



IoT Final project

Francesco Stucci

Back-end

- **Emitter:** bluetooth device assigned to each employees
- **Scanner:** bluetooth scanner installed on each floor/room we need to monitor
- **Server:** server to handle the scanners data
 - **Database:** to store data collected from scanners

Protocol

All backend devices communicates among them via **MSQTT** protocol



Framework



Server, scanners and emitters are implemented by using the **NodeJs** framework

DBMS

All data are stored with **MongoDB**, a NoSQL DBMS.



Stored data

Emitters

```
{  
  clientId: int  
}
```

Data

```
{  
  clientId: int,  
  topis: string,  
  topicDesc: string,  
  people: int,  
  date: Date  
}
```

Emitter

```
module.exports = class BLEmitter {
  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() * (1885 - 1880) + 1880);
    if(auxScannerPort !== self.scannerPortToConnectTo){
      if(self.scannerConnectedTo){
        self.scannerConnectedTo.end();
      }

      self.scannerPortToConnectTo = auxScannerPort;
      console.log("Scanner port to connect to: " + self.scannerPortToConnectTo);
      this.clientSettings = {
        port: self.scannerPortToConnectTo,
        clientId: self.emitterId
      }
    }

    self.scannerConnectedTo = mqtt.connect(settings.msqttUrl, this.clientSettings);
  }
}
```

Scanner

```
module.exports = class BLEscanner{
  constructor(scannerDevice){
    this.people = 0;
    this.clientId = scannerDevice.clientId;
    this.topic = scannerDevice.topic;
    this.topicDesc = scannerDevice.topicDesc;
    this.clientSettings = {
      port: scannerDevice.port
    }
    this.startScanner();
  }

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

    scanner.on('ready', function() {
      console.log(Date() + ' ' + self.clientId + ': ' + self.topic + ' is running...');
    });
    scanner.on('clientDisconnected', function(client) {
      serverConnection.publish(self.topic, JSON.stringify({ clientId: self.clientId, topic: self.topic, topicDesc: self.topicDesc, people: (self.people ? --self.people : 0), date: Date.now(), port: self.clientId }));
    });
    scanner.on('clientConnected', function(client) {
      serverConnection.publish(self.topic, JSON.stringify({ clientId: self.clientId, topic: self.topic, topicDesc: self.topicDesc, people: (++self.people), date: Date.now(), port: self.clientId }));
    });
  }
}
```


Server

```
var self = this;
this.dbManager.dbInitilisedEmitter.on('initilised', function() {
  self.startServer();
});
}

startServer(){
  var self = this;
  this.server.on('ready', function() {
    console.log(Date() + ' Server is running...');
  });

  this.server.on('clientConnected', function(client) {
    console.log(Date() + ' ' + client.id + ' connected');
  });

  this.server.on('clientDisconnected', function(client) {
    console.log(Date() + ' ' + client.id + ' disconnected');
  });

  this.server.on('published', function(packet, client) {
    if(client && packet.topic){
      self.dbManager.insertData(JSON.parse(packet.payload));
    }
  });
}
```

Front-end

- **Client:** remote desktop pc which is supposed to monitor the building

Protocol



Client and server transfer data through
WebSocket protocol

Framework



The implementation of the frontend has been developed by using **Angular** framework

Server

```
this.webServer.listen(8080, function() {
  console.log((new Date()) + ' Web Server is listening on port 8080');
});

this.socketServer.on('request', function(request) {
  console.log((new Date()) + ' Connection accepted. ');

  setInterval(function() {
    self.dbManager.getData().then(function(promises) {
      Promise.all(promises).then(function(res) {
        connection.send(JSON.stringify(res.map(x => { return { clientId: x.clientId, people: x.people, topicDesc: x.topicDesc, date: x.date } })));
      });
    });
  }, settings.dataRefreshMs);

  connection.on('close', function() {
    console.log((new Date()) + ' Peer ' + connection.remoteAddress + ' disconnected. ');
  });
});
}
```

Client

