SenSys: A Health Monitoring System

# Requirements Definition

1. **Introduction and context**

With technological advances in medical and healthiness areas, the demands on personal health tracking are increasing for many purposes, from everyday fitness, patients monitoring to military applications. On the market, it is common to find all types of wearable devices that can read heart rates, blood pressure, temperature, sugar level and such. However, most of those devices work separately as sensors to the only tasks they were designed for. We deemed that is a waste of potential diagnostic power, which can be achieved through combination of data from many such sensors. Therefore we propose a system that can combine all health sensors on one person to accurately predict his/her/its healthiness level, give advises, or even call emergency support on time when the client is unable to. In other word, a system that can monitor people’s health and save lives.

On a personal level, such system would only require the customer to put on sensors that he/she/it can afford, and work on that limited information. On large scale level, the system learn from data coming from thousands of customers, plus medical advises from experts, to predict and give advises to people in need as early as possible. A mobile phone should act as a medium between the system and customer and the sensors. We call such system as Sensys.

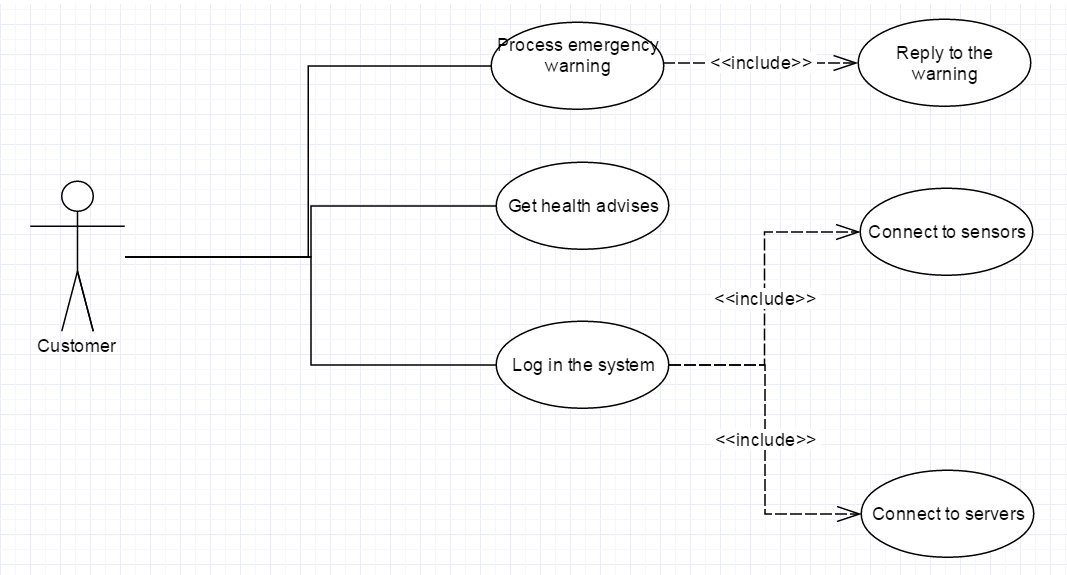
However, deployment for such system requires too much effort on non-software development aspects, such as finding customers, hiring experts to help building an analysis model for the data, getting contracts with hospitals and medical services, etc. Therefore, we decided to build a simulation of Sensys to demonstrate the effectiveness of the system.

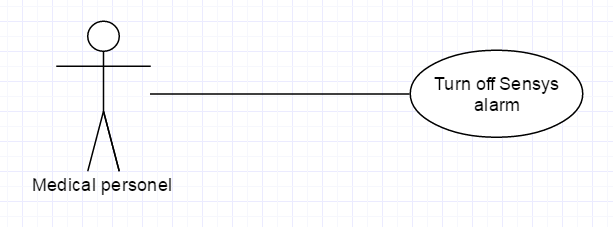
This document describes the user goals and requirements for a simulated Sensys. Section 2 describes the user and their goals in more detail. Section 3 describes functional requirements for a proposed set of software features that will satisfy these user goals. A suggested overall organization of the user-interface for these features is described in Section 4. The software will be built using an incremental development process, constrained by the non‐functional requirements enumerated in Section 5.

1. **Users and their goals**

The UML Use Case Diagrams in Figures 1 describe the key actors and user goals for *Sensys.*

**Figure 1 – Key actors**





As shown below, the only human actors in this system is the customer and the medic, we expect them to be familiar with using a smartphone and be able to use our app to connect with the sensors or turn off the alarm with their code. Such tasks should be trivial enough for them to understand and learn on how-to.

Below is an example scenarios that illustrates a couple of key user goals.

Scenario #1

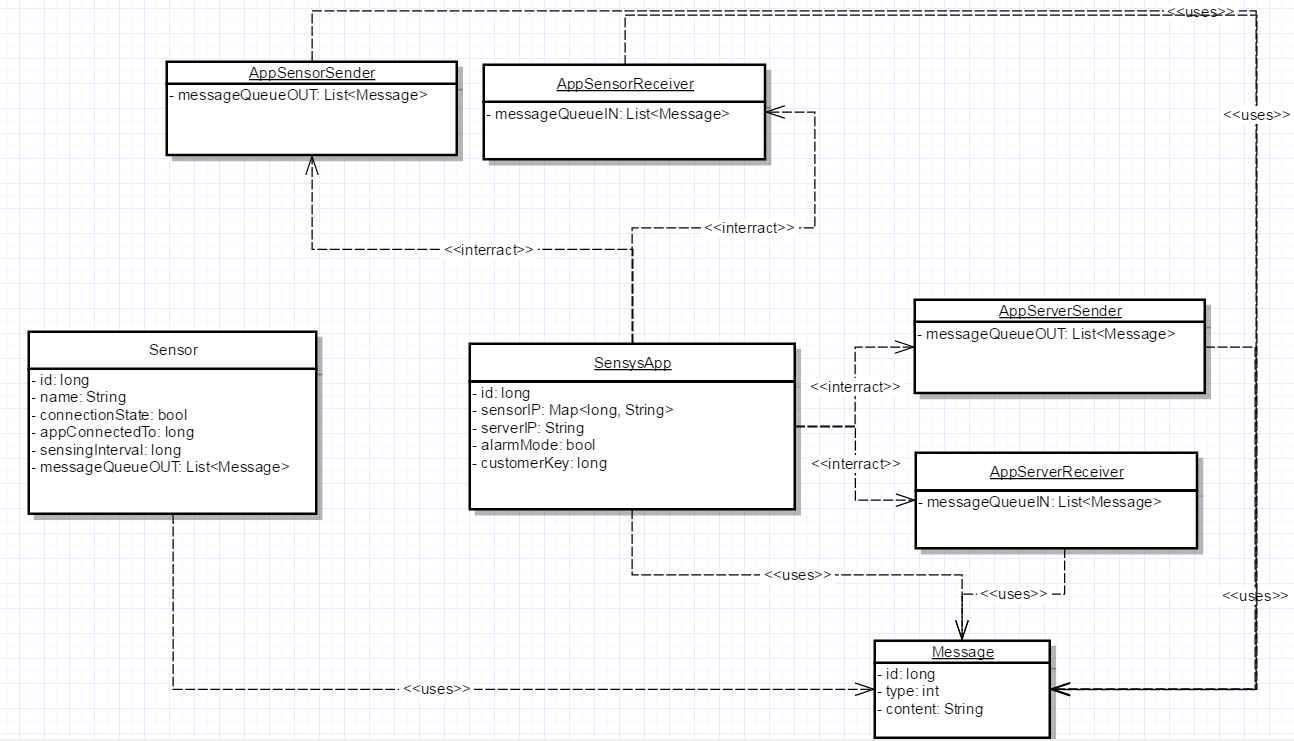
1. Bob, a patient with diabetes, wears a sugar level sensor and a blood pressure sensor.
2. He logs in Sensys app on his phone using user name and pin number provided before.
3. He turns on sensor detector mode of the app, and turns on the two sensor he has. The app detects the two sensors and connect to them automatically.
4. He then proceeds to eat three large big macs with extra cheese, and drinks a gallon of diet coke. 15 minutes later, his sugar level spikes up.
5. His phone gets a message from Sensys, saying: “Your sugar level is at dangerous level, we recommend you to run very slowly for 15 mins on a treadmill to burn the extra calories you have consumed.”
6. Bob ignores the message, and goes to his office up stair to work. He decides to walk the stars instead of riding the elevator because Elena is using the elevator, and he does not want to embarrass himself by making it goes “over-weight limit” again in front of her. On the third stair, his blood pressure goes up, Bob gets a heart stroke and falls down the stairway.
7. His phone gets a message from Sensys, saying: “Your sugar level and blood pressure is at an extremely dangerous level, please reply our message saying you are okay within 60 seconds or we will call a nearby paramedic”.
8. Bob does not respond because he is unconscious.
9. Sensys actives the emergency light and alarm on his phone and calls the building’s paramedic.
10. The paramedic arrives, turn off the alarm with his emergency code, gives Bob some CPR and transfer him to the local hospital.
11. Bob is alive. He could have been in critical condition if he did not receive help in time, but he is fine now.

The following table contains common terminology for this domain and their definitions:

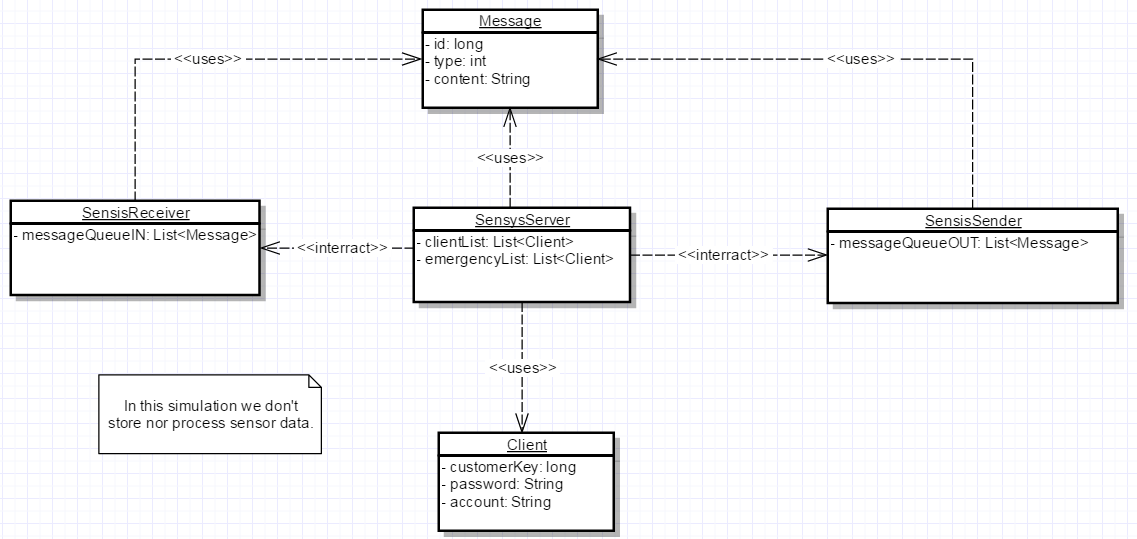
|  |  |
| --- | --- |
| **Common Term** | **Definition** |
| Paramedic | A healthcare professional, predominantly in the pre-hospital and out-of-hospital environment, and working mainly as part of emergency medical services (EMS), such as on an ambulance. |
| Health sensor | A wearable device that reads specific bodily information. |
| Emergency code | A code unique to the paramedic or hospital given by Sensys. |

1. **Classes of objects and the relationships**

**Figure 2** – Sensor reads Simulator Components



**Figure 3** – Server Side Simulator Components



Simulation data:

Simulation data are just random numbers sending over the sensors to test if communication works.

1. **Functional requirement**

The following functional requirements describe features that will satisfy the user goals listed in section 2.

1. User authentication

1.1. The system must require all users to authenticate themselves before establishing a communication line.

1.1.1. On login, the customer enters their given id and pin (given to them before).

1.1.2. If the credential is correct, the app may receive a key credential from the server. This key needs to be included in every data messages sent to the server.

1.2. The Sensys server only accepts one connection to an account at a time. If a second phone requests, no key credential will be sent to it.

1.3. User can log out any time and server must be able to end communication when that happens.

2. Sensor data message

2.1. Each sensor has a time interval for reading, the sensor read message must include the last time it was read.

2.2. The messages must be ensured to arrive.

3. Maintaining communication

3.1. Both the app and the server must know when a connection is lost.

3.1.1. Both can attempt to reconnect for 5 minutes.

3.1.2. If connection is lost for more than 5 minutes, user has to re-authenticate again.

3.2. The app must tell the server when user choose to close the connection.

4. Sensor connection

4.1. The app must automatically detect and connect to the sensors when they are turned on.

4.2. The sensor can only connect to one app at a time, until connection is lost or app is closed or disconnected.

5. Emergency mode

5.1. When detected critical condition, server and app act on their own clock.

5.1.1. Server sends emergency clarification to app.

5.1.2. Server calls external emergency agency if it does not receive reply after 60s.

5.1.3. If the connection is through, and customer does not reply within 60s since the app received emergency message, app automatically updates its location to server and turn on alarm and light.

5.2. To serve emergency in case of losing connection, app sends location periodically.

5.3. Medical personnel handling

5.3.1. App turns off emergency mode and disconnect sensors when handler’s code is inputted.

5.3.2. App sends handled message back to server to notify the situation has been handled. This message must include the code of the agency handling it.

5.3.3. Close the app and all connections when the above things are done.

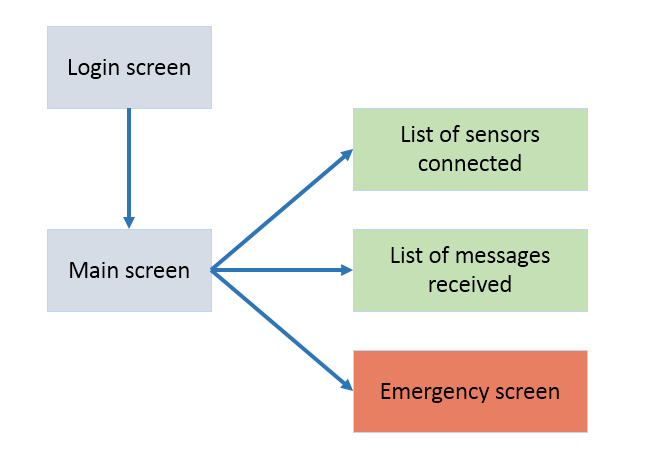
5.1. On emergency mode, server must act on its own clock.

5.1. On emergency mode, server must act on its own clock.

1. **Non-functional requirement**
2. Application‐level logic will be tested using Unit tests.
3. Emergency situation must be prioritized above all other cases.
4. The simulation must be able to simulate on at least 20 apps and the maximum number of sensor per app can be up to 5 sensors.
5. The app is on Android devices.
6. The server should be protected against direct attacks.
7. There is not much data flying about the system, but all are critical data, therefore the system should be reliable.
8. **User Interface Organization**

Figure 4 illustrates an initial (and incomplete) user‐interface organization. This is to be considered a starting point for design, not a definitive list of screens and possible navigations.

**Figure 4** – User interface organization for the app



|  |  |
| --- | --- |
| Screen | Purpose/content |
| Login screen | Allow user to login with id and pin. |
| Main screen | Allow user to navigate to other screens or close app. |
| List of sensor connected | List the sensors that are connected to the app |
| List of messages received | List all the messages received from the server |
| Emergency screen | Goes into this screen after receiving emergency message from the server. It has a replying box for user to reply to the server saying he/she/it is ok and an emergency code box for the medics to input their code. |

1. **Future Features**

Below is a list of future features:

1. Allow connection to actual sensors, not just fake processes.
2. Utilize the notification feature of the mobile systems to handle messages and run app in background.
3. Develop apps in Windows and iOS.
4. Put in actual data processing models for the server to handle actual data.
5. Increase scale of the system to thousands of customers with dozens of sensors each.