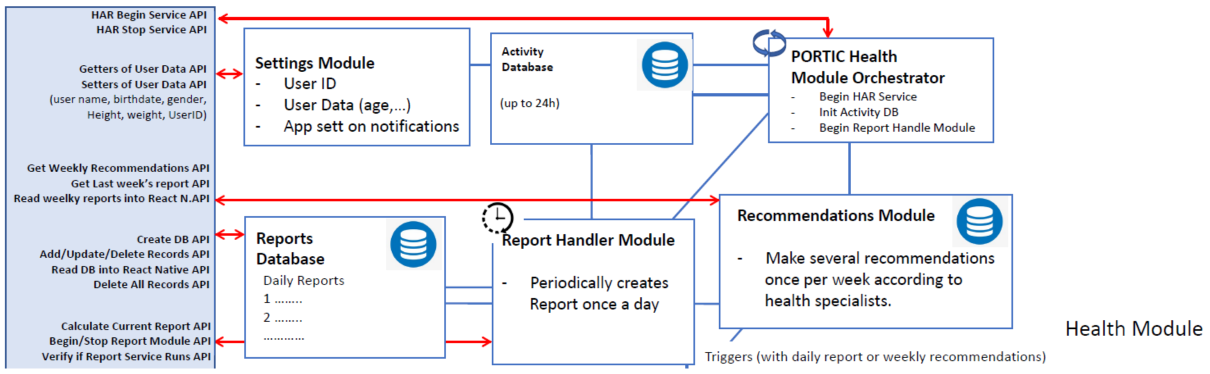
PORTIC Health module – Technical Report

Project: TECH
Date: 2022.01.12
Author: Dr. Eng. Francisco Xavier dos Santos Fonseca
N.º OE: 84598
Version: 1

Index

1. Overall Architecture Description

The overall architecture of the PORTIC health module for the TECH project can be found below:



HAR Begin Service API
HAR Stop Service API
Getters of User Data API
Setters of User Data API
(user name, birthdate, gender, Height, weight, UserID)
Get Weekly Recommendations API
Get Last week's report API
Read weelky reports into React N.API
Create DB API
Add/Update/Delete Records API
Read DB into React Native API
Delete All Records API
Calculate Current Report API
Begin/Stop Report Module API
Verify if Report Service Runs API
Settings Module
- User ID
- User Data (age,...)
- App sett on notifications
Activity Database
(up to 24h)
PORTIC Health Module Orchestrator
- Begin HAR Service
- Init Activity DB
- Begin Report Handle Module
Reports Database
Daily Reports
1 ....
2 ....
....
Report Handler Module
- Periodically creates Report once a day
Recommendations Module
- Make several recommendations once per week according to health specialists.
Health Module
Triggers (with daily report or weekly recommendations)

Figure 1: Overall software architecture of the PORTIC health module for the TECH project.

Figure 1 shows several modules that implement specific functionality required to provide health services to the AMaaS application of the TECH project. The blue-filled rectangle on the left-hand side refers to the overall public APIs that are exposed to the host application (the AMaaS application from the TECH project).

The "**Settings Module**" refers to the module responsible for the storage of information that is common and required across the AMaaS application, particularly on the user's details. The "**PORTIC Health Module Orchestrator**" is the module responsible for beginning the recognition of the physical activity that the user does from the background, including the GPS coordinates of the phone, and stores all this information in the "**Activity Database**" module. The PORTIC Health Module is also responsible for the scheduling of the "**Report Handler Module**", which oversees the creation of a recurrent daily report about the physical activity of the user. The Report Handler Module stores the daily reports in the "**Reports Database**". Once a week, the Report Handler Module produces a weekly report and a set of recommendations through the "**Recommendations Module**". These recommendations can then be used by the overall application to for e.g., visually show the recommendations to the user, or trigger the play module.

2. How to run the PORTIC health module

To run the PORTIC health module in a host application (AMaaS), the following steps must be executed, in order:

Step 0: Compiling and running

The context of the modules should be passed to the PORTIC health module. This is done with the following instructions in code:

For the context, put the following lines of code inside the **MainActivity.java**, to set the same activity context of the host application in the background services.

```
context = this;
HARModuleManager.getInstance(context);
ReportModuleManager.getInstance(context);
```

In the **build.gradle** file of the project, the following instruction needs to be included:

```
dependencies {
    classpath 'io.realm:realm-gradle-plugin:+'  //realmDB
}
```

In the **build.gradle** file for the application, the following code needs to be added:

```
apply plugin: 'realm-android'             // realmDB

implementation 'com.google.android.gms:play-services-location:18.0.0' //HAR
implementation 'com.google.code.gson:gson:2.8.6'
implementation 'com.fasterxml.jackson.core:jackson-databind:+'
implementation 'com.facebook.android:facebook-android-sdk:+'
```

For the PORTIC health module to be successfully integrated with the host application, the **AndroidManifest.xml** must include the following changes:

Permissions:

```
<uses-permission
android:name="com.google.android.gms.permission.ACTIVITY_RECOGNITION" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
```

Broadcast services within Main Application activity:

```
<receiver

android:name="pt.portic.tech.modules.HARModule.BackgroundServicesRestarter"
    android:enabled="true"
    android:exported="true">
    <intent-filter>
        <action android:name="Restart_AMaaS_Service" />
    </intent-filter>
</receiver>

<receiver
android:name="pt.portic.tech.modules.ReportHandlerModule.ReportAlarm"
    android:enabled="true"
    android:exported="true">
    <intent-filter>
        <action
            android:name="Report_Handler_Alarm"/>
    </intent-filter>
</receiver>
```

Services:

```
<service
android:name="pt.portic.tech.modules.HARModule.DetectedActivitiesIntentServ
ice"
    android:exported="false" />

<service
android:name="pt.portic.tech.modules.HARModule.BackgroundDetectedActivities
Service"
    android:enabled="true"/>
```

Step 1: User Registration

The PORTIC health module uses the user ID across all the databases used to store the generated data. This user ID should be provided to the user shared preferences (the settings module from Figure 1) BEFORE the PORTIC health module orchestrator is called. Parallel to this, if the cardiorespiratory risk of the user is not low, then this also must be set before the PORTIC health module is set to begin (even though it can be changed at any time without impact on performance).

Regarding the User ID:

```
UserProfileManager.getInstance().Set_User_ID("…");
```
Where the argument passed is a String.

Optionally, regarding the activity risk level:

```
UserProfileManager.getInstance().Set_Health_Activity_Risk(1);
```
Replace the argument with another risk level (1 -> low; 2 -> moderate; 3 -> high).

Step 2: Begin PORTIC Health Module Orchestrator

To start the PORTIC health module orchestrator, use the following instruction **after step 1**:

```
HARModuleManager.getInstance().HAR_Begin_Service();
```

Step 3: Double-check permissions of location and of execution in battery mode

Depending on the Smartphone, this step will likely be in different menu settings. Checking location permissions:

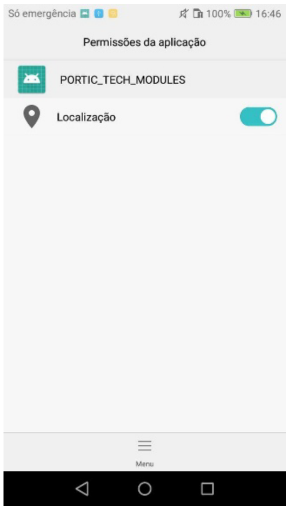
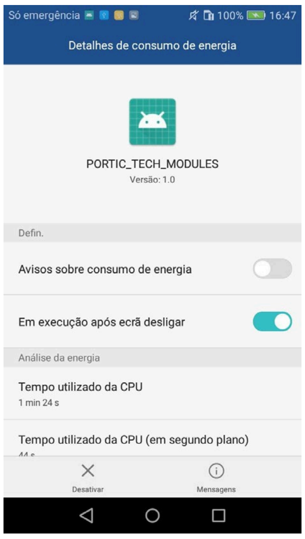


Figure 2: Location permission.

Go to system Settings -> Applications -> PORTIC_TECH_MODULES -> Permissions

Guarantee that the permission is given.

Checking permission of extensive execution under battery:


Só emergência 100% 16:47

Detalhes de consumo de energia

PORTIC_TECH_MODULES
Versão: 1.0

Defin.

Avisos sobre consumo de energia

Em execução após ecrã desligar

Análise da energia

Tempo utilizado da CPU
1 min 24 s

Tempo utilizado da CPU (em segundo plano)

Figure 3: Permissions to run foreground services while screen is turned off.

Go to system Settings -> Applications -> PORTIC_TECH_MODULES -> Battery

And give permission to execute after screen turns off. If this option is grayed out, it is likely because you opened the application while charging your phone and that it was never run while on battery. Unplug the smartphone, open the host application, and then revisit these settings a bit later. You will be able to change it then.

[Optional Step]: Stopping the PORTIC health module

```
HARModuleManager.getInstance().HAR_Stop_Service();
```

3. Overall Structure of the Software of the PORTIC health module

Figure 2 shows the organization of the software within the PORTIC health module, between public APIs, packages, and classes implementing the logic.

The **Public_API_UserProfile_Module** is the public interface implementing the module “Settings Module” from Figure 1, and is implemented in the package “UserProfile” shown in Figure 2.

The **Public_API_ReportManager_Module** is the public interface implementing the module “Report Handler Module” from Figure 1, and is implemented in the package “ReportHandlerModule” shown in Figure 2.

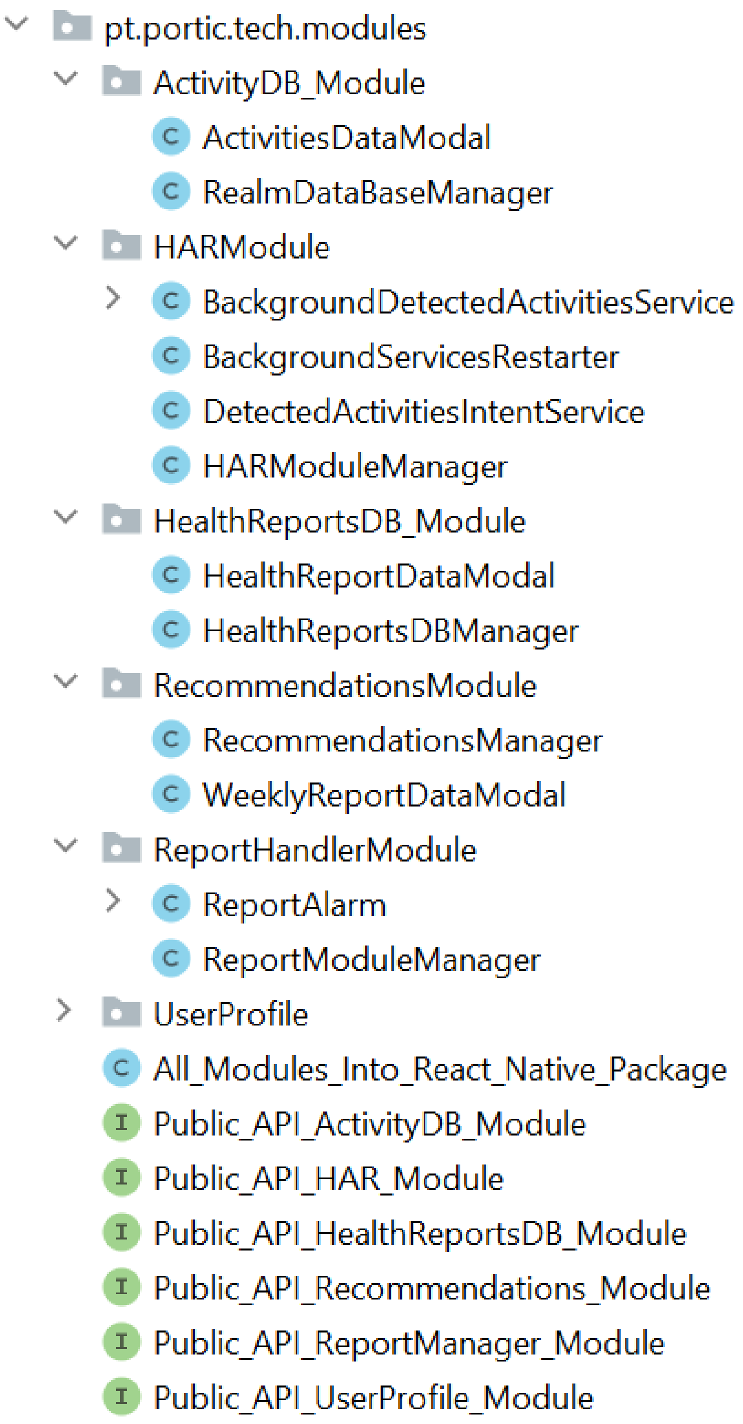
The **Public_API_Recommendations_Module** is the public interface implementing the module “Recommendations Module” from Figure 1, and is implemented in the package “RecommendationsModule” shown in Figure 2.

The **Public_API_HealthReportsDB_Module** is the public interface implementing the module “Reports Database” from Figure 1, and is implemented in the package “HealthReportsDB_Module” shown in Figure 2.

The **Public_API_HAR_Module** is the public interface implementing the module “PORTIC Health Module Orchestrator” from Figure 1, and is implemented in the package “HARModule” shown in Figure 2.

The **Public_API_ActivityDB_Module** is the public interface implementing the module “Activity Database” from Figure 1, and is implemented in the package “ActivityDB_Module” shown in Figure 2.

The package **pt.portic.tech.modules** is the overall native package containing all the mentioned interfaces and packages that implement the PORTIC Health Module, and that, together with specific code in the AndroidManifest.xml file, must be included in the native AMaaS solution of the TECH project to provide the negotiated health services.



```
pt.portic.tech.modules
    ActivityDB_Module
        ActivitiesDataModal
        RealmDataBaseManager
    HARModule
        BackgroundDetectedActivitiesService
        BackgroundServicesRestarter
        DetectedActivitiesIntentService
        HARModuleManager
    HealthReportsDB_Module
        HealthReportDataModal
        HealthReportsDBManager
    RecommendationsModule
        RecommendationsManager
        WeeklyReportDataModal
    ReportHandlerModule
        ReportAlarm
        ReportModuleManager
    UserProfile
    All_Modules_Into_React_Native_Package
    Public_API_ActivityDB_Module
    Public_API_HAR_Module
    Public_API_HealthReportsDB_Module
    Public_API_Recommendations_Module
    Public_API_ReportManager_Module
    Public_API_UserProfile_Module
```

Figure 4: Overall structure of the PORTIC Health Module for the TECH project. Classes in green are the public APIs; grey folders are packages; blue icons are classes.

4. API Description

Public_API_UserProfile_Module

`void Set_User_ID(String id);`
This method sets the user id in the application as shared preferences. Required in all internal databases to associate the data collected to the user' registry.

`void Set_User_Name(String name);`
This method sets the user name in the application as shared preferences.

void Set_User_BirthDate(String date);

This method sets the user birthdate in the application as shared preferences. The format of the string is YYYYMMDD (and will be converted into and stored as LONG number).

`void Set_User_Gender(String gender);`
This method sets the user gender in the application as shared preferences. Possible value: M / F.

`void Set_User_Height(String height);`
This method sets the user height in the application as shared preferences. Stored as FLOAT, format: 1.75.

`void Set_User_Weight(String weight);`
This method sets the user weight in the application as shared preferences. Stored as FLOAT, format: 65.1.

`void Set_Health_Activity_Risk(String risk);`
This method sets the user health activity risk in the application as shared preferences. Stored as INTEGER, and should be either 1 (Low risk), 2 (Moderate risk) or 3 (High risk). Default value is 1.

void Set_Date_Of_Last_Weekly_Report(String date);

This method sets the date of the last weekly report in the application as shared preferences. This is to avoid duplication of weekly reports, and count when 7 days have passed to generate a new weekly report. Stored as String in the format dd-mm-yyyy.

`Map<String, ?> Get_User_ALL_Data();`
This method gets all the shared preferences of the application, stored as String.

`String Get_User_ID();`
This method gets the user id from the application in the shared preferences. Stored as STRING.

`String Get_User_Name();`
This method gets the user name from the application in the shared preferences. Stored as STRING.

`Long Get_User_BirthDate();`
This method gets the user birthdate from the application in the shared preferences. The format of the string is YYYYMMDD (and should be converted from STRING in the format YYYYMMDD into a LONG number).

String Get_User_Gender();

This method gets the user gender from the application in the shared preferences. Stored as STRING, in the format M/F.

Float Get_User_Height();

This method gets the user height from the application in the shared preferences. Stored as FLOAT, format 1.75.

Float Get_User_Weight();

This method gets the user weight from the application in the shared preferences. Stored as FLOAT, format 75.1.

`Integer Get_Health_Activity_Risk();`
This method gets the user's health activity risk from the application in the shared preferences. Stored as INTEGER, and should be either 1 (Low risk), 2 (Moderate risk) or 3 (High risk). Default value is 1.

`String Get_Date_Of_Last_Weekly_Report();`
This method gets the date of the last weekly report in the application as shared preferences. This is to avoid duplication of weekly reports, and count when 7 days have passed to generate a new weekly report. Stored as String in the format dd-mm-yyyy.

Public_API_ReportManager_Module

`void Begin_Report_Handler_Module();`
This method plants a planned intent service to be run every day at a convenient time after 20H. The planned intent service is of the type ReportAlarm, which will produce a daily health report from the sensed activities through the operating system. The report is then stored in the database, and, if at the time there are 7 days between the last weekly report and that current day, the weekly report is also produced. Every time a weekly report is produced, all the activities stored in the database are deleted (thus, once a week, the activities database is cleaned to release space). When the PORTIC health module is started, this method is also called.

void Stop_Report_Handler_Module();

This method stops and removes the planted service in the Begin_Report_Handler_Module() method.

`void CalculateCurrentReport();`
When called, this method will immediately produce a daily report from whichever activities are stored in the database. BE AWARE: if at the time there are 7 days between the last weekly report and that current day, the weekly report is also produced, which WILL DELETE all the activities stored in the database.

void VerifyIfReportServiceIsRunning();

When executed, this function verifies if the report service is running. If it is, it leaves things as they are. if it is not, schedules it with Begin_Report_Handler_Module(). You can call this to verify if the report service is running at will, because if it's running then nothing will occur. If it is not, the service will be scheduled.

Public_API_Recommendations_Module

void ProduceWeeklyRecommendations();

This method produces the weekly report from the available daily reports. It is called from the CalculateCurrentReport() in the report manager module (i.e., everytime the weekly report should be produced).

RealmList<String> GetWeeklyRecommendations(WeeklyReportDataModal weeklyReport);

This method gets the list of recommendations that were given to a specific weekly report. It returns a List of Strings, each String being a recommendation based on the past week's activities and sedentary habits.

```
RealmList<String> GetWeeklyRecommendations();
```
This method is the same as above, but since a weekly report is not passed as argument, this method will always fetch the last available weekly report from the Database, and from which the list of recommendations are extracted.

```
WeeklyReportDataModal GetLastWeeklyReport();
```
This method returns the most recent weekly report available in case it exists. Otherwise, it returns null.

WeeklyReportDataModal GetWeeklyReport(int index);

This method returns the weekly report from the position passed as parameter index. Index goes from 0 to the number of reports in the database. Example: if you pass index 0, it will get you the most recent report; with index = 1, you will get the before last index; index = 2, you'll get the weekly report from three weeks ago. It returns the weekly report from the solicited position, or null if it does not exist.

```
void ReadAllWeeklyReportsFromDBIntoReactNative(Callback successCallback)
throws JSONException;
```
This method sends all the weekly reports back into React Native, as a WritableMap.

Public_API_HealthReportsDB_Module

```
void CreateDB (Context context);
```
This method is used to set up the database used for the daily reports, under the name "HealthReports.db".

```
void AddDataToDB(String userID, int amountTimeStillHours,
        int amountTimeStillMinute, int amountTimeStillSeconds,
        int amountTimeInVehicleHours, int amountTimeInVehicleMinute,
        int amountTimeInVehicleSeconds, int amountTimeWalkingHours,
        int amountTimeWalkingMinute, int amountTimeWalkingSeconds,
        int amountTimeRunningHours, int amountTimeRunningMinute,
        int amountTimeRunningSeconds, int amountTimeOnBicycleHours,
        int amountTimeOnBicycleMinute, int amountTimeOnBicycleSeconds,
        int totalAmountSedentaryHours, int totalAmountSedentaryMinutes,
        int totalAmountActiveActivityInMinutes, double metsTotais,
        double metsIntBaixa,double metsIntModerada, double metsIntVigorosa,
        int metsIntBaixaPercentage, int metsIntModeradaPercentage,
        int metsIntVigorosaPercentage, int etsIntBaixaBicyclePercentage,
        int metsIntBaixaWalkingPercentage,
        int metsIntBaixaRunningPercentage,
        int metsIntModeradaBicyclePercentage,
        int metsIntModeradaWalkingPercentage,
        int metsIntModeradaRunningPercentage,
        int metsIntVigorosaBicyclePercentage,
        int metsIntVigorosaWalkingPercentage,
        int metsIntVigorosaRunningPercentage,
        long amountTimeStillMilliseconds,
        long amountTimmeInVehicleMilliseconds,
        long amountTimeWalkingMilliseconds,
        long amountTimeRunningMilliseconds,
        long amountTimeOnBicycleMilliseconds,
        long totalAmountSedentaryMilliseconds,
        long totalAmountActiveActivityMilliseconds,
        double distanceWalking, double distanceRunning,
        double distanceBicycle);
```
This method adds a daily report to the database.

```
List<HealthReportDataModal> ReadAllDataFromDB();
```
This method returns all the daily reports from the database, as a list of reports from the type HealthReportDataModal (format defined at a section below).

```
HealthReportDataModal ReadLastRecordFromDB();
```
This method returns the last produced daily report (the most recent).

```
void ReadAllDataFromDBIntoReactNative(Callback successCallback) throws
JSONException;
```
This method reads all daily health reports from the database and passes them back into React Native through a WritableMap.

```
void UpdateDataInDB(int position, String userID,
        int amountTimeStillHours,int amountTimeStillMinute,
        int amountTimeStillSeconds,int amountTimeInVehicleHours,
         int amountTimeInVehicleMinute,int amountTimeInVehicleSeconds,
        int amountTimeWalkingHours,int amountTimeWalkingMinute,
        int amountTimeWalkingSeconds, int amountTimeRunningHours,
        int amountTimeRunningMinute, int amountTimeRunningSeconds,
         int amountTimeOnBicycleHours,int amountTimeOnBicycleMinute,
        int amountTimeOnBicycleSeconds, int totalAmountSedentaryHours,
         int totalAmountSedentaryMinutes,
        int totalAmountActiveActivityInMinutes, double metsTotais,
        double metsIntBaixa,double metsIntModerada, double metsIntVigorosa,
        int metsIntBaixaPercentage, int metsIntModeradaPercentage,
        int metsIntVigorosaPercentage,
        int metsIntBaixaBicyclePercentage,
        int metsIntBaixaWalkingPercentage,
        int metsIntBaixaRunningPercentage,
        int metsIntModeradaBicyclePercentage,
        int metsIntModeradaWalkingPercentage,
        int metsIntModeradaRunningPercentage,
        int metsIntVigorosaBicyclePercentage,
        int metsIntVigorosaWalkingPercentage,
         int metsIntVigorosaRunningPercentage,
        long amountTimeStillMilliseconds,
        long amountTimmeInVehicleMilliseconds,
        long amountTimeWalkingMilliseconds,
        long amountTimeRunningMilliseconds,
        long amountTimeOnBicycleMilliseconds,
        long totalAmountSedentaryMilliseconds,
        long totalAmountActiveActivityMilliseconds,
        double distanceWalking, double distanceRunning,
        double distanceBicycle, Date date);
```
This method updates a daily report at a given position, if it exists.

```
void DeleteRecordFromDB(int position);
```
This method deletes a daily health report from the database, at the position passed as parameter.

```
void DeleteAllRecordsFromDB();
```
This method deletes all daily health reports from the database. Use with care.

Public_API_HAR_Module

boolean HAR_Begin_Service();

This method is responsible for starting the entire PORTIC health module. By calling it, it will be initiated (1) the human activity recognition module, (2) the GPS sensing, (3) the daily report handler module, and (4) the weekly recommendations module. This will be permanently executed as a foreground native service on Android, and will display a fixed notification to the user in the system.

boolean HAR_Stop_Service();

This method will revert everything that the method HAR_Begin_Service() did, by permanently stopping the foreground service of the PORTIC health module, with all its pending intents and scheduled alarms. The permanent system's notification will disappear from the system.

```
void SetNightTimePeriodToIgnoreActivity(int timeA, int timeB);
```
This method will set the period of time where still activities will be ignored, so to not account for the sleeping period in the sedentary lifestyle. By default, timeA is 22h, and timeB is 09H, meaning that the resting time is defined between 22h-09h. Within this resting period, active activities (walking, running, cycling,) and the sedentary activity in vehicle will count for the produced reports, but the still activity will not be logged in (which may probably indicate sleeping time during this period).

Public_API_ActivityDB_Module

public void CreateDB (Context context);

This method sets the database to be used to store the recognised activities by the system..

public void AddDataToDB(String userID, String timeStamp, int activityType, String activityDescription, int confidence, double lat, double lon);

This method adds a recognised activity into the database specified in the CreateDB (Context context) method.

```
public List<ActivitiesDataModal> ReadAllDataFromDB();
```
This method reads all the activities stored in the database and returns a list containing all the records (stored as the formatting class `ActivitiesDataModal`). The formatting class can be consulted below.

public ActivitiesDataModal ReadLastRecordFromDB();

This method returns the last activity stored in the database (last, as in, most recent).

public void UpdateDataInDB(int position, String userID, String timeStamp, int activityType, String activityDescription, int confidence, double lat, double lon);

This method updates the activity stored in the database at the position specified in the first argument. Example: UpdateDataInDB(5, "98765", "2021-11-19 18:29:07.018",3,"STILL", 100);

```
void DeleteRecordFromDBActivitiesDataModal(int position);
```
This method deletes a stored activity from the database in the specified position.

```
void DeleteRecordFromDBHealthReportDataModal(int position);
```
This method deletes a stored daily health record from the database in the specified position.

```
void DeleteRecordFromDBWeeklyReportDataModal(int position);
```

This method deletes a stored weekly report and its weekly recommendations from the database in the specified position.

```
void DeleteAllRecordsFromDB();
This method deletes all records from the activities database, the daily
reports database, and the weekly reports + recommendations database. All
internal databases from the PORTIC health module are wiped clean.
```

```
void DeleteAllActivityRecordsFromDB();
```
This method deletes all stored activities from the database. The activities database is wiped clean with this method.

5. The format of the database (the class objects used to store data at a NoSQL RealmDB database)

The PORTIC health module has 3 internal databases (one for the recognized activities, one for the daily reports, and one for the weekly reports with the recommendations based on that week), each of which with its own format. The format is specified with java classes, shown as follows:

1 – Database for the recognized activities

Type	Name	Description
long	id	Primary key of the database.
String	userID	Unique registry ID of the user.
String	timestamp	Date and time that the activity was registered.
int	activityType	Type of the registered activity (0 -> "IN_VEHICLE"; 1 -> "ON_BICYCLE"; 2 -> "ON_FOOT"; 3 -> "STILL"; 4 -> "UNKNOWN"; 5 -> "TILTING"; 7 -> "WALKING"; and 8 -> "RUNNING")
String	activityDescription	Description of the registered activity depending on the type (0 -> "IN_VEHICLE"; 1 -> "ON_BICYCLE"; 2 -> "ON_FOOT"; 3 ->

		"STILL"; 4 -> "UNKNOWN"; 5 -> "TILTING"; 7 -> "WALKING"; and 8 -> "RUNNING")
int	confidence	Degree of certainty of the recognised activity in percentage (always equal or above 75%).
double	latitude	Latitude GPS coordinate of the recognised activity.
double	longitude	Longitude GPS coordinate of the recognised activity.

2 – Database for the daily reports

Type	Name	Description
long	id	Primary key.
String	userID	Unique registration id of the user.
long	amountTimeStillMilliseconds	Total amount of time spent still in milliseconds.
long	amountTimeInVehicleMilliseconds	Total amount of time spent in a vehicle in milliseconds.
long	amountTimeWalkingMilliseconds	Total amount of time walking, in milliseconds.
long	amountTimeRunningMilliseconds	Total amount of time running, in milliseconds.
long	amountTimeOnBicycleMilliseconds	Total amount of time on a bicycle, in milliseconds.
long	totalAmountSedentaryMilliseconds	Total amount of time on a sedentary activity (still, plus in a vehicle), in milliseconds.
long	totalAmountActiveActivityMilliseconds	Total amount of time spent in active activities (walking, running, on a bicycle), in milliseconds.
int	amountTimeStillHours	Total amount of time spent still, in hours.
int	amountTimeStillMinute	Total amount of time spent still in minutes (to be used together with the amountTimeStillHours).
int	amountTimeStillSeconds	Total amount of time spent still, in seconds (to be used together with amountTimeStillHours, and amountTimeStillMinute).
int	amountTimeInVehicleHours	Total amount of time in vehicle, in hours.
int	amountTimeInVehicleMinute	Total amount of time in vehicle, in minutes (to be used together with amountTimeInVehicleHours).
int	amountTimeInVehicleSeconds	Total amount of time in vehicle, in seconds (to be used together with amountTimeInVehicleHours, and amountTimeInVehicleMinute).
int	amountTimeWalkingHours	Total amount of time walking, in hours.
int	amountTimeWalkingMinute	Total amount of time walking, in minutes (to be used together with amountTimeWalkingHours).
int	amountTimeWalkingSeconds	Total amount of time walking in seconds (to be used together with amountTimeWalkingHours and amountTimeWalkingMinute).
int	amountTimeRunningHours	Total amount of time running, in hours.
int	amountTimeRunningMinute	Total amount of time running, in minutes (to be used together with amountTimeRunningHours).
int	amountTimeRunningSeconds	Total amount of time spent running, in seconds (to be used together with amountTimeRunningHours and amountTimeRunningMinute).
int	amountTimeOnBicycleHours	Total amount of time spent on a bicycle in hours.
int	amountTimeOnBicycleMinute	Total amount of time spent on a bicycle, in minutes (to be used together with amountTimeOnBicycleHours).
int	amountTimeOnBicycleSeconds	Total amount of time spent on a bicycle, in seconds (to be used together with amountTimeOnBicycleHours and amountTimeOnBicycleMinute).
int	totalAmountSedentaryHours	Total amount of time in sedentary activities (still + in vehicle) in hours.
int	totalAmountSedentaryMinutes	Total amount of time in sedentary activities (still + in vehiche) in minutes (to be used together with totalAmountSedentaryHours).
int	totalAmountActiveActivityInMinutes	Total amount of time spent in active activities (running + walking + bicycle) in minutes.
double	metsTotais	Total amount of METs for the day (above 1.6 METs/minute).
double	metsIntBaixa	Total amount of METs for the day of low intensity ([1.6;2.9]).
double	metsIntModerada	Total amount of METs for the day of moderate intensity ([3;5.9]).
double	metsIntVigorosa	Total amount of METs for the day of vigorous intensity ([6; [).
int	metsIntBaixaPercentage	Of total METs, the percentagem of METs of low intensity.
int	metsIntModeradaPercentage	Of total METs, the percentagem of METs of moderate intensity.
int	metsIntVigorosaPercentage	Of total METs, the percentagem of METs of vigorous intensity.

int	metsIntBaixaBicyclePercentage	Of total METs of low intensity,the amount achieved with a bicycle.
int	metsIntBaixaWalkingPercentage	Of total METs of low intensity,the amount achieved walking.
int	metsIntBaixaRunningPercentage	Of total METs of low intensity,the amount achieved running.
int	metsIntModeradaBicyclePercentage	Of total METs of moderate intensity, the amount achieved on a bicycle.
int	metsIntModeradaWalkingPercentage	Of total METs of moderate intensity, the amount achieved walking.
int	metsIntModeradaRunningPercentage	Of total METs of moderate intensity,the amount achieved running.
int	metsIntVigorosaBicyclePercentage	Of total METs of vigorous intensity,the amount achieved on a bicycle.
int	metsIntVigorosaWalkingPercentage	Of total METs of vigorous intensity,the amount achieved walking.
int	metsIntVigorosaRunningPercentage	Of total METs of vigorous intensity,the amount achieved running.
double	distanceWalking	Total distance walked, in metros.
double	distanceRunning	Total distance ran, in metros.
double	distanceBicycle	Total distance on a bicycle, in metros.
String	dateOfReport	Date that the report was produced

3 – Database for the weekly reports and recommendations

Type	Name	Description
long	id	Primary key.
String	userID	Unique registration id of the user.
long	amountTimeStillMilliseconds	Total amount of time spent still in milliseconds.
long	amountTimeInVehicleMilliseconds	Total amount of time spent in a vehicle in milliseconds.
long	amountTimeWalkingMilliseconds	Total amount of time walking, in milliseconds.
long	amountTimeRunningMilliseconds	Total amount of time running, in milliseconds.
long	amountTimeOnBicycleMilliseconds	Total amount of time on a bicycle, in milliseconds.
long	totalAmountSedentaryMilliseconds	Total amount of time on a sedentary activity (still, plus in a vehicle), in milliseconds.
long	totalAmountActiveActivityMilliseconds	Total amount of time spent in active activities (walking, running, on a bicycle), in milliseconds.
int	amountTimeStillHours	Total amount of time spent still, in hours.
int	amountTimeStillMinute	Total amount of time spent still in minutes (to be used together with the amountTimeStillHours).
int	amountTimeStillSeconds	Total amount of time spent still, in seconds (to be used together with amountTimeStillHours, and amountTimeStillMinute).
int	amountTimeInVehicleHours	Total amount of time in vehicle, in hours.
int	amountTimeInVehicleMinute	Total amount of time in vehicle, in minutes (to be used together with amountTimeInVehicleHours).
int	amountTimeInVehicleSeconds	Total amount of time in vehicle, in seconds (to be used together with amountTimeInVehicleHours, and amountTimeInVehicleMinute).
int	amountTimeWalkingHours	Total amount of time walking, in hours.
int	amountTimeWalkingMinute	Total amount of time walking, in minutes (to be used together with amountTimeWalkingHours).
int	amountTimeWalkingSeconds	Total amount of time walking in seconds (to be used together with amountTimeWalkingHours and amountTimeWalkingMinute).
int	amountTimeRunningHours	Total amount of time running, in hours.
int	amountTimeRunningMinute	Total amount of time running, in minutes (to be used together with amountTimeRunningHours).
int	amountTimeRunningSeconds	Total amount of time spent running, in seconds (to be used together with amountTimeRunningHours and amountTimeRunningMinute).
int	amountTimeOnBicycleHours	Total amount of time spent on a bicycle in hours.
int	amountTimeOnBicycleMinute	Total amount of time spent on a bicycle, in minutes (to be used together with amountTimeOnBicycleHours).
int	amountTimeOnBicycleSeconds	Total amount of time spent on a bicycle, in seconds (to be used together with amountTimeOnBicycleHours and amountTimeOnBicycleMinute).

int	`totalAmountSedentaryHours`	Total amount of time in sedentary activities (still + in vehicle) in hours.
int	`totalAmountSedentaryMinutes`	Total amount of time in sedentary activities (still + in vehiche) in minutes (to be used together with `totalAmountSedentaryHours`).
int	`totalAmountActiveActivityInMinutes`	Total amount of time spent in active activities (running + walking + bicycle) in minutes.
double	`metsTotais`	Total amount of METs for the day (above 1.6 METs/minute).
double	`metsIntBaixa`	Total amount of METs for the day of low intensity ([1.6;2.9]).
double	`metsIntModerada`	Total amount of METs for the day of moderate intensity ([3;5.9]).
double	`metsIntVigorosa`	Total amount of METs for the day of vigorous intensity ([6; [).
int	`metsIntBaixaPercentage`	Of total METs, the percentagem of METs of low intensity.
int	`metsIntModeradaPercentage`	Of total METs, the percentagem of METs of moderate intensity.
int	`metsIntVigorosaPercentage`	Of total METs, the percentagem of METs of vigorous intensity.
int	`metsIntBaixaBicyclePercentage`	Of total METs of low intensity,the amount achieved with a bicycle.
int	`metsIntBaixaWalkingPercentage`	Of total METs of low intensity,the amount achieved walking.
int	`metsIntBaixaRunningPercentage`	Of total METs of low intensity,the amount achieved running.
int	`metsIntModeradaBicyclePercentage`	Of total METs of moderate intensity, the amount achieved on a bicycle.
int	`metsIntModeradaWalkingPercentage`	Of total METs of moderate intensity, the amount achieved walking.
int	`metsIntModeradaRunningPercentage`	Of total METs of moderate intensity,the amount achieved running.
int	`metsIntVigorosaBicyclePercentage`	Of total METs of vigorous intensity,the amount achieved on a bicycle.
int	`metsIntVigorosaWalkingPercentage`	Of total METs of vigorous intensity,the amount achieved walking.
int	`metsIntVigorosaRunningPercentage`	Of total METs of vigorous intensity,the amount achieved running.
double	`distanceWalking`	Total distance walked, in metros.
double	`distanceRunning`	Total distance ran, in metros.
double	`distanceBicycle`	Total distance on a bicycle, in metros.
String	`dateOfReport`	Date that the report was produced
RealmList <String>	`recommendations`	List of strings, each of which being a recommendation.