

WAP-3-080

Device Edge Data Design

Version 1.1
9th January 2025

Date	Change	Author	Description	Version
13 th August 2024	Creation	Hao Nguyen	Document creation	1.0
9 th January 2025	Modification	Tom Hao	Update sequence of session data logs	1.1

1. Table of Contents

4.	Session data logs	3
1.1	Data synchronization flow	3
1.2	Data structure in the form of packages	7
1.3	Data structure on the SQLite	9
2.	User session information	10
2.1	Action type when user using session	10
2.2	Data structure on SQLite	11
3.	Mobile Healthy data logs	11
3.1	Data store flow	11
3.2	Data structure on the SQLite	12
4.	Size of SQLite after 7 days storage	12

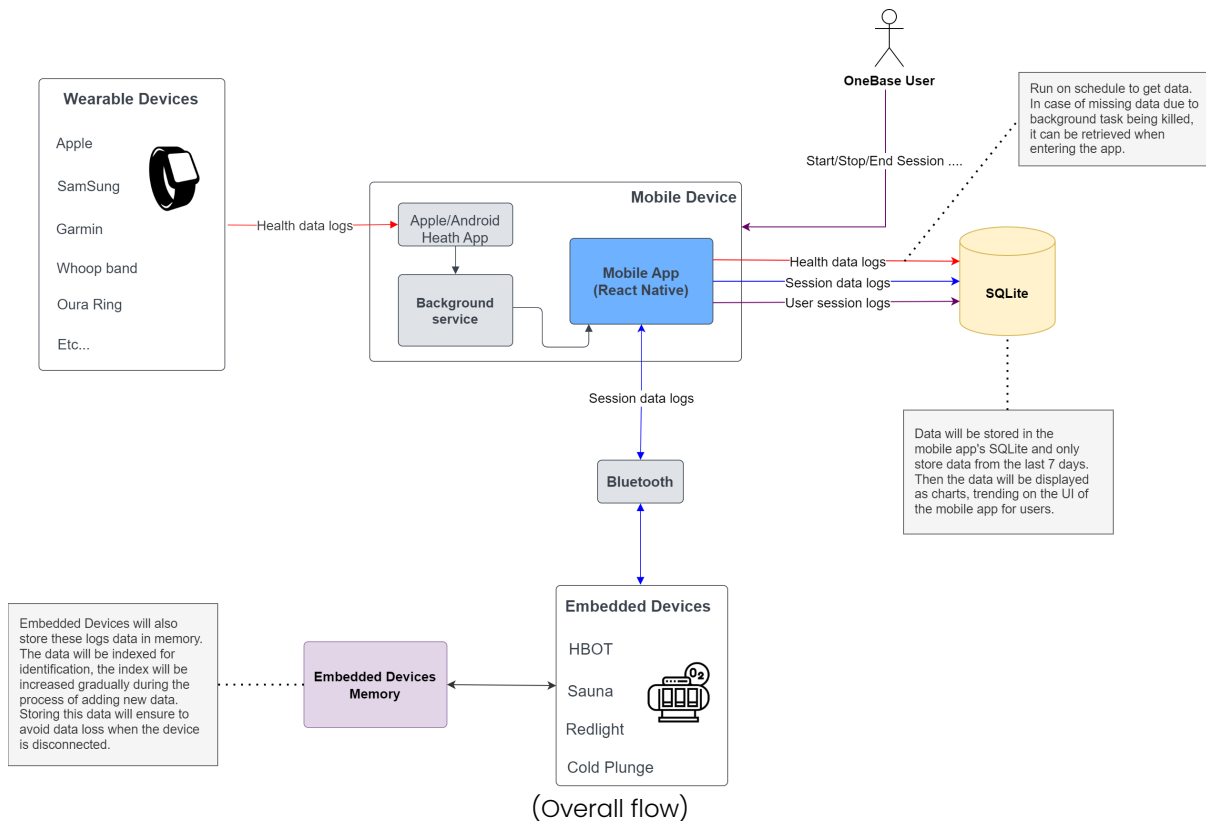
2. Scope

The document describes in detail the types of data that will be stored during a session as well as the data obtained from the Health App of the mobile device. In addition, it will design a flow to synchronize data from Embedded Device to the Mobile App via Bluetooth.

3. Details

OneBase will aggregate data related to user health as well as Embedded Device parameters on and off session usage. The data is divided into three main sources:

- **Session Data Logs:** This includes data from Embedded Device (HBOT, Sauna, Red Light, etc.) during session usage, such as on/off status, temperature, pressure, etc.
- **User Session Information:** This includes data on user actions during session usage, such as session parameters, timestamps for start, close, skip, end, etc.
- **Mobile Health Data Logs:** This includes health information from Apple HealthKit and Android HealthKit.

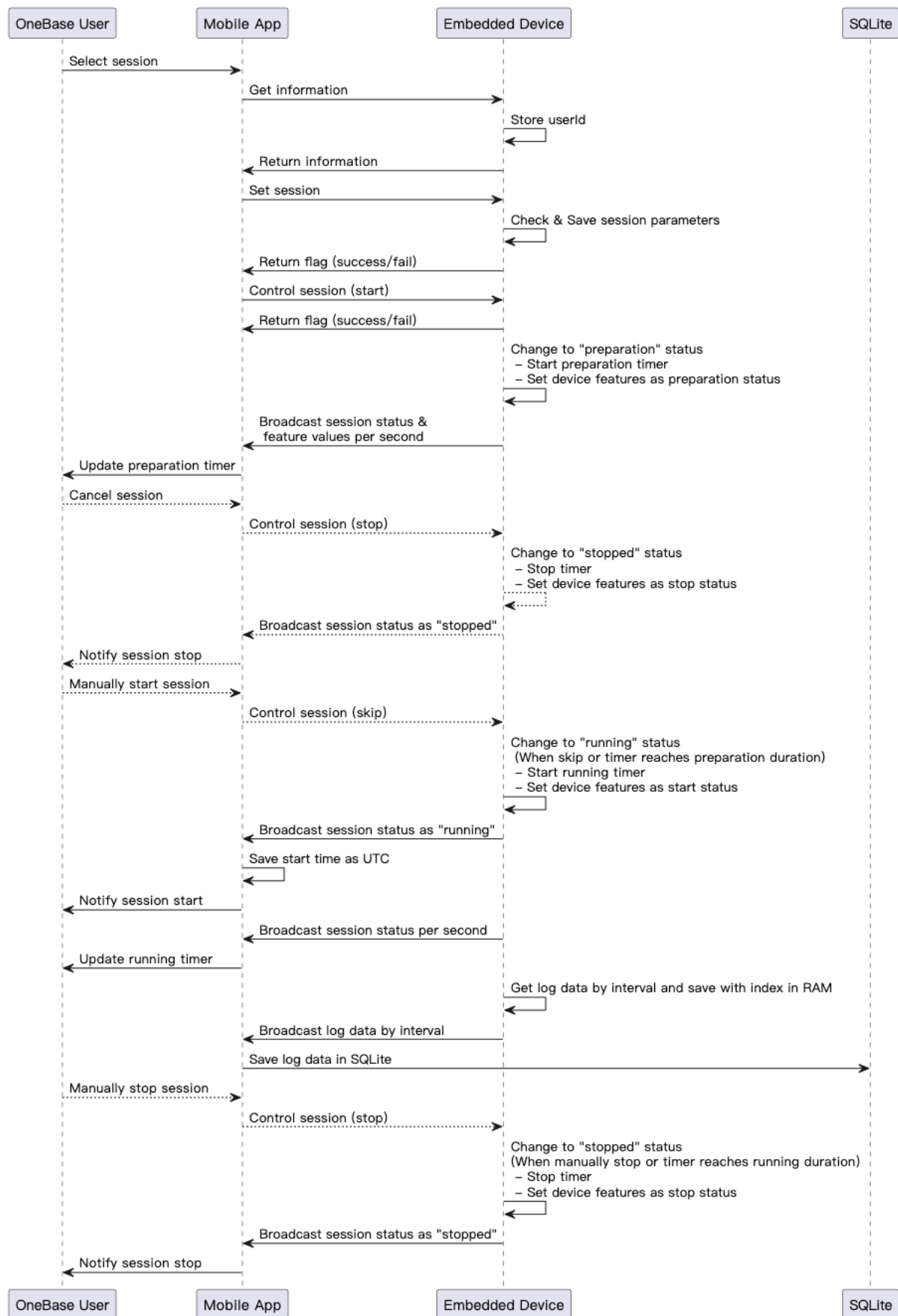


This data will be collected during session usage or on a schedule (for data from Mobile Health App). Data will be stored in the mobile app's SQLite and only store data from the last 7 days. Then the data will be displayed as charts, trending on the UI of the mobile app for users.

4. Session data logs

1.1 Data synchronization flow

During the session, Embedded Device will continuously get parameters from Physical devices (HBOT, Sauna, RedLight, ...) and send them to the mobile device via Bluetooth. In addition, this data will also be stored in the memory of Embedded Device as an array of data identified by indexes. Storing data in Embedded Device will ensure that data is not lost if the device is disconnected during session usage.



(Data synchronization flow)

- When a user selects a session to start, the mobile app will prompt user to scan and connect an Embedded Device with selected device model. After connected, mobile app will send a "Get information" request to Embedded Device to get device information. The request will include userId to identify the user. Embedded Device will store this value as global variables. After that, Embedded Device can return information to mobile app such as hardware and software versions, etc...
- The mobile app will also send a "Set session" request with the sessionId and session parameters attached. Embedded Device will check the session parameters. If session parameters fail to pass the check, Embedded Device will send a failure response with a reason code. If parameters are acceptable, Embedded Device will prepare for the session start, such as deleting session data logs in memory if old data still exists.
- A session has two stages: preparation and running, both duration time and device status for the two stages are set in session parameters. When user starts the session, Embedded Device will set physical device as preparation status, start a preparation timer, and start broadcasting "session status" every one second to app. The session status will contain a timer value in seconds, app should use this value for the timer on UI.
- During preparation, the user can choose to cancel the session, the mobile app will send a "Control session" request with "cancel" action to Embedded Device. The Embedded Device will stop the preparation timer and set the physical device to stop status.
- During preparation, the user can choose to skip it and start running session right away, the mobile app will send a "Control session" request with "skip" action to Embedded Device.
- When preparation is skipped or the preparation duration ended, the Embedded Device will set physical device as start status and start a running timer, the broadcasting of session status keep going. In addition, Embedded Device will start broadcasting logs data with an interval set in session parameters, also store these logs data in memory. The data will be indexed for identification, the index will be increased gradually during the process of adding new data. Storing this data will ensure to avoid data loss when the device is disconnected.
- In addition, when starting a session, we will save the start time on the mobile app. It will correspond to the record with index = 0. Based on the index, interval and start time, we can calculate the time for the next records.

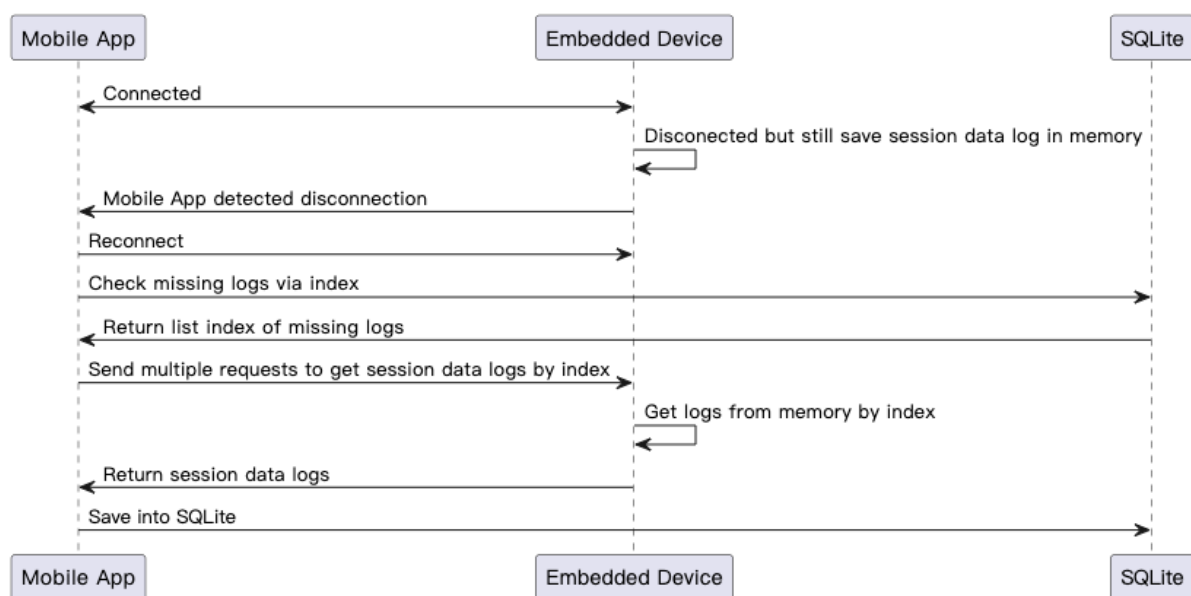
Index	Buffer
0	<Buffer 48 65 6c 6c 6f 2c 20 72 61 77 20 62 75 66 66 65 72 21>
1	<Buffer 48 65 6c 6c 6f 2c 20 72 61 77 20 62 75 66 66 65 72 21>
2	<Buffer 48 65 6c 6c 6f 2c 20 72 61 77 20 62 75 66 66 65 72 21>

(Example of Logs data structure on memory of Embedded Device)

- Mobile app will store logs data in SQLite. To avoid SQLite being full after long-term use, data will only be stored within the last 7 days. Data in SQLite will also be stored with an index corresponding to the data stored in Embedded Device. This will help the mobile app know which data is corrupted during the process of sending logs

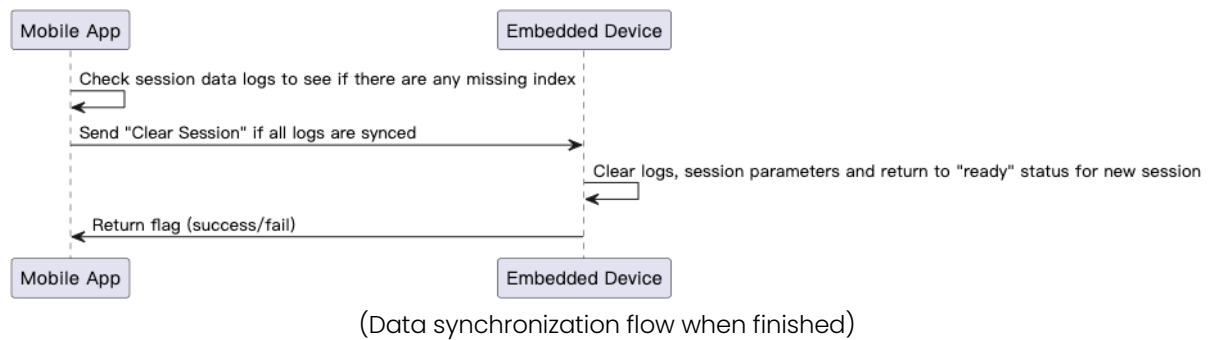
from Embedded Device to the mobile app, has not been stored and can resend a new request to retrieve that data.

- During session running, the user can choose to stop it, the mobile app will send a "Control session" request with "stop" action to Embedded Device. When a session is stopped or the running duration ends, Embedded Device will set the physical device to stop status and stop recording and sending data logs to the mobile app.
- When the session running timer ends, the Embedded Device will set the physical device to stop status automatically and send session status as "stopped" to app.
- During the use of the device, there may be a problem of disconnection, which will lead to the Embedded Device not being able to send logs to the Mobile App. Storing data in Embedded Device will help the mobile app to retrieve lost logs after reconnecting to the Embedded Device.



(Data synchronization flow when disconnect)

- When the connection is lost, Embedded Device continue to store logs data in memory. When reconnected, Embedded Device will continue to send session status and logs data automatically to the mobile app. However, the mobile app will also check which logs are missing based on the index. Send a request to get the missing logs information according to the index to Embedded Device and proceed to save the logs as usual.
- The mobile app can send a single request with a starting index and length of logs it wants, but due to packet size limitation of Bluetooth, the Embedded Device will send a response for each index of the logs data, so this could be a one-to-many request/response cycle.
- After all data logs are synced, the mobile app will send a "Clear session" request to Embedded Device to clear session data and parameters on the device and make it "ready" for new session.



1.2 Data structure in the form of packages

From connecting the device to starting and closing the session, we will have the main functions to communicate between the mobile app and Embedded Device such as:

- Get information

This function will be called when the user connects the device via Bluetooth. It will help the mobile app to get more detailed information from the device.

Input	Output
userId: number	chipId: string hardwareVersion: number firmwareVersion: number firmwareRevision: number firmwareBuild: number

Note: current firmware has a one-byte productId field for historical reason, which is not actually used. Since now we use two-byte deviceCode for device model, we can ignore productId. The firmware may change in future to include deviceCode in the output of this call. Currently the app can get deviceCode from advertising data.

- Set session

This function will be called when the user selects a session. Embedded Device will calculate if there would be enough space to store the logs data according to logInterval, logFeatures and runningDuration, it will respond with a fail flag along with a reason code if no enough space or there's other issues in the parameters.

Input	Output
sessionId: number, generated by app, unique by user sessionParameters: <ul style="list-style-type: none"> - preparationDuration*: in seconds - runningDuration: in seconds - logInterval: in seconds - logFeatures[]: an array of featureIds to log during the session - preparationStatus[]: an array of multiple feature id and values <ul style="list-style-type: none"> - featureId: number - featureValue: number boolean string 	Flag: success/fail Reason: number

<ul style="list-style-type: none"> - sessionId: an array of multiple time and features(at least two items: start and stop) <ul style="list-style-type: none"> - time: in seconds (0 for start; runningDuration for stop) - features[]: an array of multiple feature id and values <ul style="list-style-type: none"> - featureId: number - featureValue: number boolean string 	
---	--

* This parameter is different from device to device. For example, with HBOT it's the pressurization duration, and with Sauna it's the pre-heat duration, with Cold Plunge and Red Light it's a simple delay time.

- Get session

Call this function to get current sessionId and session parameters set by other user. This would be useful if a user want to join other user for a session.

Input	Output
empty	The same as input of "Set session", empty if no session configured.

- Control session

This function will be called when user want to start/skip/stop a session.

=> The "Control session" function with "Start" action can be called right after "Set session".

Input	Output
sessionId: number action: enum 0 - start(from preparation) 1 - skip(skip preparation, start the session timer and logging) 2 - stop(either in preparation or running mode)	Flag: success/fail

- Session status

Input	Output
empty	sessionId: number sessionId: enum 0 - ready(not configured), 1 - configured(have a session, but not started), 2 - preparation, 3 - running, 4 - stopped sessionTime: in seconds(preparation time in preparation mode, running time in running mode) logIndex: the latest index number of session log

Note: when session is in preparation or running status, session status will be sent from Embedded Devices to mobile app automatically every 1 second, with sessionTime changing, the app should use this value for timer displayed in UI.

- Session log (Automatically from Embedded Device side)

During session usage, logs will be automatically sent to mobile app from Embedded Device by interval set in session parameters. The features contained in log data are also set in session parameters.

Input	Output
	sessionId: number logIndex: number features[:] - featureId: number - featureValue: number boolean string

- Get logs (Manual from mobile App side)

Mobile app can proactively fetch logs if it detects missing logs based on index. The mobile app will send a request with the start index and length of logs and Embedded Device will return a packet for each index.

Input	Output
sessionId: number index: number length: number	The same with "Session log"

- Clear session

After successfully retrieving logs from Embedded Device and storing completed, the mobile app will send a request to delete everything about this session and the Embedded Device will return to a state ready for a new session.

Input	Output
sessionId: number	Flag: success/fail

1.3 Data structure on the SQLite

Data stored in SQLite will only be stored for the last 7 days. Data from more than 7 days ago will be automatically deleted when the app starts. We can base on the createdAt field to delete expired data.

Table SessionDataLogs:

Column name	Column type	Description
id	INTEGER PK	
index	INTEGER	Index corresponds to the index in memory of Embedded Device
sessionId	INTEGER	
data	TEXT(json) { humidity: 10, co2: 10, noise: 10, }	SQLite does not have a json data type, so we will store data in TEXT format. This is the information that Embedded Device returns

	tvoc: 10, pm: 10, airPressure: 10, o2: 10, o3: 10, hcho: 10, measuredAirTemperature: 10 }	during the session such as temperature, pressure, etc...corresponding to each device.
type	INTEGER	0 => Automatic 1 => Manual Note that this log data is automatically returned from the Embedded Device side or called up from the mobile app side.
createdAt	INTEGER (timestamp)	Logging time, use this to delete expired data

2. User session information

2.1 Action type when user using session

During the user's session, the user will have different actions, we will store these actions as logs. Data will be stored mainly based on user actions through mobile apps.

#	Action name	Action data logs	Note
1	Select Session	sessionId: App generated session number: DB index user configuration: <ul style="list-style-type: none"> - preparationDuration - runningDuration - temperature/pressure Timestamp	Available sessions list are pulled from database, each one should have a number as index in database. Detailed configuration of this session could be fetched from database with this number.
2	Start Prepare Session	Timestamp	Only for Sauna and HBOT Sauna: Start Pre-heat HBOT: Start Pressurizing
3	Skip Prepare Session	Timestamp	Only for Sauna and HBOT
4	Cancel Session		During the prepare session, the user can press cancel session.
5	Completed Prepare Session	Timestamp	When the user waits for the session preparation process to finish
6	Start using the device	Timestamp	Once the preparation is complete, the user starts using the actual devices

7	Stop Session	Timestamp	When the user is using a session, the usage time has not ended but they press stop
8	End Session	Timestamp	When the usage time is up and the user presses to end the session, or it automatically ends after a period.

2.2 Data structure on SQLite

Data stored in SQLite will only be stored for the last 7 days. Data from more than 7 days ago will be automatically deleted when the app starts. We can base on the createdAt field to delete expired data.

Table UserSessionInformations

Column name	Column type	Description
id	INTEGER PK	
sessionId	INTEGER	
eventName	VARCHAR	Start Session Start Prepare Session Cancel Session Stop Session ...
data	TEXT(json)	
createdAt	INTEGER(timestamp)	Logging time

3. Mobile Healthy data logs

3.1 Data store flow

During the process of users using Embedded Device, to know exactly the changes of users, we will collect more information about users' health through mobile app health applications. These apps will collect data through Wearable devices like Apple Watch, Garmin, etc...

For each different operating system we will have different apps and different libraries. For example, iOS will use Apple Health, Android will use Android Health.

Data is divided into two main types: data collected when the user uses the session and data collected when not in use.

Metrics	Type	Units	Expected Sampling Rate	Presentation
Heart Rate	In session	bpm	10 minutes	
O2 saturation	In session	%	10 minutes	
Respiratory Rate	In session	bpm	10 minutes	
Body temperature	In session	°C	10 minutes	

Heart Rate Variability (HRV)	Out of session	ms	Hourly	Current value; 3 day trend
Resting heart rate (RHR)	Out of session	bpm	Hourly	Current value; 3 day trend
Active Energy	Out of session	kJ	Hourly	24 hour bar
Sleep	Out of session	Hour	Daily	24 hour bar
Exercise	Out of session	Hour	Daily	24 hour bar

This data will be scheduled to be retrieved, by minute, hour or day. We will use a background library to create a schedule task to get data according to a pre-set schedule for each data type.

However, for the iOS operating system, the operating system can automatically kill these background tasks. To avoid this, we will create an additional mechanism when the user enters the app, it will check the stored data, and if the data is missing for a certain period, it will be retrieved.

3.2 Data structure on the SQLite

Data stored in SQLite will only be stored for the last 7 days. Data from more than 7 days ago will be automatically deleted when the app starts. We can base on the createdAt field to delete expired data.

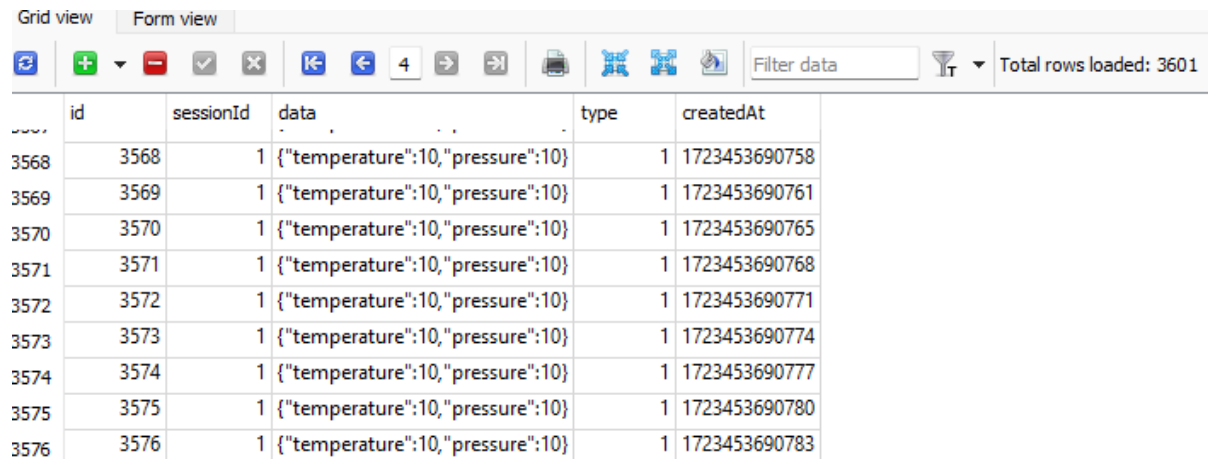
Table HealthDataLogs

Column name	Column type	Description
id	INTEGER PK	
sessionId	INTEGER	
metric	VARCHAR	Heart Rate O2 saturation Respiratory Rate ...
value	VARCHAR	
unit	VARCHAR	bpm % ...
type	INTEGER	0 => Automatic 1 => Manual Note that this log data is automatically returned from background task or get when user enter app
createdAt	INTEGER (timestamp)	Logging time

4. Size of SQLite after 7 days storage

We will test with 1 user who will undertake 4 sessions per day and continuously for 7 days, each session will run within 1 hour. We will check with the SessionDataLogs table because this table will be continuously written during the session. On average, 1 new record will be created every second according to the logic above. Within 1 hour, we will create $60 * 60 = 3600$ records.

We created an auto-run script to write 3600 records into SQLite:



	id	sessionId	data	type	createdAt
3568	3568	1	{"temperature":10,"pressure":10}	1	1723453690758
3569	3569	1	{"temperature":10,"pressure":10}	1	1723453690761
3570	3570	1	{"temperature":10,"pressure":10}	1	1723453690765
3571	3571	1	{"temperature":10,"pressure":10}	1	1723453690768
3572	3572	1	{"temperature":10,"pressure":10}	1	1723453690771
3573	3573	1	{"temperature":10,"pressure":10}	1	1723453690774
3574	3574	1	{"temperature":10,"pressure":10}	1	1723453690777
3575	3575	1	{"temperature":10,"pressure":10}	1	1723453690780
3576	3576	1	{"temperature":10,"pressure":10}	1	1723453690783

After measurement, the size of the SQLite file will be 188KB for a 60-minute session.

So, the total size of SQLite after 7 days will be: $188 * 4 * 7 = 5264\text{KB} \sim 5.1\text{MB}$. This size is not too large for a mobile application.

For the data in the HealthDataLogs and UserSessionInformations tables, I will not list them because the data in these two tables is relatively small compared to the SessionDataLogs.