CODE of the client’s Android BuildConfig. This was implemented later and is optional for Apps with Version Code < 18.

# Synchronization

We decide between two different types of synchronization processes:

1. Synchronize data between Garmin wearable device and smartphone
2. Synchronize data between smartphone and backend server

The Garmin wearable sensor, the smartphone and the backend server all have an internal memory for storing data. Most of this data is exchanged between these devices only during synchronization procedures. These Synchronization procedure are only planned for users having the role „Participant“.

Loading all data exchange between devices out to synchronization procedures brings the following two advantages:

1. Participants do not need to wear the smartphone in physical closeness to the wearable device all the time
2. Participants do not need to have their Bluetooth activated on their phones all the time
3. Participants do not need to be connected to the internet all the time
4. Participants do not need to use their mobile data transfer for transmitting the acquired data to the server. Instead, they can delay the transfer until they are connected to the internet through an unmetered WiFi connection

## Synchronization Wearable <> Smartphone

The Garmin Sync process is implemented as defined by Garmin specifications. It will be triggered implicitly by an Garmin Health SDK background service and notifies the app about when it‘s done.

The app remembers the time of last successful synchronization and informs the user about this time in the status fragment. The user should be warned, if too long time (more than 12 hours?) no synchronization with device was possible and be asked to bring charged device and Smartphone close together with having Bluetooth activated on his smartphone.

## Synchronization Smartphone <> Backend Server

Following data is synced between smartphone and backend server during the synchronization.

**From Server to Smartphone:**

* All ADT, Enum and Schema definitions\*\*
* Data entered by study nurse (LabData)\*

**From Smartphone to Server:**

* Data acquired by asking questions to participant (UserData)\*
* Data acquired from wearable device (SensorData)\*

\* This is limited to all data, which have been created or modified since the last successful data synchronization and is limited to data refering to the user, who is currently authenticated in the app

\*\* This data is also downloaded from server on every log-in or App start, if an internet connection is available. It‘s not logged as part of the synchronization process.

## StaticData

Changes in static user data (participant profile) are not part of the synchronized data. Users need to be online to modify their static data.

# Data Type Reference

## Primitive Types / Enums

The following primitive data types are used.

* **String**
* **Bool** (UI todo!)
* **Float**
* **Int**
* **Enumeration (pre-defined)**  
   pre-defined Enums:
  + **FieldType:** StringType, BoolType, FloatType, IntType, EnumType, TimeType, ListType, ADT, Container
  + **Role**: Administrator, Supervisor, Nurse, Participant
  + **Permission**: Read, Edit, AddElement, RemoveElement
  + **DataType:** UserData, LabData, StaticData, SensorData
  + **SensorDataType**: Energy, Pulse, Temperature, PulseOx, HRV, Acceleration, Respiration, StepCount, StressScore, HeartMeasurementEar
  + **RequestableDataType**: (SensorDataType Enum with additional element „UserData“)
  + **Side** : Left, Right
  + **LogInMode:** Credentials, AppGeneratedToken, EMailToken
* **Enueration (variable)**
* **Time** (= full date+time)
* **Foodlist** – *a JSON Array containing all carotinoid-relevant meals consumed on a day*
* **List<** FieldType> - *used e.g. for collecting multiple BIA measurements*

## Enumeration Types

Enumerations whose visual element representations are displayed in the UI are stored server-side. Enumerations are lists of elements, of which the user can pick one (or maybe also more than one in future scenarios).

They are defined in the *enums.json* file in the “enums” object.

### Enum sequence transitions

For acquiring some kinds of data, users will be shown a selection of elements (words or images) from which they can select one. These selections are based on server-side defined enumerations. The Android App has implemented this functionality in the *ImageEnumActivity* class.

Currently this input method is used only for prompting the consumed meals, but there could be other scenarios in the future using this input method.

If choosing an element out of more than just one enumeration is needed, it must be defined, which enum is to be presented to the user next after he/she selected an element of the currently displayed options. So a dependency chain is needed describing the order in which these lists are presented to the used. Example: *The user selected ‘Apple’ from the displayed food type. Next, he needs to be asked, how many apples he had (‘PiecesMedium’).*

The enum to show next might not only depend on which *enum* was shown before, but also what *element* the user had selected. Example: *If the user selected ‘Fruit’ as fruit type, a list of possible fruits will be shown next. But if he selected ‘Vegetable’ a list of vegetables should be displayed instead.*

To realize this, **enum transitions**can be defined. These are defined in the *enums.json* file in the “enum\_transitions” object. In this object is defined, which is the next following enum to display, depending on the current enum and the current selection. If there is no following enum defined, an input process is completed.

**Add Tags to the sequence**

Within the chain of enums, tags can be added, which can be recognized and evaluated as a condition in a later step of the chain.

Tags have to be added using a ‘+’ symbol one **value** of the transition object.

Addition of multiple tags is possible using more than one element marked with ‘+’ .

Example:

"VegetableType" :   
{  
 "pumpkin" : "VegetableSate+UseWeight"

"cucumber " : "VegetableSate+UseWeight+GreenColor"  
}

If the user selects “pumpkin”, an “UseWeight”-Tag is added to the enum sequence, which can be evaluated by following selections. If he selects “cucumber”, an additional “GreenColor” tag will be added.

See the following paragraph to understand, what an added tag is useful for.

**Recognize Tags**

Tags can be recognized by using a ‘+’ symbol in one **key** of the transition object and used as a condition for the next forwarding. The source enum this key refers to will only be considered, when the tag had been added by some other value in the chain before.

Example:

"VegetableState " :   
{  
 "\*" : "AmountVariable",

"\*+UseWeight": “AmountGramms"  
}

In this example, if the user selects any entry from the VegetableState, he will be redirected to the AmountVariable enum, except a UseWeight tag has been added before

**Order**

If multiple keys fulfill the criteria, the target enum referred by the value defined at **last** is used.

**Asterisk Placeholder**

Using “\*” as key defines a default for **all** transitions. It should be used as first key in the transition objects, so that it can be overridden by following individual definitions.

The asterisk can be combined with tag recognitions (“\*+Tag1+Tag2...”) and so only apply, when the listed tags have been defined.

**Forced termination**

In combination with the asterisk placeholder, a *null* value can be used for a certain entry to exclude it from the asterisk placeholder coverage and force the enum chain to terminate when this item is selected, even when the asterisk placeholder refers to a following enum.

### Contextual Enum definitions

The following definitions are used and keep stated in this document as long as they are not part of the server backend schema files. They will then be removed from this document, since they are actually more part of contextual data than part of the specification.

* **Scale5**: very high, high, neutral, low, very low
* **FoodType**: Gemüse, Obst, Saft, Verarbeitet, Supplement
* **VegetableType**: Tomate, Paprika, Peperoni, Kürbis, Möhre, Mais, Salat, Spinat, Grünkohl, Blattkohl
* **VegetableState**: roh, gegart, fermentiert
* **FruitType**: Zitrusfrucht, Wassermelone, Papaya, Sonstige (FARBE?)
* **JuiceType**: Fruchtsaft, Gemüsesaft, Multivitamin
* **ConvenienceType**: Ketchup, Passiert, Pizza
* **Gender**: male, female, other
* **ConsultationType**: Aufnahme, Zwischenuntersuchung, Abschlussuntersuchung, Sonstige
* **SupplementType:** ***TODO!***
* **…**.

**see** *schema/enums.json***of the backend server implementation for the complete list of defined enumerations!**

## Abstract Types

In the following subsections you find tables showing all abstract data type definitions. Each subsection title equals the type’s name.

### User (id: “user”)

|  |  |  |  |
| --- | --- | --- | --- |
| **field** | **field type** | **NULL** | **Description** |
| id | String |  | unique data set identifier |
| role | Enum (Role) |  | assigned role |
| username | String |  | If role is Participant, the chosen username must not relate to the actual participant’s name. |
| ~~email~~ | ~~String~~ | ~~X~~ | Email address will not be stored any longer due to separation of personal data. Field is still used temporarily for sending user a one-time registration email on create or update. |
| salt | String |  | server-stored password salt |
| password\_hash | String |  | server-stored SHA256 hash sum of salted client password |
| hsz\_identifier | String | X | an identifier used by the “Humanstudienzentrum” to determine each participants datasets |
| created\_by | uuid |  | the user ID of the user (nurse or admin) who initially created this user |
| creation\_date | Time |  | the date and time in UTC when this user was created |
| client\_version | Int | X | The BuildConfig.VERSION\_CODE of the Android App sent with the last server log message. (VERSION\_CODDE >= 18 only) |
| anamnesis\_data | StaticData | X | static participant data, see 5.3.5 |

### Session (backend only)

|  |  |  |  |
| --- | --- | --- | --- |
| **field** | **field type** | **NULL** | **Description** |
| id | String |  | unique data set identifier  (not the session token!) |
| username | String |  | username of the user, which this session relates to  (used for easier identification during development) |
| userid | String |  | unique identifier of the user, which this session relates to |
| ~~salt~~ | ~~String~~ |  | currently, salt is hard-coded for all hashed session tokens |
| hashed\_token | String |  | SHA256 hash sum of server-generated and client-side-stored unique session token in combination with a hard-coded server salt |
| expiration | Time | X | (not implemented yet) |
| creation\_time | Time |  | The time when this session was created. |
| last\_use | Time | X | The last time a client requested data from server using the token of this session |
| login\_mode | LogInMode |  | The mode of authentication used for creating this session |

### SyncProcess (backend only)

|  |  |  |  |
| --- | --- | --- | --- |
| **field** | **field type** | **NULL** | **Description** |
| id | String |  | unique sync process identifier |
| start\_time | Time |  | Time when this sync process was initiated  (client requested GET /sync) |
| finish\_time | Time | X | time when this syn session was successfully finished  (client requested GET /sync/{SyncProcessID}/finish ) |
| user\_id | String |  | the unique identifier of the user, who initiated and uses this sync process |
| num\_added\_*X*\* | Int | X | number of added datasets of DataType X during this sync session |
| num\_modified\_*X*\* | Int | X | number of added datasets of DataType X during this sync session |
| num\_deleted\_*X*\* | Int | X | number of removed datasets of DataType X during this sync session |

\* the “X” in these fields stand for the datatype (according to DataType enum). These fields can appear multiple times, each for each datatype.

‘finish\_time’ field being not NULL indicates that a sync process was successfully finished. ‘finish\_time’ being NULL means, the process is either currently ongoing or has been interrupted by an error (e.g. on connection loss during synchronization).

If a user requests

### UserData (id: “user\_data”)

This ADT describes a dataset for all questions of the day entered by user.

|  |  |  |  |
| --- | --- | --- | --- |
| **field** | **field type** | **NULL** | **Description** |
| id | String |  | unique identifier of this dataset. |
| last\_sync\_id | String | X | the ID of the SyncProcess when this data was synced or NULL, if data was not yet synced or when it was was modified after last sync (added by server!) |
| creation\_time | Time |  | the time when this dataset was created by user |
| modification\_time | Time | X | the time when this dataset was modified the last time |
| effective\_day | Time |  | the day which this dataset regards to |
| user\_id | String |  | The user this dataset was acquired for |
| sunlight\_time | Int | X | number of minutes in presence in sunlight |
| sports\_minutes | Int | X | number of minutes spent doing sports activities |
| bicycle\_minutes | Int | X | number of minutes spent riding bicycle |
| foodlist | Foodlist | X | list of consumed food items  (the exact options and transitions of selectable food items can be found in the *schema/enums.json* file of the implementation) |
| pss\_*N* | Scale5 |  | Question no. with from the Perceived Stress Scale (PSS)[[1]](#footnote-1) relating to the last 7 days.  User is asked for entering this data only on day one and then every 7th day of the study period. |
| subjective\_health\_1 | Scale5 |  | Answer to statement: “Ich fühle mich grundsätzlich gesund.” |
| subjective\_health\_2 | Scale5 |  | Answer to statement: “Ich fühle körperliche Schmerzen.” |
| subjective\_health\_3 | Scale5 |  | Answer to statement: “Ich fühle mich motiviert, mich zu bewegen.” |
| subjective\_health\_4 | Scale5 |  | Answer to statement: “Ich fühle mich kraftlos oder träge.” |
| subjective\_health\_5 | Scale5 |  | Answer to statement: “Ich fühle mich durch körperliche Beschwerden bei der Bewerkstelligung meines Tages eingeschränkt.” |
| subjective\_health\_6 | Scale5 |  | Answer to statement: “Ich fühle mich müde.” |

### StaticData (id: “anamnesis\_data”)

|  |  |  |  |
| --- | --- | --- | --- |
| **field** | **field type** | **NULL** | **Description** |
| gender | Enum (Gender) | X | gender |
| height | Int | X | size in centimeters |
| occupation | String | X | Occupied activity |
| weight | Int | X | Weight in kg |
| waist\_size | Int | X | “Taillenumfang“ in cm |
| hip\_size | Int | X | “Hüftumfang” in cm |
| bia\_fat\_mass | Int | X | body fat mass in g |
| bia\_musc\_mass | Int | X | body fat-free mass in g |
| bia\_phase\_angle | Int | X | phase angle in mDeg |
| general\_health\_state | Scale5 | X | Subjective general health state: very low = very bad,  very high = very good |
| study\_begin\_date | Time | X | An arbitrary point of time on the first day of the study period |
| study\_end\_date | Time | X | An arbitrary point of time on the last day of the study period |
| avg\_sleep\_start | Time | X | The time the participant regularly goes to bed. |
| avg\_sleep\_end | Time | X | The time the participant regularly wakes up. |
| wearing\_arm | Enum (Side) | X | The arm used to wear tracker device |
| birth\_year | Int | X | The year of birth |
| bia\_measurements | List<ADT(BIA-Measurement)> | X | List of Bioimpedance Measurements |
| consultations | List <ADT(Consultation)> | X | List of participant’s scheduled appointments at HSZ |

### BIA-Measurement (id: “bia\_measurement”)

|  |  |  |  |
| --- | --- | --- | --- |
| **field** | **field type** | **NULL** | **Description** |
| measurement\_day | Date |  | Date of the measurement |
| bia\_fat\_mass | Float |  | Body fat mass in grams |
| bia\_musc\_mass | Float |  | Muscle mass in grams |
| bia\_phase\_angle | Float |  | BIA Phase angle in mDeg |

### Consultation (id: “consultation”)

|  |  |  |  |
| --- | --- | --- | --- |
| **field** | **field type** | **NULL** | **Description** |
| consultation\_date | Date |  | Date of consultation |
| consultation\_time | Time |  | Time of consultation on the specified consultation date |
| consultation\_type | Enum (ConsultationType) |  | Type of appointment |
| consultation\_notes | String | X | Optional notes for the appointment |

### SensorData

To keep overhead data low, one SensorData data set can consist of multiple measurements of the same data type. For instance, one set could contain all acquired step counts since the last time of data synchronization between wearable and smartphone.

For this reason, all effective data ist typed as Lists. The same index for the different list fields refers to the same data acquisition.

|  |  |  |  |
| --- | --- | --- | --- |
| **field** | **field type** | **NULL** | **Description** |
| id | String |  | unique identifier of this dataset |
| last\_sync\_id | String | X | the unique identifier of the SyncProcess when this data was synced or NULL, if data was not yet synced or when it was was modified after last sync  (added by server!) |
| user\_id | String |  | The user this dataset was acquired for  (added by server!) |
| creation\_time | Time |  | the time when this dataset was created by user |
| modification\_time | Time | X | the time when this dataset was modified the last time |
| timestamps | List<Long> |  | Timestamp of this sensor data acquisition in milliseconds |
| time\_spans | List<Long> | X | If the values are accumulated values (e.g. counted steps), the time span in ms for which the single base values were integrated |
| type | SensorDataType |  | the type of sensor data |
| values | List <Float or Int> | X | the main values of this sensor data (maybe null only for type=Acceleration) |
| rr\_history | List<List<Float>> | X | A list of lists of previously recorded rr intervalls since the last recorded bpm value (only for type = HeartMeasurementEar) |
| ear\_position\_quality |  | X | Quality of the measured value depending on a proper in-ear sensor positioning.  Only values with a corresponding measurement\_quality > 50 should be used for evaluation !  (only for type = HeartMeasurementEar) |
| zero\_crossings | List<Float> | X | Number of Zero Crossings (only for type = Energy ) |
| x | List<Int> | X | accumulated acceleration in X direction (only for type = Acceleration) |
| y | List<Int> | X | accumulated acceleration in Y direction (only for type = Acceleration) |
| z | List<Int> | X | accumulated acceleration in Z direction (only for type = Acceleration) |

## ADT Schema, Permission and Enum Specifications

All abstract datatypes have a corresponding Schema definition.

The schema is both used to generate GUIs on front-end (App) and for data storage management on back-end (Server).

The type of one field in a schema can be set as an primitive type, as Enum (both see section 6.1) or as another ADT. The app fetches the schema definitions such as the definitions of the Enums from the server in the **App-Backend-Synchronization** process. This allows a later change of the data model, which the users uses to enter or read data without the necessity of modifying the App implementation.

**Permissions**

The specification also contains parameters for access permissions for the individual fields. These can be partially pre-defined (e.g. a participant is never allowed to manipulate Static information oneself or a supervisor is not allowed to manipulate data at all), but the permissions could also be further restricted on-the-fly in the app front-end depending on app state.

Permissions are defined in “default restrict” mode. All permissions must be explicitly granted by adding them to the permissions list. An exception is the *Administrator* role, who has all permissions by default. Permissions *AddElement* and *RemoveElement* are only applicable when datatype is *List*.

The *Edit* permission implicitly covers the *Read* permission!

**Field Containers**

Fields can be separated into groups by adding fields with the datatype “Container”. Every container field added to an ADT will create a new view container for the upcoming fields. Containers can have optional “label” (used as container title) or “helpText” (used as container description) parameters, whose contents are displayed before showing the fields.

If the very first field in an ADT is not a container, a *default container* without label or helpText is created implicitly containing all fields or containing all fields until the end or the next container field.

The only purpose of container fields is the grouped display of form fields. They do not have editable content and will not be stored in the dataset.

Also containers need a permission to be displayed. If the permission is not granted, fields following the container field will be attached to the previous or default container.

Containers not containing any read-accessible fields are not displayed.

**Overview**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Type** | **field** | **field type** | **NULL** | **Description** |
| **ADT** | id | String |  | ADT Identifier |
| fields | List<Field> | X | List of fields this ADT consists of (see below) |
| **Field** | id | String |  | Field identifier |
| label | String |  | Field label |
| helpText | String | X | Additional description text |
| datatype | Enum (FieldType) |  | Field type |
| elements\_type | Enum (FieldType) | X | Type of the elements, if datatype is *ListType* |
| adt\_enum\_id | String | X | - Enum or ADT identifier of this field, if datatype is Enum or ADT  - Enum or ADT identifier of the list elements Enum or ADT, if datatype is *ListType* and elements\_type is Enum or ADT |
| defaultValue | *as defined in  datatype* | X | Default value for new datasets |
| minValue | Float | X | Minimum allowed value  (only for Int or Float types) if not specified, platform limits will be used |
| maxValue | Float | X | Maximum allowed value  (only for Int or Float types) if not specified, platform limits will be used |
| useSlider | Bool | X | if true, value is selected via Slider instead of an text input field |
| sliderMinLabel | String | X | Label text at the maximum of slider |
| sliderMaxLabel | String | X | Label text at the minimum of slider |
| sliderStepSize | Float | x | If neither 0 or null, slider lets choose discrete values with the specified step size |
| unitString | String | X | specifies the physical unit of the field value |
| qrCodeInput | Bool | X | Shows a button, which allows Input via scanned QR/Barcode. Only for *StringType, FloatType, IntType* and no *unitString* specified. |
| displayPeriodDays | Int | X | This field only takes effect when following requisites are met:   * the user currently signed on has a valid date specified for [‘anamnesis\_data’][‘study\_begin\_date’] * the current dataset used for building the input form has a field ‘effective\_date’ with a valid value set greater or equal to ‘study\_begin\_date’   Setting this field to an int value *n* will display this field in a generated form only, if the ‘effective\_date’ field is set to a day, which is a multiple of *n* days started counting from study\_begin\_date or started counting the day after study\_begin\_date, if displayDayOne is set. |
| displayDayOne | Bool | X | This field only takes effect when the same requisites are met as specified for displayPeriodDays.  Setting this field to true will also show the field when the ‘errective\_date’ refers to same day as ‘study\_begin\_date’ |
| maybeNull | Bool | X | if true, this field does not need to have a value |
| permissions | List<Permission> | X | list of role-based permissions  (see below) |
| **Per-mis-sion** | type | Enum(Permission) |  |  |
| role | Enum(Role) or “all” |  |  |
| **Enum** | id | String |  | Enum identifier |
| label | String | X | Enum title |
| helpText | String | X | Help text for enum input |
| element\_ids | List<String> |  | Enum element identifiers |
| element\_  labels | List<String> |  | Desciription to be displayed corresponding to elements |
| element\_image\_urls | List<String> | X | Relative Server URL where an image file corresponding to this entry can be found. Single entries may be null. Clients will cache the images on server sync. They will only be re-downloaded when the local device cache is cleared or the URL entry has changed! (Just substituting image files is not sufficient) |
| contains\_food\_items | Bool | X | Must be set, if this enum **contains at least one element**, which is a final food element (not a [sub-]category, not an amount or state, etc.).  The app will add an additional “food\_id” to the food consumption object containing the value of the selected element\_id of the last enum element in the enum transition chain, which has the contains\_food\_items flag set (if any).  This way each consumed food can uniformly be identified by the “food\_id” field without the need to model all the possible enum variants in the later analyses. |

## Types specific for Data access

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Type** | **field** | **field type** | **NULL** | **Description** |
| **Stats** | user\_id | string |  | The user id this information belongs to |
| datatype | Enum (Requestable- DataType ) |  | The type of data described by this Stats |
| timestamp\_first | Int |  | UNIX Timestamp of the first available dataset |
| timestamp\_last | Int |  | UNIX Timestamp of the most recent available dataset |
| num\_datasets | Int |  | amount of all available datasets |
| **Data- Request** | user\_id | string OR List<String>\* |  | The user id(s) of the users for which data is requested |
| datatype | Enum (Requestable-DataType) OR List<Enum (Requestable-DataType)>\* |  | The type(s) of data to be requested. |
| timestamp\_first | Int |  | UNIX Timestamp of minimum date of the data to be requested |
| timestamp\_last | Int |  | UNIX Timestamp of maximum date of the data to be requested |
| count\_only | Bool | X | if set to true, request will only return the number of matched datasets, not the datasets themselves |
| skip | Int | X | the number of leading datasets to skip for the request (0, if unspecified ) |
| limit | Int | X | the maximum number of returned datasets (all, if unspecified) |
| filter\_duplicates | Bool | X | if set, duplicate values are filtered out before sending the data. This reduces performance significantly. |
| **Requested- Data** | user\_id | string |  | The user\_id of the user the data in the data field was recorded from |
| datatype | Enum (Requestable-DataType) |  | The type of the data in the data field |
| data | *either* SensorData,  *or* List<UserData> *depending on* datatype |  | requested datasets |
|  |  |  |  |  |

\* If a list of users or datatypes is set, the data for the individual users and individual datatypes will each be separated into different RequestedData objects in the server response

1. https://www.northottawawellnessfoundation.org/wp-content/uploads/2018/04/PerceivedStressScale.pdf [↑](#footnote-ref-1)