Fall Detection Response

—

By: Iliya Karac 100703933, Nick Kvrgic (100613386), Pranjal Saloni (100653360)

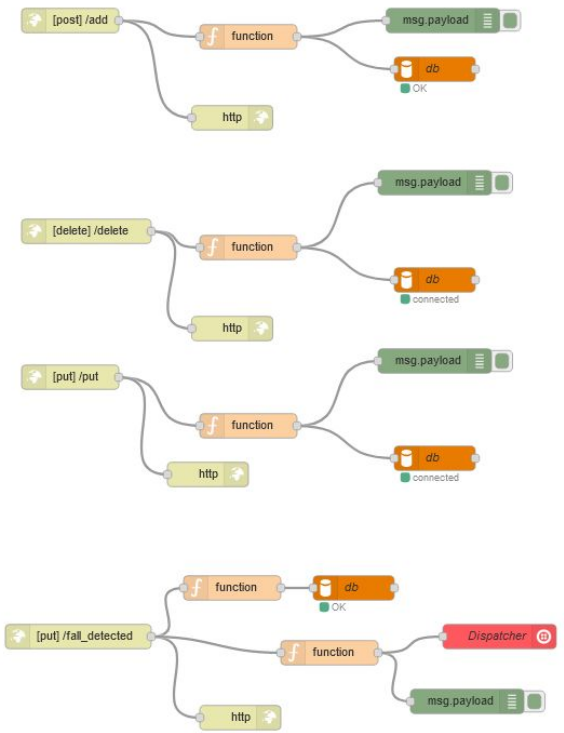
Fall detection Response

Our service provides end users with help in the event if a fall.

The service can be set up for private users in their own homes or in any care facility that needs to monitor their patients health status while allowing them the liberty of walking unattended. A few examples of this are hospitals and retirement homes.

In the case of an accident a dispatcher will immediately deploy emergency services directly to the patients location.

Flows



-these flows provide a restful api for external http requests

-the last one is the action flow where the functionality of the edge device is handled appropriately

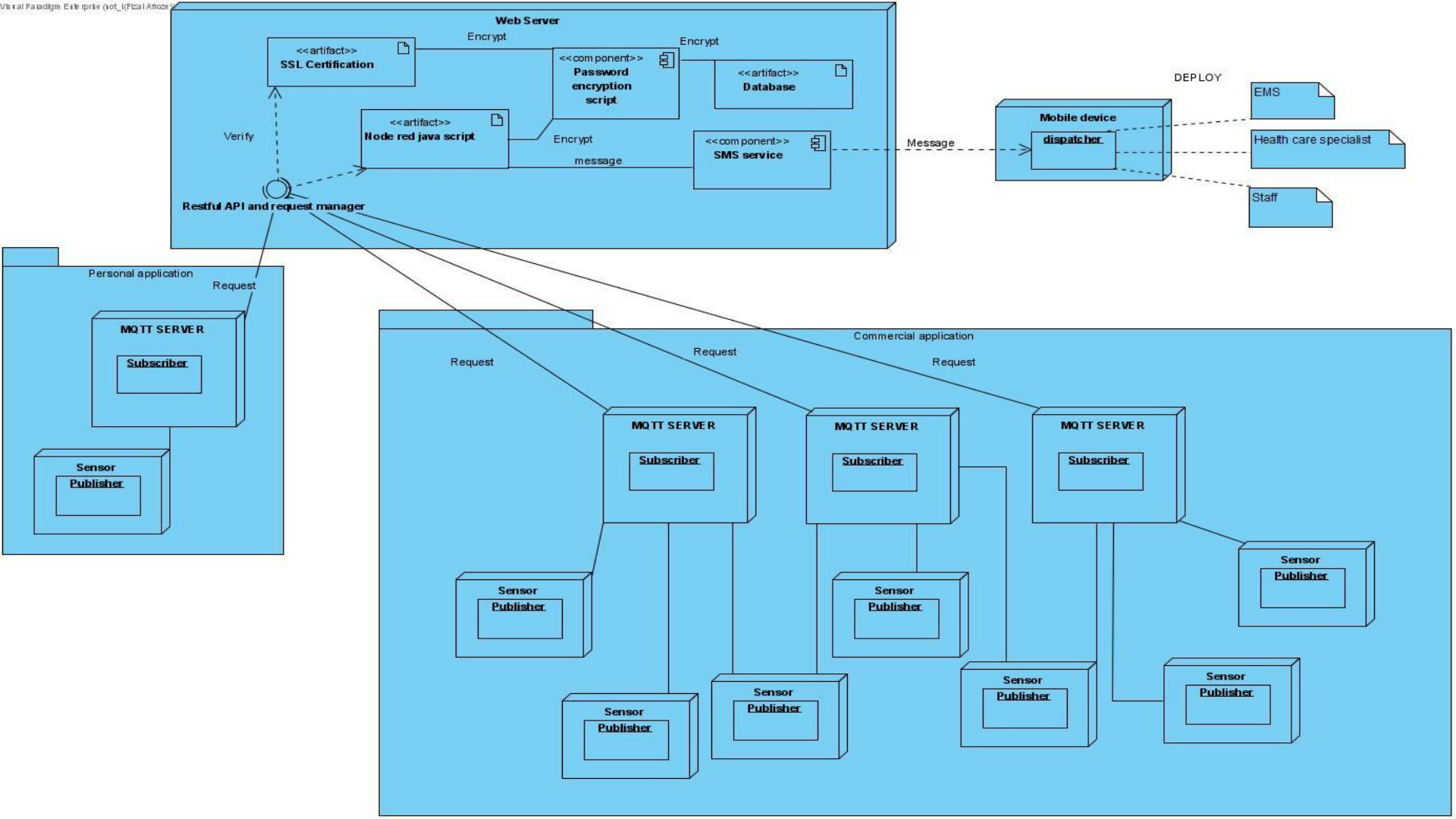
-the database it links to is a simple mysql database that stores data

Architecture design

Our system follows the basic flow of a sensor that creates a message when a fall is detected, it is sent to the MQTT server that then sends an https request to the server with certificate validation. Once that clears the server saves the data in the database then it activates a node to send an SMS to a dispatcher. The dispatcher deploys the emergency services.

The Architecture follows a pyramid like hierarchy with sensors as the foundation, MQTT servers in the middle layer and a web server as the top of the hierarchy.

Here is a visualization of the architecture:



Acceptance tests

Fall Detection accuracy: this aspect cannot be tested exhaustively but it has a 100% success rate at detecting a fall with the current version of the prototype

Latency: in the case of a fall the dispatcher has to be notified in less than a minute. The slower recorded time for the dispatcher to get notified was 33.47 seconds.

Cloud server databases should automatically update the fallen users record: by the design of the system it already has this functionality

Easily deployable: It is extremely easy to deploy and maintain for a technician but not for a user. This of course was the intention. To deploy this system for any new user you add their credentials to the database, turn on the MQTT server at their location and specify their user id on it along with the wifi information, and lastly we would add the wifi and MQTT information to the sensor.

Design decisions

- The inclusion of local MQTT servers is to allows multiple users access you our service
- There are scripts on all admin pages that block users without the proper permissions to access that page for better security
- The client requests require the correct certificate in order to access the web server this is prevents sensitive data from being leaked
- SMS is more noticeable than an email and a phone could be modified to have an alarm to signify urgency

Code

Python:

- Simple user input that is formatted into an https request and sent to the server
- Or it is set up to loop forever and subscribe to an MQTT server to receive a message and then send out a preformatted http request

```python
import requests

id_ = input("Enter Id: ")
parameters = {"id":id_}

r = requests.delete("https://www.falldetectionresponse.ca:1880/delete", data = parameters, verify=False)
print(r.url)
```

```python
import paho.mqtt.client as mqtt
import time
import requests

def response(client, userdata, msg):
    print("Recived: " + msg.payload.decode())
    if msg.payload.decode() == "fall detected":
        print("Calling Emergency services")

        id_ = 2
        name = "john doe"
        address = "456 anywhere cresent"
        parameters = {"id":id_, "name":name, "address":address}
        r = requests.put("https://www.falldetectionresponse.ca:1880/fall_detected", data = parameters, verify=False)
        print(r)
        #put the PUT request here



client = mqtt.Client("Subscriber")
client.on_message = response
```

Code

Arduino:

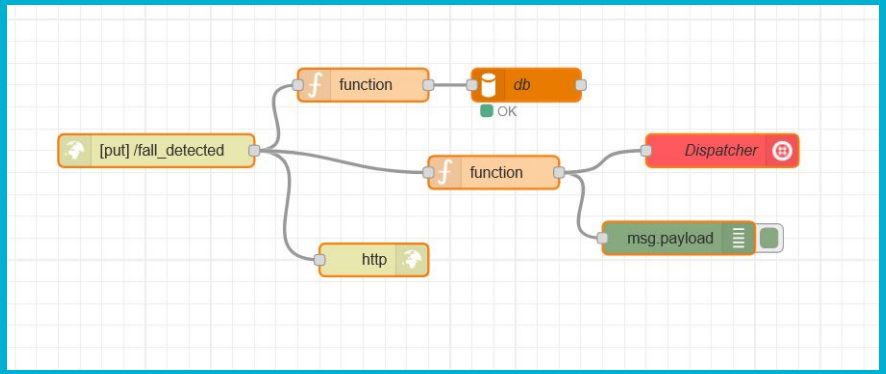
- Inspired by the Osoyoo kit tutorial 12 at https://osoyoo.com/2017/05/16/nodemcu-lesson-12-tilt-sensor-mqtt/ with several key changes made to the code (the physical build is the same the code is different)
- It incorporates MQTT and connects to the internet
- Other than that it loops forever and if it detects the conditions of a fall from the tilt sensor it publishes a message to the MQTT server

```
76  void loop() {
77    if (!client.connected()) {
78      reconnect();
79    }
80    client.loop();
81    long now = millis();
82    //send data every 6 second
83
84
85    if (now - lastMsg > 600) {
86       lastMsg = now;
87      int val=analogRead(Tiltsensor);
88       String msg=" ";
89       msg= msg+ val;
90      Serial.println(msg);
91      char message[58];
92
93      if(val<1000 ){
94        count++;
95      }
96      else if(val>1000){
97        count = 0;
98      }
99
100     if(count>60){
101        msg = "fall detected";
102        msg.toCharArray(message,58);
103        Serial.println(message);
104       client.publish("User_falls", message);
105       count = 0;
106     }
107
108     if(val>1000 && prev1<20 && prev2>1000){
109        msg = "fall detected";
110        msg.toCharArray(message,58);
111        Serial.println(message);
112       client.publish("User_falls", message);
113     }
114     else if(val<20 && prev1>1000 && prev2<20){
115        msg = "fall detected";
116       msg.toCharArray(message,58);
117       Serial.println(message);
118       client.publish("User_falls", message);
119     }
120
```

Code

Node red:

- Node red primarily uses nodes that connect to each other to form a flow
- However we still have to write javascript in order for the flow ro accomplish its goal
- The function nodes typical deconstruct the payload message and formulate a new message payload
- The request node(put in this case) creates an endpoint to receive https requests
- The db node receives queries from the function
- The Dispatcher receives a payload and sends that directly to the operator
- The debug node shows the payload on the server for debugging purposes
- The http response node is a page that shows the JSON format of a request, again for debug purposes




```
var fall = msg.payload.fall;
var address = msg.payload.address;
var id = parseInt(msg.payload.id);
var name = msg.payload.name;

var address2 = "'" + address + "'";
var fall2 = "'" + fall + "'";
var name2 = "'" + name + "'";


msg.topic = "INSERT INTO fallTable VALUES (" + id + ", " + name2 + ", " + address2+ ", " + fall2 +");"
return msg;
```

DEMO

QUESTIONS?