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. We accomplish this by easing the communication between the ones mentioned. The goal for this project is to let anybody or anything be the user. From embedded devices to official authorities, with BonoPastore anyone can be a good shepherd and notify it’s clients or help them in case of disasters.

# Project features

In order to address a wide range of users and a wide range of usages, the BonoPastore suite is built upon the logic that disasters can be separated in two main categories :

* public disasters, representing huge natural disasters which affects all people (earthquakes, floods, tornadoes). Information about those are usually received from official authorities.
* private disasters, representing isolated disasters that usually have to do with a single user or a small group of users. Those are signaled by users or different devices which belong to the users (fire, car crash, s.a).

For the moment, the suite provides 4 main features:

1. A web application by which an authorized organization (like ISU) can signal and manage disasters on the map and send them to the main server which will store the information about the disasters.
2. Remember a set of last locations about every user so they can be approximately found in case of a disaster. The locations are easy on network since they are sent through basic network sockets and then received by the main server.
3. When public disasters are triggered, users in the proximity of the zone are notified. If a user is in the radius of the disaster zone, his friends will also be notified.
4. The objects owned by a user can send private disasters alerts to its owners.

# Operating Environment

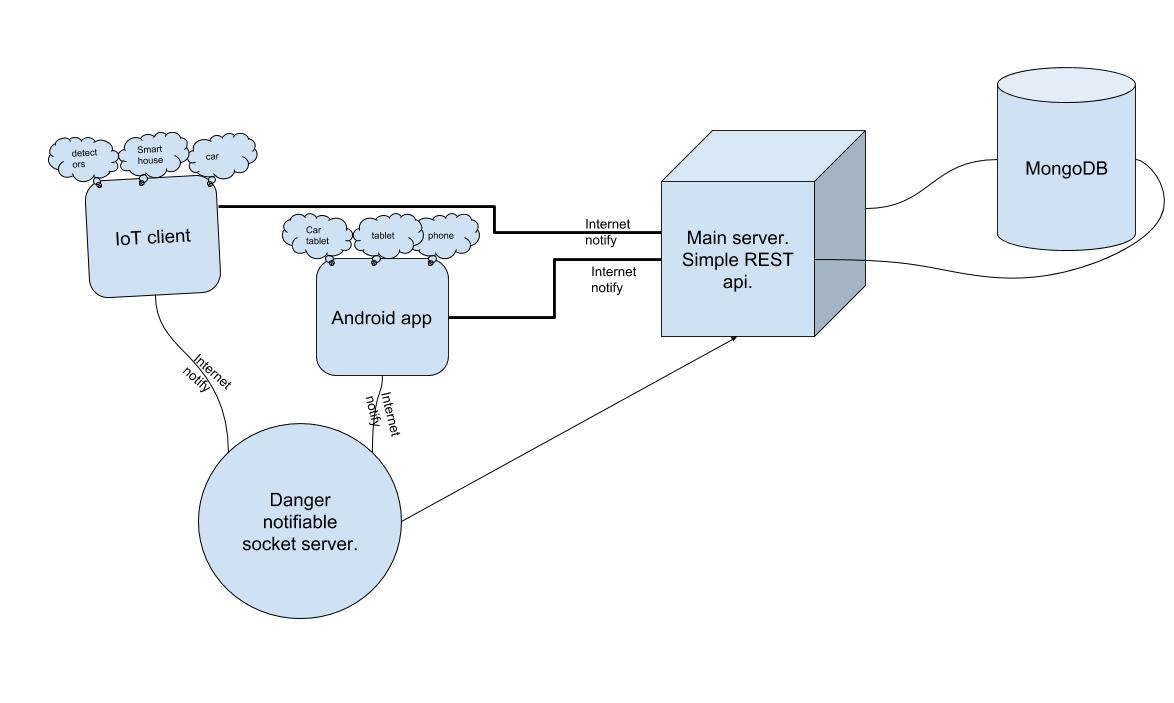
**

Figure 1 - BonoPastore diagram

The BonoPastore Suite is made up of 5 applications:

1. The first application, called *main server*, is written in nodejs. So in order to install it the node engine needs to be present in the environment and also the *node package manager* (npm). This server will act as a RESTApi, serving requests on a custom port through http requests. Also it needs a running *mongodb*.
2. The notification management web page is a simple html page that operates with the main server via javascript. If the *main server* is up, it doesn’t need anything more, than a token key for the mapbox service.
3. The client application is developed in Android and gives the main support for users to see the disasters on a map. It needs a smartphone with at least Android 8.0. Also the *main server* needs to be up.
4. The notifiable socket server is a python server and it 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. It runs standalone.
5. 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.

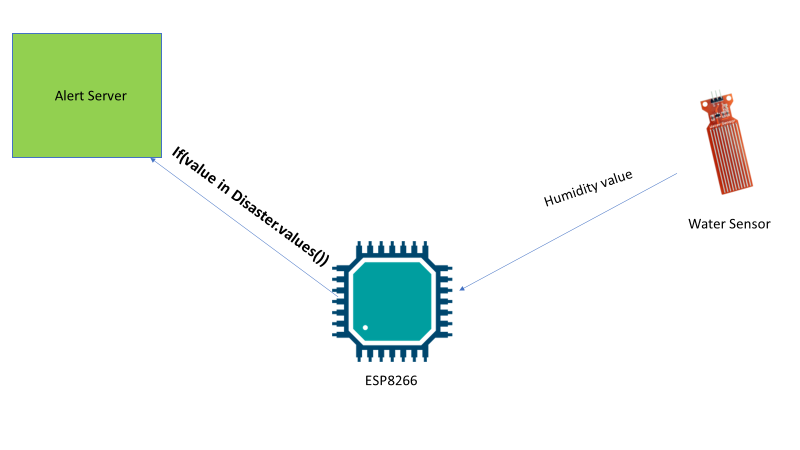


Figure 2 – IoT diagram

## 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.

## 

Figure 3 – Client app diagram

## 4.4.MAIN SERVER

The main server is the center of all actions. CRUD, Express and MongoDB are big implementations for this server.

[Express](https://expressjs.com/) is a framework for building web applications on top of Node.js. It simplifies the server creation process that is already available in Node. In case you were wondering, Node allows you to use JavaScript as your server-side language.

[MongoDB](https://www.mongodb.com/) is a database. This is the place where you store information for your web websites (or applications).

[CRUD](https://en.wikipedia.org/wiki/Create,_read,_update_and_delete) is an acronym for Create, Read, Update and Delete. It is a set of operations we get servers to execute (POST, GET, PUT and DELETE respectively). This is what each operation does:

* Create (POST) - Make something
* Read (GET)\_- Get something
* Update (PUT) - Change something
* Delete (DELETE)- Remove something

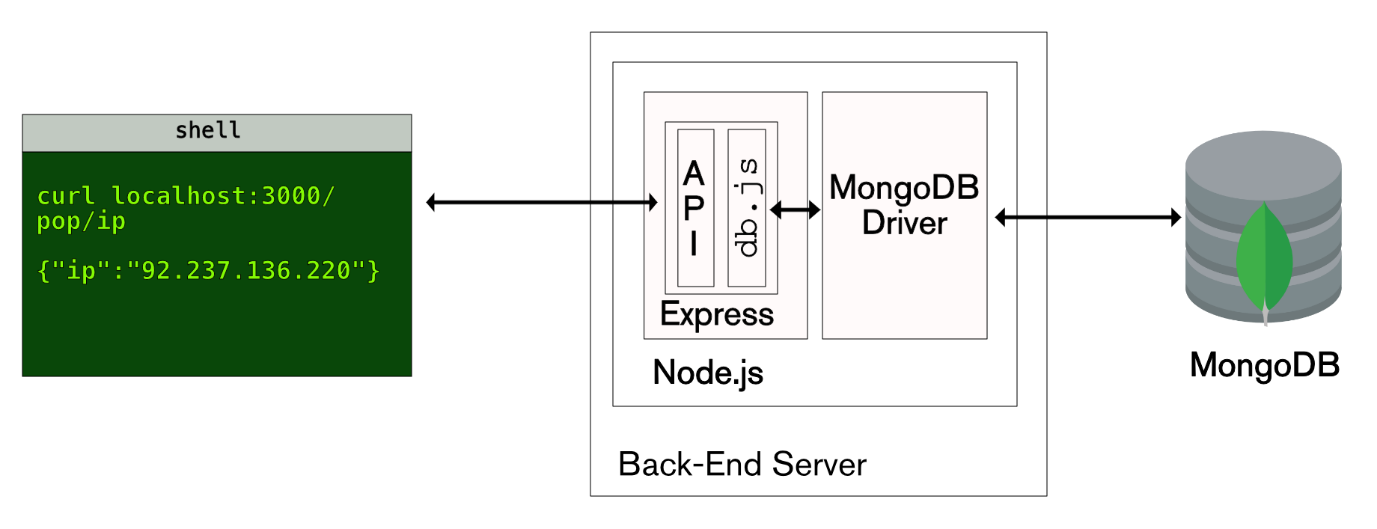


Figure 4 – Main server diagram

## Notification management web server

