"Monitorização e Controlo Remoto de Informação Ambiental através de Sistemas de Comunicações Celulares"

09/07/2019

Departamento de Ciências e Tecnologias

Orientador:

Professor Doutor Raul Dionísio

Alunos:

João Soares Nº20160313

Rúben Teixeira Nº20160240

Cláudia Santos Nº 20160291

Agradecimentos

Agradecemos ao Professor Doutor Raúl Dionísio de nos ter lecionado a cadeira de Laboratório de Projeto, motivando e orientando para a conceção de um projeto que visa a implementação tecnológica de um sistema de monitorização e controlo remoto de informação ambiental através da rede de comunicações celulares.

Agradecemos também o apoio que os familiares mais próximos e amigos de cada um dos elementos do grupo deu face ao tempo despendido na realização deste projecto.

Resumo (PT)

No âmbito da disciplina de Laboratório de Projeto da Licenciatura de Engenharia Informática e Informática de Gestão, foi decidido que se iria implementar um projecto tecnológico de monitorização e controlo remoto de informação ambiental através da rede de comunicações celulares, tendo como objectivo a aplicação de transdutores para recolha de informação, atuadores para ações de controlo, interface e uso de rede de comunicações celulares.

Este projeto enquadra-se relativamente à indústria 4.0, também conhecida por "*Internet of Things*", sendo que o conceito é bastante abrangente e não se resume apenas a interligação de sistemas eletrónicos. O tratamento e interpretação dos dados gerados por dispositivos IoT acrescenta valor aos tradicionais projectos M2M.

Abstract

Within the ambit of Project Laboratory of the Degree in Computer Engineering and Information Technology Management, it was decided that we would implement a technological project of monitoring and remote control of environmental information through cellular communications network, with the goal of applying transducers to collect information, actuators for control actions, interface and use of cellular communications network.

This project fits in industry 4.0, also known as "*Internet of Things*", and the concept is quite broad, and it's not just the interconnection of electronic systems. The processing and interpretation of data generated by IoT devices adds value to the traditional M2M projects.

Índice

Índice de Figuras

Índice de Tabelas

Lista de abreviaturas / siglas

CO	Monóxido de carbono
CO_2	Dióxido de carbono
GND	Ground
ICSP	In Circuit Serial Programming
IoT	Internet of Things
IOS	Iphone Operating System
LED	Light Emissor Diode
MCRIA	Monitorização e Controlo Remoto de Informação Ambiental
MHZ	Hertz
M2M	Machine to Machine
NH4	Amónio
NTC	Negative Temperature Coefficient
PPM	Partes por milhão
RISC	Reduced Instruction Set Computer
RX	Receptor
TX	Transmissor
Transdutor	Dispositivo que converte uma forma de energia noutra, para efeitos de medição e transferência de informação
USB	Universal Serial Bus
VCC	Voltage Common Collector

Introdução

Contextualização do Tema do Projeto

O presente projeto chama-se JRC, representa um grupo de trabalho composto por dois estudantes do curso de Informática de Gestão, o João e o Rúben, e uma estudante do curso de Engenharia Informática, a Cláudia. As iniciais do primeiro nome de cada um dos elementos representam a equipa que dá o nome ao projecto.

A realização do JRC compreende o desenvolvimento de um sistema que permite a monitorização e controlo de informação ambiental, recorrendo a tecnologias informáticas, utilizando recursos de hardware e software que serão detalhados nos capítulos correspondentes.

Objectivos e Motivação

O objectivo deste projeto passa pela aplicação de transdutores para a recolha de informação ambiental, atuadores para ações de controlo, interface e uso de rede de comunicações celulares.

O motivo que nos levou a abraçar este projeto foi a ambição de descoberta e exploração de conceitos e novas formas de captar, analisar, avaliar e controlar informação que é proveniente do meio ambiente que nos rodeia, face a indicadores de poluição de ar, temperatura, humidade e gases atmosféricos, pela utilização de tecnologias informáticas.

Organização do Relatório (Arduíno/Aplicação/Cloud)

O Arduíno surge de um projecto de investigação realizado no ano 2000 por *Massimo Banzi*, *David Cuartielles*, *Tom Ingoe*, *Gianluca Martino* e *David Mellis* no Instituto de Design e Interação *Ivrea* em Itália, baseado na linguagem de programação "*Processing*", desenvolvida para fins de artes visuais, desenvolvida por *Casey Reas* e *Ben Fry*, assim como na tese do projecto "*wiring board*" de *Hernando Barragan*.

Em 2005 surge a primeira placa microcontrolador concebida para ajudar estudantes na conceção dos seus projectos de prototipagem, que não tinham conhecimentos prévios de eletrónica e programação de microcontroladores. Desta forma o Arduíno cresceu exponencialmente até aos dias de hoje, aplicado em áreas como a educação e investigação com o propósito de ajudar as

pessoas e as empresas a desenvolver um mundo tecnológico inteligente, automatizando desta forma ações e tarefas que só eram possíveis de fazer de forma manual, tornando-as mais rápidas e eficientes.

O projeto JRC tem como base a utilização de uma placa Arduíno que é uma placa microcontrolador baseada no microchip ATmega328. Optou-se por esta solução porque é ideal para o controlo de componentes eletrónicos (tais como sensores, módulos, led's, lcd's,etc), na execução de tarefas automáticas repetitivas, com baixo consumo de energia e baixo custo de aquisição. O Arduíno tem como base uma plataforma de código aberto (open-source) que permite efetuar desenvolvimento à medida, o que nos permitiu customizar o projecto desenvolvido.

A nossa aplicação para telemóvel foi desenvolvida com recurso ao programa Android Studio, proprietário da Google, que usámos para fazer a comunicação remota com o Arduíno e a Cloud. Optámos pelo desenvolvimento de uma aplicação Android, visto todos os nossos telemóveis terem este sistema operativo.

A Cloud implementada no projeto recorre à plataforma de desenvolvimento móvel da Google, a Firebase. Esta ferramenta baseia-se numa plataforma de código aberto que suporta várias "frameworks" e ambientes, com a integração com várias linguagens de programação como o Java, esta permite armazenar e sincronizar bases de dados, autenticar utilizadores, criar relatórios e estatísticas, monitorizar, aumentar a qualidade da performance das aplicações pela análise de eventos, criar ambientes de testes, assim como fazer previsões com base em ações comportamentais, permitindo a integração com vários serviços nomeadamente com a "PlayStore", onde disponibilizamos a aplicação JRC desenvolvida.

Componentes

Hardware

Arduíno

O modelo usado no nosso projecto foi o modelo Uno R3, que é uma placa microcontroladora baseada no microchip ATmega328P, que é um microcontrolador de 8 bits com uma flash programável de 32kbytes de arquitectura RISC com características energéticas de baixo consumo. É através desta placa que controlamos os sensores DHT11, Q135 e os módulos Bluetooth HC-05 e o LCD I2C que iremos detalhar mais à frente.

Tem catorze entradas/saídas de pins digitais, seis entradas analógicas, um cristal quartzo de 16 mhz, uma ligação USB, uma ligação de energia, um ICSP header (circuito seriável programável) e um botão de reset.

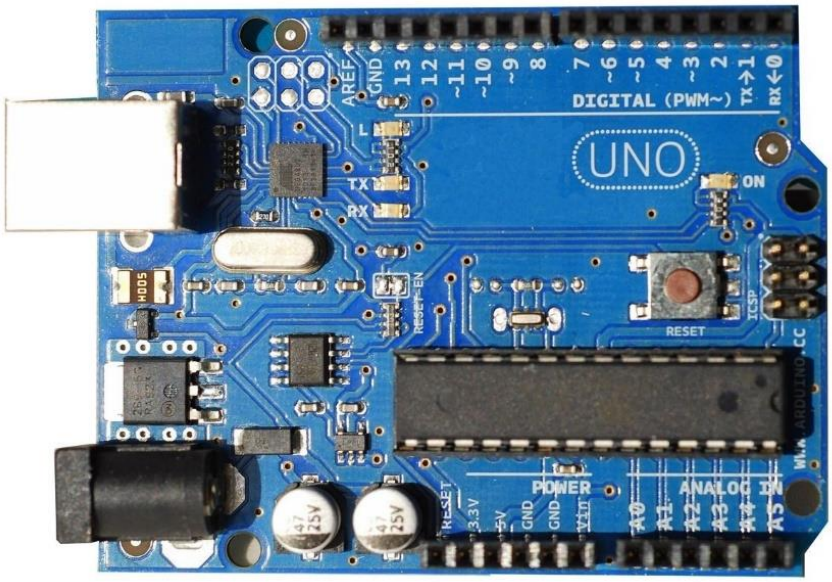


Figura 1 - Arduíno R3

Software

Arduíno IDE

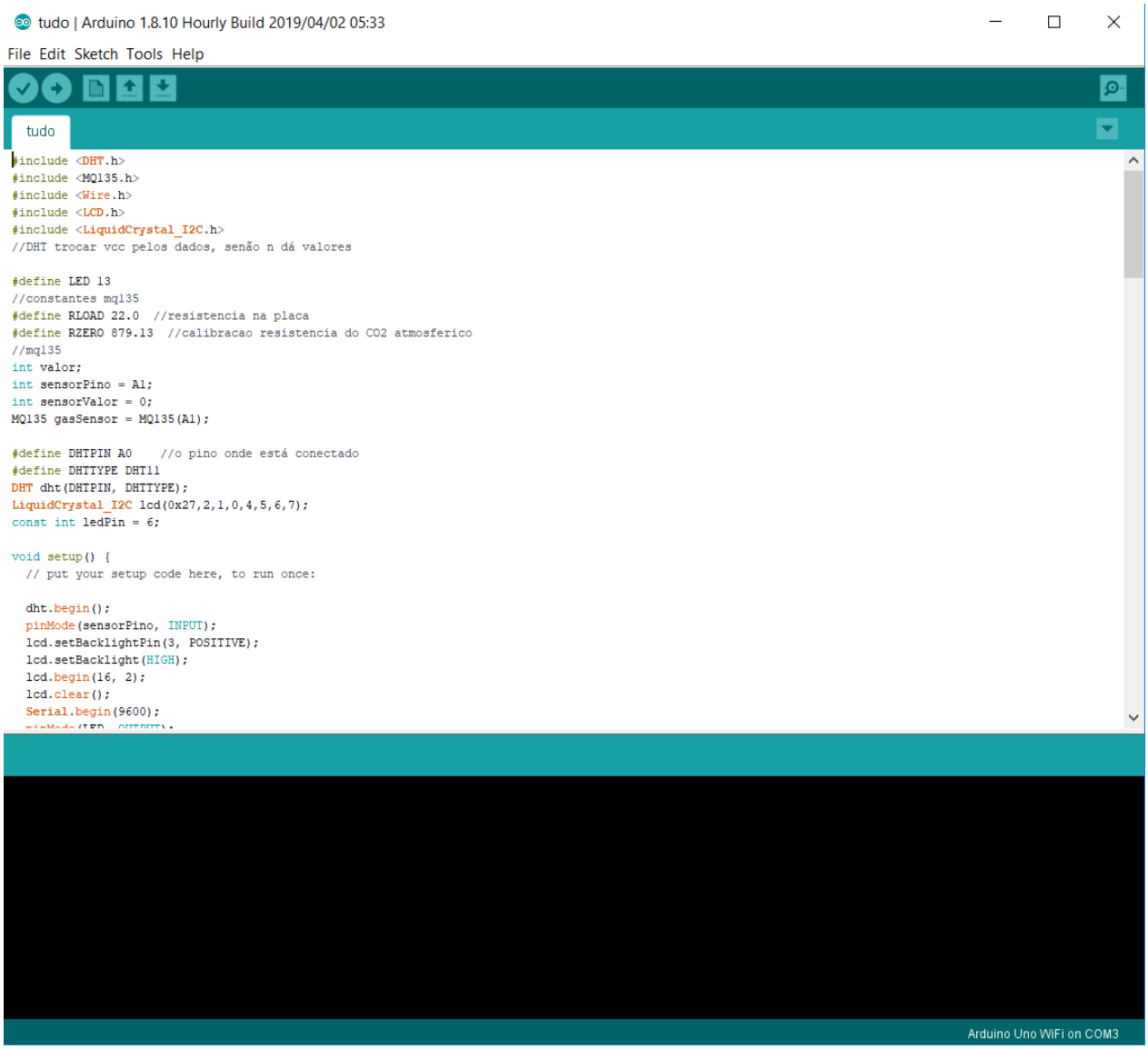


Figura 2 - Arduíno IDE

A programação da placa foi escrita na linguagem de programação Arduíno, baseada na linguagem C/C++.

As bibliotecas que usámos foram as bibliotecas adequadas para os componentes usados neste projecto, nomeadamente bibliotecas referentes ao sensor DHT11 e ao sensor MQ35, bem como as bibliotecas referentes ao LCD (Wire.h, LCD.h e LiquidCrystal_I2C.h).

Definimos os pinos onde vamos ler os dados de cada sensor, após isso definimos as variáveis de leitura, tais como temperatura e humidade, referente ao sensor DHT11, CO, CO2, Etanol,

Tolueno, NH4 e Acetona referentes ao sensor MQ135, bem como o pino onde estão conectados o LED e os sensores.

O LCD I2C vai mostrar o valor das variáveis de cada parâmetro do sensor, após isso definimos uma função para ler os sensores que será usada para enviar os dados para a aplicação Android.

Sensores

DHT11

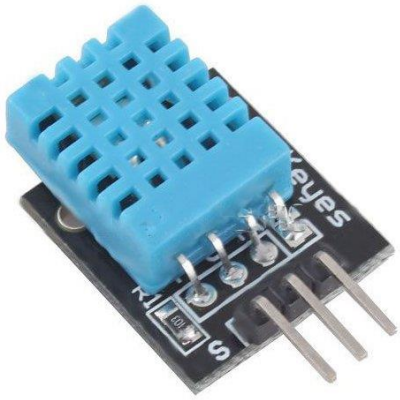


Figura 3 - Sensor DHT11

O DHT11 é um sensor de temperatura (0-50 º) e humidade (20-90%), que vai gerar uma saída digital calibrada. Apresenta baixo custo, e proporciona confiabilidade e estabilidade a longo prazo.

Tem 3 componentes principais: um componente de medição da humidade do tipo resistivo, um componente de medição da temperatura NTC (a resistência baixa com o aumento da temperatura), e um microcontrolador de 8 bits, que converte os sinais analógicos de ambos e envia um único sinal digital.

Ao medir a temperatura e comparar as mudanças ao longo do tempo, vai poder ser monitorizado o efeito de estufa. A humidade refere-se à percentagem de vapor de água no ar a uma determinada temperatura. Quando o ar a uma certa temperatura tem todo o vapor de água que pode conter a essa temperatura, a humidade relativa é de 100%.

A humidade protege-nos dos raios UV do sol, mas à medida que aumenta, o calor no planeta Terra também aumenta. O excesso de humidade pode levar ao crescimento de mofo e ácaros.

Configuração Pinos DHT11

Figura 4 - Pinos DHT11

Nº Pino	Nome	Descrição
1	VCC	Ligação do sensor, de +3.3V a +5V.
2	Data	Produz valores de temperatura e humidade através de data serial.
3	Ground	Conexão ao ground do circuito.

Tabela 1 - Pinos DHT11

MQ135

Figura 5 - Sensor MQ135

É usado em equipamentos para controlar a qualidade do ar. Este tipo de sensores é sensível a uma variedade de gases, detetando níveis de NH3, CO, CO_2, Álcool, entre outros.

A saída é um sinal analógico e pode ser lida com uma entrada analógica do Arduíno.

Tipos de Gases

CO

Também conhecido como monóxido de carbono, é um gás incolor sem cheiro ou sabor, inflamável e perigoso. As suas fontes emissoras naturais podem ser actividades vulcânicas, descargas elétricas e emissão de gás natural.

Aproximadamente 60% do monóxido de carbono provém de ações humanas, produto da combustão incompleta, como queimas com pouco oxigénio de combustíveis, sistemas de aquecimento, queima de biomassa e tabaco. As fontes emissoras que lançam maior concentração deste gás na atmosfera são as queimadas (incêndios) e o gás emitido do tubo de escape dos automóveis.

CO_2

O dióxido de carbono é o que mais contribui para o efeito de estufa, não tendo cheiro ou sabor, o que o torna difícil de detetar. A sua concentração em excesso leva à poluição do ar, chuva ácida, desequilíbrio do efeito de estufa (o que leva a que a temperatura aumente), que resulta numa grande degradação ambiental para os nossos ecossistemas.

Ao usar fontes de energias renováveis, substituindo os combustíveis mais poluentes por outros como energia solar e eólica, teremos uma diminuição das emissões do dióxido de carbono.

NH4

O amónio pode estar presente na água, maioritariamente proveniente de processos de materiais residuais, com origem vegetal e animal. O amónio em reação com a água pode dar origem ao amoníaco.

Etanol

Também denominado álcool etílico, obtido pela fermentação dos açúcares, produzido a partir da cana-de-açúcar (um exemplo de produtor), vai ter em média cerca de 7% de água, e um teor alcoólico mínimo de 92, 6º. Pode ser usado directamente como combustível nos automóveis.

A sua combustão completa produz água, dióxido de carbono e energia. A sua queima é uma fonte de poluição, apesar de ser menos poluente que a gasolina e diesel (produzindo cerca de 25% menos dióxido de carbono que a gasolina).

Tolueno

É inflamável, bastante perigoso para a saúde se for ingerido ou inalado. O tolueno é proveniente da gasolina e do petróleo, e também está presente em colas, tintas, fumo de cigarro e produtos de limpeza.

Acetona

A acetona é emitida na atmosfera tanto por fontes naturais como por ações humanas. As fontes naturais vêm de plantas e árvores, erupções vulcânicas e incêndios florestais. De ações humanas refere-se à fabricação de produtos químicos, fumo de tabaco, queima de madeira, combustão de resíduos, produção de petróleo e uso de solventes.

Configuração Pinos MQ135

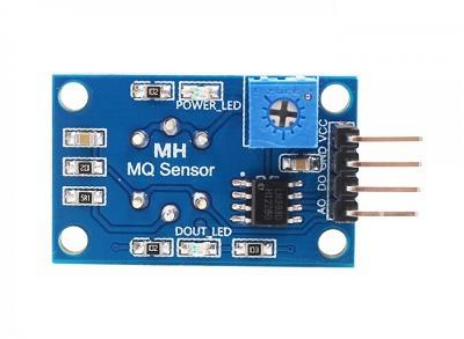


Figura 6 - Pinos MQ135

Nº Pino	Nome	Descrição
1	VCC	Usado para ligar o sensor, normalmente a voltagem é +5V.
2	Ground	Usado para conectar o módulo ao ground do sistema.
3	Digital Out	Obtém saídas de dados digitais do sensor.
4	Analog Out	Produz saídas analógicas de 0-5V com base na intensidade do gás.

Tabela 2 - Pinos MQ135

Outros componentes

Módulo Bluetooth HC-05

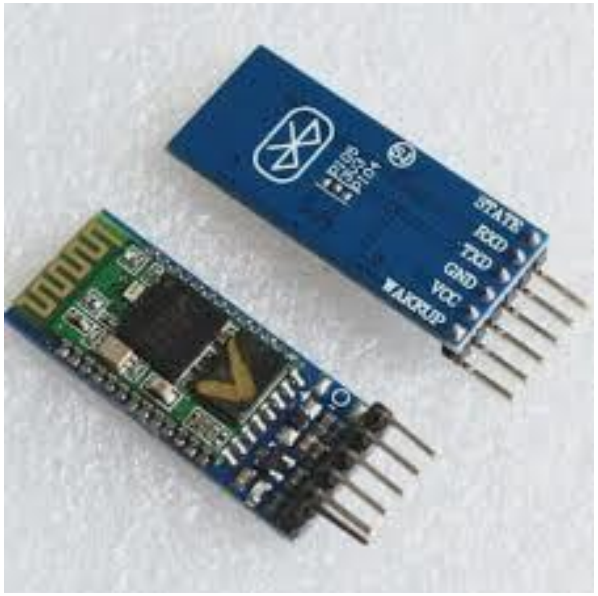


Figura 7 - Módulo Bluetooth HC-05

É composto por um módulo wireless HC-05 usado para a comunicação entre o Arduíno e o telemóvel.

Configuração Pinos HC-05

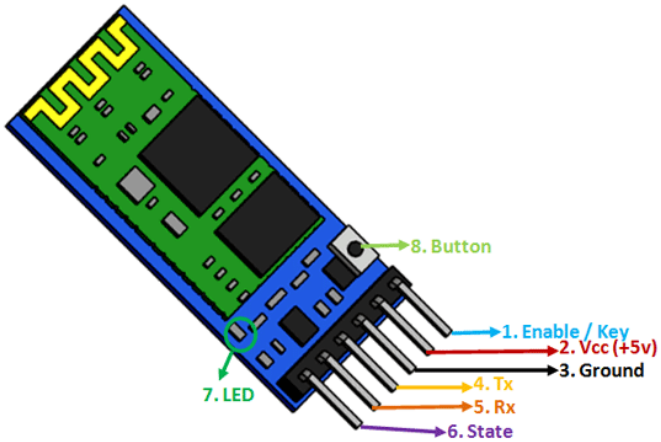


Figura 8 - Pinos Módulo Bluetooth HC-05

Nº Pino	Nome	Descrição
1	EN-M – Enable / Key	Este pino é usado para alternar entre modo de dados e modo de comando AT. Está no modo de dados (padrão).
2	VCC / +5V	Liga o módulo, conectando-se a uma tensão de alimentação (+5V).
3	GND	Usado para ligar ao ground da placa.
4	TX	Transmite dados seriais. Tudo o que é recebido por Bluetooth vai ser dado neste pino.
5	RX	Recebe dados seriais. Cada dado neste pino é transmitido via Bluetooth.
6	State	Está conectado ao LED no módulo, pode ser usado para verificar se o Bluetooth está a funcionar corretamente.
7	LED	Indica o estado do módulo. Se piscar uma vez em 2 segundos, o módulo está no modo de comando. Se piscar repetidamente, está a aguardar a conexão no modo de dados. Se piscar duas vezes num segundo, a conexão foi bem-sucedida no modo de dados.
8	Button	Usado para controlar o pino 1 para alternar entre os dados e o modo de comando.

Tabela 3 - Pinos Módulo Bluetooth HC-05

LCD 16x2 I2C

Figura 9 - LCD 16x2 I2C

É composto por um ecrã com fundo azul que serve de interface com o módulo integrado I2C.

Configuração Pinos LCD

Figura 10 - Pinos LCD 16x2 I2C

Nº Pino	Nome	Descrição
1	Ground	Ligação ao ground da placa
2	VCC	Usado para ligar o sensor, normalmente a voltagem é +5V.
3	SDA	Serial Data Line
4	SCL	Serial Clock Line

Tabela 4 - Pinos LCD I2C

Cabo USB

Figura 11 - Cabo USB

É o cabo usado como interface para ligar o Arduíno ao computador, usado como forma de transmissor de energia e dados.

Breadboard de 830 pontos

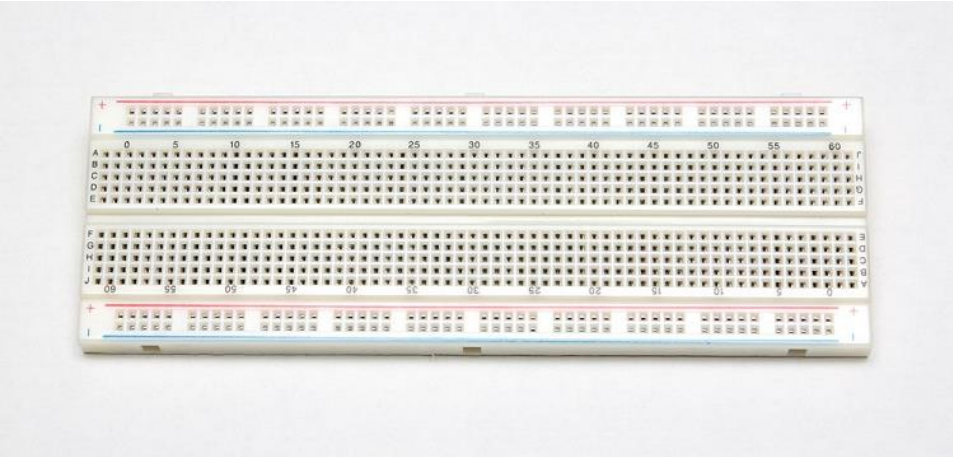


Figura 12 - Breadboard 830 pontos

É uma placa que contém um circuito integrado de 830 pontos, de uma malha metálica condutora isolada, onde são inseridos os componentes eletrónicos tais como: sensores, fios de ligação, módulos, led's, entre outros, usada na conceção de protótipos.

Pilha de 9 volts

Figura 13 - Pilha 9 volts

É utilizada uma pilha de 9 volts para fornecer energia ao Arduíno e garantir o seu funcionamento.

Fios de interligação

Figura 14 – Fios de interligação - Jumper Wires

São constituídos por cabo de metal condutor revestido por plástico isolante utilizados na ligação entre o Arduíno e a Breadboard.

LED

Figura 15 – LED

São componentes que convertem energia elétrica em energia luminosa. Têm polaridade, ou seja, a corrente flui apenas num sentido, logo a entrada de energia deve ser conectada no polo positivo (perna maior) e o GND no lado negativo.

É usado entre a transmissão de dados dos sensores para a aplicação, acende a luz antes da transmissão e desliga após transmitir os dados.

Caixa estanque

Figura 16 – Caixa Estanque

Para proteger os componentes utilizámos uma caixa estanque com proteção IP65, desta forma os componentes ficam protegidos totalmente contra a entrada de pó e resíduos, bem como de jatos de água de baixa pressão de qualquer direção.

Esquema Final

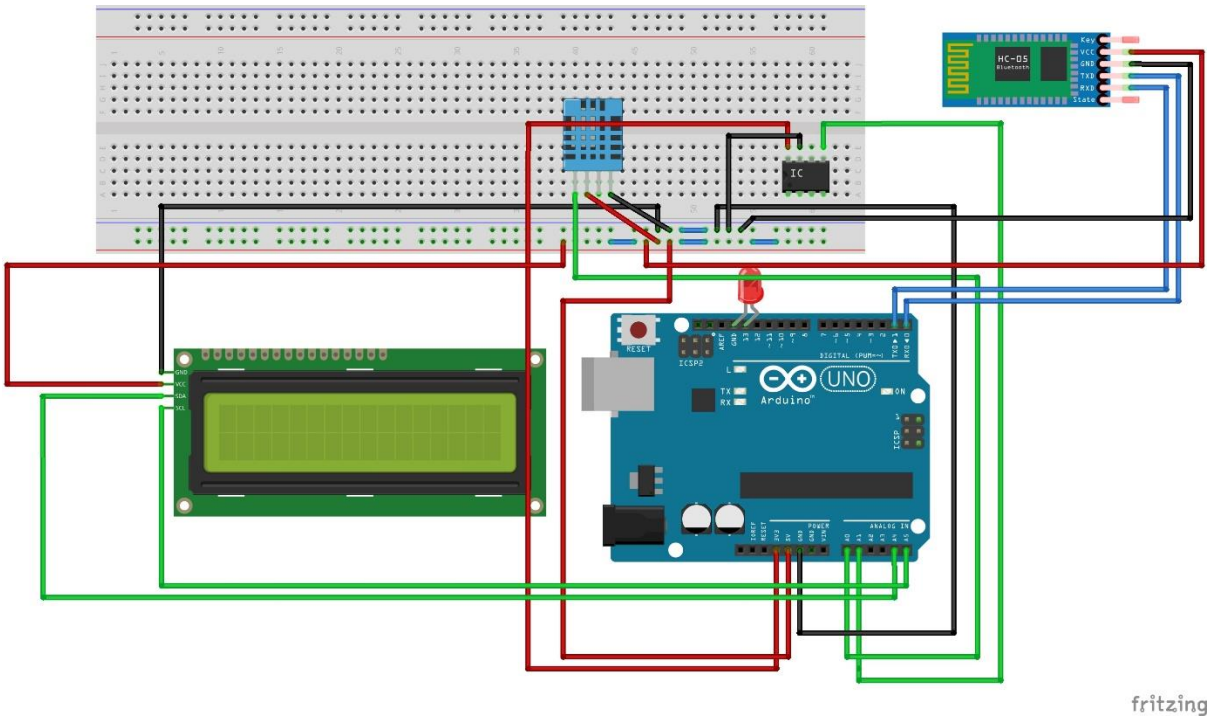


Figura 17 – Esquema de ligações

Para desenhar o circuito acima, utilizámos uma aplicação fornecida pela Fritzing, disponibilizada a versão 0.9.2.

Na figura 16 temos no esquema de ligações um módulo denominado IC que na verdade corresponde ao sensor MQ135, porque na aplicação utilizada para desenhar o circuito, não estava disponível o sensor pretendido.

Prova de Conceito - Esquema Final

Figura 18 – Caixa Esquema Final Fechada

Na figura 18, tal como indicam as setas, tem que se desaparafusar para poder ter acesso aos componentes da figura 19.

Nota: O led verde não está em funcionamento, foi colocado para permitir novas funcionalidades no futuro.

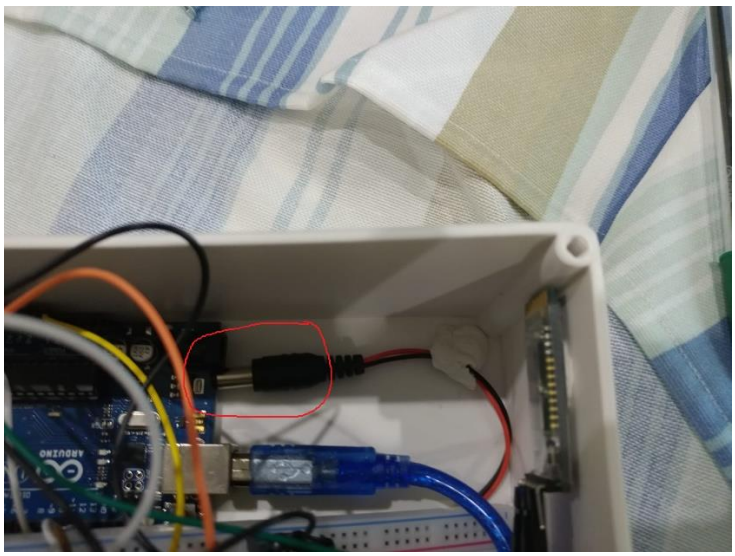


Figura 19 - Caixa Esquema Final Aberta

Nesta figura, assinalado a vermelho, o esquema é alimentado com uma pilha de 9 volts.

Custos

Componente	Estimativa de Custo (Euros)
Placa eletrónica Arduíno Uno + Cabo USB	18,70€
Breadboard 830 pontos	6€
Fios interligação - Jumper wires	4,10
Módulo Sensor DHT11	4,15€
Sensor MQ135	6,15€
LCD 16x2 I2C	7,20€
Módulo Bluetooth HC-05	8,30€
Pilha 9 volts	4,89€
Fio conector da pilha ao Arduíno	1,20€
LED	0,16
Caixa estanque	2,89
Total	63,74€

Tabela 5 - Custos

Aplicação

A aplicação tem como objetivo registar informação de dados tais como dados podem ser luminosidade, humidade, CO_2, de modo que o utilizador possa verificar as condições do meio ambiente onde se situa.

Para que seja possível receber estes dados, a aplicação utiliza os requisitos físicos do aparelho GSM, ou utiliza um periférico como uma board Arduíno para obter os dados que necessita. Após a leitura dos dados, a aplicação irá registar os dados na memória física do equipamento onde então podem ser enviados para uma *Firebase* ou partilhados para outro aparelho GSM que possua a aplicação.

Para a distribuição da aplicação vamos utilizar os serviços da Google, mais concreto a Google PlayStore que se encontra em qualquer aparelho que corra o sistema operativo Android.

De momento ainda não existem planos para a elaboração do projecto para Apple Store.

Para a elaboração da aplicação necessitamos de diversas ferramentas:

- ➔ 1 PC que consiga correr o IDE Android
- ➔ 1 Smartphone que utilize o sistema operativo Android e que suporte ligações Bluetooth
- ➔ SDK Platform Tools que se pode encontrar através deste link: https://developer.android.com/studio/releases/platform-tools

Para o IDE utilizamos um específico para o desenvolvimento de aplicações Android. Android Studio, baseado no IDE da Intellij IDEA da Jet Brains onde podemos desenvolver uma aplicação através das línguagens de programação JAVA ou KOTLIN. Em ambas as línguas, a personalização é efetuada em XML

No nosso caso, para esta aplicação ocorremos pela utilização da linguagem de programação JAVA.

O próprio IDE já fornece uma série bibliotecas que permite que seja efetuada a utilização dos recursos físicos do aparelho GSM como o Bluetooth e os sensores tais como:

- Android.hardware.Sensor;
- Android.hardware.SensorManager;
- Android.hardware.EventListener;
- Android.bluetooth.BluetoothAdapter;
- Android.bluetooth.BluetoothDevice;
- Android.bluetooth.BluetoothSocket;

Quanto aos requisitos do aparelho GSM, necessitamos dos seguintes recursos:

- Sistema operativo Android 4.1.1 Jelly Bean;
- Rede Wi-Fi ou móvel;
- Bluetooth;
- Sensores;

Embora muitas das aplicações hoje em dia sejam desenvolvidas para suportar a versão android 4.4 Kit Kat, optámos por baixar ainda mais por questões de custo de requisitos.

➔ Rede Wi-Fi ou móvel não é obrigatória para o principal funcionamento da aplicação. No entanto para que os registos sejam enviados para a base de dados, é necessário estabelecer uma ligação à internet.

➔ O Bluetooth será o meio de comunicação principal para que estabeleça uma ligação entre a board Arduíno visto que os aparelhos GSM não possuem todos os sensores necessários para aplicação.

Sensores GSM

Existe uma variedade significativa de sensores em um aparelho GSM. Segue-se uma lista dos sensores disponíveis:

Sensor	Tipo	Para que serve?
Acelerómetro	Hardware	Detetor de movimentos
Temperatura	Hardware	Monitorizar a temperatura
Gravidade	Software ou Hardware	Detetor de movimentos
Giroscópio	Hardware	Detetor de rotações
Luminosidade	Hardware	Detetor de luminosidade
Aceleração Linear	Software ou Hardware	Monitorizar aceleração em apenas um axis
Campo magnético	Hardware	Criar um compasso
Orientação	Software	Determinar posição do aparelho
Pressão	Hardware	Monitorizar a pressão de ar
Proximidade	Hardware	Determinar a posição do aparelho GSM durante uma chamada
Humidade	Hardware	Monitorizar a humidade do ambiente
Vetor rotativo	Hardware ou Software	Detetor de rotações e movimentos

Tabela 6 – Sensores GSM Disponíveis

Existem uns aparelhos que possuem mais sensores do que outros. Também não é possível deduzir que uma gama de aparelhos irá possuir os mesmos sensores ao longo da sua evolução. O melhor exemplo que encontrámos para este caso é o Samsung Galaxy da gama A entre o A5 2017 e o A7 2018. Ambos possuem um sensor de luminosidade para o controlo de iluminação do ecrã, mas apenas o Samsung A5 possui um sensor de pressão.

Dos sensores referidos acima, serão utilizados os sensores de temperatura, luminosidade, pressão e humidade.

Nota: O sensor de temperatura do aparelho GSM não nos pode fornecer uma leitura correta da temperatura ambiente pois a temperatura da bateria entra sempre em conflito com o a leitura da temperatura ambiente.

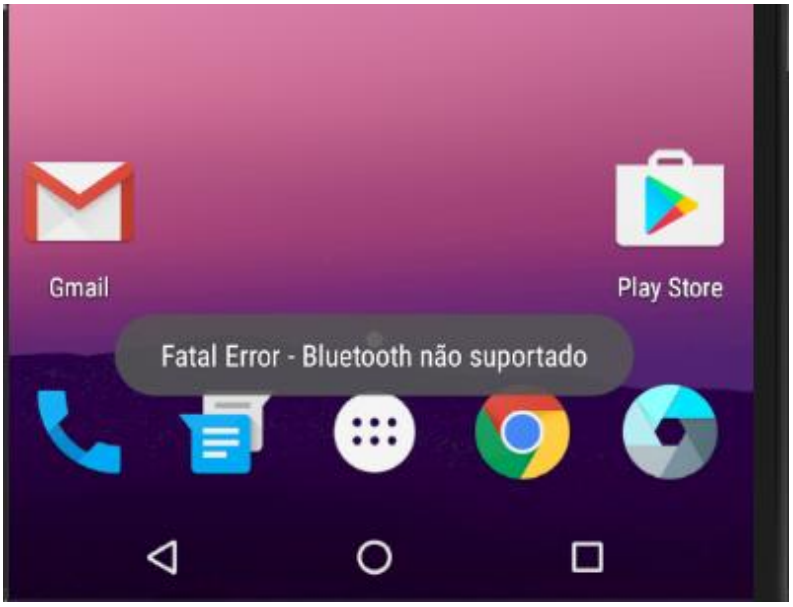
Dispositivos Disponíveis

Segue-se a lista de dispositivos que temos disponíveis:

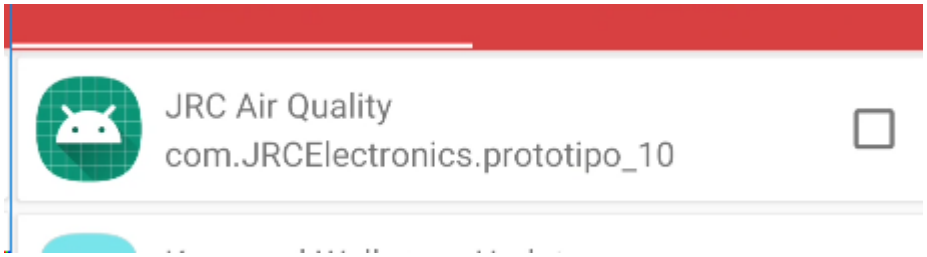
Marca	Modelo	Versão Android
Samsung	Galaxy S2	4.1.2
Samsung	Galaxy Note 1	4.1.2
LG	V20	8.0
Huawei	P20 Lite	9.0
Samsung	Galaxy A7	9.0
Samsung	Galaxy A5	8.0
Samsung	Galaxy J5	8.0
Lenovo	P780	4.4.2
Sony	Xperia E4	4.4

Tabela 7 – Lista de dispositivos disponíveis

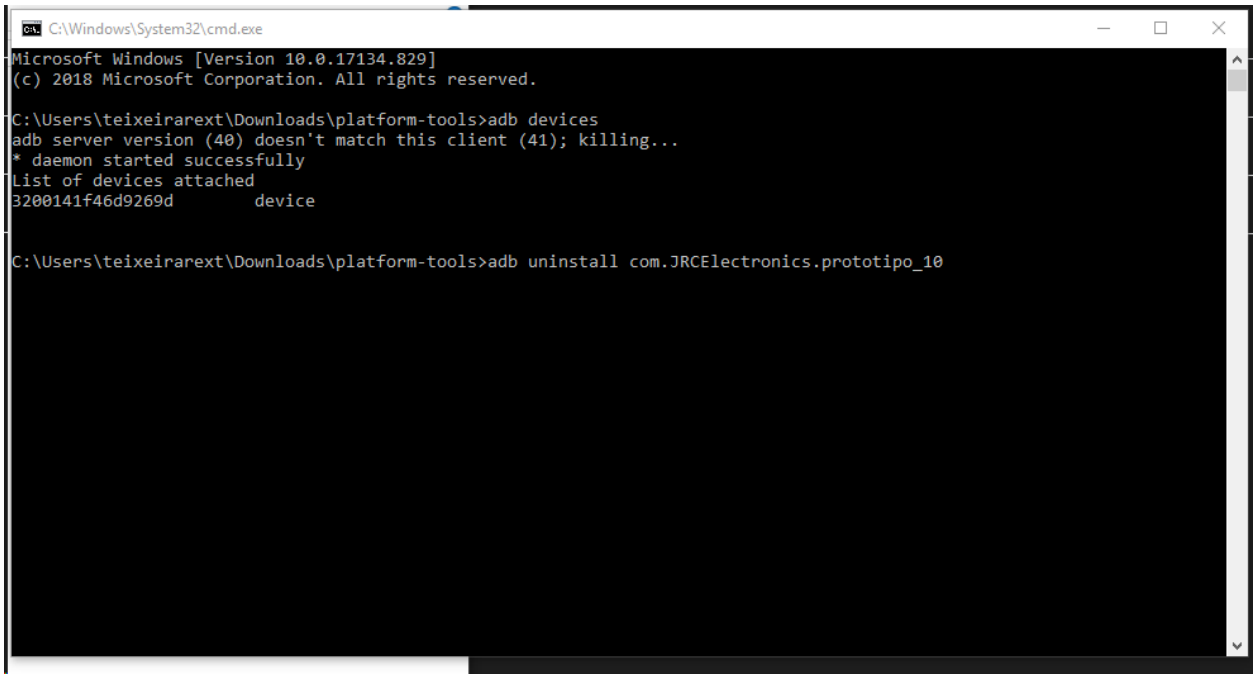
Para a elaboração desta aplicação a utilização de um emulador está fora de questão pois os emuladores não possuem Bluetooth.



SDK tools são ferramentas para o desenvolvimento e debugging. Utilizámos esta ferramenta para eliminar pacotes que ficam guardados no aparelho GSM de modo que não haja conflito entre as diversas versões da aplicação ao qual está a ser desenvolvida.



Em comparação, o processo é semelhante ao do Windows. Quando é efetuada a desinstalação de um software o mesmo pode deixar ficheiros mesmo já não tendo o software instalado. No entanto, não é possível aceder a estes ficheiros como no Windows, daí o uso das SDK Tools.



Fluxograma

Segue-se o fluxograma da aplicação:

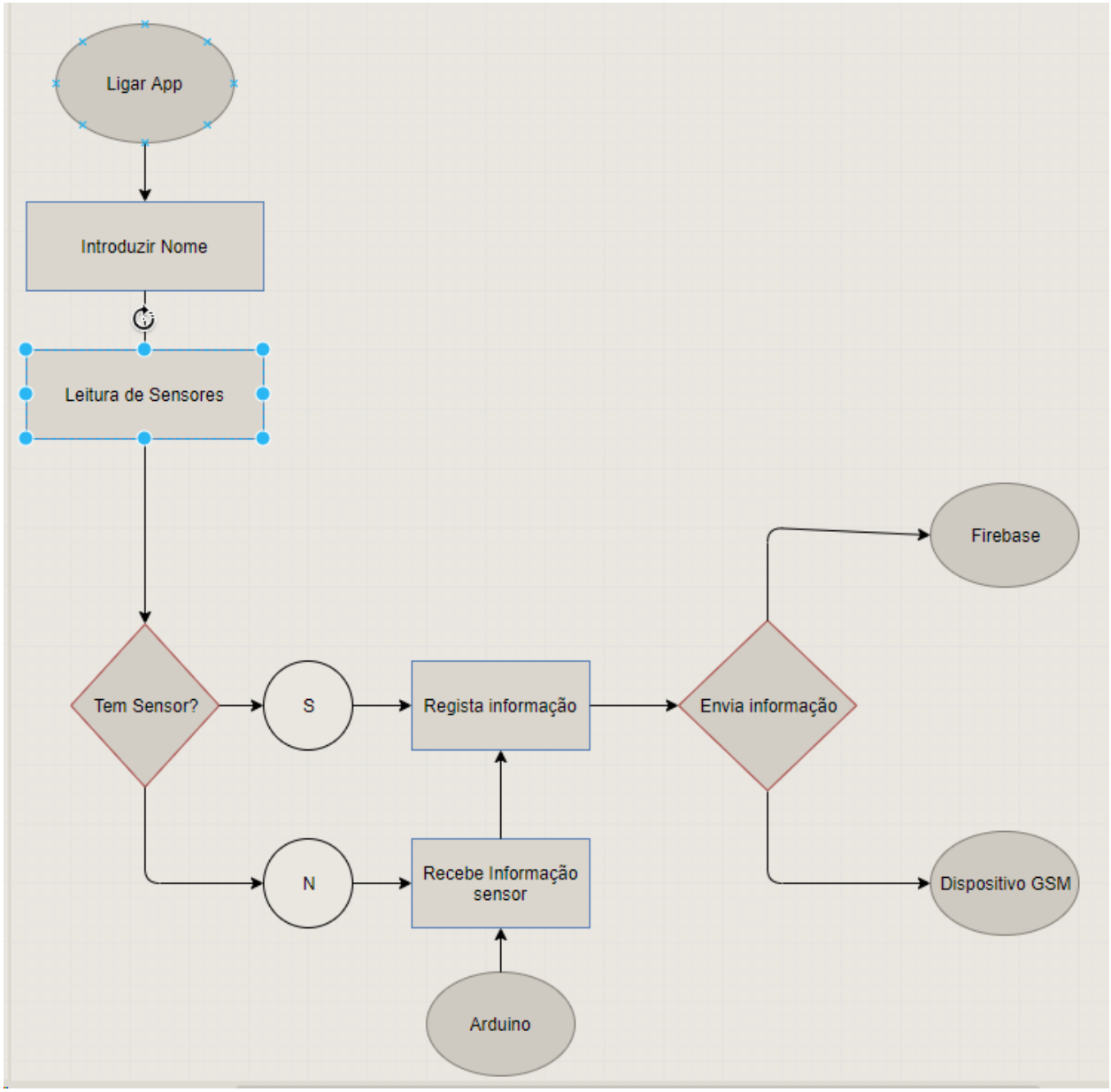


Figura 20 - Fluxograma da aplicação

Estrutura de dados

Segue-se a estrutura de dados da aplicação

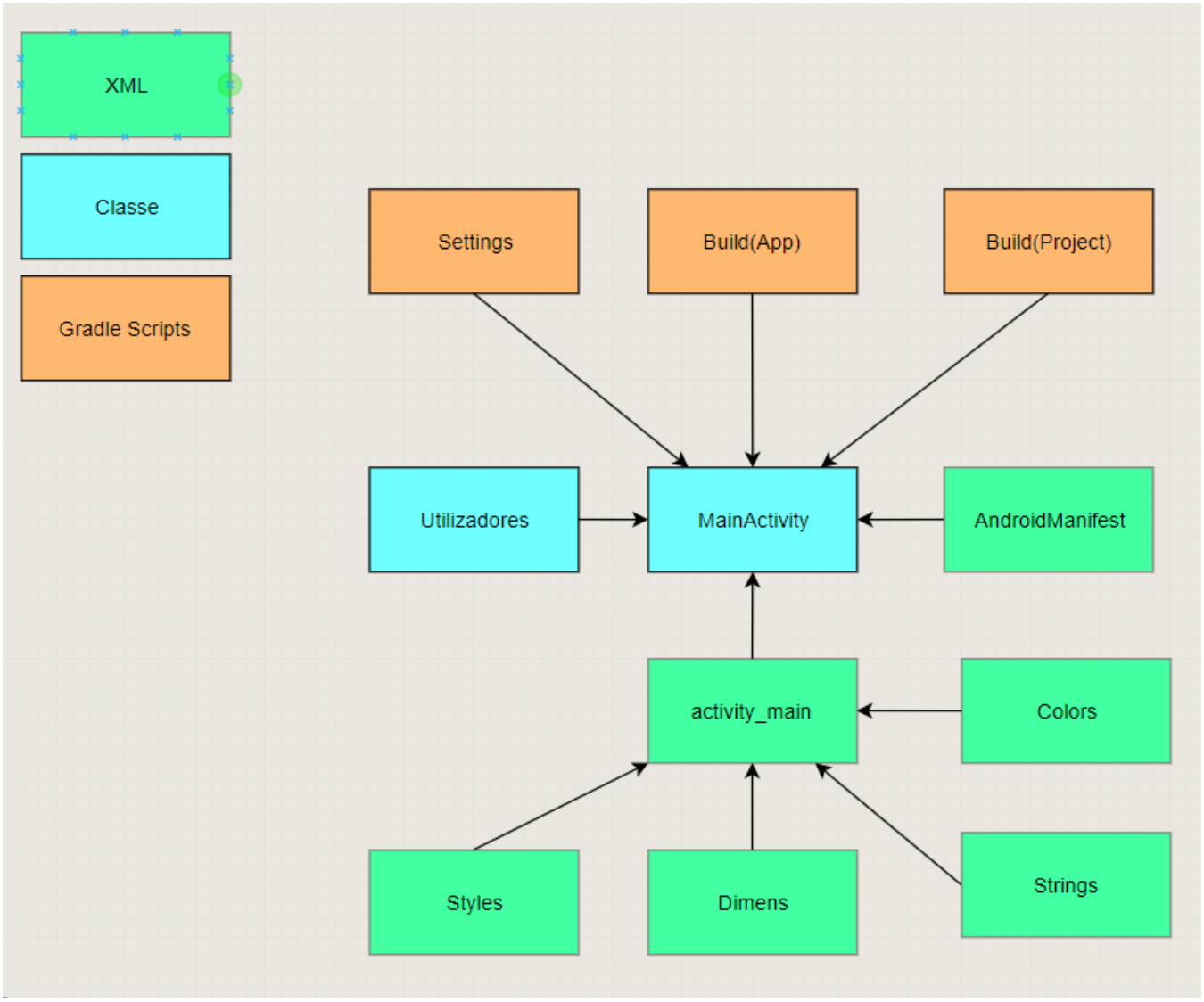


Figura 21 - Estrutura de dados da aplicação

Descrição da estrutura

No Android Studio quando um projecto é criado, são gerados os seguintes ficheiros:

- MainActivity.Class
- AndroidManifest.xml
- Activity_main.xml
- Colors.xml
- Dimens.xml
- Styles.xml
- Build.gradle(Project)
- Build.gradle(app)

Foi criada uma classe utilizadores para propósitos de registo e envio para a *Firebase*. Esta classe contém os seguintes atributos com os respectivos getters:

- UtilizadorID
- UtilizadorNome
- Arduino
- AndroidLuz
- AndroidPress
- AndroidTemp
- AndroidHumi

O AndroidManifest.xml é dos ficheiros já criados pelo Android Studio, no entanto é necessário colocar as permissões do Bluetooth.

As próximas duas linhas fazem que a aplicação consiga efectuar qualquer ligação Bluetooth

<uses-permission android:name=”android.permission.BLUETOOTH”/>

<uses-permission android:name=”android.permission.BLUETOOTH_ADMIN”/>

Esta linha, permite que o Bluetooth obtenha informação sobre a localização onde o utilizador se encontra.

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

Activity_main.xml é onde se efetua a personalização da aplicação.

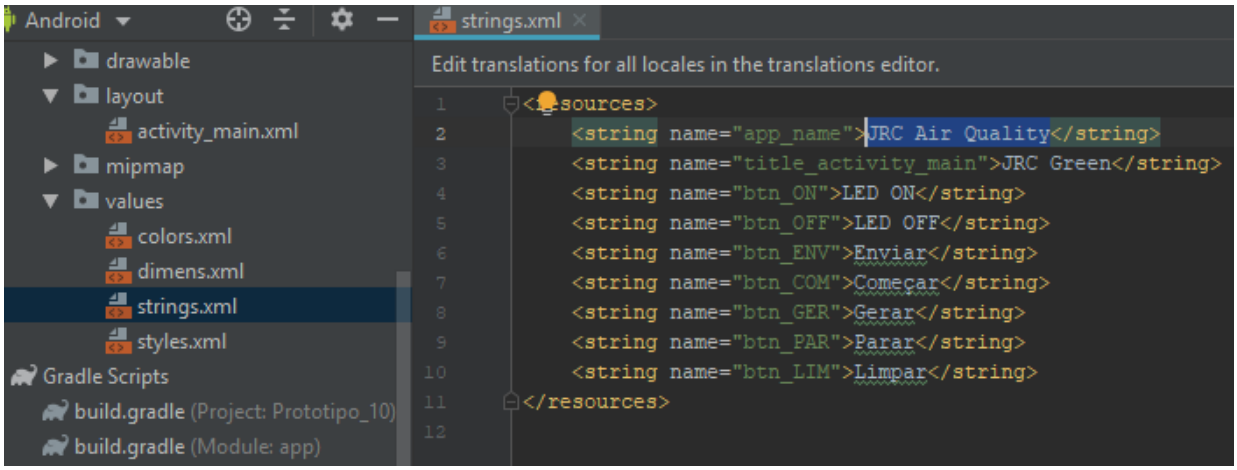
Segue-se um exemplo de uma estrutura para uma caixa de texto:

```
<EditText
    android:id="@+id/editText2"
    android:layout_width="391dp"
    android:layout_height="90dp"
    android:layout_above="@+id/editText"
    android:layout_alignParentStart="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentEnd="true"
    android:layout_alignParentRight="true"
    android:layout_marginStart="0dp"
    android:layout_marginLeft="0dp"
    android:layout_marginTop="10dp"
    android:layout_marginEnd="4dp"
    android:layout_marginRight="4dp"
    android:layout_marginBottom="-54dp"
    android:hint="Introduza o seu nome" />
```

Qualquer objecto é constituído pelas suas propriedades (id, alinhamento, nome, etc).

Os ficheiros pré criados pelo Android Studio como colors,dimens,style, e strings.xml vão ser chamados pelo ficheiro activity_main.xml.

Estes ficheiros vão conter variáveis em que uma vez são chamadas vão devolver uma determinada personalização.



Build.gradle e Build.gradle são scripts pré criados pelo IDE onde são colocadas todas as dependências e serviços para aplicação. Dois exemplos são os serviços da Google:

classpath 'com.google.gms:google-services:4.2.0'

E a implementação da *Firebase*:

implementation 'com.google.firebase:firebase-database:16.0.4'

MainActivity.Class é onde são implementadas as funções para o funcionamento da aplicação.

Estas funções vão interagir com o ficheiro xml para apresentação dos dados.

As funções que estão incluídas nesta classe são as seguintes:

- Criação de um socket(pacote) para efectuar a comunicação entre os dispositivos Bluetooth:

  "private BluetoothSocket createBluetoothSocket(BluetoothDevice device) throws IOException"

- Inicialização da comunicação Bluetooth:

  "public boolean BTinit()"

- Estabelecimento da ligação Bluetooth:

  "public boolean BTLigado ()"

- Verificação do estado Bluetooth:

  "private void VerificaLigacaoBT ()"

- Exceção para fechar aplicação em caso de erro:

  private void errorExit(String title, String message)

- Adicionar utilizador à firebase

  public void addUtilizador

- Função obrigatória para implementação dos sensores:

  Nota: A função é obrigatória de estar no código, no entanto não tem conteúdo

  public void onAccuracyChanged(Sensor sensor, int i)

- Apresentação de valores dos sensores do Android:

  public void onSensorChanged(SensorEvent sensorEvent)

- Botão para iniciar a ligação Bluetooth da aplicação através do botão "Começar":

  public void onClickComecar (View view)

- Botão para receber os dados do Arduíno:

  public void onClickGerar (View view)

- Botão para parar a leitura de dados e desligar a ligação Bluetooth:

  public void onClickParar (View view) throws IOException

- Limpa a vista onde apresenta os dados recebidos pelo Arduíno:

  public void onClickLimpar (View view)

- Método para inicializar os componentes:

  protected void onCreate(Bundle savedInstanceState)

- Método para resumir atividade caso alguma interrupção na actividade tal como mudar de aplicação:

  public void onResume()

- Método para colocar atividade em pausa caso aplicação seja minimizada:

  public void onPause()

- Prepara o Arduíno para a leitura dados à aplicação:

  void beginListenForData()

- Método que recebe os dados do Arduino e coloca no buffer:

  private class ConnectedThread extends Thread

Cloud

A IDC, consultora Norte-Americana de renome no que diz respeito a análise de mercado tecnológico e previsões/tendências para o Futuro diz* que até 2020, mais de 90% das empresas vão utilizar plataformas ou serviços *cloud*.

**Retirado de: https://www.idc.com/research/viewtoc.jsp?containerId=US42014717*

Tal como será de esperar, a maioria de aplicações móveis necessita de Servidores *Backend* para executar as mais diversas tarefas como, autenticação de utilizadores, sincronização de dados e interpretação dos mesmos. No entanto, a criação destes Servidores e respetiva gestão tornar-se-ia pela sua complexidade um outro projeto distinto.

Sendo um dos desafios deste Projeto a partilha das informações recolhidas dos diversos sensores, independentemente da localização geográfica dos mesmos, os dados deixam de ser persistentes aos dispositivos que os geram, ainda para mais com a quantidade de dados gerados tendo em conta o crescimento exponencial de dispositivos inteligentes. Torna-se então necessário pensar em soluções remotas que nos permitam centralizar e sincronizar os dados entre os vários dispositivos.

É aqui que entra uma plataforma chamada *Firebase*, plataforma esta que pertence à Google, que também é a gestora do Sistema Operativo Android que utilizámos na componente da Aplicação, o casamento perfeito. O *Firebase* é um *Backend as a Service (BaaS)* com inúmeras funcionalidades como Autenticação de Utilizadores, Armazenamento de dados, Análises e Relatórios de falhas, etc, permitindo assim libertar os programadores de montarem e gerirem a sua Infraestrutura, tendo todos os benefícios que a *Cloud* oferece como disponibilidade 24x7x365, poupança energética e redução de custos no investimento inicial.

O *Firebase* está disponível com 3 planos de subscrição:

Produtos	Plano Spark Limites generosos para amadores Gratuito	Plano Flame Fixed pricing for growing apps $25/month	Plano Blaze Calculate pricing for apps at scale Pagamento po utilização Uso gratuito do plano Spa incluso*
Teste A/B	Gratuito		
Analytics	Gratuito		
App Indexing	Gratuito		
Autenticação			
Phone Auth - US, Canada, and India	10 mil/mês	10 mil/mês	US$ 0,01/verificação
Phone Auth - All other countries	10 mil/mês	10 mil/mês	US$ 0,06/verificação
Other Authentication services	✓	✓	✓
Cloud Firestore			
Dados armazenados	1 GiB total	2.5 GiB total	$0.18/GiB
Largura de banda	10GiB/month	20GiB/month	Google Cloud pricing
Gravações de documento	20 mil/dia	100 mil/dia	US$ 0,18/100 mil

Figura 22 - Planos de Preços Firebase

Sendo que adotámos o plano *Spark*, pois para além de ser gratuito, suporta todas as métricas de operações de Entrada/Saída para o desenvolvimento do nosso projeto.

A nível de armazenamento de dados no *Firebase*, estão disponíveis dois tipos de Base de Dados, sendo ambas do tipo NoSQL:

- **Realtime Database** é o banco de dados original do *Firebase*. É uma solução eficiente e de baixa latência para aplicativos móveis que exigem estados sincronizados entre clientes em tempo real;
- **Cloud Firestore** é o novo banco de dados principal do *Firebase* para o desenvolvimento de aplicativos para dispositivos móveis. Ele oferece resultados ainda melhores que o Realtime Database com um novo modelo de dados mais intuitivo. O Cloud *Firestore* também tem consultas mais avançadas e rápidas e melhor escalabilidade que o Realtime Database;

Optámos pelo *Realtime Database* dado a sua eficiência e baixa latência, além de ser mais simples de gerir do que o *Cloud Firestore*.

A estrutura dos dados apresentados na forma de *Realtime Database*:

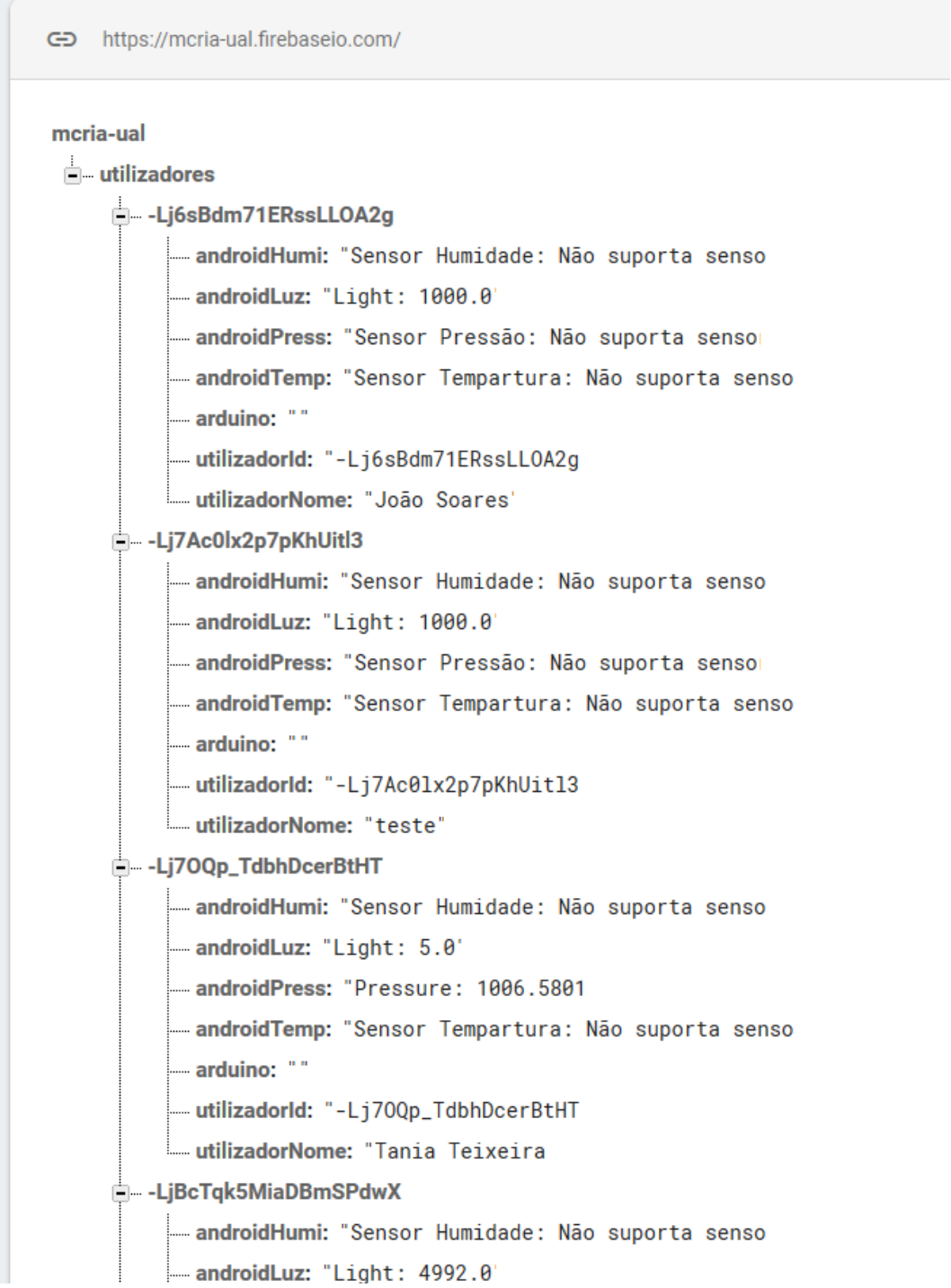


Figura 23 - Realtime Database com dados enviados pela aplicação

Sendo que o *Firebase* permite ainda que estes sejam consultados por terceiros através de um *Web Service*.

A comunicação entre a Aplicação e o Firebase é feito via HTTPS (Porta TCP 443) de forma a garantir segurança numa perspetiva Cliente-Servidor.

A nível de analítica podemos obter o consumo da Base de Dados do Firebase:

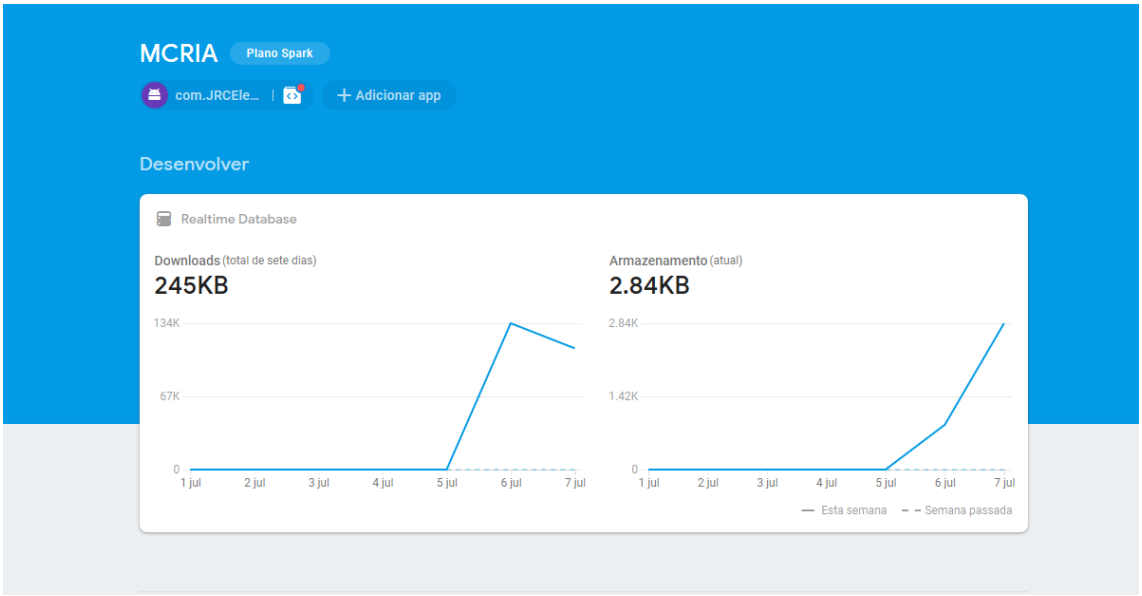


Figura 24 - Consumos de Base de Dados do Firebase

RoadMap

Entenda-se por RoadMap como uma visão estendida do futuro de um campo escolhido de investigação, no qual os autores deste projecto têm perspetivado:

- Lançamento para produção da aplicação MCRIA via Google Play Store;
- Desenvolvimento da aplicação MCRIA para Apple IOS;
- Acrescentar sensor de humidade do solo ao Arduíno;
- Construção de caixa estanque com dispositivo móvel embutido.

Os autores não se comprometem com prazos para implementação destas funcionalidades, mas é expectável que as mesmas demorem um semestre.

Conclusão

É com grande orgulho e satisfação termos tido o privilégio de realizar este projeto, no qual os seus autores deram provas da sua capacidade em ultrapassar os diversos desafios e interligar as várias componentes, cada uma com a sua especificidade.

Acreditamos que com este projeto possamos contribuir para um fim algo maior apresentando assim uma base tecnológica que possa ser melhorada permitindo assim acrescentar novos sensores, novos sistemas operativos, conseguindo alcançar um maior número de casos de uso.

Se for objectivo dos autores prosseguirem ou continuarem os estudos académicos, este projecto servirá de base para algo mais aprofundado, tal como projeto de mestrado, dando assim continuidade ao MCRIA.

Bibliografia

A bibliografia apresentada foi efetuada de acordo com a norma portuguesa NP 405-1 e NP 405-4, respeitando a especifidade de áreas de conhecimento, conforme a norma da APA(American Psychological Associtation) sendo que os links de repositórios de programas são exceção.

NAGAR, Jyoti; MARG, Sahkar – Privacy Policy. Circuitloop Technologies LLP. Jaipur,[Consult. 12 Abr. 2019]. Disponível em https://components101.com/privacy-policy

NAGAR, Jyoti; MARG, Sahkar - MQ-135 Gas Sensor for Air Quality. Circuitloop Technologies LLP. Jaipur, (27 Fev. 2018), [Consult. 12 Abr. 2019]. Disponível em https://components101.com/sensors/mq135-gas-sensor-for-air-quality

NAGAR, Jyoti; MARG, Sahkar – DHT11 Temperature and Humidity Sensor. Circuitloop Technologies LLP. Jaipur, (4 Jan. 2018), [Consult. 12 Abr. 2019]. Disponível em https://components101.com/dht11-temperature-sensor

NAGAR, Jyoti; MARG, Sahkar - HC-05 Bluetooth Module. Circuitloop Technologies LLP. Jaipur, (10 Mar. 2018), [Consult. 13 Abr. 2019]. Disponível em https://components101.com/wireless/hc-05-bluetooth-module

CAMPBELL, Scott; – How to set up the DHT11 humidity sensor on Arduino. Circuit Basics, [Consult. 4 Mai. 2019]. Disponível em http://www.circuitbasics.com/how-to-set-up-the-dht11-humidity-sensor-on-an-arduino

Mybotic - DHT11 Humidity Sensor Module Interface With Arduino. Instructables. [Consult. 11 Mai. 2019]. Disponível em https://www.instructables.com/id/DHT11-Humidity-Sensor-Module-Interface-With-Arduin

Arduino Project Hub - Air Meter Making #2: Use the Arduino DHT 11 Module. Arduino. [Consult. 11 Mai. 2019]. Disponível em https://create.arduino.cc/projecthub/62390/air-meter-making-2-use-the-arduino-dht-11-module-a4b583

Arduino Project Hub - DHT11 Temperature Humidity Sensor . Arduino. [Consult. 11 Mai. 2019]. Disponível em https://create.arduino.cc/projecthub/techno_z/dht11-temperature-humidity-sensor-98b03b

Microcontrollers Lab - DHT11 interfacing with arduino and weather station. Microcontrollers Lab. (Out. 2016). [Consult. 13 Mai. 2019]. Disponível em https://microcontrollerslab.com/dht11-interfacing-arduino-code/

NEDELKOVSKI, Dejan - DHT11 & DHT22 Sensors Temperature and Humidity Tutorial using Arduino. HowToMechatronics, Macedonia, (Mar. 2016). [Consult. 14 Mai. 2019]. Disponível em https://howtomechatronics.com/tutorials/arduino/dht11-dht22-sensors-temperature-and-humidity-tutorial-using-arduino

Last Minute Engineers - How DHT11 DHT22 Sensors Work & Interface With Arduino. Last Minute Engineers. [Consult. 15 Mai. 2019]. Disponível em https://lastminuteengineers.com/dht11-dht22-arduino-tutorial

PRASAD, Ravi - DHT11 Humidity Sensor with ESP8266 and ThingSpeak. Helectronics Hub. (Abr. 2018). [Consult. 18 Mai. 2019]. Disponível em https://www.electronicshub.org/dht11-humidity-sensor-with-esp8266

Electrofun - Sensor De Gás MQ-135. [Consult. 18 Mai. 2019]. Disponível em https://www.electrofun.pt/sensores-arduino/sensor-de-gas-mq135

Electrofun - Módulo Sensor Temperatura E Humidade DHT11. [Consult. 18 Mai. 2019]. Disponível em https://www.electrofun.pt/sensores-arduino/sensor-temperatura-humidade-dht11

KHONDAKER, Mahamudul - Arduino And MQ 135 Gas Sensor With Arduino Code. Arduino Project Hub. (Mar. 2016). [Consult. 13 Mai. 2019]. Disponível em https://create.arduino.cc/projecthub/karimmufte/arduino-and-mq-135-gas-sensor-with-arduino-code-a8c1c6

MAJUMDAR, Swagatam - MQ-135 Air Quality Sensor Circuit – Working and Interfacing with Program Code. Homemade Circuit Projects. (Mai. 2019) [Consult. 21 Mai. 2019]. Disponível em https://www.homemade-circuits.com/mq-135-air-quality-sensor-with-arduino

NEWTON, Alex - Arduino Smoke Level Detector using MQ-135 Sensor with Alert Alarm. How to Electronics. (Jun. 2018). [Consult. 13 Mai. 2019]. Disponível em https://www.how2electronics.com/arduino-smoke-level-detector-using-mq-135-sensor

Deyson – How To: Create an Android App With Android Studio to Control Led. Instructables. [Consult. 11 Jun. 2019]. Disponível em https://www.instructables.com/id/Android-Bluetooth-Control-LED-Part-2/

ZakariyeA – Air Quality Monitoring Device Using Arduino. Instructables. [Consult. 11 Jun. 2019]. Disponível em https://www.instructables.com/id/Air-Quality-Monitoring-Device-Using-Arduino/

ThothLoki – Portable Arduino Temp/Humidity Sensor With LCD. Hackster.io (Feb. 2016). [Consult. 8 Jun. 2019]. Disponível em https://www.hackster.io/ThothLoki/portable-arduino-temp-humidity-sensor-with-lcd-a750f4

WIJAYA, Ricky – Temperature, Humidity and Air Quality Control. Hacster.io (Jan. 2018) [Consult. 6 Jun. 2019] Disponível em https://www.hackster.io/ricky-wijaya/temperature-humidity-and-air-quality-control-3fc819

PRASAATH K, Hari – Arduino Based Air Quality Monitoring IOT Project. EngineersGarage [Consult. 18 Jun. 2019] Disponível em https://www.engineersgarage.com/contribution/arduino-based-air-quality-monitoring-iot-project

GUL, Sezgin – Bluetooth Wireless LCD Data Transfer. Instructables [Consult. 8 Jun. 2019] Disponível em https://www.instructables.com/id/Bluetooth-wireless-LCD-Data-Transfer/

Appdroidrm – Control Leds With Arduino and Bluetooth. Instructables. [Consult. 9 Jun. 2019] Disponível em https://www.instructables.com/id/Control-Leds-With-Arduino-and-Bluetooth/

Hive RD – Arduino and DHT11 Output to LCD module. Hive-RD [Consult. 8 Jun. 2019] Disponível em https://www.hive-rd.com/blog/arduino-dht11-output-lcd-module/

SURYATEJA, Sai – Bluetooth Control LEDs. Arduino Project Hub. (Mai. 2018). [Consult. 13 Mai. 2019]. Disponível em https://create.arduino.cc/projecthub/SURYATEJA/bluetooth-control-leds-27edbd

AQIB, Muhammad – IOT Based Air Pollution Monitoring System using Arduino. Circuit Digest. (Dec. 2016) [Consult. 3 Jun. 2019] Disponível em https://circuitdigest.com/microcontroller-projects/iot-air-pollution-monitoring-using-arduino

WIRSING, Brian. Sending and Receiving Data via Bluetooth with an Android Device. (Mar. 2014). [Consult. 16 Jun. 2019] Disponível em https://www.egr.msu.edu/classes/ece480/capstone/spring14/group01/docs/appnote/Wirsing-SendingAndReceivingDataViaBluetoothWithAnAndroidDevice.pdf

AJ – Air Quality Monitoring. IONOTS. (Feb. 2019) [Consult. 6 Mai. 2019] Disponível em https://www.ionots.com/air-quality-monitoring/

JavaWorld – Android Studio For Beginners. [Consult. 20 Abr. 2019] Disponível em https://www.javaworld.com/blog/android-studio-for-beginners/

RICHARDSON, Bryan – Mobile Development for Android Part 2. Stable Kernel. (Mai. 2016) [Consult. 26 Abr 2019]. Disponível em https://stablekernel.com/mobile-development-for-arduino-part-2

General Filters Incorporated – How Relative Humidity Affects Indoor Air Quality.GeneralFilters. [Consult. 1 Jul 2019]. Disponível

em https://www.generalfilters.com/wholesaler-center/How-Relative-Humidity-Affects-Indoor-Air-Quality_AE146.html

Microcontrollers Lab - Interfacing of MQ135 Gas Sensor with Arduino. (Abr. 2017). [Consult. 18 Mai. 2019]. Disponível em https://microcontrollerslab.com/interfacing-mq-135-gas-sensor-arduino

KHONDAKER, Mahamudul - Arduino And MQ 135 Gas Sensor With Arduino Code. Hackster.io, an Avnet Community. (Mar. 2016). [Consult. 13 Mai. 2019]. Disponível em https://www.hackster.io/karimmufte/arduino-and-mq-135-gas-sensor-with-arduino-code-a8c1c6

QUEIRÓS, Ricardo - Android Profissional Desenvolvimento moderno de aplicações. Lisboa: FCA - Editora de Informática, 2018. 208p. ISBN 9789727228744

MONK, Simon – Programming Arduino, Getting Started with Sketches. United States: McGraw-Hill, 2012. p. 177. ISBN 978-0-07-178422-1

Robo India Learning Portal - Arduino – Controlling LCD using BT Module HC-05. Robo India.[Consult. 19 Mai. 2019]. Disponível em https://roboindia.com/tutorials/i2c-lcd-hc-05-arduino/

Team East West University - Indoor Air Quality Monitoring System. Hacster.io (Set. 2018). [Consult. 6 Jun. 2019]. Disponível em https://www.hackster.io/east-west-university/indoor-air-quality-monitoring-system-5b5244

SOUZA, Líria - Monóxido de carbono. Mundo Educação.[Consult. 19 Mai. 2019]. Disponível em https://mundoeducacao.bol.uol.com.br/quimica/monoxido-carbono.htm

Comissão especializada da Qualidade da Água – FT-QI-13-Amónio. APDA – Associação Portuguesa de Distribuição e Drenagem de Águas. (Mar. 2016). [Consult. 13 Mai. 2019]. Disponível em https://www.apda.pt/site/ficheiros_eventos/201302260954-ft_qi_13_amonio.pdf

ZÁRATE, Moisés – ETANOL E POLUIÇÃO. Rotary Club Jahu Leste. (Jul. 2012).[Consult. 13 Mai. 2019]. Disponível em http://clubeficaz.com.br/clubes/rotaryjahuleste/2012/07/17/etanol-e-poluicao

RIBEIRO, Joana André Matias - Propostas de mitigação para as emissões poluentes das aeronaves : Aplicação ao Aeroporto de Lisboa. Lisboa: Instituto Superior Técnico, 2010. 168p. Dissertação de Mestrado

STEVENSON, Doug - What is Firebase? The complete story, abridged. (Set. 2018). [Consult. 8 Abr. 2019]. Disponível em https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0

Android - Android Studio. [Consult 27 Abr 2019]. Disponível em https://developer.android.com/studio

Firebase - Spark Plan. [Consult 21 Abr 2019]. Disponível em https://firebase.google.com/pricing?authuser=0

Fritzing Software - Friends of Fritzing foundation. [Consult. 19 Mai. 2019]. Disponível em https://fritzing.org/download/

Firebase – Documentation. [Consult 21 Abr 2019]. Disponível em https://firebase.google.com/docs/libraries

Anexos

Código Arduíno IDE

```
#include <DHT.h>
#include <MQ135.h>
#include <Wire.h>
#include <LCD.h>
#include <LiquidCrystal_I2C.h>
//DHT trocar vcc pelos dados, senão n dá valores

#define LED 13
//constantes mq135
#define RLOAD 22.0  //resistencia na placa
#define RZERO 879.13  //calibracao resistencia do CO2 atmosferico
//mq135
int valor;
int sensorPino = A1;
int sensorValor = 0;
MQ135 gasSensor = MQ135(A1);

#define DHTPIN A0    //o pino onde está conectado
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
LiquidCrystal_I2C lcd(0x27,2,1,0,4,5,6,7);
const int ledPin = 6;

void setup() {
  // put your setup code here, to run once:

  dht.begin();
```

```
  pinMode(sensorPino, INPUT);
  lcd.setBacklightPin(3, POSITIVE);
  lcd.setBacklight(HIGH);
  lcd.begin(16, 2);
  lcd.clear();
  Serial.begin(9600);
  pinMode(LED, OUTPUT);

}

void loop() {
  char c;
  if (Serial.available())
  {
    c = Serial.read();
    if(c=='t');
    digitalWrite(LED, HIGH);
    readSensores();
    digitalWrite(LED, LOW);
  }

  delay(1000);
  // put your main code here, to run repeatedly:
  int chk = dht.read(DHTPIN);
  float temp=(dht.readTemperature());
  float hum=(dht.readHumidity());

  //DHT11
  lcd.print("Temp:          C");
  lcd.setCursor(6,0);
```

```
lcd.print(temp);
lcd.setCursor(0,1);
lcd.println("Hum:          %");
lcd.setCursor(6,1);
lcd.print(hum);
delay(2000);

lcd.clear();
//MQ135
lcd.setCursor(0,0);
float co = gasSensor.getCOPPM();
lcd.println("CO:         ppm");
lcd.setCursor(6,0);
lcd.print(co);

lcd.setCursor(0,1);
float co2 = gasSensor.getCO2PPM();
lcd.println("CO2:        ppm");
lcd.setCursor(6,1);
lcd.print(co2);

delay(2000);
lcd.clear();

lcd.setCursor(0,0);
float etanol = gasSensor.getEthanolPPM();
lcd.println("Etanol:     ppm");
lcd.setCursor(7,0);
lcd.print(etanol);
```

```
  lcd.setCursor(0,1);
  float toluene = gasSensor.getToluenePPM();
  lcd.print("Tolueno:     ppm");
  lcd.setCursor(8,1);
  lcd.print(toluene);

  delay(2000);
  lcd.clear();

  lcd.setCursor(0,0);
  float nh4 = gasSensor.getNH4PPM();
  lcd.println("NH4:         ppm");
  lcd.setCursor(6, 0);
  lcd.print(nh4);

  lcd.setCursor(0,1);
  float acetona = gasSensor.getAcetonePPM();
  lcd.println("Acetona:     ppm");
  lcd.setCursor(8, 1);
  lcd.print(acetona);

  delay(2000);
  lcd.clear();
}

void readSensores() {
  delay(1000);
  //DHT11
  int chk = dht.read(DHTPIN);
  float temp = dht.readTemperature();
```

```
float hum = dht.readHumidity();

//MQ135
float co = gasSensor.getCOPPM();
float co2 = gasSensor.getCO2PPM();
float etanol = gasSensor.getEthanolPPM();
float toluene = gasSensor.getToluenePPM();
float nh4 = gasSensor.getNH4PPM();
float acetona = gasSensor.getAcetonePPM();

if (isnan(hum) || isnan(temp)) {
  Serial.println("Failed to read from DHT sensor!");
  //return;
  }

//DHT11
float hic = dht.computeHeatIndex(temp, hum, false);
Serial.print("Humidade: ");
Serial.print(hum);
Serial.print(" %\t");
Serial.print("\nTemperatura: ");
Serial.print(temp);
Serial.print(" *C ");
Serial.print("\nHeat index: ");
Serial.print(hic);
Serial.print(" *C ");

//MQ135
Serial.print("\nCO: ");
Serial.print(co);
```

```
  Serial.print(" ppm ");
  Serial.print("\nCO2: ");
  Serial.print(co2);
  Serial.print(" ppm ");
  Serial.print("\nEtanol: ");
  Serial.print(etanol);
  Serial.print(" ppm ");
  Serial.print("\nTolueno: ");
  Serial.print(toluene);
  Serial.print(" ppm ");
  Serial.print("\nNH4: ");
  Serial.print(nh4);
  Serial.print(" ppm ");
  Serial.print("\nAcetona: ");
  Serial.print(acetona);
  Serial.print(" ppm");

}
```

Código Aplicação Android

MainActivity

```
package com.JRCElectronics.prototipo_10;


//Bibliotecas
import androidx.appcompat.app.AppCompatActivity;
import android.content.Context;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.Intent;
import android.os.Build;
import android.os.Bundle;
import android.os.Handler;
import android.text.TextUtils;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
```

```
import android.widget.Toast;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.lang.reflect.Method;
import java.util.Set;
import java.util.UUID;

public class MainActivity extends AppCompatActivity implements SensorEventListener {
    Button startButton, sendButton,clearButton,stopButton, btnEnv;; //Para os objectos XML
    TextView textView, txtArduino, Light, pressure, temp, humi; //Para apresentação dos dados
    EditText editText, editText2; //campo para editar na interface gráfica
    Handler h;
    Thread thread;
    boolean deviceConnected=false;
    byte buffer[];
    int bufferPosition;
    boolean stopThread;
    final int RECIEVE_MESSAGE = 1;        // Status  para o Handler

    private static final String TAG = "JRC Air Quality";
    private final String DEVICE_ADDRESS="00:18:E4:36:3B:32"; //private final String DEVICE_NAME="MyBTBee";
```

```
    private final UUID PORT_UUID = UUID.fromString("00001101-0000-1000-8000-00805f9b34fb");//Serial Port Service ID

    private BluetoothDevice device;

    private BluetoothSocket socket;

    private BluetoothAdapter btAdapter = null;

    private BluetoothSocket btSocket = null;

    private OutputStream outputStream;

    private InputStream inputStream;

    private Sensor mLight, mPressure, mTemp, mHumi;

    private SensorManager sensorManager;

    private StringBuilder sb = new StringBuilder();

    private ConnectedThread mConnectedThread;


    DatabaseReference databaseReference;


    //Bluetooth

    private BluetoothSocket createBluetoothSocket(BluetoothDevice device) throws IOException {

        if (Build.VERSION.SDK_INT >= 10) {

            try {

                final Method m = device.getClass().getMethod("createInsecureRfcommSocketToServiceRecord", new Class[]{UUID.class});

                return (BluetoothSocket) m.invoke(device, PORT_UUID);

            } catch (Exception e) {

                Log.e(TAG, "Could not create Insecure RFComm Connection", e);
```

```
        }
    }
    return device.createRfcommSocketToServiceRecord(PORT_UUID);
}

//Inicialização da ligação bluetooth
public boolean BTinit() {
    boolean found=false;
    BluetoothAdapter bluetoothAdapter=BluetoothAdapter.getDefaultAdapter();
    if (bluetoothAdapter == null) {
        Toast.makeText(getApplicationContext(),"Dispositivo não suporta Bluetooth",Toast.LENGTH_SHORT).show();
    }
    if(!bluetoothAdapter.isEnabled())
    {
        Intent enableAdapter = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        startActivityForResult(enableAdapter, 0);
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    Set<BluetoothDevice> bondedDevices = bluetoothAdapter.getBondedDevices();
```

```
    if(bondedDevices.isEmpty())
    {
        Toast.makeText(getApplicationContext(),"Por favor emparalhe com um dispositivo primeiro",Toast.LENGTH_SHORT).show();
    }
    else
    {
        for (BluetoothDevice iterator : bondedDevices)
        {
            if(iterator.getAddress().equals(DEVICE_ADDRESS))
            {
                device=iterator;
                found=true;
                break;
            }
        }
    }
    return found;
}

//Acções após a ligação estabelica
public boolean BTconnect() {
    boolean connected=true;
    try {
        socket = device.createRfcommSocketToServiceRecord(PORT_UUID);
```

```
            socket.connect();
        } catch (IOException e) {
            e.printStackTrace();
            connected=false;
        }
        if(connected)
        {
            try {
                outputStream=socket.getOutputStream();
            } catch (IOException e) {
                e.printStackTrace();
            }
            try {
                inputStream=socket.getInputStream();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        return connected;
    }

    private void checkBTState() {
        // Verifica se o dispositivo suporta Bluetooth e de seguida verifica se o mesmo está ligado
        // Emulador não suporta Bluetooth por isso vai devolver nulo
```

```
        if (btAdapter == null) {
            errorExit("Fatal Error", "Bluetooth não é suportado");
        } else {
            if (btAdapter.isEnabled()) {
                Log.d(TAG, "...Bluetooth Ligado...");
            } else {
                //Notifica o utilizador para permitir a liqação bluetooth
                Intent enableBtIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
                startActivityForResult(enableBtIntent, 1);
            }
        }
    }
    //Bluetooth

    //Recebe os dados do Arduino e coloca no buffer
    private class ConnectedThread extends Thread {
        private final InputStream mmInStream;
        private final OutputStream mmOutStream;

        public ConnectedThread(BluetoothSocket socket) {
            InputStream tmpIn = null;
            OutputStream tmpOut = null;
```

```
        //Obtem os streams input e output utilizando os objectostemp(tmpIn e tmpOut) pois os membros de Stream são finais
        try {
            tmpIn = socket.getInputStream();
            tmpOut = socket.getOutputStream();
        } catch (IOException e) {
        }

        mmInStream = tmpIn;
        mmOutStream = tmpOut;
    }

    public void run() {
        byte[] buffer = new byte[256];  // buffer guardados para o stream
        int bytes; // bytes devolvidos do método read()
        //Continua a receber do InputStream até ocorrer uma exeção
        while (true) {
            try {
                // Continua a receber do InputStream até ocorrer uma exeção
                bytes = mmInStream.read(buffer);            //Vai buscar o número de bytes e a mensagem do Buffer
                h.obtainMessage(RECIEVE_MESSAGE, bytes, -1, buffer).sendToTarget();     // Send to message queue Handler
            } catch (IOException e) {
                break;
```

```
            }
        }
    }

    // Chama esta função para a MainActivity para enviar a informação para o Android
    public void write(String message) {
        Log.d(TAG, "...Data to send: " + message + "...");
        byte[] msgBuffer = message.getBytes();
        try {
            mmOutStream.write(msgBuffer);
        } catch (IOException e) {
            Log.d(TAG, "...Error data send: " + e.getMessage() + "...");
        }
    }
}

//Prepara o Arduino para a leitura dados á aplicação
void beginListenForData() {
    final Handler handler = new Handler();
    stopThread = false;
    buffer = new byte[1024];
    Thread thread  = new Thread(new Runnable()
    {
        public void run()
```

```
{
   while(!Thread.currentThread().isInterrupted() && !stopThread)
   {
      try
      {
         int byteCount = inputStream.available();
         if(byteCount > 0)
         {
            byte[] rawBytes = new byte[byteCount];
            inputStream.read(rawBytes);
            final String string=new String(rawBytes,"UTF-8");
            handler.post(new Runnable() {
               public void run()
               {
                  textView.append(string);
               }
            });
         }
      }
      catch (IOException ex)
      {
         stopThread = true;
      }
   }
```

```
            }
        });

        thread.start();
    }

    public void setUiEnabled(boolean bool) {
        startButton.setEnabled(!bool);
        sendButton.setEnabled(bool);
        stopButton.setEnabled(bool);
        textView.setEnabled(bool);
    }

    //Função para excepção em caso de erro
    private void errorExit(String title, String message) {
        Toast.makeText(getBaseContext(), title + " - " + message,
Toast.LENGTH_LONG).show();
        finish();
    }

    //Adicionar utilizaor á firebase
    public void addUtilizador(){

        String UtilizadorNome = editText2.getText().toString();          //Nome do utilizador
        String UtilizadorArduino = txtArduino.getText().toString();      //Dados Arduino
```

```
String UtilizadorAndroidLuz = Light.getText().toString();       //Sensor de Luz do Android

String UtilizadorAndroidPressao = pressure.getText().toString();  //Sensor de Pressão do Android

String UtilizadorAndroidTemp = temp.getText().toString();         //Sensor de temperatura do Android

String UtilizadorAndroidHumi = humi.getText().toString();         //Sensor de humidade do Android


if (!TextUtils.isEmpty(UtilizadorNome)){ //Se o campo nome não estiver vazio

    String id = databaseReference.push().getKey(); //Gera um id para utilizador


    //Adiciona utilizador

    Utilizadores utilizador = new Utilizadores(id, UtilizadorNome, UtilizadorArduino, UtilizadorAndroidLuz, UtilizadorAndroidPressao, UtilizadorAndroidTemp, UtilizadorAndroidHumi);

    databaseReference.child(id).setValue(utilizador);

} else { //Caso o campo nome esteja vazio, mostra a mensagem "Por favor introduza um nome"

    Toast.makeText(MainActivity.this,"Por favor introduza um nome", Toast.LENGTH_LONG).show();

    }

}

//Método para inicializar os componentes

@Override

protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
```

```
//Variáveis necessárias para ligação com o XML

setContentView(R.layout.activity_main);

startButton = (Button) findViewById(R.id.buttonStart);          //inicia a ligação bluetooth com o arduino

sendButton = (Button) findViewById(R.id.buttonSend);          //Obtem o valor introduzido na variável EditText para comunicar com o arduino

clearButton = (Button) findViewById(R.id.buttonClear);        //Botão para limpar os dados devolvidos pela variável txtarduino

stopButton = (Button) findViewById(R.id.buttonStop);          //Botão para parar a ligação com Arduino e Bluetooth

editText = (EditText) findViewById(R.id.editText);          //Campo para introduzir valores para os dados do Arduino

editText2 =(EditText) findViewById(R.id.editText2);          //Campo para introduzir o nome do utilizador

textView = (TextView) findViewById(R.id.textView);                  //Campo onde será apresentado os dados do arduino        //Botão LED OFF

txtArduino = (TextView) findViewById(R.id.txtArduino);                  //Devolve informaçãoobtida pelo arduino

btnEnv = (Button)findViewById(R.id.btnSave);          //Envia o registo(Nome, leituras) para a firebase

setUiEnabled(false);

//Variáveis necessárias para ligação com o XML


//Handler

h = new Handler() {

    public void handleMessage(android.os.Message msg) {

        switch (msg.what) {
```

```
                case RECIEVE_MESSAGE:                                     // Se receber a
mensagem
                    byte[] readBuf = (byte[]) msg.obj;
                    String strIncom = new String(readBuf, 0, msg.arg1);                 // Cria uma
string de uma array de bytes
                    sb.append(strIncom);                                // Acrescenta a string
                    int endOfLineIndex = sb.indexOf("\r\n");                    // Determina o dim
da linha
                    if (endOfLineIndex > 0) {                                   // se o fim da linha,
                       String sbprint = sb.substring(0, endOfLineIndex);            // Extrai a String
                       sb.delete(0, sb.length());                               // Limpa
                       txtArduino.setText("Dados do Arduino: " + "\r\n" +  sbprint);            // faz
update ao TextView
                    }
                    break;
            }
        }
        ;
    };
    //Handler

    //Bluetooth
    btAdapter = BluetoothAdapter.getDefaultAdapter();       // Busca adaptador bluetooth
    checkBTState();

    //Bluetooth
```

```
//Sensores Android
//Variáveis para que seja efectuada a ligação com o XML da aplicação
Light = (TextView) findViewById(R.id.Light); //Luminosidade

pressure = (TextView) findViewById(R.id.pressure); //Pressão

temp = (TextView) findViewById(R.id.temp); //Temperatura

humi = (TextView) findViewById(R.id.humi); //Humidade

Log.d(TAG, "onCreate: initializing Sensor Services");
sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);

//Para cada um dos sensores, caso os mesmos não sejam detectados devolve a mensagem "Não suporta sensor"
//Luminosidade
mLight = sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
if (mLight != null) {
   sensorManager.registerListener(MainActivity.this, mLight, SensorManager.SENSOR_DELAY_NORMAL);
   Log.d(TAG, "onCreate: Registered Light listener");
} else {
   Light.setText("Sensor Luz: Não suporta sensor");
}
```

```
        //Pressão
        mPressure = sensorManager.getDefaultSensor(Sensor.TYPE_PRESSURE);
        if (mPressure != null) {
            sensorManager.registerListener(MainActivity.this, mPressure,
SensorManager.SENSOR_DELAY_NORMAL);
            Log.d(TAG, "onCreate: Registered Pressure listener");
        } else {
            pressure.setText("Sensor Pressão: Não suporta sensor");
        }

        //Temperatura
        mTemp =
sensorManager.getDefaultSensor(Sensor.TYPE_AMBIENT_TEMPERATURE);
        if (mTemp != null) {
            sensorManager.registerListener(MainActivity.this, mTemp,
SensorManager.SENSOR_DELAY_NORMAL);
            Log.d(TAG, "onCreate: Registered Temperature listener");
        } else {
            temp.setText("Sensor Tempartura: Não suporta sensor");
        }

        //Humidade
        mHumi = sensorManager.getDefaultSensor(Sensor.TYPE_RELATIVE_HUMIDITY);
        if (mHumi != null) {
            sensorManager.registerListener(MainActivity.this, mHumi,
SensorManager.SENSOR_DELAY_NORMAL);
```

```
            Log.d(TAG, "onCreate: Registered Humidity listener");
        } else {
            humi.setText("Sensor Humidade: Não suporta sensor");
        }
        //Sensores Android


        //Firebase
        databaseReference = FirebaseDatabase.getInstance().getReference("utilizadores");


        btnEnv.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                addUtilizador();
            }
        });
        //Firebase
    }


    //Método para resumir atividade caso alguma interrupção na actividade tal como mudar de aplicação
    @Override
    public void onResume() {
        super.onResume();


        Log.d(TAG, "...onResume - try connect...");
```

```
        BluetoothDevice device = btAdapter.getRemoteDevice(DEVICE_ADDRESS);

        try {
            btSocket = createBluetoothSocket(device);
        } catch (IOException e) {
            errorExit("Fatal Error", "In onResume() and socket create failed: " + e.getMessage() +
".");
        }

        btAdapter.cancelDiscovery();

        // Estabelece uma ligação. Isto bloqueia até ligar
        Log.d(TAG, "...A ligar...");
        try {
            btSocket.connect();
            Log.d(TAG, "....Ligação ok...");
        } catch (IOException e) {
            try {
                btSocket.close();
            } catch (IOException e2) {
                errorExit("Fatal Error", "In onResume() and unable to close socket during connection
failure" + e2.getMessage() + ".");
            }
        }
```

```
        //Cria um stream de dados para comunicar com o servidor
        Log.d(TAG, "...Criar Socket...");

        mConnectedThread = new ConnectedThread(btSocket);
        mConnectedThread.start();
    }

    //Método para colocar atividade em pausa caso aplicação seja minimizada
    @Override
    public void onPause() {
        super.onPause();

        Log.d(TAG, "...Em onPause()...");

        try {
            btSocket.close();
        } catch (IOException e2) {
            errorExit("Fatal Error", "Em onPause() e falhou o fecho do socket." + e2.getMessage()
+ ".");
        }
    }

    //Função obrigatória para implementação do SensorEventListener
    @Override
```

```
public void onAccuracyChanged(Sensor sensor, int i){
}

//Função obrigatória para implementação do SensorEventListener
@Override
public void onSensorChanged(SensorEvent sensorEvent){
    Sensor sensor = sensorEvent.sensor;

        //Apresentação dos valores do Sensor da Luz
    if (sensor.getType() == Sensor.TYPE_LIGHT){
        Light.setText("Luz: " + sensorEvent.values[0]);

        //Apresentação dos valores do Sensor da Pressão
    } else if (sensor.getType() == Sensor.TYPE_PRESSURE){
        pressure.setText("Pressão: " + sensorEvent.values[0]);

        //Apresentação dos valores do Sensor da Temperatura
    } else if (sensor.getType() == Sensor.TYPE_AMBIENT_TEMPERATURE){
        temp.setText("Temperatura: " + sensorEvent.values[0]);

        //Apresentação dos valores do Sensor de Humidade
    } else if (sensor.getType() == Sensor.TYPE_RELATIVE_HUMIDITY){
        humi.setText("Humidade: " + sensorEvent.values[0]);
    }
```

```
    }
    //Iniciar a ligação bluetooth da aplicação através da botão "Começar"
    public void onClickStart(View view) {
        if(BTinit())
        {
            if(BTconnect())
            {
                setUiEnabled(true);
                deviceConnected=true;
                beginListenForData();
                textView.append("\nLigação Estabelecida!\n" + "\r\n");
            }
        }
    }

    //Acção ao carregar no botão Send
    public void onClickSend(View view) {
        String string = editText.getText().toString(); //Recebe o input colocado no editext, neste caso a letra t
        string.concat("\n");
        try {
            outputStream.write(string.getBytes());
        } catch (IOException e) {
            e.printStackTrace();
        }
```

```
        textView.append("\nDados Recebidos:"+string+"\n");
    }

    //Ao Carregar no botão Stop, desliga a ligação bluetooth
    public void onClickStop(View view) throws IOException {
        stopThread = true; //Para o Thread
        outputStream.close(); //Fecha o stream de saída
        inputStream.close(); //Fecha o Stream de entrada no editext
        socket.close(); //Fecha o Socket Bluetooth
        setUiEnabled(false);
        deviceConnected=false; //Desliga Bluetooth
        textView.append("\nLigação Desligada!\n"); //Adiciona a mensagem referida
    }

    //Limpa os dados recebidos pelo arduino
    public void onClickClear(View view) {
        textView.setText("");
    }
}
```

Activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
tools:context=".MainActivity">

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="4dp">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical" />
    </ScrollView>

    <EditText
        android:id="@+id/editText2"
        android:layout_width="391dp"
        android:layout_height="90dp"
        android:layout_above="@+id/editText"
```

```
        android:layout_alignParentStart="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentEnd="true"
        android:layout_alignParentRight="true"
        android:layout_marginStart="0dp"
        android:layout_marginLeft="0dp"
        android:layout_marginTop="10dp"
        android:layout_marginEnd="4dp"
        android:layout_marginRight="4dp"
        android:layout_marginBottom="-54dp"
        android:hint="Introduza o seu nome" /><!--Campo para introduzir o valor dos dados do Arduino-->


    <EditText
        android:id="@+id/editText"
        android:layout_width="389dp"
        android:layout_height="wrap_content"
        android:layout_alignParentStart="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_alignParentEnd="true"
        android:layout_alignParentRight="true"
        android:layout_centerInParent="true"
        android:layout_marginStart="0dp"
        android:layout_marginLeft="0dp"
```

```
    android:layout_marginTop="75dp"
    android:layout_marginEnd="6dp"
    android:layout_marginRight="6dp"
    android:hint="Introduza a letra t" /> <!--Campo para introduzir o nome-->

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/btn_COM"
    android:id="@+id/buttonStart"
    android:layout_below="@+id/editText"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:onClick="onClickStart"/>  <!--Botão Começar-->

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/btn_GER"
    android:id="@+id/buttonSend"
    android:onClick="onClickSend"
    android:layout_below="@+id/editText"
    android:layout_toRightOf="@+id/buttonStart"
    android:layout_toEndOf="@+id/buttonStart" /> <!--Botão para gerar dos dados do
Arduino-->
```

```
<Button
    android:id="@+id/buttonStop"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/btn_PAR"
    android:layout_below="@+id/editText"
    android:layout_toRightOf="@+id/buttonSend"
    android:layout_toEndOf="@+id/buttonSend"
    android:onClick="onClickStop"/> <!--Botão Parar-->

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/btn_LIM"
    android:id="@+id/buttonClear"
    android:layout_below="@+id/editText"
    android:layout_toRightOf="@+id/buttonStop"
    android:layout_toEndOf="@+id/buttonStop"
    android:onClick="onClickClear"/> <!--Botão Limpar-->

<Button
    android:id="@+id/btnSave"
    android:layout_below="@+id/buttonStop"
```

```
        android:layout_marginBottom="25dp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="@string/btn_ENV" /> <!--Botão para enviar os dados para Firebase-->

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="200dp"
        android:layout_below="@+id/humi"
        android:layout_alignEnd="@+id/editText"
        android:layout_alignRight="@+id/editText"
        android:layout_alignParentStart="true"
        android:layout_alignParentLeft="true"
        android:layout_marginEnd="0dp"
        android:layout_marginRight="0dp"
        android:singleLine="true" /> <!--Text View que apresenta os dados da ligação bluetooth
e do Arduino-->

    <TextView
        android:id="@+id/txtArduino"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
```

```
    android:layout_below="@+id/textView"
    android:layout_alignEnd="@+id/editText"
    android:layout_alignRight="@+id/editText"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:layout_marginLeft="13dp"
    android:layout_marginTop="388dp"
    android:layout_marginEnd="1dp"
    android:layout_marginRight="1dp"
    android:text="" /> <!--Text View que apresenta os dados do Arduino-->

<TextView
    android:id="@+id/Light"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/btnSave"
    android:layout_centerHorizontal="true"
    android:text="Sensor Luz = ????"
    android:textAppearance="?android:attr/textAppearanceMedium" /> <!--Sensor de Luz-->

<TextView
    android:id="@+id/pressure"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```
        android:layout_below="@+id/Light"
        android:layout_centerHorizontal="true"
        android:text="Sensor Pressão = ????"
        android:textAppearance="?android:attr/textAppearanceMedium" /> <!--Sensor de Pressão-->

    <TextView
        android:id="@+id/temp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/pressure"
        android:layout_centerHorizontal="true"
        android:text="Sensor Temperatura= ????"
        android:textAppearance="?android:attr/textAppearanceMedium" /> <!--Sensor de Temperatura-->

    <TextView
        android:id="@+id/humi"
        android:layout_marginBottom="25dp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/temp"
        android:layout_centerHorizontal="true"
        android:text="Sensor Humidade = ????"
```

android:textAppearance="?android:attr/textAppearanceMedium" /> <!--Sensor de Humidade-->

</RelativeLayout>