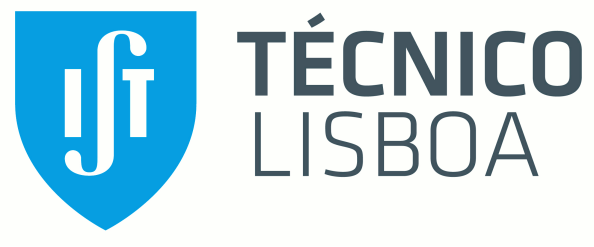

Redes Móveis Sem Fios

Deteção de Incêndios

Grupo N.º 14

Autores:

Leandro Almeida 84112
Rúben Gomes 84180

Professor:

António Grilo

21 de Maio de 2020

Conteúdo

Resumo

O trabalho a ser reportado no seguinte relatório tem em vista o desenvolvimento de um sistema de `IOT` com o objetivo de detenção de incêndios em edifícios, através de temperatura, concentração de gases e humidade do local onde instalado o dispositivo. No caso de deteção de incêndio, o sistema alerta os utilizadores (para a sua aplicação Android) associados ao dispositivo e deve tentar apagar o incêndio através de uma bomba de água. Existe um servidor na `Cloud` com um sistema de base de dados para armazenar a informação de todos os utilizadores registados que posssuem um ou mais dispositivos, assim como informações do sistema (temperatura, humidade, incêndios ocorridos, ..) e de administradores. A comunicação estabelecida entre os sensore necessários com o seridor é realizada através de *SigFox*. O servidor recebe dados do *SigFox* armazenando na base de dados, e responde a pedidos `GET` e `POST` por parte dos utilizadores (pela aplicação), conforme o armazenado na base de dados. Também existe a situação em que o servidor envia informação para o *SigFox* de acordo com o que foi enviado por parte do administrador.

O projeto foi pensado de forma a que haja escalabilidade, isto é, que cada utilizador tenha a possibilidade de possuír e ter acesso na mesma aplicação a vários dispositivos de forma independente, e que o servidor consiga gerir e armazenar informação de diferentes dispositivos de maneira coerente para diferentes utilizadores.

Palavras-chave: *akeru*, *sigfox*, `Cloud`, *uplink*, *downlink*, *Firebase*, `Django`, *proxy*

1 State of the Art

O sistema desenvolvido no âmbito deste projeto, apresenta uma estrutura semelhante ao do projeto estudado em [1] e em [2].

O sistema de deteção e extinsão de incêndios num espaço protegido, estudado em [1], é composto por um detetor de incêndio por cada porção localizada de um espaço protegido e por uma unidade de controlo central, responsável pela receção de dados dos sensores e armazennamento dos dados, bem como de dados previamente medidos indicadores de uma fase inicial de incêndio. É composto também por um sistema de comunicação entre cada detetor de incêndio e a unidade de controlo central e por um sistema controlado de extinsão de fogo, que é ativado quando há uma certa correlação entre os dados medidos e os dados indicadores de uma fase inicial de incêndio.

A nossa unidade de controlo central é um servidor desenvolvido em `Django`, e o sistema de comunicação utilizado entre um detetor de incêndio e esta unidade é a tecnologia `SigFox`.

Cada dispositivo detetor de incêndios do sistema estudado em [1], apresenta pelo menos o seguinte conjunto de sensores: sensor de monóxido de carbono, sensor de temperatura, sensor de humidade e detetor sensível a radioatividade proveniente de chamas, composto por um sensor de radiação infra-vermelha. O sistema implementado no âmbito deste projeto é composto por sensores de gases (que pode ser utilizado para medir monóxido de carbono) e um sensor de humidade e temperatura. Além do atuador responsável pela extinsão do incêndio, tal como no sistema estudado, o nosso sistema inclui um alarme que é ativado na ocorrência de incêndio.

O nosso sistema oferece ainda uma funcionalidade não disponível no sistema estudado em [1]: uma aplicação `Android`, que permite a um utilizador registar um dispositivo

e receber informações e alertas do mesmo e que permite ao administrador monitorizar os diversos dispositivos e utilizadores registados.

Também em [2] é abordado um projeto idêntico em que o foco são *smart cities*. Neste projeto, a comunicação realizada entre os sensores e o servidor é feita através de `Wi-fi`, ao passo que no nosso projeto é realizada através de *SigFox*, que tem vantagens, como o gasto de energia mais reduzido.

2 Arquitetura

A arquitetura do sistema encontra-se representada na figura 1.

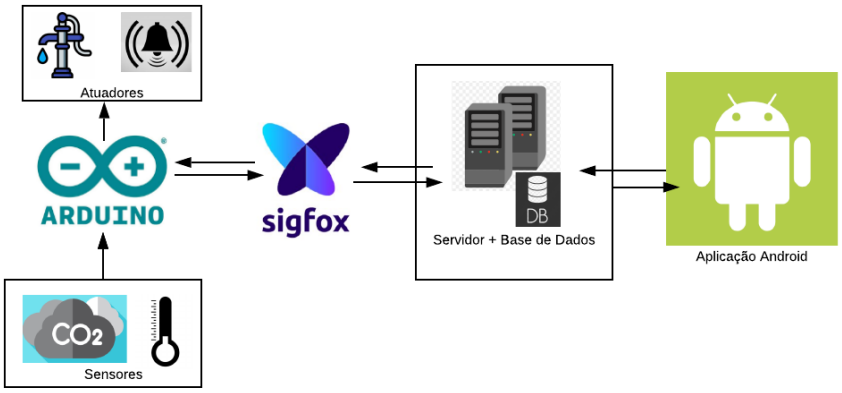

Figura 1: Arquitetura do Sistema

Como se pode observar, utiliza-se um microcontrolador *Akeru* com módulo *SigFox*. O *Akeru* conecta-se diretamente aos atuadores e sensores de temperatura, de gás para ler informaação destes sensores e atuar nos atuadores de alarme e à bomba de água. A transmissão dos dados lidos dos sensores para o servidor (através de um protocolo por nós definido e abordado em baixo) é realizada através do *SigFox*. Assim, o *Akeru* transmite para o *SigFox* que reencaminha para o servidor que se encontra hospedado na `Cloud` da Google. Por fim, estes dados podem ser acedidos por uma aplicação móvel para Android.

Um dos maiores desafios do projeto foi realizar um sistema facilmente escalável que tivesse a capacidade de ter vários utilizadores e vários dispositivos (tanto para o mesmo utilizador como para diferentes utilizadores) funcionando todo o sistema de forma coerente e tendo cada utilizador apenas acesso à informação referente aos dispositivos que possui.

Além da opção de utilizador, a aplicação disponibiliza a opção administrador, que é visto como o 'vendedor' do produto, que tem informação de todos os dispositivos existentes, todos os utilizadores, todos os incêndios. É o administrador que, ao vender o produto, adiciona um novo produto/dispositivo, ao qual é associado um *Token* para que o utilizador, na sua aplicação, consiga associar o dispositivo que comprou à sua conta, tendo acesso aos dados desse dispositivo, através do uso do *Token*.

3 Hardware

3.1 Esquema de ligações

Na figura 2 encontra-se o esquema de ligações do circuito implementado, realizado no *software* `Fritzing`. Como se pode verificar, o esquema divide-se em 4 grupos principais: microcontrolador, sensores e atuadores e circuito que impõe a corrente necessária ao funcionamento da bomba de água. É importante realçar que o microcontrolador e o sensor de gás representados no esquema não são os utilizados, contudo semelhantes, devido à sua ausência no *software* utilizado. Representam-se as ligações ao GND e ao VCC (5V) com fios pretos e vermelhos, respetivamente.

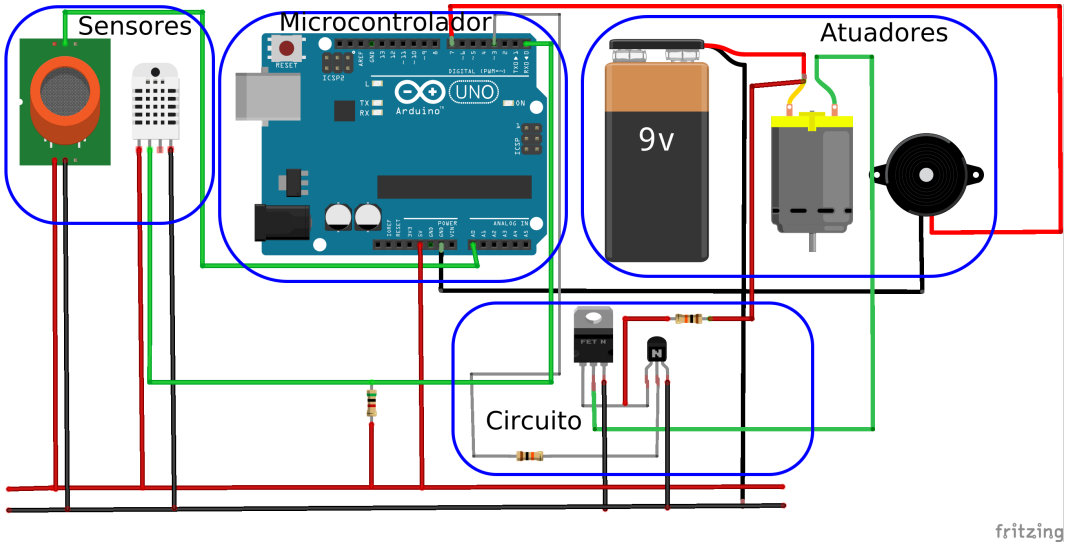

Figura 2: Esquema de ligações do *Hardware* utilizado

Os quatro grupos serão abordados nas secções seguintes.

3.2 Arduino *Akeru*

O microcontrolador utilizado é um *Akeru* β 3.3, que nos permite desenvolver o projeto com tecnologia *IoT SigFox* e linguagem *Arduino*. Esta tecnologia é *Low-Power*, o que é uma característica essencial na maioria das aplicações *IoT*. O microcontrolador é responsável pela monitorização dos valores recebidos pelos sensores e pela ativação dos atuadores. Além disso, envia os dados recolhidos periodicamente para o servidor.

3.3 Sensores

3.3.1 Sensor de temperatura e Humidade - DHT11

Para a monitorização dos valores de temperatura e de humidade relativa, utilizou-se o sensor DHT11, que se encontra representado na figura 3.

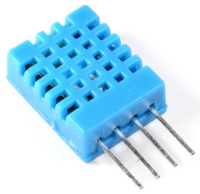

Figura 3: Sensor DHT11

Este sensor é composto por quatros pinos, no entanto, apenas três são utilizados (GND, VCC, DATA). O pino DATA é ligado ao arduino num pino digital. Este sensor necessita de uma alimentação entre 3.3 V a 5 V (DC) e apresenta uma potência de consumo reduzida. Tem um alcance de 20 a 90% no caso da leitura da humidade relativa e um alcance de 0 a 50 °C no caso da leitura da temperatura. É importante notar a presença de uma resistência de *pull-up* ($R = 5k$), que se pode verificar no esquema de ligações, entre VCC e o pino DATA que é recomendada para um cabo de conexão inferior a 20 metros. A comunicação e sincronização entre o sensor e o arduíno é realizada de acordo com o formato de dados *single-bus*, sendo que uma transmissão de dados completa envia 40 bits e leva cerca de 4 ms (informação retirada de [3]). Para realizar a comunicação com o DHT11, foi utilizada uma API que nos permite declarar uma instância do tipo `DHT`, que facilita a leitura dos valores recolhidos por este sensor. Ao declarar a instância `dht11` do tipo `DHT`, associada ao pino `DHT_PIN`, obtém-se a temperatura e a humidade do sensor através da invocação dos seguintes métodos: `DHT.read11(DHT_PIN)`, `DHT.temperature`, `DHT.humidity`. Ou seja, invoca-se o método `read11()` e, de seguida, pode-se aceder aos atributos *temperature* e *humidity* da instância utilizada. Outro aspeto a referir é que o período de amostragem do sensor de temperatura toma o valor mínimo de 1 segundo, pelo que a leitura dos sensores deve ser efetuada com um intervalo de tempo superior a este valor.

3.3.2 Detetor de gás - MQ -135

Para monitorizar os valores de gases presentes no ar, utilizou-se o sensor MQ-135, que se encontra representado na figura 4.

Figura 4: Sensor MQ135

Este sensor é composto por 4 pinos: um é ligado ao VCC, outro ao GND, outro permite fazer a leitura analógica da concentração de gases (valor entre 0 - baixas concentrações - e 1023 - altas concentrações) e o último permite fazer uma leitura digital da concentração de gases (0 ou 1 se a concentração estiver abaixou ou acima de um dado *threshold*, respetivamente, configurado através de um potenciómetro). Este sensor pode ser calibrado, de forma a fazer a leitura de gases, tais como dióxido/monóxido

de carbono, por exemplo (informação retirada de [4]). No *datasheet* é indicado que o sensor tem um tempo de pré-aquecimento de 24 horas, pelo que se torna difícil a calibração deste sensor, bem como a fiabilidade dos valores lidos numa demonstração. Para obter o valor do sensor utilizou-se o pino analógico, através da invocação da função `analogRead(pinMQ)`, em que `pinMQ` corresponde ao pino analógico utilizado para ler o valor do sensor.

3.4 Atuadores

Neste projeto, considerou-se necessária a utilização de uma bomba de água, que é ativada assim que o incêndio é detetado, com o objetivo de extinguir o incêndio rapidamente e a utilização de um alarme (*buzzer*), que é ativado no mesmo instante que a bomba de água e que tem o objetivo de alertar as pessoas que estão nas imediações do local de incêndio. Para dar liberdade ao utilizador, implementou-se a possibilidade de este desativar o alarme através da sua aplicação *Android*.

3.4.1 Bomba de água

A bomba de água utilizada encontra-se na figura 5

Figura 5: Bomba de água

Este atuador tem as seguintes tem uma tensão de funcionamento entre 4 V a 12 V, sendo que a sua tensão ótima é de 9 V. A corrente de funcionamento é de 0.8 A (informação retirada de [5]).

Uma dificuldade encontrada durante o planeamento do circuito foi o facto de o arduino não conseguir fornecer a corrente necessária à bomba de água, pelo que se teve de recorrer a um circuito auxiliar, que se encontra na figura 6:

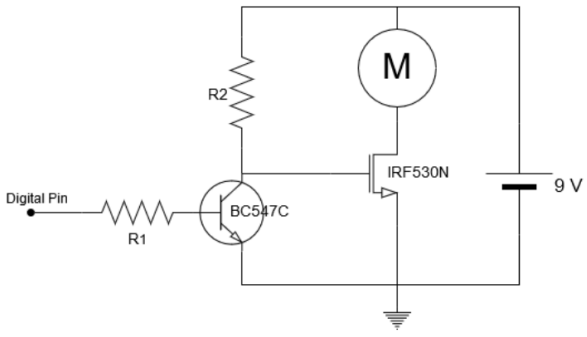

Figura 6: Circuito introduzido para obter a corrente necessária na bomba de água

Foram utilizadas duas resistências R_1 e R_2 de 10 k, que limitam a corrente do transístor BC547C. Com este circuito, a ativação da bomba é feita colocando o pino a

0, o que não é muito conveniente, uma vez que pode ativar a bomba quando se faz o *reset* do *Akeru*, ou quando este está a ser programado. Uma forma de contornar este problema seria utilizar um interruptor independente do *Akeru* que inibia a ativação da bomba sempre que o interruptor estivesse na posição OFF.

3.4.2 Alarme

Na figura 7 encontra-se um exemplo semelhante ao mini *buzzer* utilizado.

Figura 7: Mini buzzer

Ao contrário do caso anterior, o arduino tem a capacidade de fornecer tensão e corrente necessária para o correto funcionamento do alarme, pelo que se torna fácil construir o circuito necessário à sua ativação. Quando os *thresholds* definidos de temperatura, concentração de gases e humidade é ultrapassada, ativa-se o alarme através da função `tone(pin, frequency)`, que gera uma onda quadrada com a frequência especificada, com 50% de *duty cycle*, no pino indicado. Para a sua desativação, assim que os valores voltam a estar abaixo dos *thresholds* definidos, invoca-se a função *noTone(pin)*, que pára a geração da onda quadrada.

4 *SigFox*

A comunicação através de o protocolo de comunicação *SigFox* facilita a comunicação entre o arduino e o servidor, via rádio. A tecnologia LPWAN *SigFox* opera numa banda de rádio publicamente disponível. Na europa usa 868 MHz, conseguindo ter um alcance de 13 km. A tecnologia utilizada é *Ultra Narrow Band*, usando modulação BPSK, alcançando um *bitrate* de 100 bps.

SigFox é uma grande opção para considerar pois é fácil de implementar e os dispositivos são baratos, oferecendo uma grande duração de bateria. Os utilizadores têm de pagar uma licensa para o seu uso e depois não têm de se preocupar em no desenvolvimento e manutenção das suas estações de base.

Refere-se que em média ao se usar o *SigFox*, o tempo de vida das baterias é de 10 anos de vida e que o custo do dispositivo ronda os $5.

Como evidenciado em [6], a *SigFox* está a implantar em todo o país um novo tipo de rede sem fio, tornando mais barato e prático de ligar objetos a requisitos de baixa largura de banda - como o caso de deteção de incêndios - à internet. *SigFox* é uma rede global que permite que milhões de dispositivos se conectem à Internet de maneira direta, consumindo o mínimo de energia possível. Além desta grande caraterística, tem um custo bastante reduzido, e grande escalabilidade global. A simplicidade é também uma caraterística que esta rede possui : não é necessário qualquer configuração, o dispositivo fica logo apto a correr.

Analisou-se também a cobertura mundial da *SigFox* em [7], e com mais detalhe em Portugal. Como se pode observar, Portugal já tem uma grande cobertura de *SigFox*, algo também notado por nós na realização do trabalho, pois não tivemos qualquer problema nos locais utilizados a testar o projeto.

Para estudar se este tipo de comunicação rádio é adequada para o problema em questão, fez-se uma pesquisa por diversos sites.

Como especificado em [8], a colocação de sensores autónomos, capazes de alertar de imediato a eclosão de um foco de incêndio, mesmo em áreas remotas ou a meio da noite, só é possível através da utilização de uma rede de *SigFox* (em Portugal esta rede tem por nome *NarrowNet*). De facto, os sensores não necessitam de uma fonte permanente de energia, que não está disponível na maioria das florestas, para estarem operacionais durante vários anos. A deteção e informação do rumo e direção que tomam os incêndios também ajudarão na frente de combate, e no evitar que bombeiros fiquem encurralados.

Os dispositivos encontrados em [9], [10], [11] também têm como função a deteção de incêndios e fazem uso de *SigFox*.

Como especificado nestes dispositivos e já referido, o uso de *SigFox* permite um baixo consumo de energia. Os três produtos contém uma bateria que se estima durar cerca de 5 anos.

O produto em [9] tem um *range* de temperatura de -10ºC a 50ºC (o nosso tem um *range* entre 0ºC e 50ºC). Este produto apresenta algumas funcionalidades avançadas, como por exemplo: ativação de alarme no caso de deteção de incêndio ou danificação, indicador de bateria fraca, botão de teste para teste regular do alarme e envio de e-mail e SMS para contactos fornecidos. Uma das funcionalidades a melhorar na segunda entrega é adicionar a funcionalidade de envio de e-mail.

O produto em [10] tem um LED que indica se o alarme está a receber potência. As mensagens de *Uplink* enviadas ao *SigFox* incluem os valores de alarme de fumo, alarme de calor, alarme de falha, bateria fraca, capacidade de bateria, etc. Tirou-se ideias deste produto para se estabelecer o protocolo de comunicação de *Uplink* encontrado em baixo.

Já o produto em [11], que é um detetor de fumo totalmente autónomo que usa a rede *SigFox* para comunicar a monitorização de dia-a-dia e alertas em tempo real, consegue avisar através de chamada telefónica, SMS e e-mail. Tem a particularidade de emparelhar dois detetores, que agem como repetidores, assegurando que os dados são transmitidos para a rede *SigFox*, para maximizar a segurança.

Também em caso de incêndio, a internet deixa de funcionar rapidamente, pelo que comunicação por *wi-fi*, além de gastar muito mais energia, não é adequada ao caso em estudo. Com *wi-fi* ter-se-ia também de obrigar o utilizador a ter internet, algo não desejável, pois este pode querer instalar o aparelho por exemplo numa casa de férias sem internet.

Tentámos contactar elementos da empresa *SigFox*, assim como empresas que vendem produtos de deteção de incêndio, no entanto não obtivemos quaisquer respostas o que nos levou a pesquisar mais sobre projetos com o mesmo objetivo ou com características semelhantes.

4.1 Implementação

A rede de *SigFox* é bi-direcional, ou seja envia-se dados no sentido dispositivo-cloud e cloud-dispositivo. Para a realização da comunicação e troca de dados, foi utilizada a biblioteca `Akeru.h` no código de arduino, com funções de escrita para o *SigFox* e de

leitura. O servidor da *SigFox* ao receber dados vindos do arduíno, reencaminha para os *endpoints* do servidor, que são definidos nos *Callbacks* do *SigFox*. Na figura 8 e 9 ilustra-se a configuração executada no servidor da *SigFox* para lidar com a ação de *Uplink* e *Downlink*, respetivamente. No caso de *Downlink* reencaminha para o arduino os dados enviados pelo servidor. Refere-se que é necessário especificar o identificador do dispositivo pois o servidor está preparado para escalar para muitos dispositivos, e mesmo o servidor da *SigFox* necessita de receber uma mensagem neste formato.

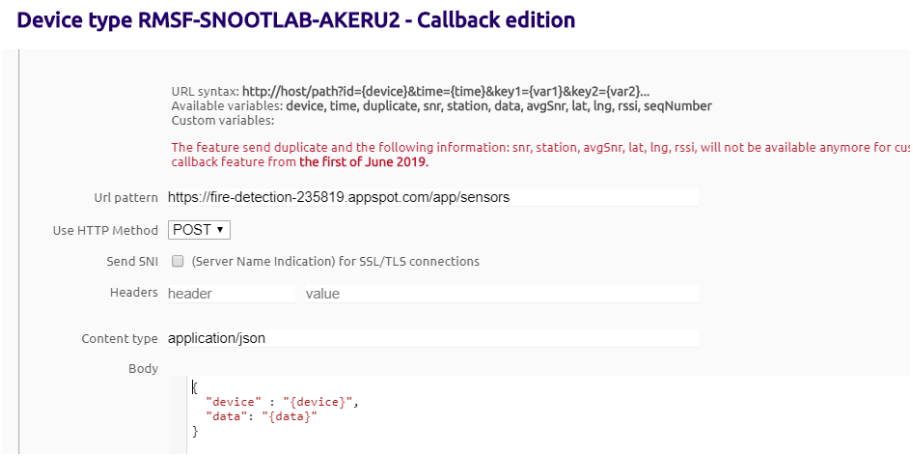

Figura 8: Configuração do *Callback* de *Uplink* no servidor da *SigFox*

O servidor da *SigFox* ao receber um pedido de *Uplink* envia os dados para o *endpoint* especificado na imagem, no qual o servidor da `Google` descodifica a informação e armazena na base de dados. Em caso de ocorrência de incêndio, alerta logo os utilizadores associados a este dispositivo.

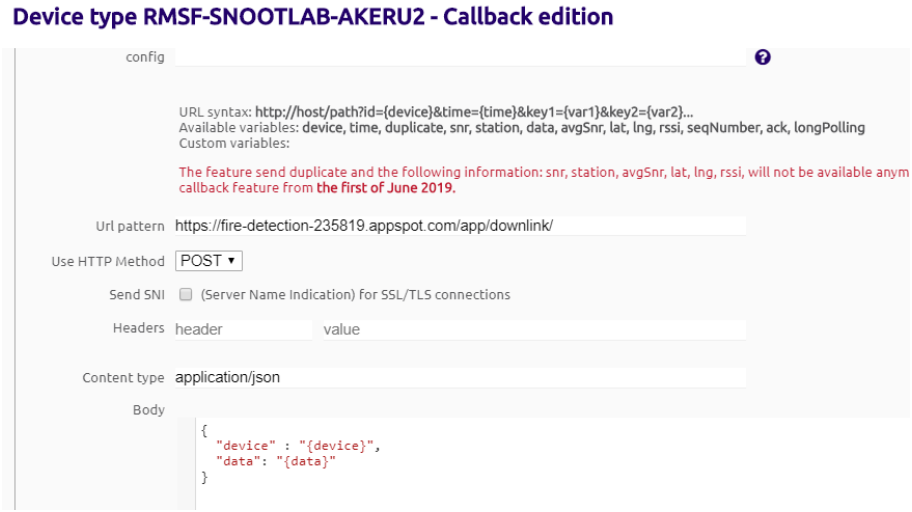

Figura 9: Configuração do *Callback* de *Downlink* no servidor da *SigFox*

No lado do servidor da `Google`, ao receber um *request* neste *endpoint*, reencaminha para o servidor da *SigFox* os dados definidos pelo protocolo de comunicação de *Downlink* (secção em baixo).

4.2 Protocolo de Comunicação

Os dados enviados em *Uplink* e *Downlink* seguem um protocolo específico, no qual se tem em atenção os formatos utilizados no *SigFox*. Como específicado em [12], o protocolo de *Sigfox* é desenhado para ser extremamente eficiente e permitir que dispositivos a durarem alguns anos com a mesma bateria. Assim sendo, existe uma limitação das mensagens enviadas.

4.2.1 *Uplink*

No *Uplink*, o protocolo permite enviar mensagens com um *payload* entre 0-12 bytes, e o cabeçaho (que contém o identificador do dispositivo por exemplo). De [12], sabe-se que só é permitido o envio de 140 mensagens de *Uplink* por dia, o que dá um número de 6 mensagens por hora - 1 mensagem a cada 10 minutos. No entanto, na nossa implementação colocou-se a enviar a mensagem do estado atual de 20 em 20 minutos caso não esteja a ocorrer nenhum incêndio. Caso esteja a haver incêndio envia-se de 1 em 1 minuto. Assim, para o caso de ocorrerrência de algum incêndio, consegue-se enviar com uma maior frequência que as 1 mensagem por cada 10 minutos e passar-se a 1 mensagem por cada 1 minuto.

O *payload* da mensagem enviada do dispositivo para o *SigFox* e posteriormente para o servidor é:

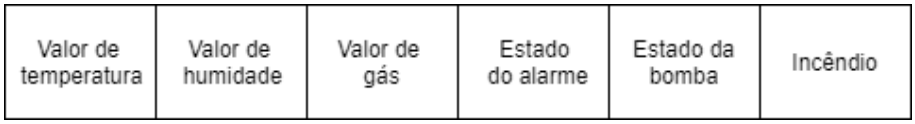

Figura 10: *Payload* de 6 bytes enviado no *Uplink*

Envia-se então um *byte* para o valor da temperatura atual, da humidade e também do gás. Um outro byte para dizer se o alarme e a bomba estão ligados ou não, e outro a dizer se está a haver incêndio.

4.2.2 *Downlink*

O *Downlink* segue regras específicas a cumprir. Têm de ser sempre enviados 8 bytes de *payload* do servidor para o *Sigfox*.

O *payload* da mensagem enviada do dispositivo para o *SigFox* e posteriormente para o servidor é:

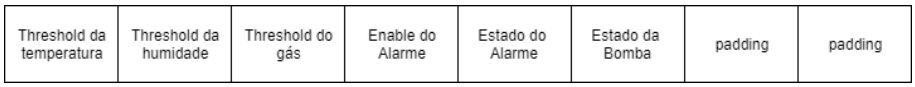

Figura 11: *Payload* de 8 bytes enviado no *Downlink*

Envia-se os *Thresholds* definidos pelo administrador (1 *byte* para cada um dos parâmetros), 1 *byte* para ativação/desativação do alarme. Refere-se que os bytes enviados para o estado do Alarme e estado da Bomba são para uma funcionalidade adicional: o utilizador a qualquer altura pode tomar a decisão de ativar a bomba e o alarme caso entenda que é necessário. Os dois *bytes* de *padding* são só para fazer os 8 *bytes* necessários. No servidor para se responder a um pedido de *Downlink* retorna-se neste formato:

```
return HttpResponse(json.dumps({deviceId: {"downlinkData": sendData}}), content_type="application/json")
```

Figura 12: Retorno do servidor

em que o identificador do dispositivo (4 *bytes* vai no cabeçalho e o *downlinkData* é o *payload* que segue o formato especificado com 8 *bytes*.

O número médio de mensagens diárias de *Downlink* é de 4 mensagens por dia (por dispositivo). No entanto se se solicitar mais mensagens tentar-se enviar à mesma, só que com menor prioridade que as mensagens enviadas por outros dispositivos que não tenham atingido o limite. Em testes no desenvolver do projeto conseguímos enviar sempre muito mais que 4 mensagens de *Downlink* por dia e todas elas foram recebidas com sucesso.

5 Funcionamento Geral

Na figura 13 apresenta-se o fluxograma do programa criado em Arduino IDE que resume a sequência de atividades realizadas pelo microcontrolador.

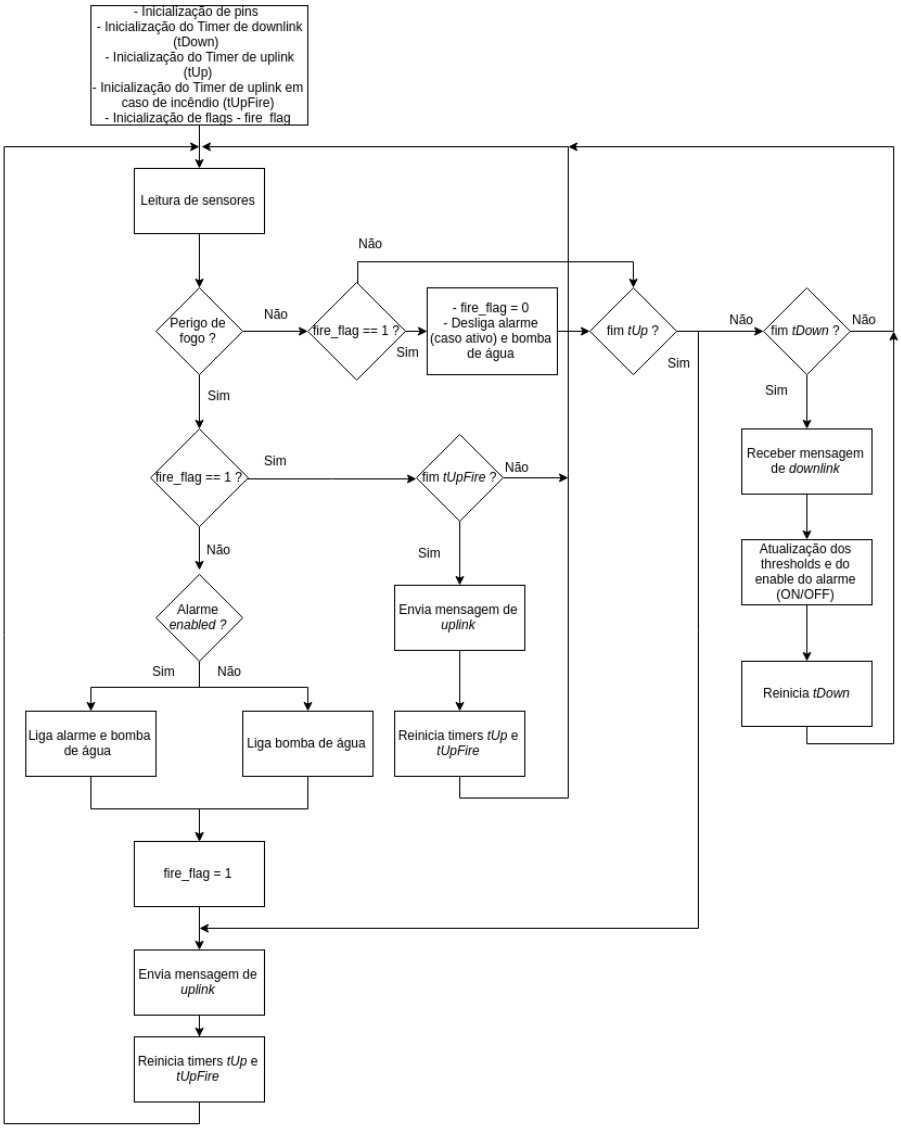

Figura 13: Fluxograma do programa implementado em Arduino IDE

Começa-se por fazer uma inicialização de variáveis associadas aos *timers*, *flags* e *pins* necessários. O programa lê os valores dos sensores e se detetar um novo incêndio (Perigo de fogo == 1 e fire_flag == 0), ativam-se a bomba de água, o alarme (caso esteja *enabled*), a *fire_flag* que indica que está a ocorrer incêndio e envia-se uma mensagem de *uplink* para o servidor da *SigFox*. Se a *flag_fire* estiver ativa e os valores dos sensores indicarem que não há perigo de incêndio, significa que o fogo foi extinguido e consequentemente desativam-se os atuadores e a *flag_fire* (*flag_fire* = 0).

É de referir que se considera perigo de fogo se a temperatura medida for superior a 35 ºC, ou se num intervalo de tempo de 2 minutos existir uma variação superior a 10 ºC e a temperatura for superior a 30 ºC. Para determinar a concentração de gases padrão numa situação de incêndio, realizou-se um teste com auxílio de um isqueiro e

expiração, tendo-se observado um aumento do valor lido pelo sensor de 400 para 420, pelo que o *threshold* associado à concentração de gases irá rondar este último valor.

6 Servidor

O servidor permite fazer a ponte entre o dispositivo *Akeru + SigFox* com a aplicação *Android* dos utilizadores.

O servidor `HTTP` *Backend* foi realizado em `Python`, na *webframework* `Django`. Com esta *framework* existe uma grande facilidade em trabalhar com modelos (bases de dados) e além disso, com esta *framework* cada URL tem associado um evento, o que facilita bastante a implementação. Elaborou-se uma api para permitir a utilização da base de dados desenvolvida em novas aplicações que outros utilizadores possam querer realizar. Esta api baseia-se no estilo arquitetural `REST`, onde são feitos pedidos `GET` e `POST`, `PUT` ou `REMOVE` à api.

Refere-se que o servidor foi implementado de forma a se garantir escalabilidade. Ou seja, cada utilizador ao fazer um pedido `GET` ou `POST`, é passado no *Body* da mensagem um *Token* que identifica o dispositivo que pretende fazer determinada ação. Desta forma cada utilizador consegue aceder apenas aos dispositivos que a si pertencem, mas o servidor faz a gestão e guarda nas bases de daddos informação referente a todos os dispositivos que forem vendidos, assim como todos os utilizadores registados.

6.1 Base de dados

Uma vez necessário que toda a informação indispensável para as funcionaliades do programa fosse guardada de forma sólida e segura, tornou-se óbvio que os dados teriam de ser armazenados em memória presistente (em disco). Para esta causa fez-se uso de uma base de dados baseada na linguagem `SQL`.

6.1.1 Sistema de gerenciamento da base de dados relacional (RDMS)

O sistema de administração utilizado para interagir com a base de dados foi o `MySQL`. Inicialmente, apenas para efeitos locais, começou por utilizar-se o sistema de administração `SQLite`, no entanto, quando chegou a altura de escalar o programa (para uma *cloud*, como será explicado mais à frente), houve a necessidade de correr a uma base de dados alojada num sistema operativo focado para a mesma, método que não é suportado pelo `SQLite`. Já o `MySQL`, não só suporta este método, como fácilmente se conseguiu pôr a correr numa máquina Linux. Há ainda que referir que usando o `SQLite` ter-se-iam perdido várias capacidades como consistência imediata, métodos de replicação, etc.

6.1.2 Arquitetura da base de dados

Vai-se agora de encontro ao desenho da base de dados em sí, isto é, aos dados que armazena, como os armazena, e os seus significados.

Realça-se as relações existentes entre as tabelas: a tabela dos `Devices` tem um utilizador associado (da tabela `Users`). A tabela `Conditions` tem um dispositivo associado da tabela `Devices`. A tabela que contém todos os incêndios tem um dispositivo associado, da tabela `Devices`.

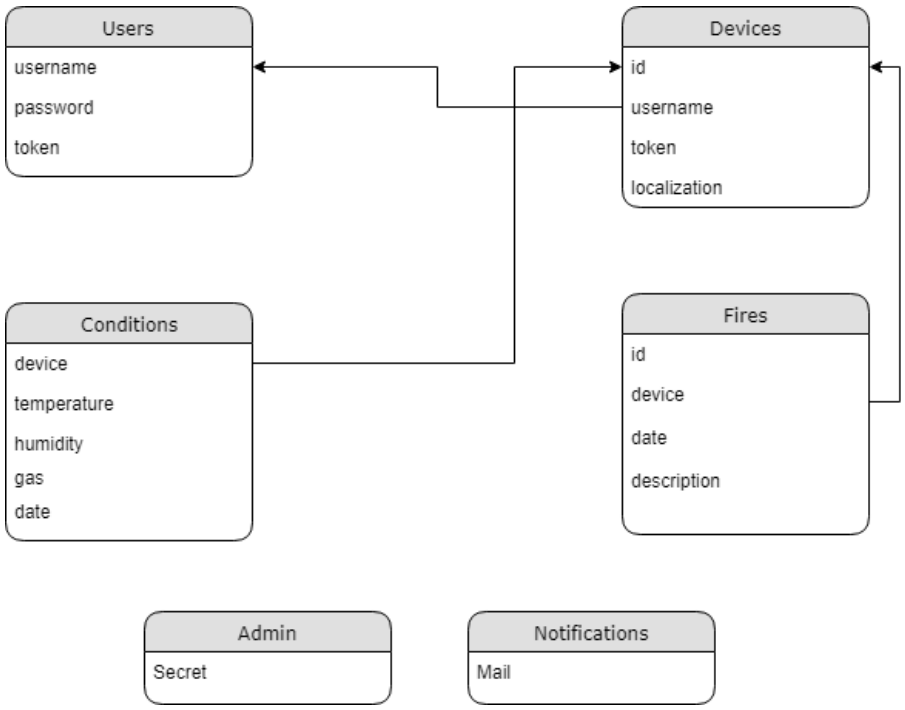

Figura 14: Base de Dados e relações entre tabelas

A tabela `Notifications` serve para guardar os e-mails de utilizadores que preencheram os `Newsletter` com o seu e-mail no site realizado, que se encontra online e que é acedido pelo browser. Refere-se que a tabela `Admin`, que guarda *secrets* é um código gerado pelo servidor e dado ao administrador quando este faz `LogIn` no sistema. O administrador para a execução de qualquer atividade, tem de enviar este *secret* que lhe foi atribuído (caso não exista não consegue). O mesmo acontece para o *token* da tabela de `Users`.

6.2 `REST API` - *endpoints*

Os *endpoints* começados por `/app` são para os utilizadores regulares(para a aplicação em sí), os começados em `/admin` para as operações do administrador,e só o administrador tem acesso. Também se tem os que permitem interagir com o *SigFox*.

6.2.1 Administrador

- ***endpoint***: `/admin/`, **Método** `POST` - Autenticação no sistema. Envia um *json* com o *username* e a *password*. É retornado um *json* com um *secret* caso seja bem sucedido, e -1 caso contrário.
- ***endpoint***: `/admin/users/`, **Método** `POST` - Permite ao administrador obter todos os utilizadores registados. Envia um *json* com o seu *secret* e recebe de retorno um *jsonArray* com todos os utilizadores.
- ***endpoint***: `/admin/devices/`, **Método** `POST` e `GET` - Se for um método GET, permite obter um *jsonArray* com todos os dispositivos. Se for um `POST` permite registar um novo dispositivo. Neste caso, além de enviar *json* com o seu *secret*,

envia também o identificador do novo dispositivo, e é-lhe retornado um *Token* para esse dispositivo, para ser fornecido ao utilizador.

- ***endpoint***: `/admin/thresholds/`, **Método** `POST` e `GET`. Se for um pedido `GET` é retornado um *json* com os *Thresholds* atuais. Se for um `POST`, permite definir os novos *Thresholds* para o sistema, que têm de ser enviados num *json*.

- ***endpoint***: `/admin/fires/`, **Método** `POST` e `GET` - Se for um pedido `GET` é retornado um *jsonArray* com todos os incêndios registados de todos os dispositivos. Se for um `POST`, permite adicionar uma descrição a um dos incêndios registados. Neste caso, tem de enviar a descrição, bem como o dispositivo associado num *json*.

A figura 15 ilustra resumidamente todos os *endpoints* referentes ao administrador

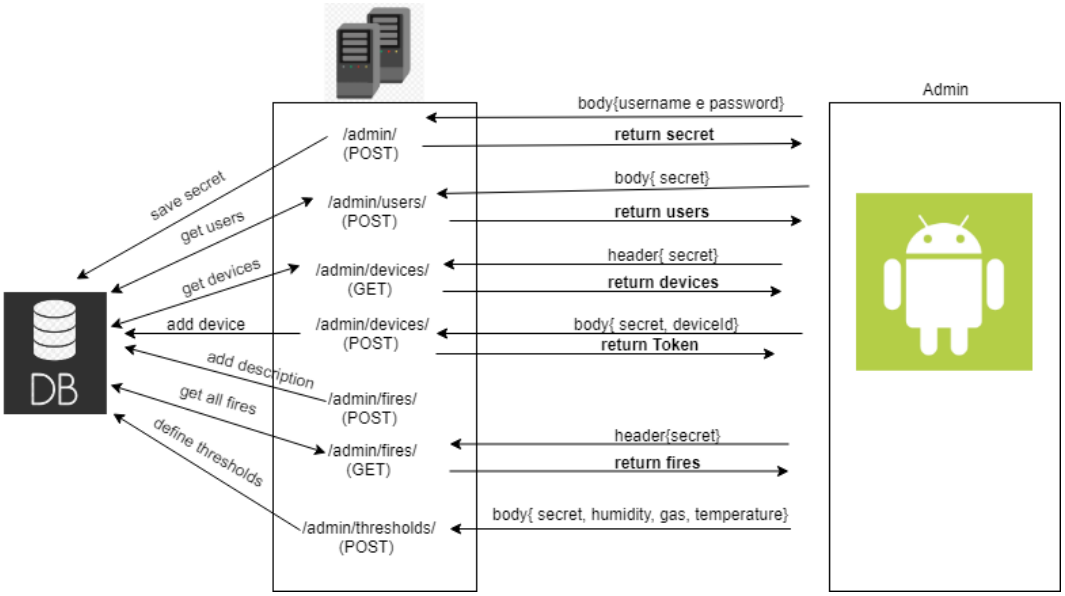

Figura 15: Fluxograma da interação do administrador com a `REST API`

6.2.2 Utilizador

- ***endpoint***: `/app/`, **Método** `POST` - Autenticação no sistema no modo utilizador. Envia um *json* com o *username* e a *password*. Recebe um *json* com um *token* caso seja bem sucedido, e 0 caso contrário.

- ***endpoint***: `/app/signup/`, **Método** `POST` - É enviado um *json* com *username* e *password*. É retornado um *json* com um 1 caso se consiga registar, e 0 caso já exista esse utilizador registado.

- ***endpoint***: `/app/devices/`, **Método** `POST`, `GET` - Caso seja um pedido `POST` o utilizador pretende registar em sua posse um dispositivo (que o administrador lhe vendeu). Para isso o administrador enviou-lhe um *token* (que se encontra na tabela base de dados `Devices`). Assim o utilizador envia no *body* do pedido o *token* do dispositivo que pretende registar como seu, a localização do dispositivo (cozinha, sala, ..), e um *token* que lhe foi atribuído quando iniciou sessão na aplicação, para confirmação de que é um utilizador registado. Caso seja um pedido `GET` é

retornado um *JsonArray* com todos os dispositivos que este utilizador tem em sua posse. Mais uma vez, o utiliador tem de enviar o seu *token* no pedido realizado.

- ***endpoint***: `/app/conditions/`, **Método** `POST` - O utilizador envia um *json* com o seu *token*, bem como o identificador do dispositivo que pretende verificar as condições atuais, e é-lhe retornado um *json* com as condições atuais (temperatura, humidade, gás).

A figura 16 resume toda a interação do utilizador com o servidor.

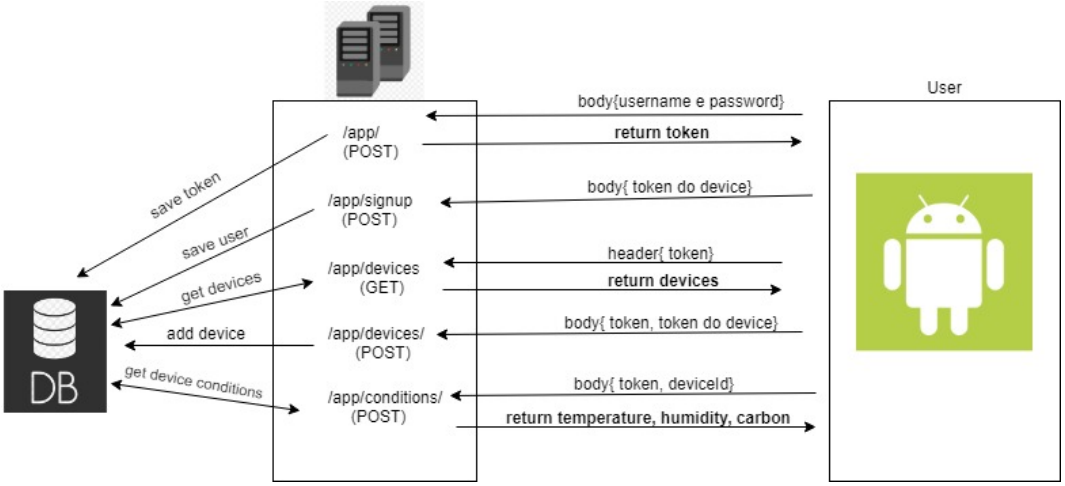

Figura 16: Fluxograma da interação do utilizador com a `REST API`

Refere-se que a distinção entre *token* e *token* do device. O *token* é o código gerado aquando do início de sessão na aplicação por parte do utilizador. O *token* do device é o *token* atribuído ao **administrador** quando registou o dispositivo. O utilizador ao comprar o dispositivo, o administrador fornece-lhe este *token*.

6.2.3 *SigFox*

- ***endpoint***: `/sensors/`, **Método** `POST` - O *SigFox* envia para o servidor os dados periódicos no formato representado na figura 10. Além disto envia o identificador do dispositivo. O servidor armazena os dados numa base de dados, e caso seja enviado o estado de incêndio, invoca a função que utiliza *Push-Notifications* para enviar notificação aos utilizadores que têm este dispositivo registado.

- ***endpoint***: `/downlink/`, **Método** `POST` - Utilizado no *Downlink*. No corpo da mensagem enviada pelo *SigFox* no pedido é enviado o identificador do dispositivo. O servidor retorna para o *SigFox* um *json* com os dados bem como o formato representado nas figuras 11 e 12, assim como o identificador do dispositivo.

6.2.4 *Push-Up notiffications - Firebase*

Na figura 17 encontra-se representado o fluxo de comunicação entre os diferentes componentes do sistema na ocorrência de um incêndio.

Se o arduino detetar que os valores recolhidos pelos sensores estão acima dos *thresholds*, é enviada uma mensagem de *Uplink* com os dados dos sensores e a *flag* de

Figura 17: *Push Notification* na ocorrência de um incêndio

alerta ativa para o servidor da `SigFox`, que reencaminha para o servidor que se encontra alojado na `Google Cloud` (através do *endpoint* `/app/sensors/`, **Método** `POST`). Este, ao descodificar os dados e perceber que há alerta de incêndio, identifica o dispositivo e envia um alerta para a `Firebase Cloud Messaging`, que trata de enviar uma *Push Notification* para todos os utilizadores que estão associados ao dispositivo. Quando o utilizador ao comprar o dispositivo recebe um *token* para o dispositivo, ao registá-lo, no código da aplicação, é ativado a notificação referente a esse dispositivo através do código *token*. Depois ao ser enviado o alerta, envia-se uma *Push Notification* para todos os que ativaram com este *token*.

Na figura 18 mostra-se a notificação recebida pelo utilizador *fire_user*, onde se alerta a deteção de um incêndio pelo dispositivo localizado na *CEO room*.

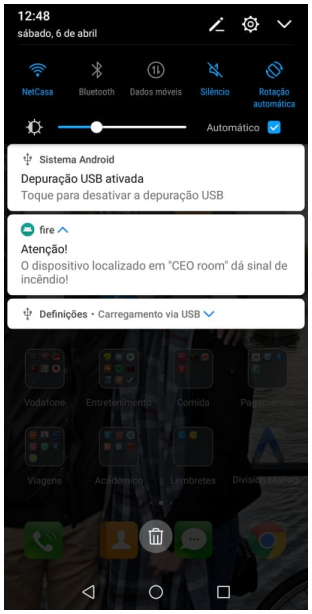

Figura 18: Notificação *push* de deteção de um incêndio

6.3 Site no Browser

Foi também realizado um site a ser acedido pelo *Browser* através de: https://fire-detection-235819.appspot.com/app/. A página inicial do site ilustra-se na figura .

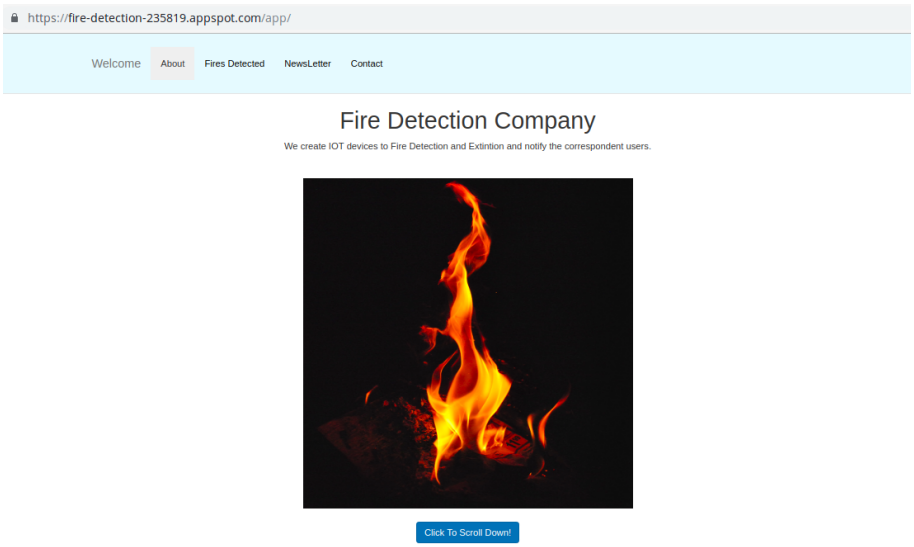

Figura 19: Página inicial do site

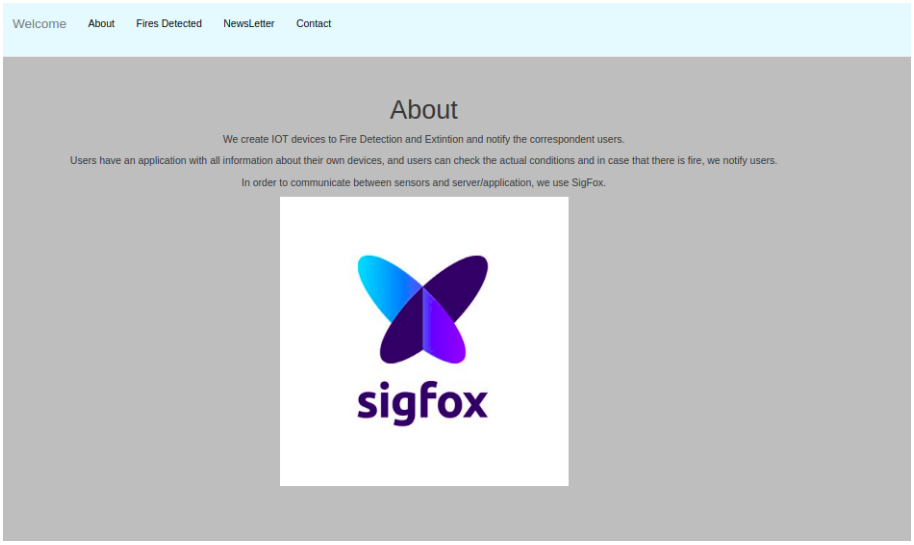

Figura 20: Secção *About* do site

Decidiu-se colocar uma secção no site com todos os incêndios ocorridos e registados na base de dados `Fires`. A descrição é colocada pelo administrador, que tem a opção de adicionar uma descrição a um dado incêndio na dua aplicação. Para que se consiga aparecer uma tabela com todos os incêndios, usou-se `Ajax` para fazer pedidos `GET` ao *endpoint* que retorna todos os incêndios.

No que toca à *Newsletter*, decidiu-se que caso se inserisse o e-mail, fica registado na base de dados na tabela `Notifications` (ver 14). Caso o administrador realize a ação enviar uma mensagem informativa para todos os utilizadores registados, todos os que estão inseridos nesta tabela recebem a informação por e-mail.

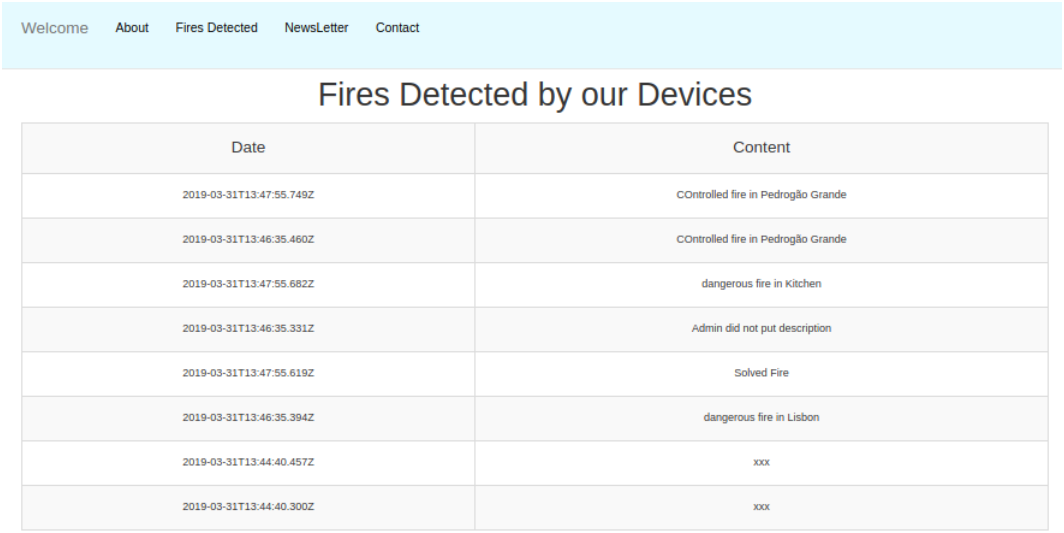

Figura 21: Todos os incêndios ocorridos que se encontram na base de dados

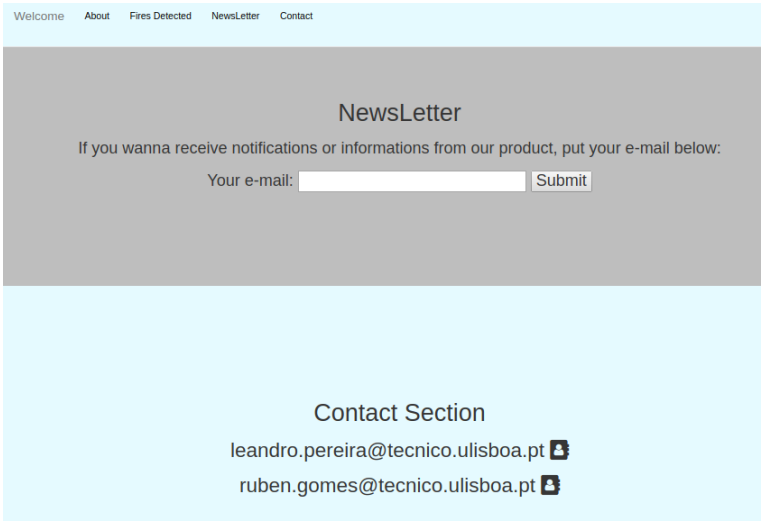

Figura 22: *NewsLetter* e informação dos contactos

6.4 Implementação do sistema na Google Cloud Platform

Depois do desenvolviemento parcial (já quase total) da aplicação, esta foi implantada na `Google Cloud`, num dos serviçoes que esta oferece - `Google App Engine flexible environment`.

Começou então por aceder-se à `Google Cloud Platform`, a partir de onde se podem administrar todos os recursos alojados na `cloud`, e instanciar uma máquina virtual com o sistema operativo Linux (distribuição *Debian*). É de notar que neste passo se escolheu o serviço gratuito, pelo que grandes necessidades de memória ou processamento não serão satisfeitas com esta escolha, no entanto, há sempre a possibilidade de assinar pelos serviços pagos, não sendo necessário realojar nem reiniciar todos os componentes já instalados e funcionais.

De seguida, antes de correr a aplicação remotamente, foi necessário lançar um servidor para a base de dados utilizada. Uma vez que numa fase inicial (apenas local), se estava a utilizar a base de dados `SQLite`, que não suporta ser corrida num servidor com

o propósito de manusear uma base e dados, alterou-se para `MySQL`. Criou-se então uma instância `DBMS` do tipo `MySQL (Google Cloud SQL instance)`, e de seguida, não só para facilidade de desenvolvimento como para facilidade de testes e *debug* gerou-se um *proxy* para que fosse possível, através deste, correr a aplicação localmente, mas usando a base de dados remota instanciada na `cloud` da `Google`.

Agora, na aplicação em si, é necessário alterar o programa, de modo a que este direcione a informação que pretende guardar de forma persistente na base de dados instanciada. Uma vez que esta base de dados é um servidor a correr de forma independente, acede-se a esta tal como a outra máquina qualquer, através de um IP e de um porto. Será também necessário, para um acesso válido, indicar o nome da base de dados e as credenciais de acesso.

Depois de já ter a base de dados alojada e com todas as funcionalidades desejadas (todos os testes à base de dados podem ser feitos localmente através do *proxy*), é ainda necessário escrever/gerar um ficheiro "requirements.txt"com a informação de quais os pacotes e dependências que serão necessários instalar na máquina virtual que está a correr na `cloud`. Uma vez que se trata também de um computar a funcionar de forma independente, para correr a aplicação que lhe será implantada, precisa de saber quais os pacotes que são necessários, tal como foi feito no desenvolvimento local.

Agora sim, está-se em condições de implantar a aplicação no serviço da `cloud` instanciado. O ficherio que ordena a implantação (*deploy*), é o ficehiro "app.yaml". É necessário indicar neste qual será o interpretador (e a sua versão) a ser utilizado, neste caso é a versão mais recente de python.

Para fazer a implantação basta, com todo este *background* já preparado, correr o seguindo comando no terminal:

```
$ g app deploy
```

A implantação leva alguns minutos, visto que se está a reiniciar um servidor e não apenas a correr um programa. Por esta razão torna-se bastante útil poder recorrer à base de dados através de um *proxy* pois fica-se com a liberdade de correr o programa localmente, que para correr desta forma é quase instantâneo. Uma vez finalizada a implantação, vê-se no terminal qual o link de acesso à aplicação por um browser, após o qual o processo lançado pelo comando `$ google app deploy` se vê concluido.

No caso deste projeto, para aceder à aplicação, basta aceder ao link:
https://fire-detection-235819.appspot.com/app/

7 Aplicação Android

7.1 Arquitetura da aplicação

Um dos objetivos do projeto é a realização de uma aplicação *Android*, que permite ao utilizador consultar informações em tempo real sobre o seu produto. Durante a implementação da aplicação, pensou-se num contexto prático e decidiu-se criar funcionalidades para um administrador do produto e outras funcionalidades de utilizador, relacionadas com obtenção de informação sobre os seus dispositivos. Nas seguintes secções, ilustra-se a arquitetura da aplicação, criada em `Android Studio`, nos modos utilizador e administrador, fazendo-se referência às principais diferenças.

7.1.1 Administrador

Como se verifica na figura 23, para o administrador aceder à aplicação, tem de fazer *Login* com o *username* e *password* de administrador e terá de ativar a *check box* para indicar que pretende fazer uma autenticação de administrador. Se a autenticação for bem sucedida, existe a mudança de atividade para o menu de administrador, onde poderá escolher entre listar os utilizadores registados na aplicação, listar a ocorrência de fogos ou alterar os *thresholds* classificadores de um incêndio.

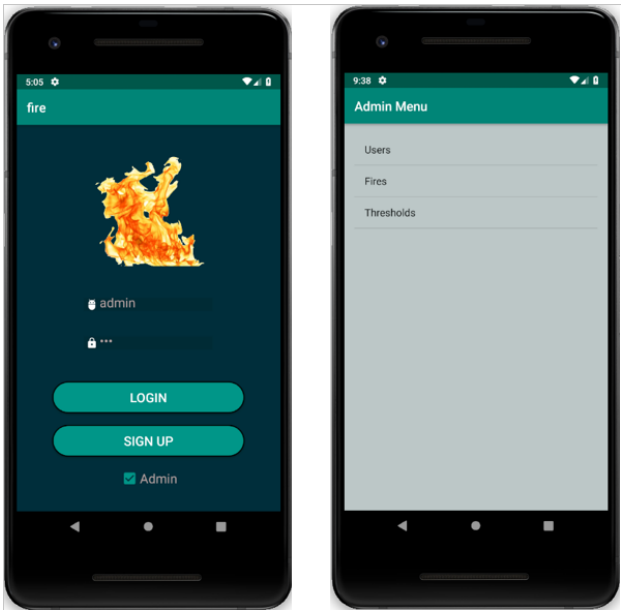

Figura 23: Página de *Login* e do menu de opções do administrador

Ao escolher a opção *users*, existe uma mudança de atividade para uma página que lista todos os utilizadores registados. Como se pode verificar na figura 24, ao escolher um dos utilizadores (no exemplo escolheu-se o utilizador *fire_user*), aparece a lista de dispositivos associada ao mesmo. O administrador tem a possibilidade de selecionar um dispositivo, podendo consultar o estado atual (humidade relativa e temperatura), se o alarme está ativo sem o poder alterar e o registo de incêndios associado ao dispositivo.

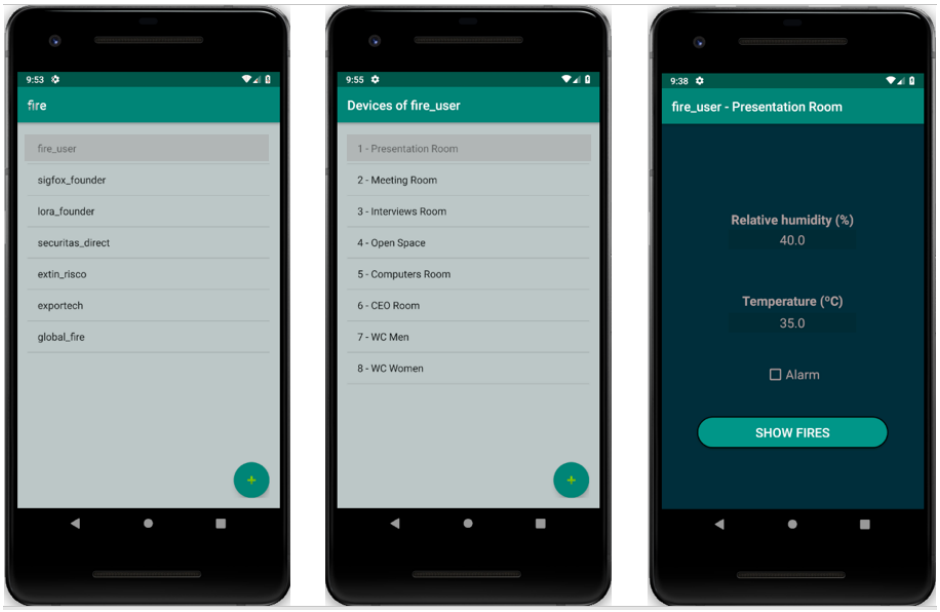

Figura 24: Página com lista de utilizadores e respetivos dispositivos

Se o administrador escolher a opção *Fires*, é mostrada uma lista com todos os incêndios ocorridos associados a todos os dispositivos. Como se verifica na figura 25, se o administrador selecionar a opção *Thresholds*, é apresentada uma página com os valores atuais e onde poderá editá-los. Ao clicar em *apply* é iniciada uma comunicação com o servidor, com o intuito de alterar os *thresholds* em todos os dispositivos registados.

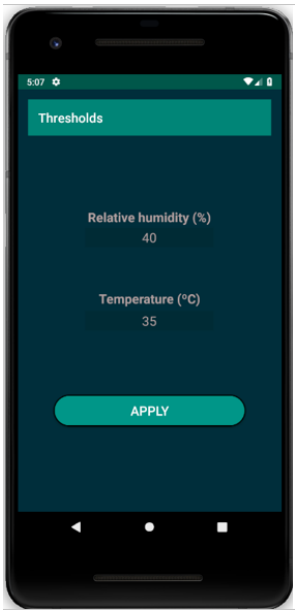

Figura 25: Página onde o administrador pode alterar os *thresholds*

7.1.2 Utilizador

Um utilizador ao descarregar a aplicação terá de se registar na base de dados. Para isso, na página de *login*, basta selecionar a opção *Sign Up*, onde define o seu *username*

e *password*. Para fazer *login*, o utilizador terá de introduzir o seu *username* e *password* com a *check box* de administrador desativada. Se a autenticação for bem sucedida, há uma mudança de atividade, onde é apresentada a lista dos seus dispositivos.

O utilizador tem a possibilidade de selecionar um dispositivo, podendo consultar o estado atual (humidade relativa e temperatura) e se o alarme está ativo. Além desta consulta, o utilizador poderá ainda ativar/desativar o alarme e consultar o registo de incêndios associados ao dispositivo. É importante referir que caso o utilizador introduza credenciais incorretas, é-lhe enviado um *Toast* com a mensagem "*Incorrect details*".

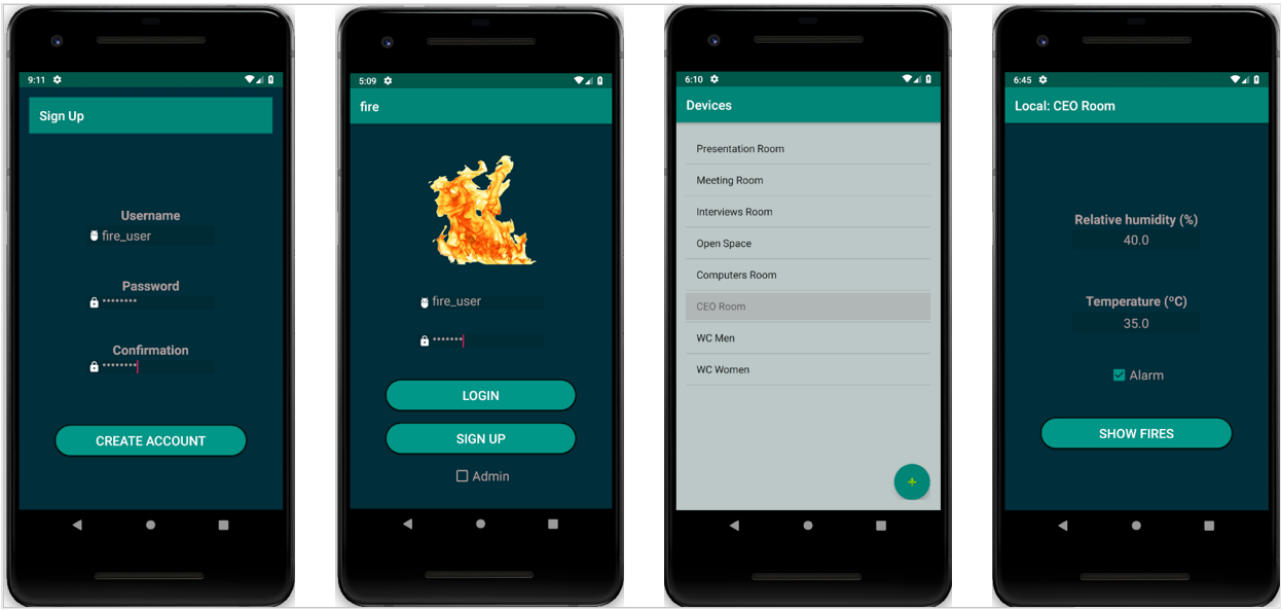

Figura 26: Sequência de páginas associada à funcionalidade de um utilizador

8 Objetivos para a próxima entrega

- Completar todas as funcionalidades da aplicação.
- Melhorar o site
- Ajustar melhor a deteção de incêndio
- Funcionalidade de enviar e-mail a todos os subscritos
- Fazer um vídeo do funcionamento do projeto

Referências

[1] https://patentimages.storage.googleapis.com/84/6d/b7/5163fb2585b760/US5486811.pdf

[2] https://www.researchgate.net/profile/R_Ragupathy/publication/322881482_-Efficient_Smart_Emergency_Response_System_for_Fire_Hazards_using_-IoT/links/5adf6444aca272fdaf8b1942/Efficient-Smart-Emergency-Response-System-for-Fire-Hazards-using-IoT.pdf

[3] https://www.mouser.com/ds/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf

[4] https://www.olimex.com/Products/Components/Sensors/Gas/SNS-MQ135/resources/SNS-MQ135.pdf

[5] https://media.digikey.com/pdf/Data%20Sheets/Seeed%20Technology/108990019_-Web.pdf

[6] https://www.sigfox.com/en

[7] https://www.sigfox.com/en/coverage?fbclid=IwAR1eslg_OZnwZRFHeIgeVH-sieu0XoFYjWBeG6D4yCSJCqyU91T9QJ9Ag3o8

[8] http://narrownet.pt/

[9] https://partners.sigfox.com/products/sifi

[10] https://partners.sigfox.com/products/gs558d-smoke-alarm-with-wireless-interconnect-and-heat-detection

[11] https://partners.sigfox.com/products/smockeo

[12] https://build.sigfox.com/steps/study