**Argus**

**Design Document**

Version: 0.10

Last Revision Date: 2016-02-12

**Table of Contents**

Argus - Design Document 3

Document Information 3

Revision History 3

Terminology - Abbreviations 3

1 Introduction 4

1.1 Purpose 4

1.2 Structure 4

1.3 Assumptions 4

2 High Level Design 5

2.1 System architecture 5

2.2 Design constraints 6

2.3 Development and Test environment 6

3 Detailed Design 7

3.1 Functions detailed design 7

3.1.1 Global Design 7

3.1.2 Synchronization 8

3.2 Class diagrams 9

3.3 Data detailed design 10

3.3.1 Data Structure 10

3.3.2 Specificities 10

3.4 Error management 11

4 ANNEXES 12

4.1 Server Action 12

4.2 Helpers 12

# 

# Argus -

## Document Information

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Project Name | Project Acronym | Client | Author | Document Status | Document Version |
| Argus | Argus |  | Alexandre Haulotte | Draft | 0.10 |
|  |  |  |  |  |  |

## Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author(s) | Revision |
| 0.1 | 2016-02-12 | Alexandre Haulotte |  |
|  |  |  |  |

## Terminology - Abbreviations

|  |  |
| --- | --- |
| Term / Abbreviation | Definition |
|  |  |
|  |  |

# Introduction

## Purpose

The purpose of this document is to describe the design of Argus for future developers.

## Structure

The document contains the main architecture of the application, some detail function design and a class hierarchy corresponding with each interface.

## Assumptions

This project target destination is underdeveloped country. The main challenge of Argus is to manage data in this environment.

# High Level Design

## System architecture

Access Layer

Background Receiver

Data Structure

Persistence

Dashboard:

ActivityDashboard

FragmentDashBoard

Report and Alert

Cf: 3.2 Class diagrams

Diagram 1 - 2.

History

ActivityHistory

FragmentHistory

AdapterHistory

Synchronization

ActivitySynchronization

FragmentSynchronization

Detail Report

ActivityReportDetail

FragmentReportDetail

Settings

ActivitySettings

FragmentSettings

History Detail

ActivityHistoryDetail

FragmentHistoryDeatil

SmsReceiver.java

Catch all sms

BootReceiver.java

Call when boot is finished

ServiceReminder.java

Catch system Alarm build in the application to remind not confirmed sms.

Config.java

Main keyword of data (dictionary for key like Diseases, month…)

Sms.java

Structure for saved sms in database

Status.java

Structure for sms status (Sent, Received, Error…)

TypeSms.java

Type of sms (Config, Alert, Report…)

SubTypeSms.java

SubType of config Sms (Model Report, Model Alert…)

TypeData.java

Data type of a field

(Int, Str, Week, Year…)

Database

Save config sms and sent sms

Preferences

Save the server number, last synch date and time…

## Design constraints

* The GSM network must pass all data shared by the application.
* The application must handle lost packet (SMS)
* The application must be dynamic to be adapted without recompiling the project

## Development and Test environment

The application is developed to support Android from version 9 to 21.

The application must be tested with multiple smartphones to ensure compatibility with all these versions.

# Detailed Design

## Functions detailed design

### Global Design

Dashboard

Report

Alert

History

Fill the Alert

For each disease one Report Details view

Fill all diseases

(All report detail view)

Sending:

Confirmation and Send

Get all templates sms in database for the selected Report or Alert

Generate all fields in function of there type

Fill the history adapter with all sent sms (from the database, however their status)

Show the history with each sms status

Open detailed history for report

Re-sending report or alert with an “error” status

### Synchronization

Server

Ask for Synchronization:

FragmentSynchronization

SmsReceiver.java

Parser.java:

Parse the sms to be sure it is for synch

HelperPreference.java

Save the sms in database

Count the number of received sms

FragmentSynchronization.java

SmsReceiver.java

Restart the application

FragmentSynchronization.java

## Class diagrams

Diagram 1: Report and Alert activities structure

AbstractActivitySesSms

AbstractActivityReport

ActivityAlert

ActivityReportWeekly

ActivityReportMonthly

Diagram 2: Report and Alert fragments structure

AbstractFragmentSend

AbstractFragmentReport

FragmentAlert

FragmentReportWeekly

ActivityReportMonthly

AbstractFragment

## Data detailed design

### Data Structure

In this project there is only one Table created in SesDatabase.java.

Table name : SMS

|  |  |  |
| --- | --- | --- |
| Field Name | Type | Description |
| \_ID | Integer, 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 receive with the same ANDROIDID |

### Specificities

Most of request are made using Cursor.

To do such request Classes need to extend from LoaderCallback<Cursor>. With this, classes must implement

* onCreateLoader to fill the request with arguments like a “where” condition in SQL.
* onLoadFinish to handle request result
* onLoaderReset.

In the project these loader are present in most of fragments classes.

**Eg :**

Call loader :

getLoaderManager().initLoader(*LOADER\_DASHBOARD*, null, this);

tag to identify the loader the loader callback

// Find all sms of type MODEL

public Loader<Cursor> onCreateLoader(final int identifiant, final Bundle bundle) {  
 String selection = SesContract.Sms.*TYPE* + "=?";  
 String[] selectionArgs = {String.*valueOf*(TypeSms.*MODEL*.toInt())};  
 CursorLoader loader = new CursorLoader(mContext, SesContract.Sms.*CONTENT\_URI*, null, selection, selectionArgs, null);  
 return loader;  
}

public void onLoadFinished(final Loader<Cursor> cursorLoader, final Cursor cursor) {  
 if (cursorLoader.getId() == *LOADER\_DASHBOARD*) {  
 if (cursor != null && cursor.moveToFirst()) {  
 Config.*getInstance*(mContext).loadDataFromCursor(cursor, mContext);  
 activateOrDesactivateButtons();  
 }  
 }  
}

## Error management

# ANNEXES

## Server Action

**The server can send “ANDROID\_SYNC\_REQUEST” to force a synchronization to refresh the application data.**

## Helpers

Much functionality is manage by Helpers. Here few examples.

* HelperSmsSender: All messages are sent with this helper.
* HelperPreference: Store and Load data from Preference AND Database.
* …