Redes Móveis e Sem Fios

2º Semestre 2017/2018

Relatório final
Sistema de vigilância IoT

Grupo 6

João Filipe Monteiro Rodrigues Nº 80895
Miguel Catela Saraiva Nº 81584

Índice

1 Introdução

O objetivo final deste projeto consiste em realizar um sistema de segurança que consegue detetar intrusões num certo espaço físico (através de sensores de infravermelhos e magnético), registá-las num servidor e interagir com uma aplicação de telemóvel de modo a ativar/desativar o sistema, notificar o utilizador de alguma entrada não autorizada, entre outras funcionalidades. O sistema pode ser dividido em três partes essenciais: o servidor, a aplicação e o microcontrolador. Cada uma delas é fundamental ao funcionamento do sistema e interagem entre si através de protocolos pré-definidos, os quais serão explicados mais à frente.

2 Componentes do sistema e funcionalidades

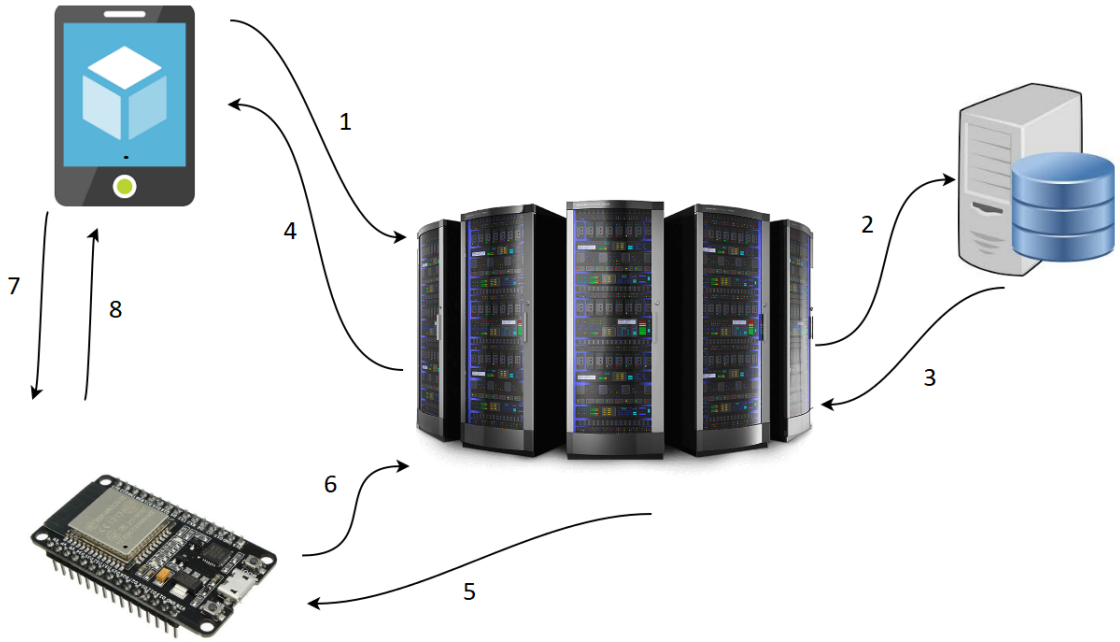


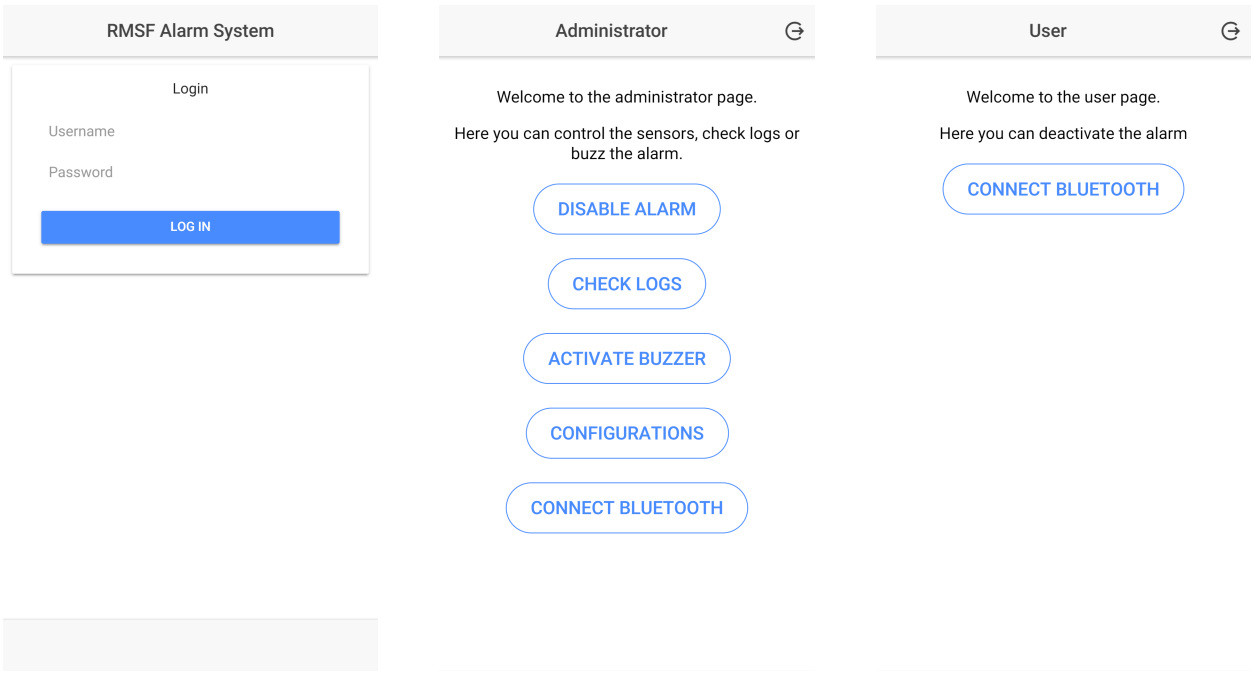
Figura 1: Esquema geral do sistema

Na figura 1 pode-se ver um esquema geral do funcionamento do sistema em que (1) e (4) representam as comunicações feitas entre a app e o servidor por HTTP, (2) e (3) representam os acessos do servidor à base de dados, (5) e (6) correspondem à comunicação via MQTT entre o servidor e o microcontrolador e por último (7) e (8) fazem a ligação por bluetooth e respetivo feedback entre a app e o microcontrolador.

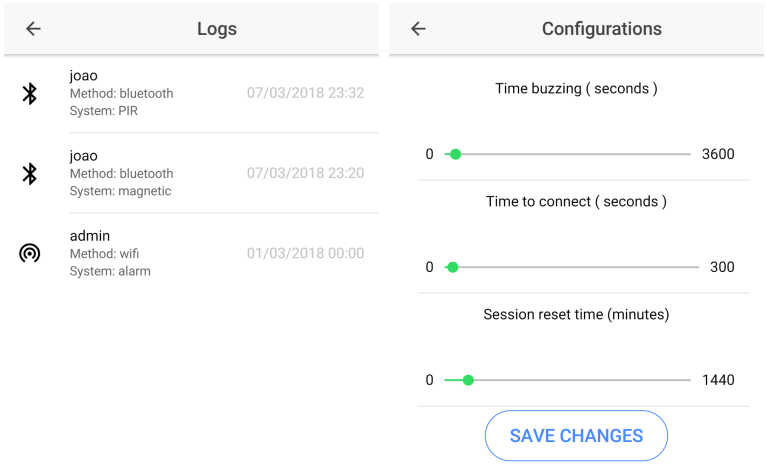
2.1 Aplicação

A aplicação foi criada utilizando o framework open-source Ionic. O Ionic permite criar aplicações móveis híbridas com acesso aos recursos nativos utilizando o mesmo código para Android e iOS. A escolha deste software deveu-se à linguagem de programação utilizada, JavaScript, às suas plataformas de teste e à versatilidade entre diferentes sistemas operativos.

A aplicação tem como página inicial um formulário de login (fig.2a), após o login a app é dividida em duas interfaces: interface de utilizador (fig.2c) e a de administrador (fig.2b). A interface de administrador tem as funcionalidades de: desativar alarme, ativar alarme, ativar o "buzzer", aceder aos "logs" (fig.2d), mudar as configurações (fig.2e) e conectar por bluetooth. A de utilizador pode apenas fazer a conexão por bluetooth para desativar o alarme temporariamente.



(a) Login (b) Administrador (c) Utilizador



(d) Logs (e) Configurações

Figura 2: Páginas da aplicação

2.2 Servidor

O servidor foi desenvolvido com Node.js e comunica com a aplicação e com o microcontrolador, processando a informação recebida, e quando necessário, guardando-a numa base de dados MySQL.

As funcionalidades do servidor passam por:
- Garantir o login do utilizador na aplicação;
- Guardar e enviar as configurações definidas na aplicação para o microcontrolador;
- Receber as informações enviadas pelo microcontrolador.

Na base de dados a que o servidor está ligado são guardados certas configurações do sistema, todas as contas dos utilizadores e a hash das suas passwords, e os registos gerados pelo microcontrolador quando há uma intrusão ou acesso autorizado ao espaço.

Para permitir a comunicação com a aplicação via HTTP utilizou-se o Express que é um framework web para o Node.js.

Para a comunicação com o microcontrolador utilizou-se o protocolo MQTT e para tal foi necessário instalar o servidor MQTT Mosquitto.

O servidor Node.js e MQTT estão alojados no DigitalOcean num VPS a correr Ubuntu. A base de dados MySQL é a que o Instituto Superior Técnico disponibiliza gratuitamente aos seus utilizadores.

2.3 Microcontrolador

O microcontrolador utilizado foi o ESP32 (versão ESP-WROOM-32) da Espressif. Neste caso vem soldado numa placa juntamente com um conversor Serial-USB e regulador de tensão, podendo ser montado numa breadboard.

O ESP32 é um circuito integrado de baixo custo que pode ser programado tal como um Arduino. Inclui um processador dual-core, um co-processador de baixa potência e tem como funcionalidades principais Wi-Fi, Bluetooth 4.0 e BLE integrados.

O microcontrolador é responsável por receber os sinais gerados pelos sensores infravermelhos e magnético, ativar o buzzer quando definido e estabelecer as comunicações via Wi-Fi com o servidor e via Bluetooth com a aplicação.

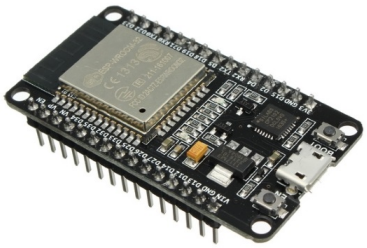


Figura 3: ESP32

Na figura 4 é possível observar os sensores magnético e infravermelhos ligados à placa, tal como o buzzer.

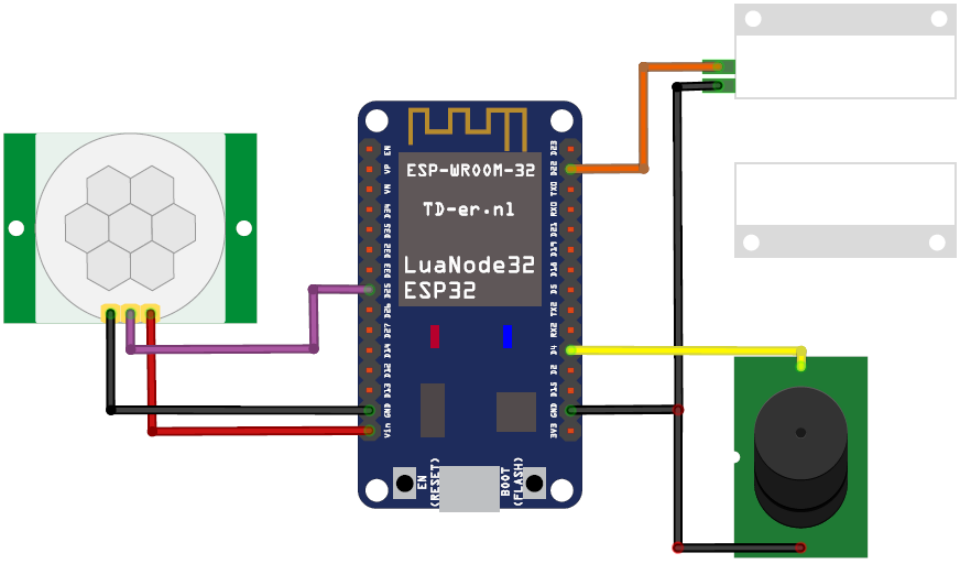


Figura 4: Circuito com os sensores e buzzer

Quando o sistema é ligado, é feito um pedido ao servidor para este enviar as configuração relacionadas com o estado do alarme e temporizadores. Se as configurações indicarem que o alarme deve estar ligado então o sistema fica no estado *"on"*, caso contrário fica no estado *"off"* até ser ativado novamente pela aplicação.

Existem duas situações em que a aplicação pode desativar o alarme via bluetooth. A primeira ocorre quando o sistema está no estado *"on"* e o utilizador conecta-se via bluetooth com credenciais válidas, neste caso o sistema passa para o estado *"reactivate"*. A segunda ocorre quando algum sensor é ativado e nesse caso o utilizador tem um intervalo de tempo para se ligar via bluetooth com credencias válidas, caso contrário o buzzer começa a tocar (estado *"buzzer on"*).

No estado *"reactivate"* existe um temporizador com o tempo definido pelo administrador ao fim do qual o alarme volta a ser ativado. Se nas configurações recebidas o valor do temporizador indicar 0, significa que o sistema transita para o estado *"off"* até ser ativado manualmente pelo administrador via Wi-Fi.

O tempo que o buzzer fica ligado também é controlado por um temporizador, que pode ser configurado. No estado *"on"* o administrador também pode ligar o buzzer pela aplicação.

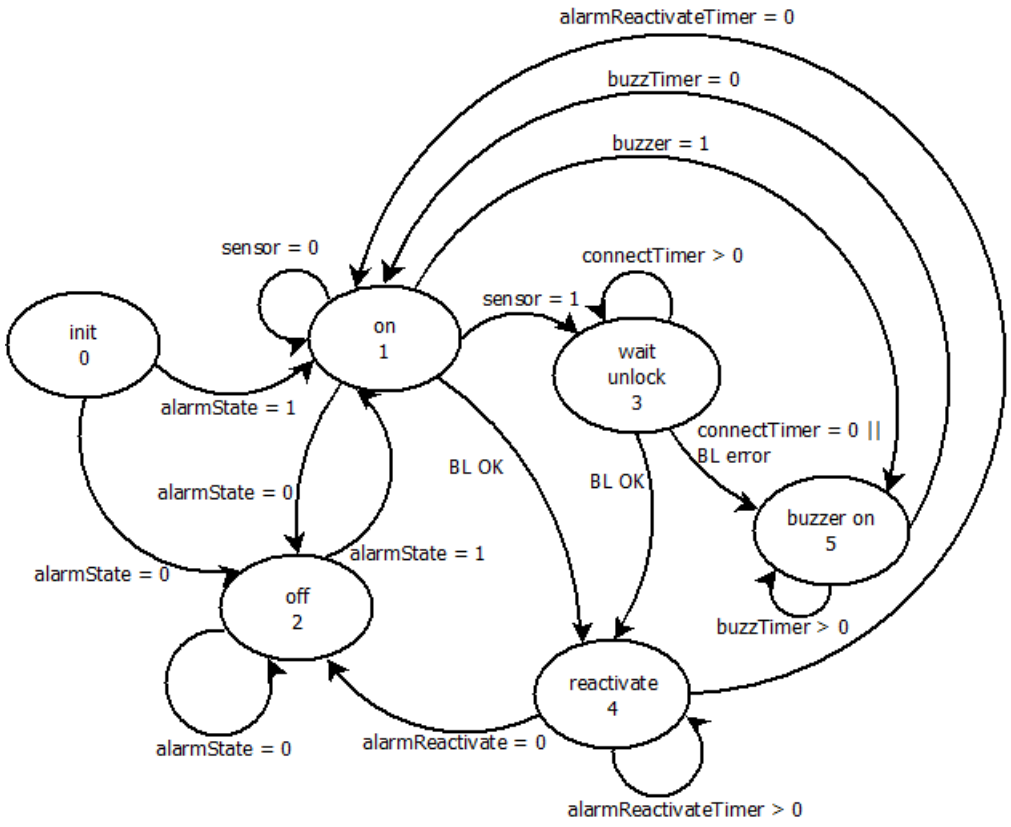


Figura 5: Máquina de estados

3 Interação entre componentes

3.1 Aplicação – Servidor

De modo a permitir a comunicação entre a aplicação e o servidor desenvolveu-se uma API baseada em pedidos HTTP POST. Foram criados routes no servidor Node.js através do framework Express, alguns deles protegidos, requerendo permissões de administrador.

Depois de um utilizador efetuar o login corretamente, o servidor responde com um token do tipo "JSON Web Tokens". Estes tokens são trocados no formato Base64 e são compostos por 3 partes (header.payload.signature).

Na parte do header indica-se o tipo de algoritmo utilizado para gerar a signature e o tipo do token.

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

No payload indica-se o nome de utilizador, se é ou não administrador, a data em que foi criado e a data em que irá expirar.

```
{
  "username": "admin",
  "admin": 1,
  "iat": 1520463269,
  "exp": 1520466869
}
```

Através da signature o servidor consegue garantir se o token é válido ou se foi alterado, uma vez que para gerar a signature é necessário utilizar uma chave secreta que só o servidor conhece, para além do header e do payload. Para isso basta gerar a signature com o header e payload recebidos e comparar com a signature que faz parte do token recebido.

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

• **Login:**
Quando um utilizador efetua um login com as credenciais corretas o servidor responde com um token que a aplicação vai guardar para realizar pedidos futuros enquanto o token não expirar ou o utilizador não fizer logout. Caso ocorra um erro no login, a reposta indica que ocorreu um erro.

Pedido Aplicação → Servidor

```
POST /users/login HTTP/1.1
Host: rmsf.hadrons.xyz
Content-Type: application/json

{"username": "example", "password": "example"}
```

Respostas Servidor → Aplicação

```
{
  "error": 0,
  "token": "token",
  "admin": 1
}

{
  "error": 1,
  "data": "Username and Password does not match"
}
```

• **Aceder aos logs:**
Se o administrador quiser aceder aos logs basta enviar o pedido para a route /admin/logs com um token válido. A resposta do servidor contém um array com todos os logs guardados na base de dados.

Pedido Aplicação → Servidor

```
POST /admin/logs HTTP/1.1
Host: rmsf.hadrons.xyz
Content-Type: application/json

{"token": "token"}
```

Resposta servidor → Aplicação

```
{
  "error": 0,
  "data": [
    {
      "username": "admin",
      "datetime": "2018-03-01T00:00:00.000Z",
      "method": "wifi",
      "system": "alarm",
      "identifier": 0
    }
  ]
}
```

• **Ativar buzzer:**
É possível ativar o buzzer durante o intervalo de tempo definido nas configurações através da aplicação.

Pedido Aplicação → Servidor

```
POST /admin/buzzer HTTP/1.1
Host: rmsf.hadrons.xyz
Content-Type: application/json

{"token": "token"}
```

• **Ativar ou desativar alarme:**
Se o administrador necessitar de ativar ou desativar o sistema de alarme pode fazê-lo remotamente através da aplicação via Wi-Fi.

Pedidos Aplicação → Servidor

```
POST /admin/enableAlarm HTTP/1.1
Host: rmsf.hadrons.xyz
Content-Type: application/json

{"token": "token"}

POST /admin/disableAlarm HTTP/1.1
Host: rmsf.hadrons.xyz
Content-Type: application/json

{"token": "token"}
```

• **Enviar configurações:**
As configurações são compostas por 3 parâmetros:
- buzzTime: tempo em segundos em que o buzzer fica ativo;
- connectTime: tempo em segundos que um utilizador tem para desligar via bluetooth o alarme após ativar um dos sensores;
- alarmReactivate: tempo em segundos que demora até o sistema de alarme ficar ativo novamente, se for 0 o alarme não volta a ficar ativo automaticamente.

Pedido Aplicação → Servidor

```
POST /admin/configs HTTP/1.1
Host: rmsf.hadrons.xyz
Content-Type: application/json

{"token": "token", "buzzTime": 30, "connectTime": 30, "alarmReactivate": 120}
```

• **Receber as configurações guardadas:**
Este pedido serve para a aplicação receber as configuração que estão guardadas no servidor.

Pedido Aplicação → Servidor

```
POST /admin/savedInfo HTTP/1.1
Host: rmsf.hadrons.xyz
Content-Type: application/json

{"token": "token"}
```

Resposta Servidor → Aplicação

```
{
"error": 0,
"buzzTime": 30,
"connectTime": 40,
"alarmReactivate": 140
}
```

Quando um pedido é feito com um token inválido o servidor responde com uma mensagem de erro.

```
{
"error": 1,
"data": "Token is invalid"
}
```

Se um utilizador sem privilégios de administrador tentar aceder a um recurso que necessita desse privilégio, o servidor também envia uma mensagem de erro.

```
{
"error": 1,
"data": "Forbidden"
}
```

3.2 Servidor – Microcontrolador

Para fazer qualquer pedido entre o servidor e o microcontrolador utiliza-se o protocolo MQTT. O protocolo MQTT utiliza um mecanismo de *publish-subscribe*, quem subscreve um tópico fica à escuta de quem publica nesse tópico. O servidor MQTT (MQTT broker) faz a distribuição das mensagens.

Para isto, primeiro é necessário ligar o servidor Node.js ao servidor MQTT que se encontra na mesma máquina previamente referida através do comando

```
var client = mqtt.connect('mqtt://rmsf.hadrons.xyz').
```

Depois, para o servidor receber mensagens do microcontrolador subscreve diferentes tópicos:

```
client.subscribe('bluetooth/login');
client.subscribe('startup');
client.subscribe('sensor');
```

Para enviar mensagens para o microcontrolador publica nos seguintes tópicos:

```
client.publish('buzzer', 'on');
client.publish('state', 1/0);
client.publish('configs', buzzTime + ':' + connectTime + ':' + alarmReactivate);
client.publish("bluetooth/confirm", 1/0);
```

O microcontrolador subscreve os tópicos em que o servidor publica:

```
client.subscribe("buzzer");
client.subscribe("configs");
client.subscribe("state");
client.subscribe("bluetooth/confirm");
```

Por fim, as mensagens que o microcontrolador envia para o servidor são as seguintes:

```
client.publish("bluetooth/login", "username:password");
client.publish("sensor", "magnetic"/"infrared");
client.publish("startup", "");
```

3.3 Aplicação – Microcontrolador

A comunicação entre a aplicação e o microcontrolador é feita estritamente por Bluetooth Low Energy (BLE). Os dispositivos BLE são detetados através de um mecanismo de broadcast de pacotes que os anuncia periodicamente, com um delay aleatório de até 10 ms de modo a evitar colisões.

Nas ligações BLE existe um servidor e vários clientes. Um servidor emite uma ou mais características, que podem ter várias propriedades tais como escrita ou leitura. A um conjunto de características chama-se serviço. Tanto as características como os serviços são identificados por UUIDs (Universally Unique Identifier).

Neste projeto o microcontrolador é o servidor e o telemóvel com a aplicação é o cliente. Na aplicação, quando é ativado o botão de conexão por bluetooth, começa-se por fazer uma procura do dispositivo com o nome correspondente ao microcontrolador, neste caso, *"MyESP32"*. Assim que esse dispositivo é encontrado estabelece-se uma conexão e a aplicação escreve para a característica emitida pelo microcontrolador o nome de utilizador e a password: *"username:password"*.

O microcontrolador emite na sua característica o estado do alarme, *"Locked"* ou *"Unlocked"*. Após receber os dados enviados pela aplicação, estes são reencaminhados para o servidor via Wi-Fi. Se a resposta do servidor indicar que os dados são válidos, então a característica emitida é alterada para *"Unlocked"*.

4 Bibliografia

https://ionicframework.com/docs/
https://jwt.io/introduction/
https://github.com/espressif/arduino-esp32
https://techtutorialsx.com/2017/04/24/esp32-publishing-messages-to-mqtt-topic/