

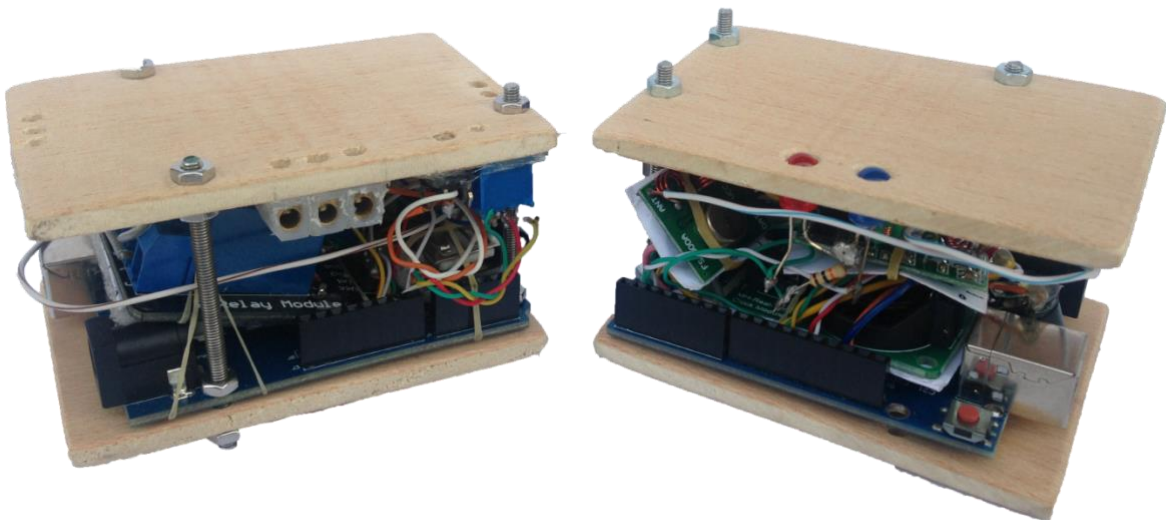


**DEEC**

DEPARTAMENTO DE ENGENHARIA  
ELETROTÉCNICA E DE COMPUTADORES

**TÉCNICO LISBOA**

# Domus Sapiens



15 de setembro de 2018

BitChallenge

Leonardo Pedroso Duarte

## Índice

<b>Motivação .....</b>	<b>3</b>
<b>Introdução .....</b>	<b>3</b>
<b>Modo de Funcionamento.....</b>	<b>4</b>
<b>Dispositivo central.....</b>	<b>4</b>
Funcionamento global .....	4
Gestão de armazenamento de dados.....	6
<b>Dispositivo secundário .....</b>	<b>6</b>
Funcionamento global .....	6
<b>Comunicação entre dispositivos .....</b>	<b>8</b>
Codificação de comunicação .....	8
Segurança .....	9
Protocolo de emparelhamento .....	9
<b>Interação com o utilizador .....</b>	<b>10</b>
Interface gráfica .....	11
Codificação de comunicação .....	11
<b>Problemas/Melhoramentos .....</b>	<b>12</b>
<b>Código.....</b>	<b>13</b>
<b>Dispositivo Central .....</b>	<b>13</b>
<b>Dispositivos Secundários.....</b>	<b>46</b>
<b>Aplicação .....</b>	<b>66</b>
<b>Esquemas Elétricos .....</b>	<b>92</b>
<b>Dispositivo Central .....</b>	<b>92</b>
<b>Dispositivos Secundários.....</b>	<b>93</b>
<b>Avaliação Curricular .....</b>	<b>94</b>
<b>Material utilizado e respetivos custos.....</b>	<b>95</b>
<b>Anexos .....</b>	<b>96</b>

## Motivação

Quando fui confrontado com o desafio de formular um projeto nesta categoria, a primeira ideia que me assaltou foi imediatamente a da domótica e de como automatizar algumas componentes da minha casa, algo em que me seduzia já algum tempo.

A minha ideia inicial surgiu da minha vontade de controlar remotamente as persianas da minha casa e de as programar com o meu telemóvel, para levantar e baixar a uma determinada hora do dia.... Deparei-me, no entanto, com um problema: apenas uma pequena fração das casas e apartamentos, hoje em dia, tem um sistema de domótica integrado. Para além disso, integrar um sistema desta natureza numa casa que não tenha sido construída nesse sentido, como é o caso da minha, ainda que tudo seja controlado eletricamente, incluindo as persianas, ainda é excessivamente dispendioso. Após alguma pesquisa, descobri alguns dispositivos de empresas bastante conhecidas no ramo da tecnologia que, não sendo muito dispendiosos, conseguiam automatizar apenas *smart switches* e alguns eletrodomésticos. Encontrei também algumas invenções *DIY*, mas estavam todas aquém de uma solução eficaz e facilmente implementável a uma escala maior. Em toda a minha pesquisa não encontrei forma de controlar as persianas elétricas numa casa que não tivesse sido inicialmente projetada com um sistema de domótica de raiz ou em que se instalasse esse sistema sem que a montagem e equipamento fossem caríssimos e tivessem de ser instalados por uma empresa especializada.

Assim, o meu objetivo foi encontrar uma forma de integrar um sistema de domótica para controlo de persianas económico, eficaz, *user-friendly* e que pudesse ser produzido em grande escala.

## Introdução

O sistema que desenvolvi é composto por um dispositivo central principal (DC) e vários outros dispositivos secundários (DS), que estarão associados, cada um, a uma persiana a controlar. Desta forma, este sistema é modular o que lhe confere uma grande versatilidade, podendo-se ir adquirindo dispositivos secundários, ao longo do tempo, e integrá-los de forma muito simples.

O dispositivo central é um paralelepípedo pequeno e, por isso, discreto que foi projetado para ser colocado na casa, por forma a que seja possível ligá-lo a um cabo de rede. Foi desenhado para ser, de preferência, colocado atrás do *router* de *internet*

de uma habitação, onde é fácil estabelecer uma ligação à *internet* sem, ao mesmo tempo, comprometer a estética da casa.

É de salientar o facto de cada um dos dispositivos secundários não estar conectado, fisicamente, ao central, nem a qualquer um dos outros, comunicando entre si através de uma ligação *wireless* de 433MHz. Esta solução permite o controlo, sem a necessidade de instalar ligações físicas, o que careceria de muito cuidado e certamente experiência na sua montagem. Para além disso, simplifica, de forma significativa, a forma de tratar a interação utilizador/dispositivos, através da centralização dessa interação.

Assim, todos os comandos que o utilizador efetuar passam, obrigatoriamente, numa primeira instância, pelo módulo central, que o analisará e transmitirá a informação relativa a essa instrução aos equipamentos secundários em questão. Assim, para além de facilitar essa interação, apenas o equipamento central terá ligação à *internet*, algo que torna a solução para o problema inicialmente identificado mais simples. Esta é uma alternativa inovadora, simples e que serve o propósito previamente idealizado.

Cada um dos equipamentos secundários será desenhado com a dimensão máxima de 80mm x 80mm x 50mm, a dimensão *standard* de uma caixa de derivação na parede onde são feitas as ligações das persianas, permitindo, assim, um controlo discreto e simples.

Finalmente, para facilitar a comunicação entre o utilizador e o sistema de domótica, desenvolvi uma aplicação para *smartphone*, amigável e com uma *interface* gráfica cuidada, que permite ao utilizador definir alarmes para abertura e fecho de cada uma das persianas. Para além de temporizar o alarme, o utilizador pode também definir a respetiva periodicidade, isto é, pode selecioná-la entre as seguintes opções: todos os dias, dias úteis, fim de semana ou desativado. Permite, também, dar instruções de abertura e fecho às persianas a partir da aplicação.

## Modo de Funcionamento

### Dispositivo central

#### Funcionamento global

O dispositivo central é aquele que recebe as instruções do utilizador, através da conexão à *internet*, conseguida pela ligação a um cabo e rede. Para conseguir enviar informação, a partir da aplicação, para este dispositivo e receber *feedback* do sucesso da ação correspondente, optei por configurá-lo como um servidor. Assim, a aplicação

envia um *request* para o *arduino* que interpreta esse *request* e extrai o comando de acordo com uma convenção estabelecida, executa a ação correspondente, enviando, seguidamente, um documento em JSON para a aplicação. Os detalhes desta comunicação são explicados com mais pormenor num subtópico mais adiante.

Assim, é o responsável por tratar das comunicações com os dispositivos secundários, através de comunicação por radiofrequência: ordens de abertura, atualização de informação e de tempo e protocolo de emparelhamento de novos dispositivos secundários.

Está, também, encarregue de guardar toda a informação de cada uma das persianas de forma permanente, em EEPROM, nomeadamente, as horas de abertura, fecho e respetiva frequência. Sempre que é ligado, no processo de iniciação deste equipamento, todas as informações guardadas em EEPROM são enviadas aos equipamentos secundários, bem como a hora exata. Assim, a informação não tem de estar guardada em SRAM durante o funcionamento do sistema, o que seria completamente inexecutável. Na verdade, para além de libertar memória do *Arduíno*, evita o uso de um cartão SD e respetivo módulo de *interface*, o que aumenta a rapidez de *upload* e escrita de dados, mas, por outro lado, torna a gestão dos dados em armazenamento mais complexa, pois requer um algoritmo adaptado a este projeto, por forma a tirar o máximo rendimento da EEPROM. A gestão de memória é um tópico explorado com mais profundidade num subtópico adiante.

Por forma a desempenhar todas estas tarefas, o dispositivo central faz uso de:

- Um *arduino Uno*;
- Um módulo de receção e outro de transmissão de ondas eletromagnéticas de 433MHz de frequência, para permitir a comunicação com os dispositivos secundários;
- Um *step up boost converter* para disponibilizar uma tensão de 12V que permite atingir o maior alcance possível no módulo de transmissão;
- Um *chip* DS1302 e cristal adequado que, alimentado por uma pequena pilha, permite a manutenção das horas após o dispositivo central ter sido desconectado da fonte de alimentação;
- Um *ethernet module* para permitir a ligação à *internet* através de um cabo de rede;
- Dois *LEDs* para indicar o estado dos processos, que ocorrem neste equipamento.

A alimentação deste dispositivo é assegurada por um transformador de tomada 230V AC para 5V DC, ligada à entrada USB-B do Arduino.

### Gestão de armazenamento de dados

Para cada dispositivo é necessário guardar o seu ID único, horas e frequências dos alarmes de subida e descida, o seu nome definido pelo utilizador (uma *string* de, no máximo, 15 caracteres, incluindo o terminador) e o *Shared Secret* (a chave de 128 *bits* usada para a encriptação da comunicação com os dispositivos secundários, tópico que é abordado adiante). No total 39 *bytes*, pelo que, como a EEPROM de um *arduino Uno* tem a capacidade de 2Kb, é possível guardar 52 destes blocos completos na EEPROM.

Assim, desenvolvi um algoritmo que converte o objeto com toda a informação num *array* de *bytes* e que os guarda, na EEPROM, em blocos. O endereço em que os blocos são guardados na EEPROM não é, no entanto, o mais baixo que esteja vazio nem obedece a uma ordenação. De facto, de modo a aumentar a longevidade da EEPROM, desenvolvi um algoritmo simples de *ware-levelling*. Assim, de cada vez que é guardada informação na EEPROM, esse endereço fica guardado numa variável global e, da próxima vez que for guardado um dispositivo, o algoritmo coloca-o no bloco livre seguinte, pelo que todos os endereços serão, eventualmente, usados, uniformizando o número de ciclos *read/write* em cada bloco. O primeiro bloco, onde é guardado um dispositivo, é escolhido aleatoriamente. Para esta aplicação não era absolutamente necessário usar um algoritmo de *ware-levelling*, no entanto, não sendo computacionalmente exigente, e altamente pedagógico, decidi implementá-lo.

### Dispositivo secundário

#### Funcionamento global

Cada dispositivo secundário está associado a uma persiana ~~e tem como~~ cuja função principal é gerir a sua abertura e fecho. Este módulo recebe ordens do utilizador de duas formas: através dos interruptores, a que anteriormente a persiana estava ligada, e através da aplicação.

Em primeiro lugar, os dois interruptores, que anteriormente selecionavam a qual dos dois cabos do motor ~~é que~~ era feita a ligação da fase, alternando o sentido do motor, agora, estão apenas ligados ao *arduino*. Assim, nesta configuração os interruptores

selecionam um dos dois pinos *I/O*, configurados como *PULLUP*, ligando-o a *GND*. Desta forma, o controlo manual das persinas não é perdido.

Para que, ao alternar entre estados da persina, no controlo manual, o comportamento da persiana não seja errático, implementei um filtro passa baixo muito simples, além de que bastante eficaz. Na verdade, o *arduino* recolhe várias amostras, ao longo do tempo, dos *inputs* do interruptor, fazendo um balanço entre as amostras, eliminando frequências muito elevadas, muito comuns em mudanças de estado, e facilmente visíveis num osciloscópio.

Em segundo lugar, os comandos vindos da aplicação, que passam, numa primeira instância, pelo dispositivo central, podem ser ordens de abertura/fecho, atualização de alarmes e *pairing*. Recebe também atualizações da hora automáticas. Este dispositivo responde a todas as mensagens que recebe com uma mensagem de confirmação de receção, de modo a garantir o sucesso da comunicação ao utilizador, conferindo uma grande fiabilidade ao sistema.

A alimentação deste dispositivo é feita através de um transformador de 230V AC para 5V DC, embebido no módulo.

Desta forma, a este módulo têm de ser feitas 8 ligações ao módulo dentro da caixa de derivação da persiana:

- Fase e neutro que alimentam o módulo e o motor;
- 3 ligações ao interruptor nomeadamente, *GND* e aos dos pinos *I/O* com *PULLUP*;
- 3 ligações ao motor, fases em ambos os sentidos de rotação e neutro.

É de notar que a ligação à terra do motor é feita externamente ao módulo.

Assim, este dispositivo é constituído por:

- um *arduino UNO*;
- um relé de dois canais;
- um transformador de 230V AC para 5V DC;
- Um módulo de receção e outro de transmissão de ondas eletromagnéticas de 433MHz de frequência, para permitir a comunicação com os dispositivos secundários;
- Um *step up boost converter* para disponibilizar uma tensão de 12V, que permite atingir o maior alcance possível no módulo de transmissão;
- Uma resistência variável por forma a controlar o tempo de abertura;
- Um *push button* para iniciar o protocolo de emparelhamento.

Apenas um dos dois cabos do motor pode estar ligado à fase. Se ligados simultaneamente poderão ser induzidos estragos permanentes no motor. Na configuração de raiz da persiana, esta condição era garantida por um impedimento mecânico no interruptor. No entanto, como agora o controlo de motor é executado através de um microcontrolador, foi necessário ter particular atenção a este detalhe, garantindo que só se podia ativar um sentido, se o outro estivesse desativado, ainda que a ordem ativação tivesse origens distintas.

Outra característica que este sistema tem é a possibilidade de alterar o tempo de subida/descida de cada uma das persianas. É apenas necessário alterar o valor original, para persianas mais exóticas, ou se se quiser abrir metade da persiana de cada vez. Este ajuste é feito no próprio dispositivo secundário através de uma resistência variável, que começa com um valor predefinido, adequado à grande maioria das persianas, e calibrado, de modo a que cada rotação completa do parafuso da resistência variável corresponda a um aumento (ou decréscimo, dependendo do sentido) de um tempo bem definido, cerca de 7 segundos por rotação completa.

Para além disso, ainda que não constitua um constrangimento o facto de o interruptor ficar na posição de subida ou descida, durante muito tempo, uma vez que o circuito do motor tem um sistema de proteção, o microcontrolador cessa a ordem de abertura/fecho ao motor, depois de passado o tempo definido pelo utilizador.

## Comunicação entre dispositivos

### Codificação de comunicação

Toda a comunicação entre dispositivos e troca de informação, através de radiofrequência, é feita por envio de cadeias de *bytes*, de acordo com uma codificação por mim criada, por forma a que as mensagens sejam fáceis de interpretar e o mais compactas possível.

Assim, a codificação que usei na comunicação DC-DS foi a seguinte: os primeiros 5 caracteres correspondem ao ID do DS, a que se destina a mensagem. De seguida, um caractere indica o intuito da mensagem. Os restantes caracteres são dependentes da finalidade da mensagem e muito variados (cf. código DC). Uma exceção é o envio de códigos de emparelhamento, em que o ID não consta da mensagem, uma vez que dois DS não podem emparelhar simultaneamente e, por esta razão, apenas um está à espera de receber uma mensagem com essa informação.



Já no que diz respeito à codificação, que implementei na comunicação DS-DC, o primeiro caractere está relacionado com o intuito da mensagem e os restantes, se existentes, completam a sua finalidade.

Os caracteres usados e forma de construir cada uma das diferentes mensagens estão detalhadamente explicados, nos comentários do código de ambos os dispositivos.

### Segurança

Uma vez que é muito fácil intercetar e emitir mensagens na frequência usada, decidi implementar um sistema de encriptação da comunicação entre dispositivos.

Assim, como usar uma chave predefinida é demasiado inseguro, optei por usar uma chave diferente para encriptar as comunicações com cada um dos dispositivos secundários, uma chave de 128 *bits*, que é criada, aquando da primeira comunicação com o DS, pelo que é impossível sabê-la *a priori*. Desta forma, é necessário que o DS faça um *handshake* com o DC, de modo a concordarem na chave que vão usar na comunicação entre si.

Decidi usar encriptação AES com uma chave de 128 *bits*, que é conseguida por uma biblioteca, desenvolvida por terceiros, creditados nos comentários do código.

### Protocolo de emparelhamento

Para conseguir comunicar com um dispositivo secundário, a partir da aplicação, é necessário, numa primeira instância, emparelhar os dispositivos de acordo com um protocolo por mim definido e que se fundamenta na troca pública de chaves, segundo o algoritmo de *Diffie-Hellman*. Este protocolo tem como principal objetivo a troca da chave comum de 128 *bits*, segundo a qual a comunicação entre ambos será encriptada e guardar o ID do DS no DC.

A ordem de emparelhamento é dada através do utilizador na aplicação. Nesse momento, o dispositivo principal começa a tentar intercetar um *ping* de um DS com uma mensagem de emparelhamento que contém o seu ID. Assim, para iniciar o protocolo, o utilizador tem que pressionar o *push button* do dispositivo secundário, que inicia o *ping*. Enquanto envia o *ping*, o DS está à espera que o dispositivo principal o intercete e lhe envie o primeiro código, a segunda fase do *pairing*.

Esta segunda fase consiste nos seguintes procedimentos, ordenados e executados imediatamente a seguir ao DP ter intercetado um *ping* de um DS:

- O DP cria a sua chave privada, um número aleatório de 128 bits,  $A$ ;
- O DP gera e envia a sua chave pública,  $A^* = G^A \bmod P$ ;

- O DS recebe a chave pública do DP e gera a sua chave privada, um outro número aleatório de 128 bits,  $B$ ;
- O DS gera e envia a sua chave pública,  $B^* = G^B \bmod P$ ;
- O DP gera o *shared secret*,  $K = G^{B^*} \bmod P$ ;
- O DS gera o *shared secret*,  $K = G^{A^*} \bmod P$ ;
- O DP envia um comando de teste para o DS, já encriptado com a chave obtida, o que assegura a troca bem-sucedida da chave, alertando o utilizador caso contrário.

É de notar que os valores de  $G$  e de  $P$  são predefinidos.

### Interação com o utilizador

De modo a facilitar a comunicação com o utilizador, desenvolvi uma aplicação para IOS (sistema operativo em que consigo depurar erros e testar o sistema sem recorrer a emuladores) com uma interface gráfica cuidada. Usei o *Xcode IDE* para desenvolver a aplicação e recorri, principalmente, a linguagem *Objective-C* e pequenos scripts de *Swift*.

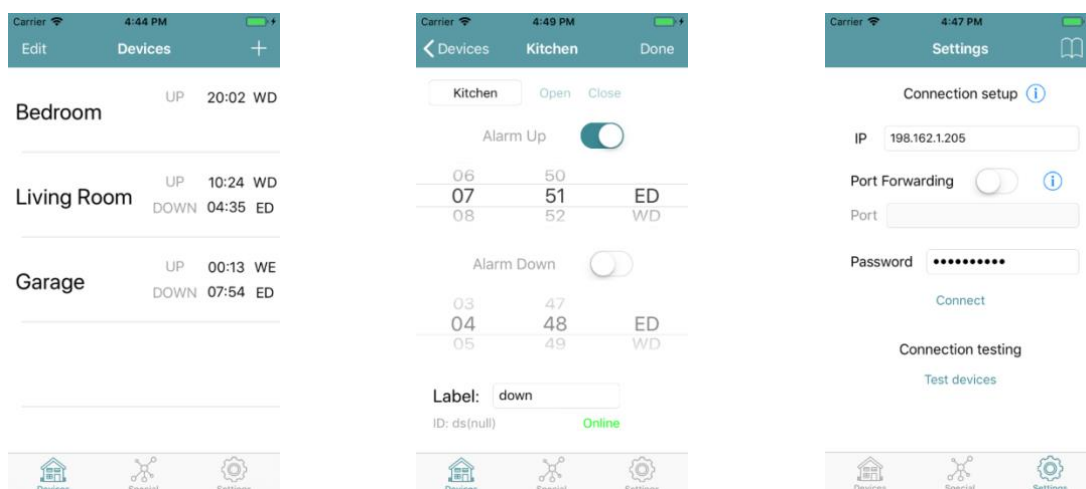
Esta aplicação comunica diretamente com o dispositivo central, permitindo, assim, abrir instantaneamente qualquer uma das persianas emparelhadas com o DC, definir todos os alarmes de abertura e fecho e emparelhar novos dispositivos.

Uma mais-valia da aplicação é também a possibilidade de definir os nomes para cada uma das persianas, bem como as suas etiquetas. Estas etiquetas são guardadas na memória do telemóvel, permitindo libertar a EEPROM do DC de uma grande quantidade de dados. Uma das vantagens desta configuração é que as etiquetas de cada persiana podem ser diferentes de utilizador para utilizador, uma vez que é a aplicação que associa o ID do dispositivo a uma etiqueta guardada, na memória de cada telemóvel.

Outra vantagem é a capacidade de ordenar as persianas, sendo possível deslocar para o topo da lista as mais usadas. A ordem escolhida é, também, armazenada no telemóvel e, por isso, é diferente para cada utilizador.

## Interface gráfica

Seguem-se exemplos da interface gráfica da aplicação:



## Codificação de comunicação

Para conseguir a comunicação entre a aplicação e o DC, configurei o *arduino* como um servidor. Assim, a aplicação envia um *request* para o *arduino*, de acordo com uma codificação por mim convencionada, executa a ação correspondente e responde com um documento em JSON.

A aplicação pode ser usada na rede local ou com qualquer ligação à *internet*, desde que seja possível configurar o *router* a que o *arduino* está ligado com um endereço de IP estático. É possível alternar entre estas duas configurações nas definições da aplicação, onde é possível introduzir todos os endereços necessários.

A segurança deste canal de comunicação está já assegurada pela ligação *wifi* do telemóvel, quando numa rede local, mas não é tão segura, quando funciona através de *port forwarding* do *router*, pelo que cada DC tem uma *password* predefinida, que tem de ser introduzida nas definições da aplicação.

Cada *request* feito pela aplicação obedece a uma codificação por mim estabelecida: os primeiros 10 caracteres constituem a *password* do DC, com que estamos a tentar comunicar, de seguida o intuito do *request* codificado por um único caractere e, se necessário, informação complementar seguidamente. O facto de as respostas serem dadas em JSON pelo *arduino* faz com que seja muito fácil converter a informação recebida num dicionário de dicionários, a forma mais vantajosa de trabalhar numa aplicação deste género e numa linguagem tão alto-nível como Objective-C.

## Problemas/Melhoramentos

No desenvolvimento deste projeto, deparei-me com vários problemas, para os quais, na maior parte das situações consegui descobrir uma solução. No entanto, quer por condicionantes do projeto, quer por falta de tempo, há ainda alguns aspetos a melhorar. Assim, este projeto deve ser visto, a meu ver, como uma prova de conceito e não como um produto acabado, porque, ainda que completamente funcional, há espaço para melhorias. Pelo que continuará, após a entrega para o concurso, a ser maturado, até que o considere uma boa primeira versão para implementar em minha casa.

Em primeiro lugar, as ligações elétricas dos dispositivos não só são relativamente frágeis, como também levam muito tempo a completar. Assim, penso que tendo já um modelo funcional, o próximo passo é desenhar uma PCB, onde se possam soldar os vários chips necessários, incluindo um *ATmega* em vez da placa *arduino*, algo em que já comecei a trabalhar. Assim, ainda que os modelos de DC e DS sejam de dimensões dentro dos limites estabelecidos à partida, é possível diminuir de forma significativa o espaço ocupado por eles com esta nova alteração.

Em segundo lugar, os módulos de *rf* usados, quando testados individualmente conseguem comunicar a sensivelmente 75 metros através de duas paredes. No entanto, quando associados ao resto do projeto numa caixa, o alcance cai por cerca de um fator de 10, não sendo por isso viável para uma grande parte das aplicações que tinha em mente. Após pesquisar, encontrei um módulo alternativo que me parece adequado, mas o seu custo é superior e não a conseguiria implementar no projeto a tempo, dado o longo período de espera entre a compra e a entrega do mesmo. No entanto, o protocolo de comunicação seria muito fácil de alterar, para comunicar com os novos módulos, pelo que não requer alterações significativas no código.

Finalmente, a aplicação carece ainda de mais algum cuidado, mas para isso é necessário aprender bastante acerca do *IDE* e da linguagem *Objective-C*, visto que o primeiro contacto que tive com eles foi a propósito deste projeto e, por isso, me falta experiência, nomeadamente no que diz respeito a *exception handling*. Para além disso, tinha também em mente, na aba *special* da aplicação, permitir que o utilizador definisse alarmes para as persianas, com base na sua etiqueta.

## Código do Dispositivo Principal

Fcheiros:

1. DomusSapiens.ino
2. DeviceData.h
3. DeviceData.cpp
4. StorageManagement.h
5. StorageManagement.cpp
6. deviceCommunication.h
7. deviceCommunication.cpp
8. DiffieHellman.h
9. DiffieHelman.cpp
10. timeHandling.h
11. timeHandling.cpp
12. MACROS\_LIGACAO.h
13. user\_communication.h
14. user\_communication.cpp

Todos os ficheiros podem ser consultados em:

[github.com/LeonardoPedroso/DomusSapiens](https://github.com/LeonardoPedroso/DomusSapiens)

```

1 //Incluir funções necessárias de outros ficheiros
2 #include "user_communication.h"
3 #include "timeHandling.h"
4 #include "deviceCommunication.h"
5 #include "DeviceData.h"
6 using namespace deviceCommunication;
7
8 //variaveis globais
9 byte lastAddedBlockAddr;
10
11
12 void setup()
13 {
14
15     //Para depuração de erros
16     #ifndef NO_DEBUG
17     Serial.begin(230400);
18     pinMode(4, INPUT_PULLUP);
19     #endif
20
21
22     //Tenta ligação à internet
23     while (!setUpConnection()){
24         #ifndef NO_DEBUG
25         Serial.println(F("Connection failed"));
26         #endif
27         delay(1000);
28     }
29
30     //inicia o Real Time Clock Module
31     setUpRTC();
32     if (!updateRTC()) {
33         #ifndef NO_DEBUG
34         Serial.println(F("Clock error"));
35         #endif
36     }
37
38     deviceCommunication::initDeviceCommunication(); //inicia a comunicação por rf
39     delay(1000);
40     deviceCommunication::setUpDevices(); //atualizar a informação dos dispositivos
41
42 }
43
44 void loop()
45 {
46     delay(100);
47     char * cmd = checkReception(); //verifica a receção de um pedido
48     if(cmd)
49         analyseCommand(cmd); //analisa e executa o pedido recebido
50     checkNeedToUpdateTime(); //verifica a necessidade de atualizar as horas
51 }

```

```

1  /*
2  * Ficheiro DeviceData.h
3  * Header que serve de suporte ao ficheiro "DeviceData.cpp"
4  * Está neste ficheiro definida a classe DeviceData usada para a gestão mais
5  * organizada e eficiente da interação com os dispositivos secundários
6  */
7
8  #ifndef DEVICE_DATA_H
9  #define DEVICE_DATA_H
10
11 //----- INCLUDES -----
12 #include "Arduino.h"
13 #include "MACROS_LIGACAO.h"
14 #include <AES.h>
15 /*Autores: Brian Gladman
16             Karl Malbrain
17             Mark Tillotson
18 */
19 //----- MACROS -----
20 #define AES_KEY_LENGTH 128
21
22 //defines para os construtor da classe
23 #define ID_ONLY 1
24 #define READ_EEPROM 2
25
26 //----- CLASSE DeviceData -----
27 class DeviceData
28 {
29 public:
30     //constructors
31     DeviceData(char * json);
32     DeviceData(char * json, const byte mode);
33     DeviceData(unsigned int id);
34
35     //gestão da informação na EEPROM
36     byte removeFromEeprom(); //remove info from EEPROM
37     byte saveOnEeprom(); //save to EEPROM
38     byte getDeviceFromEeprom(byte deviceN); //upload device
39     byte getSharedSecretFromEeprom();
40
41     //pairing
42     byte getParingDevice();
43     //Extração da informação da string em formato JSON
44     void getJsonString(char * json);
45     void getJsonStringParing(char * json);
46
47     //device commands
48     byte openCmd();
49     byte closeCmd();
50     byte test();
51     byte sendUpdatedInfo();
52     byte sendDeactivation();
53     byte sendTime();
54
55     //debugging
56     #ifdef DEBUG
57     void show();
58     #endif
59
60     //variáveis públicas
61     unsigned int id;
62     byte hourAlarmUp;
63     byte sharedSecret[16];
64
65 private: //variáveis definidas como privadas e que integram
66     char nameDev[15];
67     byte minuteAlarmUp;
68     char freqUp;
69     byte hourAlarmDw;
70     byte minuteAlarmDw;
71     char freqDw;
72 };
73
74 #endif

```

```

1  /*
2  * Ficheiro DeviceData.cpp
3  * Este ficheiro é o que contem as funções que regulam a comunicação com os vários dispositivos
4  * secundários e gerem a informação guardada, para isso usa as funções dos ficheiros
5  * "StorageManagement.h" e "deviceCommunication.h".
6  * Define também a classe DeviceData que permite fazer esta gestão de forma mais organizada e eficiente
7  */
8
9  //----- INCLUDES -----
10 #include "DeviceData.h"
11 #include "StorageManagement.h"
12 #include "deviceCommunication.h"
13
14 using namespace StorageManagement;
15 using namespace deviceCommunication;
16
17 //instância da classe AES que permite encryptar a comunicação com os dispositivos secundários
18 AES aes;
19
20 //----- FUNÇÕES -----
21
22 //----- CONSTRUCTORS -----
23 //Constructor que instancia um objeto a partir de uma string em formato JSON
24 DeviceData::DeviceData(char * json)
25 {
26     if (json){
27
28         //extraí o id
29         unsigned int id = 0;
30         for (byte i = 1; i <= 5; i++)
31             id += ((*json + i) - '0') * StorageManagement::power(10, 5 - i);
32         this->id = id;
33         json += 6;
34
35         //extraí o nome
36         byte i;
37         for (i = 0; ; i++) {
38             if ((*json + i) <= '9' && *(json + i) != SPACE_MARKER) {
39                 this->nameDev[i] = '\0';
40                 break;
41             } else
42                 this->nameDev[i] = *(json + i);
43         }
44
45         for (byte j = i + 1; j < 15; j++)
46             this->nameDev[j] = '\0';
47         json += i;
48
49         //extraí o alarm up
50         this->hourAlarmUp = ((*json) - '0') * 10 + ((*json + 1) - '0');
51         this->minuteAlarmUp = ((*json + 2) - '0') * 10 + ((*json + 3) - '0');
52         this->freqUp = *(json + 4);
53
54         //extraí o alarm dw
55         this->hourAlarmDw = 10 * ((*json + 5) - '0') + ((*json + 6) - '0');
56         this->minuteAlarmDw = 10 * ((*json + 7) - '0') + ((*json + 8) - '0');
57         this->freqDw = *(json + 9);
58         json += 10;
59     }
60 }
61
62 }
63

```



```

64 //Constructor alternativo
65 DeviceData::DeviceData(char * json, const byte mode)
66 {
67     switch (mode) {
68         case ID_ONLY: {
69             unsigned int id = 0;
70             for (byte i = 1; i <= 5; i++)
71                 id += ((long)(*(json + i) - '0')) * StorageManagement::power(10, 5 - i);
72             this->id = id;
73             break;
74         }
75     }
76 }
77
78 //Constructor que instancia um objeto apenas com um id fornecido com argumento
79 DeviceData::DeviceData(unsigned int id)
80 {
81     this->id = id;
82 }
83
84
85 //-----GESTÃO DE DADOS GUARDADOS -----
86 //Função que faz upload
87 byte DeviceData::getDeviceFromEeprom(byte deviceN)
88 {
89     return StorageManagement::readFromEEPROM(this, deviceN);
90 }
91
92 //Função que guarda (ou atualiza) a informação de um dispositivo em EEPROM
93 byte DeviceData::saveOnEeprom()
94 {
95     #ifndef NO_DEBUG
96         Serial.println(F("Saving"));
97         Serial.flush();
98     #endif
99     return StorageManagement::saveOnEEPROM(this);
100 }
101
102 //Função que elimina informação de um dispositivo em EEPROM
103 byte DeviceData::removeFromEeprom()
104 {
105     #ifndef NO_DEBUG
106         Serial.println(F("Removing from EEPROM"));
107         Serial.flush();
108     #endif
109     return StorageManagement::deleteFromEEPROM(this);
110 }
111
112 //Função que faz apenas o upload do sharedSecret da EEPROM
113 byte DeviceData::getSharedSecretFromEeprom()
114 {
115     return StorageManagement::getSharedSecretFromEeprom(this);
116 }
117
118

```

```

119 //-----JSON STRINGS -----
120
121 //Função que recebe um apontador para uma string na qual vai escrever toda a informação do dispositivo
122 //no formato JSON
123 void DeviceData::getJSONString(char * json)
124 {
125     //Nome
126     *json = '{';
127     *(json + 1) = STRING_DELIMITATOR;
128     *(json + 2) = NAME;
129     *(json + 3) = STRING_DELIMITATOR;
130     *(json + 4) = TWO_DOTS;
131     *(json + 5) = STRING_DELIMITATOR;
132     json += 6;
133     byte i;
134     for (i = 0; ; i++) {
135         if (*(this->nameDev + i) == '\\0')
136             break;
137         else
138             *(json + i) = *(this->nameDev + i);
139     }
140     *(json + i) = STRING_DELIMITATOR;
141     *(json + i + 1) = COMA;
142     json += i + 1;
143
144     //ID
145     *(json + 1) = STRING_DELIMITATOR;
146     *(json + 2) = ID;
147     *(json + 3) = STRING_DELIMITATOR;
148     *(json + 4) = TWO_DOTS;
149     *(json + 5) = STRING_DELIMITATOR;
150     sprintf(json + 6, "%05u", this->id);
151     *(json + 11) = STRING_DELIMITATOR;
152     *(json + 12) = COMA;
153     json += 12;
154
155     //Alarm UP
156     *(json + 1) = STRING_DELIMITATOR;
157     *(json + 2) = HOUR_ALARM_UP;
158     *(json + 3) = STRING_DELIMITATOR;
159     *(json + 4) = TWO_DOTS;
160     sprintf(json + 5, "%d", this->hourAlarmUp);
161     if (this->hourAlarmUp < 10)
162         json += 6;
163     else
164         json += 7;
165     *json = COMA;
166
167     *(json + 1) = STRING_DELIMITATOR;
168     *(json + 2) = MINUTE_ALARM_UP;
169     *(json + 3) = STRING_DELIMITATOR;
170     *(json + 4) = TWO_DOTS;
171     sprintf(json + 5, "%d", this->minuteAlarmUp);
172     if (this->minuteAlarmUp < 10)
173         json += 6;
174     else
175         json += 7;
176     *json = COMA;
177
178

```

```

179 //Alarm Down
180 *(json + 1) = STRING_DELIMITATOR;
181 *(json + 2) = HOUR_ALARM_DOWN;
182 *(json + 3) = STRING_DELIMITATOR;
183 *(json + 4) = TWO_DOTS;
184 sprintf(json + 5, "%d", this->hourAlarmDw);
185 if (this->hourAlarmDw < 10)
186     json += 6;
187 else
188     json += 7;
189 *json = COMA;
190
191 *(json + 1) = STRING_DELIMITATOR;
192 *(json + 2) = MINUTE_ALARM_DOWN;
193 *(json + 3) = STRING_DELIMITATOR;
194 *(json + 4) = TWO_DOTS;
195 sprintf(json + 5, "%d", this->minuteAlarmDw);
196 if (this->minuteAlarmDw < 10)
197     json += 6;
198 else
199     json += 7;
200 *json = COMA;
201
202 //Alarm Frequency UP
203 *(json + 1) = STRING_DELIMITATOR;
204 *(json + 2) = FREQ_ALARM_UP;
205 *(json + 3) = STRING_DELIMITATOR;
206 *(json + 4) = TWO_DOTS;
207 *(json + 5) = STRING_DELIMITATOR;
208 *(json + 6) = this->freqUp;
209 *(json + 7) = STRING_DELIMITATOR;
210 json += 8;
211 *json = COMA;
212
213 //Alarm Frequency Down
214 *(json + 1) = STRING_DELIMITATOR;
215 *(json + 2) = FREQ_ALARM_DOWN;
216 *(json + 3) = STRING_DELIMITATOR;
217 *(json + 4) = TWO_DOTS;
218 *(json + 5) = STRING_DELIMITATOR;
219 *(json + 6) = this->freqDw;
220 *(json + 7) = STRING_DELIMITATOR;
221 *(json + 8) = '>';
222 *(json + 9) = '\0';
223
224 }
225
226 //Função que cria a string em formato JSON por forma a transmitir para a aplicação o
227 //ID do dispositivo que acabou de ser emparelhado
228 void DeviceData::getJSONStringParing(char * json)
229 {
230     *json = '{';
231     *(json + 1) = STRING_DELIMITATOR;
232     *(json + 2) = PARING_DEVICE;
233     *(json + 3) = STRING_DELIMITATOR;
234     *(json + 4) = TWO_DOTS;
235     *(json + 5) = STRING_DELIMITATOR;
236     sprintf(json + 6, "%05u", this->id);
237     *(json + 11) = STRING_DELIMITATOR;
238     *(json + 12) = '>';
239     *(json + 13) = '\0';
240 }
241
242

```

```

243 //----- PAIRING -----
244
245 //Função que procura um dispositivo com que fazer pairing e o guarda em memória
246 //Retorn TRUE se o pairing é bem sucedido e FALSE caso contrário
247 byte DeviceData::getPairingDevice() {
248
249     #ifdef DEBUG
250     Serial.println(F("PAIRING"));
251     #endif
252
253     if (deviceCommunication::handshakeProtocol(this)) {
254         //remove da memória um dispositivo com o mesmo id para garantir a inexistência de definições duplas
255         this->removeFromEeprom();
256         this->hourAlarmUp = 61; //indicador de falta de informação dada pelo utilizador
257         this->saveOnEeprom();
258         return TRUE;
259     }
260     return FALSE;
261 }
262
263
264 //----- COMMANDS -----
265 //Função que envia o comando de abertura para o dispositivo secundário
266 //Este comando está encriptado
267 //Retorna TRUE se o destino recebeu com sucesso a mensagem e FALSE caso contrário
268 byte DeviceData::openCmd()
269 {
270     #ifdef DEBUG
271     Serial.println(F("Open"));
272     Serial.flush();
273     #endif
274
275     //Criação da string que será enviada
276     char openCmd[16];
277     sprintf(openCmd, "%05u", this->id); //id
278     openCmd[5] = OPEN; //identificação do comando
279
280     //geração aleatória dos restantes bytes por forma a prefazer os 16 bytes,
281     //o comprimento standard da mensagem na comunicação com os dispositivos secundários
282     byte i = 0;
283     for (byte k = 6; k < 15; k++) {
284         openCmd[k] = rand() % 256;
285         i += openCmd[k];
286     }
287     openCmd[15] = i; //checksum no final
288
289     aes.set_key (this->sharedSecret, AES_KEY_LENGTH);
290     aes.encrypt ((byte *)openCmd, (byte *)openCmd); //encripta a mensagem com o sharedSecret
291
292     //Envia o comando e aguarda a confirmação de receção
293     if (deviceCommunication::sendByteArray(this, (byte *)openCmd, DEFAULT_BYTE_ARRAY_LENGTH))
294         return TRUE;
295
296     return FALSE;
297 }
298
299

```

```

300 //Função que envia o comando de fecho para o dispositivo secundário,
301 //é muito semelhante à função de abertura, mas por forma a melhorar a leitura
302 //do código optei por separar as funções
303 //Este comando está encriptado
304 //Retorna TRUE se o destino recebeu com sucesso a mensagem e FALSE caso contrário
305 byte DeviceData::closeCmd()
306 {
307     #ifdef DEBUG
308     Serial.println(F("Close"));
309     Serial.flush();
310     #endif
311
312     //Criação da string que será enviada
313     char closeCmd[16];
314     sprintf(closeCmd, "%05u", this->id); //id
315     closeCmd[5] = CLOSE; //identificação do comando
316
317     //geração aleatória dos restantes bytes por forma a prefazer os 16 bytes,
318     //o comprimento standard da mensagem na comunicação com os dispositivos secundários
319     byte i = 0;
320     for (byte k = 6; k < 15; k++) {
321         closeCmd[k] = rand() % 256;
322         i += closeCmd[k];
323     }
324     closeCmd[15] = i;
325
326     aes.set_key (this->sharedSecret, AES_KEY_LENGTH);
327     aes.encrypt ((byte *)closeCmd, (byte *)closeCmd); //encripta a mensagem com o sharedSecret
328
329     //Envia o comando e aguarda a confirmação de receção
330     if (deviceCommunication::sendByteArray(this, (byte *) closeCmd, DEFAULT_BYTE_ARRAY_LENGTH))
331         return TRUE;
332
333     return FALSE;
334 }
335
336 //Função que envia uma atualização de informação para um dispositivo secundário
337 //Retorna TRUE se o destino recebeu com sucesso a mensagem e FALSE caso contrário
338 byte DeviceData::sendUpdatedInfo()
339 {
340     #ifndef NO_DEBUG
341     Serial.println(F("Update device"));
342     Serial.flush();
343     #endif
344
345     char updateCmd[16];
346     sprintf(updateCmd, "%05u", this->id); //id
347     //Informação ordenada por ordem convencional
348     updateCmd[5] = INFO;
349     updateCmd[6] = (char)this->hourAlarmUp;
350     updateCmd[7] = (char)this->minuteAlarmUp;
351     updateCmd[8] = this->freqUp;
352     updateCmd[9] = (char)this->hourAlarmDw;
353     updateCmd[10] = (char)this->minuteAlarmDw;
354     updateCmd[11] = this->freqDw;
355
356     //restantes bytes preenchidos aleatoriamente
357     byte i = 0;
358     for (byte k = 12; k < 15; k++) {
359         updateCmd[k] = rand() % 256;
360         i += updateCmd[k];
361     }
362     updateCmd[15] = i;
363
364     aes.set_key (this->sharedSecret, AES_KEY_LENGTH);
365     aes.encrypt ((byte *)updateCmd, (byte *)updateCmd); //encriptar o comando
366
367     //envia o comando e espera confirmação de receção
368     if (deviceCommunication::sendByteArray(this, (byte *) updateCmd, DEFAULT_BYTE_ARRAY_LENGTH))
369         return TRUE;
370     return FALSE;
371 }

```

```

372
373 //Função que envia comando de desativação do dispositivo secundário
374 //É necessário enviar desativação quando o utilizador elimina um dispositivo secundário da lista
375 //Retorna TRUE se o destino recebeu com sucesso a mensagem e FALSE caso contrário
376 byte DeviceData::sendDeactivation()
377 {
378     #ifdef DEBUG
379     Serial.println(F("Deactivation Sent"));
380     #endif
381
382     char deactCmd[16];
383     sprintf(deactCmd, "%05u", this->id); //id
384     deactCmd[5] = DISABLE; //identificação do comando
385     byte i = 0;
386     for (byte k = 6; k < 15; k++) {
387         deactCmd[k] = rand() % 256;
388         i += deactCmd[k];
389     }
390     deactCmd[15] = i;
391
392     aes.clean();
393     aes.set_key (this->sharedSecret, AES_KEY_LENGTH);
394     aes.encrypt ((byte *)deactCmd, (byte *) deactCmd); // encriptação da mensagem
395
396     //envia o comando e espera resposta
397     if (deviceCommunication::sendByteArray(this, (byte *) deactCmd, DEFAULT_BYTE_ARRAY_LENGTH))
398         return TRUE;
399
400     return FALSE;
401 }
402
403 //Função que verifica se um determinado dispositivo está "online"
404 //Retorna TRUE se o destino recebeu com sucesso a mensagem e FALSE caso contrário
405 byte DeviceData::test()
406 {
407     char testCmd[16];
408     sprintf(testCmd, "%05u", this->id); //id
409     testCmd[5] = TEST; //identificação do comando
410
411     byte i = 0;
412     for (byte k = 6; k < 15; k++) {
413         testCmd[k] = rand() % 256;
414         i += testCmd[k];
415     }
416     testCmd[15] = i; //checksum
417
418     aes.set_key (this->sharedSecret, AES_KEY_LENGTH);
419     aes.encrypt ((byte *)testCmd, (byte *)testCmd); //encriptação do comando
420
421     if (deviceCommunication::sendByteArray(this, (byte *) testCmd, DEFAULT_BYTE_ARRAY_LENGTH))
422         return TRUE;
423     return FALSE;
424 }
425

```

```

426 //Função que envia uma atualização da hora para o dispositivo secundário
427 //É necessário enviar desativação quando o utilizador elimina um dispositivo secundário da lista
428 //Retorna TRUE se o destino recebeu com sucesso a mensagem e FALSE caso contrário
429 byte DeviceData::sendTime()
430 {
431     char timeCmd[16];
432     sprintf(timeCmd, "%05u", this->id); //id
433     timeCmd[5] = TIME; //identificação do comando
434     unsigned long t = now();
435     char * p = (char *) (void *)&t;
436     for (byte i = 0; i < 4; i++)
437         timeCmd[6 + i] = *(p + i);
438     //preencher a restante mensagem aleatoriamente
439     byte i = 0;
440     for (byte k = 10; k < 15; k++) {
441         timeCmd[k] = rand() % 256;
442         i += timeCmd[k];
443     }
444     timeCmd[15] = i;
445
446     #ifdef DEBUG
447     for (byte j = 0; j < 16; j++)
448         Serial.print(timeCmd[j]);
449     Serial.println(F(" "));
450     #endif
451
452     aes.set_key (this->sharedSecret, AES_KEY_LENGTH);
453     aes.encrypt ((byte *)timeCmd, (byte *)timeCmd); //encriptar a mensagem
454
455     //envia a mensagem e aguarda confirmação de receção
456     if (deviceCommunication::sendByteArray(this, (byte *) timeCmd, DEFAULT_BYTE_ARRAY_LENGTH)){
457         #ifdef DEBUG
458         Serial.println(F("Time Sent"));
459         #endif
460         return TRUE;
461     }
462
463     #ifdef DEBUG
464     Serial.println(F("Time update failed"));
465     #endif
466     return FALSE;
467 }
468
469 //----- DEBUG -----
470 #ifdef DEBUG
471 void DeviceData::show()
472 {
473     Serial.print(F("Id: "));
474     Serial.flush();
475     Serial.println(this->id);
476     Serial.flush();
477     Serial.println(this->nameDev);
478     Serial.flush();
479     Serial.println(this->hourAlarmUp);
480     Serial.println(this->minuteAlarmUp);
481     Serial.println(this->freqUp);
482     Serial.println(this->hourAlarmDw);
483     Serial.println(this->minuteAlarmDw);
484     Serial.println(this->freqDw);
485     Serial.flush();
486     for (byte i = 0; i < 16; i++) {
487         Serial.print(*(this->sharedSecret + i), HEX);
488         Serial.print(F(" "));
489     }
490     Serial.println(F(" "));
491     Serial.flush();
492 }
493 #endif

```

```

1 /*Ficheiro StorageManagement.h
2  * Header de suporte ao ficheiro StorageManagement.cpp
3  * A escrita de dados na EEPROM é feita em blocos na EEPROM do tamanho da classe DeviceData
4  * e os endereços de escrita dos blocos de dados são geridos por um algoritmo de ware-levelling
5  * simples que garante uma utilização uniforme da EEPROM, aumentando, desta forma, a sua vida útil.
6  * Para esta aplicação não era absolutamente necessário usar um algoritmo de ware-levelling, no entanto,
7  * como não é computacionalmente exigente e altamente pedagógico decidi implementá-lo
8  */
9
10 #ifndef STORAGE_MANAGEMENT_H
11 #define STORAGE_MANAGEMENT_H
12
13 //----- INCLUDES -----
14 #include "Arduino.h"
15 #include "DeviceData.h"
16 #include "MACROS_LIGACAO.h"
17 #include <EEPROM.h>
18 //Default Arduino library under a Creative Commons Attribution-ShareAlike 3.0 License
19
20 #include <TimeLib.h>
21 /*Autores: Michael Margolis
22          Paul Stoffregen
23 Disponível em: https://github.com/PaulStoffregen/Time
24 */
25 #include "DiffieHellman.h"
26
27 //----- MACROS -----
28 #define BLOCK_SIZE sizeof(DeviceData) //em bytes
29
30
31 //----- VAR GLOBAIS -----
32 //guarda a posição do ultimo bloco em que foi escrita informação
33 extern byte lastAddedBlockAddr;
34
35 namespace StorageManagement
36 {
37
38 unsigned int power(int b, byte e);
39
40 #ifdef DEBUG
41 void showEEPROM();
42 void clearEEPROM();
43 #endif
44
45 void initWithWareLeveling();
46
47 byte saveOnEEPROM(DeviceData *deviceAp);
48 byte deleteFromEEPROM(DeviceData *deviceAp);
49 byte readFromEEPROM(DeviceData * deviceAp, byte deviceN);
50 byte getSharedSecretFromEeprom(DeviceData * deviceAp);
51
52 //private
53 byte EEPROM_writeDevice(unsigned int addr, DeviceData * deviceAp);
54 byte EEPROM_readDevice(unsigned int addr, DeviceData * deviceAp);
55 byte isEmpty(byte blockReal);
56 unsigned int realAddr(byte blockAddr);
57 };
58
59 #endif

```



```

1 /* Ficheiro StorageManagement.cpp
2  * Ficheiro onde estão definidas as funções que regulam a escrita e leitura de dados
3  * na EEPROM
4  * A escrita de dados na EEPROM é feita em blocos na EEPROM do tamanho da classe DeviceData
5  * e os endereços de escrita dos blocos de dados são geridos por um algoritmo de ware-levelling
6  * simples que garante uma utilização uniforme da EEPROM, aumentando, desta forma, a sua vida útil.
7  * Para esta aplicação não era absolutamente necessário usar um algoritmo de ware-levelling, no entanto,
8  * como não é computacionalmente exigente e altamente pedagógico decidi implementá-lo
9  * Assim, de cada vez que é guardada informação na EEPROM esses endereço fica guardado numa variável
10 * global, pelo que da proxima vez que for guarado um dispositivo o algoritmo guarda-o no próximo bloco
11 * livre, pelo que todos os enderecos serão eventualmente usados uniformizando o número de ciclos
12 * read/write em cada bloco. O primeiro bloco onde é guardo um dispositivo é escolhido aleatoriamente
13 * É de notar a diferenca entre apontador de um endereço na EEPROM (endereço de um byte na EEPROM) e um
14 * apontador para um bloco da EEPROM (número do bloco correspondente À divisão da EEPROM em blocos do
15 * tamanho da classe DeviceData). A função "realAddr" converte endereços de blocos para endereços na
16 EEPROM */
17
18 //incluir o header
19 #include "StorageManagement.h"
20
21 //----- FUNÇÕES PÚBLICAS -----
22
23 //Função que guarda um dispositivo em EEPROM
24 //Recebe o apontador para uma instância da classe DeviceData que guardará
25 //Retorna TRUE se tiver sido guardado com sucesso e FALSE em caso contrário
26 byte StorageManagement::saveOnEEPROM(DeviceData * deviceAp)
27 {
28     byte endReached = FALSE;
29     byte blockAddr;
30
31     //se o bloco onde foi escrito da ultima vez é o último, começa a procurar no primeiro
32     if (lastAddedBlockAddr != (EEPROM.length() / BLOCK_SIZE) - 1)
33         blockAddr = lastAddedBlockAddr + 1;
34     else
35         blockAddr = 0;
36
37     //itera ao longo dos blocos até encontrar um vazio
38     while (!StorageManagement::isEmpty(blockAddr)) {
39         blockAddr++;
40         if (blockAddr > EEPROM.length() / BLOCK_SIZE) {
41             if (endReached)
42                 return FALSE; //EEPROM cheia
43             else {
44                 blockAddr = 0;
45                 endReached = TRUE;
46             }
47         }
48     }
49     //se chegar a este ponto é porque há um espaço livre em memória no endereço indicado pela
50     //variável blockAddr
51     if (EEPROM_writeDevice(realAddr(blockAddr), deviceAp) != BLOCK_SIZE){ //guarda a informação nesse endereço
52         lastAddedBlockAddr = blockAddr;
53         return FALSE;
54     }
55
56     lastAddedBlockAddr = blockAddr; //atauliza a variavel que indica a posição do último bloco guardado
57     return TRUE;
58 }
59

```

```

60 //Função que elimina um dispositivo em EEPROM
61 //recebe o apontador contendo pelo menos o id do dispositivo a eliminar
62 //Retorna TRUE se a eliminação foi feita com sucesso e FALSE caso contrário
63 byte StorageManagement::deleteFromEEPROM(DeviceData *deviceAp)
64 {
65     //instância de DeviceData temporária para fazer upload dos dados em EEPROM e verificar
66     //se o id corresponde aquele que queremos apagar
67     DeviceData temp((char *)NULL);
68     byte found = FALSE;
69     byte i;
70
71     for (i = 0; i < (EEPROM.length() / BLOCK_SIZE); i++) { //itera ao longo da EEPROM
72         if (StorageManagement::isEmpty(i)) //Se um bloco estiver vazio passa à frente
73             continue;
74         //caso contrário faz upload da informação nesse endereço para a instância de DeviceData temporária
75         StorageManagement::EEPROM_readDevice(StorageManagement::realAddr(i), &temp);
76         if (temp.id == deviceAp->id) { //se o id for o mesmo apagamos a informação, caso contrário continuamos a procurar
77             found = TRUE;
78             for(byte i = 0; i>16; i++)
79                 *(deviceAp->sharedSecret + i) = *(temp.sharedSecret+i);
80             //copia o shared secret do dispositivo que queremos eliminar por forma a ser possível enviar comando de desativação
81             break;
82         }
83     }
84     if (found) {
85         for (byte j = 0; j < BLOCK_SIZE; j++)
86             EEPROM.update(StorageManagement::realAddr(i) + j, 0); //reset ao bloco da EEPROM
87         return TRUE;
88     }
89     return FALSE;
90 }
91
92 //Função lê o (deviceN)ésimo bloco com informação na EEPROM
93 //recebe o apontador com o objeto onde guardar a informação
94 //Retorna TRUE se a leitura foi feita com sucesso e FALSE caso não exista um bloco na posição pretendida
95 byte StorageManagement::readFromEEPROM(DeviceData * deviceAp, byte deviceN)
96 {
97
98     byte blockAddr = 0;
99     byte blockCount = 0;
100     while (blockAddr < EEPROM.length() / BLOCK_SIZE) { //itera ao longo da EEPROM
101         if (!StorageManagement::isEmpty(blockAddr))
102             blockCount++;
103         if (blockCount == deviceN)
104             break;
105         blockAddr++;
106     }
107     //se já iterou a EEPROM toda e já não existe o bloco na posição pedida retorna FALSE
108     if (blockCount != deviceN)
109         return FALSE;
110     //lê o dispositivo nesse endereço
111     EEPROM_readDevice(StorageManagement::realAddr(blockAddr), deviceAp);
112
113     //se encontrar, por acaso, um bloco com informação inválida, que acontece quando o pairing falha por
114     //aguma razão, elimina-o. E como é um bloco inválido terá de procurar novamente o bloco na
115     // (deviceN)ésima posição
116     if(deviceAp->hourAlarmUp == 61){
117         deviceAp->removeFromEeprom();
118         return StorageManagement::readFromEEPROM(deviceAp, deviceN);
119     }
120     return TRUE;
121 }
122

```

```

123 //Função que lê o SharedSecret de um dispositivo guardado em EEPROM
124 //recebe o apontador contendo pelo menos o id do dispositivo do qual queremos extrair o sharedSecret
125 //Retorna TRUE se foi encontrado o dispositivo, e FALSE caso contrário
126 byte StorageManagement::getSharedSecretFromEeprom(DeviceData * deviceAp)
127 {
128     DeviceData temp((char *)NULL); //instância auxiliar da classe DeviceData
129     byte blockAddr = 0;
130     byte blockCount = 0;
131     while (blockAddr < EEPROM.length() / BLOCK_SIZE) { //itera ao longo da EEPROM
132         //lê os blocos não vazios da EEPROM e verifica se era esse o dispositivo do qual queremos saber o sharedSecret
133         if (!StorageManagement::isEmpty(blockAddr))
134             EEPROM_readDevice(StorageManagement::realAddr(blockAddr), &temp);
135
136         if (temp.id == deviceAp->id){
137             for(byte i = 0; i<16; i++){
138                 *(deviceAp->sharedSecret + i) = *(temp.sharedSecret + i); //Faz upload do sharedSecret
139             }
140             return TRUE;
141         }
142         blockAddr++;
143     }
144     return FALSE;
145 }
146
147 //----- FUNÇÕES PRIVADAS -----
148 //estas funções são usadas apenas com auxiliares às outras funções contidas neste ficheiro
149
150 //Função que guarda a informação de um dispositivo na EEPROM
151 //Recebe o endereço da EEPROM onde começar a guardar a informação
152 byte StorageManagement::EEPROM_writeDevice(unsigned int addr, DeviceData *deviceAp)
153 {
154     //converte o apontador para o objeto para um apontador para um array de bytes
155     const byte* p = (byte*)(void*)deviceAp;
156     byte i;
157     for (i = 0; i < BLOCK_SIZE; i++)
158         EEPROM.update(addr++, *p++); //guarda na EEPROM
159     return i;
160 }
161
162
163
164
165 //Função que lê a informação de um dispositivo na EEPROM
166 //Recebe o endereço da EEPROM onde começar a ler a informação
167 byte StorageManagement::EEPROM_readDevice(unsigned int addr, DeviceData * deviceAp)
168 {
169     //converte o apontador para o objeto para um apontador para um array de bytes
170     byte* p = (byte*)(void*)deviceAp;
171     byte i;
172     for (i = 0; i < BLOCK_SIZE; i++)
173         *p++ = EEPROM.read(addr++); //lê da EEPROM
174     return i;
175 }
176
177 //Função power alternativa à função recursiva
178 unsigned int StorageManagement::power(int b, byte e)
179 {
180     unsigned int result = 1;
181     for (byte i = 0; i < e; i++)
182         result *= b;
183     return result;
184 }
185

```

```

186 //Função que dado um endereço de um bloco o transforma num endereço da EEPROM
187 unsigned int StorageManagement::realAddr(byte blockAddr)
188 {
189     return (blockAddr) * BLOCK_SIZE;
190 }
191
192 byte StorageManagement::isEmpty(byte blockAddr)
193 {
194     const unsigned int realAddr = StorageManagement::realAddr(blockAddr);
195     for (byte i = 0; i < BLOCK_SIZE; i++) {
196         if (EEPROM.read(i + realAddr))
197             return FALSE;
198     }
199     return TRUE;
200 }
201
202 //Função que inicia o algoritmo de ware-levelling escolhendo aleatoriamente o endereço do bloco
203 //onde começar a escrever
204 void StorageManagement::initWithWareLeveling()
205 {
206     srand(analogRead(0)); //geração aleatória da seed dos números aleatórios
207     //contrangimento dos valores obtidos para o endereço de um bloco
208     lastAddedBlockAddr = rand()%(EEPROM.length()/BLOCK_SIZE);
209 }
210
211 //----- DEBUG -----
212 #ifdef DEBUG
213 void StorageManagement::clearEEPROM()
214 {
215     for (unsigned int i = 0; i < EEPROM.length(); i++)
216         EEPROM.update(i, 0);
217 }
218 void StorageManagement::showEEPROM()
219 {
220     for (byte i = 0; i < 15; i++)
221         Serial.print("-");
222     for (unsigned int i = 0; i < EEPROM.length(); i++) {
223         if (i % BLOCK_SIZE == 0)
224             Serial.println(F(" "));
225         Serial.print(i);
226         Serial.print(F(":"));
227         Serial.print(EEPROM.read(i));
228         Serial.print(F("|"));
229     }
230     Serial.flush();
231 }
232 #endif

```

```

1 /*
2  * Ficheiro deviceCommunication.h
3  * Header que serve de suporte ao ficheiro " deviceCommunication.cpp"
4  * Estão neste ficheir definidos os protótipos das funções que tratam da comunicação com
5  * os dispositivos secundários, incluídas no namespace deviceCommunication
6  */
7
8 #ifndef DEVICE_COMMUNICATION_H
9 #define DEVICE_COMMUNICATION_H
10
11 //----- INCLUDES-----
12 #include "Arduino.h"
13 #include "DeviceData.h"
14 #include "MACROS_LIGACAO.h"
15 #include <VirtualWire.h>
16 /*Autor: Mike McCauley
17 Disponível em: http://www.airspayce.com/mikem/arduino/VirtualWire/
18 */
19
20 //----- MACROS -----
21 #define DEVICE_COMMUNICATION_TX_PIN 2
22 #define DEVICE_COMMUNICATION_RX_PIN 3
23 #define DEVICE_COMMUNICATION_BAUD_RATE 2000
24
25 #define BUFFER_LENGTH 18
26 #define RESPONSE_WAIT_TIMEOUT 600
27 #define RESPONSE_WAIT_TIMEOUT_RECEPTION 200
28 #define DEFAULT_BYTE_ARRAY_LENGTH 17
29
30 //----- PROTÓTIPOS -----
31 namespace deviceCommunication
32 {
33 void initDeviceCommunication();
34 void setUpDevices();
35 byte sendByteArray(DeviceData * deviceAp, byte * byteArray, byte byteArrayLen);
36 byte checkReception(unsigned int id);
37 byte handshakeProtocol(DeviceData * device);
38 byte getCode(byte * code128bit);
39 };
40
41 #endif

```

```

1  /* Ficheiro deviceCommunication.cpp
2  * Neste ficheiro estão definidas as funções necessárias à comunicação com os dispositivos secundários
3  * Está neste ficheiro também definido o protocolo usado para o pairing (handshake) com os dispositivos
4  * Este handskake consiste em: 1) procurar uma mensagem de tentativa de emparelhamento de um dispositivo
5  *                               2) gerar uma chave privada, gerar a pública e enviá-la
6  *                               3) receber a chave publica do dispositivo e calcular o shared secret com
                                   base na privada do central
7  *                               4) guardar o shared secret
8  * Este é o algoritmo de Diffie Hellman.
9  *
10 */
11
12 //----- INCLUDES -----
13 #include "deviceCommunication.h"
14 #include "StorageManagement.h"
15 #include "deviceCommunication.h"
16
17 using namespace StorageManagement;
18
19 //----- RF BUFFER DATA -----
20 uint8_t rfBuffer[BUFFER_LENGTH+1];
21 uint8_t rfBufferLen = BUFFER_LENGTH +1 ;
22
23
24 //----- FUNÇÕES -----
25
26 //Função que prepara a comunicação com os dispositivos secundários
27 //Inicia a comunicação rf através da biblioteca VirtualWire
28 void deviceCommunication::initDeviceCommunication()
29 {
30     //define os pins de TX e RX
31     vw_set_tx_pin(DEVICE_COMMUNICATION_TX_PIN);
32     vw_set_rx_pin(DEVICE_COMMUNICATION_RX_PIN);
33     //vw_set_ptt_pin(5);
34     vw_set_ptt_inverted(true); //Necessário para o módulo que usei
35     //Definir a velocidade de transferencia de dados
36     vw_setup(DEVICE_COMMUNICATION_BAUD_RATE);
37     #ifdef DEBUG
38     Serial.println(F("Main ready"));
39     #endif
40 }
41
42 //Função que envia toda a informação guardada em EEPROM para os respetivos dispositivos
43 //Se a energia falhar os dispositivos são atualizados automaticamente, ainda que seja possível
44 //atualizá-los manualmente através da app
45 void deviceCommunication::setUpDevices()
46 {
47     DeviceData device((char*)NULL);
48     //itera os blocos com informação guardada na EEPROM
49     for (byte i = 0; ; i++) {
50         if (!device.getDeviceFromEeprom(i))
51             break;
52         device.sendUpdatedInfo(); //envia o update
53         device.sendTime(); //atualiza a hora
54     }
55 }
56 }
57
58

```

```

59 //Função que envia um array de 16 bytes e aguarda pela confirmação de resposta do dispositivo
60 //Retorna TRUE se a mensagem foi enviada e recebida pelo dispositivo com sucesso e FALSE em caso contrário
61 byte deviceCommunication::sendByteArray(DeviceData * deviceAp, byte * byteArray, byte byteArrayLen)
62 {
63     byte sentMsgs = 0; //contagem do número de x que a mensagem foi enviada
64     do {
65         if (sentMsgs == 10)
66             break;
67         vw_send(byteArray, DEFAULT_BYTE_ARRAY_LENGTH); //envia a mensagem
68         sentMsgs++;
69         vw_rx_start();
70     } while (!(deviceCommunication::checkReception(deviceAp->id)));
71     //espera algum tempo pela confirmação de recebimento
72     //se não tiver recebido a aconfoirmação tenta outra vez até um máximo de 5x
73     vw_rx_stop();
74     if (sentMsgs < 10)
75         return TRUE;
76     return FALSE;
77 }
78 //Função que aguarda por uma mensagem de confirmação de receção proveniente do dispositivo secundário
79 // para onde foi enviada a mensagem
80 //Retorna TRUE se intercetar uma mensagem de confirmação de receção e FALSE caso contrário
81 byte deviceCommunication::checkReception(unsigned int id)
82 {
83     //Espera um tempo máximo de RESPONSE_WAIT_TIMEOUT para receber a mensagem de confirmação de receção
84     if (vw_wait_rx_max(RESPONSE_WAIT_TIMEOUT) && vw_get_message(rfBuffer, &rfBufferLen))
85     {
86         if (rfBuffer[0] == CONFIRMATION_MESSAGE) { //se a mensagem for do tipo de confirmação de receção
87             unsigned int receivedId = 0;
88             for (byte i = 1; i <= 5; i++) //identifica o dispositivo que emitiu a mensagem
89                 receivedId += StorageManagement::power(10, 5 - i) * (rfBuffer[i] - '0');
90             if (receivedId == id) //se for o dispositivo que estavamos à espera que confirmasse a receção
91                 return TRUE;
92         }
93     }
94     return FALSE;
95 }
96
97 //Função responsável pelo handshake, ie pairing, com os dispositivo secundário
98 //Retorna TRUE se foi bem sucedido e FALSE caso contrário
99 byte deviceCommunication::handshakeProtocol(DeviceData * device)
100 {
101     #ifdef DEBUG
102     Serial.println(F("Starting Handshake - waiting for message"));
103     #endif
104
105     for (byte i = 0; i < 16; i++) //faz reset do shared secret do dispositivo
106         *(device->sharedSecret + i) = 0;
107
108     vw_rx_start();
109     //espera por uma mensagem de um dispositivo de pedido de emparelhamento
110     if (vw_wait_rx_max(30000) && vw_get_message(rfBuffer, &rfBufferLen)) {
111         Serial.println("REcebi");
112         Serial.flush();
113         Serial.println(rfBuffer[0]);
114         if (rfBuffer[0] == PAIRING) {
115             unsigned int receivedId = 0;
116             for (byte i = 1; i <= 5; i++) //identifica o ID do emissor
117                 receivedId += StorageManagement::power(10, 5 - i) * (rfBuffer[i] - '0');
118             device->id = receivedId;
119
120             //cria a chave privada do dispositivo central aleatoriamente
121             uint64_t mainPrivate1 = randomint64();
122             uint64_t mainPrivate2 = randomint64();
123             //calcula a chave publica do dispositivo central através da chave privada determinada anteriormente
124             uint64_t mainPublic1 = compute(G, mainPrivate1, P);
125             uint64_t mainPublic2 = compute(G, mainPrivate2, P);
126             byte codeM[17];
127             codeM[0] = CODE; //cria o vetor de bytes a ser enviado
128

```

```

129 //conversão em array de bytes da chave pública
130 byte * p = (byte *) &mainPublic1;
131 for (byte i = 0; i < 8; i++)
132     codeM[8 - i] = *(p + i);
133 p = (byte *) &mainPublic2;
134 for (byte i = 0; i < 8; i++)
135     codeM[16 - i] = *(p + i);
136
137 #ifdef DEBUG
138 Serial.println(F("Sent: "));
139 for (byte i = 1; i <= 16; i++)
140 {
141     Serial.print(codeM[i], HEX);
142     Serial.print(F(" "));
143 } Serial.println(F(" "));
144 Serial.flush();
145 #endif
146
147 //Envia o código até receber o código público do dispositivo secundário
148 byte count = 0;
149 do {
150     count++;
151     vw_send(codeM, DEFAULT_BYTE_ARRAY_LENGTH+1); //envia o código
152     vw_wait_tx();
153     #ifdef DEBUG
154     Serial.println(F("Sent code"));
155     #endif
156     //Envia o código no máximo 15x vezes
157     if (count == 30)
158         return FALSE;
159 }while (!deviceCommunication::getCode(codeM));
160
161 #ifdef DEBUG
162 Serial.println("Received:");
163 for (byte i = 0; i < 16; i++)
164 {
165     Serial.print(codeM[i], HEX);
166     Serial.print(F(" "));
167 } Serial.println(F(" "));
168 Serial.flush();
169 #endif
170
171 //converte chave recebida sob a forma de array de bytes em dois uint64_t
172 mainPublic1 = 0;
173 for (byte i = 0; i < 8; i++)
174     mainPublic1 += (uint64_t)codeM[i] << 8 * (7 - i);
175 mainPublic2 = 0;
176 for (byte i = 0; i < 8; i++)
177     mainPublic2 += (uint64_t)codeM[i + 8] << 8 * (7 - i);
178
179 //Calcula o shared secret a partir da chave publica do dispositivo e da chave privada do dispositivo central
180 mainPublic1 = compute(mainPublic1, mainPrivate1, P);
181 mainPublic2 = compute(mainPublic2, mainPrivate2, P);
182
183 //verifica que a chave foi de facto recebida corretamente e que o shared secret não é nulo
184 count = FALSE;
185 for (byte i = 0; i = 8; i++) {
186     if (mainPublic1 != 0) {
187         count = TRUE;
188         break;
189     }
190 }
191 if (!count)
192     return FALSE;
193 count = FALSE;
194 for (byte i = 0; i = 8; i++) {
195     if (mainPublic2 != 0) {
196         count = TRUE;
197         break;
198     }
199 }
200 if (!count)
201     return FALSE;
202

```



```

203      //converte os 2 uint_64t num array de bytes que são guardados no objeto do dispositivo e
posteriormente em EEPROM
204      p = (byte *) &mainPublic1;
205      for (byte i = 0; i < 8; i++)
206          *(device->sharedSecret + 7 - i) = *(p + i);
207      p = (byte *) &mainPublic2;
208      for (byte i = 0; i < 8; i++)
209          *(device->sharedSecret + 15 - i) = *(p + i);
210
211      /* #ifdef DEBUG
212      Serial.println(F("Shared secret: "));
213      for (byte i = 0; i < 16; i++) {
214          Serial.print(*(device->sharedSecret + i), HEX);
215          Serial.print(F(" "));
216      }
217      Serial.println(F(" "));
218      Serial.flush();
219      #endif*/
220
221      vw_rx_stop();
222      return TRUE;
223  }
224
225      #ifdef DEBUG
226      Serial.println(F("Na ta ca nada"));
227      #endif
228
229      vw_rx_stop();
230      return FALSE;
231  }
232
233      return FALSE;
234  }
235
236  //Função que aguarda a recepção de uma mensagem que contenha um código que guarda no vetor que é passado
237  // por endereço. Retorna TRUE se recebeu um código válido e FALSE caso contrário
238  byte deviceCommunication::getCode(byte * code128bit)
239  {
240      vw_rx_start();
241      #ifdef DEBUG
242      Serial.println(F("Trying to get code"));
243      #endif
244
245      //Espera o recebimento de uma mensagem
246      if (vw_wait_rx_max(RESPONSE_WAIT_TIMEOUT) && vw_get_message(rfBuffer, &rfBufferLen))
247      {
248          vw_rx_stop();
249          if (rfBuffer[0] == CODE) { //verifica se é um código
250              #ifdef DEBUG
251              Serial.println(F("Código recebido"));
252              #endif
253              for (byte i = 1; i <= 16; i++) {
254                  *(code128bit + i - 1) = rfBuffer[i]; //extraí o código do buffer da comunicação rf
255                  //Serial.println(*(code128bit + i - 1), HE);
256              }
257              return TRUE;
258          }
259      }
260      vw_rx_stop();
261      return FALSE;
262  }

```

```

1 /*
2  * Ficheiro DiffieHellman.h
3  * Header que serve de suporte ao ficheiro "DiffieHellman.cpp"
4  * Este ficheiro contem os protótipos das funções e a definição de algumas
5  * constantes necessasárias no algoritmo de Diffie Hellman
6  */
7
8 //----- INCLUDES -----
9 #include "Arduino.h"
10
11 //----- MACROS -----
12 #define P 0xffffffffffffc5ul //maior primo de 64 bits
13 #define G 5 //constante para a geração do shared secret
14
15 //----- FUNÇÕES -----
16 uint64_t randomint64();
17 uint64_t compute(uint64_t a, uint64_t m, uint64_t n);

```

```

1 /*Ficheiro DiffieHellman.cpp
2  * Este ficheiro incorpora as funções necessárias para a geração do sharedSecret,
3  * a chave de 128 bits única para cada par persiana-dispositivo central que é definida
4  * aquando da primeira comunicação entre os dois dispositivos.
5  * Nesta primeira comunicação os dois dispositivos fazem uma troca pública de chaves
6  * baseada no algoritmo de Diffie Hellman.
7  * Desta forma, tendo estabelecido uma chave secreta todas as comunicações podem ser
8  * encriptadas para que seja substancialmente mais difícil o acesso de dispositivos não autorizadas
9  * à persiana. Esta encriptação é feita através do algoritmo AES com uma chave de 128 bits.
10 * Para além desta encriptação todas as mensagens são enviadas com alguns bytes aleatórios
11 * cuja soma mod 256 é igual ao último byte. Assim, é impossível controlar as persianas através
12 * de uma inteceção das comunicações e sua réplica.
13 */
14
15 //inclusão do header
16 #include "DiffieHellman.h"
17
18 //Função que gera um número aleatório de 64 bits
19 uint64_t randomint64() {
20     //os número gerados pela função rand() têm 16bits pelo que é necessário gerar 4
21     uint64_t a = rand();
22     uint64_t b = rand();
23     uint64_t c = rand();
24     uint64_t d = rand();
25     //gerar um numero de 64 bits inserindo quatro blocos de 16 bits
26     return a << 48 | b << 32 | c << 16 | d;
27 }
28
29 //Função que calcula a^n mod n e retorna o resultado
30 uint64_t compute(uint64_t a, uint64_t m, uint64_t n)
31 {
32     uint64_t r;
33     uint64_t y = 1;
34     while (m > 0){
35         r = m % 2;
36         if (r == 1)
37             y = (y * a) % n;
38         a = a * a % n;
39         m = m / 2;
40     }
41     return y;
42 }

```

```

1  /*
2  * Ficheiro timeHandling.h
3  * Header que serve de suporte ao ficheiro "timeHandling.cpp"
4  * Estão definidos neste ficheiro os protótipos das funções que tratam da gestão do tempo
5  */
6
7  #ifndef TIME_HANDLING_H
8  #define TIME_HANDLING_H
9
10 //----- INCLUDES -----
11 #include <DS1302RTC.h>
12 /*Autor: Timur Maksimov
13 */
14
15 #include <TimeLib.h>
16 /*Autores: Michael Margolis
17           Paul Stoffregen
18 Disponível em: https://github.com/PaulStoffregen/Time
19 */
20
21 #include <EtherCard.h>
22 /*Autores: Jean-Claude Wippler
23           Pascal Stang
24           Guido Socher
25 Disponível em: https://github.com/jcw/ethercard.git
26 */
27
28
29 #include "user_communication.h"
30
31 //----- MACROS -----
32 #define DAY 1440000UL // 1 dia em segundos
33
34
35 //----- PROTÓTIPOS -----
36 void setUpRTC();
37 byte updateRTC();
38 unsigned long getNtpTime();
39 void checkNeedToUpdateTime();
40
41 #ifndef NO_DEBUG
42 void showTime();
43 #endif
44
45 #endif

```

```

1  /*
2  * Ficheiro timeHandling.cpp
3  * Este ficheiro é o que contem a definição das funções que gerem o tempo, nomeadamente as
4  * que gerem a forma de guardar a informação do tempo, atualizam o tempo através da ligação
5  * à internet quando necessário
6  * O tempo é atualizado a cada 24 horas
7  */
8  //incluir o header
9  #include "timeHandling.h"
10
11
12 unsigned long lastUpdate = 0;
13 // Set pins: CE(rst), IO(dat),CLK
14 DS1302RTC RTC(9, 8, 7);
15
16
17 //endereço de onde será atualizada a informação do tempo
18 const char website[] PROGMEM = "0.pool.ntp.org";
19 uint8_t ntpMyPort = 123;
20
21 //Função que inicia o Real Time Clock (RTC)
22 void setUpRTC()
23 {
24     RTC.haltRTC(false);
25     RTC.writeEN(false);
26     setSyncProvider(RTC.get);
27     setSyncInterval(5);
28 }
29
30 //Função que atualiza o tempo através da ligação à internet
31 byte updateRTC()
32 {
33     //Só é possível obter informação do tempo se for fornecido
34     // Net Mask, the Gateway IP address, and the DNS Server IP address
35     if (!ether.dhcpSetup())
36         return FALSE;
37     if (!ether.dnsLookup(website))
38         return FALSE;
39
40     #ifdef DEBUG
41         setTime(23, 31, 30, 13, 2, 2018);
42     #else
43         setTime(getNtpTime());
44     #endif;
45
46     RTC.set(now()); //set the RTC from the system time
47     if(!ether.staticSetup(myip*, gwip*))
48         return FALSE;
49     return TRUE;
50 }
51
52 //Função que atualiza o tempo e converte o tempo recebido da internet para o formato de
53 //tempo desde 1 de janeiro de 1970
54 //Retorna 0 se não conseguiu atualizar o tempo ou o tempo desde 1 de janeiro de 1970
55 //caso contrário
56 unsigned long getNtpTime()
57 {
58     byte count = 0;
59     unsigned long timeFromNTP;
60     const unsigned long seventy_years = 2208988800UL;
61     ether.ntpRequest(ether.hisip, ntpMyPort);
62     while(true) {
63         word length = ether.packetReceive();
64         ether.packetLoop(length);
65         if(length > 0 && ether.ntpProcessAnswer(&timeFromNTP, ntpMyPort))
66             return (timeFromNTP - seventy_years);
67         if(count > 254) //limitar as iterações para prevenir um ciclo infinito
68             return 0;
69         count++;
70     }
71     return 0;
72 }
73

```

```

74 //Função que verifica se é necessário fazer a atualização do tempo do dispositivo central
75 //Nesse caso atualiza-o
76 void checkNeedToUpdateTime()
77 {
78     unsigned long currentMillis = millis();
79     if(currentMillis-lastUpdate > DAY){
80         lastUpdate = currentMillis;
81         if(!updateRTC()){
82             #ifdef DEBUG
83                 Serial.println(F("ERROR updating time"));
84             #endif
85         }
86         #ifdef DEBUG
87         else
88             Serial.println(F("Updated"));
89         #endif
90     }
91 }
92
93 #ifdef DEBUG
94 void showTime()
95 {
96     time_t t = now();
97     Serial.print(F("DAY: "));
98     Serial.print(day(t), DEC);
99     Serial.print(F("/"));
100    Serial.print(month(t));
101    Serial.print(F("/"));
102    Serial.println(year(t), DEC);
103    Serial.print(F("WEEK: "));
104    Serial.println(weekday(t), DEC);
105    Serial.print(F("TIME: "));
106    Serial.print(hour(t), DEC);
107    Serial.print(F(":"));
108    Serial.print(minute(t), DEC);
109    Serial.print(F(":"));
110    Serial.println(second(t), DEC);
111 }
112 #endif

```

```

1 /*Ficheiro MACROS_LIGACAO.h
2  * Ficheiro onde são definidas as principais constantes necessárias para a interpretação
3  * das mensagens recebidas do utilizador através da aplicação
4  * A maioria deste ficheiro é semelhante aos ficheiros homónimos da aplicação
5  */
6
7 #ifndef MACROS_LIGACAO_H
8 #define MACROS_LIGACAO_H
9
10
11 // #define NO_DEBUG
12 #define DEBUG
13
14 #define TRUE 1
15 #define FALSE 0
16
17 //----- APP COMMUNICATION MACROS -----
18
19 #define OPEN 'o'
20 #define CLOSE 'c'
21 #define DATA 'D'
22 #define PAIRING_DEVICE 'p'
23 #define DEVICE_DELETE 'd'
24 #define DEVICE_EDIT 'e'
25 #define DEVICE_ADD 'a'
26 #define MAIN_TEST 'T'
27 #define DEVICE_TEST 't'
28 #define HTTPOK 'h'
29
30 #define ID 'i'
31 #define NAME 'n'
32 #define SPACE_MARKER '&'
33
34 #define HOUR_ALARM_UP 'H'
35 #define MINUTE_ALARM_UP 'M'
36 #define FREQ_ALARM_UP 'F'
37
38 #define HOUR_ALARM_DOWN 'h'
39 #define MINUTE_ALARM_DOWN 'm'
40 #define FREQ_ALARM_DOWN 'f'
41
42 #define DISABLED 'd'
43 #define EVERYDAY 'e'
44 #define WEEKDAYS 'w'
45 #define WEEKENDS 'W'
46
47 #define STRING_DELIMITATOR '"'
48 #define TWO_DOTS ':'
49 #define COMA ','
50
51 //----- DEVICE COMMUNICATION MACROS -----
52 #define TIME 't'
53 #define INFO 'i'
54 #define DISABLE 'd'
55 #define CONFIRMATION_MESSAGE 'r'
56 #define PAIRING 'p'
57 #define CODE 'c'
58 #define TEST 'T'
59
60 #endif

```

```

1 /*
2  * Ficheiro user_communication.h
3  * Header que serve de suporte ao ficheiro "user_communication.cpp"
4  * Está neste ficheiro definida a classe DeviceData usada para a gestão mais
5  * organizada e eficiente da interação com os dispositivos secundários
6  */
7
8 #ifndef USER_COMMUNICATION_H
9 #define USER_COMMUNICATION_H
10
11 //----- INCLUDES -----
12 #include <SPI.h>
13 //default Arduino library under a Creative Commons Attribution-ShareAlike 3.0 License
14
15 #include <EtherCard.h>
16 /*Autores: Jean-Claude Wippler
17           Pascal Stang
18           Guido Socher
19 Disponível em: https://github.com/jcw/ethercard.git
20 */
21
22 #include <stdlib.h>
23 //standard c++ library
24
25 #include "MACROS_LIGACAO.h"
26 #include "Arduino.h"
27 #include "DeviceData.h"
28 #include "StorageManagement.h"
29
30 using namespace StorageManagement;
31
32 //----- PROTÓTIPOS -----
33 byte setUpConnection();
34 char * checkReception();
35 byte analyseCommand(char * cmd);
36 void sendJson(char * json);
37 void fillBufferWithJson(char * json);
38 char * extractCommand(char * data);
39 void deleteEqualRequests();
40
41 //----- DETALHES DO SERVIDOR -----
42 static byte myip[] = {192, 168, 1, 205}; // ethernet interface ip address
43 static byte gwip[] = {192, 168, 1, 1}; // gateway ip address
44
45 //----- JSON MESSAGES -----//
46
47 //default header
48 const char http_header[] PROGMEM =
49 "HTTP/1.0 200 OK\r\n"
50 "Content-Type: application/json;charset=utf-8\r\n"
51 "Server: Arduino\r\n"
52 "Retry-After: 600\r\n"
53 "Connection: close\r\n\r\n";
54
55 const char OK_Json[] PROGMEM = "{\"h\":1}\r\n";
56 const char NOTOK_Json[] PROGMEM = "{\"h\":0}\r\n";
57 const char PROGMEM END_OF_TRANSMISSION[] = "{\"x\":1}\r\n";
58
59
60 #endif

```



```

1  /*
2  * Ficheiro user_communication.cpp
3  * Este ficheiro é o que contem as funções que regulam a comunicação com a app e que
4  * funcionam como ponte entre o utilizador e os dispositivos
5  * O Arduino funciona como um servidor que recebe requests da app contendo a ação que o
6  * Arduino deve executar e envia a resposta para a app correspondente
7  * O request contem a ação e informação necessária de forma codificada de caordo com uma
8  * convenção adotada
9  */
10
11
12 #include "user_communication.h" //incluir o header
13
14 //----- VARIÁVEIS GLOBAIS -----
15 byte Ethernet::buffer[350]; //buffer
16 BufferFiller bfill;
17
18 //----- DETALHES DO SERVIDOR -----//
19 byte PASSWORD[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0};
20 static byte mymac[] = {0x74, 0x69, 0x69, 0x2D, 0x30, 0x31}; //ethernet mac address
21
22 //----- FUNÇÕES -----//
23
24 //Função que inicia a ligação À internet através do ethernet module e comunicação SPI
25 //e também inicia o algoritmo de gestão da EEPROM
26 //Retorna TRUE se estabelecido com sucesso e FALSE em caso contrário
27 byte setUpConnection()
28 {
29     ENC28J60::initSPI(); //inicia comunicação SPI
30     if (!ether.begin(sizeof Ethernet::buffer, mymac))
31         return FALSE;
32
33     if (!ether.staticSetup(myip*, gwip*)) //Inicia servidor de IP estático
34         return FALSE;
35
36     //predefinições necessárias ao tipo de comunicação pretendida
37     ENC28J60::disableBroadcast(false);
38     ENC28J60::disableMulticast();
39     StorageManagement::initWithWareLeveling();
40     return TRUE;
41 }
42
43 //Função que envia a resposta a um request feito pela aplicação na forma de JSON
44 void sendJson(char * json)
45 {
46     bfill = ether.tcpOffset(); //apontador para o buffer
47     bfill.emit_p(PSTR("$F""$S\r\n"), http_header, (word) json); //preencher o buffer
48     ether.httpServerReply(bfill.position()); //enviar a resposta
49     #ifdef DEBUG
50     Serial.println(json);
51     #endif
52 }
53
54 //Função que recebe como argumento TRUE ou FALSE
55 //Se receber TRUE envia uma resposta de HTTP OK caso contrário envia
56 //uma resposta de erro
57 void sendOK(byte ok)
58 {
59     bfill = ether.tcpOffset(); //apontador para o buffer
60     switch (ok) {
61         case TRUE: {
62             bfill.emit_p(PSTR("$F""$F"), http_header, OK_Json); //preencher o buffer com HTTP OK
63             break;
64         }
65         case FALSE: {
66             bfill.emit_p(PSTR("$F""$F"), http_header, NOTOK_Json); //preencher o buffer com erro
67             break;
68         }
69     }
70     ether.httpServerReply(bfill.position()); //enviar a resposta
71 }
72

```

```

73 //Função que verifica a recepção de um request enviado pela aplicação
74 //Se identificar um request executa a ação corresponde e envia a resposta
75 //Retorna o apontador para uma string que contem o comando propriamente dito e que será analisado por
    outra função
76 char * checkReception()
77 {
78     word pos = ether.packetLoop(ether.packetReceive()); //verifica a recepção
79     if (pos) { //se a posição no buffer da informação recebida não for NULL, ie, tiver recebido informação
80         char *data = (char *) Ethernet::buffer + pos; //data aponta para o começo da informação recebida
81         if (strncmp("GET /", data, 5) || data[5] > '9' || data[5] < '0') //Verifica se se trata de um request
82             return NULL;
83         else {
84             byte passRight = TRUE;
85             for (int i = 5; i < 15; i++) { //verifica a password numa posição convencionada do request
86                 if (data[i] - '0' != PASSWORD[i - 5]) {
87                     passRight = FALSE;
88                     break;
89                 }
90             }
91
92             if (!passRight)
93                 return NULL;
94             else
95                 return extractCommand(data); //extraí a parte do comando relevante recebido se a password estiver correta
96         }
97     }
98
99     return NULL;
100 }
101
102 //Função que coloca um caracter terminador no final do comando que é relevante e retorna
103 //o apontador para o seu início
104 char * extractCommand(char * data)
105 {
106     int n_caracteres_comando = strcspn(data, " HTTP");
107     data[n_caracteres_comando] = '\0';
108     return data + 15;
109 }
110
111 //Função que analisa o comando que recebe através de um apontador para uma string
112 //e executa a ação correspondente
113 byte analiseCommand(char * cmd)
114 {
115     switch (cmd[0]) { //o primeiro bit refere-se à ação a tomar
116         case DATA: { //request de informação do dispositivo na posição deviceN ocupada da EEPROM
117             DeviceData device((char *)NULL);
118             byte deviceN = 10 * (cmd[1] - '0') + (cmd[2] - '0'); //extraí o deviceN do dispositivo
119             if (device.getDeviceFromEeprom(deviceN)) { //faz upload da informação da EEPROM
120                 char json[96];
121                 device.getJsonString(json); //converte a informação para JSON
122                 sendJson(json); //envia a informação em JSON
123             } else {
124                 //se não houver dispositivo na posição deviceN envia string JSON convencionado com esse significado
125                 sendJson("{\"x\":1}");
126             }
127             break;
128         }
129         case OPEN: { //request para abrir persiana
130             DeviceData device (cmd, ID_ONLY); //extraí o id do dispositivo e coloca-o no objeto
131             device.getSharedSecretFromEeprom(); //extraí o shared secret necessário a comunicação da instrução de abertura
132             if (device.openCmd()) //envia comando de abertura e aguarda pela resposta
133                 sendOK(TRUE); //Se recebeu confirmação de recepção responde com HTTP OK
134             else
135                 sendOK(FALSE); //caso contrário envia mensagem de erro
136             break;
137         }
138     }

```

```

138 case CLOSE: { //request para fechar persiana
139     DeviceData device (cmd, ID_ONLY); //extraí o id do dispositivo e coloca-o no objeto
140     device.getSharedSecretFromEeprom(); //extraí o shared secret necessário a comunicação da instrução de fecho
141     if (device.closeCmd()) //envia comando de fecho e aguarda pela resposta
142         sendOK(TRUE); //Se recebeu confirmação de receção responde com HTTP OK
143     else
144         sendOK(FALSE); //caso contrário envia mensagem de erro
145     break;
146 }
147 case PARING_DEVICE: { //request para tentar fazer pairing
148     DeviceData device((char *)NULL);
149     char json[14];
150
151     if(!device.getParingDevice()){
152         //se o pairing não for bem sucedido remove o dispositivo que possa ter sido guardado em memória
153         device.removeFromEeprom();
154         break;
155     }
156     Serial.println("HERE COMM OKKKK");
157     device.getJsonStringParing(json);
158     sendJson(json);
159     device.sendTime();
160     break;
161 }
162 case DEVICE_DELETE: { //request para eliminar dispositivo
163     DeviceData device (cmd, ID_ONLY); //extraí o id do dispositiv do comando
164     device.getSharedSecretFromEeprom(); //upload do shared secret
165     //falha ao desativar dispositivo pelo que não pode ser eliminado, porque continuaria a abrir e
166     //fechar e não teríamos controlo sobre ele
167
168     if (device.sendDeactivation()) {
169         device.removeFromEeprom(); //remove da EEPROM
170         sendOK(TRUE); //envia HTTP OK
171     } else
172         sendOK(FALSE); //envia ERRO
173     break;
174 }
175 case DEVICE_EDIT: { //request para editar a informação de um dispositivo
176     DeviceData device(cmd); //extraí informação do comando cerca dos alarmes
177     device.getSharedSecretFromEeprom(); //extraí o shared secret
178     //só se conseguir remover o antigo adicionar o novo e enviar a atualização é que a edição é
179     //autorizada na aplicação
180
181     if (device.removeFromEeprom() && device.saveOnEeprom() && device.sendUpdatedInfo())
182         sendOK(TRUE); //envia HTTP OK
183     else
184         sendOK(FALSE); //envia ERRO
185     break;
186 }
187 case DEVICE_ADD: { //request para adicionar um dispositivo novo
188     DeviceData device(cmd); //extraí informação do comando cerca dos alarmes
189     device.getSharedSecretFromEeprom(); //extraí o shared secret
190     //só se conseguir adicionar o novo dispositivo e enviar a atualização é que a adição é
191     //autorizada na aplicação
192
193     if (device.saveOnEeprom() && device.sendUpdatedInfo())
194         sendOK(TRUE); //envia HTTP OK
195     else
196         sendOK(FALSE); //envia ERRO
197     break;
198 }

```

```

193 case DEVICE_TEST: { //request verificar se o dispositivo está comunicável
194     DeviceData device(cmd, ID_ONLY); //extraí o id do dispositivo
195     device.getSharedSecretFromEeprom();//faz upload do shared secret
196     if (device.test()) { //faz ping ao dispositivo
197         sendOK(TRUE); //envia HTTP OK
198         //device.show();
199         #ifndef NO_DEBUG
200         Serial.println(F("Test OK"));
201         Serial.flush();
202         #endif
203     } else {
204         sendOK(FALSE); //envia ERRO
205         #ifndef NO_DEBUG
206         Serial.println(F("Test NOT OK"));
207         Serial.flush();
208         #endif
209     }
210     break;
211 }
212 case MAIN_TEST:
213 { //request para verificar a conexão da aplicação ao dispositivo central envia também o número de
    //dispositivos guardados que estão online
214     byte count = 0;
215     DeviceData device((char *) NULL);
216     for(byte i = 0; ; i++){
217         if(!device.getDeviceFromEeprom(i))
218             break;
219         if(device.test())
220             count++;
221     }
222
223     //cria uma string no formato JSON
224     char msg[9];
225     msg[0] = '{';
226     msg[1] = STRING_DELIMITATOR;
227     msg[2] = HTTPOK;
228     msg[3] = STRING_DELIMITATOR;
229     msg[4] = TWO_DOTS;
230
231     if(count == 0)
232         count = 61;
233
234     sprintf(msg+5, "%u", count);
235     if(count > 9){
236         msg[7] = '}';
237         msg[8] = '\0';
238     }else{
239         msg[6] = '}';
240         msg[7] = '\0';
241         msg[8] = '\0';
242     }
243     sendJson(msg);
244     break;
245 }
246 default: {
247     #ifndef NO_DEBUG
248     Serial.println(F("NADA"));
249     Serial.flush();
250     #endif
251     sendOK(FALSE);
252 }
253 }
254 deleteEqualRequests(); //elimina pedidos iguais
255 return TRUE;
256 }
257

```

```
258 //Função que elimina requests feitos ao servidor que sejam repetidos
259 //Como o Arduino demora algum tempo a processar o comando e tem de esperar pela confirmação de receção
260 //dos dispositivos secundários a app vai enviando requests iguais até receber uma resposta ou passar
261 //o tempo de timeout
262 //Mesmo que sejam apagados requests diferentes a app vai enviá-los novamente por falta resposta
263 void deleteEqualRequests()
264 {
265     for (byte i = 0; i < 100; i++) {
266         ether.packetLoop(ether.packetReceive());
267         delay(1); //delay para estailidade
268     }
269 }
270 }
```

## Código do Dispositivo Secundário

Fcheiros:

1. DomusSapiensDevice.ino
2. alarm.h
3. alarm.cpp
4. switching.h
5. switching.cpp
6. MACROS\_COMMUNICATION.h
7. communication.h
8. communication.cpp
9. DiffieHellman.h
10. DiffieHellman.cpp

Todos os ficheiros podem ser consultados em:

[github.com/LeonardoPedroso/DomusSapiens](https://github.com/LeonardoPedroso/DomusSapiens)

```

1 //Ficheiro DomusSapiensDevice.ino
2
3 //incluir funções necessárias dos outros ficheiros
4 #include "communication.h"
5 #include "alarm.h"
6 #include "switching.h"
7 using namespace communication;
8
9 //comentar instrução para desativar modo de debug
10 #define DEBUG
11
12 void setup() {
13
14     #ifdef DEBUG
15     Serial.begin(57600);
16     #endif
17
18     initAlarm(); //fazer reset ao alarm
19     communication::initCommunication(); //inicia as comunicações de rf
20     initSwitches(); //inicia os botoes
21 }
22
23 void loop() {
24     checkAlarm(); //verifica se é hora de alarme
25     char cmd[16];
26     if (communication::checkReception(cmd)) //verifica receção
27         communication::analyseCommand(cmd);
28     checkSwitches(); //verifica estado dos botoes
29 }

```

```

1  /*Ficheiro alarm.h
2  * Header de suporte ao ficheiro alarm.cpp em que é definida a estrutura ALARM
3  * e são definidos os protótipos das funções necessárias para o controlo do alarme
4  */
5
6  #ifndef ALARM_H
7  #define ALARM_H
8
9  //----- includes -----
10 #include "Arduino.h"
11 #include <TimeLib.h>
12 /*Autores: Michael Margolis
13           Paul Stoffregen
14 Disponível em: https://github.com/PaulStoffregen/Time
15 */
16 #include "switching.h"
17
18 //----- MACROS -----
19 #define DISABLED 'd'
20 #define EVERYDAY 'e'
21 #define WEEKDAYS 'w'
22 #define WEEKENDS 'W'
23
24 //----- estrutura ALARM -----
25
26 //estrutura que guarda a informação do alarme do dispositivo
27 struct ALARM{
28     byte hourAlarmUp;
29     byte minuteAlarmUp;
30     char freqUp;
31     byte hourAlarmDw;
32     byte minuteAlarmDw;
33     char freqDw;
34 };
35
36 //----- variáveis globais -----
37 extern byte sharedSecret[16];
38
39 //----- protótipos -----
40 void initAlarm();
41 void showTime();
42 void showAlarm();
43 void checkAlarm();
44
45 #endif

```



```

1  /*Ficheiro alarm.cpp
2  * Ficheiro onde estão definidas as funções necessáias para o funcionamento
3  * do alarme.
4  */
5
6  //incluir o header
7  #include "alarm.h"
8
9  //----- variáveis globais -----
10 ALARM alarm;
11
12 //função que inicia o alarm, desativando-o
13 void initAlarm()
14 {
15     alarm.hourAlarmUp = 0;
16     alarm.minuteAlarmUp = 0;
17     alarm.freqUp = DISABLED;
18
19     alarm.hourAlarmDw = 0;
20     alarm.minuteAlarmDw = 0;
21     alarm.freqDw = DISABLED;
22 }
23
24 /*
25 * Função que retira a hora do relógio do sistema e verifica se essa hora corresponde a algum
26 * alarme, se sim verifica a periodicidade definida e consoante o dia da semana abre/fecha a.
27 persiana ou não */
28 void checkAlarm()
29 {
30     time_t t = now(); //hora atual
31     //verifica se está na hora de subir a persiana
32     if(alarm.freqUp!=DISABLED && hour(t)==alarm.hourAlarmUp && minute(t)==alarm.minuteAlarmUp && second(t)==0){
33         if(alarm.freqUp == EVERYDAY)
34             Open();
35         else if(alarm.freqUp == WEEKDAYS){
36             if(weekday(t)>1 && weekday(t)<=6) //verifica o dia da semana
37                 Open();
38             }else if (weekday(t)==1 || weekday(t)==7)
39                 Open();
40         }//verifica se está na hora de descer a persiana
41     else if(alarm.freqDw!=DISABLED && hour(t)==alarm.hourAlarmDw && minute(t)==alarm.minuteAlarmDw && second(t)==0){
42         if(alarm.freqDw == EVERYDAY)
43             Close();
44         else if(alarm.freqDw == WEEKDAYS){
45             if(weekday(t)>1 && weekday(t)<=6) //verifica o dia da semana
46                 Close();
47             }else if (weekday(t)==1 || weekday(t)==7) //verifica o dia da semana
48                 Close();
49         }
50 }
51

```

```

52 //Funções de debug
53 #ifdef DEBUG
54 void showTime(){
55     time_t t = now();
56     Serial.print(F("Day:"));
57     Serial.print(day(t), DEC);
58     Serial.print(F(" Month:"));
59     Serial.print(month(t));
60     Serial.print(F(" Year:"));
61     Serial.print(year(t), DEC);
62     Serial.print(F(" Week: "));
63     Serial.print(weekday(t), DEC);
64
65     Serial.print(hour(t), DEC);
66     Serial.print(F(":"));
67     Serial.print(minute(t), DEC);
68     Serial.print(F(":"));
69     Serial.print(second(t), DEC);
70     Serial.println(F("\n"));
71 }
72
73 void showAlarm(){
74     Serial.println(alarm.hourAlarmUp);
75     Serial.println(alarm.minuteAlarmUp);
76     Serial.println(alarm.freqUp);
77     Serial.println(alarm.hourAlarmDw);
78     Serial.println(alarm.minuteAlarmDw);
79     Serial.println(alarm.freqDw);
80 }
81 #endif

```

```

1 /*Ficheiro switching.h
2  * Header de suporte ao ficheiro switching.cpp em que são definidas várias funções
3  * necessárias para a o controlo de abertura e fecho das persianas, bem como de os botões
4  * que também as controlam
5  */
6 #ifndef SWITCHING_H
7 #define SWITCHING_H
8
9 //----- includes -----
10 #include "alarm.h"
11 #include "communication.h"
12
13 //----- MACROS -----
14 #define SWITCH_UP_PIN 7
15 #define SWITCH_DW_PIN 8
16
17 #define RELAY_UP_PIN 5
18 #define RELAY_DW_PIN 6
19
20 #define PAIRING_BUTTON_PIN 4
21
22 #define TIME_ON_ANALOG_PIN A0
23
24 #define OFF 0
25 #define REMOTE_UP 1
26 #define REMOTE_DW 2
27 #define BUTTON_UP 3
28 #define BUTTON_DW 4
29
30 #define BLOCKED 5
31
32 #define DEBOUNCE_SAMPLES 10
33
34
35 //----- protótipos -----
36 void checkSwitches();
37 void initSwitches();
38
39 void checkTimeOn(byte relayPin);
40
41 void Open();
42 void Close();
43
44 unsigned int timeOn();
45
46 byte debouceSwitch(byte pin);
47
48 #endif

```

```

1  /*
2   Ficheiro switching.cpp
3   Ficheiro onde estão definidas as funções necessárias para a o controlo do estado das
4   persianas, nomeadamente a verificação dos estados dos botões e funções que ativam a relé
5   que aciona o motor das persianas.*/
6
7  #include "switching.h"
8
9  //incluir o header
10 using namespace communication;
11
12 //variaveis globais
13 unsigned long hasBeenOnSince;
14 byte state;
15 unsigned int TIME_ON;
16
17 /*
18  * Função que inicializa todos os pins necessários para o controlo das persianas, e que
19  * inicializa também as variáveis de controlo */
20 void initSwitches(){
21
22
23
24   pinMode(SWITCH_UP_PIN, INPUT_PULLUP);
25   pinMode(SWITCH_DW_PIN, INPUT_PULLUP);
26
27   pinMode(RELAY_UP_PIN, OUTPUT);
28   pinMode(RELAY_DW_PIN, OUTPUT);
29
30   digitalWrite(RELAY_UP_PIN, HIGH);
31   digitalWrite(RELAY_DW_PIN, HIGH);
32
33   TIME_ON = timeOn();
34   state = OFF;
35 }
36

```

```

37 /*
38  * Função que verifica o estado de todos os botões e que evoca as funções necessárias
39  */
40 void checkSwitches()
41 { //se o botão de pairing for premido tenta o handshake
42   if (!debouceSwitch(PAIRING_BUTTON_PIN)){
43     tryHandShakeProtocol();
44     return;
45   }
46
47   if (state == OFF) { //se estado é OFF verifica se os botões para cima/baixo estão ligados
48     if (!debouceSwitch(SWITCH_UP_PIN)) {
49       digitalWrite(RELAY_DW_PIN, HIGH);
50       delay(1);
51       digitalWrite(RELAY_UP_PIN, LOW);
52       hasBeenOnSince = millis(); //tempo desde que está ligado
53       state = BUTTON_UP; //atualiza estado
54     } else if (!debouceSwitch(SWITCH_DW_PIN)) {
55       digitalWrite(RELAY_DW_PIN, LOW);
56       hasBeenOnSince = millis(); //tempo desde que está ligado
57       state = BUTTON_DW; //atualiza estado
58     }
59   }
60
61   } else { //se estiver a subir verifica se já passou o tempo limite
62     if (state == REMOTE_UP || state == BUTTON_UP){
63       checkTimeOn(RELAY_UP_PIN);
64       //verifica se o botão de subida já foi desligado, nesse caso o estado assa a OFF
65       if (state == BUTTON_UP) {
66         if (debouceSwitch(SWITCH_UP_PIN)) {
67           digitalWrite(RELAY_UP_PIN, HIGH);
68           state = OFF;
69         }
70       }
71       //se estiver a descer verifica se já passou o tempo limite
72       } else if (state == REMOTE_DW || state == BUTTON_DW){
73         checkTimeOn(RELAY_DW_PIN);
74         //verifica se o botão de descida já foi desligado, nesse caso o estado assa a OFF
75         if (state == BUTTON_DW) {
76           if (debouceSwitch(SWITCH_DW_PIN)) {
77             digitalWrite(RELAY_DW_PIN, HIGH);
78             state = OFF;
79           }
80         }
81       }
82       //se o estado é bloqueado (passou o tempo limite um botão ativo)
83     } else if (state == BLOCKED) {
84       //se ambos estiverem desligados atualiza o estado para OFF
85       if (debouceSwitch(SWITCH_UP_PIN) && debouceSwitch(SWITCH_DW_PIN))
86         state = OFF;
87     }
88   }
89 }
90 }
91 }
92 }
93 }
94 }

```

```

85 /*
86  * Função que verifica se o tempo durante o qual o relé está ativo ultrapassou o limite
87  * Nesse caso atualiza o estado dependendo do tipo de pedido de abertura.
88  * Se o pedido foi feito remotamente através da aplicação atualiza para OFF se foi feito
89  * através de um botão o estado passa a bloqueado que só desbloqueará quando o botão que
90  * está a ser premido for desativado */
91 void checkTimeOn(byte relayPin) {
92     if (millis() - hasBeenOnSince > TIME_ON) {
93         digitalWrite(relayPin, HIGH);
94         if (state == REMOTE_UP || state == REMOTE_DW)
95             state = OFF;
96         else
97             state = BLOCKED;
98     }
99 }
100
101 /*
102  * Função que ativa o relé e abre a persiana
103  */
104 void Open() {
105     digitalWrite(RELAY_DW_PIN, HIGH);
106     delay(1);
107     digitalWrite(RELAY_UP_PIN, LOW);
108     state = REMOTE_UP;
109     hasBeenOnSince = millis();
110 }
111
112 /*
113  * Função que ativa o relé e fecha a persiana
114  */
115 void Close() {
116     digitalWrite(RELAY_UP_PIN, HIGH);
117     delay(1);
118     digitalWrite(RELAY_DW_PIN, LOW);
119     state = REMOTE_DW;
120     hasBeenOnSince = millis();
121 }
122
123 /*
124  * Função que no início do programa com base no valor de um potenciômetro define o tempo
125  * limite de subida/descida das persianas em milisegundos
126  */
127
128 unsigned int timeOn(){
129     return round(analogRead(TIME_ON_ANALOG_PIN) * 0.055 + 5) * 1000;
130 }
131
132 /*
133  * Função que recolhe algumas amostras de um pin e retorna o valor que obteve mais vezes
134  * O propósito desta função é filtrar interferências que possa haver e que podem causar um
135  * funcionamento imprevisível
136  */
137 byte debounceSwitch(byte pin){
138     byte count = 0;
139     for(byte i = 0; i<DEBOUNCE_SAMPLES; i++){
140         if(!digitalRead(pin))
141             count++;
142     }
143     if(count>DEBOUNCE_SAMPLES/2) //se mais de metade das amostras é low retorna LOW
144         return LOW;
145     return HIGH; //caso contrário retorna HIGH
146 }

```

```

1 /*Ficheiro MACROS_COMMUNICATION.h
2  * Ficheiro onde são definidas as principais constantes necessárias para a interpretação
3  * das mensagens recebidas.
4  * A maioria deste ficheiro é semelhante aos ficheiros homónimos da aplicação e do código do
5  * dispositivo central
6  */
7
8 #ifndef MACROS_COMMUNICATION_H
9 #define MACROS_COMMUNICATION_H
10
11 //id único de cada dispositivo secundário
12 const unsigned int ID = 428;
13
14 //#define DEBUG
15
16 #define OPEN 'o'
17 #define CLOSE 'c'
18 #define TIME 't'
19 #define INFO 'i'
20 #define DISABLE 'd'
21 #define CONFIRMATION_MESSAGE 'r'
22 #define PAIRING 'p'
23 #define TEST 'T'
24
25 #define CODE 'c'
26
27 #define TRUE 1
28 #define FALSE 0
29
30 #endif

```

```

1  /*Ficheiro communication.h
2  * Header de suporte ao ficheiro communication.cpp em que são definidas várias funções
3  * necessárias para a comunicação através de rf e análise dos comandos
4  */
5
6  #ifndef COMMUNICATION_H
7  #define COMMUNICATION_H
8
9  //----- includes -----
10
11 #include "Arduino.h"
12 #include <VirtualWire.h>
13 /*Autor: Mike McCauley
14 Disponível em: http://www.airspayce.com/mikem/arduino/VirtualWire/
15 */
16 #include "MACROS_COMMUNICATION.h"
17 #include "alarm.h"
18 #include "DiffieHellman.h"
19 #include <TimeLib.h>
20 /*Autores: Michael Margolis
21           Paul Stoffregen
22 Disponível em: https://github.com/PaulStoffregen/Time
23 */
24 #include <AES.h>
25 /*Autores: Brian Gladman
26           Karl Malbrain
27           Mark Tillotson
28 */
29
30 //----- MACROS -----
31
32 #define DEFAULT_BYTE_ARRAY_LENGTH 17
33 #define AES_KEY_LENGTH 128
34 #define DEVICE_COMMUNICATION_TX_PIN 2
35 #define DEVICE_COMMUNICATION_RX_PIN 3
36 #define DEVICE_COMMUNICATION_BAUD_RATE 2000
37 #define RESPONSE_WAIT_TIMEOUT 1000
38
39 #define PAIRING_RESPONSE_WAIT_TIMEOUT 30000
40
41 //----- variáveis globais -----
42
43 //definição dos protótipos das funções no namespace communication
44 namespace communication
45 {
46     void initCommunication();
47
48     byte checkReception(char * cmd);
49     byte checkSum(byte * byteArray, byte pos);
50     void analyseCommand(char * cmd);
51
52     void sendByteArray(byte * byteArray);
53     void sendReceptionConfirmation();
54
55     byte tryHandShakeProtocol();
56     byte getCode(byte * code128bit);
57
58     unsigned int power(int b, byte e);
59 };
60
61 #endif

```



```

1  /*
2   Ficheiro communication.cpp
3   Ficheiro onde estão definidas as funções necessárias para a comunicação através de rf
4   com o dispositivo central, nomeadamente no que diz respeito à receção, descriptação
5   e análise de comandos. Incluir também o algoritmo para o handshake onde é estabelecido o
6   shared secret entre os dois dispositivos aquando do pairing.
7  */
8
9  //incluir o header
10 #include "communication.h"
11
12 //----- variáveis globais -----
13 //instância da classe AES que permitirá descodificar os comandos recebidos
14 AES aes;
15
16 //criar o buffer para a comunicação por rf
17 uint8_t buf[DEFAULT_BYTE_ARRAY_LENGTH];
18 uint8_t buflen = DEFAULT_BYTE_ARRAY_LENGTH;
19
20 byte sharedSecret[16];
21
22 extern struct ALARM alarm;
23
24 /*
25   Função que é chamada no setup e que inicia as comunicações com o dispositivo central
26   e botão de pairing
27 */
28 void communication::initCommunication(){
29   getSharedSecret(); //lê o shared secret da eeprom
30   pinMode(PAIRING_BUTTON_PIN, INPUT_PULLUP); //inicia o botão de pairing
31   aes.set_key(sharedSecret, AES_KEY_LENGTH); //define o shared secret na instacia da classe AES
32   vw_set_tx_pin(DEVICE_COMMUNICATION_TX_PIN); //define os pins para tx e rx dos módulos de rf
33   vw_set_rx_pin(DEVICE_COMMUNICATION_RX_PIN);
34   //vw_set_ptt_pin(DEVICE_COMMUNICATION_PTT_PIN);
35   vw_set_ptt_inverted(true); // Required for DR3100
36   vw_setup(DEVICE_COMMUNICATION_BAUD_RATE); //define a velocidade de transferência de dados
37   vw_rx_start(); //inicia a receção de comandos
38 #ifdef DEBUG
39   Serial.println(F("Device ready"));
40 #endif
41   //define a seed para a geração de numeros aleatórios
42   srand(analogRead(5));
43 }
44
45 /*
46   Função que verifica se algum comando foi recebido, nesse caso retorna TRUE
47   caso contrário retorna FALSE
48   Recebe o apontador para um array de bytes (=char) para o qual copiará o comando
49   recebido
50 */
51 byte communication::checkReception(char * cmd){
52   vw_rx_start();
53   //verifica se neste instante alguma coisa está a ser recebida
54   if (vw_wait_rx_max(50) && vw_get_message(buf, &buflen)) {
55     strcpy(cmd, (char *) buf); //copia os 16 primeiros bytes do buffer para cmd
56     aes.decrypt((byte *)cmd, (byte *)cmd); //descodifica a mensagem recebida
57     return TRUE;
58   }
59   return FALSE;
60 }

```

```

61 /*
62  Por questões de segurança todas as mensagens são enviada com alguns bytes aleatórios
63  cuja soma soma mod 256 tem de ser igual ao ultimo byte do vetor.
64  Esta função trata da verificação desta medida de segurança retornando TRUE se é
65  verificada ou FALSE no caso contrário.
66  Com o intuito que cada comando tenha o maior numero de dígitos aleatórios possível,
67  cada um tem um número diferente deles pelo que esta função recebe o apontador para o
68  byte 5 do comando bem como a posição do primeiro em relação a esse byte
69 */
70 byte communication::checksum(byte * byteArray, byte pos){
71     byte sum = 0; //valor da soma dos aleatórios
72     //itera cada uma das posições desde pos até à posição 15 do vetor ou seja a posição 9          relativa ao
apontador recebido
73     for (byte i = pos; i < DEFAULT_BYTE_ARRAY_LENGTH - 7; i++)
74         sum += byteArray[i];
75     //ao somar continuamente a uma variável de 8 bits quando atinge o limite recomeça do 0 pelo que é
equivalente a calcular o mod 256
76
77     if (sum == byteArray[DEFAULT_BYTE_ARRAY_LENGTH - 7]) //verifica se a soma corresponde
78         return TRUE;
79
80     return FALSE;
81 }
82
83
84 /*
85  Função que analisa o comando recebido verifica que é este o dispositivo para
86  qual o comando foi enviado e executa a respetiva ordem nesse caso.
87  Esta função trata também de enviar a confirmação de receção para o dispositivo
88  central para que o utilizador possa ter a certeza que o comando foi recebido e
89  será executado.
90 */
91 void communication::analyseCommand(char * cmd){
92     unsigned int receivedId = 0; //variável usada para extrair o id do comando
93     for (byte i = 0; i < 5; i++) //itera os primeiros 5 bytes e gera o numero a partir da string
94         receivedId += communication::power(10, 4 - i) * (cmd[i] - '0');
95     if (receivedId == ID) //verifica se o destinatário da mensagem é este dispositivo
96         communication::sendReceptionConfirmation(); //nesse caso envia a confirmação de que a mensagem foi recebida
97     else
98         return; //sai da função
99
100     //cmd passa a apontar para o byte na posição 5 uma vez que já leu os primeiros 5 bytes  referentes ao id
101     cmd += 5;
102     switch (*cmd) { //na posição 5 do vetor esta um caracter que identifica cada uma das ações
103     case OPEN: {
104         if (!communication::checksum((byte *)cmd, 1)) { //verifica checksum
105             #ifdef DEBUG
106                 Serial.println(F("Check sum NOT OK"));
107             #endif
108             break;
109         }
110         #ifdef DEBUG
111             Serial.println(F("OPEN"));
112         #endif
113         Open();
114         break;
115     }
116     case CLOSE: {
117         if (!communication::checksum((byte *)cmd, 1)) { //verifica checksum
118             #ifdef DEBUG
119                 Serial.println(F("Check sum NOT OK"));
120             #endif
121             break;
122         }
123         #ifndef DEBUG
124             Serial.println(F("CLOSE"));
125         #endif
126         Close();
127         break;
128     }
129 }

```

```

128 case TIME: {
129     if (!communication::checksum((byte *)cmd, 5)) { //verifica checksum
130         #ifdef DEBUG
131             Serial.println(F("Check sum NOT OK"));
132         #endif
133         break;
134     }
135     #ifndef DEBUG
136     Serial.println(F("TIME"));
137     #endif
138     //extrai o unsigned long dos 4 bytes a partir da posição 6
139     unsigned long * timeReceived = (unsigned long *) (void *) (cmd + 1);
140     setTime(*timeReceived); //atualiza a hora do sistema
141     break;
142 }
143 case INFO: {
144     if (!communication::checksum((byte *)cmd, 7)) { //verifica checksum
145         #ifdef DEBUG
146             Serial.println(F("Check sum NOT OK"));
147         #endif
148         break;
149     }
150     //atualiza a informação do horário
151     alarm.hourAlarmUp = (byte) cmd[1];
152     alarm.minuteAlarmUp = (byte) cmd[2];
153     alarm.freqUp = cmd[3];
154     alarm.hourAlarmDw = (byte) cmd[4];
155     alarm.minuteAlarmDw = (byte) cmd[5];
156     alarm.freqDw = cmd[6];
157
158     #ifdef DEBUG
159     Serial.println(F("INFO"));
160     showAlarm();
161     #endif
162     break;
163 }
164 case DISABLE: {
165     if (!communication::checksum((byte *)cmd, 1)) { //verifica checksum
166         #ifdef DEBUG
167             Serial.println(F("Check sum NOT OK"));
168         #endif
169         break;
170     }
171     #ifdef DEBUG
172     Serial.println(F("DISABLE"));
173     #endif
174     //desativa alarm
175     alarm.freqUp = DISABLED;
176     alarm.freqDw = DISABLED;
177     break;
178 }
179 default: {
180     #ifdef DEBUG
181     Serial.println(F("NADA (PROVAVELMENTE SO UM TESTE)"));
182     #endif
183 }
184 }
185 }
186

```

```

187 /*
188  * Função que envia a confirmação de que o comando enviado pelo dispositivo central foi
189  * recebido e que será executada a ação correspondente.
190  */
191 void communication::sendReceptionConfirmation()
192 {
193     Serial.println("Sending confirmation");
194     char confirmationMessage[6]; //array de caracteres para a mensagem
195     confirmationMessage[0] = CONFIRMATION_MESSAGE; //primeiro caractere
196     sprintf(confirmationMessage + 1, "%05u", ID); //os restantes bytes são ocupados pelos dígitos do id
197     for(byte i = 0; i<6 ; i++){
198         vw_send((byte *)confirmationMessage, DEFAULT_BYTE_ARRAY_LENGTH); //mandar a mensagem por rf
199         //vw_wait_tx();
200         delay(83);
201     }
202 }
203 }
204
205 /*
206  * Função que é chamada quando quer ser feita uma tentativa de pairing com um dispositivo central,
207  * através de um handshake e troca de chaves secretas através do algoritmo de Diffie_Hellman
208  * Está encarregue de enviar repetidas mensagens até obter uma resposta, a chave publica do
209  * dispositivo principal, ou até se atingir um limite temporal e o pairing ser abortado.
210  * Assim que receber a chave publica do dispositivo principal gera a sua própria chave privada e
211  * a chave publica que envia para o dispositivo central. Seguidamente gera o shared secret com
212  * base na chave que recebeu do dispositivo principal que depois guarda como o
213  * novo shared secret */
214
215 byte communication::tryHandShakeProtocol(){
216     #ifdef DEBUG
217         Serial.println(F("Starting handshake"));
218     #endif
219
220     //variável que permitirá controlar o tempo durante o qual será emitida a mensagem de pairing do dispositivo
221     unsigned long start = millis();
222     byte pairingMessage[6]; //mensagem de emparelhamento
223     byte code128bit[16]; //código privado de 128bits que servirá de base à troca de chaves secretas segundo o
224                                     algoritmo de Diffie_Hellman
225
226     pairingMessage[0] = PAIRING; //o primeiro caracter da mensagem de emparelhamento é 'p'
227     sprintf((char *) (pairingMessage + 1), "%05u", ID); //os seguintes caracteres dizem respeito ao id
228     //equanto não receber uma mensagem a acusar a receção (chave publica do dispositivo central) continua a
229                                     enviar mensagens de emparelhamento até a um certo limite de tempo
230     while (millis() - start < PAIRING_RESPONSE_WAIT_TIMEOUT){
231         vw_send(pairingMessage, DEFAULT_BYTE_ARRAY_LENGTH); //Manda a mensagem de emparelhamento
232         if(communication::getCode(code128bit))//verifica a receção da chave publica do dispositivo central e
233                                     coloca-a no vetor code128bit
234             break; //se receber quebra o ciclo
235     }
236     //se tiverem passado o limite de tempo sem resposta retorna FALSE
237     if (millis() - start >= PAIRING_RESPONSE_WAIT_TIMEOUT){
238         #ifdef DEBUG
239             Serial.println(F("Pairing failed"));
240         #endif
241         return FALSE;
242     }
243
244     initAlarm();//o alarm é desativado
245
246     #ifdef DEBUG
247         Serial.println(F("Main device public key: "));
248         for (byte i = 0; i < 16; i++){
249             Serial.print(code128bit[i], HEX);
250             Serial.print(F(" "));
251             Serial.print(i);
252             Serial.print(F(" "));
253         } Serial.println(F(" "));
254     #endif

```

```

254 //gera aleatoriamente um número de 128 bits (duas variáveis de 64 bits), ie a chave secreta
255 uint64_t devicePrivate1 = randomint64();
256 uint64_t devicePrivate2 = randomint64();
257 //gera a chave pública do dispositivo secundário para enviar
258 uint64_t devicePublic1 = compute(G, devicePrivate1, P);
259 uint64_t devicePublic2 = compute(G, devicePrivate2, P);
260
261 byte codeD[17]; //vetor que servira de suporte ao envio da chave pública.
262 codeD[0] = CODE; //Tem 1 byte inicial para a indicação que a mensagem se trata de um código
263 byte * p = (byte *) &devicePublic1;
264 for (byte i = 0; i < 8; i++) //preenchimento do vetor da mensagem com a chave pública
265     codeD[8 - i] = *(p + i);
266 p = (byte *) &devicePublic2;
267 for (byte i = 0; i < 8; i++)
268     codeD[16 - i] = *(p + i);
269
270 for (byte j = 0; j < 10; j++) {
271     vw_send(codeD, DEFAULT_BYTE_ARRAY_LENGTH);
272     //envia a mensagem 6 vezes para garantir que é recebida pelo dispositivo central
273     delay(50);
274 }
275
276 #ifdef DEBUG
277 Serial.println(F("Public key sent"));
278 /*for (byte i = 1; i <= 16; i++){
279     Serial.print(codeD[i], HEX);
280     Serial.print(F(" "));
281 }Serial.println(F(" "));*/
282 #endif
283
284 //Nas linhas que se seguem é utilizada a variável devicePublic apenas por forma a não utilizar outra
285 //variável, na realidade seria mainPublic
286
287 devicePublic1 = 0;
288 for (byte i = 0; i < 8; i++)
289     devicePublic1 += (uint64_t)code128bit[i] << 8*(7 - i); //extraí a primeira parte do número de 128 bits
290
291 devicePublic2 = 0;
292 for (byte i = 0; i < 8; i++)
293     devicePublic2 += (uint64_t)code128bit[i + 8] << 8*(7 - i); //extraí a segunda parte do número de 128
294 bits
295
296 devicePublic1 = compute(devicePublic1, devicePrivate1, P); //gera a primeira e segunda parte do shared secret
297 devicePublic2 = compute(devicePublic2, devicePrivate2, P);
298
299 //coloca o novo shared secret na variável global
300 p = (byte *) (void *) &devicePublic1;
301 for (byte i = 0; i < 8; i++)
302     sharedSecret[7 - i] = *(p + i);
303 p = (byte *) (void *) &devicePublic2;
304 for (byte i = 0; i < 8; i++)
305     sharedSecret[15 - i] = *(p + i);
306
307 #ifdef DEBUG
308 Serial.println(F("Shared secret: "));
309 /*for (byte i = 0; i < 16; i++) {
310     Serial.print(sharedSecret[i], HEX);
311     Serial.print(F(" "));
312 }
313 Serial.println(F(" "));*/
314 #endif
315
316 saveSharedSecret(); //guarda o sharedSecret em EEPROM
317 aes.set_key(sharedSecret, AES_KEY_LENGTH); //seleciona um novo sharedSecret para a comunicação
318
319 #ifdef DEBUG
320 Serial.println(F("Pairing handshake successful"));
321 #endif
322 return TRUE;
323 }
324

```

```

321 /*
322  * Função que envia um array de bytes e espera pelo seu envio
323  */
324 void communication::sendByteArray(byte * byteArray){
325     vw_send(byteArray, DEFAULT_BYTE_ARRAY_LENGTH); //Envia o array de bytes por rf
326     vw_wait_tx(); //Espera até a mensagem ter sido enviada
327 }
328
329 /*
330  * Função que tenta interceptar a transmissão de um código, recebe um array de bytes no qual inserirá o
331  * código se receber um código
332  */
333 byte communication::getCode(byte * code128bit){
334     vw_rx_start(); //inicia a recepção de rf
335     //espera um determinado tempo pela recepção de uma mensagem
336     if (vw_wait_rx_max(RESPONSE_WAIT_TIMEOUT) && vw_get_message(buf, &buflen)){
337         vw_rx_stop();
338         if (buf[0] == CODE){ //verifica-se a mensagem recebida é um código
339             for (byte i = 1; i <= 16; i++)
340                 *(code128bit + i - 1) = buf[i]; //copia o código para os endereço de memória passado por argumento
341             return TRUE;
342         }
343     }
344     vw_rx_stop();
345     return FALSE;
346 }
347
348 /*
349  * Função que calcula potências de modo iterativo.
350  * uma alternativa à função pow cujo uso de sram é excessivo para números grandes
351  */
352 unsigned int communication::power(int b, byte e)
353 {
354     unsigned int result = 1;
355     for (byte i = 0; i < e; i++)
356         result *= b;
357     return result;
358 }

```

```

1 /*
2  * Ficheiro DiffieHellman.h
3  * Header do ficheiro DiffieHellman.h
4  * Define protótipos de funções e constantes necessária para a geração do shared secret
5  */
6 #ifndef DIFFIEHELLMAN_H
7 #define DEFFIEHELLMAN_H
8 //funções de outros ficheiros usadas
9 #include "Arduino.h"
10 #include "communication.h"
11 #include <EEPROM.h>
12 //Default Arduino library under a Creative Commons Attribution-ShareAlike 3.0 License
13
14 //maior primo de 64 bits
15 #define P 0xffffffffffffffffc5ul
16 #define G 5 //constante para a geração do shared secret
17
18 //inclui sharedSecret, variável global definida em outro documento
19 extern byte sharedSecret[16];
20
21 //----- protótipos -----
22 uint64_t randomint64();
23 uint64_t compute(uint64_t a, uint64_t m, uint64_t n);
24
25 void saveSharedSecret();
26 void getSharedSecret();
27
28 #endif

```

```

1 /*Ficheiro DiffieHellman.cpp
2  * Este ficheiro incorpora as funções necessárias para a geração do sharedSecret,
3  * a chave de 128 bits única para cada par persiana-dispositivo central que é definida
4  * aquando da primeira comunicação entre os dois dispositivos.
5  * Nesta primeira comunicação os dois dispositivos fazem uma troca pública de chaves
6  * baseada no algoritmo de Diffie Hellman.
7  * Contém também funções que guardam esse sharedSeceret em EEPROM, para que após uma falha
8  * de energia seja possível estabelecer comunicação com esta persiana sem ter de fazer
9  * outro pairing.
10 * Desta forma, tendo estabelecido uma chave secreta todas as comunicações podem ser
11 * encriptadas para que seja substancialmente mais difícil o acesso de dispositivos não
12 * autorizadas à persiana. Esta encriptação é feita através do algoritmo AES com uma chave de
128 bits.
13 * Para além desta encriptação todas as mensagens são enviadas com alguns bytes aleatórios
14 * cuja soma mod 256 é igual ao último byte. Assim, é impossível controlar as persianas
15 * através de uma interceção das comunicações e sua réplica.
16 */
17
18
19
20 //incluir o header
21 #include "DiffieHellman.h"
22
23 //Função que retorna gera um número aleatório de 64bits
24 uint64_t randomint64() {
25     //os número gerados pela função rand() têm 16bits pelo que é necessário gerar 4
26     uint64_t a = rand();
27     uint64_t b = rand();
28     uint64_t c = rand();
29     uint64_t d = rand();
30     //gerar um numero de 64 bits inserindo quantro blocos de 16 bits
31     return a << 48 | b << 32 | c << 16 | d;
32 }
33
34 //Função que calcula a^n mod n
35 uint64_t compute(uint64_t a, uint64_t m, uint64_t n)
36 {
37     uint64_t r;
38     uint64_t y = 1;
39     while (m > 0){
40         r = m % 2;
41         if (r == 1)
42             y = (y*a) % n;
43         a = a*a % n;
44         m = m / 2;
45     }
46     return y;
47 }
48
49 //Função que guarda o sharedSecret em EEPROM
50 void saveSharedSecret()
51 {
52     #ifdef DEBUG
53     Serial.println(F("saving shared secret"));
54     #endif
55     /*Uma vex que este valor só será atualizado cada vez que for feita uma tentativa de pairing
56     * é desnecessária a introdução de um algoritmo de ware leveling. Assim, o sharedSecret é
57     * guardado nos primeiros 16 bytes da EEPROM do Arrduino
58     */
59     for(byte i = 0; i<16; i++)
60         EEPROM.update(i, sharedSecret[i]);
61 }
62

```



```
63 //Função que extrai o sharedSecret da EEPROM
64 void getSharedSecret()
65 {
66     #ifdef DEBUG
67     Serial.println(F("Getting sharedSecret from eeprom"));
68     #endif
69     for(byte i = 0; i<16; i++) //extrair os primeiros 16 bytes
70         sharedSecret[i]=EEPROM.read(i);
71 }
```

## Código da Aplicação

Fcheiros:

1. MACROS.h
2. MACROS\_CONNECTION.h
3. MyTableViewController.h
4. MyTableViewController.m
5. deviceData.h
6. deviceData.m
7. deviceViewController.h
8. deviceViewController.m
9. SettingsViewController.h
10. SettingsViewController.m

Todos os ficheiros podem ser consultados em:

[github.com/LeonardoPedroso/DomusSapiens](https://github.com/LeonardoPedroso/DomusSapiens)

```

1 //
2 //  MACROS.h
3 //  DomusSapiens
4 //
5 //  Created by Leonardo Pedroso on 13/01/18.
6 //  Copyright © 2018 Leonardo Pedroso. All rights reserved.
7 //
8
9 #ifndef MACROS_h
10 #define MACROS_h
11
12 #define ID @"i"
13 #define NAME @"n"
14
15 #define HOUR_ALARM_UP @"H"
16 #define MINUTE_ALARM_UP @"M"
17 #define FREQ_ALARM_UP @"F"
18
19 #define HOUR_ALARM_DOWN @"h"
20 #define MINUTE_ALARM_DOWN @"m"
21 #define FREQ_ALARM_DOWN @"f"
22
23 #define LABEL @"l"
24 #define DEFAULT_LABEL @"default"
25 #define LABEL_DEFAULTS @"L"
26
27 #define ROW @"r"
28
29 #define DISABLED @"d"
30 #define EVERYDAY @"e"
31 #define WEEKDAYS @"w"
32 #define WEEKENDS @"W"
33
34 #define EVERYDAY_SHOW @"ED"
35 #define WEEKDAYS_SHOW @"WD"
36 #define WEEKENDS_SHOW @"WE"
37
38 #endif /* MACROS_h */

```

```

1 //
2 //  MACROS_CONNNECTION.h
3 //  DomusSapiens
4 //
5 //  Created by Leonardo Pedroso on 15/01/18.
6 //  Copyright © 2018 Leonardo Pedroso. All rights reserved.
7 //
8
9 #ifndef MACROS_CONNNECTION_h
10 #define MACROS_CONNNECTION_h
11
12 #define IP @"ip"
13 #define PORT @"port"
14 #define PASSWORD @"pass"
15 #define ROW_ORDER @"rowOrder"
16
17 #define OPEN @"o"
18 #define CLOSE @"c"
19
20
21 #define HTTPOK @"h"
22 #define DATA @"D"
23
24 #define PARING_DEVICE @"p"
25 #define DEVICE_DELETE @"d"
26 #define DEVICE_EDIT @"e"
27 #define DEVICE_ADD @"a"
28 #define MAIN_TEST @"T"
29 #define DEVICE_TEST @"t"
30
31 #define END_OF_RECEPTION @"x"
32
33 // #define CONNECTION
34
35 #endif /* MACROS_CONNNECTION_h */

```

```

1 //
2 //  MyTableViewController.h
3 //  DomusSapiens
4 //
5 //  Created by Leonardo Pedroso on 10/01/18.
6 //  Copyright © 2018 Leonardo Pedroso. All rights reserved.
7 //
8
9 #import <UIKit/UIKit.h>
10 #import "MyTableViewCell.h"
11 #import "deviceData.h"
12 #import "deviceViewController.h"
13 #import "MACROS_CONNECTION.h"
14
15 @interface MyTableViewController : UITableViewController <returnDeviceDataDelegate>
16
17 -(void) disconnectedAppearance;
18 -(void) connectedAppearance;
19
20 @property (nonatomic) NSMutableDictionary * data;
21 @property (nonatomic, strong) NSString * connection;
22
23 - (IBAction)editButton:(id)sender;
24 @property (weak, nonatomic) IBOutlet UIBarButtonItem *editButtonOutlet;
25
26 @end

```

```

1 //
2 // MyTableViewController.m
3 // DomusSapiens
4 //
5 // Created by Leonardo Pedrosa on 10/01/18.
6 // Copyright © 2018 Leonardo Pedrosa. All rights reserved.
7 //
8
9 #import "MyTableViewController.h"
10
11 @interface MyTableViewController ()
12
13 //define function
14 #define UIColorFromRGB(rgbValue) [UIColor colorWithRed:((float)((rgbValue & 0xFF0000) >> 16))/255.0
green:((float)((rgbValue & 0xFF00) >> 8))/255.0 blue:((float)(rgbValue & 0xFF))/255.0 alpha:0.7]
15
16 @end
17
18 @implementation MyTableViewController
19
20 - (void)viewDidLoad {
21     [super viewDidLoad];
22     //activate notification center to allow communication between view controllers
23     [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(updateData) name:@"update" object:nil];
24     [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(showConnectionFailMessage:) name:@"failD" object:nil];
25     [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(cancelPairing:) name:@"cancelPairing" object:nil];
26     self.data = [deviceData getDevicesData]; //get devices info
27     if(!_data)
28         [self disconnectedAppearance]; //show disconnected screen
29 }
30
31
32 - (void)didReceiveMemoryWarning {
33     [super didReceiveMemoryWarning];
34     // Dispose of any resources that can be recreated.
35 }
36
37 #pragma mark - notification center
38 - (void)dealloc
39 {
40     //deactivate notification center
41     [[NSNotificationCenter defaultCenter] removeObserver:self name:@"update" object:nil];
42     [[NSNotificationCenter defaultCenter] removeObserver:self name:@"failD" object:nil];
43     //[[super dealloc];
44 }
45
46 //function that updates the data in the tableview
47 - (void)updateData
48 {
49     //get devices info
50     self.data = [deviceData getDevicesData];
51     if(!_data)
52         [self connectedAppearance]; //show connected screen
53     else
54         [self disconnectedAppearance]; //show disconnected screen
55     [self.tableView reloadData]; //reload table data
56 }
57
58 //function which is responsible for showing
59 - (void)showConnectionFailMessage:(NSNotification *) notification
60 {
61     NSDictionary* userInfo = notification.userInfo; //put notification in a dictionary
62     NSString * msg;
63     //if the notification shows the pairing failed
64     if([userInfo[@"M"] isEqualToString:@"PF"]){
65         msg = @"Pairing failed, check the connection and try again.";
66     }else{
67         msg = [NSString stringWithFormat:@"Unable to reach device 'ds%@\nCheck the connection and
68     ]
69     [self alertMsg:msg]; //show alert msg
70 }
71

```

```

72 //function that is called to prevent errors when pairing is aborted
73 - (void)cancelPairing:(NSNotification *) notification
74 {
75     NSLog(@"REMOVE RECENTLY PAIRED");
76     NSDictionary* userInfo = notification.userInfo;
77     //send command to the arduino to delete the device
78     [deviceData sendIdToDelete:userInfo[@"M"]];
79 }
80
81 //function that shows a pop up message with msg
82 -(void>alertMsg:(NSString *)msg
83 {
84     UIAlertController* alert = [UIAlertController alertControllerWithTitle:@"Connection failed"
85                                     message:msg
86                                     preferredStyle:UIAlertControllerStyleAlert];
87
88     UIAlertAction* defaultAction = [UIAlertAction actionWithTitle:@"OK" style:UIAlertActionStyleDefault
89                                     handler:^(UIAlertAction * action) {}];
90
91     [alert addAction:defaultAction];
92     [self presentViewController:alert animated:YES completion:nil];
93 }
94 }
95
96 //function that changes the screen to the disconnected appearance
97 -(void) disconnectedAppearance
98 {
99     [self.navigationController.navigationBar setBarTintColor:[UIColor redColor]];
100    [self.navigationItem.rightBarButtonItem setTintColor:[UIColor clearColor]];
101    [self.navigationItem.leftBarButtonItem setTintColor:[UIColor clearColor]];
102    [self.navigationItem.rightBarButtonItem setEnabled:NO];
103    [self.navigationItem.leftBarButtonItem setEnabled:NO];
104    self.navigationItem.title = @"Disconnected";
105 }
106
107
108
109
110 //function that changes the screen to the connected appearance
111 -(void) connectedAppearance
112 {
113     [self.navigationController.navigationBar setBarTintColor:[UIColorFromRGB(0x3C8994)]];
114     [self.navigationItem.rightBarButtonItem setTintColor:[UIColor whiteColor]];
115     [self.navigationItem.leftBarButtonItem setTintColor:[UIColor whiteColor]];
116     [self.navigationItem.rightBarButtonItem setEnabled:YES];
117     [self.navigationItem.leftBarButtonItem setEnabled:YES];
118     self.navigationItem.title = @"Devices";
119 }
120 #pragma mark - Table view data source
121 //functions for the number of sections and rows
122 - (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
123     return 1;
124 }
125 - (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section {
126     if(!_data)
127         return 0;
128     return [[self.data allKeys] count];
129 }
130

```

```

131 //function which is in charge of organising the list of all the devices connected accordingly to the
132 // order selected by the user which is stored in the phone
133 - (MyTableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {
134
135     MyTableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:@"Cell" forIndexPath:indexPath];
136     for(id key in self.data)
137     {
138         if ([[self.data objectForKeyedSubscript:key][ROW] integerValue] == indexPath.row) {
139             //write device name to teh list
140             cell.nameLabel.text = [self.data objectForKeyedSubscript:key][NAME];
141             //write alarm up to the list if activated
142             if ([[self.data objectForKeyedSubscript:key][FREQ_ALARM_UP] isEqualToString:DISABLED] ||
143                 [self.tableView isEditing])
144             {
145                 cell.timeUpLabel.hidden = TRUE;
146                 cell.UPLabel.hidden = TRUE;
147                 cell.freqUpLabel.hidden = TRUE;
148             }
149             else
150             {
151                 cell.timeUpLabel.hidden = FALSE;
152                 cell.UPLabel.hidden = FALSE;
153                 cell.freqUpLabel.hidden = FALSE;
154                 cell.timeUpLabel.text = [NSString stringWithFormat:@"%02li:%02li",
155                     [[self.data objectForKeyedSubscript:key][HOUR_ALARM_UP] integerValue],
156                     [[self.data objectForKeyedSubscript:key][MINUTE_ALARM_UP] integerValue]];
157                 if ([[self.data objectForKeyedSubscript:key][FREQ_ALARM_UP] isEqualToString:EVERYDAY])
158                     cell.freqUpLabel.text = EVERYDAY_SHOW;
159                 else if ([[self.data objectForKeyedSubscript:key][FREQ_ALARM_UP] isEqualToString:WEEKDAYS])
160                     cell.freqUpLabel.text = WEEKDAYS_SHOW;
161                 else
162                     cell.freqUpLabel.text = WEEKENDS_SHOW;
163             }
164             //write alarm down to the list if activated
165             if ([[self.data objectForKeyedSubscript:key][FREQ_ALARM_DOWN] isEqualToString:DISABLED] ||
166                 [self.tableView isEditing])
167             {
168                 cell.timeDwLabel.hidden = TRUE;
169                 cell.DOWNLabel.hidden = TRUE;
170                 cell.freqDwLabel.hidden = TRUE;
171             }
172             else
173             {
174                 cell.timeDwLabel.hidden = FALSE;
175                 cell.DOWNLabel.hidden = FALSE;
176                 cell.freqDwLabel.hidden = FALSE;
177                 cell.timeDwLabel.text = [NSString stringWithFormat:@"%02li:%02li",
178                     [[self.data objectForKeyedSubscript:key][HOUR_ALARM_DOWN] integerValue], [[self.data
179                     objectForKeyedSubscript:key][MINUTE_ALARM_DOWN] integerValue]];
180                 if ([[self.data objectForKeyedSubscript:key][FREQ_ALARM_DOWN] isEqualToString:EVERYDAY]){
181                     cell.freqDwLabel.text = EVERYDAY_SHOW;
182                 }
183                 else if ([[self.data objectForKeyedSubscript:key][FREQ_ALARM_DOWN] isEqualToString:WEEKDAYS]){
184                     cell.freqDwLabel.text = WEEKDAYS_SHOW;
185                 }
186                 else{
187                     cell.freqDwLabel.text = WEEKENDS_SHOW;
188                 }
189             }
190         }
191     }
192     return cell;
193 }

```



```

190 //called when deleting a device
191 - (void)tableView:(UITableView *)tableView commitEditingStyle:(UITableViewCellEditingStyle)editingStyle
forRowAtIndexPath:(NSIndexPath *)indexPath{
192
193     if (editingStyle == UITableViewCellEditingStyleDelete) {
194         // Delete the row from the data source
195         for (id key in _data) {
196             if ([_data[key][ROW] isEqualToNumber:[NSNumber numberWithInt:indexPath.row]]) {
197                 if (![deviceData sendIdToDelete:_data[key][ID]])
198                 {
199                     NSString * msg = [NSString stringWithFormat:@"Unable to reach device
200                     [self alertMsg:msg];
201                     return;
202                 }
203                 [_data removeObjectForKey:key];
204                 break;
205             }
206         }
207         for (id key in _data) {
208             if ([_data[key][ROW] integerValue] > indexPath.row) {
209                 NSNumber * aux = [NSNumber numberWithInt:([_data[key][ROW] integerValue] - 1)];
210                 [_data[key] removeObjectForKey:ROW];
211                 [_data[key] setValue:aux forKey:ROW];
212             }
213         }
214
215         [deviceData saveRowOrder:_data];
216         [tableView deleteRowsAtIndexPaths:@[indexPath] withRowAnimation:UITableViewRowAnimationFade];
217         [tableView reloadData];
218     }
219 }
220
221 //function to readjust list order and save it to the phone's storage
222 - (void)tableView:(UITableView *)tableView moveRowAtIndexPath:(NSIndexPath *)fromIndexPath
223
224     id from = nil;
225     id to = nil;
226     for (id key in _data){
227         if ([_data[key][ROW] isEqualToNumber:[NSNumber numberWithInt:fromIndexPath.row]]) {
228             from = key;
229         }else if([_data[key][ROW] isEqualToNumber:[NSNumber numberWithInt:toIndexPath.row]]){
230             to = key;
231         }
232     }
233     NSNumber * aux = _data[from][ROW];
234     [_data[from] removeObjectForKey:ROW];
235     [_data[from] setValue:_data[to][ROW] forKey:ROW];
236     [_data[to] removeObjectForKey:ROW];
237     [_data[to] setValue:aux forKey:ROW];
238     [tableView reloadData];
239     [deviceData saveRowOrder:_data];
240 }
241
242 - (BOOL)tableView:(UITableView *)tableView canMoveRowAtIndexPath:(NSIndexPath *)indexPath {
243     return YES;
244 }
245

```

```

248 //preparation before navigation
249 - (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
250     if ([sender isKindOfClass:[UITableViewCell class]] && [segue.destinationViewController
251         {
252         deviceViewController * nextDeviceView = segue.destinationViewController;
253         NSIndexPath * path = [self.tableView indexPathForCell:sender];
254
255         for (id key in self.data){
256             if ([self.data objectForKeyedSubscript:key][ROW] integerValue] == path.row){
257                 nextDeviceView.deviceData = [self.data objectForKey:key];
258                 break;
259             }
260         }
261         nextDeviceView.delegate = self;
262     }
263     else if([sender isKindOfClass:[UIBarButtonItem class]] && [segue.destinationViewController
264         deviceViewController * nextDeviceView = segue.destinationViewController;
265         nextDeviceView.deviceData = nil;
266         nextDeviceView.delegate = self;
267     }
268 }
269 }
271 #pragma mark - return data delegate
272 //adds new or edited device to the dictionary data
273 -(void)returnData:(NSMutableDictionary *)deviceDataDict
274 {
275     BOOL exists = FALSE;
276     for(id key in _data){
277         if ([_data[key][ID] isEqualToString:deviceDataDict[ID]]) {
278             [deviceDataDict removeObjectForKey:ROW];
279             [deviceDataDict setObject:_data[key][ROW] forKey:ROW];
280             [self.data removeObjectForKey:key];
281             [self.data setObject:deviceDataDict forKey:key];
282             exists = TRUE;
283             if(![deviceData sendEditedDevice:_data[key]]){
284                 //erro
285             }
286             break;
287         }
288     }
289     if(!exists){
290         [deviceDataDict setValue:[NSNumber numberWithInt:[[_data allKeys] count]] forKey:ROW];
291         [self.data setObject:deviceDataDict forKey:deviceDataDict[ID]];
292         /*if(![deviceData sendNewDevice:_data[deviceDataDict[ID]]){
293             //erro
294         }*/
295         if(![deviceData sendEditedDevice:_data[deviceDataDict[ID]]){
296             //erro
297         }
298     }
299     [deviceData saveLabelDefaults:_data];
300     [deviceData saveRowOrder:_data];
301     [self.navigationController popViewControllerAnimated:YES];
302     [self.tableView reloadData];
303 }
304
305 //change button from 'edit' to 'OK' and vice-versa
306 - (IBAction)editButton:(id)sender {
307
308     if ([self.tableView isEditing]) {
309         [self.tableView setEditing:NO animated:YES];
310         [self.editButtonOutlet setTitle:@"Edit"];
311     }
312     else{
313         [self.tableView setEditing:YES animated:YES];
314         [self.editButtonOutlet setTitle:@"OK"];
315     }
316     [self.tableView reloadData];
317
318 }
319
320 @end

```

```

1 //
2 //  deviceData.h
3 //  DomusSapiens
4 //
5 //  Created by Leonardo Pedroso on 11/01/18.
6 //  Copyright © 2018 Leonardo Pedroso. All rights reserved.
7 //
8
9 #import <Foundation/Foundation.h>
10 #import "MACROS_DATA.h"
11 #import "MACROS_CONNECTION.h"
12
13 @interface deviceData : NSObject
14
15 #pragma mark - testing
16 +(BOOL) testConnectionMain;
17 +(int) testConnectionMainFull;
18 +(BOOL) testConnectionDevice: (NSString *)deviceId;
19
20 #pragma mark - commands with object response
21 +(NSString *) getParingDevice;
22 +(NSMutableDictionary *) getDevicesData;
23
24 #pragma mark - commands with HTTP OK response
25 +(BOOL) sendCommandOpenCloseDirection:(NSString *)direction id:(NSString *)deviceId;
26
27 +(BOOL) sendEditedDevice:(NSMutableDictionary *)deviceEditedData;
28 +(BOOL) sendNewDevice:(NSMutableDictionary *)deviceNewData;
29 +(BOOL) sendIdToDelete:(NSString *)deviceId;
30
31 #pragma mark - manage row order
32 +(void)atributeRows: (NSMutableDictionary *) data;
33 +(void)saveRowOrder: (NSMutableDictionary *) data;
34 +(void)atributeLabels: (NSMutableDictionary *) data;
35 +(void)saveLabelDefaults: (NSMutableDictionary *) data;
36
37 +(BOOL) checkInputData:(NSString *) str;
38
39 +(void) alertDevice;
40
41
42 @end

```

```

1 //
2 //  deviceData.m
3 //  Domus Sapiens
4 //
5 //  Created by Leonardo Pedroso on 11/01/18.
6 //  Copyright © 2018 Leonardo Pedroso. All rights reserved.
7 //
8
9 #import "deviceData.h"
10
11
12 @interface deviceData()
13 {
14
15 }
16
17 //private class method propotypes
18 +(NSString*) getJsonStringFromDict: (NSMutableDictionary *) dict;
19 +(NSString*) getUrl;
20 +(BOOL) sendCommandWithString:(NSString *)msg;
21 +(NSDictionary *) getJsonDictfromCommand: (NSString *)msg;
22
23 @end
24
25 @implementation deviceData
26
27 /*----- testing -----*/
28
29 /*This method sends a request for main device asking to check the connection to each of the apired devices and
30 return how many of them are available throuh JSON {"tma":#} which is returned*/
31 +(int) testConnectionMainFull;
32 {
33     NSDictionary * received = [deviceData getJsonDictfromCommand:MAIN_TEST];
34     NSLog(@"CONNECTION MAIN DATAAAAA");
35     if (received == nil) { //check if something was received
36         return -1;
37     }
38     return (int)[received[HTTPOK] integerValue]; //return the number of avable devices
39 }
40
41 /*----- commands asking for object response -----*/
42 /*This method sends a resquest asking for the id of any deviced which is trying to be paired, which is returned
43 Receives JSON in the form {"par":"dsxxxxxx"}*/
44 +(NSString *) getParingDevice
45 {
46     NSDictionary * received = [deviceData getJsonDictfromCommand:PARING_DEVICE]; //get the data
47     if (received == nil) { //check if something was received
48         return nil;
49     }
50     NSLog(@"paring");
51     NSLog(@"%@", received);
52     return received[PARING_DEVICE]; //return
53 }
54

```

```

55 /*This method sends a request asking for the data of all the devices, which returns via a mutable dictionary*/
56 +(NSMutableDictionary *) getDevicesData
57 {
58     #ifndef CONNECTION //debugging purposes
59     NSMutableDictionary * json = [[NSMutableDictionary alloc] init];
60     NSData *data;
61     NSString *url_string;
62
63
64     int i = 1;
65     while(TRUE)
66     {
67         url_string = [NSString stringWithFormat:@"%%%@", [deviceData getUrl],
68                     [NSString stringWithFormat:@"%02i", DATA, i]]; //setup url
69         NSURLResponse* urlResponse;
70         NSError* error;
71         NSMutableURLRequest* urlRequest = [NSMutableURLRequest requestWithURL:[NSURL URLWithString:url_string]
72 cachePolicy:NSURLRequestReloadIgnoringCacheData timeoutInterval:10];
73         data = [NSURLConnection sendSynchronousRequest:urlRequest returningResponse:&urlResponse error:&error];
74         //data = [NSData dataWithContentsOfURL:[NSURL URLWithString:url_string]]; //get data
75         if(error){
76             NSLog(@"ERROR GETTING DATA");
77             return nil;
78         }
79         id subJson = [NSJSONSerialization JSONObjectWithData:data options:NSJSONReadingMutableContainers error:nil];
80
81         NSLog(@"%@", subJson);
82
83         if ([subJson objectForKey:END_OF_RECEPTION])
84             break;
85         else{
86             NSString * name = subJson[NAME];
87             NSLog(@"%@", subJson);
88             for (NSInteger charIdx=0; charIdx<name.length; charIdx++){
89                 if ([name characterAtIndex:charIdx] == '&')
90                     name = [name stringByReplacingCharactersInRange:NSMakeRange(charIdx, 1) withString:
91 [NSString stringWithFormat:@"%c", ' ']];
92             }
93             [subJson removeObjectForKey:NAME];
94             [subJson setObject:name forKey:NAME];
95             [json setObject:subJson forKey:[subJson objectForKey:ID]];
96         }
97         i++;
98     }
99
100     #else
101     NSString * jsonString
102     =@"{"0000":{"H":20,"M":2,"h":21,"m":2,"F":"w","f":"d","n":"Bedroom","i":"0000","l":"Ci
103 ma"},"0001":{"H":10,"M":24,"h":4,"m":35,"F":"w","f":"e","n":"Garage","i":"0001","l":"
104 Baixo"}}";
105     NSData *data = [jsonString dataUsingEncoding:NSUTF8StringEncoding];
106     id json = [NSJSONSerialization JSONObjectWithData:data options:NSJSONReadingMutableContainers error:nil];
107     #endif
108
109     for (id key in json) {
110         [json[key] setObject:@-1 forKey:ROW];
111     }
112
113     for (id key in json) {
114         [json[key] setObject:DEFAULT_LABEL forKey:LABEL];
115     }
116
117     NSLog(@"%@", json);
118
119     /*by default the rows are set to -1 in all the devices.
120     This allows each user of the same Domus Sapiens system to have different configurations of the order in
121     which the devices are presented. Therefore, a method that attributes row based on previous data is required.*/
122     [deviceData attributeRows:json];
123     [deviceData attributeLabels:json];
124     return json;
125 }
126
127
128
129

```

```

141 /*----- commands asking for object response -----*/
142
143 /*main method which sends command and waits for HTTPOK which is returned*/
144 +(BOOL) sendCommandWithString: (NSString *)msg
145 {
146     #ifndef CONNECTION //debugging purposes
147         NSDictionary * received = [deviceData getJsonDictfromCommand:msg];
148         if (received == nil) {
149             return 0;
150         }
151         return [received[HTTPOK] integerValue];
152     #endif
153
154     NSLog(@"Sending %@", msg);
155     NSLog(@"Successfully sent");
156     return TRUE;
157 }
158
159 /*teste cnnction to main device*/
160 +(BOOL) testConnectionMain;
161 {
162     return [deviceData sendCommandWithString:MAIN_TEST];
163 }
164
165 /*send commands to open or close a certain device*/
166 +(BOOL) sendCommandOpenCloseDirection:(NSString *)direction id:(NSString *)deviceId
167 {
168     return [deviceData sendCommandWithString:[NSString stringWithFormat:@"%s%s", direction , deviceId]];
169 }
170
171 /*send JSON regarding edited device*/
172 +(BOOL) sendEditedDevice:(NSMutableDictionary *)deviceEditedData;
173 {
174     NSString * dataToSend = [deviceData getJsonStringFromDict:deviceEditedData];
175     return [deviceData sendCommandWithString:[NSString stringWithFormat:@"%s%s", DEVICE_EDIT, dataToSend]];
176 }
177 /*send JSON regarding new device*/
178 +(BOOL) sendNewDevice:(NSMutableDictionary *)deviceNewData;
179 {
180     NSString * dataToSend = [deviceData getJsonStringFromDict:deviceNewData];
181     return [deviceData sendCommandWithString:[NSString stringWithFormat:@"%s%s", DEVICE_ADD, dataToSend]];
182 }
183 /*send id of device to delete*/
184 +(BOOL) sendIdToDelete:(NSString *)deviceId
185 {
186     return [deviceData sendCommandWithString:[NSString stringWithFormat:@"%s%s", DEVICE_DELETE, deviceId]];
187 }
188
189 /*test connection with a particular device*/
190 +(BOOL) testConnectionDevice: (NSString *)deviceId
191 {
192     return [deviceData sendCommandWithString:[NSString stringWithFormat:@"%s%s", DEVICE_TEST, deviceId]];
193 }
194
195
196 /*----- auxiliary methods -----*/
197
198 /*method that returns url base on the connection entered by the user */
199 +(NSString *)getUrl
200 {
201     NSUserDefaults *connection = [NSUserDefaults standardUserDefaults]; //opens user defaults
202     NSString *url;
203     NSString * ip = [connection objectForKey:IP];
204
205     if ([connection objectForKey:PORT] isEqualToString:@""]) { //check whether port forwarding is active
206         url = [NSString stringWithFormat:@"http://%s", ip, [connection objectForKey:PASSWORD]];
207     }else{
208         url = [NSString stringWithFormat:@"http://%s:%s", ip, [connection objectForKey:PORT],
209             [connection objectForKey:PASSWORD]];
210     }
211     return url;
212 }

```

```

213
214
222 /*method that returns string equivalent to dictionary in JSON notation*/
223 +(NSString*) getJsonStringFromDict:(NSMutableDictionary *) dict
224 {
225
226     NSString * str = [NSString stringWithFormat:@"%02li%02li%02li%02li%", dict[ID], dict[NAME],
[dict[HOURL_ALARM_UP] integerValue], [dict[MINUTE_ALARM_UP] integerValue], dict[FREQ_ALARM_UP], [dict[HOURL_ALARM_DOWN]
integerValue], [dict[MINUTE_ALARM_DOWN] integerValue], dict[FREQ_ALARM_DOWN]];
228     NSLog(@"%@", str);
229     // NSError *error;
230     // //data parsing
231
232     //
233     // if (! jsonData) { //check for an error
234     //     NSLog(@"bv_jsonStringWithPrettyPrint: error: %@", error.localizedDescription);
235     //     return @"{}";
236     // } else {
237     //     return [[NSString alloc] initWithData:jsonData encoding:NSUTF8StringEncoding];
238     // }
239
240     return str;
241 }
242
243
244 /*method which is responsible to attribute rows to all the devices paired, based on previous user configurations*/
245 +(void)attributeRows: (NSMutableDictionary *) data
246 {
247     NSUserDefaults *defs = [NSUserDefaults standardUserDefaults];
248     //dictionary may not exist
249     if ([defs objectForKey:ROW_ORDER] == NULL) {
250         [defs setObject:[NSMutableDictionary alloc] init] forKey:ROW_ORDER];
251     }
252     //delete inexistent keys and attribute row to those already defined
253     BOOL found = FALSE;
254     int nFound = 0;
255     for (id key in [defs objectForKey:ROW_ORDER]){
256         found = FALSE;
257         for (id keyData in data) {
258             //NSLog(@"%@", [keyData s]);
259             if ([key isEqualToString: keyData]) {
260                 data[keyData][ROW] = [defs objectForKey:ROW_ORDER][key];
261                 nFound++;
262                 found = TRUE;
263                 break;
264             }
265         }
266         if(!found){
267             //[[defs objectForKey:ROW_ORDER] removeObjectForKey:key];
268         }
269     }
270     //order values for rows
271     found = FALSE;
272     for (int i = 0; i < nFound ; i++) {
273         found = FALSE;
274         for (id key in data) {
275             if ([data[key][ROW] integerValue] == i) {
276                 found = TRUE;
277                 break;
278             }
279         }
280         if(!found){ //if number i was not found
281             for (id key in data) {
282                 if ([data[key][ROW] integerValue] > i) {
283                     int aux = (int)([data[key][ROW] integerValue] - 1);
284                     [data[key] removeObjectForKey:ROW];
285                     [data[key] setObject:[NSNumber numberWithInt:aux] forKeyedSubscript:ROW];
286                 }
287             }
288             i--;
289         }
290     }

```

```

291
292 //attribute row for the ones that stil are -1
293 for (id key in data) {
294     if ([data[key][ROW] integerValue] < 0) {
295         [data[key] removeObjectForKey:ROW];
296         [data[key] setObject:[NSNumber numberWithInt:nFound] forKey:ROW];
297         nFound++;
298     }
299 }
300
301 //save configuration
302 [deviceData saveRowOrder:data];
303 }
304
305
306
307 /*method in charge of saving current configuration of the rows of the user*/
308 +(void)saveRowOrder: (NSMutableDictionary *) data
309 {
310     NSUserDefaults *defs = [NSUserDefaults standardUserDefaults]; //access user defaults
311     [defs removeObjectForKey:ROW_ORDER]; //delete previous configuration
312
313     NSMutableDictionary * rows =[[NSMutableDictionary alloc] init]; //setup and add configuration
314     for (id key in data) {
315         [rows setObject:data[key][ROW] forKey:key];
316     }
317     [defs setObject:rows forKey:ROW_ORDER];
318     [defs synchronize]; //synch data
319 }
320
321
322
323 +(void)attributeLabels: (NSMutableDictionary *) data
324 {
325
326     NSUserDefaults *defs = [NSUserDefaults standardUserDefaults];
327     //dictionary may not exist
328     if ([defs objectForKey:LABEL_DEFAULTS] == NULL) {
329         [defs setObject:[NSMutableDictionary alloc] init] forKey:LABEL_DEFAULTS];
330     }
331     //delete inexistent keys and attribute row to those already defined
332     BOOL found = FALSE;
333     int nFound = 0;
334     for (id key in [defs objectForKey:LABEL_DEFAULTS]){
335         found = FALSE;
336         for (id keyData in data) {
337             NSLog(@"%@", [keyData s]);
338             if ([key isEqualToString: keyData]) {
339                 data[keyData][LABEL] = [defs objectForKey:LABEL_DEFAULTS][key];
340                 nFound++;
341                 found = TRUE;
342                 break;
343             }
344         }
345         if(!found){
346             [[[defs objectForKey:LABEL_DEFAULTS] removeObjectForKey:key];
347         }
348     }
349 }
350
351
352
353
354
355 //save configuration
356 [deviceData saveLabelDefaults:data];
357 }
358
359
360
361
362
363 }
364

```



```

365 +(void)saveLabelDefaults: (NSMutableDictionary *) data
366 {
367     NSUserDefaults *defs = [NSUserDefaults standardUserDefaults]; //access user defaults
368     [defs removeObjectForKey:LABEL_DEFAULTS]; //delete previous configuration
369
370     NSMutableDictionary * labels =[[NSMutableDictionary alloc] init]; //setup and add configuration
371     for (id key in data) {
372         [labels setObject:data[key][LABEL] forKey:key];
373     }
374     [defs setObject:labels forKey:LABEL_DEFAULTS];
375     [defs synchronize]; //synch data
376 }
377
378 +(BOOL) checkInputData:(NSString *) str
379 {
380     for (NSInteger charIdx=0; charIdx<str.length; charIdx++){
381         char ch = [str characterAtIndex:charIdx];
382         if ( !(ch == ' ' || (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z')) ) {
383             return FALSE;
384         }
385     }
386     return TRUE;
387 }
388
389 /* method that sends a command to the main device and waits for a JSON response which is parsed into a dictionary to be returned*/
390 +(NSDictionary *) getJsonDictfromCommand:(NSString *) msg
391 {
392     for (NSInteger charIdx=0; charIdx<msg.length; charIdx++){
393         char ch = [msg characterAtIndex:charIdx];
394         if (ch == ' ')
395             msg = [msg stringByReplacingCharactersInRange:NSMakeRange(charIdx, 1) withString:[NSString stringWithFormat:@"%c", '&']];
396     }
397     NSMutableDictionary * json = [[NSMutableDictionary alloc] init];
398     NSString *url_string = [NSString stringWithFormat:@"%s", [deviceData getUrl],msg]; //setup url
399     NSURLResponse* urlResponse;
400     NSError* error;
401     NSMutableURLRequest* urlRequest = [NSMutableURLRequest requestWithURL:[NSURL URLWithString:url_string]
402     cachePolicy:NSURLRequestReloadIgnoringCacheData timeoutInterval:10];
403     NSData* data = [NSURLConnection sendSynchronousRequest:urlRequest returningResponse:&urlResponse
404     error:&error];
405
406     NSLog(@"url %@", url_string);
407     NSLog(@"data %@", data);
408     if (data == nil) { //check reception
409         return nil;
410     }
411     json = [NSJSONSerialization JSONObjectWithData:data options:NSJSONReadingMutableContainers error:nil];
412     NSLog(@"json %@", json);
413
414     return json;
415 }
416 @end

```

```

1 //
2 //  deviceViewController.h
3 //  TableView
4 //
5 //  Created by Leonardo Pedrosa on 12/01/18.
6 //  Copyright © 2018 Leonardo Pedrosa. All rights reserved.
7 //
8
9 #import <UIKit/UIKit.h>
10 #import "MACROS_DATA.h"
11 #import "deviceData.h"
12 #import "MACROS_CONNECTION.h"
13
14 @protocol returnDeviceDataDelegate <NSObject>
15
16 @required
17 -(void)returnData: (NSMutableDictionary *) deviceDataDict;
18
19 @end
20
21
22 @interface deviceViewController : UIViewController <UIPickerViewDataSource, UIPickerViewDelegate>
23
24 //delegate
25 @property (weak, nonatomic) id <returnDeviceDataDelegate> delegate;
26
27 //data
28 @property (nonatomic, strong) NSMutableDictionary * deviceData;
29
30 //OK button
31 - (IBAction)doneButton:(UIBarButtonItem *)sender;
32
33 //text field
34 @property (weak, nonatomic) IBOutlet UITextField *deviceNameTextFieldText;
35 @property (weak, nonatomic) IBOutlet UILabel *invalidNameLabel;
36 - (IBAction)deviceNameTextField:(id)sender;
37 @property (weak, nonatomic) IBOutlet UINavigationController *titleNavBar;
38
39
40 //buttons open close
41 - (IBAction)openButton:(UIButton *)sender;
42 - (IBAction)closeButton:(UIButton *)sender;
43 @property (weak, nonatomic) IBOutlet UIButton *openButtonOutlet;
44 @property (weak, nonatomic) IBOutlet UIButton *closeButtonOutlet;
45
46 //alarm up
47 @property (weak, nonatomic) IBOutlet UISwitch *disableAlarmUpState;
48 - (IBAction)disableAlarmUp:(id)sender;
49 @property (weak, nonatomic) IBOutlet UIPickerView *alarmPickerUp;
50
51 //alarm down
52 @property (weak, nonatomic) IBOutlet UISwitch *disableAlarmDwState;
53 - (IBAction)disableAlarmDw:(id)sender;
54 @property (weak, nonatomic) IBOutlet UIPickerView *alarmPickerDw;
55
56 //label
57 @property (weak, nonatomic) IBOutlet UITextField *labelTextField;
58 @property (weak, nonatomic) IBOutlet UILabel *invalidLabel;
59 - (IBAction)labelTextFieldAction:(id)sender;
60
61 //id
62 @property (weak, nonatomic) IBOutlet UILabel *idLabel;
63 @property (weak, nonatomic) IBOutlet UILabel *onlineLabel;
64
65
66 @end

```

```

1 //
2 // deviceViewController.m
3 // DomusSapiens
4 //
5 // Created by Leonardo Pedrosa on 12/01/18.
6 // Copyright © 2018 Leonardo Pedrosa. All rights reserved.
7 //
8
9 #import "deviceViewController.h"
10
11 @interface deviceViewController ()
12 {
13     NSArray *_pickerData;
14     NSMutableArray *_pickerData0;
15     NSMutableArray *_pickerData1;
16     NSArray *_pickerData2;
17
18     BOOL dataToBeReturned;
19     NSString * deviceID;
20 }
21 @end
22
23 @implementation deviceViewController
24
25 - (void)viewDidLoad {
26     [super viewDidLoad];
27     // Do any additional setup after loading the view.
28
29     //make keyboard disappear
30     UITapGestureRecognizer *tap = [[UITapGestureRecognizer alloc] initWithTarget:self action:@selector(dismissKeyboard)];
31     [self.view addGestureRecognizer:tap];
32
33     [_deviceNameTextFieldText resignFirstResponder];
34     dataToBeReturned = FALSE;
35     //appearance
36     [self.navigationController.navigationBar setTintColor:[UIColor whiteColor]];
37
38
39     if (_deviceData != nil) {
40         _labelTextField.text = _deviceData[LABEL];
41         _idLabel.text = [NSString stringWithFormat:@"ID: ds%@", _deviceData[ID]];
42
43         if ([deviceData testConnectionDevice:_deviceData[ID]]) {
44             _onlineLabel.text = @"Online";
45             [_onlineLabel setTextColor:[UIColor greenColor]];
46         }else{
47             _onlineLabel.text = @"Offline";
48             [_onlineLabel setTextColor:[UIColor redColor]];
49
50             NSDictionary* userInfo = @{@"M": [_deviceData objectForKey:ID]};
51             NSNotificationCenter* nc = [NSNotificationCenter defaultCenter];
52             [self.navigationController popViewControllerAnimated:YES];
53             [nc postNotificationName:@"failD" object:self userInfo:userInfo];
54
55         }
56     }
57     else
58     {
59         //verificar se esta algum a emparelhar
60         deviceID = [deviceData getParingDevice];
61         _idLabel.text = [NSString stringWithFormat:@"ID: ds%@", deviceID];
62
63         if ([deviceData testConnectionDevice:deviceID]) {
64             _onlineLabel.text = @"Online";
65             [_onlineLabel setTextColor:[UIColor greenColor]];
66         }else{
67             _onlineLabel.text = @"Offline";
68             [_onlineLabel setTextColor:[UIColor redColor]];
69             NSDictionary* userInfo = @{@"M": @"PF"};
70             NSNotificationCenter* nc = [NSNotificationCenter defaultCenter];
71             [self.navigationController popViewControllerAnimated:YES];
72             [nc postNotificationName:@"failD" object:self userInfo:userInfo];
73

```

```

74     }
75
76     _labelTextField.text = @"default";
77
78     [_openButtonOutlet setEnabled:FALSE];
79     [_openButtonOutlet setAlpha:0.5];
80     [_closeButtonOutlet setEnabled:FALSE];
81     [_closeButtonOutlet setAlpha:0.5];
82 }
83 //pickerdata
84 // Initialize Data
85 _pickerData0 = [[NSMutableArray alloc] initWithCapacity:60];
86 _pickerData1 = [[NSMutableArray alloc] initWithCapacity:24];
87 _pickerData2 = @[@"ED", @"WD", @"WE"];
88 for (int i = 0; i < 60; i++) {
89     _pickerData1[i] = [NSString stringWithFormat:@"%02i", i];
90 }
91 for (int i = 0; i < 24; i++) {
92     _pickerData0[i] = [NSString stringWithFormat:@"%02i", i];
93 }
94 _pickerData = [[NSArray alloc] initWithObjects:_pickerData0, _pickerData1, _pickerData2, nil];
95 // Connect data
96 self.alarmPickerUp.dataSource = self;
97 self.alarmPickerUp.delegate = self;
98 self.alarmPickerDw.dataSource = self;
99 self.alarmPickerDw.delegate = self;
100
101 /*-----inicial conditions-----*/
102
103 //name
104 [_invalidNameLabel setHidden:TRUE];
105 if (_deviceData != nil) {
106     _titleNavBar.title = _deviceData[NAME];
107     _deviceNameTextFieldText.text = _deviceData[NAME];
108 }else{
109     _titleNavBar.title = @"New device";
110     _deviceNameTextFieldText.text = nil;
111 }
112
113 //Alarm Up
114 int k = 0;
115 if (_deviceData != nil) {
116     if ([_deviceData[FREQ_ALARM_UP] isEqualToString:WEEKDAYS]) {
117         k=1;
118     }else if ([_deviceData[FREQ_ALARM_UP] isEqualToString:WEEKENDS]){
119         k=2;
120     }
121
122     [self.alarmPickerUp selectRow:_deviceData[HOURL_ALARM_UP] integerValue inComponent:0 animated:YES];
123     [self.alarmPickerUp selectRow:_deviceData[MINUTE_ALARM_UP] integerValue inComponent:1 animated:YES];
124 }else{
125     NSDate *currentTime = [NSDate date];
126     NSDateFormatter *dateFormatter = [[NSDateFormatter alloc] init];
127     [dateFormatter setDateFormat:@"hh"];
128     [self.alarmPickerUp selectRow:[dateFormatter stringFromDate: currentTime] integerValue inComponent:0 animated:YES];
129     [dateFormatter setDateFormat:@"mm"];
130     [self.alarmPickerUp selectRow:[dateFormatter stringFromDate: currentTime] integerValue inComponent:1 animated:YES];
131 }
132
133 [self.alarmPickerUp selectRow:k inComponent:2 animated:YES];
134 if ([_deviceData[FREQ_ALARM_UP] isEqualToString:DISABLED]) {
135     [_disableAlarmUpState setOn:FALSE animated:TRUE];
136     [_alarmPickerUp setUserInteractionEnabled:FALSE];
137     [_alarmPickerUp setAlpha:0.6 ];
138 }
139

```

```

152 //Alarm Down
153 k = 0;
154 if(_deviceData != nil){
155     if ([_deviceData[FREQ_ALARM_DOWN] isEqualToString:WEEKDAYS]) {
156         k=1;
157     }else if([_deviceData[FREQ_ALARM_DOWN] isEqualToString:WEEKENDS]){
158         k=2;
159     }
160     [self.alarmPickerDw selectRow:[_deviceData[HOOR_ALARM_DOWN] integerValue] inComponent:0 animated:YES];
161     [self.alarmPickerDw selectRow:[_deviceData[MINUTE_ALARM_DOWN] integerValue] inComponent:1 animated:YES];
162 }else{
163     NSDate *currentTime = [NSDate date];
164     NSDateFormatter *dateFormatter = [[NSDateFormatter alloc] init];
165     [dateFormatter setDateFormat:@"%hh"];
166     [self.alarmPickerDw selectRow:[dateFormatter stringFromDate: currentTime] integerValue] inComponent:0 animated:YES];
167     [dateFormatter setDateFormat:@"%mm"];
168     [self.alarmPickerDw selectRow:[dateFormatter stringFromDate: currentTime] integerValue] inComponent:1 animated:YES];
169 }
170 }
171
172 [self.alarmPickerDw selectRow:k inComponent:2 animated:YES];
173 if ([_deviceData[FREQ_ALARM_DOWN] isEqualToString:DISABLED]) {
174     [_disableAlarmDwState setOn:FALSE animated:TRUE];
175     [_alarmPickerDw setUserInteractionEnabled:FALSE];
176     [_alarmPickerDw setAlpha:0.6 ];
177 }
178
179
180
181 [_invalidLabel setHidden:TRUE];
182
183
184 }
185
186 - (void)viewWillDisappear:(BOOL)animated {
187     [super viewWillDisappear:animated];
188
189     // check if the back button was pressed
190     if (self.isMovingFromParentViewController && !dataToBeReturned && !_deviceData) {
191         NSDictionary* userInfo = @{@"M":deviceID};
192         NSNotificationCenter* nc = [NSNotificationCenter defaultCenter];
193         [self.navigationController popViewControllerAnimated:YES];
194         [nc postNotificationName:@"cancelPairing" object:self userInfo:userInfo];
195     }
196 }
197
198 - (void)didReceiveMemoryWarning {
199     [super didReceiveMemoryWarning];
200     // Dispose of any resources that can be recreated.
201 }
202
203 /*-----picker-----*/
204 // The number of columns of data
205 - (NSInteger)numberOfComponentsInPickerView:(UIPickerView *)pickerView
206 {
207     return 3;
208 }
209
210 // The number of rows of data
211 - (NSInteger)pickerView:(UIPickerView *)pickerView numberOfRowsInComponent:(NSInteger)component
212 {
213     if([_pickerData[component] isEqualToArray:_pickerData0]){
214         return 24;
215     }
216     else if([_pickerData[component] isEqualToArray:_pickerData1]){
217         return 60;
218     }
219     else{
220         return 3;
221     }
222 }
223
224 }
225

```

```

226 // The data to return for the row and component (column) that's being passed in
227 - (NSString*)pickerView:(UIPickerView *)pickerView titleForRow:(NSInteger)row forComponent:(NSInteger)component
228 {
229     return _pickerData[component][row];
230 }
231
232
233
234 - (void)pickerView:(UIPickerView *)pickerView didSelectRow:(NSInteger)row inComponent:(NSInteger)component
235 {
236     // This method is triggered whenever the user makes a change to the picker selection.
237     // The parameter named row and component represents what was selected.
238
239 }
240
241 /*
242 #pragma mark - Navigation
243
244 // In a storyboard-based application, you will often want to do a little preparation before navigation
245 - (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
246     // Get the new view controller using [segue destinationViewController].
247     // Pass the selected object to the new view controller.
248 }
249 */
250
251 /*-----buttons-----*/
252 - (IBAction)openButton:(UIButton *)sender {
253     [deviceData sendCommandOpenCloseDirection:OPEN id:_deviceData[ID]];
254     [self.navigationController popViewControllerAnimated:YES];
255 }
256
257 - (IBAction)closeButton:(UIButton *)sender {
258     [deviceData sendCommandOpenCloseDirection:CLOSE id:_deviceData[ID]];
259     [self.navigationController popViewControllerAnimated:YES];
260 }
261
262 /*-----text fields-----*/
263 - (IBAction)deviceNameTextField:(id)sender
264 {
265     _titleNavBar.title = _deviceNameTextFieldText.text;
266     if (![deviceData checkInputData:_deviceNameTextFieldText.text])
267         [_invalidNameLabel setHidden:FALSE];
268     else
269         [_invalidNameLabel setHidden:TRUE];
270 }
271
272 -(void)dismissKeyboard
273 {
274     [_deviceNameTextFieldText resignFirstResponder];
275     [_labelTextField resignFirstResponder];
276 }
277
278 - (IBAction)disableAlarmUp:(id)sender
279 {
280     if ([_disableAlarmUpState isOn]) {
281         [_alarmPickerUp setUserInteractionEnabled:TRUE];
282         [_alarmPickerUp setAlpha:1];
283     }
284     else{
285         [_alarmPickerUp setUserInteractionEnabled:FALSE];
286         [_alarmPickerUp setAlpha:0.6];
287     }
288 }
289
290 - (IBAction)disableAlarmDw:(id)sender
291 {
292     if ([_disableAlarmDwState isOn]) {
293         [_alarmPickerDw setUserInteractionEnabled:TRUE];
294         [_alarmPickerDw setAlpha:1];
295     }
296     else{
297         [_alarmPickerDw setUserInteractionEnabled:FALSE];
298         [_alarmPickerDw setAlpha:0.6];
299     }
300 }
301
302
303
304 }

```

```

305
306 - (IBAction)doneButton:(UIBarButtonItem *)sender {
307     NSMutableDictionary * dataToReturn = [self returnDeviceDataDict];
308     if(dataToReturn!=nil){
309         dataToBeReturned = TRUE;
310         [self.delegate returnData:dataToReturn];
311     }
312 }
313
314 -(NSMutableDictionary *) returnDeviceDataDict
315 {
316     NSMutableDictionary *data = [[NSMutableDictionary alloc] init];
317
318     BOOL valid = TRUE;
319     if ([_deviceNameTextFieldText.text isEqualToString:@""] || ![deviceData checkInputData:_deviceNameTextFieldText.text]) {
320         [_invalidNameLabel setHidden:FALSE];
321         valid = FALSE;
322     }
323     [data setObject:_deviceNameTextFieldText.text forKey:NAME];
324     [data setObject:_idLabel.text forKey:ID];
325     [data setObject:[NSNumber numberWithInt:[_alarmPickerUp selectedRowInComponent:0]] forKey:HOURL_ALARM_UP];
326     [data setObject:[NSNumber numberWithInt:[_alarmPickerUp selectedRowInComponent:1]] forKey:MINUTE_ALARM_UP];
327     if([_pickerData2[_alarmPickerUp selectedRowInComponent:2] isEqualToString:EVERYDAY_SHOW])
328         [data setObject:EVERYDAY forKey:FREQ_ALARM_UP];
329     else if([_pickerData2[_alarmPickerUp selectedRowInComponent:2] isEqualToString:WEEKDAYS_SHOW])
330         [data setObject:WEEKDAYS forKey:FREQ_ALARM_UP];
331     else
332         [data setObject:WEEKENDS forKey:FREQ_ALARM_UP];
333
334     if (![disableAlarmUpState isOn]) {
335         [data setObject:DISABLED forKey:FREQ_ALARM_UP];
336     }
337
338     [data setObject:[NSNumber numberWithInt:[_alarmPickerDw selectedRowInComponent:0]] forKey:HOURL_ALARM_DOWN];
339     [data setObject:[NSNumber numberWithInt:[_alarmPickerDw selectedRowInComponent:1]] forKey:MINUTE_ALARM_DOWN];
340
341     if([_pickerData2[_alarmPickerDw selectedRowInComponent:2] isEqualToString:EVERYDAY_SHOW])
342         [data setObject:EVERYDAY forKey:FREQ_ALARM_DOWN];
343     else if([_pickerData2[_alarmPickerDw selectedRowInComponent:2] isEqualToString:WEEKDAYS_SHOW])
344         [data setObject:WEEKDAYS forKey:FREQ_ALARM_DOWN];
345     else
346         [data setObject:WEEKENDS forKey:FREQ_ALARM_DOWN];
347
348     if (![disableAlarmDwState isOn]) {
349         [data setObject:DISABLED forKey:FREQ_ALARM_DOWN];
350     }
351
352     [data setObject:[NSNumber numberWithInt:-1] forKey:ROW];
353     [data setObject:[_idLabel.text componentsSeparatedByString:@": ds"][1] forKey:ID]; //aqui erro para data nil
354     if ([_labelTextField.text isEqualToString:@""] || ![deviceData checkInputData:_labelTextField.text]) {
355         [_invalidLabel setHidden:FALSE];
356         valid = FALSE;
357     }
358     [data setObject:_labelTextField.text forKey:LABEL];
359
360     if (valid) {
361         NSLog(@"%@", data);
362         return data;
363     }
364     return nil;
365 }
366
367 - (IBAction)labelTextFieldAction:(id)sender {
368     if (![deviceData checkInputData:_labelTextField.text])
369         [_invalidLabel setHidden:FALSE];
370     else
371         [_invalidLabel setHidden:TRUE];
372 }
373 @end

```

```

1 //
2 //  SettingsViewController.h
3 //  DomusSapiens
4 //
5 //  Created by Leonardo Pedroso on 15/01/18.
6 //  Copyright © 2018 Leonardo Pedroso. All rights reserved.
7 //
8
9 #import <UIKit/UIKit.h>
10 #import "MACROS_CONNECTION.h"
11 #import "deviceData.h"
12
13 @interface SettingsViewController : UIViewController
14 @property (weak, nonatomic) IBOutlet UINavigationController *titleNavBar;
15
16 @property (strong, nonatomic) NSString * port;
17 @property (strong, nonatomic) NSString * ip;
18
19 @property (weak, nonatomic) IBOutlet UITextField *ipTextField;
20
21 @property (weak, nonatomic) IBOutlet UITextField *portTextField;
22 @property (weak, nonatomic) IBOutlet UISwitch *portForwardingSwitchState;
23 - (IBAction)portForwardingSwitchAction:(id)sender;
24
25 @property (weak, nonatomic) IBOutlet UITextField *passwordTextField;
26
27 @property (weak, nonatomic) IBOutlet UILabel *portLabel;
28
29 - (IBAction)connectButton:(id)sender;
30
31
32 - (IBAction)testButton:(id)sender;
33
34
35 @property (weak, nonatomic) IBOutlet UILabel *statusLabel;
36
37 @end

```



```

1 //
2 //  SettingsViewController.m
3 //  DomusSapiens
4 //
5 //  Created by Leonardo Pedrosa on 15/01/18.
6 //  Copyright © 2018 Leonardo Pedrosa. All rights reserved.
7 //
8
9 #import "SettingsViewController.h"
10
11 @interface SettingsViewController ()
12
13
14 @end
15
16 @implementation SettingsViewController
17
18 - (void)viewDidLoad {
19     [super viewDidLoad];
20     // Do any additional setup after loading the view.
21     //make keyboard disappear
22     UITapGestureRecognizer *tap = [[UITapGestureRecognizer alloc] initWithTarget:self action:@selector(dismissKeyboard)];
23     [self.view addGestureRecognizer:tap];
24
25     NSUserDefaults *connection = [NSUserDefaults standardUserDefaults];
26
27     _ipTextField.text = [connection objectForKey:IP];
28
29     if ([[connection objectForKey:PORT] isEqualToString:@""]) {
30         _portTextField.text = @"";
31         [_portLabel setEnabled:FALSE];
32         [_portTextField setEnabled:FALSE];
33         [_portForwardingSwitchState setOn:FALSE];
34     }
35     else{
36         _portTextField.text = [connection objectForKey:PORT];
37         [_portLabel setEnabled:TRUE];
38         [_portTextField setEnabled:TRUE];
39         [_portForwardingSwitchState setOn:TRUE];
40     }
41
42     _passwordTextField.text = [connection objectForKey:PASSWORD];
43     [_statusLabel setHidden:TRUE];
44 }
45
46 - (void)didReceiveMemoryWarning {
47     [super didReceiveMemoryWarning];
48     // Dispose of any resources that can be recreated.
49 }
50
51 /*
52 #pragma mark - Navigation
53 // In a storyboard-based application, you will often want to do a little preparation before navigation
54 - (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
55     // Get the new view controller using [segue destinationViewController].
56     // Pass the selected object to the new view controller.
57 }
58 */

```

```

59 //test invalidity of fields
60 - (IBAction)connectButton:(id)sender {
61
62     [_statusLabel setTextColor:[UIColor orangeColor]];
63     _statusLabel.text = @"Connecting...";
64     [_statusLabel setHidden:FALSE];
65
66     if ([_ipTextField.text isEqualToString:@""]) {
67         [self.statusLabel setTextColor:[UIColor redColor]];
68         _statusLabel.text = @"Invalid IP address";
69     }else if ([_portForwardingSwitchState.isOn] && [_portTextField.text isEqualToString:@""]){
70         [self.statusLabel setTextColor:[UIColor redColor]];
71         _statusLabel.text = @"Invalid port";
72     }else if ([_passwordTextField.text isEqualToString:@""]){
73         [self.statusLabel setTextColor:[UIColor redColor]];
74         _statusLabel.text = @"Invalid password";
75     }
76     else
77     {
78         NSUserDefaults *connection = [NSUserDefaults standardUserDefaults];
79         [connection setObject:[NSString stringWithString:_ipTextField.text] forKey:IP];
80         [connection setObject:[NSString stringWithString:_portTextField.text] forKey:PORT];
81         [connection setObject:[NSString stringWithString:_passwordTextField.text] forKey:PASSWORD];
82         [connection synchronize];
83
84         [self performSelector:@selector(checkConnection) withObject:nil afterDelay:0.01];
85
86         [[NSNotificationCenter defaultCenter] postNotificationName:@"update" object:nil];
87
88     }
89 }
90
91 //change appearance accordingly to the connection status
92 -(void) checkConnection {
93     if([deviceData testConnectionMain]>0){
94         [_statusLabel setTextColor:[UIColor greenColor]];
95         _statusLabel.text = @"Successfully connected";
96     }else{
97         [_statusLabel setTextColor:[UIColor redColor]];
98         _statusLabel.text = @"Could not establish a connection";
99     }
100 }
101
102 //test connection to main device
103 - (IBAction)testButton:(id)sender {
104     [_statusLabel setTextColor:[UIColor orangeColor]];
105     _statusLabel.text = @"Connecting...";
106     [_statusLabel setHidden:FALSE];
107     [self performSelector:@selector(testConnection) withObject:nil afterDelay:0.01];
108 }
109
110 //test connection to main devices and check how many secondary devices are connected
111 -(void) testConnection{
112     int n = [deviceData testConnectionMainFull];
113     [_statusLabel setHidden:FALSE];
114
115     if (n>0 && n!= 1 && n!= 61) {
116         [_statusLabel setTextColor:[UIColor greenColor]];
117         _statusLabel.text = [NSString stringWithFormat:@"Succesfully reached %i devices", n];
118     }else if (n==1){
119         [_statusLabel setTextColor:[UIColor greenColor]];
120         _statusLabel.text = [NSString stringWithFormat:@"Succesfully reached %i device", n];
121     }else if (n == 61){
122         [_statusLabel setTextColor:[UIColor orangeColor]];
123         _statusLabel.text = [NSString stringWithFormat:@"No devices seem to be available"];
124     }else{
125         [_statusLabel setTextColor:[UIColor redColor]];
126         _statusLabel.text = [NSString stringWithFormat:@"Connection failed, unable to reach any device"];
127     }
128
129 }
130

```

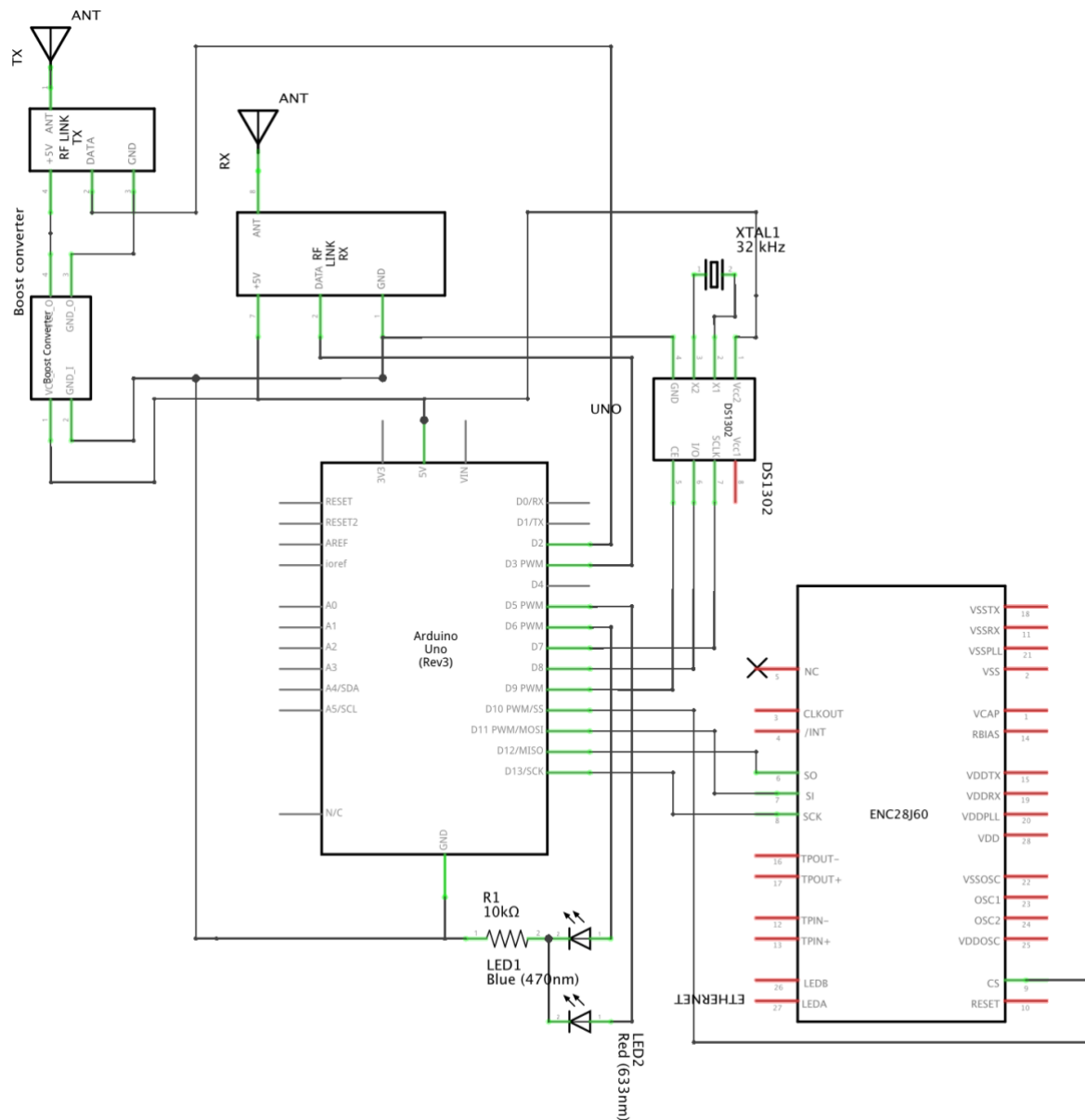
```

131 //switch between appearances
132 - (IBAction)portForwardingSwitchAction:(id)sender {
133     if ([_portForwardingSwitchState isOn]) {
134         [_portLabel setEnabled:TRUE];
135         [_portTextField setEnabled:TRUE];
136     }
137     else{
138         _portTextField.text = @"";
139         [_portLabel setEnabled:FALSE];
140         [_portTextField setEnabled:FALSE];
141     }
142 }
143
144 -(void)dismissKeyboard
145 {
146     [_ipTextField resignFirstResponder];
147     [_portTextField resignFirstResponder];
148     [_passwordTextField resignFirstResponder];
149     [_statusLabel setHidden:TRUE];
150 }
151
152 @end

```

# Esquemas Elétricos

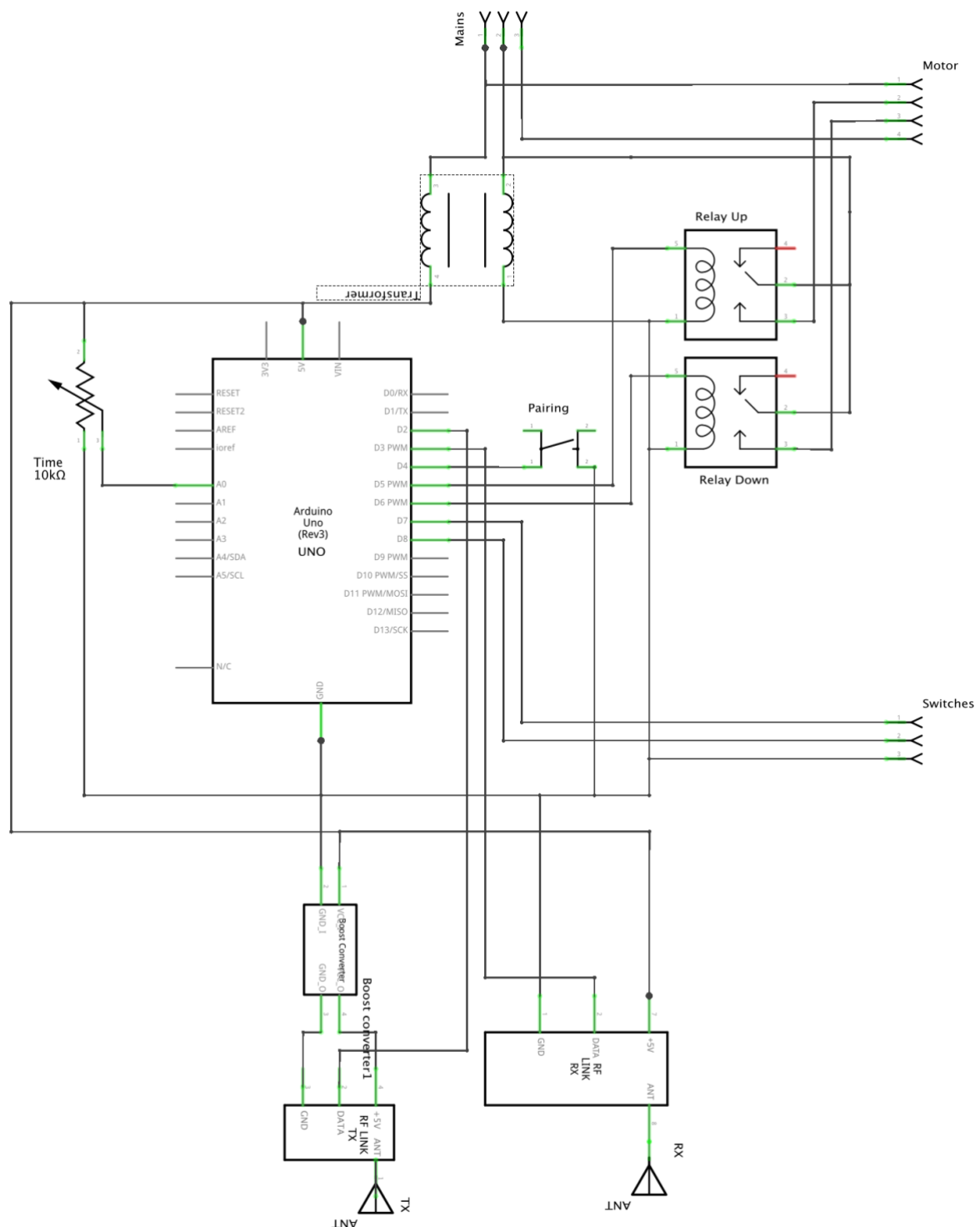
## Dispositivo Central



fritzing

Figura 4 – esquema elétrico do dispositivo central desenhado na ferramenta 'fritzing'

## Dispositivos Secundários



fritzing

Figura 5 – esquema elétrico dos dispositivos secundários desenhado na ferramenta 'fritzing'

# Avaliação Curricular

## Currículo de Matrícula

Aluno: Leonardo Pedroso Duarte (89691) - Bilhete de Identidade (14626458)

### Informação do apuramento

A Matrícula do Aluno ainda não foi submetida ao Apuramento Final.

## Aprovações

### Resumo

Aprovações	Total Créditos ECTS	Média	Ano Curricular
11	60.0	19.08	2

### Notas

Inscrições	Nota	Peso	Ano Lectivo	
Álgebra Linear	19	6.0	2017/2018	1 Sem.
Cálculo Diferencial e Integral I	18	6.0	2017/2018	1 Sem.
Desenho e Modelação Geométrica I	18	4.5	2017/2018	1 Sem.
Programação	19	6.0	2017/2018	1 Sem.
Química	19	6.0	2017/2018	1 Sem.
Seminário Aeroespacial I	18	1.5	2017/2018	1 Sem.
Cálculo Diferencial e Integral II	19	7.5	2017/2018	2 Sem.
Ciência de Materiais	19	6.0	2017/2018	2 Sem.
Gestão	20	4.5	2017/2018	2 Sem.
Mecânica e Ondas	20	6.0	2017/2018	2 Sem.
Sistemas Digitais	20	6.0	2017/2018	2 Sem.

Figura 6 – screenshot da plataforma Fénix da avaliação curricular

## Material utilizado e custo

1	2x <i>arduino Uno</i>	2x EUR 3.35
2	2x par módulos de <i>rf</i>	EUR 0.75 + EUR 0.66
3	2x <i>step up boost converter</i>	2x EUR 0.66
4	1x <i>DS1302 module</i>	EUR 0.66
5	1x <i>ethernet module</i>	EUR 2.14
6	2x <i>LED</i>	2x EUR 0.0131 *
7	1x resistência de 10k $\Omega$	EUR 0.0085 *
8	1x relé de dois canais	EUR 1.18
9	1x transformador (230V AC – 5V DC)	EUR 2.75
10	1x resistência variável	EUR 0.10 *
11	1x <i>push button</i>	EUR 0.0176 *
12	1x conectores elétricos	EUR 0.59 **
13	1x placa de madeira	EUR 1.79
14	20x porcas M3	(preço de 25 parafusos e 30 porcas) EUR 1.99
15	1x varão roscado M3	EUR 0.49 **

Tabela 1 – lista de materiais usados e respectivos preços

### Notas:

\* comprado em kit, valor aproximado comprado no Ebay

\*\* encontrei estes materiais em casa, fiz uma estimativa com base no website do Leroy Merlin

Total	EUR 21.17
-------	-----------

As faturas das compras recibos e link para os produtos estimados podem ser encontrados nos anexos, numerados de acordo coma a tabela acima.

# Anexos

## Compra de materiais



Anexo 1 – fatura material nº 9




Anexo 2 – fatura material nº13



Anexo 3 – fatura material nº14

**ORDER DATE**  
Dec 20, 2017

**1 item sold by** koala-ok



**ORDER TOTAL**  
EUR 3.35

**ITEM PRICE:**  
EUR 3.35

**ITEM DESCRIPTION:**  
UNO R3 ATmega328P Development Board With Boot Loader For Arduino UNO DS (282540765595)

[View similar items](#)

[View seller's other items](#)

[More actions](#)

Anexo 4 – comprovativo de compra de 1 Arduino Uno no Ebay. O segundo Arduino Uno que usei vinha no primeiro kit que comprei pelo que aproximarei o seu preço a EUR 3.35.



ORDER DATE  
Jun 24, 2018

1 item sold by [extrasp](#)



1 pair of RF 433Mhz Transmitter and Receiver Module Kit for Arduino Raspberry Pi  
(311858165335)

Estimated delivery Aug 06, 2018 - Oct 30, 2018

Tracking number: XYGPY8902353454YQ

This item has been shipped.

ORDER TOTAL  
EUR 0.75

Free shipping



ITEM PRICE:  
EUR 0.75

[Return this item](#)

[Leave feedback](#)

[More actions](#)

ORDER DATE  
Jun 24, 2018

1 item sold by [good-module](#)



433Mhz Wireless RF Transmitter Module+ Receiver Alarm Super Regeneration Arduino  
(262123832438)

Estimated delivery Jul 10, 2018 - Aug 13, 2018

Tracking number: ID18133683129500CN

This item has been shipped.

[Add note](#)

ORDER TOTAL  
EUR 0.66

Free shipping



ITEM PRICE:  
EUR 0.66

[Return this item](#)

[Leave feedback](#)

[More actions](#)

Anexo 5 – comprovativo de compra de 2 pares de módulos rf no Ebay.

ORDER DATE  
Dec 20, 2017

1 item sold by [elec-module58](#)



DC-DC Boost Converter 3.3v 5v 9v 12v 2A Adjustable Step Up Power Supply Module  
(112468350952)

Quantity: 2

[Add note](#)

ORDER TOTAL  
EUR 1.32

ITEM PRICE:  
EUR 1.32

[View similar items](#)

[View seller's other items](#)

[More actions](#)

Anexo 6 – comprovativo de compra de 2 step up boost converters no Ebay.

ORDER DATE  
Sep 25, 2016

1 item sold by [good-module](#)



New Arduino RTC DS1302 Real Time Clock Module For AVR ARM PIC SMD GM  
(262136732932)

[Add note](#)

ORDER TOTAL  
EUR 0.66

ITEM PRICE:  
EUR 0.66

[View similar items](#)

[View seller's other items](#)

[More actions](#)

Anexo 7 – comprovativo de compra de um DS1302 RTC module no Ebay.

ORDER DATE  
Oct 05, 2016

ORDER TOTAL  
▶ EUR 2.14

[View similar items](#)

[View seller's other items](#)

[More actions](#) ▼

1 item sold by [alice1101983](#)



New ENC28J60 Ethernet LAN Network Module For Arduino  
SPI AVR PIC LPC STM32  
( 310670027142 )

ITEM PRICE:  
▶ EUR 2.14

Anexo 8 – comprovativo de compra de um Ethernet module no Ebay.

ORDER DATE  
Jun 07, 2016

ORDER TOTAL  
▶ EUR 1.18

[View similar items](#)

[View seller's other items](#)

[More actions](#) ▼

1 item sold by [feidu\\_hu](#)



New 2 Channel DC 5V Indicator Light LED Relay Module  
Arduino ARM PIC AVR DSP  
( 231300098343 )

ITEM PRICE:  
▶ EUR 1.18

Anexo 9 – comprovativo de compra de um relé de dois canais no Ebay.

6	<a href="https://www.ebay.com/itm/100-Pcs-5mm-Red-Green-White-Blue-LED-Light-Emitting-Diodes-DC-2-5V-3V/172618155078?epid=581328509&amp;hash=item2830d80446%3Aq%3A57oAAOSwhQhY6gvN&amp;_sacat=0&amp;_nkw=led+5mm&amp;_from=R40&amp;rt=nc&amp;_trksid=m570.11313&amp;LH_TitleDesc=0%7C0">https://www.ebay.com/itm/100-Pcs-5mm-Red-Green-White-Blue-LED-Light-Emitting-Diodes-DC-2-5V-3V/172618155078?epid=581328509&amp;hash=item2830d80446%3Aq%3A57oAAOSwhQhY6gvN&amp;_sacat=0&amp;_nkw=led+5mm&amp;_from=R40&amp;rt=nc&amp;_trksid=m570.11313&amp;LH_TitleDesc=0%7C0</a>
7	<a href="https://www.ebay.com/itm/100PCS-Film-Resistors-Resistance-10K-Ohms-OHM-1-4W-5-Carbon-Film-Assortment-NEW/311740665507?hash=item4895310ea3%3Aq%3A5KIAAOSWB09YKXq0&amp;_sacat=0&amp;_nkw=resistencia+10k&amp;_from=R40&amp;rt=nc&amp;_trksid=m570.11313&amp;LH_TitleDesc=0%7C0">https://www.ebay.com/itm/100PCS-Film-Resistors-Resistance-10K-Ohms-OHM-1-4W-5-Carbon-Film-Assortment-NEW/311740665507?hash=item4895310ea3%3Aq%3A5KIAAOSWB09YKXq0&amp;_sacat=0&amp;_nkw=resistencia+10k&amp;_from=R40&amp;rt=nc&amp;_trksid=m570.11313&amp;LH_TitleDesc=0%7C0</a>
10	<a href="https://www.ebay.com/itm/10Pcs-Resistance-Variable-Haute-Precision-3296-W-Potentiometre-Tondeuse-103-10K/222970410765?hash=item33ea12770d%3Aq%3AoN0AAOSwbShbPdv4&amp;_sacat=0&amp;_nkw=variable+resistence+10k&amp;_from=R40&amp;rt=nc&amp;_trksid=m570.11313&amp;LH_TitleDesc=0%7C0">https://www.ebay.com/itm/10Pcs-Resistance-Variable-Haute-Precision-3296-W-Potentiometre-Tondeuse-103-10K/222970410765?hash=item33ea12770d%3Aq%3AoN0AAOSwbShbPdv4&amp;_sacat=0&amp;_nkw=variable+resistence+10k&amp;_from=R40&amp;rt=nc&amp;_trksid=m570.11313&amp;LH_TitleDesc=0%7C0</a>
11	<a href="https://www.ebay.com/itm/10-Values-180PCS-Tactile-Push-Button-Switch-Mini-Momentary-Tact-Assortment-16J1/182948831180?epid=28010007799&amp;hash=item2a98999fcc%3Aq%3A4dMAAOSwke9">https://www.ebay.com/itm/10-Values-180PCS-Tactile-Push-Button-Switch-Mini-Momentary-Tact-Assortment-16J1/182948831180?epid=28010007799&amp;hash=item2a98999fcc%3Aq%3A4dMAAOSwke9</a>
	<a href="aJ-Re&amp;_sacat=0&amp;_nkw=push+button&amp;_from=R40&amp;rt=nc&amp;LH_TitleDesc=0%7C0">aJ-Re&amp;_sacat=0&amp;_nkw=push+button&amp;_from=R40&amp;rt=nc&amp;LH_TitleDesc=0%7C0</a>
12	<a href="http://www.leroymerlin.pt/Site/Produtos/Eletricidade/Conectores-fixacoes-e-caixas-derivacao/17666453.aspx">http://www.leroymerlin.pt/Site/Produtos/Eletricidade/Conectores-fixacoes-e-caixas-derivacao/17666453.aspx</a>
14	<a href="http://www.leroymerlin.pt/Site/Produtos/Ferragens/Ferragens-de-fixacao/Porcas/15659371.aspx">http://www.leroymerlin.pt/Site/Produtos/Ferragens/Ferragens-de-fixacao/Porcas/15659371.aspx</a>
15	<a href="http://www.leroymerlin.pt/Site/Produtos/Ferragens/Ferragens-de-fixacao/Anilhas-e-varoes/14858431.aspx">http://www.leroymerlin.pt/Site/Produtos/Ferragens/Ferragens-de-fixacao/Anilhas-e-varoes/14858431.aspx</a>

Anexo 10 - tabela de links para estimativas de preços