Redes Móveis e Sem Fios

Relatório Final

Development of Internet of Things sensor monitoring based on SigFox, Arduino and Android

Bernardo Gomes, 75573 Diogo Martins, 75462

10 de Maio de 2016

Conteúdo

1 Objectivo

O objectivo do projecto é o desenvolvimento de um sistema de monitorização de temperatura.

O sistema, deverá ser baseado num sensor de temperatura associado a um dispositivo *Arduino (Akeru 3.3)*, que irá comunicar as suas medições a um servidor *SigFox*, armazenando-as na *Cloud*.

Na óptica do utilizador, irá ser desenvolvida uma aplicação em ambiente *Android*, que fornecerá os dados presentes na *Cloud* com uma apresentação *user friendly*. Pretende-se ainda que seja possível que o utilizador registe um novo dispositivo a monitorizar na aplicação, bem como definir alarmes para certos valores de temperatura.

2 Solução encontrada

Tal como referido na secção anterior, a monitorização da temperatura e da qualidade de medição do sensor, irá ser feita pelo utilizador com recurso à aplicação, mas tendo a *Cloud SigFox* como intermediária.

Por forma a que os dados de cada utilizador sejam independentes do dispositivo utilizado, foi construída uma base de dados adicional, que guarda informação de *login* bem como dos *devices* e os *thresholds* de alarme de cada utilizador.

A arquitectura será então a apresentada na figura 1:

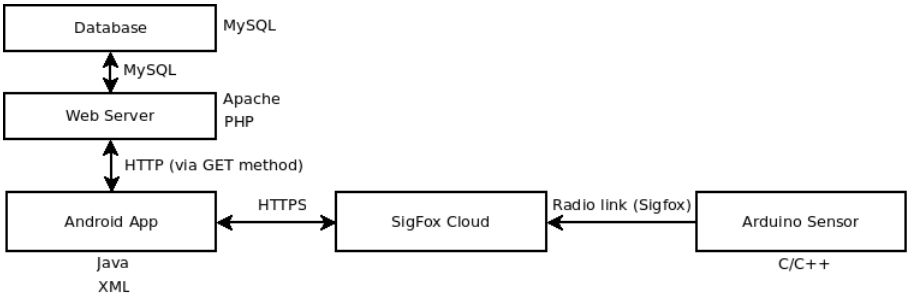

Figura 1: Arquitectura geral

2.1 Aplicação *Android*

A aplicação *Android*, com a qual o utilizador irá ter contacto directo, será constituída por cinco actividades:

- *MainActivity*;
- *CreateLogActivity*;
- *MonitorActivity*;
- *NewAlarmActivity*;
- *AddDeviceActivity*.

As relações entre as actividades descritas, encontram-se representadas na figura 2.

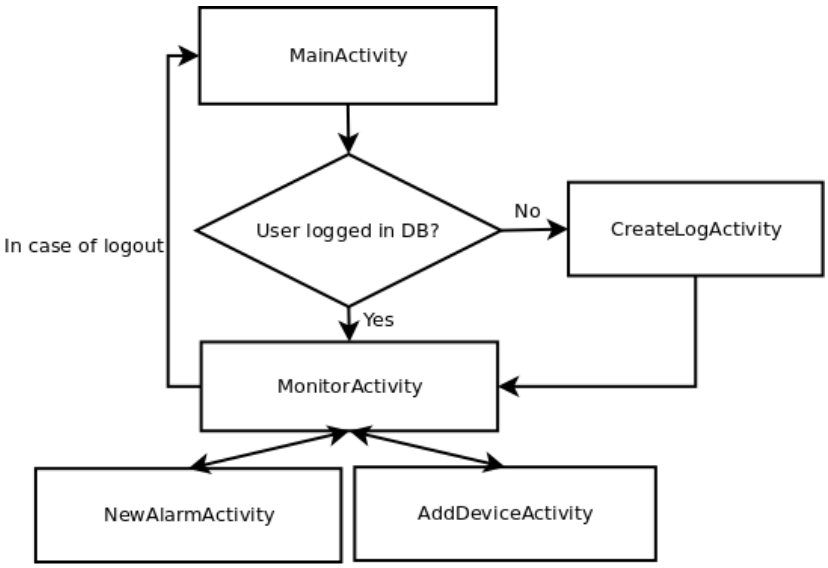

Figura 2: Arquitectura da aplicação *Android*

A *MainActivity* tem como objectivo perguntar ao utilizador o nome com o qual está registado na rede, realizando *queries* à base de dados com as informações do utilzador. No caso de existir registo prévio, as informações são armazenadas num ficheiro que irá funcionar como *cache* na aplicação e é lançada a *MonitorActivity*. Caso o utilizador não tenha registo, é lançada a *CreateLogActivity* para registar o novo utilizador.

A actividade de registo de um utilizador (*CreateLogActivity*), terá cinco campos de inserção de texto: dois para a aplicação/base de dados(*App Username, Password*) e três para o *Backend SigFox*(*SigFox Name, SigFox Password, Device-id*). Os parâmetros da aplicação, são guardados na base de dados e serão utilizados para diferenciar os utilizadores da aplicação, enquanto que os parâmetros do *Backend* são gravados no ficheiro de texto descrito anteriormente, bem como na base de dados central. Caso haja o registo de um utilizador já existente, é lançada uma mensagem de erro, permitindo no entanto que o utilizador modifique o registo por forma a não existir colisões de utilizadores. De seguida, a aplicação irá lançar a *MonitorActivity*.

A actividade de visualização da informação da *Cloud* (*Monitor*), é constituída por duas *threads* principais. A primeira consiste na obtenção das mensagens do dispositivo por pedidos HTTPS (GET) periódicos, de acordo com a informação de registo do utilizador e que é actividada por uma *checkbox*. Posteriormente a resposta será enviada para a *thread* principal (da API) que a disponibiliza ao utilizador. No caso de a *checkbox "periodic"* não estar activa, os pedidos podem ser efectuados apenas ao clicar no botão de pedido de informação.

Por defeito, as mensagens recebidas na aplicação serão apenas as medidas após a última leitura efectuada pelo utilizador. No entanto, o utilizador terá a opção de visualizar todas as mensagens armazenadas, seleccionando a *checkbox "get all messages"*, abdicando no entanto da recepção dos alarmes.

Será ainda possível apagar o ecrã, carregando no botão do termómetro, situado no canto superior direito da actividade.

O *layout* desta actividade encontra-se na Figura 3.

Esta actividade tem ainda a opção de registar um novo dispositivo para monitorização, bem como adicionar um novo alarme de temperatura. No caso de um *threshold* de temperatura, lido da base de dados no início da aplicação, ser ultrapassado, a *thread* que realiza o *parsing* da informação deverá lançar uma notificação ao utilizador.

As actividades *AddDeviceActivity* e *NewAlarmActivity* serão apenas compostas por campos de texto. Após o registo num ficheiro e na base de dados das informações recolhidas, estas actividades irão regressar à actividade principal.

No caso de o utilizador pretender fazer *logout*, é lançada a *MainActivity*, por forma a que seja possível iniciar a sessão com outro *username*.

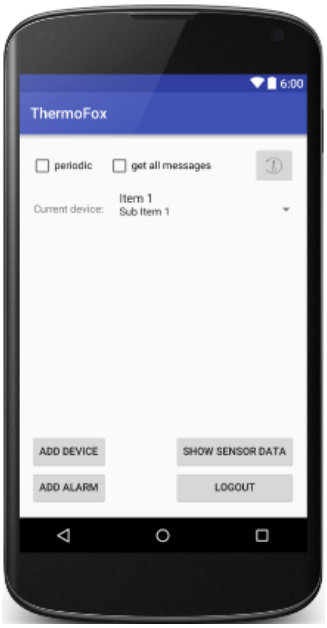

Figura 3: *Layout* da actividade *Monitor*

2.2 Servidor *SigFox*

O papel deste servidor é o armazenamento da informação medida pelo sensor e a recepção de pedidos por parte da aplicação *Android*. Consoante o tipo de pedido, irá realizar uma resposta em *JSON* com as informações das medições.

Em termos de implementação para o projecto, foi apenas necessário conhecer a forma como os pedidos devem ser realizados bem como o formato de resposta.

2.3 Sensor Arduino

O sensor deverá realizar medições periodicamente enviando-as para a *Cloud SigFox*, via rádio. Sendo possível enviar 120 mensagens para a *Cloud SigFox* por dia, envia-se 5 mensagens por hora logo 1 mensagem a cada 12 minutos.

2.4 Web Server

Por forma a realizar pedidos à base de dados, quer de leitura quer de escrita, é utilizado um *Web Server* que recolhe a informação do utilizador recebida pela aplicação através do método *GET* e realiza as operações respectivas na base de dados. Como os pedidos são feitos mediante o *input* do utilizador, este tem de ser tratado antes de se realizar o pedido. Para tal, apenas são permitidos como *input* caracteres pertencentes ao alfabeto e dígitos de 0 a 9. Tal não afectará o pedido uma vez que os parâmetros da *backend SigFox* são somente constituídos por estes caracteres. No caso de ser realizado um pedido de informação, esta é colocado num *array* que é retornado à aplicação em *JSON*. Para retornar mensagens a partir de um determinado tempo é utilizada a opção *since* no campo do *URL* de forma a realizar o pedido a partir da última mensagem lida, ou em caso de primeira utilização todas as mensagens presentes na *Cloud*.

3 Detalhes técnicos

Para a implementação da solução descrita anteriormente, para os diferentes módulos recorreu-se às seguintes tecnologias:

Para a aplicação *Android*:

- **MainAcitivity, CreateLogActivity, NewAlarmActivity e AddDeviceActivity** - Nestes módulos, realizam-se pedidos a um *Web Server* relativamente aos dados de um dado utilizador. Assim, na *MainActivity*, com base nos dados recolhidos da base de dados, é escrito um ficheiro, por forma a efectuar/consultar registos (utilizador, dispositivo ou alarme) noutras actividades. Nas restantes, o objectivo de contacto com o servidor *web* é o de acrescentar informação à base de dados, sendo também o ficheiro interno actualizado.

  Para este efeito, utiliza-se a Classe *FILE* presente em *Android* para o armazenamento interno (*cache*) e por cada pedido ao servidor, é necessária a abertura de *threads* por forma a não bloquear a *User Interface* (UI), recorrendo à classe *AsyncTask*;

- **Monitor** - Neste módulo, será essencial a abertura de uma *thread*, na medida em que a UI principal não permite efectuar pedidos periódicos pelo facto de estes a poderem bloquear. Assim, caso o utiizador queira realizar pedidos periódicos, esta *thread* será aberta (recorrendo à classe *Thread*). Caso contrário, será apenas necessário utilizar novamente a classe *AsyncTask* para os pedidos, que apenas serão realzados mediante a pressão de um botão na aplicação.

  Para efectuar os pedidos HTTPS, utiliza-se a classe *HttpURLConnection*. O formato destes pedidos seguem as normas descritas na REST API-Students fornecida pelo corpo docente.

  Ao receber a resposta, a *thread* é processada com *parsing* de *JSON* com recurso à classe *JsonReader*. Após o processamento da resposta, é verificado se a temperatura recebida ultrapassa algum *threshold* definido pelo utilizador. Em caso afirmativo, é gerada uma notificação escrita e sonora com recurso a um objecto *Notification* e *Ringtone*.

Para a implementação da base de dados para armazenar os registos de cada utilizador, foi seguido o seguinte modelo:

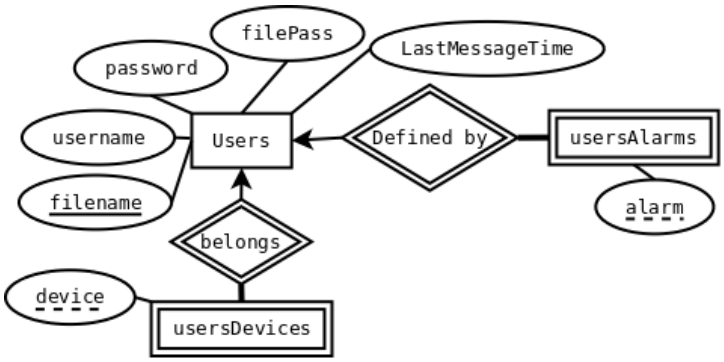

Figura 4: Arquitectura da base de dados

Como tal, a base de dados irá ter três tabelas, tal como se pode verificar no *script*, colocado na figura 5, utilizado para a sua criação.

```
create table users(
    filename varchar(40),
    filePass varchar(255),
    userName varchar(255),
    password varchar(255),
    lastMessageTime varchar(40),
    primary key(filename));

create table usersDevices(
    filename varchar(40),
    device varchar(40),
    primary key(filename, device));

create table usersAlarms(
    filename varchar(40),
    alarm varchar(40),
    primary key(filename, alarm));
```

Figura 5: Script utilizado para a criação da base de dados

Relativamente ao alojamento da base de dados e do servidor *web*, recorreu-se ao servidor do IST *sigma.tecnico.ulisboa.pt*. O acesso à base de dados encontra-se protegido contra *SQL Injection* utilizando-se para tal *prepared statements with variable binding*. O tráfego no entanto não se encontra encriptado pois não se considerou uma ameaça de grande nível para este tipo de projecto, porém a password correspondente da aplicação não é armazenada em *plaintext* mas sim em formato de *hash* com adição de *salt* utilizando as funções *PHP password_hash* para armazenamento e *password_verify* para validação da password durante o *login*. Para a implementação do sensor de temperatura, recorreu-se às bibliotecas associadas ao dispositivo *Akeru.h* disponibilizadas pelo *Snootlab* bem como a biblioteca associada ao sensor *idDHT11.h*. Para enviar mensagens periodicamente de 12 em 12 minutos utilizou-se a função *delay()* presente em ambiente *Arduino*. O sensor de temperatura usado foi o DHT11, sensor que pode medir tanto temperatura como humidade.

4 Verificações da solução encontrada

A secção que se segue, pretende mostrar algumas sequências de comandos que demonstram o bom funcionamento da aplicação, explicando as acções tomadas em cada fase.

4.1 Registo do primeiro utilizador

A sequência seguinte demonstra o funcionamento do registo do primeiro utilizador:

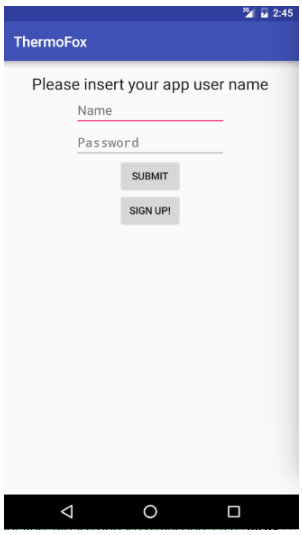

Figura 6: Abertura da App

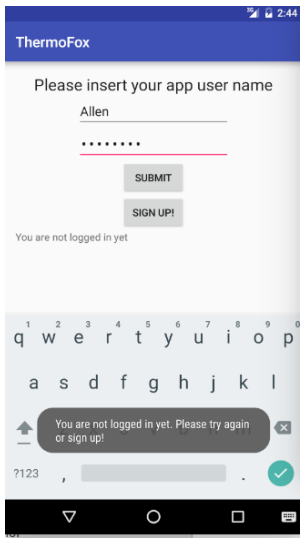

Figura 7: Tentativa de *login* (*user* não existente)

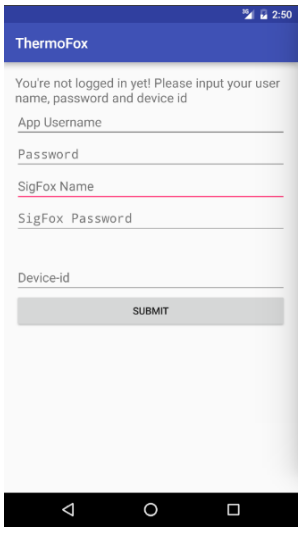

Figura 8: *Layout* de registo de um novo *user*

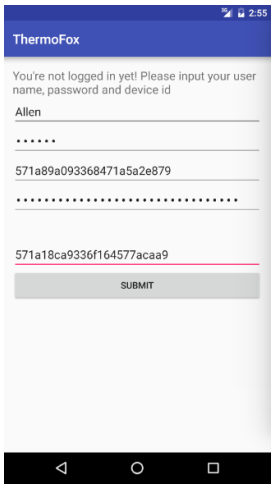

Figura 9: Registo de um novo *user*

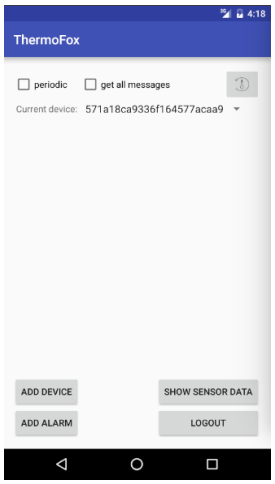

Figura 10: Actividade *Monitor*

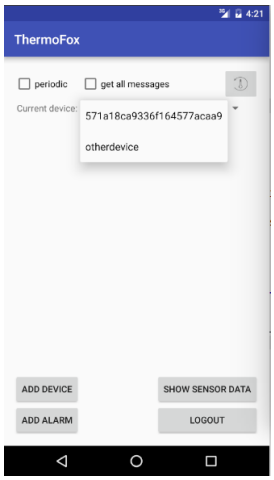

Figura 11: Diferentes dispositivos

Tal como verificado na figura 11 quando é adicionado um novo *device*, é possível seleccionar o dispositivo pretendido através de uma *drop-down list*. O processo de adicionar um novo dispositivo é o descrito na seguinte sequência:

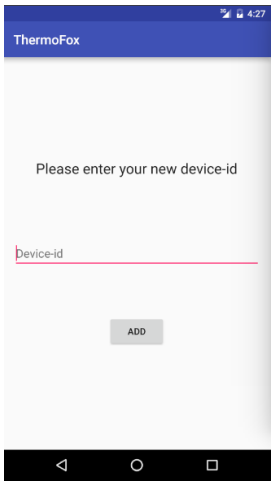

Figura 12: Registo de um novo *device*

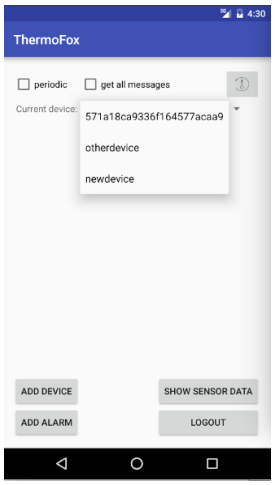

Figura 13: Dispositivo adicionado

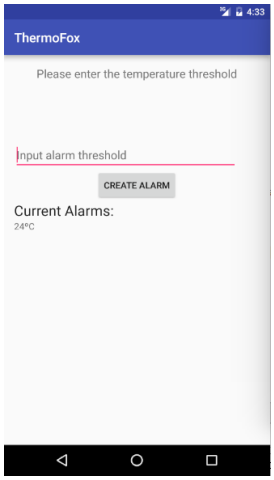

Figura 14: Adição de um novo *threshold*

Na figura 14 é representada a actividade onde se adicionam novos *thresholds* de temperatura. No caso de não existirem alarmes a mensagem de *Current alarms* aparece vazia. No caso da figura, um alarme para vinte e quatro graus já estava definido. Ao pedir os valores armazenados na *cloud*, são indicados os valores ainda não lidos e são feitas notificações para os valores que ultrapassam os *thresholds*.

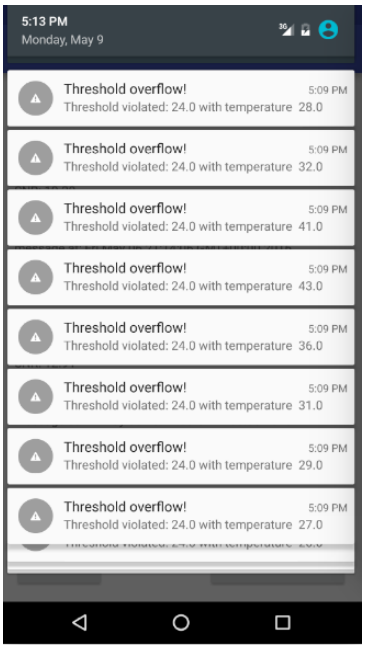

Figura 15: Lançamento das notificações

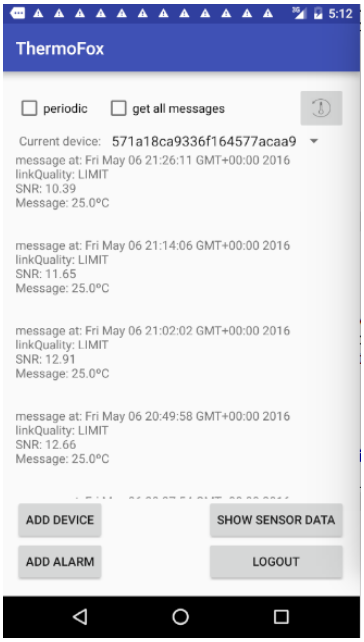

Figura 16: Lançamento das notificações dentro da *app*

Ao realizar o *logout*, a aplicação voltará à actividade inicial, pelo que se se colocar o *username* e *password* anteriores será um processo equivalente a ligar a aplicação pela segunda vez como se pode ver de seguida:

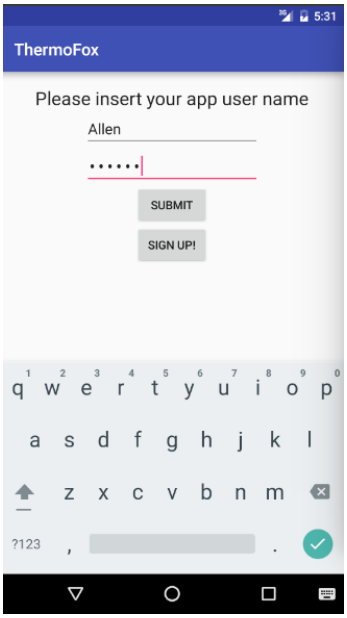

Figura 17: Retorno à actividade inicial após *logout*

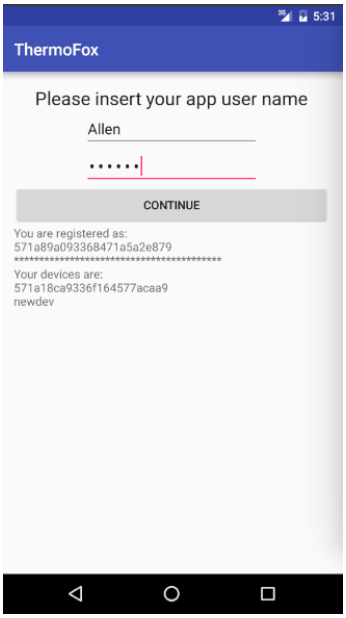

Figura 18: Informações actuais

4.2 Database

Ao realizar as acções anteriores, a base de dados irá ser atualizada em conformidade, por intermédio dos *requests* da aplicação ao servidor *web*. Desta forma, a *database* que numa fase inicial tinha as tabelas vazias (*empty set*), sofrerá as seguintes alterações:

- Ao realizar a acção da figura 9, a tabela *users* terá as seguintes entradas:

```
mysql> select * from users;
+----------+--------------------------------------------------------------+----------------------+----------------------------------+-----------------+
| filename | filePass                                                     | userName             | password                         | lastMessageTime |
+----------+--------------------------------------------------------------+----------------------+----------------------------------+-----------------+
| Allen    | $2y$10$Y9Ou2jX87h.CiGNT7yl.9ewcgUhTmtxBUeUf5YZsl1lmDYiwbysC. | 571a89a093368471a5a2e879 | e5c2bfa6b523d9028d666f9f1f6ae4de | 0               |
+----------+--------------------------------------------------------------+----------------------+----------------------------------+-----------------+
1 row in set (0.00 sec)
```

Figura 19: Entrada correspondente ao registo de um utilizador

- Ao adicionar um alarme (acção da figura 14), irá ser alterada a tabela *usersAlarms*, como se pode observar de seguida.

```
mysql> select * from usersAlarms;
+----------+-------+
| filename | alarm |
+----------+-------+
| Allen    | 24    |
+----------+-------+
1 row in set (0.00 sec)
```

Figura 20: Registo de um novo *threshold* de alarme

- Ao fazer um pedido de informação do sensor, é alterado o tempo correspondente à última leitura, de forma a que não sejam novamente enviadas sem o consenso do utilizador (opção *get all messages*).

```
mysql> select * from users;
+----------+--------------------------------------------------------------+----------------------+----------------------------------+-----------------+
| filename | filePass                                                     | userName             | password                         | lastMessageTime |
+----------+--------------------------------------------------------------+----------------------+----------------------------------+-----------------+
| Allen    | $2y$10$Y9Ou2jX87h.CiGNT7yl.9ewcgUhTmtxBUeUf5YZsl1lmDYiwbysC. | 571a89a093368471a5a2e879 | e5c2bfa6b523d9028d666f9f1f6ae4de | 1462569972      |
+----------+--------------------------------------------------------------+----------------------+----------------------------------+-----------------+
1 row in set (0.00 sec)
```

Figura 21: Alteração do valor de *lastMessageTime*

- Quando é adicionado um novo dispositivo, este será adicionado à base de dados, tal como esperado.

```
mysql> select * from usersDevices;
+----------+--------------------------+
| filename | device                   |
+----------+--------------------------+
| Allen    | 571a18ca9336f164577acaa9 |
| Allen    | newdevice                |
+----------+--------------------------+
2 rows in set (0.00 sec)
```

Figura 22: Adição de um novo dispositivo

4.3 Considerações Finais

Relativamente ao planeamento intermédio, a implementação da base de dados surgiu como um pretexto de tornar a aplicação escalável de modo a que mais do que um utilizador a pudesse utilizar. A inserção de novos dispositivos esteve sempre dependente da informação presente na conta *SigFox*, pelo que se teve de separar o registo de um utilizador no seio da aplicação e o registo de um utilizador do serviço prestado pela *Sigfox*.

Assim não foi possível testar o correcto funcionamento de um utilizador com mais de dispositivo registado na conta *SigFox*, uma vez que cada utilizador registado possuía apenas 1 dispositivo associado. Para dispositivos não existentes/associados à conta *SigFox*, existe o correcto funcionamento da aplicação, não apresentando qualquer mensagem associado a este suposto dispositivo, pelo que se assume o normal funcionamento caso houvesse realmente dois dispositivos associados.

Por fim, em termos de apresentação de forma *user friendly*, este objectivo assume-se assegurado, porém a existência de uma interface gráfica representativa da subida ou descida de temperatura, ou indicativa da qualidade de ligação podia ter sido uma forma de aumentar o nível de fluidez e clareza da informação disponibilizada ao utilizador.

Referências

[1] [FILE16] http://developer.android.com/reference/java/io/File.html, Março 2016

[2] [THREAD16] http://developer.android.com/reference/java/lang/Thread.html, Março 2016

[3] [HTTPURLCONNECTION16]
http://developer.android.com/reference/java/net/HttpURLConnection.html, Março 2016

[4] [URLCONNECTION16]
http://developer.android.com/reference/java/net/URLConnection.html, Março 2016

[5] [JSON16] http://developer.android.com/reference/android/util/JsonReader.html, Março 2016

[6] [NOTIFICATION16]
http://developer.android.com/training/notify-user/build-notification.html, Março 2016

[7] [AKERU16] https://github.com/Snootlab/Akeru, Fevereiro 2016

[8] [DHT1110] http://www.micropik.com/PDF/dht11.pdf, Julho 2010

[9] [DHT1115]
https://github.com/nicolsc/connected-temperature-sensor/blob/master/connected-temperature-sensor.ino, Fevereiro 2015