Argus architecture

April 2019

CONTENTS

1. Introduction

1.1 Purpose of the document

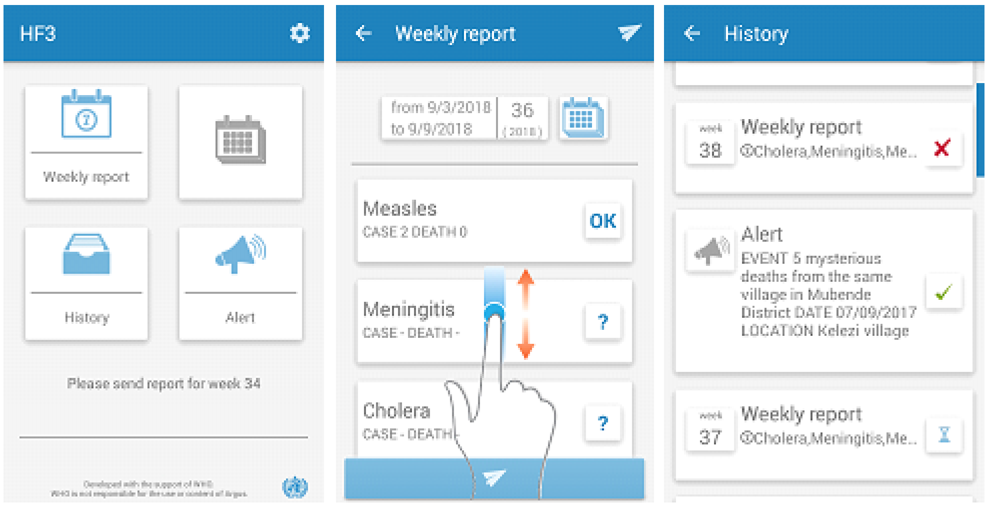
This document describes the technical architecture of the Argus solution. **It is intended to technical people who have knowledge in software development.**

Description on how to install, manage, maintain, and customize Argus is available in the Argus installation and administration document.

1.2 Argus overview

1.2.1 Purpose of the tool

The World Health Organization (WHO) has developed Argus an open source IT tool to support public health surveillance for early detection and response. It uses Short Message Service (SMS) technology for the transmission of information between the local healthcare facilities and all levels of the public health surveillance system via a mobile application (Figure 1). A web platform complements the application for data management and analysis (Figure 2).



Please send report for week 34

Developed with the support of WHO. WHO is not responsible for the use or content of Argus.

from 9/3/2018 to 9/9/2018 | 36 (2018)

Measles CASE 2 DEATH 0 OK

Meningitis CASE - DEATH - ?

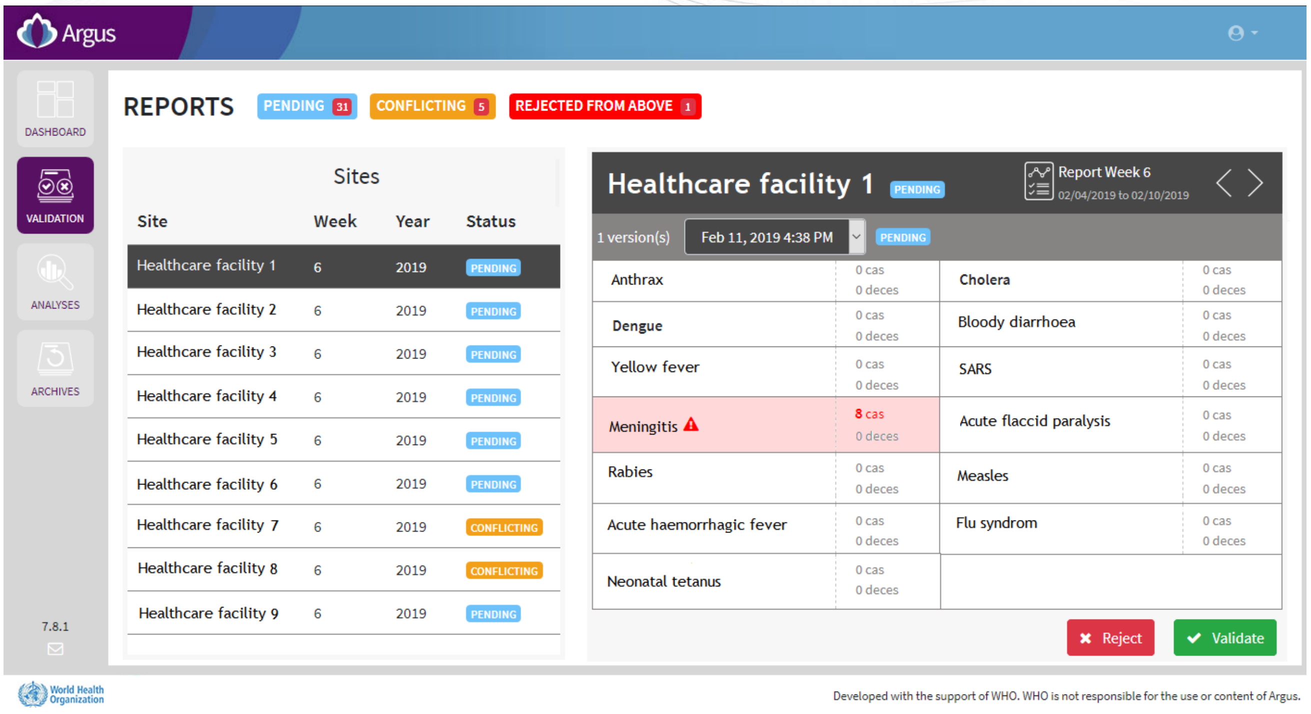
Cholera CASE - DEATH ?

week 38 Weekly report Cholera,Meningitis,Me...

Alert EVENT 5 mysterious deaths from the same village in Mubende District DATE 07/09/2017 LOCATION Kelezi village

week 37 Weekly report Cholera,Meningitis,Me...

Figure 1. Argus Android Client for mobile phones



Sites

Site	Week	Year	Status
Healthcare facility 1	6	2019	PENDING
Healthcare facility 2	6	2019	PENDING
Healthcare facility 3	6	2019	PENDING
Healthcare facility 4	6	2019	PENDING
Healthcare facility 5	6	2019	PENDING
Healthcare facility 6	6	2019	PENDING
Healthcare facility 7	6	2019	CONFLICTING
Healthcare facility 8	6	2019	CONFLICTING
Healthcare facility 9	6	2019	PENDING

Anthrax	0 cas 0 deces	Cholera	0 cas 0 deces
Dengue	0 cas 0 deces	Bloody diarrhoea	0 cas 0 deces
Yellow fever	0 cas 0 deces	SARS	0 cas 0 deces
Meningitis ⚠	8 cas 0 deces	Acute flaccid paralysis	0 cas 0 deces
Rabies	0 cas 0 deces	Measles	0 cas 0 deces
Acute haemorrhagic fever	0 cas 0 deces	Flu syndrom	0 cas 0 deces
Neonatal tetanus	0 cas 0 deces		

Figure 2. Argus web platform

Argus improves routine reporting quality and speed by reducing dependency on paper forms. It allows administrators to easily set up the public health events to be under surveillance, the variables to be collected, and the different levels of the public health surveillance system in charge of data validation and data analysis.

Argus is mainly designed to manage aggregated weekly reports of priority public health events. It also manages aggregated monthly reports of public health events, and alerts of unexpected public health events.

In practice, a central server located in the country collects the data sent by the healthcare facilities through SMS. The central server returns SMS to the healthcare facilities to acknowledge the reception of the data and posts the information on the internet through the Argus web platform for data management and analysis.

1.2.2 Technical components

At a technical level, Argus is composed of several components (Figure 3), each component will be described in detail in this document:

- A custom Argus Android Client application to be installed on Android mobile phones to be used by healthcare facilities to send alerts and reports through SMS.
- An Argus Android Gateway application to manage the reception and sending of SMS and share them with the Argus server.

    - It is a fork of the software envayaSMS (https://sms.envaya.org/).

- It is composed of a one instance of Argus Android Gateway application and several instances of Argus Android Gateway Slaves application used to overcome the limitation in the number of SMS that can be sent by an Android phone.

▶ A server composed of several components:

- An Apache service including a PHP interpreter and MySQL service provided by a XAMPP installer.
- An Argus Server web application in charge of processing all incoming and outgoing SMS.

  – It is a custom PHP application.
  – Argus Android Gateways interface with the Argus Server web application through a web service over a local WIFI network.
  – Data is stored on specific tables on the MySQL database (names starting by “ses_”).
  – A monitoring web interface is available to maintain the web application: http://localhost/ses.

- An Argus Dashboard web application in charge of the system configuration and data management.

  – It is a custom PHP application using the Symfony 2 framework.
  – An administration web interface is available for the configuration of the system on http://localhost/sesDashboard/web, it uses several javascript libraries such as “JQuery.
  – Data is stored on specific tables on the MySQL database (names starting by “sesdashboard_”).
  – It interfaces with the Argus Server web application via the file system through XML files.
  – An Argus Reports module is in charge of producing on-demand analyses and ad-hoc weekly epidemiological summaries: it is a fork of PHPreports (https://jdorn.github.io/php-reports/) using C3.js and D3.js to produce the charts.
  – A front end is available for the end user to visualize, manage and analyse the data sent by the healthcare facilities: Argus web platform http://ip_address/argus.
    - It is written in JavaScript using the Angular 5 framework, it uses several javascript libraries such as “JQuery”, “Bootstrap calendar” or “Bootstrap CSS”.
    - It can be used on desktop and mobile screens.
    - It is used to validate and reject reports.
    - It displays on-demand analyses and ad-hoc weekly epidemiological summaries produced by the Argus Reports module.

- It displays HTML dashboards produced on a regular basis by an instance of the R software.

- An Argus Monitoring application in charge of the system monitoring and of the synchronization between Argus Server web application and Argus Dashboard web application:
  - It is a custom Python application.
  - It monitors the core components of the Argus Server web application and Android Gateways.
  - It exports the reports and alerts received by Argus Server web application to Argus Dashboard web application with the support of the Argus Importer PHP script.
  - It synchronizes the configuration between Argus Server web application and Argus Dashboard web application.

- An instance of the R statistical software using custom scripts and Pandoc to produce through a scheduled task two HTML dashboards on a regular basis (one administrative and one epidemiological dashboard to monitor the performance of the public health surveillance system and the epidemiological situation).

Argus is released with a GNU-AGPL3 license, it uses other components released with a compatible license (Figure 4).

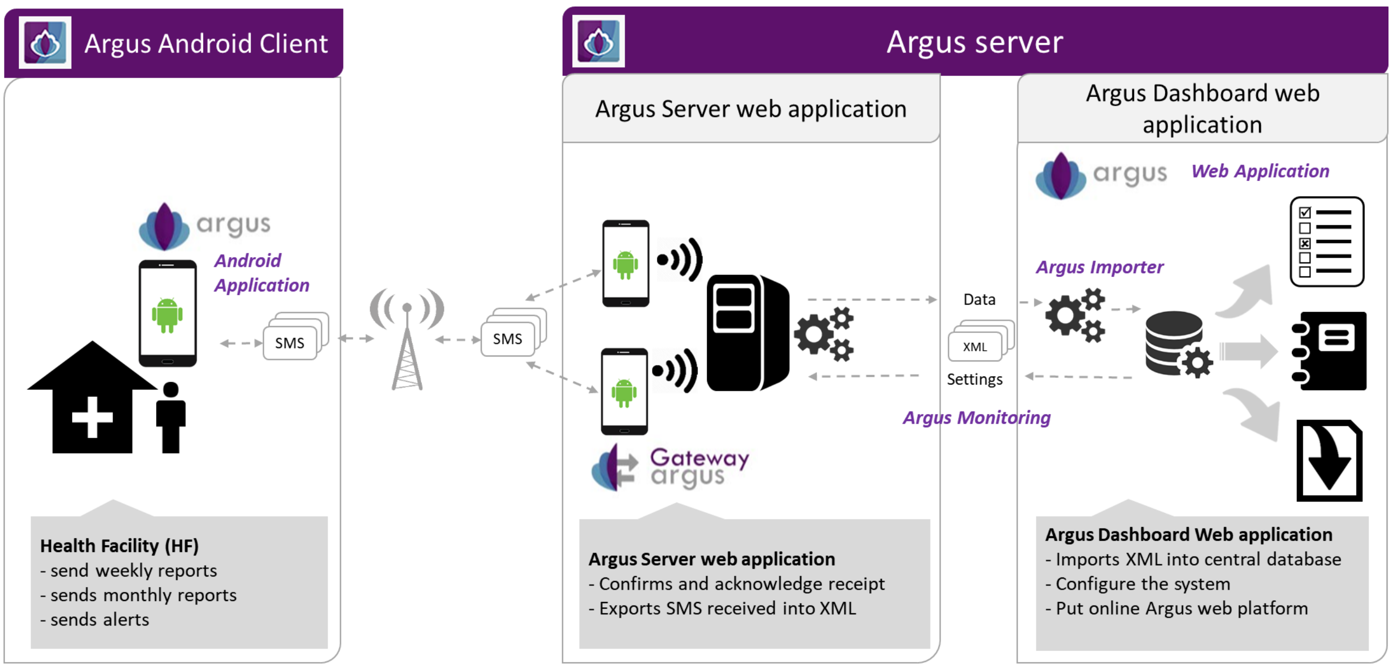

Argus Android Client
Argus server
Argus Server web application
Argus Dashboard web application
argus
Android Application
SMS
SMS
Gateway argus
Data
XML
Settings
Argus Monitoring
argus Web Application
Argus Importer
Health Facility (HF)
- send weekly reports
- sends monthly reports
- sends alerts
Argus Server web application
- Confirms and acknowledge receipt
- Exports SMS received into XML
Argus Dashboard Web application
- Imports XML into central database
- Configure the system
- Put online Argus web platform

Figure 3. Argus components

Figure 4. Some of the open source components used in Argus

2. Argus Android Client

2.1 Overview

Argus Android Client application allows users to report by SMS weekly and monthly epidemiologic reports along with alerts to the Argus server. As it only needs SMS, it only needs 2G networks.

The diseases and variables to be reported are defined on the ARGUS Server web application and synchronized by SMS with the Argus Android Client. Once configured and synchronized the android phone is ready to report data to the server.

2.2 Technology

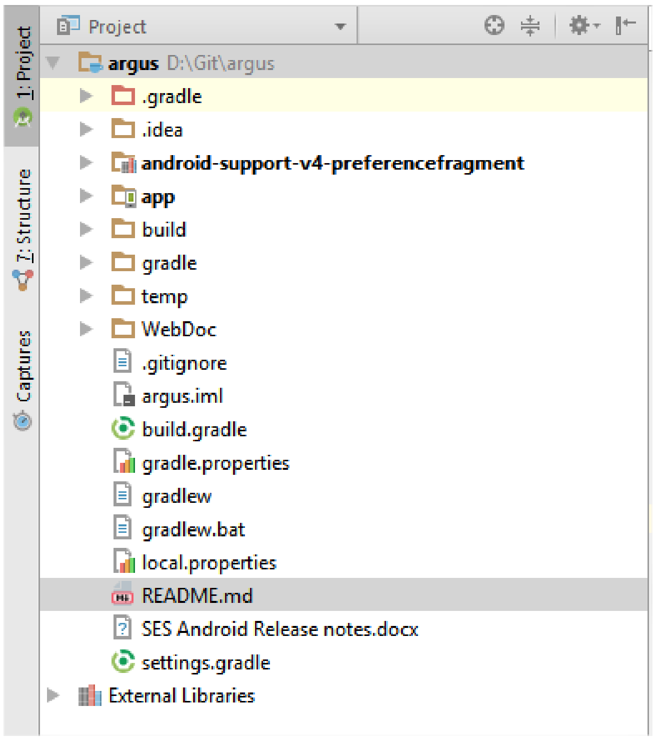
Argus Android Client is developed on the Android operating system with Android Studio IDE. Development language used is Java. The current android sdk version specified in the application settings are: minimal Sdk version : 9 (Android 2.3.2); target Sdk Version : 21; compiled Sdk Version : 21.

The external library "android-support-v4-preferencefragment" is used in this solution to add Preference Fragment compatibility layer for Android 1.6 and up (https://github.com/kolavar/android-support-v4-preferencefragment).

2.3 Source file structure

The source file structure of the project is the following (Figure 5):

- **.gradle**: Android folder gradle files.
- **.idea**: Android Studio folder.
- **android-support-v4-preferencefragment:** Library used to display the setting screen.
- **app:** Argus applications files.
- **build:** Android compiled files.
- **gradle:** Android folder used to build the application.
- **temp:** temporary folder.
- **WebDoc:** HTML documentation auto generated.



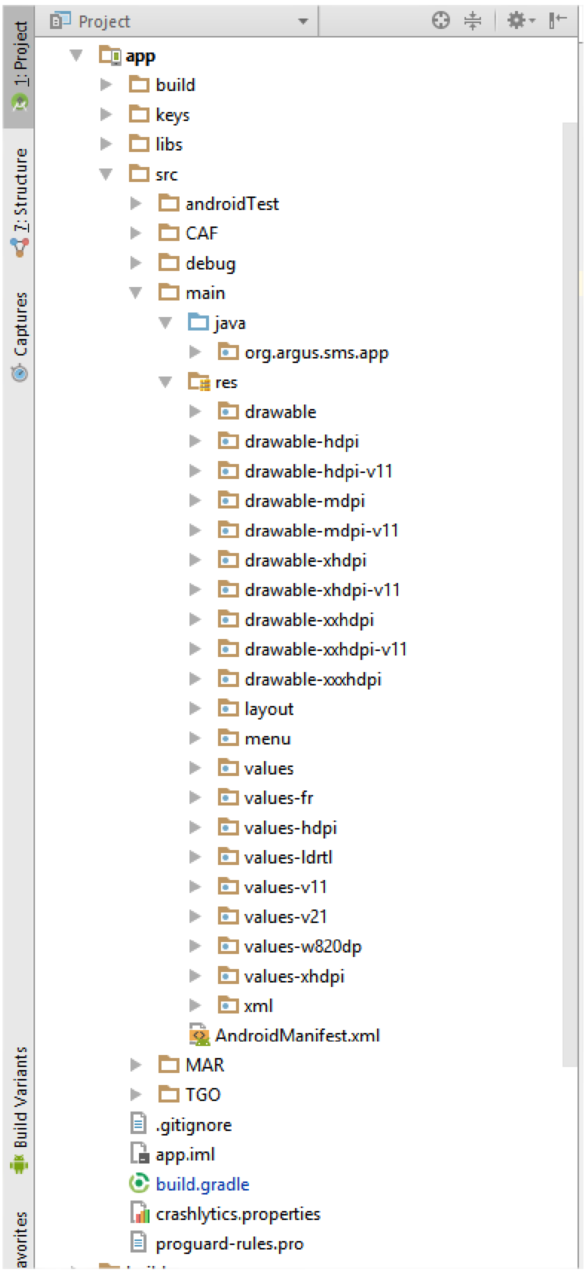
```
argus D:\Git\argus
    .gradle
    .idea
    android-support-v4-preferencefragment
    app
    build
    gradle
    temp
    WebDoc
    .gitignore
    argus.iml
    build.gradle
    gradle.properties
    gradlew
    gradlew.bat
    local.properties
    README.md
    SES Android Release notes.docx
    settings.gradle
External Libraries
```

Figure 5. Global source file structure Argus Android Client

The application files structure of the project is as follows (Figure 6):

- build: Android compiled files.
- keys: WHO signature keys to sign the release apk (documentation on https://developer.android.com/studio/publish/app-signing).
- libs: Android library folder.
- src: application source files. In this folder you will find:
    - androidTest: test files;
    - CAF, MAR, TGO: flavours folder containing all files needed to customize the application.

- main: main folder containing java application files and resources files.
- Debug: Android debug folder.



```
▼ app
    ▶ build
    ▶ keys
    ▶ libs
    ▼ src
        ▶ androidTest
        ▶ CAF
        ▶ debug
        ▼ main
            ▼ java
                ▶ org.argus.sms.app
            ▼ res
                ▶ drawable
                ▶ drawable-hdpi
                ▶ drawable-hdpi-v11
                ▶ drawable-mdpi
                ▶ drawable-mdpi-v11
                ▶ drawable-xhdpi
                ▶ drawable-xhdpi-v11
                ▶ drawable-xxhdpi
                ▶ drawable-xxhdpi-v11
                ▶ drawable-xxxhdpi
                ▶ layout
                ▶ menu
                ▶ values
                ▶ values-fr
                ▶ values-hdpi
                ▶ values-ldrtl
                ▶ values-v11
                ▶ values-v21
                ▶ values-w820dp
                ▶ values-xhdpi
                ▶ xml
              AndroidManifest.xml
        ▶ MAR
        ▶ TGO
      .gitignore
      app.iml
      build.gradle
      crashlytics.properties
      proguard-rules.pro
```

Figure 6. Application files structure Argus Android Client

2.4 Main components architecture

Activities structure (Figure 7):

- **ActivityDashboard:** activity displaying home page of the application.
- **ActivityAlert:** activity dealing with alert edition and sending.
- **ActivityReportWeekly:** activity dealing with weekly report.
- **ActivityReportMonthly:** activity dealing with monthly report.

- **ActivityReportDetail:** activity dealing with disease values inside a monthly or weekly report.
- **ActivityHistory:** activity dealing with history (sent reports and alerts).
- **ActivitySettings:** activity dealing with settings of the application.
- **ActivitySynchronization:** activity dealing with the synchronization between the Argus Android Client and the Argus server.

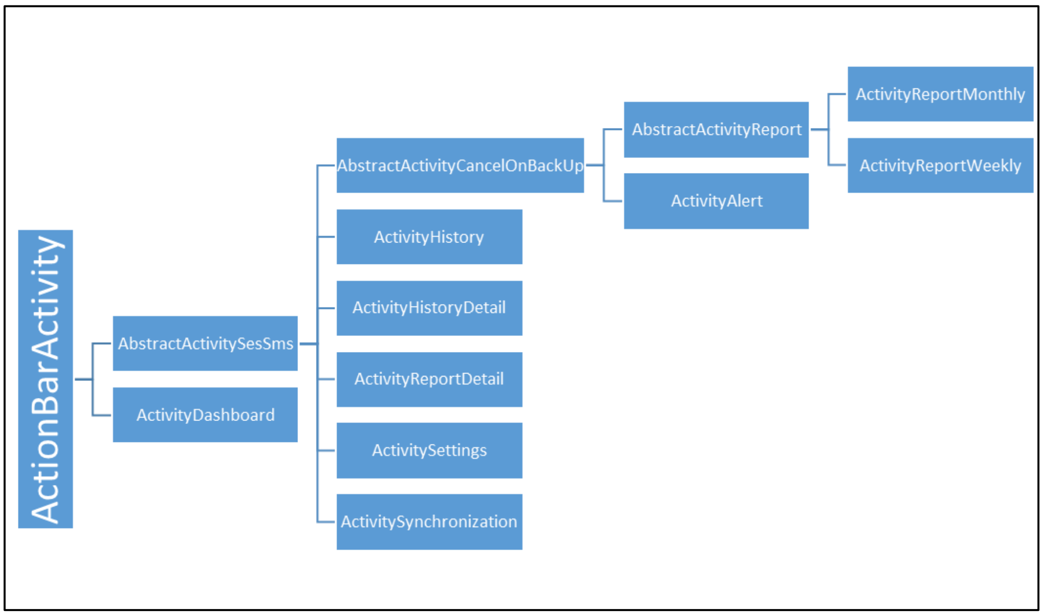


Figure 7. Activities structure Argus Android Client

Fragments structure (Figure 8). A fragment is a component that is used by an activity. The fragment is used to manage and display the content of the screen. One activity here is only a fragment container.

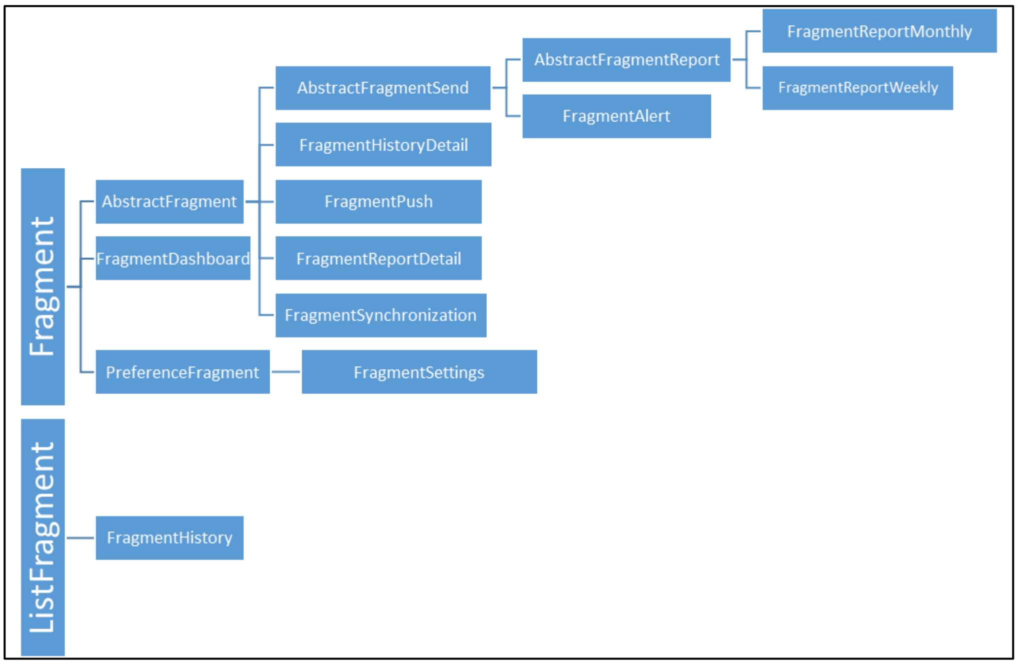


Figure 8. Fragments structure Argus Android Client

Views structure (Figure 9).

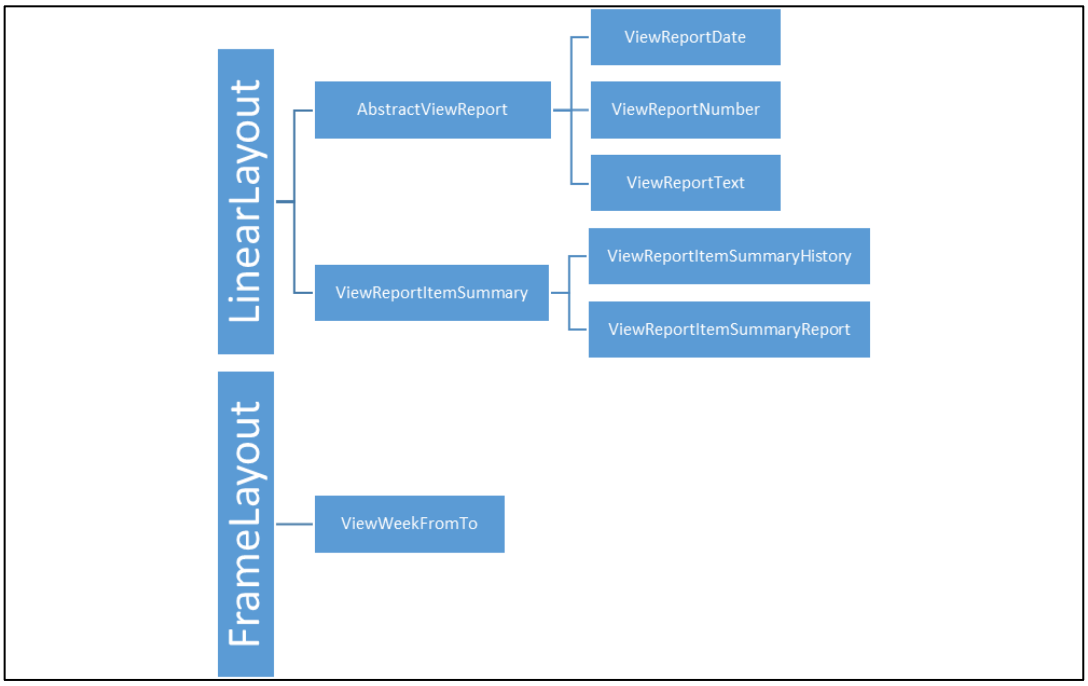


Figure 9. Views structure Argus Android Client

Receiver structure (Figure 10):

- The **BootReceiver** is used to check if the history status of each report/alert sent is consistent. It verifies that icons appearing in the history page are consistent with the database (icons to follow the status of a report).
- The **SMSReceiver** is called when the Android operating system receives an SMS. This SMS is transmitted to Argus Android Client via this receiver.

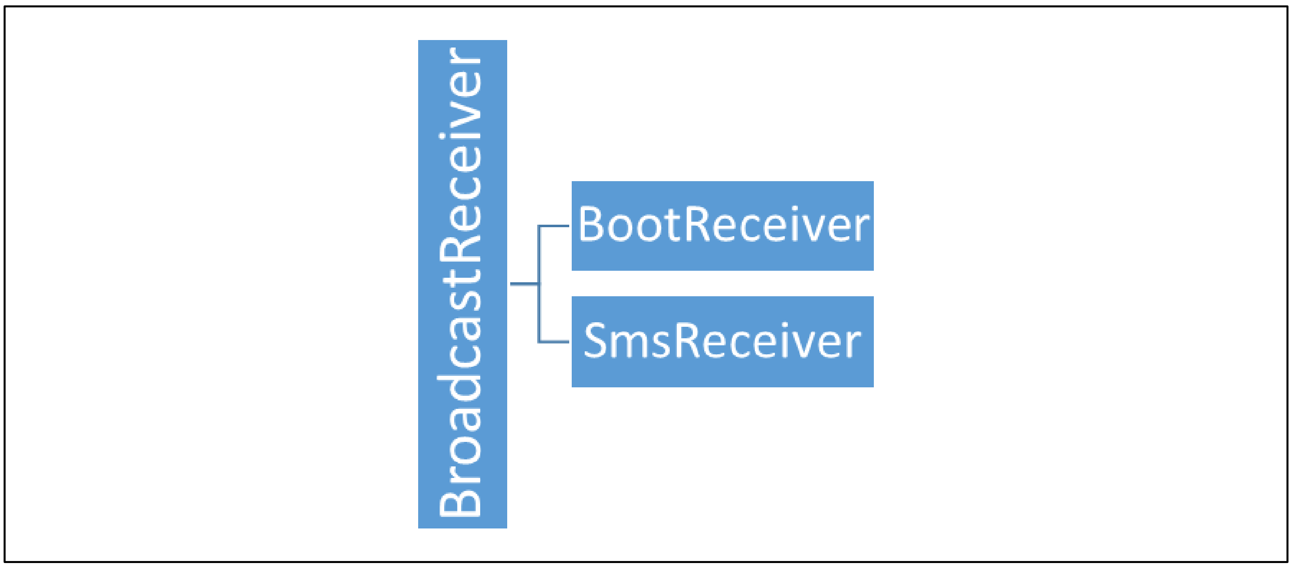


Figure 10. Receiver structure Argus Android Client

The database used is an SQLite database, there is only table used to manage the SMS data, its structure is presented in table 1.

A custom provider “SesProvider” is used to manage the database as defined in the “app/src/main/AndroidManifest.xml” file. This provider contains custom SQL requests (to display history data) and extends the Android class “ContentProvider” and use two helpers’ class: “SesContract” and “SesDatabase”.

- SesContract is used to describe the database (columns, uniform resource identifier (URI)).
- SesDatabase extends Android “SQLiteOpenHelper” class to override OnCreate and OnUpdate methods.

Table 1. Database structure of Argus Android Client

Field Name	Type	Description
_ID	Integer, primary key	Primary key
DISEASE	Text	The disease name
LABEL	Text	The label for this disease
WEEK	Text	The week number
MONTH	Text	The month number
YEAR	Text	The year
TYPE	Integer	Type of the sms (config or other)
SUBTYPE	Integer	Subtype of the sms (Alert, Report…)
ID	Integer, default -1	Id of the sms (ANDROIDID)
TEXT	Text	The sms sent in full text
TIMESTAMP	Text	Time of sending the sms (or reception)
STATUS	Integer	Status of the message (SENT, RECEIVE, ERROR…)
SMSCONFIRM	Text	The confirmation sms received with the same ANDROIDID
REPORTID	Integer	Incremental Id referencing all SMS sent for one specific weekly or monthly report

2.5 Input/output

2.5.1 Inputs

Inputs are used to synchronize Argus Android Client with Argus server as soon as the server is configured. SMS are the only inputs managed by Argus Android Client. The synchronization process involves 5 general SMS + 1 SMS for the alerts + 1 SMS for each disease to be reported. Below are examples of the SMS sent by the server during the synchronization process.

```
ANDROIDID=1 TA: ALERT EVENEMENT=Str,DATE=Dat 0,LIEU=Str 0,CAS=Int
0,DECES=Int 0
```

Keyword	Explanation
`ANDROIDID`	Unique id to identify the synchronization.
`TA:`	Inform Argus Android Client that it is the template for the Alert.
`ALERT`	Keyword to be used when Argus Android Client sends an alert.
`EVENEMENT=Str`	First field to be fulfilled in the alert form named “EVENEMENT”. String format is required, and this field is mandatory.
`DATE=Dat 0`	Second field to be fulfilled in the alert form named “DATE”. Date format is required, 0 means the field is not mandatory.
`LIEU=Str 0`	Third field to be fulfilled in the alert form named “LIEU”. String format is required, 0 means the field is not mandatory.
`CAS=Int 0`	Fourth field to be fulfilled in the alert form named “CAS”. Integer format is required, 0 means the field is not mandatory.
`DECES=Int 0`	Fifth field to be fulfilled in the alert form named “DECES”. Integer format is required, 0 means the field is not mandatory.

```
ANDROIDID=1 TW: REPORT
DISEASE=11MEN,LBL=Meningite,YEAR=Int,WEEK=Int,CAS=Int,DECES=Int
```

Keyword	Explanation
`ANDROIDID`	Unique id to identify the synchronization.
`TW:`	Inform Argus Android Client that it is the template for the Weekly report.
`REPORT`	Keyword to be used when Argus Android Client sends a Weekly Report.
`DISEASE=11MEN`	Disease identifier to be used when Argus Android Client sends data for this disease (11MEN).
`LBL=Meningite`	Label of the disease to be displayed on the weekly report form.
`YEAR=Int`	Year number of the weekly report (Integer).
`WEEK=Int`	Week number of the weekly report (Integer).
`CAS=Int`	First field to be fulfilled in the weekly report form named “CAS” (Integer).
`DECES=Int`	Second field to be fulfilled in the weekly report form named “Deces” (Integer).

```
ANDROIDID=1 TW: REPORT
DISEASE=10ROU,LBL=Rougeole,YEAR=Int,WEEK=Int,CAS=Int,DECES=Int
```

Keyword	Explanation
`ANDROIDID`	Unique id to identify the synchronization.
`TW:`	Inform Argus Android Client that it is the template for the Weekly report.
`REPORT`	Keyword to be used when Argus Android Client sends a Weekly Report.
`DISEASE=10ROU`	Disease identifier to be used when Argus Android Client sends data for this disease (10ROU).
`LBL=Rougeole`	Label of the disease to be displayed on the weekly report form.
`YEAR=Int`	Year number of the weekly report (Integer).
`WEEK=Int`	Week number of the weekly report (Integer).
`CAS=Int`	First field to be fulfilled in the weekly report form named “CAS” (Integer).
`DECES=Int`	Second field to be fulfilled in the weekly report form named “DECES” (Integer).

```
ANDROIDID=1 CF: M4ConfAlert=L alerte n a pas ete recue. Contactez votre
superviseur
```

Keyword	Explanation
`ANDROIDID`	Unique id to identify the synchronization.
`CF:`	General Configuration SMS.
`M4ConfAlert=`	Message to be displayed in Argus Android Client when Alert acknowledgement message has not been received (after a configured delay).

```
ANDROIDID=1 CF: M4ConfW=Le rapport hebdomadaire n a pas ete recu. Renvoyez
le depuis l historique
```

Keyword	Explanation
`ANDROIDID`	Unique id to identify the synchronization.
`CF:`	General Configuration SMS.
`M4ConfW =`	Message to be displayed in Argus Android Client when Weekly Report acknowledgement messages have not been received (after a configured delay).

```
ANDROIDID=1 CF: M4ConfM=Le rapport mensuel n a pas ete recu. Renvoyez le
depuis l historique
```

Keyword	Explanation
`ANDROIDID`	Unique id to identify the synchronization.
`CF:`	General Configuration SMS.
`M4ConfM=`	Message to be displayed in Argus Android Client when Monthly Report acknowledgement messages have not been received (after a configured delay).

```
ANDROIDID=1 CF: HFName=Adawlato,Server=+4193299832
```

Keyword	Explanation
`ANDROIDID`	Unique id to identify the synchronization.
`CF:`	General Configuration SMS.
`HFName=`	Health Facility name assigned to this Argus Android Client.
`Server=`	Phone number of the Argus Gateway.

```
ANDROIDID=1 CF:
NbMsg=8,NbCharMax=150,WeekStart=1,D4ConfAlert=45,D4ConfW=120,D4ConfM=120
```

Keyword	Explanation
`ANDROIDID`	Unique id to identify the synchronization.
`CF:`	General Configuration SMS.
`NbMsg=`	Number of SMS to complete the synchronization (this one included).
`NbCharMax=`	Maximum number of characters authorized in one SMS.
`WeekStart=`	Identify the first day of the epidemiological week. (1= Monday) … (7=Sunday)
`D4ConfAlert=`	Delay in minutes before displaying the message configured in `M4ConfAlert=`
`D4ConfW=`	Delay in minutes before displaying the message configured in `M4ConfW=`
`D4ConfM=`	Delay in minutes before displaying the message configured in `M4ConfM=`

2.5.2 Outputs

Outputs are used to send the data from reports and alerts to the server by SMS. SMS are formatted regarding the configuration received during the synchronization process with the server (see section above).

After each SMS sent by Argus Android Client an SMS is expected to be received in return from Argus server to acknowledge its reception.

Example of SMS sent by Argus Android Client to Argus server:

- Weekly report sent by Argus Android Client:

```
REPORT DISEASE=10ROU,YEAR=2016,WEEK=52,CAS=2,DECES=0,ANDROIDID=274,RID=2

REPORT DISEASE=11MEN,YEAR=2016,WEEK=52,CAS=1,DECES=0,ANDROIDID=273,RID=2
```

Key Word	Explanation
REPORT	Inform Argus server that it is a SMS from a weekly report.
DISEASE=10ROU	Inform Argus server the disease concerned.
YEAR=2016	Year of the report.
WEEK=52	Week of the report.
CAS=2	Number of “CAS”.
DECES=0	Number of “DECES”.
ANDROIDID=274	Identify the SMS.
RID=2	Identify the weekly report.

- Acknowledgement sent back by Argus server:

```
ANDROIDID=274 ANDROID_OK

ANDROIDID=273 ANDROID_OK
```

Key Word	Explanation
ANDROIDID=	Identifier of the SMS.
ANDROID_OK	Confirmation of the reception from the Argus server.

- Alert sent by Argus Android Client:

```
ALERT EVENEMENT=rage
case,DATE=02/01/2017,LIEU=Geneva,CAS=1,DECES=,ANDROIDID=296
```

Key Word	Explanation
ALERT	Inform Argus server that it is receiving an alert.
EVENEMENT=	“EVENEMENT” information (String).
DATE=	“DATE” information (Date).
LIEU=	“LIEU” information (String).
CAS=	“CAS” information (Integer).
DECES=	“DECES” information (Integer).
ANDROIDID=296	Identify the SMS.

- Acknowledgement sent back by Argus server:

```
ANDROIDID=296 ANDROID_OK
```

3. Argus Android Gateway

3.1 Overview

Argus Android Gateway application is the link between the Argus Android Client and the Argus server. It manages all incoming SMS from the Argus Android Client to the Argus server and outgoing SMS from the Argus server to the Argus Android Client. It does not contain any business logic as it is just a gateway receiving and forwarding SMS.

Argus Android Gateway communicates with Argus Server using web services through a local WIFI network to:

- Push to Argus Server new incoming SMS (coming from ARGUS Android Client).
- Pull from Argus Server pending SMS to be sent back from Argus Server to Argus Android Client.
- Brief regularly Argus Server on its current state (e.g. battery level, GSM operator name, power type).

Android limits the number of SMS that can be sent by a single application:

- Before Android SDK 16 (Android 4.1 Jelly Bean): 100 outgoing SMS per hour authorized.
- After Android SDK 16 (Android 4.1 Jelly Bean): 30 outgoing SMS per half an hour authorized.

Argus Android Gateway Slave is an extension to the Argus Android Gateway application that allows overcoming this limit to send SMS. Installing additional Argus Android Gateway Slave applications on the phone allows the main Argus Android Gateway application to delegate outgoing SMS to an Argus Android Gateway Slave and thus increase the overall capacity to send SMS. Each Argus Android Gateway Slave increases the outgoing SMS capacity by 100 SMS per hour or 30 SMS per 30 min depending on the SDK version. Argus Android Gateway Slave doesn’t manage incoming SMS and is only used for outgoing SMS.

3.2 Technology

Argus Android Gateway and Argus Android Gateway Slave are developed on Android with Android Studio IDE. The development language used is Java. The SDK version specified in the applications settings are:

- Argus Android Gateway:
  - Minimal SDK version: 9 (Android 2.3.2).
  - Target SDK version: 21.

- Compiled SDK version: 21.

- Argus Android Gateway Slave:

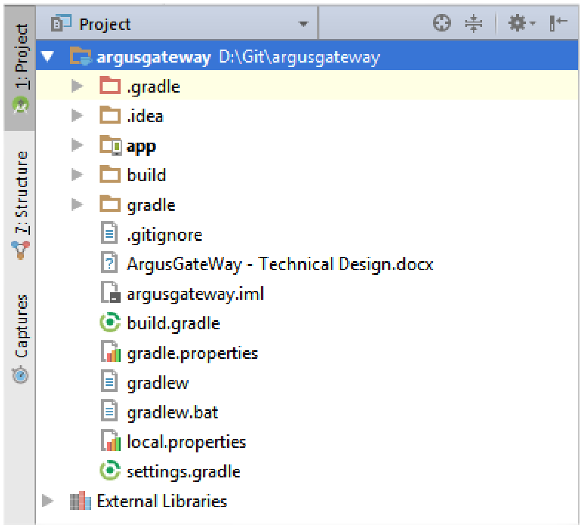
  - Minimal SDK version: 9 (Android 2.3.2).
  - Target SDK version: 23.
  - Compiled SDK version: 23.

3.3 Source file structure

3.3.1 Argus Android Gateway

The source file structure of the project is the following (Figure 11):

- .gradle: Android folder gradle files.
- .idea: Android Studio folder.
- app: Argus Android Gateway applications files.
- build: Android compiled files.
- gradle: Android folder used to build the application.



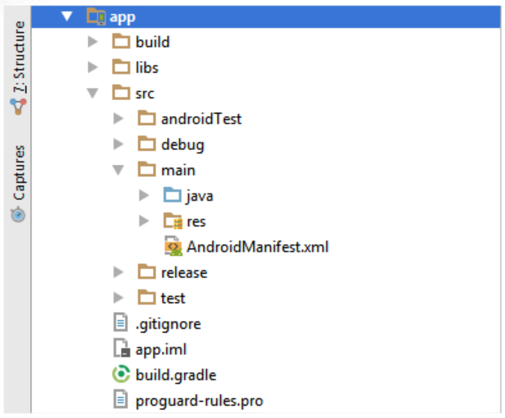
```
argusgateway D:\Git\argusgateway
  .gradle
  .idea
  app
  build
  gradle
  .gitignore
  ArgusGateWay - Technical Design.docx
  argusgateway.iml
  build.gradle
  gradle.properties
  gradlew
  gradlew.bat
  local.properties
  settings.gradle
External Libraries
```

Figure 11. Global source file structure Argus Android Gateway

The application files structure of the project is as follows (Figure 12):

- build: Android compiled files
- libs: Android library folder
- src: Application Source files. In this folder you will find:

  - androidTest: Test files
  - debug: Android debug folder
  - main: main folder containing Java application files and resources files
  - release: Android release folder



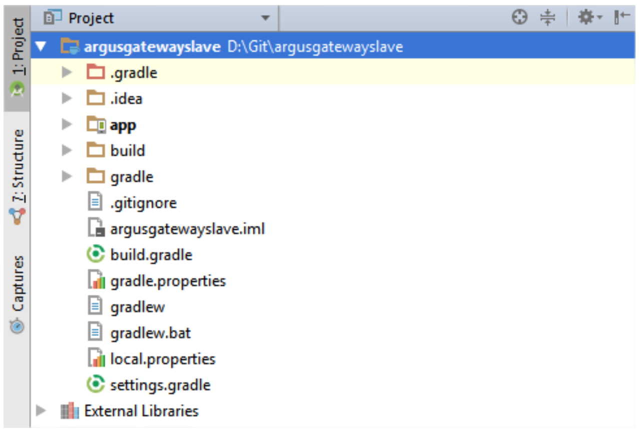
app
build
libs
src
androidTest
debug
main
java
res
AndroidManifest.xml
release
test
.gitignore
app.iml
build.gradle
proguard-rules.pro

Figure 12. Application files structure Argus Android Gateway

3.3.2 Argus Android Gateway Slave

The source file structure of the project is the following (Figure 13):

- .gradle: Android folder gradle files.
- .idea: Android Studio folder.
- app: Argus Android Gateway Slave applications files.
- build: Android compiled files.
- gradle: Android folder used to build the application.



Project

argusgatewayslave D:\Git\argusgatewayslave
.gradle
.idea
app
build
gradle
.gitignore
argusgatewayslave.iml
build.gradle
gradle.properties
gradlew
gradlew.bat
local.properties
settings.gradle
External Libraries

Figure 13. Global source file structure Argus Android Gateway Slave

The application files structure of the project is as follows (Figure 14):

- build: Android compiled files.
- src: Application Source files. In this folder you will find:
    - androidTest: Test files.
    - main: main folder containing Java application files and resources files.
    - slave01-09: flavours folder containing all files needed to customize the application.

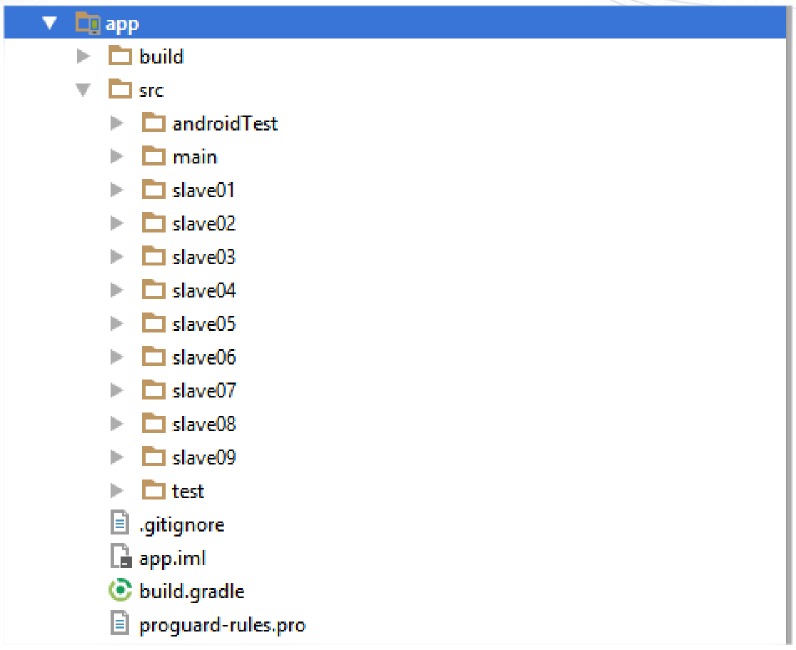


Figure 14. Application files structure Argus Android Gateway Slave

3.4 Main components architecture

3.4.1 Argus Android Gateway

Activities structure (Figure 15).

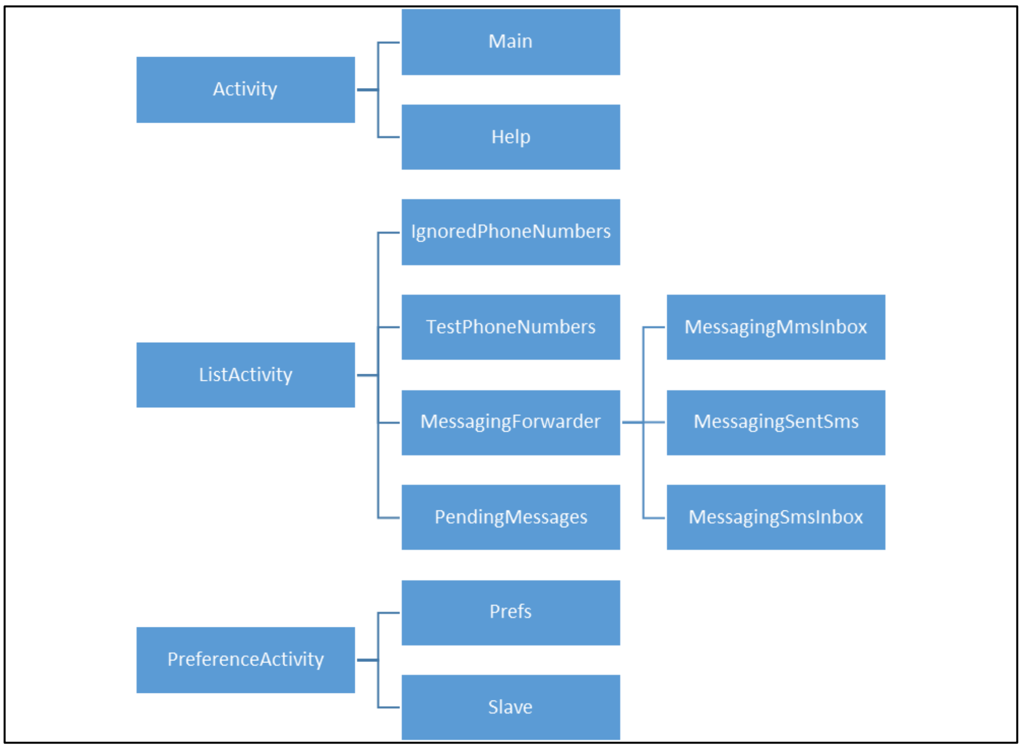


Figure 15. Activities structure Argus Android Gateway

Receivers structure (Figure 16):

- **SmsReceiver**: catch all SMS received by the phone.
- **NudgeReceiver**: called when the phone boot is complete and starts Argus Android Gateway.
- **OutgoingSmsReceiver**: when sending SMS, Argus Android Gateway starts an intent catch by this class, which will try to send the SMS. This process is made because the application treats itself as a slave in its Argus Android Gateway Slave pool (see section 3.6).
- **SlaveInstallReceiver**: receive when an Android application is installed on the phone, if it is a slave, adds it in the Argus Android Gateway Slave pool (see section 3.6).

There are multiple other receiver classes used to manage simple events such as connectivity behaviours, SMS sent callbacks, SMS receive callbacks.

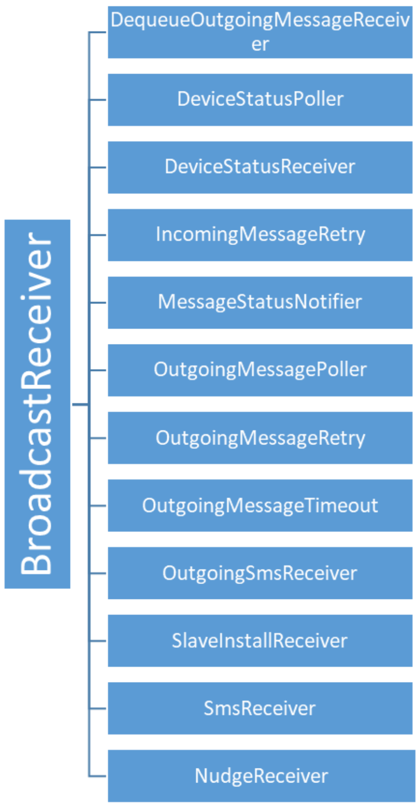


Figure 16. Receivers structure Argus Android Gateway

Services structure (Figure 17):

- **ForegroundService**: this service is used to keep Argus Android Gateway alive. By starting this service, a kind of priority status to Argus Android Gateway is added. Android will try to never kill this application if it needs memory.
- **CheckMessagingServices**: used as routine (with the use of system alarms) to check periodically if Argus Server needs to send SMS.

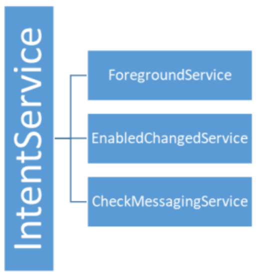


Figure 17. Services structure Argus Android Gateway

Tasks structure (Figure 18):

- **HttpTask**: basic class used to create all networking task (all communications with Argus Server).
- **ForwarderTask**: daughter of HttpTask, this is a specific task used to forward all incoming SMS catch by the SmsReceiver.
- **ServerFinder**: a task used to discover the Argus Server address.

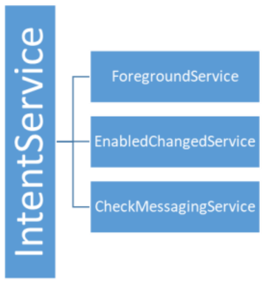


Figure 18. Tasks structure Argus Android Gateway

The database used is an SQLite database, there are two tables to manage the incoming and outgoing SMS (Tables 2 and 3).

No provider class is used to manage the Argus Android Gateway database, but the “DatabaseHelper” class (extending the Android “SQLiteOpenHelper” class) contains all the methods used to access the SQLite database.

Table 2. Incoming SMS table of Argus Android Gateway

Field Name	Type	Description
_ID	Integer, primary key	Primary key
message_type	Varchar	sms, mms, call
messaging_id	Integer	
from_number	Varchar	Sender number
to_number	Varchar	Receiver number
message	Text	Message body
direction	Integer	Incoming (0), Sent(1)
timestamp	Integer	time

Table 3. Outgoing SMS table of Argus Android Gateway

Field Name	Type	Description
_ID	Integer, primary key	Primary key
message_type	Varchar	sms, mms, call
from_number	Varchar	Sender number
to_number	Varchar	Receiver number
message	Text	Message body
priority	Integer	Descending priority (High priority to biggest value)
server_id	Text	Server Id of the message

3.4.2 Argus Android Gateway Slave

Receivers structure (Figure 19):

- **OutgoingSmsReceiver**: intent sent by the Argus Android Gateway application to ask an Argus Android Gateway Slave application to send an SMS.
- **QuerySlavesReceiver:** intent sent by the Argus Android Gateway application and catch by each Argus Android Gateway Slave to add their package name (see section 3.6).

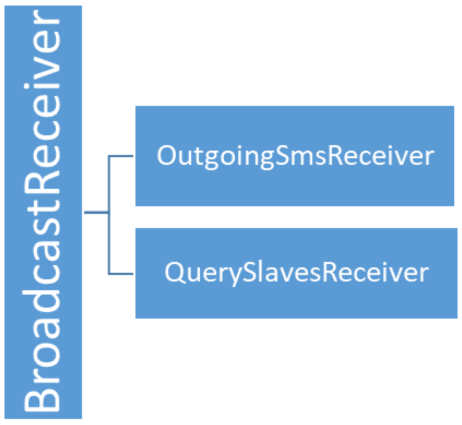
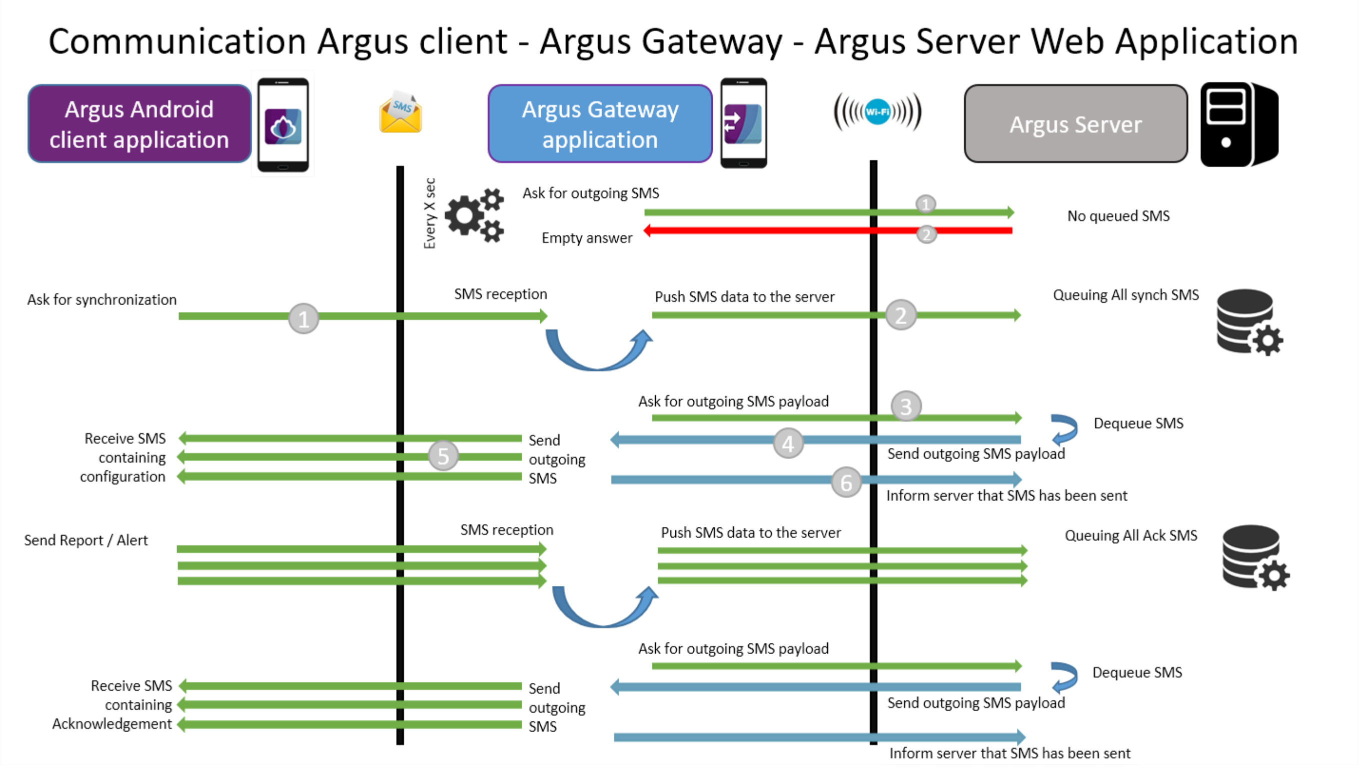


Figure 19. Receivers structure Argus Android Gateway Slave

3.5 Input/output

A communication scheme between Argus Android Client, Argus Android Gateway and Argus Server is presented in Figure 20.



Communication Argus client - Argus Gateway - Argus Server Web Application
Argus Android client application
Argus Gateway application
Argus Server
Every X sec
Ask for outgoing SMS
Empty answer
No queued SMS
Ask for synchronization
SMS reception
Push SMS data to the server
Queuing All synch SMS
Ask for outgoing SMS payload
Dequeue SMS
Receive SMS containing configuration
Send outgoing SMS
Send outgoing SMS payload
Inform server that SMS has been sent
Send Report / Alert
SMS reception
Push SMS data to the server
Queuing All Ack SMS
Ask for outgoing SMS payload
Dequeue SMS
Receive SMS containing Acknowledgement
Send outgoing SMS
Send outgoing SMS payload
Inform server that SMS has been sent

Figure 20. Communication between Argus Android Client, Argus Android Gateway and Argus Server
Abbreviations: synch SMS = synchronization SMS ; Ack SMS = Acknowledgments SMS

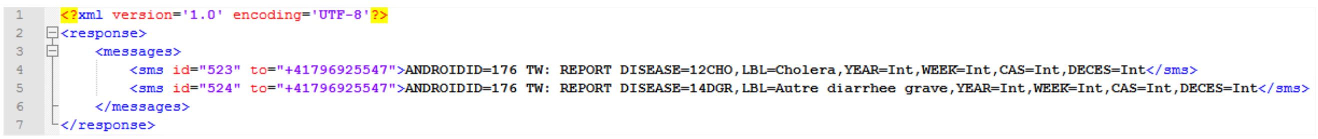
3.5.1 Inputs

Argus Android Gateway has two types of input:

- SMS inputs coming from the Argus Android Client application (see section 2.5 for examples).
- SMS payload coming from the Argus Server web application:
  - The Argus Android Gateway polls Argus server to get data. Argus Server never pushes data directly to the Argus Android Gateway application.
  - As an answer to the poll request from the Argus Android Gateway, Argus Server gives payload and destination information of the queueing SMS via HTTP requests and XML based responses.
    - Example of the XML answer from Argus Server after a poll request from the Argus Android Gateway if the SMS queue of Argus Server is empty:

```
<?xml version='1.0' encoding='UTF-8'?>
<response></response>
```

– Example of the XML answer from Argus Server after a poll request from the Argus Android Gateway if the SMS queue of Argus Server is not empty:



```xml
<?xml version='1.0' encoding='UTF-8'?>
<response>
    <messages>
        <sms id="523" to="+41796925547">ANDROIDID=176 TW: REPORT DISEASE=12CHO,LBL=Cholera,YEAR=Int,WEEK=Int,CAS=Int,DECES=Int</sms>
        <sms id="524" to="+41796925547">ANDROIDID=176 TW: REPORT DISEASE=14DGR,LBL=Autre diarrhee grave,YEAR=Int,WEEK=Int,CAS=Int,DECES=Int</sms>
    </messages>
</response>
```

- The XML answer contains two SMS that have to be sent by the Argus Android Gateway.
- The SMS field contains two attributes and a text content: the “id” attribute represents the unique id of the SMS on Argus Server side; the “to” attribute represents the destination phone number; the content is the payload of the SMS.

3.5.2 Outputs

Argus Android Gateway has two types of outputs:

- SMS outputs coming from the Poll answer of the Argus Server Web application (see section 2.5 for examples).
- SMS requests to Argus Server Web application. Even if the structure is almost identical, there are several outputs to Argus Server Web application. The form element “action” sends the action that has to be executed by Argus Server, among which:

- Test connection to the server:

  – HTTP POST to the server with specific content in the <form> element: **action**: “test”.
  – If Argus Server is reachable and sends a Status Code 200, then Argus Android Gateway considers that the connection is ok.

- Push incoming SMS to the server:

  – This happens when the Argus Android Gateway receives an SMS and pushes it directly to Argus Server.
  – It is an HTTP Post with specific content in the <form> element:
    - **action**: “incoming”;
    - **phone_number**: Gateway phone number;
    - **message_type**: “sms”;
    - **message**: message’s content;
    - **from**: sender’s phone number.
  – If the server sends a Status Code 200, then Argus Android Gateway considers that the SMS has been successfully forwarded to Argus Server.

- Poll queue SMS from the server:
  - The Argus Android Gateway polls every X seconds the server to get queued SMS.
  - It is an HTTP Post with specific content in the <form> element:
    - **action**: “outgoing”;
    - **phone_number**: Gateway phone number.
  - If Argus Server sends a Status Code 200, then the Argus Android Gateway considers that the server is reachable.
  - Argus Server will answer the Argus Android Gateway with a XML base response containing all the SMS to be send (see section 3.5.1).

- Acknowledge sending status to the server:
  - When Argus Android Gateway successfully sends an outgoing SMS or fail to send the SMS, it sends to Argus Server the status of the SMS transmission, identified with the server id.
  - It is an HTTP Post with specific content in the <form> element:
    - **action**: “send_status”;
    - **id**: unique id of the SMS on the server side;
    - **status:** status of the SMS (*sent*: when SMS is successfully sent, *failed*: when SMS has failed).
  - If Argus Server sends a Status Code 200, then the Argus Android Gateway considers that Argus Server is reachable and has registered the information.

- Send Gateway status to the server:
  - The Argus Android Gateway sends every 300 seconds its status to Argus Server.
  - It is an HTTP Post with specific content in the <form> element:
    - **action**: “device_status”;
    - **phone_number**: Gateway phone number;
    - **version**: version of the Gateway Android application;
    - **battery**: battery level of the Gateway Android phone (between 0 to 100 %);
    - **power:** power source (0: no power source, 1: USB, 2: AC);
    - **phone_operator:** network operator;
    - **poll_interval**: poll interval in seconds configured on the gateway.
  - If Argus Server sends a Status Code 200, then the Argus Android Gateway considers that Argus Server is reachable and has registered the information.

3.7 Argus Android Gateway and Argus Android Gateway Slave relations

When Argus Android Gateway application starts:

- It broadcasts an Android intent (ordered broadcast) that the installed Argus Android Gateway Slave applications installed will catch.
- Installed Argus Android Gateway Slave applications will add in the intent data their package name.
- At the end, Argus Android Gateway application will catch his own intent but with the added Argus Android Gateway Slave applications package names.

When a new Argus Android Gateway Slave application is installed on the device, the SlaveInstalledReceiver will catch the event and add it into the pool.

Steps followed by the Argus Android Gateway application to choose which app can send the next message (itself or one of the Argus Android Gateway Slave):

- The “maybeDequeueMessage()” function in the “Outbout” class is the principal entry for sending an SMS.
- The next queued outgoing message will choose which application is available to send it by calling “chooseOutgoingSmsPackage()” function of “App.java” file.
- A round-robin selection between all Argus Android Gateway Slave applications is made and a list containing all SMS already sent by the app with timestamp is checked to see if the application can send another SMS.
- If no application is available to send the SMS, a timer will retry the “maybeDequeueMessage” later.
- If an application is chosen, the SMS is sent by creating an intent using the application name and the sending intent suffix (OUTGOING_SMS) like “org.argus.gateway.slave01.OUTGOING_SMS”.

4. Argus Server web application

4.1 Overview

The previous name of Argus was initially SES. It explains why in the code, interface or database we can still have this naming convention. The first objective of the application was to manage incoming formatted SMS coming from the field. Those SMS were manually written and sent. It has then been decided to develop the Argus Android Client application to automatically send formatted SMS to avoid typos and facilitate reporting.

Argus Server web application manages the incoming SMS pushed by the Argus Android Gateway applications. When a new SMS reaches Argus Server, it is processed, and its content of interest stored in the MySQL database. ARGUS Server web application manages alerts, and weekly and monthly reports. The template of the received alerts and reports is stored on Argus Server web application and shared with the Argus Android Client applications through SMS (see section 2.5.1).

Argus server has been built to interact with existing Health Information systems. It uses input and output XML configuration files for its configuration (sites, contacts, diseases, thresholds) and to export data.

4.2 Technology

Argus Server web application is a custom PHP application (with no specific framework used) using a MySQL database.

Argus Server web application is coupled with Argus Dashboard web application. The two applications communicate by exchanging XML files.

XAMPP facilitates the installation of an Apache Distribution containing PHP and MySQL. The components provided with the installer (XAMPP v5.6.8) are totally compatible with Argus Server web application:

- Apache 2.4.12
- Php 5.6.8
- MySQL 5.6

It is possible to deploy Argus Server web application without XAMPP ensuring the right version of each above listed needed component is installed.

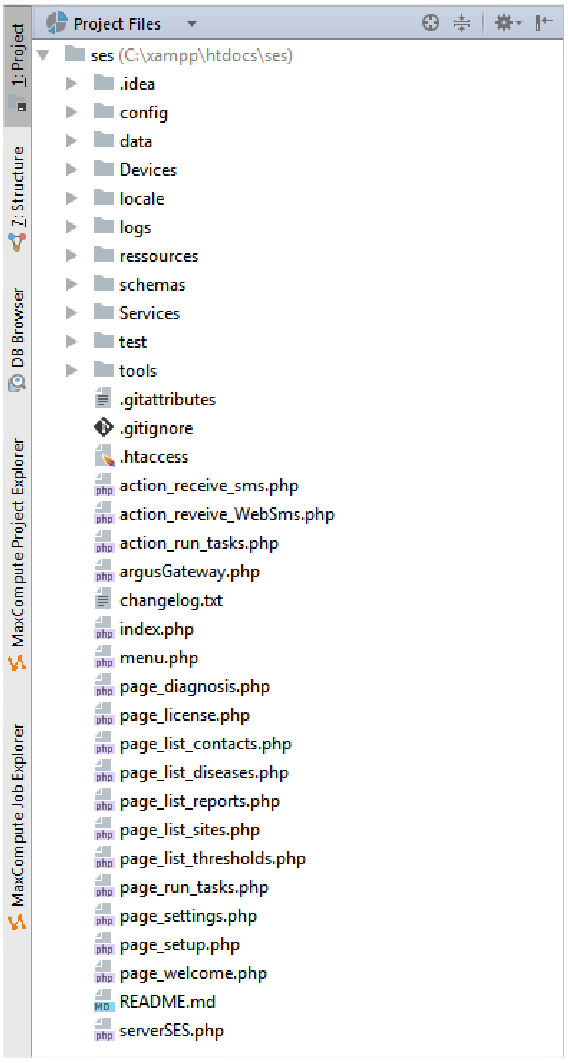
4.3 Source file structure

The source file structure of the project is the following (Figure 21):

- **.idea**: Php Storm folder.
- **config**: contains the configuration files of the application.
- **data:**
  - **input**: folder containing XML configuration files to process.
  - **output**: folder containing XML data files.
  - **processed**: folder containing XML configuration files processed.
  - **work**: folder containing XML configuration files in the process of being imported.
- **locale:** contains all translation resources files.
- **logs**: contains all log files.
- **ressources**: contains all XAMPP configuration files (PHP, MySQL, Apache), all database scripts, and sources of the Python monitoring application.
- **schemas:** contains XSD schemas to validate XML configuration files and XML data files.
- **Services:** contains PHP services class.
- **test:** contains files used to technically test the application.
- **tools:** contains all the PHP functions used in the whole application.

Files are available on “C:\xampp\htdocs\ses\”.

At the root folder are all PHP pages of the Argus Server web application and the argusGateway.php class which is the entry point for all Argus Android Gateway applications connected to the server.



```
ses (C:\xampp\htdocs\ses)
  .idea
  config
  data
  Devices
  locale
  logs
  ressources
  schemas
  Services
  test
  tools
  .gitattributes
  .gitignore
  .htaccess
  action_receive_sms.php
  action_reveive_WebSms.php
  action_run_tasks.php
  argusGateway.php
  changelog.txt
  index.php
  menu.php
  page_diagnosis.php
  page_license.php
  page_list_contacts.php
  page_list_diseases.php
  page_list_reports.php
  page_list_sites.php
  page_list_thresholds.php
  page_run_tasks.php
  page_settings.php
  page_setup.php
  page_welcome.php
  README.md
  serverSES.php
```

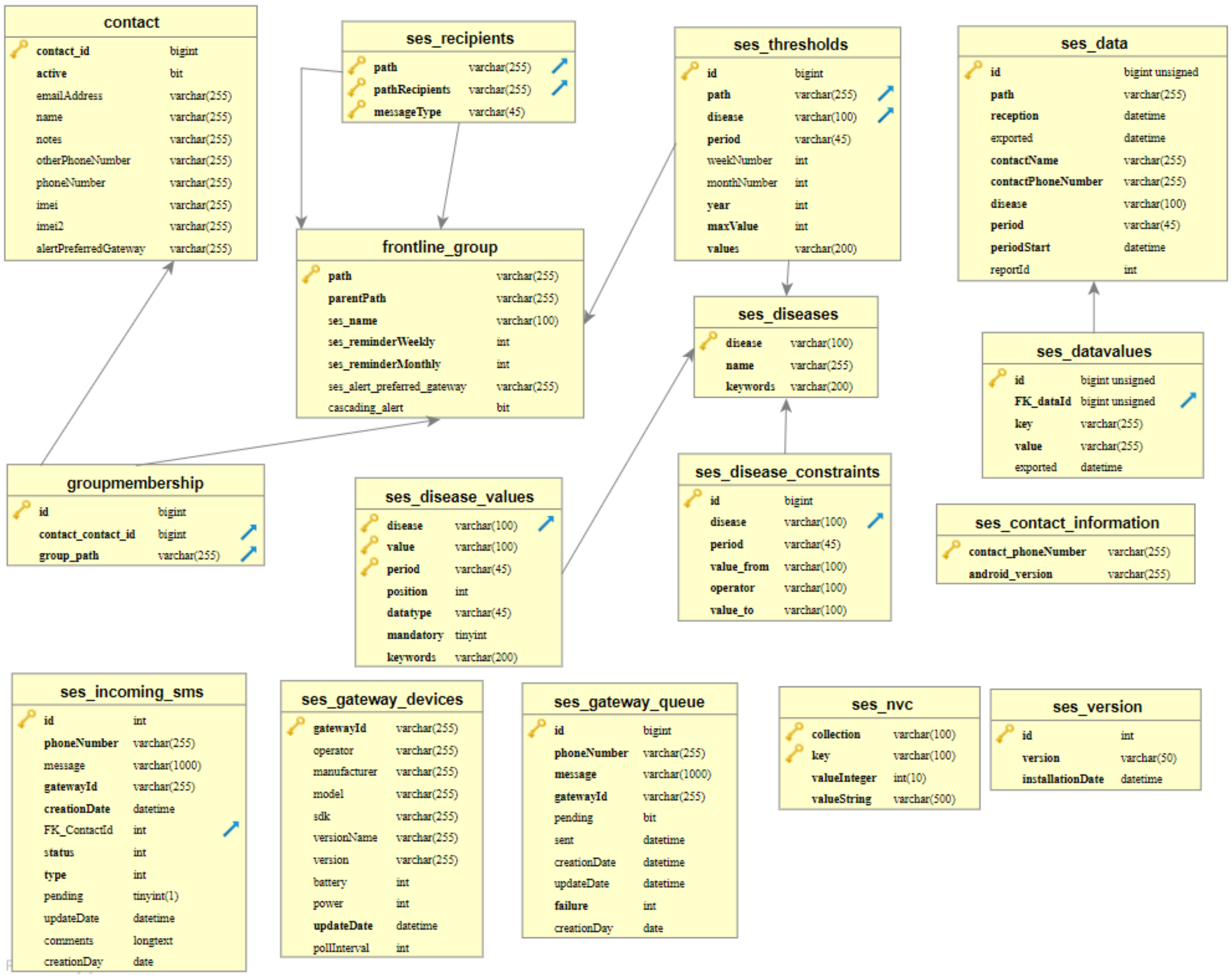
Figure 21. Global source file structure Argus Server web application

4.4 Database structure

Argus Server web application was initially developed on the FrontlineSMS database structure. All specific tables are prefixed with “ses_” and presented in Figure 22:

- “ses_data” stores all data received from the incoming SMS (weekly reports, monthly reports, alerts).
- “ses_nvc” stores all settings as key-value pairs configured in Argus Server web application.
- “ses_contact_information” stores the Argus Android Client application version used by the client. Data is stored when Argus Android Client asks for a synchronization.
- “ses_diseases” stores information about the diseases to be reported.
- “ses_diseases_values” stores the variables to be reported for each disease (e.g. nb of cases, nb of death).

- "ses_diseases_constraints" stores information about constraints between disease variables (for example, the number of cases must be superior or equal to the number of deaths).
- "ses_gateway_queue" stores the SMS intended to be sent by the Argus Gateway applications.
- "ses_gateway_devices" stores information regarding gateway devices connected to the server (e.g. model of the phone, GSM operator, battery).
- "ses_recipients" stores the paths where recipients are assigned to forward the alerts.
- "ses_thresholds" stores the definition of a threshold that can trigger some specific acknowledgement to the sender.
- "ses_version" stores the different versions number of the Argus Server web application installed and the date of the installation.
- "ses_incoming_sms" stores all the incoming SMS.



contact	
contact_id	bigint
active	bit
emailAddress	varchar(255)
name	varchar(255)
notes	varchar(255)
otherPhoneNumber	varchar(255)
phoneNumber	varchar(255)
imei	varchar(255)
imei2	varchar(255)
alertPreferredGateway	varchar(255)

ses_recipients	
path	varchar(255)
pathRecipients	varchar(255)
messageType	varchar(45)

ses_thresholds	
id	bigint
path	varchar(255)
disease	varchar(100)
period	varchar(45)
weekNumber	int
monthNumber	int
year	int
maxValue	int
values	varchar(200)

ses_data	
id	bigint unsigned
path	varchar(255)
reception	datetime
exported	datetime
contactName	varchar(255)
contactPhoneNumber	varchar(255)
disease	varchar(100)
period	varchar(45)
periodStart	datetime
reportId	int

frontline_group	
path	varchar(255)
parentPath	varchar(255)
ses_name	varchar(100)
ses_reminderWeekly	int
ses_reminderMonthly	int
ses_alert_preferred_gateway	varchar(255)
cascading_alert	bit

ses_diseases	
disease	varchar(100)
name	varchar(255)
keywords	varchar(200)

ses_datavalues	
id	bigint unsigned
FK_dataId	bigint unsigned
key	varchar(255)
value	varchar(255)
exported	datetime

groupmembership	
id	bigint
contact_contact_id	bigint
group_path	varchar(255)

ses_disease_values	
disease	varchar(100)
value	varchar(100)
period	varchar(45)
position	int
datatype	varchar(45)
mandatory	tinyint
keywords	varchar(200)

ses_disease_constraints	
id	bigint
disease	varchar(100)
period	varchar(45)
value_from	varchar(100)
operator	varchar(100)
value_to	varchar(100)

ses_contact_information	
contact_phoneNumber	varchar(255)
android_version	varchar(255)

ses_incoming_sms	
id	int
phoneNumber	varchar(255)
message	varchar(1000)
gatewayId	varchar(255)
creationDate	datetime
FK_ContactId	int
status	int
type	int
pending	tinyint(1)
updateDate	datetime
comments	longtext
creationDay	date

ses_gateway_devices	
gatewayId	varchar(255)
operator	varchar(255)
manufacturer	varchar(255)
model	varchar(255)
sdk	varchar(255)
versionName	varchar(255)
version	varchar(255)
battery	int
power	int
updateDate	datetime
pollInterval	int

ses_gateway_queue	
id	bigint
phoneNumber	varchar(255)
message	varchar(1000)
gatewayId	varchar(255)
pending	bit
sent	datetime
creationDate	datetime
updateDate	datetime
failure	int
creationDay	date

ses_nvc	
collection	varchar(100)
key	varchar(100)
valueInteger	int(10)
valueString	varchar(500)

ses_version	
id	int
version	varchar(50)
installationDate	datetime

Figure 22. Argus Server web application MySQL tables

4.5 XML and XSD files

All the configuration of Argus Server web application (i.e. diseases, sites, contacts, thresholds) and sharing of the alerts and reports data is managed by importing and exporting XML files. All XML files are validated against XSD schemes before each import and export, they are available in the folder "C:\xampp\htdocs\ses\schemas\".

There are four XML configuration files managing diseases, sites, contacts and thresholds. These XML files can be created manually and imported in Argus Dashboard web application to initialize the configuration. Administrators can also manage the configuration directly in the Argus Dashboard web application. Configuration set up in the Argus Dashboard web application will be applied to the Argus Server web application through XML files exchanged on the local file system.

There is one XML file containing incoming SMS data checked by the Argus Server web application. The Argus Server periodically exports all validated incoming SMS data an XML data file validated by the XSD scheme. These XML data files are periodically imported by the Argus Dashboard web application to update the database.

This use of XML files would allow keeping the Argus Server web application functionalities and replace the Argus Dashboard web application by any other health information management system tool (e.g. DHIS2) with minimum development work.

4.6 Input/output

4.6.1 Inputs

Argus Server web application manages the incoming SMS pushed by the Argus Gateway applications through a web service (see section 3.5.2).

Argus Server web application updates its configuration based on XML configuration files exported to « C:\xampp\htdocs\ses\data\input » on the local file system by Argus Dashboard web application (see section 4.5). This configuration update is triggered by Argus monitoring.

4.6.2 Outputs

When a new SMS reaches Argus Server web application:

- Argus Server web application processes it and stores the result in its tables on the MySQL database (see section 4.4).
- Argus monitoring export it in XML data file into the folder « C:\xampp\htdocs\ses\data\output ».

- Argus importer PHP script will import these XML data files into Argus Dashboard web monitoring.

Argus Server web application shares XML feeds containing the details of the SMS to be sent by the Argus Gateway applications through a web service (see section 3.5.1).

4.7 Business logic

Argus Server web application is in charge of all the management of incoming and outgoing SMS.

All the business logic is managed in the file named “C:\xampp\htdocs\ses\tools\AndroidVersionning\action_handlingV1.0.php”. Before calling this logic, the parsing of the SMS is done in the file “C:\xampp\htdocs\ses\tools\message_handling.php”. At this step, first controls are made like checking that the phone numbers are known or that the SMS format is correct.

If the number is unknown, the SMS is not processed and an acknowledgement SMS is sent back to the sender (acknowledgement can be activated or deactivated within the application configuration). If the SMS is malformed (e.g. unknown disease, missing keywords) Argus Server web application do not save the incoming data as it doesn’t know how to parse the message, but can send back a SMS explaining that a typo is present in the incoming SMS.

A specific algorithm has been developed to calculate the epidemiological week number, based on the first day of the week (Monday, Saturday, or Sunday). The first week of the current year is the week containing at least 4 days in the current year. Then other week numbers are calculated from this start.

All functions used for epidemiological week calculation and other more general date functions are available in the file “C:\xampp\htdocs\ses\tools\epidemiology.php”.

The first day of the week is set up in the variable $config["epi_first_day"] of the file: “C:\xampp\htdocs\ses\config\globals.php” and in the variable epi_first_day of the file “C:\xampp\htdocs\sesDashboard\app\config\parameters.yml”.

5. Argus Dashboard web application

5.1 Overview

Argus Dashboard web application is used to:

- Configure the system, the configuration is shared with the Argus Server web application using XML files.
- Manage all information received, especially the validation/rejection/aggregation at all levels of the weekly and monthly reports sent by healthcare facilities. Indeed, at each level of the system, users validate or reject the reports from their below level, and once reports are validated, aggregated reports are automatically generated at the above level.

5.2 Technology

Argus Dashboard web application is a custom PHP application developed with the Symfony 2 framework (https://symfony.com/) using a MySQL database. PHP Storm IDE was used during its development.

External JavaScript libraries are used, such as: Bootstrap (https://getbootstrap.com), Jquery (https://jquery.com), Moment (https://momentjs.com/), DateRangePicker (http://www.daterangepicker.com/).

Argus Dashboard web application is coupled with Argus Server web application. The two applications communicate by exchanging XML files (see section 4.5).

XAMPP facilitates the installation of an Apache Distribution containing PHP and MySQL. The components provided with the installer (Xampp v5.6.8) are totally compatible with Argus Dashboard web application:

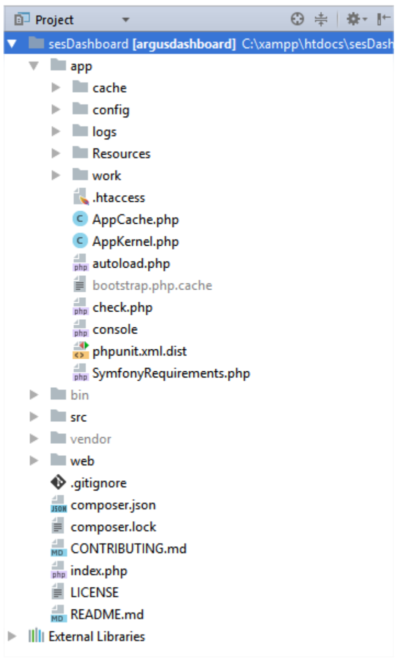
- Apache 2.4.12
- Php 5.6.8
- MySQL 5.6

It is possible to deploy Argus Dashboard web application without XAMPP ensuring the right version of each above listed needed component is installed.

5.3 Source file structure

The source file structure of Argus Dashboard web application is the following (Figure 23):

- app: Symfony 2 application files:
  - cache: cache generated by the Symfony engine.
  - config: configuration files.
  - logs: logs files.
  - Resources: all resources files used in the application (twig views, JavaScript files, css files, translation files).
  - work: work folder used to store imported data XML files coming from the ARGUS Server web application.
- bin: files generated by the Symfony engine.
- src: all PHP source files containing business logic.
- vendor: external vendors used in the solution.
- web: Symfony 2 web folder containing minified JavaScript, css, fonts files from the Argus solution and vendors.



```
Project
sesDashboard [argusdashboard] C:\xampp\htdocs\sesDash
  app
    cache
    config
    logs
    Resources
    work
    .htaccess
    AppCache.php
    AppKernel.php
    autoload.php
    bootstrap.php.cache
    check.php
    console
    phpunit.xml.dist
    SymfonyRequirements.php
  bin
  src
  vendor
  web
  .gitignore
  composer.json
  composer.lock
  CONTRIBUTING.md
  index.php
  LICENSE
  README.md
External Libraries
```

Figure 23. Global source file structure Argus Dashboard web application

5.4 Database structure

All specific tables of Argus Dashboard web application are prefixed with “sesdashboard_” and are presented in Figures 23 to 25.

5.4.1 System configuration

Tables presented in Figure 24 are used to store the configuration and are synchronized through XML files with the equivalent tables starting by “ses_xxx” used by Argus Server web application.

- “sesdashboard_sites”, “sesdashboard_sites_relationship” and “sesdashboard_indicatordimdate” store all the sites information. It is a hierarchical structure as a site can have a parent and children sites.
- “sesdashboard_contacttype”, not used (deprecated).
- “sesdashboard_contacts” stores all the contacts information. Each contact has a specific phone number. Each contact is attached to a specific site.
- “sesdashboard_sitealertrecipients” stores information to determine the recipients of the alert forwarding.
- “sesdashboard_diseases” stores all the diseases information.
- “sesdashboard_diseasesvalues” stores the variable needed to be reported for each disease (e.g. number of cases, number of deaths)
- “sesdashboard_diseasesconstraints” stores information about constraints between diseases variables (e.g. number of cases must be superior or equal to the number of deaths)
- “sesdashboard_thresholds” stores the definition of a threshold that can trigger some specific acknowledgement to the sender.

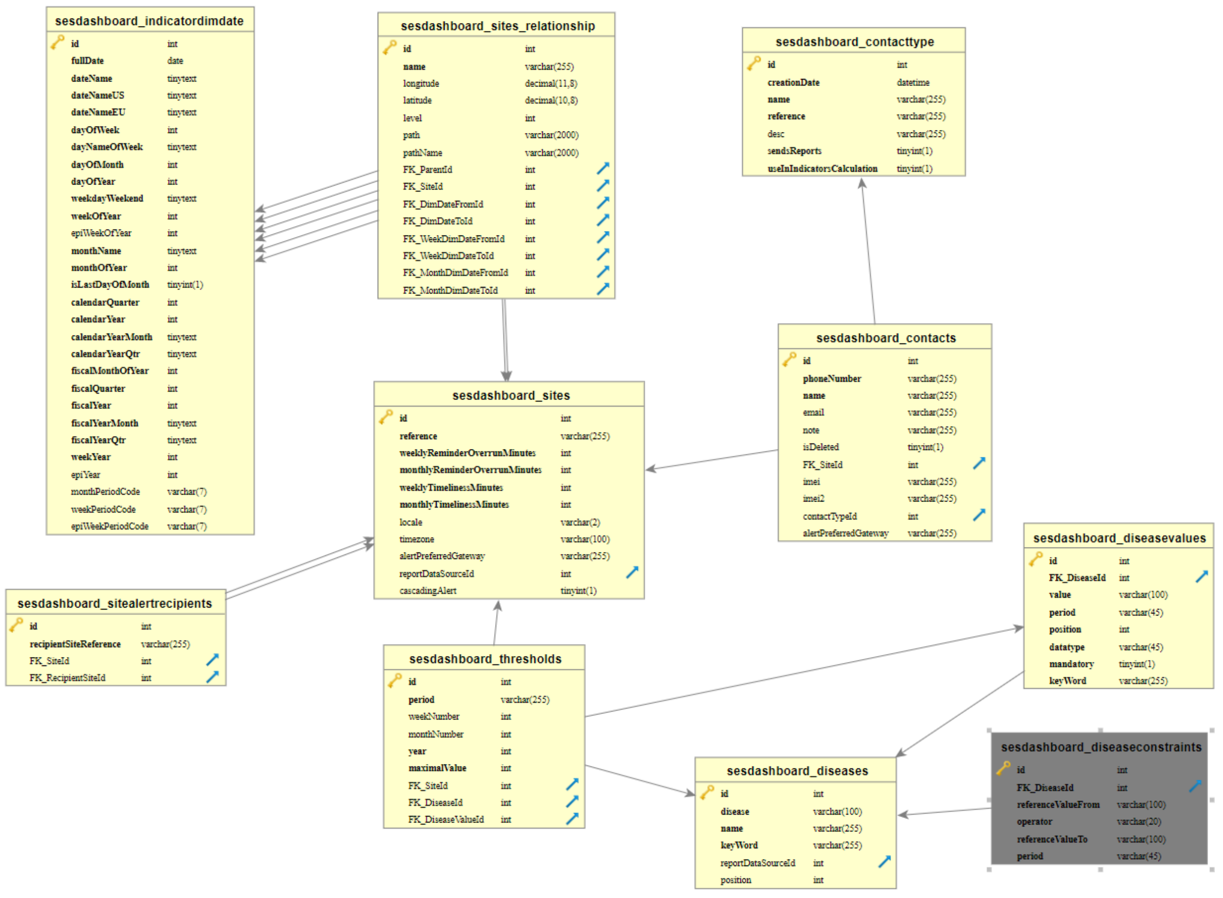


Figure 24. Tables storing system configuration data

5.4.2 Users and roles

The notions of user, role and permission are used to grant different users of the web platform, at different levels, different actions (e.g. visualization, validation, rejection, download), on different resources (e.g. weekly report, monthly reports, analyses).

A specific role is created which can contain none, one or several permissions.

A user of the web platform is assigned to one or multiple roles. Permissions of all assigned roles will apply to the user to allow or deny specific actions on specific resources.

Tables presented in Figure 25 are used to store the users, roles and permissions data:

- "sesdashboard_user" stores information about users having access to the web platform. There is no relation between users and contacts.
- "sesdashboard_role" store information about the roles created. A role contains a list of permissions giving access to some functionalities on the web platform.
- "sesdashboard_users_sesdashboard_roles" manage the assignment between users and roles.
- "sesdashboard_permission" store all permissions assigned to a specific role.

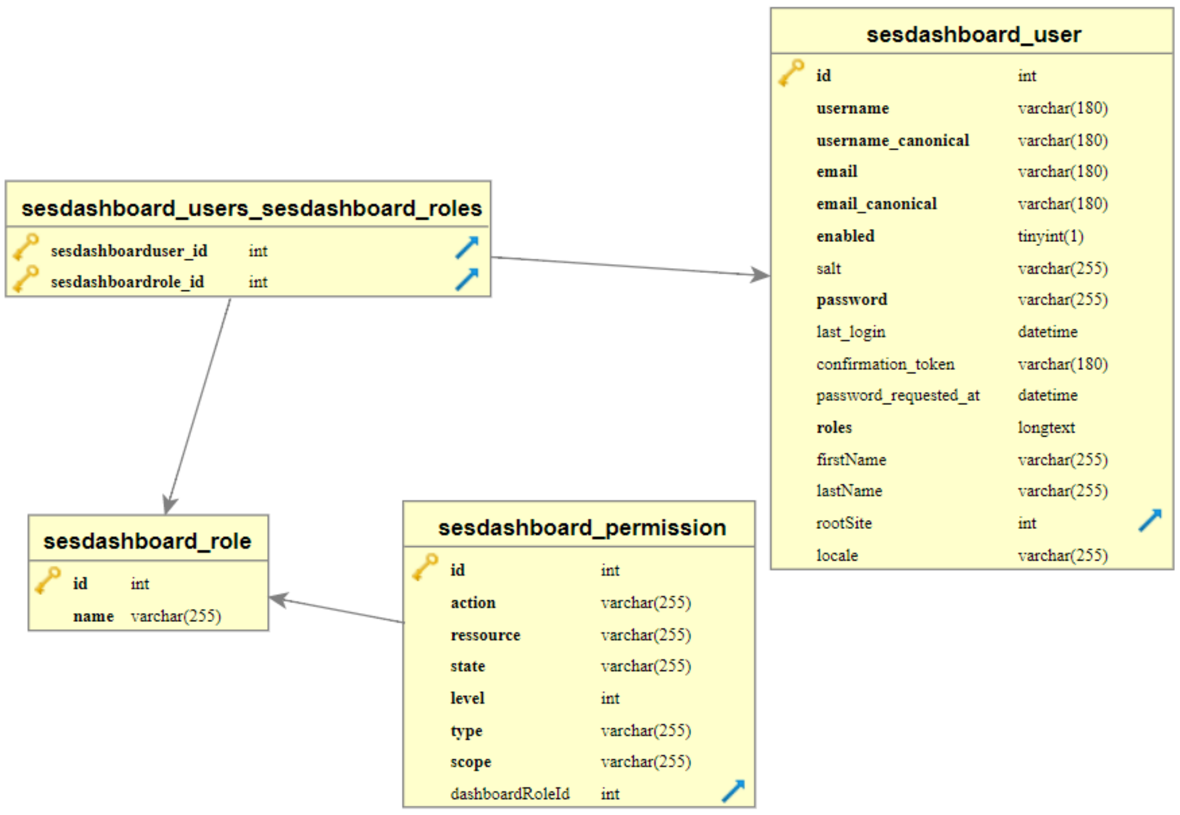


Figure 25. Tables storing user data

5.4.3 Reported data

Information received from the weekly and monthly reports is stored in different tables, see Figure 26. Figure 27 presents how they relate with a displayed report.

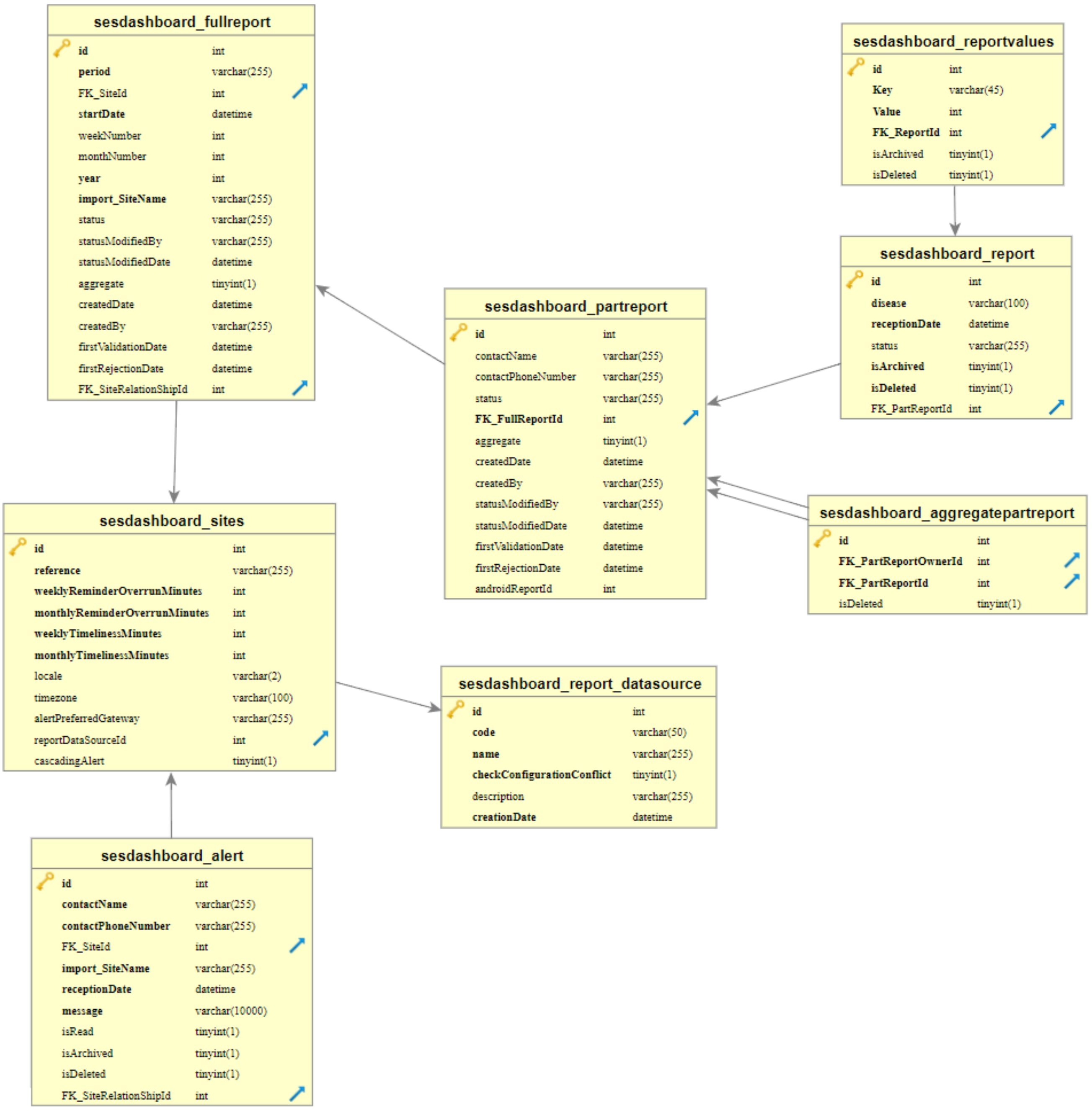
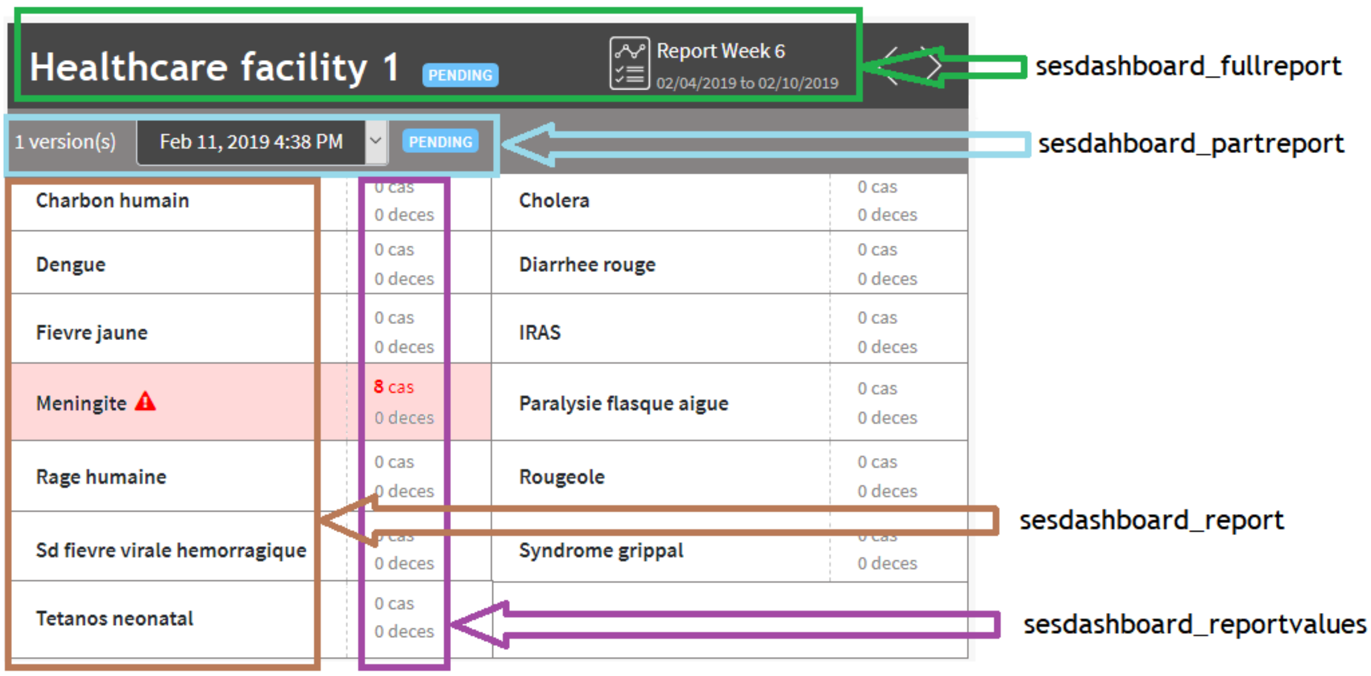


Figure 26. Tables storing reported data

- "sesdashboard_alert" stores the data from the alerts sent by the Argus Android Client applications.

- "sesdashboard_fullreport" stores data about a report for a specific site, week/month number and year. There is only one fullreport per week (or month) and per site.
- "sesdashboard_partreport" stores data about a specific version of a fullreport, including the contact who reported the data. We can have multiple versions of a partreport per week (or month) and per site.
- "sesdashboard_report" stores data about each disease reported and the reception date of the received SMS.
- "sesdashboard_reportvalues" stores data about the values reported for each disease like number of "cases" or number of "deaths".
- "sesdashboard_aggregatepartreport" stores information about all rows of "sesdashboard_partreport" used to calculate aggregated reports at above levels. (see below). Once validated, the reports received by the healthcare facilities are aggregated at the upper level. Thus, a version of a report (sesdashboard_partreport) can be an aggregation of multiple versions of reports at below levels.



Healthcare facility 1 PENDING

Report Week 6
02/04/2019 to 02/10/2019

1 version(s) Feb 11, 2019 4:38 PM PENDING

Charbon humain	0 cas 0 deces	Cholera	0 cas 0 deces
Dengue	0 cas 0 deces	Diarrhee rouge	0 cas 0 deces
Fievre jaune	0 cas 0 deces	IRAS	0 cas 0 deces
Meningite	8 cas 0 deces	Paralysie flasque aigue	0 cas 0 deces
Rage humaine	0 cas 0 deces	Rougeole	0 cas 0 deces
Sd fievre virale hemorragique	0 cas 0 deces	Syndrome grippal	0 cas 0 deces
Tetanos neonatal	0 cas 0 deces		

Figure 27. Logical elements of weekly and monthly reports stored in the database

5.5 Input/output

5.5.1 Inputs

The Argus importer PHP script is a command line function included in the Argus Dashboard web application. It is periodically executed via a Windows scheduled task. Every 5 minutes, the Argus importer PHP script checks the

"C:\xampp\htdocs\ses\data\output\" folder and imports new data received by Argus Server web application into the Argus Dashboard web application database.

It is possible to import XML configuration files via the user interface to facilitate the configuration of diseases, thresholds, contacts and sites.

5.5.2 Outputs

Argus Dashboard web application contains user interfaces to update/create the configuration and allows users to import directly XML configuration files in the system as described above.

When the configuration is modified, Argus Dashboard web application creates XML configuration files in the local folder: « C:\xampp\htdocs\ses\data\input ».

5.6 Argus Reports module

5.6.1 Overview

Argus Reports module produces the on-demand analyses and weekly epidemiological summaries.

5.6.2 Technology

Argus Reports module is a fork of an open source solution named "PHP reports" (https://jdorn.github.io/php-reports/, https://github.com/jdorn/php-reports). Changes made include:

- Replacement of Google charts by C3.js and D3.js.
- Addition of the wkhtmltopdf application (https://wkhtmltopdf.org/) to export charts and reports in PDF format.
- Addition of local Twig template (https://twig.symfony.com/doc/2.x/) to customize the layout.

Custom PHP scripts are used to compute the reports.

No specific framework is used, but external libraries are used: Wkhtmltopdf (https://wkhtmltopdf.org/), BootStrap (https://getbootstrap.com/), DateRangePicker (http://www.daterangepicker.com/), C3.js (https://c3js.org/), D3.js (https://d3js.org/), Jquery (https://jquery.com/).

XAMPP facilitates the installation of an Apache Distribution containing PHP and MySQL. The components provided with the installer (Xampp v5.6.8) are totally compatible with Argus Server web application:

- Apache 2.4.12

- Php 5.6.8
- MySQL 5.6

It is possible to deploy Argus Reports module without XAMPP ensuring the right version of each above listed needed component is installed.

5.6.3 Source file structure

The source file structure of Argus Reports module is the following (Figure 27):

- bin: contains the “wkhtmltopdf” executable.
- cache: cache generated by the application.
- classes: contains report types, report headers, report filters php classes.
- config: contains “config.php” file.
- dashboards: contains dynamic .json files used to create the Weekly epidemiologic report. Those files are generated by the Argus Dashboard web application.
- lib: contains base php classes of the application.
- public: contains css, js, fonts and images files.
- reports: contains php files that produce the analysis reports. Those files are generated by the Argus Dashboard web application.
- samples_dashboards: Dashboard samples.
- samples_reports: Reports samples.
- templates: contains twig files to render the application pages.
- vendor: external vendors used in the solution.

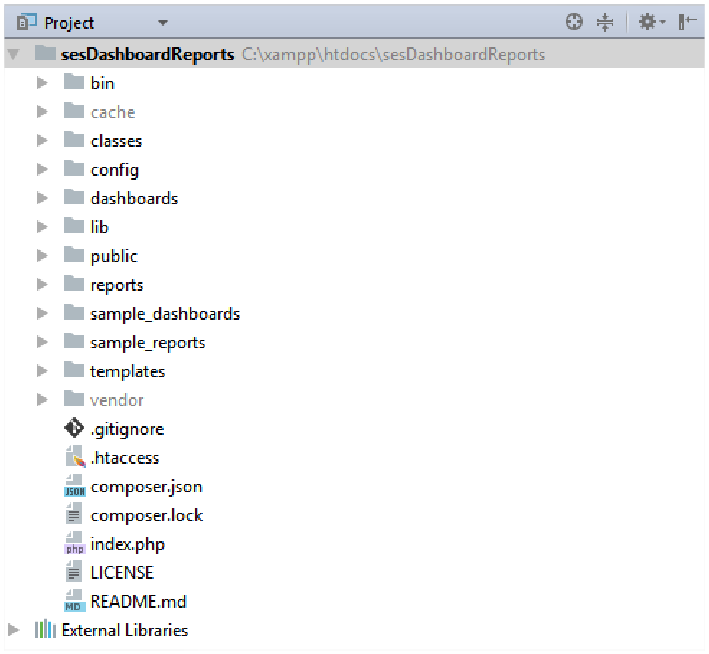

```
Project
sesDashboardReports C:\xampp\htdocs\sesDashboardReports
  bin
  cache
  classes
  config
  dashboards
  lib
  public
  reports
  sample_dashboards
  sample_reports
  templates
  vendor
  .gitignore
  .htaccess
  composer.json
  composer.lock
  index.php
  LICENSE
  README.md
External Libraries
```

Figure 28. Global source file structure Argus Reports web application

5.6.4 Input/output

Argus Reports collects the needed data from Argus Dashboard web application using web services.

It produces upon request custom analyses in HTML and weekly epidemiological summaries in PDF.

5.7 R integration

An R instance is launch on a routine manner through a Windows scheduled task to produce an administrative and epidemiological dashboard in HTML that will be displayed on the Argus web platform. Additional scripts can be added, and their outputs downloaded from the Argus web platform.

All the elements needed to produce the dashboards are located in the folder "C:\xampp\htdocs\SriptsR\", a readme file is available in the folder, the folder's structure is as below:

- dashboards: master script producing the dashboards.
    - assets: country maps for the epidemiological dashboard.
    - reports: outputs to be displayed on Argus web platform.
    - scripts: scripts called by the master script.
    - translations: translations to be used.
- db: database configuration.
- packages: binaries of R packages to be installed.