noble

PROTOTYPE OF AN INDOOR CONTACT TRACING SYSTEMS - IOT

USAI G. & TERRACCIANO V.

GOAL: ESTIMATE THE NUMBER OF PEOPLE INSIDE A BUILDING

We developed via NodeJS (v12.16.1) the **backend**:

1. Emitters (users)
2. Scanners (rooms, with a specific capacity each)
3. Central Server

All the data collected are stored in a MongoDB.

So, **Centralized** approach

THERE ARE IMPORTANT ASSUMPTIONS:

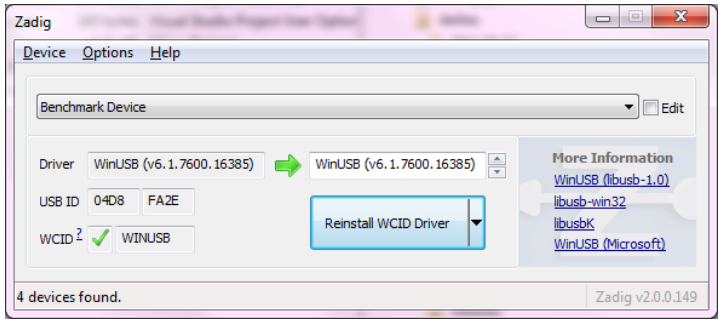
1. each room is equipped with a BLE **scanner** that can send data in real time to a **central server**;
2. the system must detect when the number of people in the room exceeds the allowed limit (**many solutions**);
3. each member of the building is equipped with a **BLE emitter** (e.g. a beacon on a bracelet, etc) with a unique identifier;
4. each user (staff and students) are registered into the system after having accepted all terms of use. (GDPR: ”data subject has given consent…”)

FRONTEND

- We developed the **FrontEnd** using Angular (Bar Chart - **ng2-charts**)

REAL WORLD BLE DEVICES BACKEND

- We integrate Noble.js BLE module inside our project in order to visualize the number of BLE devices near by our Compatible Bluetooth 4.0 USB adapter (Windows PC):
  - We notice some problems about scanning devices



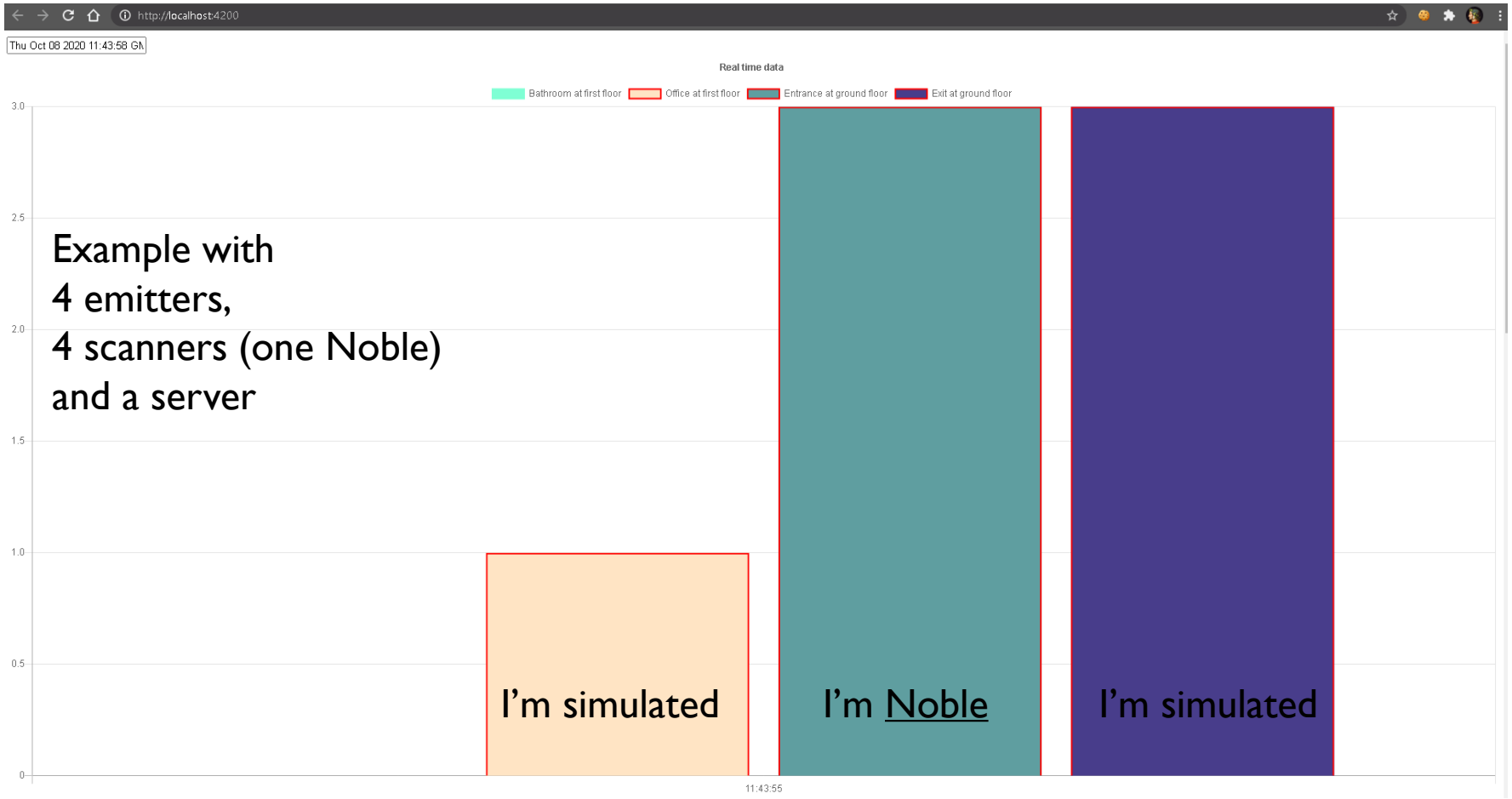
(we installed Noble using **Zadig** *USB driver installation)*

No compatible USB Bluetooth 4.0 device found! -> we have to ADD our USB adapter
Edit the file \bluetooth-hci-socket\lib\usb.js. Then I added usb.findByIds(0x8087, 0x0A2A) on line 66.
The parameters were fetched from zadig.exe (program) based on my PC.

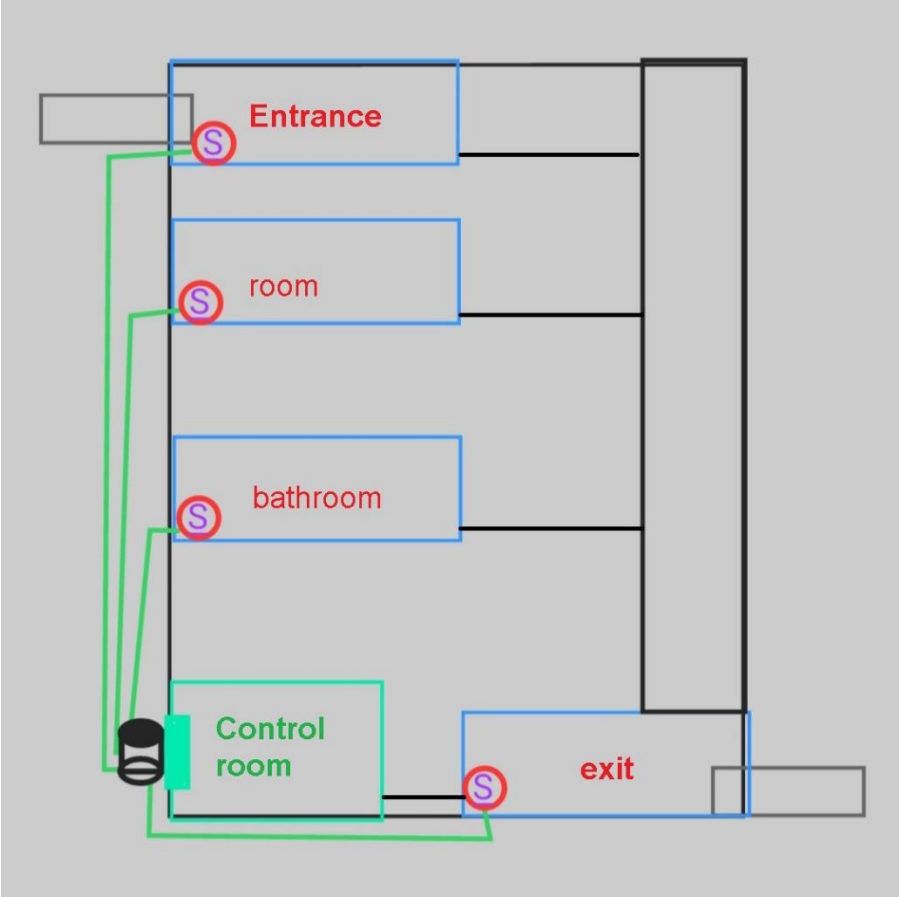
SIMULATED BLE DEVICES BACKEND

- In order to make a more realistic scenario, we developed NodeJS classes that simulate the presence of people inside the building:
  - class **BLEemitter**
  - class **BLEscanner**

DASHBOARD WITH ANONYMISED REAL-TIME DATA



DIBRIS BUILDING



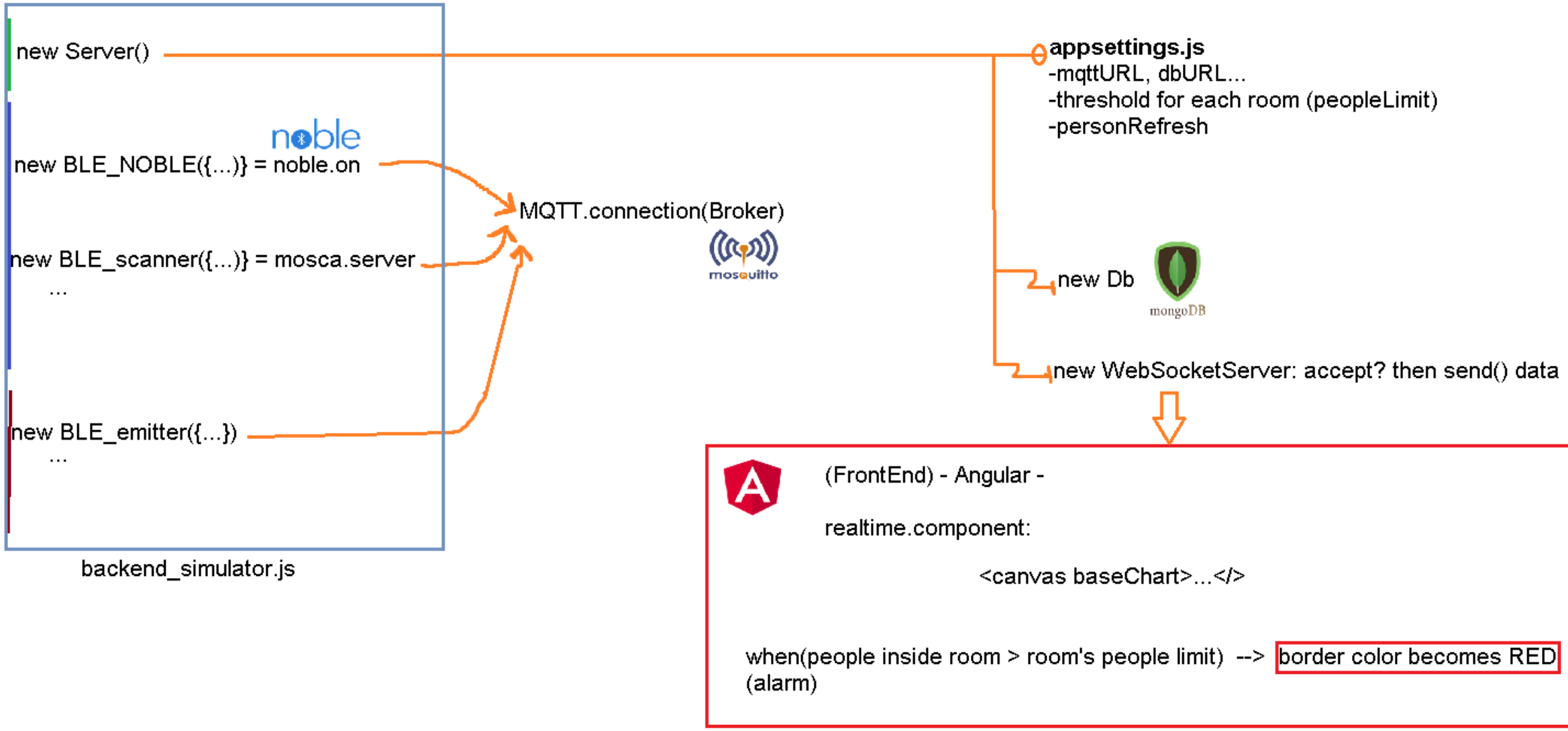
(A PIECE OF) BACKEND VIEW

```
"Amazfit Bip Watch" entered (RSSI -70) Thu Oct 08 2020 11:48:25 GMT+0200 (GMT+02:00)
groundFloor/entranceDISCOVER
"Mi Smart Band 5" exited (RSSI -65) Thu Oct 08 2020 11:50:05 GMT+0200 (GMT+02:00)
exit
"Amazfit Bip Watch" exited (RSSI -77) Thu Oct 08 2020 11:50:05 GMT+0200 (GMT+02:00)
exit
"Mi Smart Band 5" entered (RSSI -57) Thu Oct 08 2020 11:50:08 GMT+0200 (GMT+02:00)
groundFloor/entranceDISCOVER
"Amazfit Bip Watch" entered (RSSI -78) Thu Oct 08 2020 11:50:08 GMT+0200 (GMT+02:00)
groundFloor/entranceDISCOVER
"Amazfit Bip Watch" exited (RSSI -69) Thu Oct 08 2020 11:50:25 GMT+0200 (GMT+02:00)
exit
"Amazfit Bip Watch" entered (RSSI -78) Thu Oct 08 2020 11:50:25 GMT+0200 (GMT+02:00)
groundFloor/entranceDISCOVER
"Amazfit Bip Watch" exited (RSSI -71) Thu Oct 08 2020 11:51:27 GMT+0200 (GMT+02:00)
exit
"Amazfit Bip Watch" entered (RSSI -82) Thu Oct 08 2020 11:51:34 GMT+0200 (GMT+02:00)
groundFloor/entranceDISCOVER
"Amazfit Bip Watch" exited (RSSI -81) Thu Oct 08 2020 11:51:45 GMT+0200 (GMT+02:00)
exit
"Amazfit Bip Watch" entered (RSSI -69) Thu Oct 08 2020 11:51:55 GMT+0200 (GMT+02:00)
groundFloor/entranceDISCOVER
"Amazfit Bip Watch" exited (RSSI -69) Thu Oct 08 2020 11:52:05 GMT+0200 (GMT+02:00)
exit
"Amazfit Bip Watch" entered (RSSI -82) Thu Oct 08 2020 11:52:05 GMT+0200 (GMT+02:00)
groundFloor/entranceDISCOVER
"Amazfit Bip Watch" exited (RSSI -81) Thu Oct 08 2020 11:52:15 GMT+0200 (GMT+02:00)
exit
"Amazfit Bip Watch" entered (RSSI -81) Thu Oct 08 2020 11:52:17 GMT+0200 (GMT+02:00)
groundFloor/entranceDISCOVER
"Amazfit Bip Watch" exited (RSSI -81) Thu Oct 08 2020 11:52:22 GMT+0200 (GMT+02:00)
exit
```

Here we can understand important behaviour about Bluetooth Low Energy,
We have near our pc two devices (smartwatches) and they enter-exit continuosly from our Noble Scanner.

1. It's a battery save way but it can be a problem;
2. The **RSSI** it's incredible «almost» the same, but notice that both devices are stationary on the table. We tried it, but at the end we adopted the «room's people limit» solution.

SYSTEM OVERVIEW



MQTT – APPLICATION LAYER

MQTT is a publish-subscribe based "light weight" messaging protocol for use on top of the TCP/IP protocol.

1. Designed for connections with remote locations where a small code footprint is required and/or network bandwidth is limited
2. MQTT is designed to have low overhead compared to other TCP based application layer protocols

In MQTT there is a broker (server) that contains topics (rooms). Each client (emitter) can be a publisher that sends information to the broker at a specific topic or/and a subscriber (scanner) that receives automatic messages every time there is a new update

MQTT RELIABILITY

- MQTT ensures reliability by providing the option of three QoS levels:
  1. **Fire and forget**: A message is sent once and no acknowledgement is required
  2. Delivered at least once: A message is sent at least once and an acknowledgement is required
  3. Delivered exactly once: A four-way handshake mechanism is used to ensure the message is delivered exactly one time

  Remember: Topic (rooms) is case-sensitive!

  Server=test.mosquitto.org, Broker=mosquitto, port=1883, WebSocket=8080

WEBSOCKET

- Web sockets are defined as a two-way communication between the servers and the clients, which mean both the parties communicate and exchange data at the same time. The key points of Web Sockets are true concurrency and optimization of performance, resulting in more responsive and rich web applications.

- In Websocket a client initializes a handshake with a server to establish a Websocket session.
- It is designed for real-time communication, it is secure, it minimizes overhead can provide efficient messaging systems

SCANNERS IMPLEMENTATION

```
const scannerValues = [
  {
    clientId: 'scanner-entrance',
    topic: entranceTopic,
    topicDesc: "Entrance at ground floor",
    people: 0,
    port: 1880
  },
  {
    clientId: 'scanner-exit',
    topic: exitTopic,
    topicDesc: "Exit at ground floor",
    people: 0,
    port: 1881
  },
  {
    clientId: 'scanner-Room100',
    topic: firstFloorOfficeTopic,
    topicDesc: "Office at first floor",
    people: 0,
    port: 1882
  },
  {
    clientId: 'scanner-Bath100',
    topic: firstFloorBathTopic,
    topicDesc: "Bathroom at first floor",
    people: 0,
    port: 1883
  }
```

appsettings.js

```
new BLE_NOBLE({
    clientId: 'scanner-entrance',
    topic: settings.entranceTopic,
    topicDesc: "Entrance at ground floor",
    people: 0,
    port: 1880
});

new BLEscanner({
    clientId: 'scanner-exit',
    topic: settings.exitTopic,
    topicDesc: "Exit at ground floor",
    people: 0,
    port: 1881
});

new BLEscanner({
    clientId: 'scanner-Room100',
    topic: settings.firstFloorOfficeTopic,
    topicDesc: "Office at first floor",
    people: 0,
    port: 1882
});

new BLEscanner({
    clientId: 'scanner-Bath100',
    topic: settings.firstFloorBathTopic,
    topicDesc: "Bathroom at first floor",
    people: 0,
    port: 1883
});
```

backend_simulator.js

CLASSIC (EMITTERS) SCANNER

```
  startScanner(){
    var self = this;
    var scanner = new mosca.Server(this.clientSettings);
    var mqttClient = mqtt.connect(settings.mqttBrokerUrl, { clientId: this.clientId });
    mqttClient.publish(self.topic, JSON.stringify({ clientId: self.clientId, topic: self.topic,
      topicDesc: self.topicDesc, people: 0, date: Date.now(), port: self.clientSettings.port }));

    scanner.on('ready', function() {
      console.log(Date() + ' ' + self.clientId + ': ' + self.topic + ' is running...');
    });

    scanner.on('clientConnected', function() {
      mqttClient.publish(self.topic, JSON.stringify({ clientId: self.clientId, topic: self.topic,
        topicDesc: self.topicDesc, people: (++self.people), date: Date.now() }));
    });

    scanner.on('clientDisconnected', function() {
      mqttClient.publish(self.topic, JSON.stringify({ clientId: self.clientId, topic: self.topic,
        topicDesc: self.topicDesc, people: (self.people ? --self.people : 0), date: Date.now() }));
    });

  }
}
```

EMITTER IMPLEMENTATION

```
module.exports = class BLEemitter {

  min_range = 1881
  max_range = 1883

  constructor(emitterDevice){
    this.scannerPortToConnectTo = -1;
    this.scannerConnectedTo = null;
    this.emitterId = emitterDevice.clientId;
    setInterval(this.startEmitter, settings.personRefreshMs, this);
  }

  startEmitter(self){
    var auxScannerPort = Math.floor(Math.random() * (max_range - min_range) + min_range);
    if(auxScannerPort !== self.scannerPortToConnectTo){
      if(self.scannerConnectedTo){
        self.scannerConnectedTo.end();
      }

      self.scannerPortToConnectTo = auxScannerPort;

      this.clientSettings = {
        port: self.scannerPortToConnectTo,
        clientId: self.emitterId
      }
      // -- from settings
      self.scannerConnectedTo = mqtt.connect(settings.mqttUrl, this.clientSettings);
    }
  }
}
```

NOBLE (REAL WORLD) SCANNER

```
startScanner(){
  var self = this;
  var mqttClient = mqtt.connect(settings.mqttBrokerUrl, { clientId: this.clientId });

  mqttClient.publish(self.topic,
    JSON.stringify({ clientId: self.clientId, topic: self.topic,
      topicDesc: self.topicDesc, people: 0, date: Date.now(), port: self.clientSettings.port })
  );
  //__ - NOBLE - _____________________________________
  // var RSSI_THRESHOLD    = -90; //in order to deevelop the range-limit version system
  var EXIT_GRACE_PERIOD = 5000; // milliseconds
  var inRange = [];

  noble.on('discover', function(peripheral) {
    var id = peripheral.id;
    var entered = !inRange[id];

    if (entered) {
      inRange[id] = {
        peripheral: peripheral
      };
      console.log('"' + peripheral.advertisement.localName + '" entered (RSSI ' + peripheral.rssi + ') ' + new Date());
      mqttClient.publish(self.topic, JSON.stringify({ clientId: self.clientId, topic: self.topic,
        topicDesc: self.topicDesc, people: (++self.people), date: Date.now() }));
      console.log(self.topic + 'DISCOVER')
    }
    inRange[id].lastSeen = Date.now();
  });

  // -- update the peripherals inside the range
  setInterval(function() {
    for (var id in inRange) {
      if (inRange[id].lastSeen < (Date.now() - EXIT_GRACE_PERIOD)) {
        var peripheral = inRange[id].peripheral;

        console.log('"' + peripheral.advertisement.localName + '" exited (RSSI ' + peripheral.rssi + ') ' + new Date());
        delete inRange[id];
        mqttClient.publish(self.topic, JSON.stringify({ clientId: self.clientId, topic: self.topic,
          topicDesc: self.topicDesc, people: (self.people ? --self.people : 0), date: Date.now() }));
        console.log('exit')

      }
    }
```

UPGRADE

- https server with websockets on Node-js
  - https://stackoverflow.com/questions/46175397/https-server-with-websockets-on-node-js
  - https://stackoverflow.com/questions/7580508/getting-chrome-to-accept-self-signed-localhost-certificate

END

- Centralized or Decentralized? https://eprint.iacr.org/2020/531
- **https://www.npmjs.com/package/ng2-charts**
- **https://github.com/noble/noble**
- **https://nodejs.org/api/modules.html**
- **https://docs.mongodb.com/manual/mongo**
- **https://www.bluetooth.com/specifications/assigned-numbers/service-discovery/**