D4.13

Libraries for remote and D2D HR exchange - V2

ABSTRACT

This deliverable describes a demonstration of the second version of the libraries offered by the InteropEHRate Framework as a reference implementation of the device-to-device (D2D) and the remote-to-device (R2D) health record exchange protocols. It also outlines for these libraries their description regarding their current version and the used licences, as well as specific information regarding their development, followed by specific guidelines regarding the installation and the usage of these libraries, through a detailed guide.

Delivery Date	8th April 2021
Work Package	WP4
Task	T4.5
Dissemination Level	Public
Type of Deliverable	Demonstrator
Lead partner	UPRC

InteropEHRate project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826106

This document has been produced in the context of the InteropEHRate Project which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826106. All information provided in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose.

CONTRIBUTORS

	Name	Partner
Contributors	Thanos Kiourtis, Argyro Mavrogiorgou, Charis Tsigkounis	UPRC
Contributors	Alessio Graziani	ENG
Contributors	Marti Marot	A7
Contributors	Nicu Jalba	SIVECO
Contributors	Chrysostomos Symvoulidis	BYTE
Reviewers	N/A	N/A

LOG TABLE

Version	Date	Change	Author	Partner
0.1	2021-01-21	Provided initial ToC	Thanos Kiourtis, Argyro Mavrogiorgou	UPRC
0.2	2021-03-10	Provided input for the S-EHR side and HCP-side D2D Libraries	Thanos Kiourtis, Charis Tsigkounis, Martin Marot, Nicu Jalba	UPRC, A7, SIVECO/SIMAVI
0.3	2021-03-19	Finalized Section 1 and Section 2	Thanos Kiourtis, Argyro Mavrogiorgou	UPRC
0.4	2021-03-24	Provided input for the R2D Access Library	Alessio Graziani	ENG
0.5	2021-03-26	Provided input for the R2D Backup and R2D Emergency Libraries	Chrysostomos Symvoulidis	BYTE
0.6	2021-03-26	Sent for quality check	Thanos Kiourtis, Argyro	UPRC

			Mavrogiorgou	
1.0	2021-03-31	Quality check	Argyro Mavrogiorgou	UPRC
VFinal	2021-04-07	Final check and submission	Laura Pucci	ENG

ACRONYMS

Acronym	Term and definition
D2D	Device-to-Device
R2D	Remote-to-Device
HCP	Healthcare Practitioner
HR	Health Record
S-HER	Smart Electronic Health Record
FHIR	Fast Healthcare Interoperability Resources
SW	Software
IDE	Integrated Development Environment

TABLE OF CONTENT

LIST OF TABLES

1. INTRODUCTION

1.1. Scope of the document

The main goal of this document is to deliver a demonstration of the second version of the libraries offered by the InteropEHRate Framework as a reference implementation of the device-to-device (D2D) and the remote-to-device (R2D) health record exchange protocols. In more detail, the current document outlines for these libraries their description regarding their current version and the used licences, as well as specific information regarding their development (e.g. programming languages, supported platforms, etc). Moreover, this document includes specific guidelines regarding the installation and the usage of these libraries, through a detailed guide. It should be noted that this document is a SW accompany report, a pointer to the actual deliverable, regarding the design of the libraries for remote and D2D HR exchange [D4.5].

1.2.Intended audience

The current document is mainly intended for developers and manufacturers who are interested in designing and building either S-EHR applications or HCP applications, and desire to exploit and reuse either the D2D or the R2D or both these two functionalities offered by the InteropEHRate framework, in the context of their applications. Apart from that, the document is intended for researchers and developers as well, as they may be interested in installing and using the designed libraries.

1.3.Structure of the document

The current document is organized in the following Sections:

- Section 1 (current section) introduces the overall concept of the document, defining its scope, intended audience, and relation to the other project tasks and reports.
- Section 2 outlines the description of the software regarding the offered libraries, including details such as their licences, their programming languages, and their supported platforms.
- Section 3 describes for each library an installation guide, as well as a user guide.

1.4.Updates with respect to previous version (if any)

With regards to the previous version of the deliverable (D4.12 Libraries for remote and D2D HR exchange - V2 [D4.12]), the following changes have been performed for each separate section of the current document.

Section 1 - Introduction:

- Updated the scope of the document

Section 2 - SW DESCRIPTION:

- Section D2D Libraries
  - Updated the SW version of the S-EHR side D2D library.
  - Updated the SW version of the HCP side D2D library.
- Section R2D Libraries
  - Updated the SW version of the MR2D library.

Section 3 - OVERVIEW:

- Section D2D Libraries
  - S-EHR side D2D library:
    - Added the Nexus repository to publish and receive versioned applications as well as the way that the S-EHR side library is uploaded on Nexus.
    - The version of the S-EHR side D2D library has been updated to the current version.
    - Updated the operations of the public interface `D2DHRExchangeListeners`.
    - Updated the type of the files that the operations of `ConnectedThread` class can send.
  - HCP side D2D library:
    - Added the Nexus repository to publish and receive versioned applications as well as the way that the HCP side library is uploaded on Nexus and the integration steps.
    - Updated the main flow implemented in HCP app using the D2D library.
    - Updated the functionalities of `CurrentD2DConnection` class.
- Section R2D Libraries
  - R2D Access library:
    - Added support for the download of Laboratory Reports.
    - Added support for the download of Diagnostic Reports with embedded medical images.
    - Added support for the download of Diagnostic Reports referring to a DICOM study.
  - R2D Backup library:
    - This section is a new section that has been provided to D4.13
  - R2D Emergency library:
    - This section is a new section that has been provided to D4.13

2. SW DESCRIPTION

2.1. D2D Libraries

2.1.1. S-EHR side D2D Library

SW TITLE	Mobile D2D HR Exchange (M-D2D-E)
SW VERSION	0.3.0
LICENCES AND PATENTS	Apache License
PROGRAMMING LANGUAGES	Java SE 8.0
SUPPORTED PLATFORM(s)	Android (5.0 - Current Version)
SOURCE CODE	http://iehrgitlab.ds.unipi.gr/interopehrate/s-ehr-mobile-app/d2d-hr-exchange
EXECUTABLE	N.A.

Table 1 - S-EHR side D2D Library description

2.1.2. HCP side D2D Library

SW TITLE	Terminal D2D HR Exchange (T-D2D-E)

SW VERSION	0.3.0
LICENCES AND PATENTS	Apache License
PROGRAMMING LANGUAGES	Java SE 8.0
SUPPORTED PLATFORM(s)	Windows OS
SOURCE CODE	http://iehrgitlab.ds.unipi.gr/interopehrate/reference-hcp-app/terminal-d2d-hr-exchange
EXECUTABLE	N.A.

Table 2 - HCP side D2D Library description

2.2. R2D Libraries

2.2.1. R2D Access Library

SW TITLE	Mobile R2D Exchange (M-R2D-E)
SW VERSION	0.3.0
LICENCES AND PATENTS	Apache License
PROGRAMMING LANGUAGES	Java SE 8.0

SUPPORTED PLATFORM(s)	Android (from API Level 14 – 15, named Ice Cream Sandwich)
SOURCE CODE	http://iehrgitlab.ds.unipi.gr/interopehrate/s-ehr-mobile-app/r2d-hr-exchange
EXECUTABLE	N.A.

Table 3 - R2D Access Library description

2.2.2. R2D Backup Library

SW TITLE	R2D Backup
SW VERSION	0.1.0
LICENCES AND PATENTS	Apache License
PROGRAMMING LANGUAGES	Java SE 8.0
SUPPORTED PLATFORM(s)	Android
SOURCE CODE	http://iehrgitlab.ds.unipi.gr/interopehrate/s-ehr-mobile-app/r2d-backup
EXECUTABLE	N.A. (imported in a project as a library)

Table 4 - R2D Backup Library description

2.2.3. R2D Emergency Library

SW TITLE	R2D Emergency
SW VERSION	0.0.1
LICENCES AND PATENTS	Apache License
PROGRAMMING LANGUAGES	Java SE 8.0
SUPPORTED PLATFORM(s)	Any Device that can run Java applications
SOURCE CODE	http://iehrgitlab.ds.unipi.gr/interopehrate/reference-hcp-app/r2d-emergency
EXECUTABLE	N.A. (imported in a project as a library)

Table 5 - R2D Emergency Library description

3. OVERVIEW

3.1. D2D Libraries

3.1.1. S-EHR side D2D Library

The current release of the library (i.e. M-D2D-E), contains all the operations that are needed from the side of the S-EHR application developer to initially interact with the library and finally with the HCP application. This library contains different operations that have to be invoked in a specific sequence for implementing the purposes of the D2D protocol, regarding the S-EHR application. This library is a Java-based component that can be nested in any Android application. It offers a set of Java operations for establishing a D2D connection, and allowing a mobile app of a Citizen to exchange her personal health records using the D2D protocol. More details regarding these operations can be found in deliverable **[D4.5]**.

3.1.1.1. Installation guide

The installation guide of the D2D library inside the S-EHR app contains necessary information for the process that has to be followed in order to add the library to the S-EHR app Android project, and make sure that every component of the S-EHR app is able to use the library. This guide is used to install the version of the library released in March 2021, and may be no longer valid for more recent versions. The only requirement for this process is for the developer to have an Android project application running with the min-sdk version properties upper or equals to 15. In order to upload the libraries so that they can be used in other projects, the Nexus repository is being used as an open source repository where can be published and retrieved versioned applications, as well as their dependencies **[Nexus]**.

Uploading the S-EHR side D2D library on Nexus

1. For the first time, in Android Studio, go to File -> Settings -> Tools -> Terminal and change the 'Shell path' to the directory of installed 'git bash'.
2. Go to build.gradle (Module: md2de) file and change the version of the library.
3. Change the Build Variants from 'debug' to 'release'.
4. In the toolbar select Build -> Make module 'md2de'.
5. Open the terminal of Android Studio and execute the command './gradlew:md2de:publish'.
6. Return the Build Variants from 'release' to 'debug'.

Installation steps of the S-EHR side D2D library:

1. Go to the gradle file of the project for using the libraries on all projects, or in the module one for using libraries in a specific module. Add the following lines of code inside the repository section:

```
repositories {
      google()
      jcenter()
      maven {
            url 'interopehrate-nexus-url'
            content {
                  includeGroup 'eu.interopehrate'
            }
      }
```

```
}
```

By adding this code, someone adds to gradle a new repository to retrieve libraries, referring to the one created specifically for the InteropEHRate project.

2. After adding to gradle the repository, add the following code to the dependencies section:

```
implementation(group:'eu.interoperhate', name:'md2de', version: '0.3.0')
```

3. Refresh the gradle project.

3.1.1.2. User guide

This user guide is available for the version 0.3.0 of the S-EHR side D2D library. The main flow of the functionality of this library is the following:

1. Open the Bluetooth connection with the HCP application, using the Bluetooth MAC address of the HCP app running machine. In the case of the S-EHR app, this information is contained in a QR code provided by the HCP app. In more detail, the bluetooth connection is triggered with the HCP application using the operation `broadcastConnection` provided by the `BluetoothConnection` class. This will take in parameter the Bluetooth MAC address of the HCP app running machine, combined with two listeners. The first one is the `D2DHRExchangeListeners` and will be explained afterwards, while the second one is a Listener that is used to detect when the connection is closed.

```
public ConnectedThread broadcastConnection(String address, D2DHRExchangeListeners listeners, D2DConnectionListeners listenersConnection)
```

This operation returns the thread that will contain the read process, and all the operations used to send information to the HCP app. In the S-EHR app, the communication with the D2D library is regrouped into a Service class, which is called `BluetoothService`.
Regarding the following user guide, it should be mentioned that the first Listener of the `broadcastConnection` operation is the `D2DHRExchangeListeners`. This listener regroups all the operations that will be used by the connection thread to do the interaction between HCP application and S-EHR application when the bluetooth connection is activated.

```
public interface D2DHRExchangeListeners {
	void onHealthOrganizationIdentityReceived(Practitioner var1);
	void onConsentRequested(String var1);
	void onPrescriptionReceived(Bundle var1);
	void onVitalSignsReceived(Bundle var1);
	void onMedicalDocumentRequestReceived(String var1, String var2, String var3);
}
```

2. Once the connection is accepted, identity information is being exchanged for:
    - **Sending information to the citizen from the S-EHR app**: Using the `sendPersonalIdentity` operation provided by the `ConnectedThread` class, the parameter of this operation is a Patient object from the FHIR library.

- **Receiving information from the HCP app**: In order to receive the HCP identification information, this is done through the `D2DHRExchangeListeners`, that returns a Practitioner object from the FHIR library.

3. Once the involved parties are identified, the consent result is being provided. This is done using the `sendConsentAnswer` operations provided by the `ConnectedThread` class, taking as a parameter a String.
4. The next step includes the sending of Health Data, of the citizen by using each operation provided by the `ConnectedThread` class, taking as a parameter a Bundle object from the FHIR library. These operation can send :
   - Patient Summary
   - Laboratory Results
   - Medication Request
   - Image Report
   - Pathology History Information
   - Vital Signs
   - Medical Document
5. The step of health data reception deals with receiving health data through the operations provided by the `D2DHRExchangeListeners`. The data that can be omitted are:
   - Prescription
   - Vital Signs
   - Request for Medical Document
6. The citizen can close the Bluetooth connection. The Bluetooth connection is closed using the `BluetoothConnection.closeConnection` operation.

3.1.2. HCP side D2D Library

The current release of the library contains all the operations that are needed from the side of the HCP application developer to interact with the library and finally with the S-EHR application. This library contains different operations that have to be invoked in a specific sequence for implementing the purposes of the D2D protocol, regarding the HCP application. This library is a Java based component that can be embedded in any Java based application. It offers a set of operations for establishing a D2D connection and enabling the application used by an HCP to send and receive data of a Citizen near her. More details regarding these operations can be found in deliverable **[D4.5]**.

3.1.2.1. Installation guide

The installation guide of the D2D library inside the HCP contains necessary information for the process that has to be followed in order to integrate the library into the HCP Java project and make sure that every component of the HCP is able to use the library. This guide is used to integrate the version of the library released in March 2021 and may be no longer valid for more recent versions. The only requirement for this process is for the developer to have access to the repository where the library is uploaded. There is a Nexus repository used in the project, and as a result the integration of the libraries is easier because it is just needed to add the necessary dependencies in the pom.xml file in order to fetch the jar files.

Uploading the HCP side D2D library on Nexus

1. For the first time, in Eclipse, go to ${user.home}/.m2/settings.xml and add the following lines of code:

```
<?xml version="1.0" encoding="UTF-8"?>
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
http://maven.apache.org/xsd/settings-1.0.0.xsd">
   <localRepository />
   <interactiveMode />
   <usePluginRegistry />
   <offline />
   <pluginGroups />
   <servers>
      <server>
         <id>nexus</id>
         <username>$username</username>
         <password>$password</password>
      </server>
   </servers>
   <mirrors />
   <proxies />
   <profiles />
   <activeProfiles />
</settings>
```

2. Go to pom.xml and change the version of the library.
3. For the first time right click on pom.xml Run us -> Run configurations. Right click on Maven build -> New Configuration. Select the base directory and in field 'Goals' write 'package' and press Apply.
4. Go again to Maven build -> New Configuration. Select the base directory and in field 'Goals' write 'clean deploy' and press Apply.
5. Choose the two configurations and press Run.

Integration steps of the HCP side D2D library:

1. Include the necessary dependency for the D2D library jar file inside the HCP App project.
2. Add InteropEHRate Nexus Repository as source of artefacts for the HCP App project.

A future plan of extensibility for the HCP side D2D library will include the conversion to an open source approach. In this case, the installation process would be different only regarding the way of cloning the project, since the location of the library would need to change if the open source approach would be taken.

3.1.2.2. User guide

This user guide is available for the version 0.3.0 of the HCP side D2D library. Shortly, the main flow implemented in HCP app using D2D library is the following:

1. Open the Bluetooth connection using the `BluetoothConnection.listenConnection` operation.

   ```
   public ConnectedThread listenConnection(D2DHRExchangeListeners
   listeners,D2DConnectionListeners listenersConnection, String
   structureDefinitionsPath)
   ```

   The first listener of listenConnection is `D2DHRExchangeListeners` which allows a Health Practitioner to know when incoming data from the citizen has been received.

```
public interface D2DHRExchangeListeners {
	void onPersonalIdentityReceived(Patient var1);
	void onPatientSummary(Bundle var1);
	void onNoConformantPatientSummaryReceived();
	void onPrescriptionReceived(Bundle var1);
	void onNoConformantPrescriptionReceived();
	void onLaboratoryResultsReceived(Bundle var1);
	void onImageReportReceived(Bundle var1);
	void onPathologyHistoryInformationReceived(Bundle var1);
	void onMedicalDocumentConsultationReceived(Bundle var1);
	void onVitalSignsReceived(Bundle var1);
	void onConsentAnswerReceived(String var1);
}
```

2. Afterwards, as soon as the connection will be accepted, information about the current practitioner can be sent using the `ConnectedThread.sendPersonalIdentity` operation. Moreover, in order to send the digital identity of the current practitioner it must be used the `ConnectedThread.sendHCPCertificate` operation.
3. The next step is to create and send the consent to the citizen in order to get accepted and signed using the `ConnectedThread.getSignedConsent` operation.
4. The HCP can receive the signed consent from the citizen by implementing the listener `onConsentAnswerReceived(String var1)` provided by the interface `D2DHRExchangeListeners`.
5. What is more, the HCP can receive EHRs from the S-EHR by implementing the listeners provided by the interface `D2DHRExchangeListeners`. The EHRs that can be omitted are:
    - Patient Summary
    - Laboratory Results
    - Medication Request
    - Image Report
    - Pathology History Information
    - Vital Signs
    - Medical Document
6. Send Evaluation data, using the operations provided by the `ConnectedThread` class, such as:
    - Prescription
    - Vital signs
    - Request for Medical Document
7. The D2D library validates the resources coming on the HCP automatically and in the case that they are not conformant the Bluetooth connection will close using the `BluetoothConnection.closeConnection` operation.

Regarding the aforementioned operations, the functionality of the D2D library was embedded by the developers of the HCP App inside a class called `CurrentD2DConnection`. This class offers the following functionalities as public operations to the rest of the HCP app components:

- `open()`: Opens the Bluetooth connection.
- `close()`: Closes the Bluetooth connection.
- `connectionState()`: Returns the connection state (ON, OFF, or PENDING_DEVICE).

- Also it provides implementations for the listeners dedicated to managing the connection and exchange of medical information, namely the `D2DConnectionListeners` and `D2DHRExchangeListeners`.

3.2. R2D Libraries

3.2.1. R2D Access Library

The subject of the current release is the version 0.3.0 of the Android library implementing an R2D client. This library enables client apps to use R2D without specific knowledge of technical details of the R2D protocol. The library is compatible with the Android environment, it has been developed using Java Technology and is released as an Android Archive format (.AAR extension). The file is named: `mr2d-0.3.0.aar`.

This version of the library provides several operations for allowing a client app to download the following type of health data:

- Patient Summary: in FHIR format according to International Patient Summary specifications defined by HL7 **[FHIR IPS]**.
- Laboratory Reports: represented with the FHIR resource named DiagnosticReport compliant to the specific profile defined by the InteropEHRate project.
- Diagnostic Reports containing images: represented with the FHIR resource named DiagnosticReport compliant to the specific profile defined by the InteropEHRate project.
- Prescriptions: represented with the FHIR resource named MedicationRequest compliant to the specific profile defined by the InteropEHRate project.
- Vital Signs: represented with the FHIR resource named Observation compliant to the specific profile defined by the InteropEHRate project.

3.2.1.1. Installation guide

The process to integrate the MR2DA library is the same explained in Section 3.1.1.1, where the difference lies in the implementation line given to the second point, regarding the adding of the gradle to the repository. Hence, for the MR2DA, the following line needs to be inserted in the gradle file:

```
implementation(group:'eu.interoperhate', name:'mr2d', version: '0.3.0')
```

If the development team importing the library is using Maven instead of Gradle, the same dependency must be expressed with the following Maven syntax:

```
<dependency>
      <groupId>eu.interopehrate</groupId>
      <artifactId>mr2d</artifactId>
      <version>0.3.0</version>
</dependency>
```

3.2.1.2. User guide

Using the MR2DA library means obtaining an instance of the class MR2DA and then invoking its methods to download health data of the authenticated citizen. To obtain an instance of MR2DA, developers must use the static method *create()* of the class MR2DFactory, as shown in the following example:

```
MR2DA mr2da = MR2DAFactory.create("http://hospitalOne/R2D/fhir/");
```

The *create()* method takes as input parameter the endpoint of the R2D server that it must connect to. If the S-EHR app needs to connect to more than one R2D server, it must creates one MR2DA instance for every R2D server:

```
MR2DA mr2daOne = MR2DAFactory.create("http://hospitalOne/R2D/");
MR2DA mr2daTwo = MR2DAFactory.create("http://hospitalTwo/R2D/");
```

In more detail, the MR2DA interface provides methods for executing searches of health data belonging to the authenticated citizen. It is possible to search over the entire set of of defined data type or it is possible to specify one or more specific type of data as subject of the search:

- `getResources`: Returns all the health data of whatever type of the citizen starting from a date.
- `getResourcesByCategory`: Returns all the health data of a specific type. Several additional parameters allow to add more filter criteria to the search to restrict the results.
- `getResourcesByCategories`: Returns all the health data of one or more specific type provided as argument, starting from a date.
- `getMostRecentResources`: Returns the most recent *n* instances of an health data type sorted by descending date, so that the most recent are on top.
- `getResourceById`: Return a specific health data instance identified by its ID.
- `getPatientSummary`: one specific method to return this particular kind of health data. This special method avoid to the client the several queries needed to search and retrieve an instance of Patient Summary.

In the following paragraphs some code samples to show how to use the several methods of the library.

Patient Summary

```
Bundle psBundle = (Bundle)mr2da.getPatientSummary();
Composition ps = (Composition)psBundle.getEntryFirstRep().getResource();
```

Search of Diagnostic Reports from a certain date

```
GregorianCalendar gc = new GregorianCalendar(2019, Calendar.JANUARY, 01);
mr2da.getResourcesByCategory(FHIRResourceCategory.DIAGNOSTIC_REPORT,
   gc.getTime(), false);
```

Search of Diagnostic Reports and Observations from a certain date

```
GregorianCalendar gc = new GregorianCalendar(2019, Calendar.JANUARY, 01);
mr2da.getResourcesByCategories(gc.getTime(), false,
   FHIRResourceCategory.DIAGNOSTIC_REPORT,
   FHIRResourceCategory.OBSERVATION);
```

Search of Diagnostic Reports by code

```
mr2da.getResourcesByCategory(FHIRResourceCategory.DIAGNOSTIC_REPORT,
   null, "http://loinc.org|30954-2", null, false);
```

Search of DocumentReference by code (Discharge Report code)

```
mr2da.getResourcesByCategory(FHIRResourceCategory.DOCUMENT_REFERENCE,
   null, "http://loinc.org|18842-5", null, false);
```

Search of Diagnostic Reports by Sub-category

```
mr2da.getResourcesByCategory(FHIRResourceCategory.DIAGNOSTIC_REPORT,
   "LAB", null, null, false);
```

Search of structured and unstructured (document) Image Reports from a certain date

```
GregorianCalendar gc = new GregorianCalendar(2015, Calendar.JANUARY, 01);
mr2da.getResourcesByCategory(DocumentCategory.IMAGE_REPORT,
    gc.getTime(), false);
```

Search of structured and unstructured (document) Laboratory Reports from a certain date

```
GregorianCalendar gc = new GregorianCalendar(2015, Calendar.JANUARY, 01);
mr2da.getResourcesByCategory(DocumentCategory.LABORATORY_REPORT,
    gc.getTime(), false);
```

All these operations invoke remote services, consequently, they cannot be used in the main thread of an Android application. They must be executed in AsyncTask, or in a separate Thread specially created. Moreover, the following permissions must be given to the app:

- `<uses-permission android:name="android.permission.INTERNET" />`
- `<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />`

3.2.2. R2D Backup Library

The current release of the R2DBackup library, contains all the operations that are needed from the side of the S-EHR application developer to initially interact with a S-EHR Cloud provider in order for a citizen to back up their encrypted EHR to this S-EHR Cloud. This library contains different operations that have to be invoked in a specific sequence for implementing the purposes of the R2D Backup protocol, regarding the S-EHR application. This library is a Java-based component that can be nested in any Android application. The library is released as an Android Archive format (.aar). It offers a set of Java operations for establishing the connection to the S-EHR Cloud and allowing a mobile app of a Citizen to upload and download their personal health records using the R2D Backup protocol. More details regarding these operations can be found in deliverable **[D4.2]**.

3.2.2.1. Installation guide

The process of integrating the R2D-Backup library is again, similar to the process explained in the previous Installation guide subsections. In case a gradle project is created, the following line needs to be inserted in the dependencies section of the build.gradle file:

```
implementation(group:'eu.interopehrate', name:'mr2dbackup', version: '0.1.0')
```

If the development team importing the library is using Maven instead of Gradle, the same dependency must be expressed with the following Maven syntax:

```
<dependency>
	<groupId>eu.interopehrate</groupId>
	<artifactId>mr2dbackup</artifactId>
	<version>0.1.0</version>
</dependency>
```

3.2.2.2. User guide

Using the MR2DBackup library means obtaining an instance of the class `MR2DBackup` and then invoking its methods to upload encrypted health data to the S-EHR Cloud, as well as downloading and decrypting such data from it. How to obtain an instance of `MR2DBackup`, is shown in the following example:

```
MR2DBackup mr2dbackup = new MR2DBackup();
```

In addition, an instance of the `SEHRCloudClient` class needs to be instantiated that is responsible for the connection to the preferred S-EHR Cloud provider of the citizen, as shown in the code snippet below:

```
SEHRCloudInterface cloudInterface =
SEHRCloudClient.getClient().create(SEHRCloudInterface.class);
```

The R2DBackup library provides methods for uploading encrypted health data to the S-EHR Cloud, downloading and decrypting this data to the S-EHR App, providing a list of the buckets created on the S-EHR Cloud that are related to the citizen, as well as providing a list of objects (i.e. data is stored as Objects in the S-EHR Cloud) in a specific bucket.

The exact methods provided by the R2DBackup library are listed below:

- `create`: Encrypts (using the R2D Encrypted Communication library [D3.5]) and uploads health data to the S-EHR Cloud.
- `get`: Downloads and decrypts (using the R2D Encrypted Communication library [D3.5]) health data that is already uploaded on the S-EHR Cloud. If the data is not found an error message is received.
- `listBuckets`: returns a list of the buckets that are related to a Citizen.
- `listObjects`: returns a list of objects in a specific bucket.

Since the R2DBackup works with encrypted documents it can support the following type of health data:

- Patient Summary: in FHIR format according to International Patient Summary specifications defined by HL7 [FHIR IPS].
- Laboratory Reports: represented with the FHIR resource named DiagnosticReport compliant to the specific profile defined by the InteropEHRate project.
- Diagnostic Reports containing images: represented with the FHIR resource named DiagnosticReport compliant to the specific profile defined by the InteropEHRate project.
- Prescriptions: represented with the FHIR resource named MedicationRequest compliant to the specific profile defined by the InteropEHRate project.
- Vital Signs: represented with the FHIR resource named Observation compliant to the specific profile defined by the InteropEHRate project.

In the following code samples the way these methods are used is presented:

Create

```
mr2dbackup.create(ehrBundle, DocumentCategory.PATIENT_SUMMARY, authToken,
symKey, new R2DBackup.MR2DBackupI(){});
```

Get

```
mr2dbackup.get(DocumentCategory.PATIENT_SUMMARY, authToken, symKey, new
R2DBackup.MR2DBackupI(){});
```

List Buckets

```
mr2dbackup.listBuckets(authToken, symKey, new R2DBackup.MR2DBackupI(){});
```

List Objects

```
mr2dbackup.listObjects(authToken, bucket, new R2DBackup.MR2DBackupI(){});
```

The abovementioned operations require the use of the internet, since they invoke remote services such as a S-EHR Cloud provider. For this reason, the permission to use the internet must be given to the app by adding the line below in the `AndroidManifest.xml` file of the project:

- `<uses-permission android:name="android.permission.INTERNET" />`

3.2.3. R2D Emergency Library

The current release of the R2DEmergency library, contains all the operations that are needed from the side of the HCP application developer to initially interact with a S-EHR Cloud provider in order for an HCP to grant access to a citizen's health data during an emergency situation. This library contains different operations that have to be invoked in a specific sequence for implementing the purposes of the R2D Emergency protocol, regarding the HCP application. This library is a Java-based component that can be nested in any Java application. The library is released as a Java Archive format (.jar). It offers a set of Java operations for establishing the connection to the S-EHR Cloud and allowing an HCP using the HCP app to download a Citizen's personal health records and decrypt them using the R2D Emergency protocol. More details regarding these operations can be found in deliverable [D4.2].

3.2.3.1. Installation guide

The process of integrating the R2DEmergency library is again, similar to the process explained in the previous Installation guide subsections. In case a gradle project is created, the following line needs to be inserted in the dependencies section of the build.gradle file:

```
implementation(group:'eu.interopehrate', name:'R2Demergency', version: '0.0.1')
```

If the development team importing the library is using Maven instead of Gradle, the same dependency must be expressed with the following Maven syntax:

```
<dependency>
	<groupId>eu.interopehrate</groupId>
	<artifactId>r2dEmergency</artifactId>
	<version>0.0.1</version>
</dependency>
```

3.2.3.2. User guide

Using the R2DEmergency library means obtaining an instance of the class `R2DEmergencyI` and then invoking its methods to download and decrypt encrypted health data from the S-EHR Cloud. To obtain an instance of `R2DEmergencyI`, developers must use the class `R2DEmergencyFactory`, as shown in the following example:

```
R2DEmergencyI r2demergency = R2DEmergencyFactory.create();
```

The R2DEmergency library provides methods for requesting access to a Citizen's health information stored in the S-EHR Cloud, and downloading such information from the S-EHR Cloud.

The exact methods provided by the R2D Backup library are listed below:

- `get`: Download and decryption (using the R2D Encrypted Communication library [D3.5]) of health data that is already uploaded on the S-EHR Cloud. If the data is not found an error message is received.

- `listBuckets`: returns a list of the buckets that are related to a Citizen.Specification of data encryption mechanisms for mobile and web applications
- `listObjects`: returns a list of objects in a specific bucket.

Since the R2D Emergency works with encrypted documents it can support the following type of health data:

- Patient Summary: in FHIR format according to International Patient Summary specifications defined by HL7 **[FHIR IPS]**.
- Laboratory Reports: represented with the FHIR resource named DiagnosticReport compliant to the specific profile defined by the InteropEHRate project.
- Diagnostic Reports containing images: represented with the FHIR resource named DiagnosticReport compliant to the specific profile defined by the InteropEHRate project.
- Prescriptions: represented with the FHIR resource named MedicationRequest compliant to the specific profile defined by the InteropEHRate project.
- Vital Signs: represented with the FHIR resource named Observation compliant to the specific profile defined by the InteropEHRate project.

In the following code samples the way these methods are used is presented:

Get (Patient Summary)

```
r2dEmergency.get(emergencyToken, DocumentCategory. PATIENT_SUMMARY);
```

Get (Laboratory Report)

```
r2dEmergency.get(emergencyToken, DocumentCategory.LABORATORY_REPORT);
```

Get (Medication request)

```
r2dEmergency.get(emergencyToken, FHIRResourceCategory.MEDICATION_REQUEST);
```

List Buckets

```
r2dEmergency.listBuckets(emergencyToken);
```

List Objects

```
r2dEmergency.listObjects(emergencyToken, bucket);
```

The abovementioned operations require the use of the internet, since they invoke remote services such as a S-EHR Cloud provider. For this reason, the permission to use the internet must be given to the app by adding the line below in the `AndroidManifest.xml` file of the project:

- `<uses-permission android:name="android.permission.INTERNET" />`

REFERENCES

- **[D3.5]** InteropEHRate Consortium, Specification of data encryption mechanisms for mobile and web applications v1, 2020. www.interopehrate.eu/resources
- **[D4.2]** InteropEHRate Consortium, Specification of remote and D2D protocol and APIs for HR exchange V2, 2020. www.interopehrate.eu/resources
- **[D4.5]** InteropEHRate Consortium, Design of libraries for remote and D2D HR exchange, 2019. www.interopehrate.eu/resources
- **[D4.12]** InteropEHRate Consortium, Libraries for remote and D2D HR exchange, 2019. www.interopehrate.eu/resources
- **[FHIR IPS]** International Patient Summary Implementation Guide. Web site: https://build.fhir.org/ig/HL7/fhir-ips/
- **[Nexus]** Nexus Repository OSS - Software Component Management. Web site: https://www.sonatype.com/nexus/repository-oss?topnav=true