Functional specification document for BonoPastore

Team: White Hats

# Project scope

BonoPastore is a multi-component service that aims to give its users help in being, as the name suggests, good shepherds for their friends or even smart objects. In case of disasters everybody wants to know about their dear ones or about their possessions. But how much information can you know about them, when the disaster strikes? Using the BonoPastore software suite the level of known information rises significantly.

# Project features

BonoPastore separates disasters in two ways :

* public disasters, representing huge natural disasters which affects all people (earthquakes, floods, tornadoes).
* private disasters, representing isolated disasters signaled by users of different devices which belong to the users (fire, car crash, s.a)

The first feature of the application is to receive notifications from authorized organizations (like ISU) and send them to users in the unexpected event area. The responsible application within the BonoPastore suite for this is the main server. It will collect the information, structure it, and then forward it to the users.

The second feature of the BonoPastore suite represents the collection of the last locations of the users. Here, the client application will send the current location to the alert server. After that, the alert server will be responsible for sending the positions to the main server.

Another important feature is warning your friends if you are in a conflict zone. Your friends do not need to be in that area of danger. People you set as friends will be able to request your last locations from the main server.

Another feature allows the user to report an isolated disaster to the application. BonoPastore will alert all your friends.

# Operating Environment

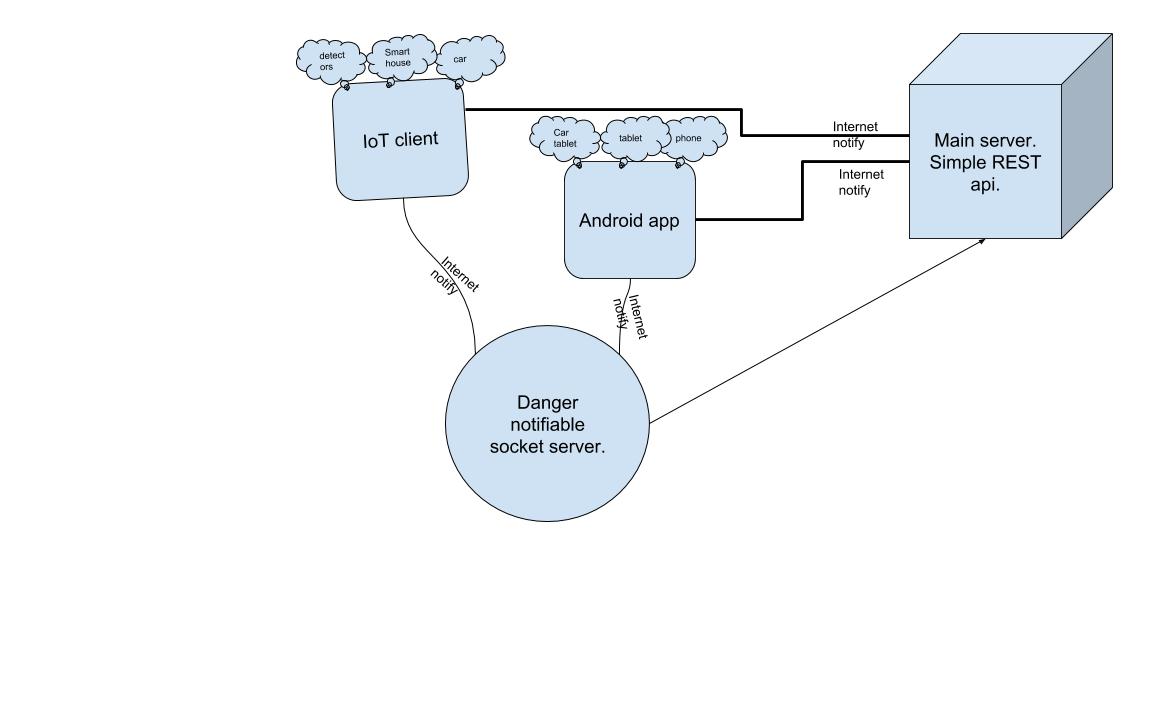
**

Figure 1 - BonoPastore diagram

BonoPastore offers 2 main services:

* disaster prevention
  + http normal server - sends notifications (push, etc)
* provide help
  + Hardcore TCP socket [danger notifiable server] - receives simple requests for help

The BonoPastore Suite is made up of 4 applications. The first application, also called the main server, has the task of communicating with the client application in particular.

The client application is developed in Android.

The alert application comes as a necessity to send small data packets. In the case of huge events, it is very important for the network to be as free as possible.

The IoT client application is represented by an object (car, house, etc.) whose protection is monitored. Any unwanted event occurs will be sent to the alert server.

# System Features

## 4.1.IOT client

Every user will be able to register one of his personal items and be informed about his status. The application uses an esp8266 and a water sensor.The application will track the humidity in the area of the registered object. If abnormal values are detected, the esp8266 will connect to the internet and send a signal to the alert server specifying the level of the situation. The alert server will report the situation to the main server, which will notify the owner of the object.

## 4.2.Alert server

It is a python server which is a link between the client and the main server. The server uses simple socket connections to save from bandwidth. In critical situations, the use of normal sockets can add to the activity on the network.

## 4.3.Client application

The client application is made in android. This allows users to register. A user once added will be notified by the main server about the disasters that affect him personally or his / her friends / objects. The client application will send the user's coordinates constantly to the alert server.

The user can decide what notifications they will receive. A user can decide on how important the notifications are for him.

*{This template illustrates organizing the functional requirements for the product by system features, the major services provided by the product. You may prefer to organize this section by use case, mode of operation, user class, object class, functional hierarchy, or combinations of these, whatever makes the most logical sense for your product.}*

## 3.1. System Feature 1

*{Don’t really say “System Feature 1.” State the feature name in just a few words.}*

### 3.1.1 Description and Priority

*{Provide a short description of the feature and indicate whether it is of High, Medium, or Low priority.}*

### 3.1.2 Stimulus/Response Sequences

*{List the sequences of user actions and system responses that stimulate the behavior defined for this feature. These will correspond to the dialog elements associated with use cases.}*

### 3.1.3 Functional Requirements

*{Itemize the detailed functional requirements associated with this feature. These are the software capabilities that must be present in order for the user to carry out the services provided by the feature, or to execute the use case. Include how the product should respond to anticipated error conditions or invalid inputs. Requirements should be concise, complete, unambiguous, verifiable, and necessary. Use “TBD” as a placeholder to indicate when necessary information is not yet available.}*

*{Multiply section 3.1. for as many features as the application has.}*