



FCEE – Faculdade de Ciências Exatas e da Engenharia
Projeto de licenciatura em Engenharia Informática

Sistema de monitorização de ambiente



Discente: Matheus Alves Moreira nº 2072313

Orientadora: Prof.^a Dra. Karolina Baras

Agradecimentos

À Prof.^a Dra. Karolina Baras, supervisora e orientadora deste projeto, um agradecimento especial pelo tempo prestado em meu auxílio assim como todo o apoio e estímulo que me deu. Sua prestabilidade foi-me muito valorizada assim como suas correções e orientações na decorrência da elaboração deste projeto.

Ao Prof. Dr. David Aveiro por ter disponibilizado seu curto tempo para me auxiliar com questões relativas às bases de dados.

Ao meu grande amigo e vizinho Emanuel Fernandes por ter testado o projeto em sua casa utilizando todo o meu equipamento, além de me ter fornecido feedback importante.

À minha querida e saudosa mãe por me ter auxiliado a escolher um design mais fluido (apesar de eu não ter conhecimentos aprofundados em design dei o meu melhor para tentar acompanhar as orientações que me foram dadas).

Ao pastor Neto Saldanha, apóstolo de Cristo, por todo apoio prestado, toda a força e palavra de ânimo que me foi dado e inclusive por algumas ideias que me foram passadas.

E claro, sem me esquecer, agradeço fielmente a Deus por me ter dado forças nos momentos de elevadíssimo stress, por ter ouvido minhas orações e me levantado o ânimo, e por ter utilizado as pessoas certas no meu caminho para que Sua Palavra fosse honrada através de minhas ações.

*“Consagre ao Senhor tudo o que fizer, e seus planos serão bem-sucedidos.”
Provérbios 16:3*

Índice

Agradecimentos	2
Introdução.....	4
Recursos e implementação base	5
• Requisitos Funcionais	5
• Requisitos não funcionais.....	5
• Hardware	6
• Software	6
Funcionamento	8
• Instruções de arranque	9
• Processo servidor	11
• HomePage.....	12
• Login page.....	13
• Registo	14
• Histórico de relatórios	15
• RaspiCheck	16
• Sensors.....	17
• Gerar relatório	18
• Implementação	19
• Processo Leitor	22
• Implementação	23
• Processo BotTelegrama	25
• Implementação	26
• Processo Arduino.....	28
• Implementação	29
Problemas e limitações	31
• Processo servidor	31
• Processo Leitor	31
• Processo BotTelegrama	32
• Processo Arduino.....	32
Testes	33
• Caso 1: Quarto do <i>developer</i>	33
• Caso 2: Casa do vizinho do <i>developer</i>	35
• Caso 3: IURD.....	37
• Conclusões finais.....	37
Conclusão	38
Referências	39

Introdução

No presente ano letivo de 2017/2018 foi-me proposto, por parte da Prof.^a Dra. Karolina Baras, a idealização de um sistema baseado em *Raspberry Pi* com o fim de monitorar um espaço. Com isto em mente, decidi elaborar um projeto no qual foi chamado de ***OlhoRaspi***.

Este projeto procura abordar aspetos da Internet das Coisas (IoT – *Internet of Things*), sendo esta definida, em poucas palavras, como uma extensão da internet utilizando objetos do dia-a-dia, onde cada um desses objetos possui uma forma de intercomunicar com outros ao seu redor, tendo estas capacidades computacionais, de comunicação e de ligação à Internet.

Este termo surgiu pela primeira vez no MIT por parte do grupo de investigação *Auto ID*, tendo suas definições acopladas às tecnologias RFID e outras tecnologias com base em sensores. À medida que o termo se tornou mais universal, suas definições passaram a envolver termos mais gerais [1], e o conceito de IoT foi ganhando terreno na habilidade de conectar objetos físicos inicialmente incapazes de gerar, transmitir e receber dados. [2]

Existem inúmeras aplicações baseadas em IoT, tendo como base o aumento da eficiência (*more output with the same input / the same output with less input*), importante em aplicações industriais, eficiência energética, agricultura, vigilância, entre outros. [3]

O principal objetivo deste projeto é monitorar as condições e características de um determinado espaço físico não muito extenso, como por exemplo uma sala de estudo ou um quarto de arrumações. Serão utilizados sensores para capturar a informação do ambiente e essas mesmas informações serão agrupadas numa plataforma *web* dinâmica. O sistema também dispõe de um *Telegram Bot* para comunicações com utilizadores externos à rede no qual o mesmo se encontra conectado.

O projeto está alojado no GitHub, sendo acessível através do *link*:

<https://github.com/matmorris93/OlhoRaspi3>

Recursos e implementação base

Nesta secção serão discutidos os requisitos funcionais e não funcionais do sistema, assim como todos os equipamentos de *hardware* utilizados e o *software* implementado.

• Requisitos Funcionais

1. O sistema deverá permitir cadastro de novos utilizadores;
2. O sistema deverá ser capaz de emitir relatórios por PDF, manualmente ou automaticamente;
3. O sistema deverá permitir consulta e escrita de dados na base de dados;
4. O sistema deverá possuir sistema de *login*;
5. O sistema deverá permitir consulta do seu estado de funcionamento;
6. O sistema deverá estar conectado a uma rede;
7. O sistema deverá guardar definições relativas ao armazenamento de seus dados;
8. O sistema deverá notificar sempre que uma anomalia é detetada em seu funcionamento;
9. O sistema deverá possuir mecanismos de reconhecimento de voz;
10. O sistema deverá possuir capacidade de comunicação por voz;
11. Cada pedido de cliente deve ser único.
12. O sistema deverá ser capaz de medir temperatura, humidade, pressão, ruído e luminosidade;
13. O sistema deverá ser capaz de detetar o número de pessoas no ambiente onde está instalado;
14. O sistema deverá permitir ao utilizador personalizar as unidades de medição;
15. O sistema só deverá responder a pedidos quando solicitados (*BOT*);
16. O sistema deverá mostrar algumas das suas medições através de displays de sete segmentos;
17. O sistema deve permitir a um utilizador remoto comunicar utilizando colunas de som ou displays de LEDs.
18. O sistema deverá permitir consultar meteorologia.
19. Em caso de dúvidas sobre comandos, o sistema deverá disponibilizar conteúdo de ajuda.

• Requisitos não funcionais

1. O sistema deve estar conectado à rede via WI-FI ou Ethernet;
2. A base de dados do sistema só deve permitir acessos de utilizadores autorizados;
3. O tempo de resposta do sistema não deverá ultrapassar os 10 segundos;
4. O sistema deverá estar disponível 24 horas por dia;
5. O sistema deverá ser compatível com Raspbian;
6. A *app Web* deve ser compatível com qualquer *browser*;
7. O sistema deverá lidar com até 5 utilizadores simultaneamente (excetuando no uso do *bot*);
8. A privacidade dos utilizadores do sistema deve ser mantida;
9. O sistema deve ser fácil de usar;
10. O sistema deverá ser desenvolvido utilizando as linguagens *Python* e *C++*;
11. A *app Web* deverá ser desenvolvida utilizando a *framework Django*;
12. A base de dados do sistema deverá ser *SQL*.
13. Comunicações externas à rede local do sistema devem ser feitas utilizando exclusivamente o *bot*;

14. O sistema deverá trabalhar com tarefas na sua programação;
15. Em caso de memória reduzida, o sistema deverá suspender o armazenamento de novos dados até que seja alvo de alguma limpeza;
16. Em caso de falhas, o utilizador do sistema deverá ser notificado;

- Hardware

A nível de *hardware* foram utilizados os seguintes equipamentos:

- *Raspberry Pi2* modelo B;
- *Arduino Mega 2560* ¹;
- Sensor de temperatura digital *DallasTemperature*;
- 3 Foto-sensores ligado a 3 resistências de 10kOhms;
- *Raspberry SenseHat* com sensor barométrico, sensor de humidade e outros;
- Microfone;
- 3 Displays de 7 segmentos;
- 2 leds comuns;
- 2 leds bicolores;
- PIR ².

- Software

A nível de *software* foi escolhido o **Python** (v3.5) como linguagem de programação principal, implementada no *Raspberry*, sendo utilizada juntamente com a framework *Django* (v2.0) e as seguintes bibliotecas:

1. *MySQLDB* – necessária para conectar um processo *Python* à base de dados *mysql*;
2. *PySerial* – utilizada para a leitura do serial do *Arduino*;
3. *SenseHat* – presente no Raspbian que contém todos os métodos para interagir com o *SenseHat*;
4. *Telepot* – que permite interagir com o *bot* utilizando o protocolo **MTProto** do *Telegram*;
5. *PyDub* – utilizada para o processamento de codecs de áudio;
6. *Google Text-to-Speech API (gTTS)* – utilizada para conversão de texto em voz;
7. *CMUSphinx for Python* – utilizada para a conversão de voz em texto [4];
8. *Matplotlib* – utilizada para a geração de gráficos no gerador de PDF;
9. *Bokeh* – utilizada para a geração de gráficos tanto no processo servidor como no *bot*

A nível de bases de dados foi utilizado o sistema gestor de base de dados **MySQL** (v.15.1 distrib 10.1.23-MariaDB).

Para o *Arduino* foi utilizada a linguagem de programação **C++** e foram importadas as seguintes bibliotecas:

- *Lampadas.h* – Biblioteca pessoal que serve para interagir com os LEDs presentes no circuito do *Arduino*;
- *DisplaySegmentos.h* – Biblioteca pessoal contendo métodos para gerir os 3 displays de sete segmentos presentes no circuito do *Arduino*;
- *OneWire.h* e *DallasTemperature.h* – Duas bibliotecas que contém os métodos para a leitura de valores vindos do sensor de temperatura;
- *TimeLib.h* e *Time.h* – Duas bibliotecas contendo os métodos para lidar com datas e horas;
- *Wire.h* e *IRremote.h* – Duas bibliotecas utilizadas para o leitor IR;
- *Thread.h* e *ThreadController.h* – Duas bibliotecas utilizadas com os métodos de gestão das pseudo-tarefas.

Foram também utilizadas as IDEs **JetBrains PyCharm** (v2017.3.3) para a programação em *Python* e o **Arduino-IDE** (v1.8.6) para a programação em *C++* para o *Arduino*.

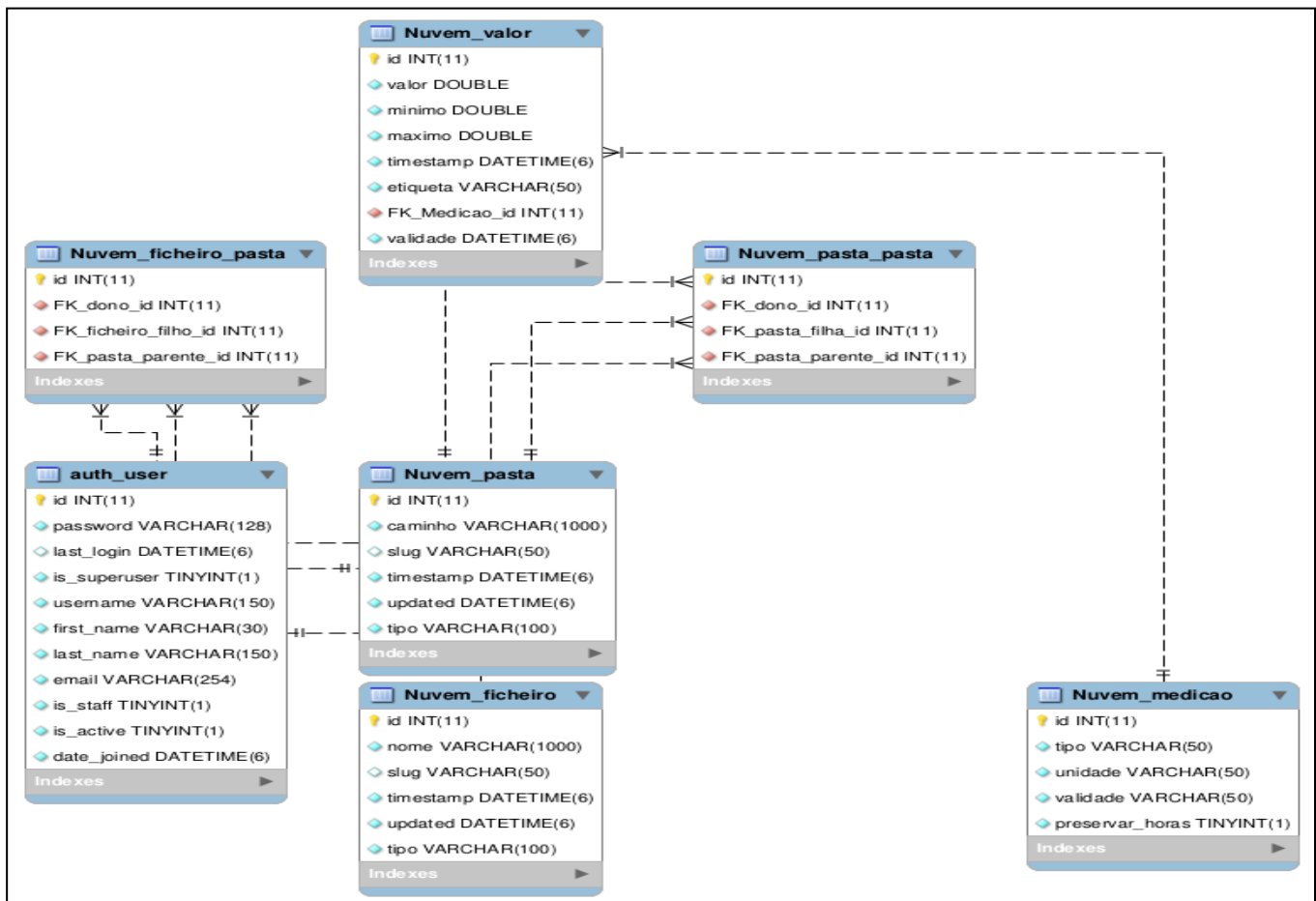


Figura 0 – Esquema da base de dados do projeto

1 - Originalmente o *Arduino* utilizado seria o Uno Rev3, entretanto por conta de problemas de hardware o mesmo foi substituído por um mais potente e com mais pinos de ligação.

2 - A criação do sensor de movimento originalmente era composta por PIR's, mas por conta dos atrasos gerados pelos mesmos, os pirs foram substituídos por 2 foto-sensores melhorando a velocidade de captação de movimento.

Funcionamento

Nesta secção serão abordados aspetos de funcionamento do sistema.

O projeto necessita de quatro processos diferentes para funcionar:

1. O processo do servidor *Django*, onde o utilizador poderá interagir com o sistema através do navegador;
2. O processo leitor, que trabalha buscando os valores vindos do Serial do *Arduino* assim como os valores vindos do *SenseHat*, que será responsável por guardar os valores obtidos pelos sensores na base dados, assim como também será responsável por gerar os relatórios automáticos;
3. O processo BotTelegrama, que será o processo onde correrá o *bot*, agindo basicamente como o *website* a fim de buscar determinadas informações, só que sem tantos privilégios.
4. O processo Arduino, que será executado no *Arduino* para a captação das leituras.

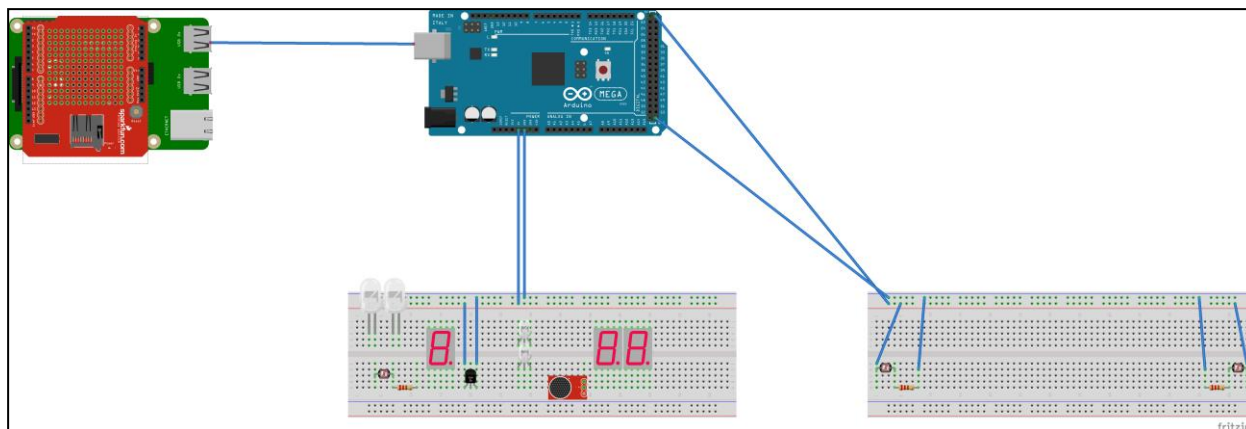


Figura 1 – Circuito simplificado do projeto.

A figura acima foi gerada utilizando o *software Fritzing*. A imagem não representa todas as conexões feitas ao *Arduino* e sim somente como estão dispostas nas *PCB's*. Alguns componentes, nomeadamente o *SenseHat* e resistências não estão em concordância com o modelo real, sendo utilizadas outras representações parecidas, isto porque o programa não disponibilizava os modelos exatos aos implementados no projeto. Para que o utilizador possa interagir com o *Raspberry* é necessário que o mesmo esteja conectado à internet (ou então ligado numa rede local).

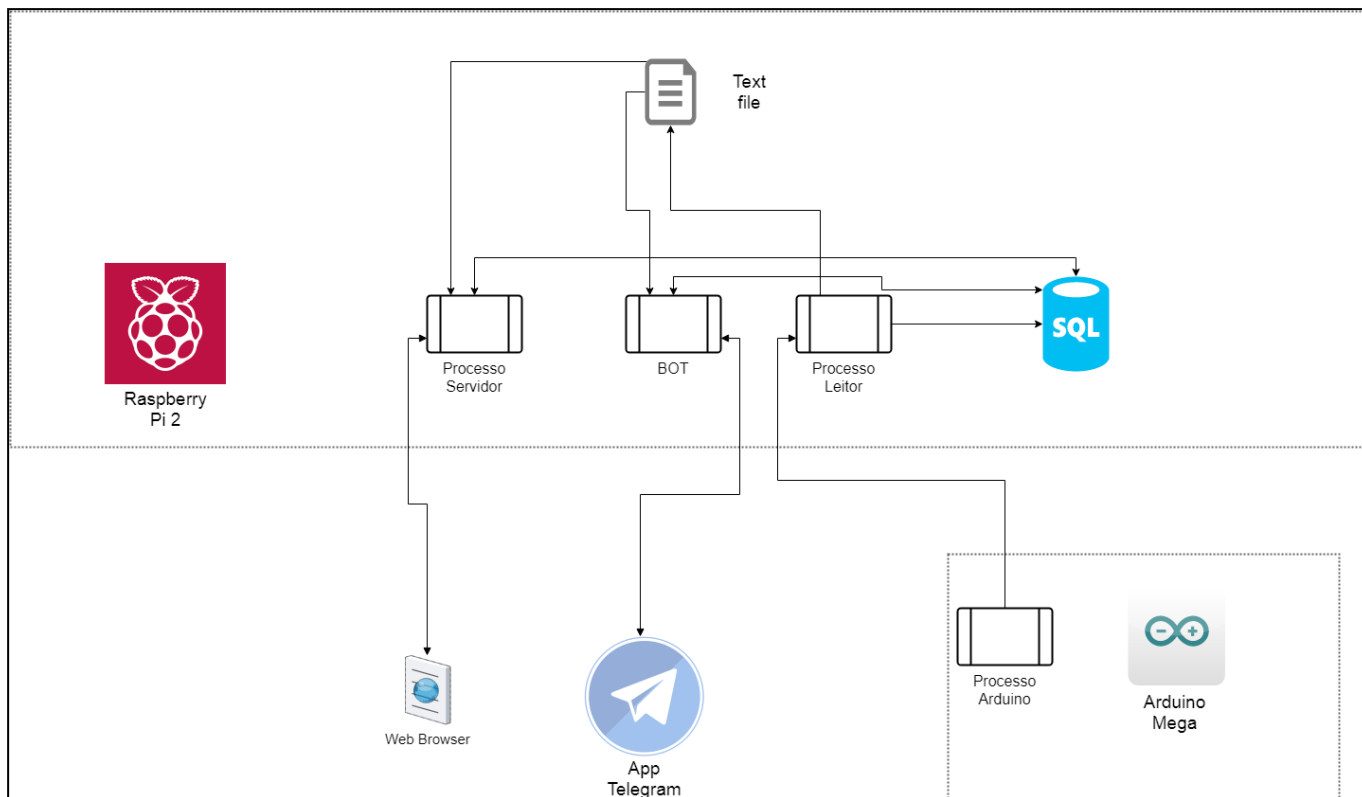


Figura 2 – Diagrama de alto nível do projeto

• Instruções de arranque

Neste sistema, os processos não comunicam entre si diretamente, sendo as ações de cada processo independentes. Cada um dos processos deve ser inicializado manualmente utilizando os seguintes comandos:

1. Primeiro deverá abrir a consola do Raspbian;
2. Após isso, deverá ir para a pasta onde está o projeto executando por exemplo:
`"cd PastaProjeto";`
3. Para executar o processo BotTelegrama execute o seguinte comando:
`"python3 BotTelegrama.py";`
4. Agora para o processo leitor execute o seguinte comando:
`"python3 leituraSomTempLuxPrjEstagio.py";`
5. Por fim para executar o processo servidor deverá executar o comando:
`"python3 manage.py runserver {IP do Raspberry}:8000";`

Uma observação importante: para conhecer o IP do *Raspberry* deverá executar o comando `"ifconfig"`. Se o *Raspberry* estiver conectado à uma rede *WI-FI* deverá procurar por `"wlan0"` e encontrar o campo `"inet"`. O IP atribuído estará aí. Se estiver conectado à uma rede *Ethernet* o procedimento é semelhante, mas em vez de `"wlan0"` deverá procurar por `"eth0"`.⁴

Ao iniciar o programa pela primeira vez, este irá pedir para especificar em dois ficheiros de texto os dados de ligação à base de dados, assim como o *token* gerado para o *bot*. Sem esses dados, nenhum dos processos irá funcionar.

Para definir os dados da base de dados terá de navegar na pasta do projeto e encontrar a pasta “*CloudV1*”. Lá dentro irá encontrar um ficheiro chamado “*DBData*”, caso não o encontre, crie um com esse nome **sem associar extensão**.

Após isso terá que escrever dentro do ficheiro os dados com exatamente esta estrutura, sem parêntesis nem vírgulas:

```
NAME: DATABASE_NAME
USER: DATABASE_USERNAME
PASSWORD: DATABASE_PASSWORD
HOST: DATABASE_HOST
PORT: DATABASE_PORT
END
```

Após isso, terá de especificar o *token* do *bot*. Para tal deverá navegar até a pasta do projeto e encontrar a pasta “*ForBot*” e procurar pelo ficheiro “*leBOT*”, caso não o encontre, crie um com esse nome **sem associar extensão**.

Após isso basta escrever o *token* do *bot* sem parêntesis nem vírgulas e guardar o ficheiro.⁵

4- Dependendo de como possui configurado o Raspberry, wlan0 poderá ser wlan1 ou wlan2, ter sempre em atenção o número de interfaces/tecnologias de rede que estão sendo utilizadas no equipamento;

5- Estou assumindo que são conhecidos os procedimentos de criação de bases de dados e de bots no telegram. No último caso, instruções de como criar o bot são mostradas em consola caso não exista nenhum bot associado.

- Processo servidor

Este processo será responsável por não só alterar definições relativas às medições, como também permite consulta e manuseio dos valores das mesmas na base de dados. A página foi elaborada com o princípio de simplicidade em mente, sem muitos menus e com possibilidade de abrir em qualquer *browser desktop* ou *mobile*.

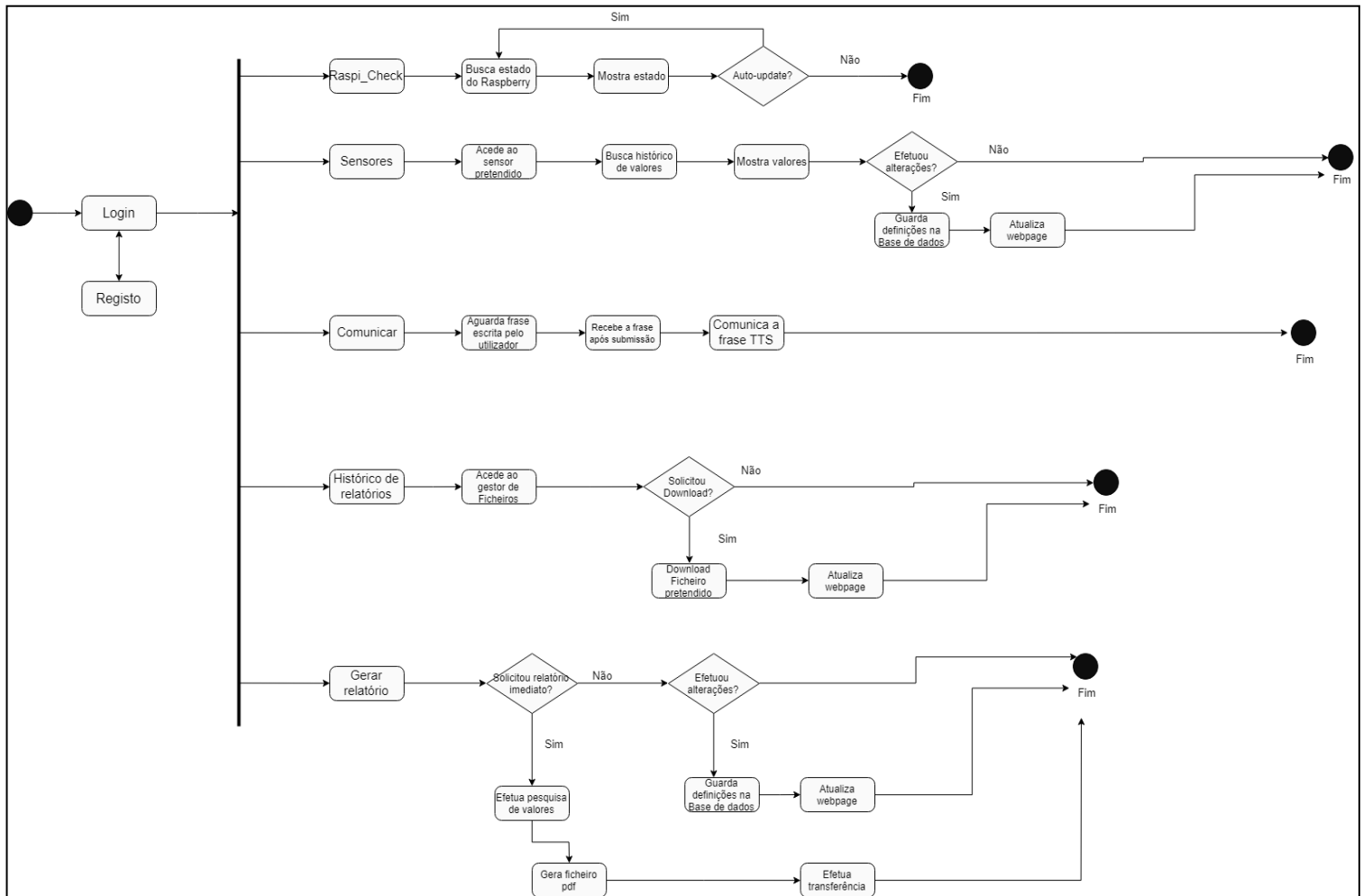


Figura 3 – Diagrama de atividades geral do projeto

A página possui 5 menus principais:

1. **Homepage** – apresenta uma mensagem resumida do estado de funcionamento do *Raspberry* e medições instantâneas dos sensores.
2. **Histórico de relatórios** – permite navegar entre várias pastas que contém os relatórios gerados automaticamente.
3. **RaspiCheck** – apresenta o estado mais detalhado do funcionamento do *Raspberry*, contendo informações como temperatura, estado do armazenamento e outros.

4. **Sensors** – permite aceder às paginas que apresentam medições ao minuto, à hora e à semana, permitindo efetuar alterações a certas definições. Esta também inclui a página de geração de relatórios e também permite especificar que medições devem ser incluídas no relatório e com que frequência os relatórios serão gerados.
5. **Comunicar** – uma página simples que permite enviar mensagens ao *Raspberry* e fazê-lo narrar a frase recebida (como uma espécie de “correio de voz”).

 HomePage

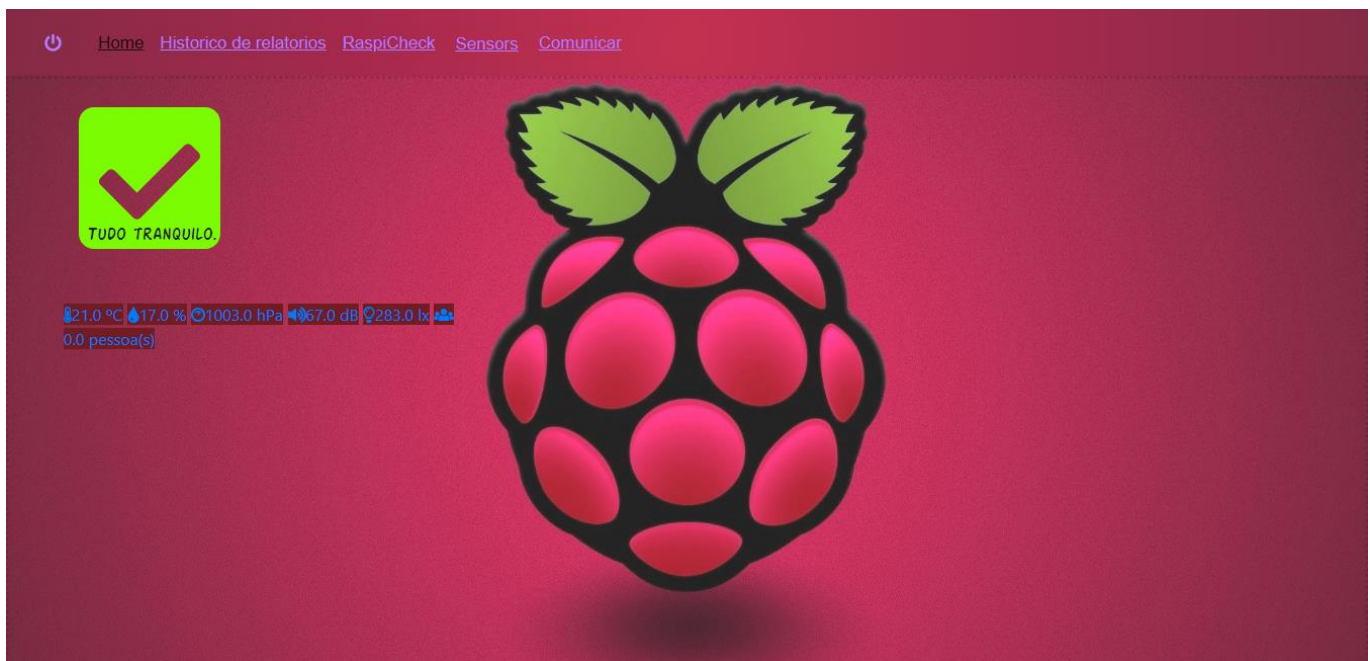
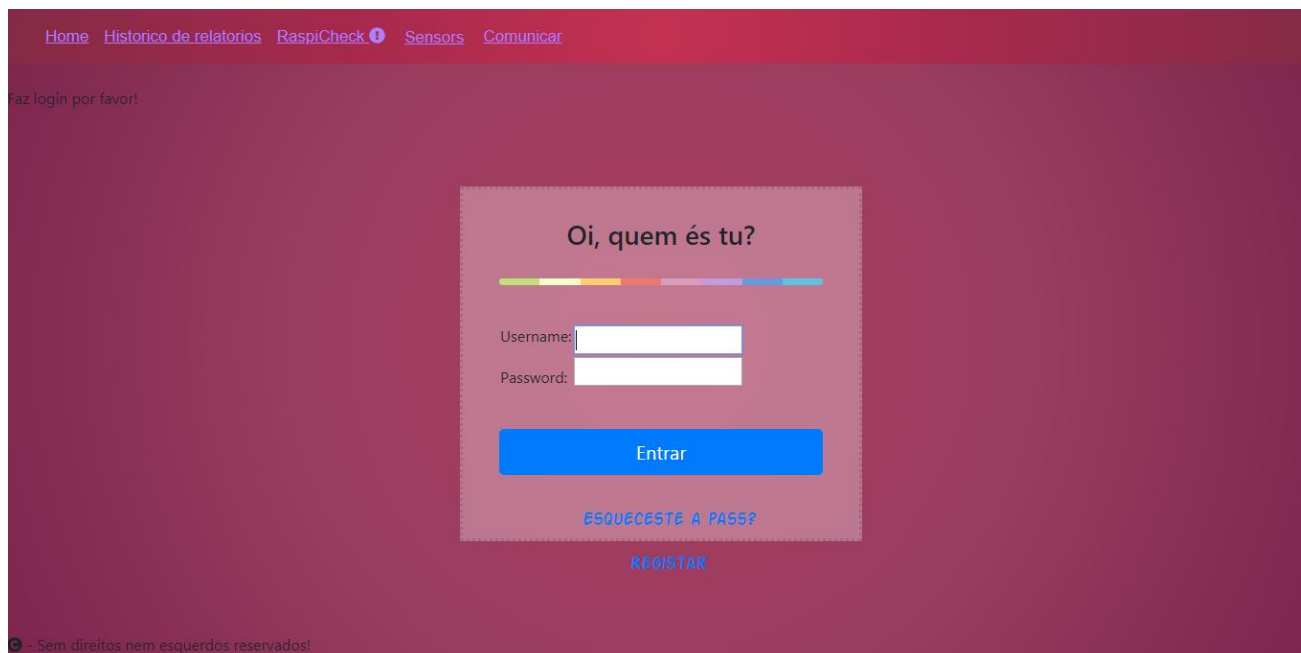


Figura 3.0 – HomePage

Trata-se de uma página onde as informações em tempo real das medições do ambiente, juntamente com uma mensagem breve do estado do sistema, são mostradas.

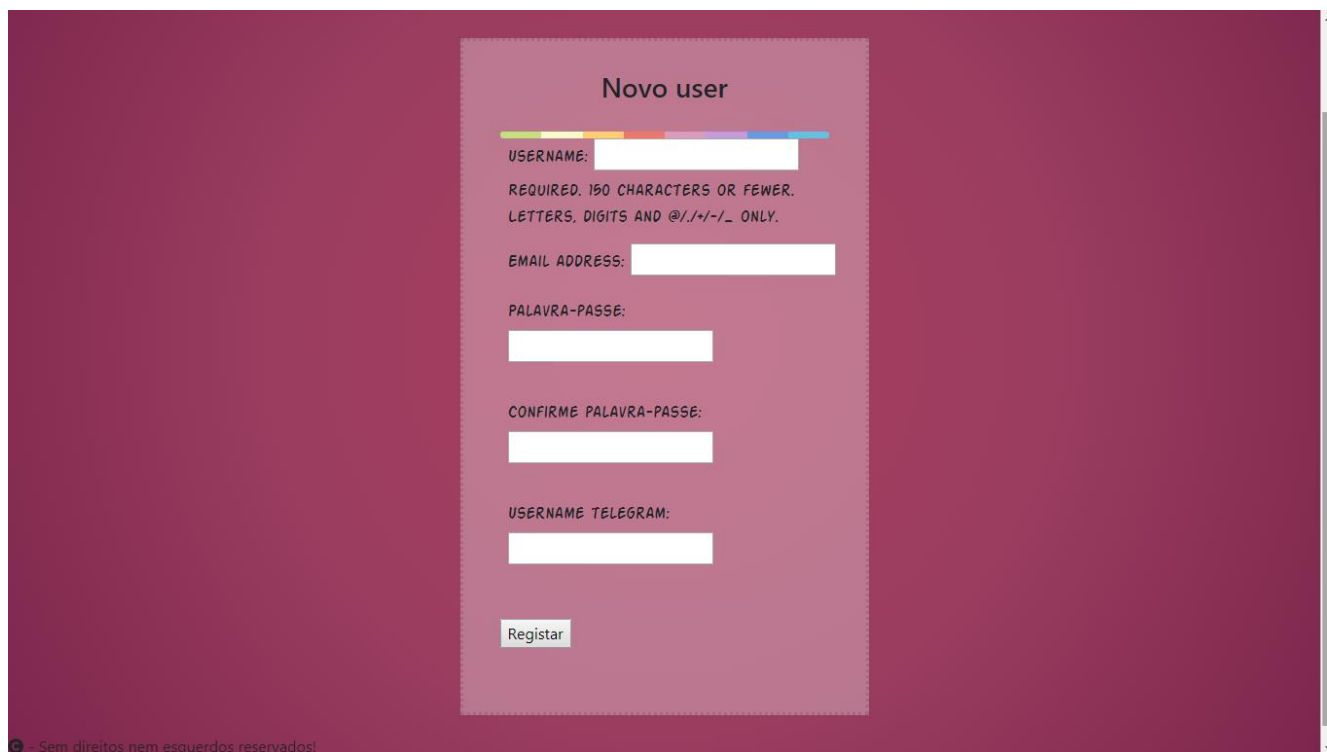
Login page



The screenshot shows a web application's login page. At the top, there is a navigation bar with links: Home, Historico de relatorios, RaspiCheck (with a help icon), Sensors, and Comunicar. Below the navigation bar, a message says 'faz login por favor!'. The main content area features a central login form with the title 'Oi, quem és tu?'. The form includes a horizontal progress bar with five colored segments (green, yellow, orange, purple, blue). Below the progress bar are input fields for 'Username:' and 'Password:'. A blue 'Entrar' button is positioned below the password field. Underneath the button are two links: 'ESQUECESTE A PASS?' and 'REGISTAR'. At the bottom left of the page, a small copyright notice reads '© - Sem direitos nem esqerdos reservados!'.

Figura 3.1 – Página de login

Está será a primeira página que irá aparecer após instalar o sistema. Para que possa utilizá-lo, é necessário que tenha uma conta de utilizador, onde após se autenticar, poderá aceder às páginas referidas anteriormente. É dada a opção de registo caso o utilizador não possua conta.



Novo user

USERNAME:

REQUIRED. 150 CHARACTERS OR FEWER.
LETTERS, DIGITS AND @/./+/-/_ ONLY.

EMAIL ADDRESS:

PALAVRA-PASSE:

CONFIRME PALAVRA-PASSE:

USERNAME TELEGRAM:

Registar

© - Sem direitos nem esquerdos reservados!

Figura 3.2 – Página de registo

Como o próprio nome diz, esta página é responsável por registar um utilizador na base de dados. Ao mesmo é especificado o seu username e palavra-passe como obrigatórios e opcionalmente um campo de endereço de *email* e de *username Telegram*.

Histórico de relatórios

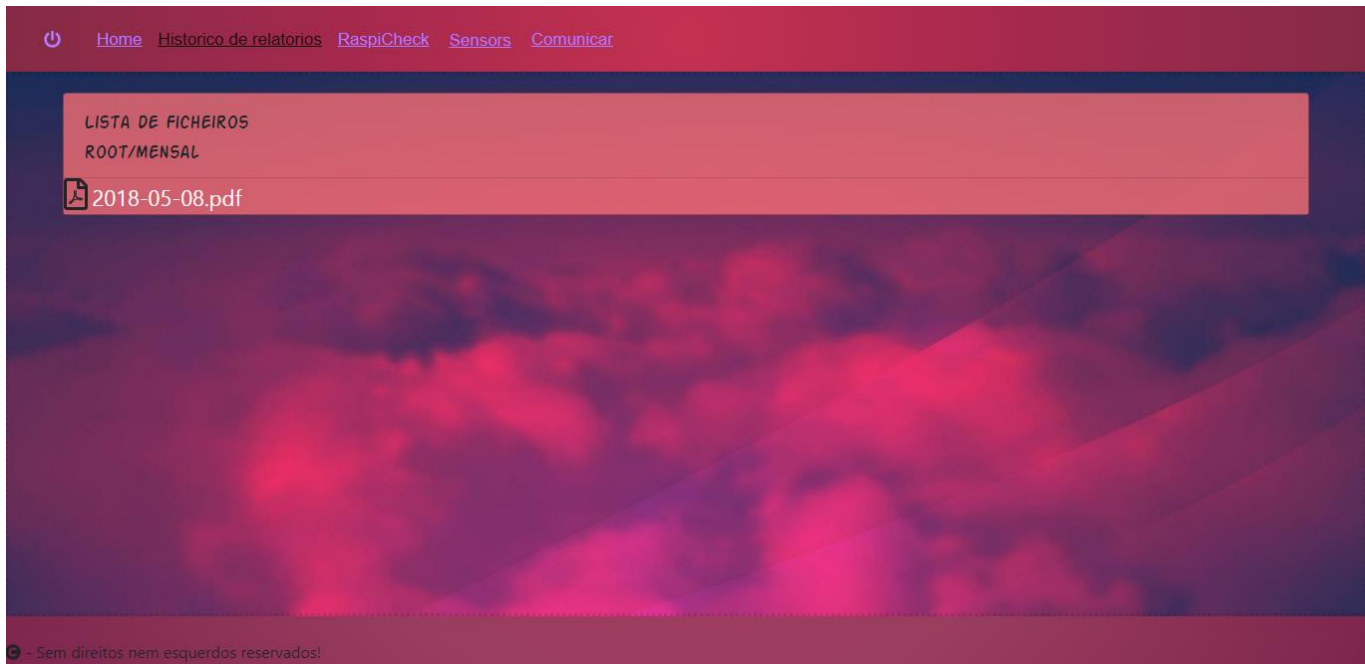


Figura 3.3 – Exemplo da página “Histórico de relatórios”

Ao abrir a página será apresentado ao utilizador uma lista de todas as pastas presentes na base de dados, nomeadamente pastas “mensal” e “semanal”, onde serão guardados os relatórios conforme a forma que são gerados. A página começa na pasta “root” e posteriormente irá avançar conforme a escolha do utilizador, mantendo-o ciente de onde está a todo o momento. Assim que estiver na pasta pretendida, serão listados todos os relatórios guardados nessa pasta, permitindo ao utilizador transferir os relatórios guardados na base de dados ou então eliminar os que estão lá presentes, bastando neste ultimo caso passar o rato sobre o nome do ficheiro que aparecerá um ícone do balde de lixo, onde ao clicá-lo será questionado se tem certeza de continuar a operação.

Independentemente de quem estiver logado no sistema, os relatórios estarão presentes para todos os utilizadores guardados no sistema.

De referir que a procura pelas pastas é feita com base no parâmetro *slug*. Trata-se de uma boa prática de programação em vez de procurar pelos ID’s na base de dados. [5]

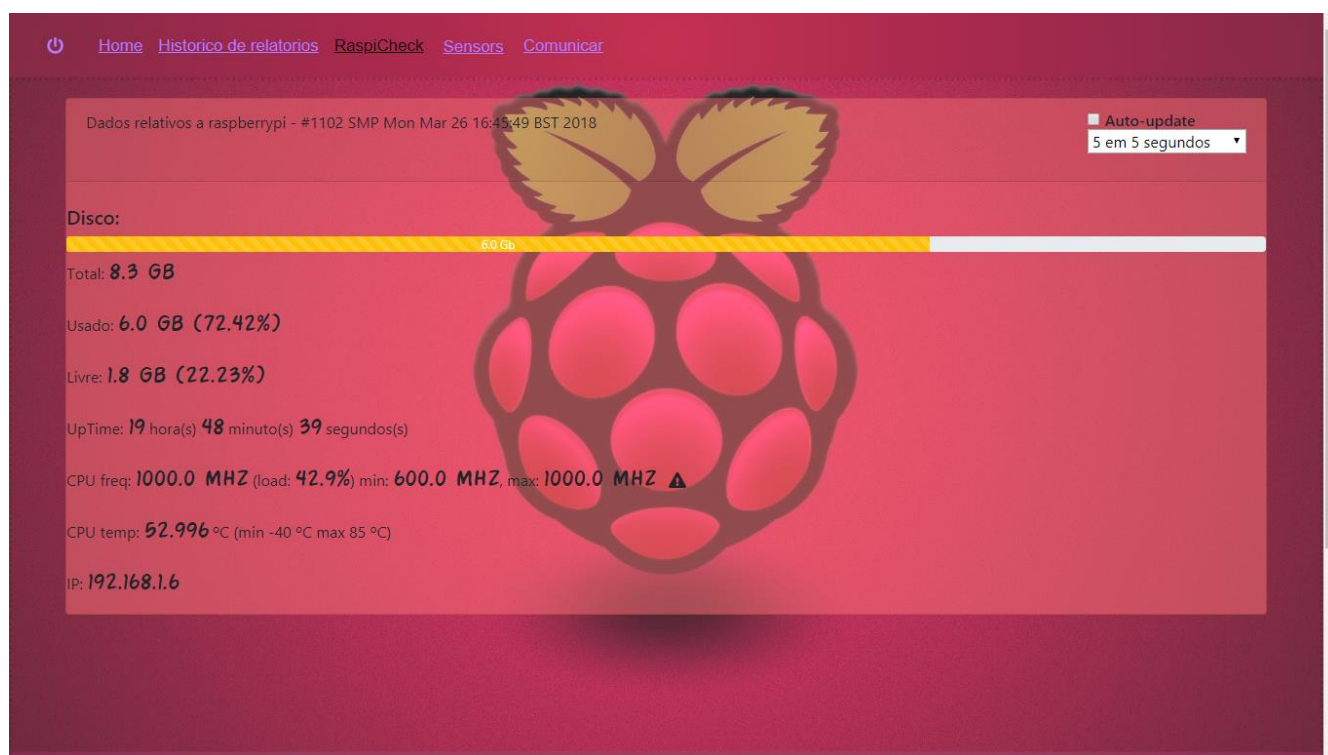
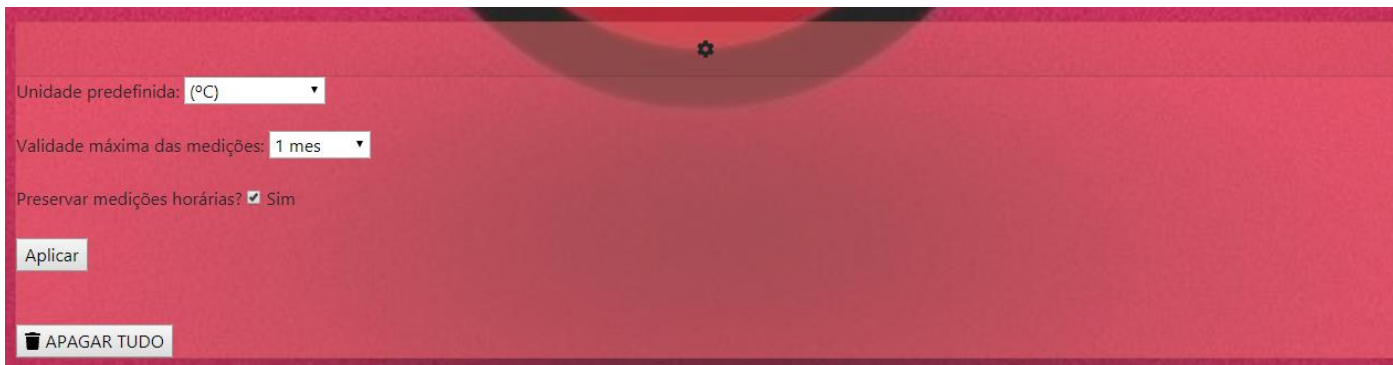


Figura 3.4 – Página “RaspiCheck”

Esta é a única página dinâmica do sistema, apresentando os estados principais de funcionamento do *Raspberry* e possibilitando a opção de atualização automática num período de tempo especificado pelo utilizador sem necessidade de fazer *refresh* à página. Sempre que algum dos aspetos de funcionamento do *Raspberry* tiver problemas, surgirá um ícone de alerta no parâmetro que aponta problemas. Esta página não possibilita alterar nenhum aspeto de funcionamento do sistema, dando somente informações sobre o mesmo.

Trata-se de um conjunto de páginas que disponibilizam gráficos temporais das medições de ambiente. Nestas, o utilizador poderá visualizar medições instantâneas, consultar históricos de medições e modificar definições.



*Figura 3.5 – Página “Temperatura” em
“sensors” - definições*

Abaixo de cada uma das páginas está o menu que permite modificar alguns parâmetros como a unidade predefinida de medição, validade máxima, preservação de medições horárias ou até eliminar todo o histórico.

Para as medições de temperatura existem 3 unidades que podem ser utilizadas:

1. Graus Celsius;
2. Graus Kelvin;
3. Graus Fahrenheit.

Para as medições de humidade atmosférica somente está presente a percentagem (“%”).

Para as medições de pressão atmosférica estão disponíveis 5 unidades:

1. Milibar;
2. Bar;
3. Atmosfera;
4. Hectopascal;
5. Milígrama de Mercúrio.

Para as medições de som somente está disponível a medição por decibel, e para a luminosidade a voltagem do sensor de luz.

O utilizador poderá escolher um prazo máximo de validade de suas medições, podendo ser de 1 dia, 1 semana ou 1 mês. Após passar o prazo, as medições serão eliminadas da base de dados.

Ao preservar medições horárias, o sistema irá manter na base de dados as medições horárias após fazer a média diária. Optou-se por manter esta opção pois por vezes o utilizador não irá necessitar de preservar medições hora a hora, bastando somente a média diária e os respetivos valores de pico. No entanto, ao preservar as medições horárias, a emissão dos relatórios estatísticos conterà um maior volume de dados para analisar, podendo levar a valores estatísticos mais precisos.

Por fim, existe a opção de eliminar todos os dados. Optei por deixar somente a opção de limpar a base de dados de determinados valores de medição em vez de eliminar a partir de uma certa data, pois existe um gerador de relatórios estatísticos onde o utilizador poderá guardar os valores de datas mais antigas sem “entupir” a base de dados.

Gerar relatório

Nesta página o utilizador poderá especificar a frequência que o sistema deverá gerar automaticamente os relatórios, as medições que pretende guardar e também inclui um gerador imediato de relatórios.

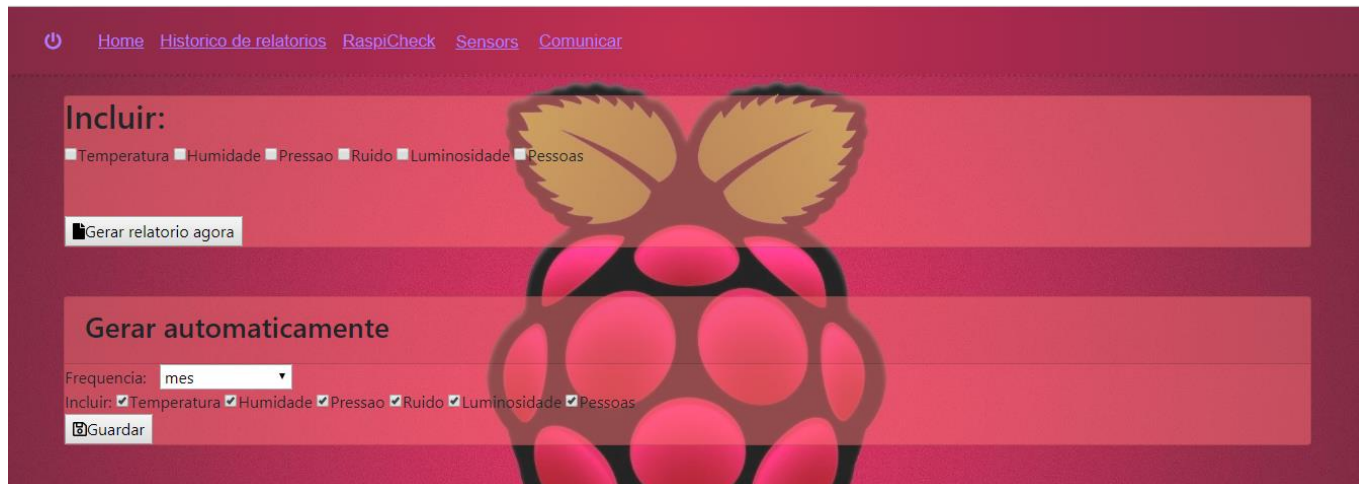


Figura 3.6 – Página “Gerar relatórios”

Implementação

As configurações iniciais começam no ficheiro *settings.py* presente na pasta do projeto. Este reconhece até 10 IPs autorizados a aceder no servidor, podendo ser modificado a qualquer instante. Foi preciso também especificar a base de dados como a *mysql* em vez da *sqlite*.

```
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = False
carlitos = ["127.0.0.1"]
for n in range(11):
    carlitos.append("192.168.1.{}".format(n+1))
ALLOWED_HOSTS = [carlos for carlos in carlitos]

LOGIN_REDIRECT_URL= "/mycloud/raspi_check/"
# Database
# https://docs.djangoproject.com/en/2.0/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': django_DATABASE["NAME"],
        'USER': django_DATABASE["USER"],
        'PASSWORD': django_DATABASE["PASSWORD"],
        'HOST': django_DATABASE["HOST"],
        'PORT': django_DATABASE["PORT"],
    }
}
```

Figura 3.7 – Excerto de código processo servidor – settings.py

Existe somente uma *app* no *Django-Server*: Nuvem.

O *website* não aceita upload de ficheiros.

Todos os métodos da *app* estão disponíveis no ficheiro *views.py*. Cada método presente neste ficheiro é chamado de **view**, havendo views definidas como simples métodos e views definidas através de classes (*class-based view*).

Um exemplo de uma *view* é *Estatisticas_View(request)* (ver **figura 3.8**). O método aceita um objeto do tipo **HttpRequest** que contém *metadata* do pedido ao servidor. Após receber o pedido, o método irá fazer uma *query* procurando pela coluna da tabela Medicao na base de dados responsável por conter informações sobre a geração automática de PDFs. O *Django* simplifica bastante esta parte, bastando para isso importar de *models.py* os modelos definidos (que basicamente representam uma tabela da base de dados) e invocar os respetivos métodos de pesquisa.

Depois de efetuar as respetivas funções a qual o método está destinado, este retornará um objeto do tipo **HttpResponse** que, neste caso, virá do método *render* que recebeu o *request*, o caminho até ao ficheiro *HTML* contendo a template da página e o dicionário de onde o *Django* irá buscar dados para renderizar na mesma.

```

def Estatisticas_View(request):
    template_name = "Nuvem/Sensores/estatisticas.html"
    try:
        data_relatorio = Medicao.objects.get(tipo__contains="relatorio")
    except Medicao.DoesNotExist:
        data_relatorio = Medicao()
        data_relatorio.tipo = "relatorio -THPRLp" #P- pressao, p-pessoas
        data_relatorio.unidade = "semana"
        data_relatorio.validade = datetime.datetime.now() + datetime.timedelta(days=7)
        data_relatorio.save()
    stored = [0,0,0,0,0,0,data_relatorio.unidade]
    auxiliar = ["T","H","P", "R", "L", "P"]
    for n in range(6):
        if auxiliar[n] in data_relatorio.tipo:
            stored[n] = 1

    return render(request, template_name, {"stored":stored, "warning":Health()})

```

Figura 3.8 – Excerto de código processo servidor – views.py – Estatisticas_View

Outro exemplo de *view* é *Sensores Base View(ListView)* (ver **figura 3.9**). Esta *view*, na verdade, serve como superclasse para o conjunto de páginas *Sensors*. A utilização de *class-based views* neste projeto simplificou a quantidade de código necessária para este conjunto de páginas, tornando a implementação mais flexível.

De uma forma resumida, este conjunto de *views* funciona da seguinte forma:

Cada sensor possui uma unidade, uma identificação (se é temperatura, pressão...), um prazo de validade em base de dados e a opção de preservar horas. Sempre que uma destas *views* é chamada, o *Django* irá fazer as pesquisas necessárias para obter o histórico de valores guardados e os parâmetros definidos pelo utilizador no que toca ao armazenamento dos mesmos. Depois irá retornar o *HttpResponse*.

Por se tratar de uma *class-based view*, será necessário substituir alguns campos predefinidos pelo *Django*, nomeadamente o contexto de página. Para tal basta só invocar o método *get_context_data(self, **kwargs)*, onde poderá obter o contexto predefinido e modificá-lo de forma que se adapte ao contexto da página. Neste caso, outros métodos definidos na superclasse serão sobrescritos para que cada página possua seu respectivos dados, sendo estes métodos *get_saved_unit(self)*, *função(self)*, *get_lista_unidades(self)* e *get_queryset(self)*.

```

class Sensores_Base_View(ListView):
    unit = "KPZ"
    what = "PZK"
    validade = "1semana"
    preservar_horas = 0

    #Rapa um contexto novo com coisas q vou adicionar
    def get_context_data(self, **kwargs):
        self.unit, self.validade, self.preservar_horas = self.get_saved_unit()
        atual, SI = LeitorAtual(self.what, self.unit)
        ultima_hora, ultimo_dia, ultima_semana = self.funcao()
        context = super().get_context_data(**kwargs)
        dados = minmedmax_LastDay_or_Week(self.unit, self.what, ultimo_dia[:19])
        dados2 = last_Hour_stats(self.unit, self.what, ultima_hora)
        dados3 = minmedmax_LastDay_or_Week(self.unit, self.what, ultima_semana[:19])
        context['corrente'] = atual
        context['script_graph_last_hour'] = dados2[0]
        context['div_graph_last_hour'] = dados2[1]
        context['script_graph_last_day'] = dados[0]
        context['div_graph_last_day'] = dados[1]
        context['script_graph_last_week'] = dados3[0]
        context['div_graph_last_week'] = dados3[1]
        context['unidades'] = self.get_lista_unidades()
        context['what'] = self.what
        context['unidade_salva'] = " ({} )".format(self.unit)
        context['u_s_escondida'] = self.unit
        context['validade'] = "1 {}".format(self.validade.split("1")[1]) if self.validade !=
"ilimitado" else "Ilimitado"
        context['preservar_horas'] = self.preservar_horas
        context['SI'] = SI
        context['warning'] = Health()
        return context

    #Procura pela unidade guardada na DB e retoma
    def get_saved_unit(self):
        pass #preciso de obj ou 404 pq assim descubro se algo correu mal na bd.. essas tabelas
devem sempre estar presente

    #Joga todos os valores relativos a esse tipo de medicao
    def funcao(self): #Nao meto a retomar obj ou 404 pois se a lista tiver vazia o grafico ja
diz
        pass

    def get_queryset(self):
        pass

    def get_lista_unidades(self):
        pass

```

*Figura 3.9 – Excerto de código processo servidor – views.py –
Sensores_Base_View*

- Processo Leitor

Este processo será responsável por buscar os valores dos sensores e disponibilizá-los para os outros processos. Os valores serão obtidos do *Arduino* e do *SenseHat* instalado no *Raspberry*.

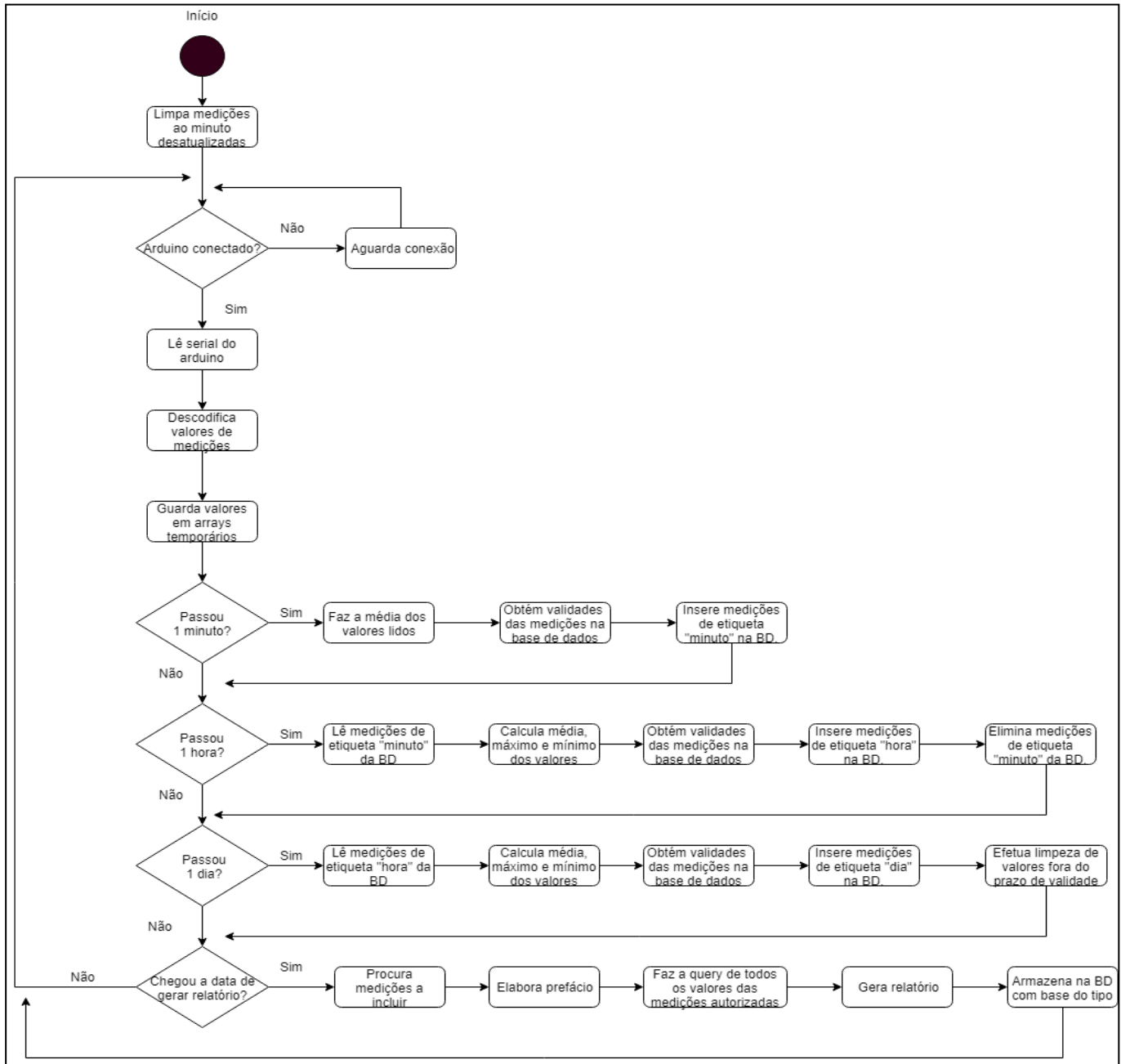


Figura 4 – Fluxograma processo Leitor

Implementação

```
class Arduino():
    def __init__(self):
        self.connectArduino()

    def getArduino(self): # procura pelo endereco do (primeiro) arduino e joga
        for porta in serial.tools.list_ports.comports():
            try:
                return porta.device if ("Arduino" in porta.manufacturer) else True
            except TypeError:
                return "/nao/encontrei/nada" # Pasta tonta so pa la em baixo dar o
exception
```

Figura 4.1 – Excerto de código processo leitor – Class Arduino

O código deste processo está contido no ficheiro “*Leitor.py*” na pasta do projeto. Nele estão importadas, dentre outras bibliotecas:

- *MySQLDB* (v.1.3.12);
- *PySerial* (v.3.2.1);
- *Sense_hat* (v.2.2.0);

Existem duas classes principais:

- *Arduino* – Classe personalizada que conterà todos os métodos de leitura do serial do *Arduino*;
- *SenseHat* – Importada da biblioteca *sense_hat* com todos os métodos de interação com o *SenseHat*.

Este ficheiro também importa funções do ficheiro *utils.py*, sendo estas:

- *LeitorAtual* – Função de reserva utilizada pela classe *Arduino*. Nada mais faz que buscar o último valor de uma respetiva medição que está guardado num ficheiro de texto. Este método somente é chamado sempre que ocorre uma falha ao obter uma determinada leitura do *Arduino* ou do *SenseHat*.
- *getDiskSpaces* – Função que buscará a percentagem de disco utilizada pelo *Raspberry*, auxiliando no “travar” do gerador de PDFs caso não haja muito espaço de armazenamento disponível.
- *PreparaPDF* – método que irá preparar o gerador de PDFs.

Ao arrancar, o processo tentará uma conexão com a base de dados. Se conectado, irá efetuar uma limpeza, eliminando medições de minuto desatualizadas (se forem de minutos senão da hora que o *Raspberry* está no momento ligado) e eliminando medições fora do prazo de validade especificado pelo utilizador. Findas estas etapas, a leitura de valores é iniciada, podendo ser terminada usando o sinal SIGINT (**CTRL + C**).

O método **LoopLeitor(db, sensor1, sensor2)** será o principal neste processo. Este receberá como parâmetros a conexão com a base de dados (num objeto do tipo `_mysql`), o **sensor1** que representa o *Arduino* e o **sensor2** o *SenseHat*. A função efetuará um *loop* (quase)infinito onde irá buscar os valores medidos **de 10 em 10 segundos**, armazenando-os em contentores de dados temporários e depois guardados na base de dados.

```
if __name__ == '__main__':
    print("Conectando a base de dados")
    db = _mysql.connect("localhost", "Raspi", "OlhoRaspi3", "CloudV1")
    sys.stdout.write("Limpendo medicoes de minuto desatualizadas")
    sys.stdout.flush()
    for n in range(5):
        sys.stdout.write(".")
        sleep(0.3)
        sys.stdout.flush()
    limpador(db, "minuto") #some com tudo do ultima hora q nao seja da hora q isto
    arranca
    limpador(db, "hora")
    print("conectado e lixo inicial limpo com sucesso!\n\n")
    Sensor1 = Arduino()
    Sensor2 = SenseHat()

    print("Iniciando a leitura...")
    try:
        LoopLeitor(db, Sensor1, Sensor2)
    except KeyboardInterrupt:
        print("Desconectando da base de dados...")
```

Figura 4.2 – Excerto de código processo leitor – Main

Uma das tarefas mais importantes é o cálculo das médias, mínimos e máximos. Estes valores são calculados pelo método **Faz_a_Media(what, db)**, método este que necessita de saber que tipo de dados estão sendo tratados e do objeto da classe `_mysql`.

Sabemos que este processo efetua leituras, guardando-as na base de dados minuto a minuto. Para guardar os valores de minuto, o sistema somente precisa de obter os valores guardados em memória e efetuar o cálculo da média para todos os valores medidos, guardando o valor médio na base de dados e apagando-os posteriormente da memória. Já para guardar os valores de hora, o sistema recorrerá aos métodos `mysql MAX(valor)`, `MIN(valor)` e `AVG(valor)`, bastando somente elaborar a *query* e efetuar a limpeza dos valores de tag minuto da base de dados. De semelhante modo trabalha para guardar os valores de dia, mas somente apaga os valores de hora de uma determinada medição se o utilizador não selecionou nas definições a opção “*Preservar medições horárias*”.

De referir que se a opção “*Preservar medições horárias*” não estiver selecionada, o sistema eliminará todas as medições horárias da respetiva medição que não sejam do presente dia que o sistema estiver ligado.

- Processo BotTelegrama

Este processo tem como função permitir a interação do utilizador com o sistema de uma forma totalmente remota utilizando como recurso a aplicação **Telegram**. O utilizador tanto pode enviar a mensagem escrito como por voz, sendo que no último ele só entende a língua inglesa. Para usufruir do *bot*, o utilizador terá de guardar nos seus dados de registo o *nickname* que utiliza no *Telegram*. Caso contrário, o *bot* não reconhecerá o utilizador e não responderá a pedidos.

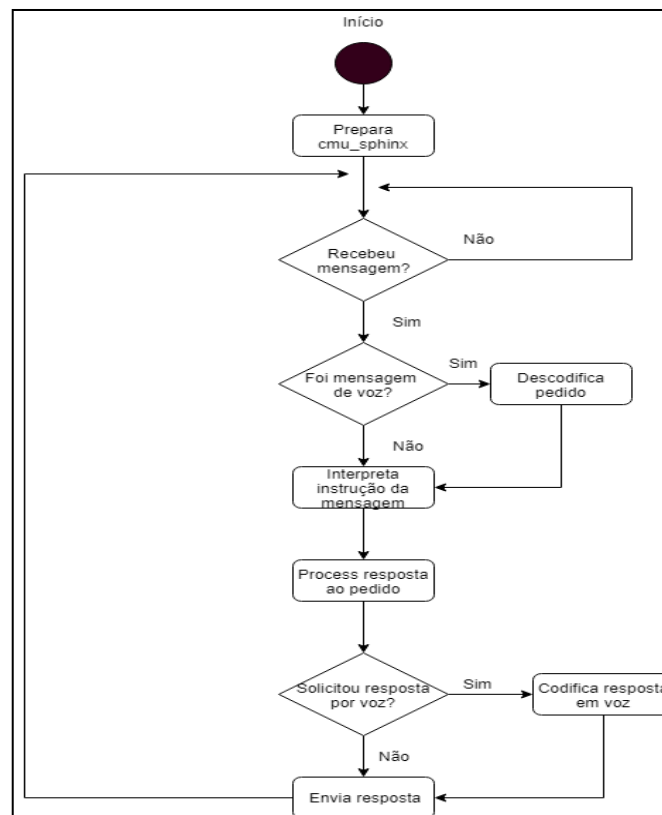


Figura 5 – Fluxograma processo BotTelegrama

Implementação

O código deste processo está contido no ficheiro “*BotTelegrama.py*” na pasta do projeto. Nele estão importadas, dentre outras bibliotecas:

- *Mysqldb* (v.1.3.12);
- *telepot* (v.12.6);
- *sense_hat* (v2.2.0);
- *Urllib*.

Este também importa todo o conteúdo do ficheiro *VoiceRecognition_NOTGOOGLE* que contém todos os métodos para interagir com a *CMUSphinx*. Esta biblioteca foi desenvolvida pela [Carnegie Mellon University](http://cmusphinx.sourceforge.net/) e é *OpenSource*. Escolhi-a precisamente por ser grátis, já que outras mais desenvolvidas possuem custos.

O processo inicia criando um objeto do tipo *telepot.Bot*, depois conecta-se à base de dados e prepara o *TTS cmu_sphinx* para receber voz. Feito isto, este irá procurar na base de dados por todos os utilizadores registrados que possuam conta no *Telegram* e irá guardar os seus respetivos *nicknames*.

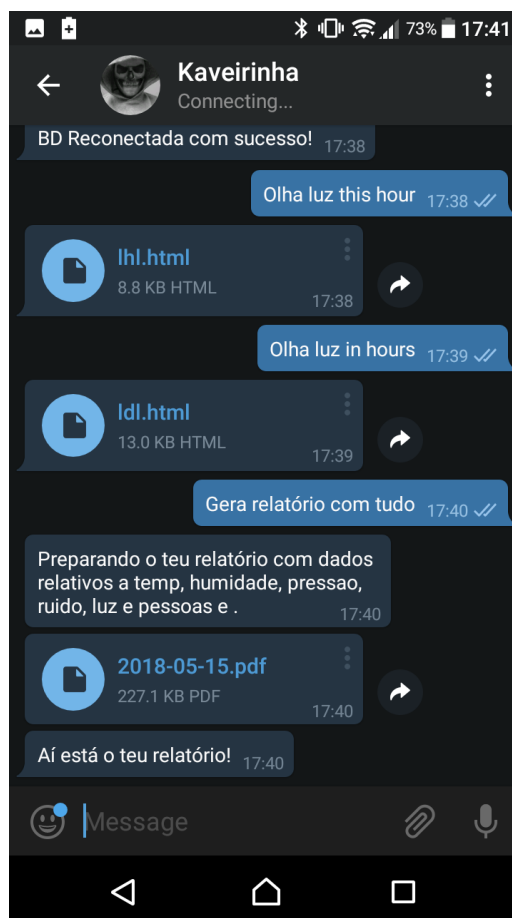
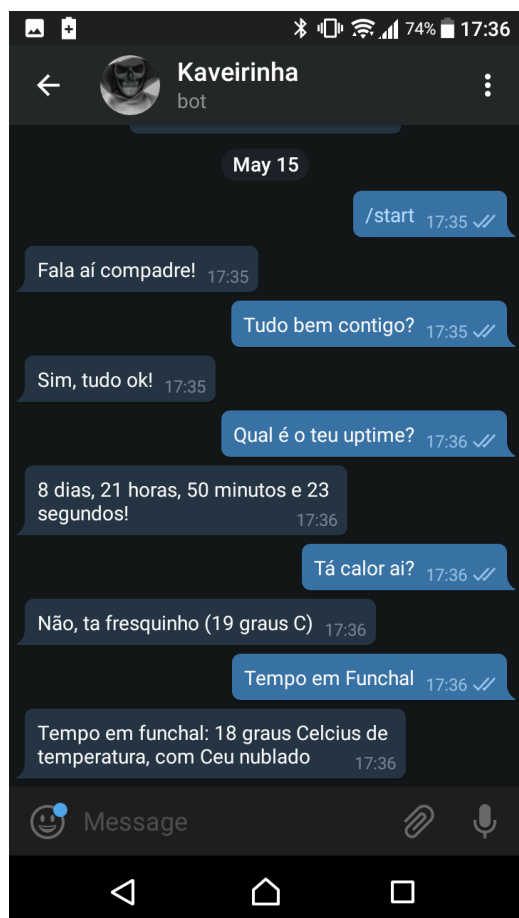
```
if __name__ == '__main__':
    db = _mysql.connect("localhost", "Raspi", "OlhoRaspi3", "CloudV1")
    bot = telepot.Bot("463315222:AAFbFDtcgYxM7hf04vWUxvwiqrK0wRNPsvA")
    mestre = []
    db.query("SELECT * FROM Nuvem_bt")
    pesquisa = db.store_result() #guarda o resultado da query
    resultado = pesquisa.fetch_row(how=1, maxrows=0)
    lenresultado = len(resultado)
    for n in range(lenresultado):
        mestre.append(resultado[n]['uname'].decode('utf-8'))

    print("Bot preparado...")
    sleep(3)
    os.system('clear')
    print("Operando...")
    bot.message_loop(Que_Faz_O_Bot)
    while True:
        pass
```

Figura 5.1 – Excerto de código processo BotTelegrama – Main

O processo necessita de estar em *loop* infinito, pois o método ***message_loop*** trabalha com tarefas, sendo necessária uma tarefa principal sempre em execução. Este mesmo método recebe como parâmetro o apontador de outro método criado pelo utilizador, e sempre que uma mensagem for enviada para o *bot*, esse método será chamado.

Atualmente o *bot* consegue entender **40 instruções diferentes**, respondendo tanto por texto como por voz ou através de ficheiros.



Figuras 5.2 e 5.3 – Exemplos de funcionamento do bot

O *bot* somente responde aos utilizadores registrados, existindo uma lista de comandos que é enviada pelo mesmo sempre que lhe envia a mensagem “ajuda-me” ou “help”, seja escrito ou seja por voz. Dentre as várias funções o *bot* permite:

- Buscar dados relativos ao funcionamento do sistema (espaço de disco, processador, etc...);
- Buscar medições do ambiente;
- Gerar relatórios e obtê-los instantaneamente;
- (BETA) Saber a meteorologia de uma localização específica;
- Mandar piadas;
- Comunicar através do sistema utilizando voz (TTS) ou através da matriz de LEDs do *SenseHat*.
- Etc...

- Processo Arduino

Trata-se de um processo que está em execução no *Arduino*. Este permite ao *Arduino* captar os valores de luminosidade, temperatura e ruído dos sensores do circuito, fornecendo os valores obtidos em tempo real tanto para o *Raspberry* utilizando o seu serial como para o utilizador através de 3 displays de 7 segmentos. Além disso o *Arduino* também mostrará valores de data/hora e fornecerá uma luz piloto que pode ser ativada manualmente ou automaticamente após a definição de preferências do utilizador.

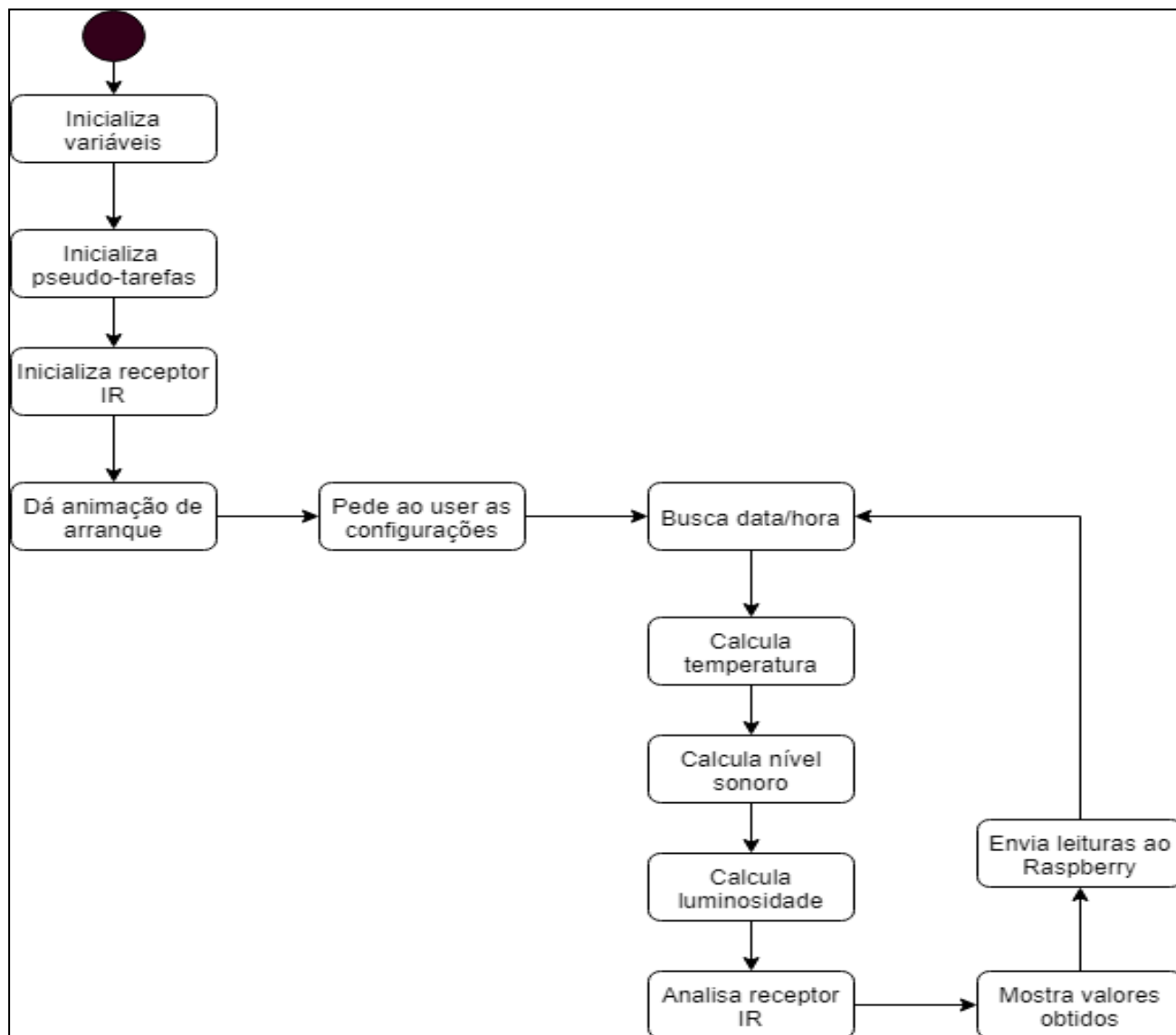


Figura 6 – Fluxograma simplificado – processo Arduino

Implementação

Ao arrancar, o processo inicializa todas as variáveis globais e prepara o recetor *IR* junto com as pseudo-tarefas. Foi optado por utilizar um recetor *IR* em vez de botões físicos presentes no circuito, assim torna-se mais rápida e flexível a interação entre o utilizador e o dispositivo. O comando utilizado foi um comando antigo de *DVD*, entretanto qualquer comando remoto funciona desde que sejam conhecidos todos os códigos enviados pelo comando ao *Arduino* a fim de garantir a comunicação.



Figura 6.1 – Exemplo de controle remoto utilizado

Após tudo estar inicializado, o sistema solicitará ao utilizador a configurações básicas, que são:

1. Dia do ano (d);
2. Mês do ano (d mais ponto);
3. Ano (A);
4. Hora (h);
5. Minuto (h mais ponto);
6. Luz-piloto automática (U) - 11 – ativar, 00- desativar;
7. Hora que ativa a luz-piloto automática (r);
8. Hora que desativa a luz-piloto automática (r mais ponto).

Para ajustar, utilize os botões direcionais do comando. O primeiro *display* de sete-segmentos da esquerda para a direita simboliza o dado a mostrar, e os outros simbolizam o dado. Por não existir muitos displays disponíveis, os valores tiveram que ser divididos de forma que sejam mostrados sequencialmente, por exemplo mostrando primeiro as horas e depois os minutos. Depois do ajuste, só premir no comando o botão *Setup* (ou outro botão se optar por “copiar” este projeto).

Uma particularidade deste processo é a existência de pseudo-tarefas, cuja *API* poderá ser obtida aqui: <https://github.com/ivanseidel/ArduinoThread>. Por se tratar de um dispositivo *single-core*, a implementação de tarefas paralelas reais é impossível, entretanto com esta *API* desenvolvida por *Ivan Seidel* poderemos utilizar as pseudo-tarefas que dão a ilusão de que várias tarefas estão em execução concorrente.

O preparo das pseudo-tarefas é relativamente simples, bastando somente criar objetos do tipo “*Thread*”, associar a estes os respectivos métodos que irão trabalhar, definir o intervalo de execução de cada tarefa e agrupá-las em grupos de tarefas (ver figura 6.2). A definição dos métodos é feita de forma normal.

```
void setup() {
    // put your setup code here, to run once:
    temperatura = 0;
    luminosidade = 0;
    Serial.begin(9600);
    tarefa_data_e_hora.onRun(metodo_tarefa_data_e_hora); //Defino para cada tarefa o metodo
correspondente
    tarefa_data_e_hora.setInterval(1); //e dou um intervalo dentre suas execuções
    tarefa_temperatura.onRun(metodo_tarefa_temperatura);
    tarefa_temperatura.setInterval(1);
    tarefa_luz.onRun(metodo_tarefa_luz);
    tarefa_luz.setInterval(1);
    tarefa_receptor_ir.onRun(metodo_tarefa_receptor_ir);
    tarefa_receptor_ir.setInterval(1);
    tarefa_display.onRun(metodo_tarefa_display);
    tarefa_display.setInterval(1500); //1.5 segundos
    //tarefa_pessoas.onRun(metodo_tarefa_pessoas);
    //tarefa_pessoas.setInterval(1);

    controll.add(&tarefa_data_e_hora);
    groupOfThreads.add(&tarefa_temperatura);
    groupOfThreads.add(&tarefa_luz);
    groupOfThreads.add(&tarefa_receptor_ir);
    groupOfThreads.add(&tarefa_display);
    //groupOfThreads.add(&tarefa_pessoas);
    controll.add(&groupOfThreads);

    irrecv.enableIRIn(); // Start the receiver
    lampada0.Interruptor(0); //Lampadas e displays off
    lampada1.Interruptor(0);
    Luz_Estado.customCor(0,0,nightMode);
    Luz_Modo.customCor(0,0,nightMode);
    DisplayModos.Escreve(0);
    BigDigito.Escreve(0);
    SmallDigito.Escreve(0);
    setTime(13, 14, 9, 30, 12, 2017); //setTime(HOUR, MINUTE, SECOND, DAY, MONTH, YEAR);
    arrancagem();
}
```

Figura 6.2 – Excerto de código processo Arduino – setup

Outra particularidade é a presença da luz piloto. Esta luz pode ser invocada manualmente premindo o botão *Power* do comando ou então através da configuração de ligação automática, sendo neste último a luz ativada durante 30 segundos até apagar. O processo ativa a lâmpada piloto no modo automático sempre que deteta uma diminuição brusca dos níveis de luminosidade (e quando a opção luz piloto automática está ativada).

Por fim também está disponível a opção *media center* quando prime o botão *Open/Close*. O nome da opção é enganoso pois esta utilidade tanto pode servir para media center como para outros propósitos. Mantive esta escolha pensando na flexibilidade, assim se possivelmente for desenvolvido algum programa que permita a interação utilizando um controle remoto, o *Arduino* irá dar como output o valor do botão premido, bastando depois o utilizador decidir o que fazer com esse valor.

Problemas e limitações

Nesta secção serão apresentados os principais problemas e limitações encontrados ao longo do projeto, estando organizadas com base nos processos onde foram encontrados.

Processo servidor

Inicialmente tinha previsto o *website* com um método em *javascript* que captasse a voz do utilizador e a decodificasse em determinadas instruções, utilizando a *Google Voice API*, mas por não possuir conhecimentos na linguagem e por não possuir tempo o suficiente para me dedicar à mesma, optei por retirar essa função deste processo.

A geração dos gráficos utilizando a biblioteca “*Bokeh v.0.12.14*” mostrou-se instável ao tentar redimensionar os gráficos utilizando o *HTML*. Isso porque os gráficos perdiam qualidade ao modificar algum parâmetro *CSS* de alguma divisão da página contendo os mesmos, perdendo valores, mostrando gráficos cortados ou com dimensões exageradamente grandes.

Há ausência de mobilidade, pois para que o mesmo seja acedido é necessário estar conectado na mesma rede que o dispositivo. A página *web* utiliza o protocolo **HTTP**, logo os dados não são encriptados, permitindo que um dispositivo na rede intercete seus dados e consiga facilmente acede-los.

Não existe reposição de tabelas, sendo esta tarefa da responsabilidade do utilizador do sistema.

Outra limitação é a ausência de domínio *DNS*, logo o acesso é feito **escrevendo como URL um endereço IP**, do género <http://192.168.1.4:8000/mycloud/home/>.

Processo Leitor

As medições do número de pessoas num determinado espaço necessitam de tratamento diferenciado face as outras medições, devido se tratar de uma medição atómica. Não podemos representar 1.5 pessoas por razões óbvias, mas assim se pretender saber o número de pessoas numa sala ao longo do dia (valores médios), os valores devem ser arredondados por excesso. Mesmo assim a precisão não tende a ser ótima.

Foi preciso muito cuidado na elaboração das queries. Inicialmente havia implementado queries dinâmicas utilizando *JOIN's*, entretanto isto provou ser um golpe na performance do *Raspberry*. Por se tratar de um dispositivo sem muito poder de processamento, as queries avançadas demoravam muito a concluir, e nos casos em que a base de dados já continha uma grande quantidade de dados, simplesmente travava. Para evitar posteriores problemas, as queries tiveram que ser divididas em pesquisas mais pequenas, o que aumentou o número de linhas de código na criação deste programa.

Processo BotTelegrama

Ao ficar muito tempo sem ser solicitado pelo utilizador, a conexão à base de dados perde-se. Isso acontece por causa da variável `wait_timeout` em `my.conf`, que está no valor padrão (8 horas). Por isso, a cada 8 horas de inatividade o comando “*reconnect*” deverá ser chamado através do *bot* para que a conexão seja reestabelecida.

Havia implementado junto um comando que permitiria a um utilizador administrador iniciar/eliminar processos do sistema, neste caso os processos principais do projeto. Entretanto por falta de disponibilidade essas funções foram removidas, era necessário definir corretamente todos os mecanismos de segurança para que estas fossem disponibilizadas.

Processo Arduino

Existe um problema de precisão nas horas nesta implementação. O *Arduino* não avança de forma precisa a contagem dos segundos, havendo pequenos atrasos que ao se acumularem fazem com que o *Arduino* se atrase 1 minuto a cada 2 dias e meio. Após alguma pesquisa, descobri que esse atraso se deve ao oscilador de cristal presente no *Arduino* que não é de melhor qualidade. Para que este problema seja solucionado a utilização de um módulo RTC (*Real Time Clock*) é mandatária. Neste projeto não foi utilizado esse módulo. [6]

De todas as medições, a captura de som é a menos precisa por ser muito difícil efetuar a medição do nível sonoro sem ser através da regressão linear matemática.

A escala de luz também teve seus problemas, onde ao efetuar os cálculos de luminosidade os valores não correspondiam aos da realidade, havendo diferenças significativas dos níveis. Por conta disso, somente será mandado o valor bruto da leitura do sensor de luz.

Testes

A fim de verificar se os dados correspondem à realidade, diversos testes foram efetuados em diferentes espaços. A pressão atmosférica, nível de ruído e a humidade do ar foram as únicas medições que não tiveram meio de comparação, pois não consegui obter equipamento próprio para essas medições. Inicialmente havia proposto comparar as medições citadas com as que eram medidas no **Observatório meteorológico do Funchal**, entretanto não considerei avançar com esta proposta visto que as localizações eram diferentes de onde o observatório se encontra.

Caso 1: Quarto do *developer*

Nesta primeira situação o sistema ficou durante **1 mês** em sessão de testes contínuos. Neste ambiente, foi possível efetuar comparações das pessoas que entravam no quarto, de temperatura (usando um termómetro digital) e de humidade usando um relógio com medidor de humidade. O número de medições manuais foi estipulado em **3 por dia**. Tanto o sistema como o relógio estão situados no mesmo local.

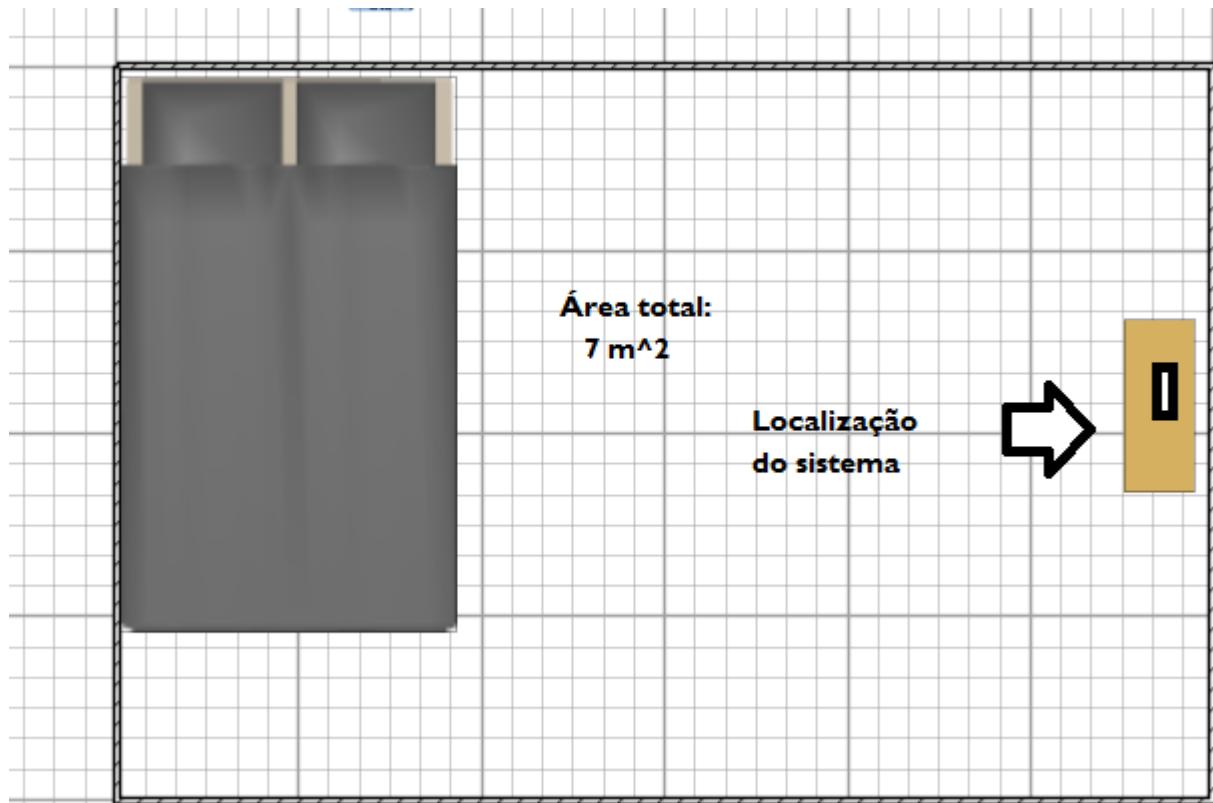


Figura 7 – Exemplo simplificado da localização do sistema em caso 1

Segundo o gráfico da **figura 7.1**, ambos os equipamentos registavam os mesmos valores de temperatura na maior parte das medições.

Entretanto verificou-se que em **13.3%** das medições os valores diferenciavam em 1 grau um do outro, o que indica um **grau de precisão elevado** neste caso de teste.

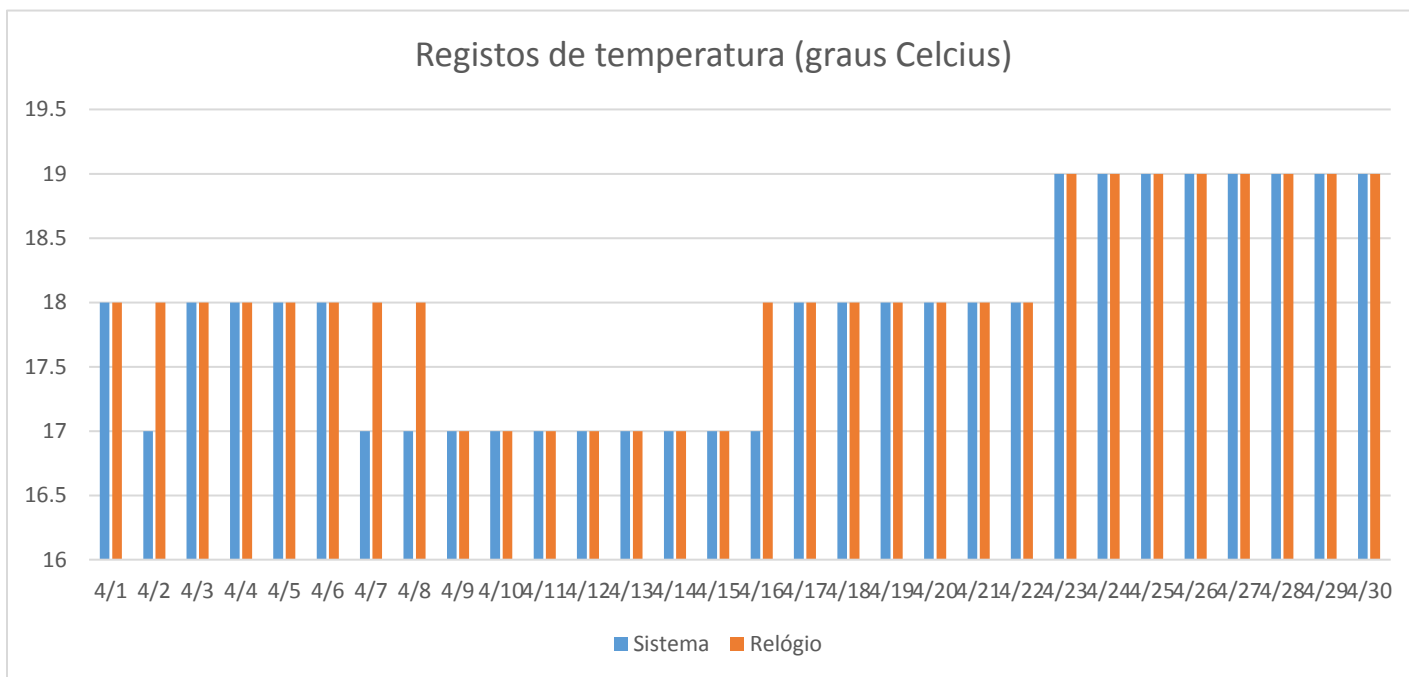


Figura 7.1 – Registo das medições de temperatura e respetivas comparações.

Segundo o gráfico da **figura 7.2**, os equipamentos registravam valores distintos de humidade. Em todos os valores podemos verificar que o equipamento adicional de medição registava sempre valores acima daqueles registados pelo sistema do *Raspberry*, havendo diferenças médias de 3% a 4% entre os valores de ambos, e em alguns casos **9%** (em referência à unidade utilizada na medição e não a variação percentual dos valores). Podemos concluir com estas observações que o sistema tem uma **precisão razoável**.

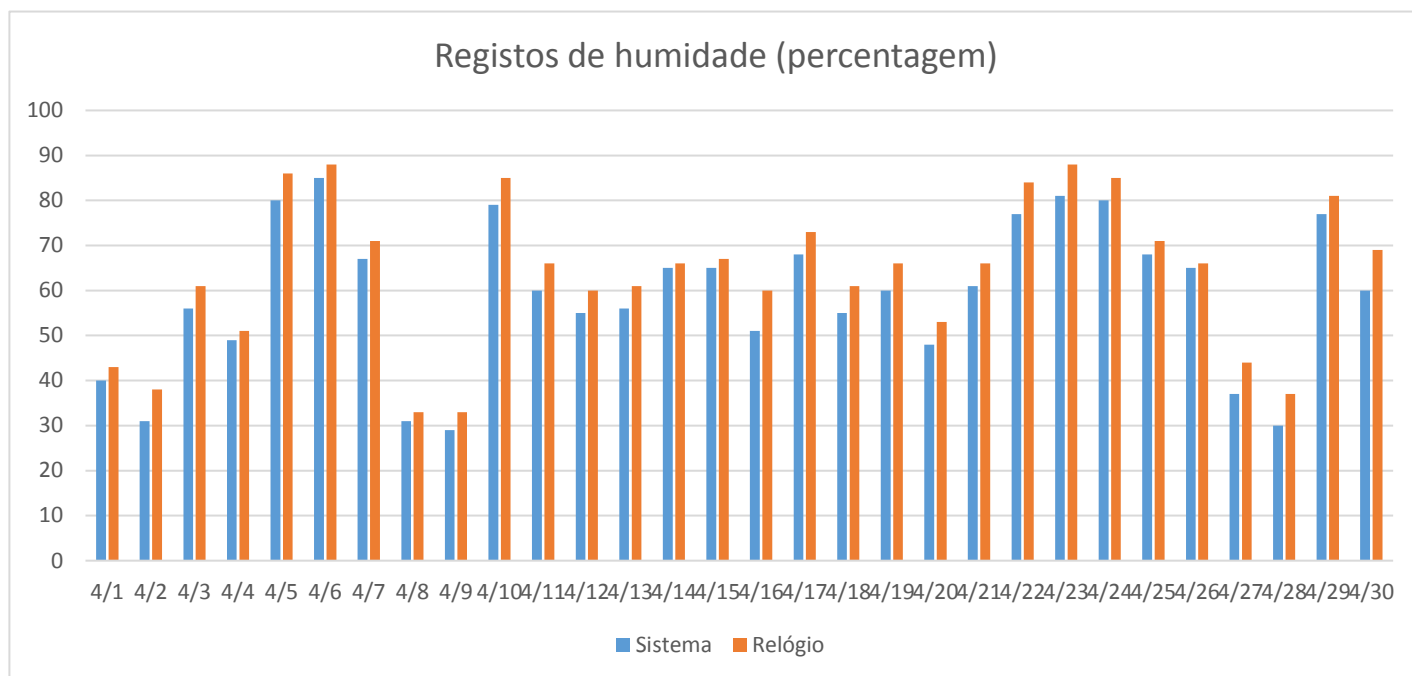


Figura 7.2 – Registo das medições de humidade e respetivas comparações.

Caso 2: Casa do vizinho do *developer*

Nesta situação o sistema ficou durante **1 semana** em sessão de testes contínuos, e foram utilizados como meios de comparação os mesmos equipamentos do caso 1. Para este caso foi solicitado ao titular do imóvel que tirasse nota das medições dos equipamentos de comparação e que registrasse a data e o horário das medições. O número de medições manuais foi estipulado em **3 por dia**. O titular do imóvel posicionou o sistema no seu quarto da mesma forma que a **figura 7.1**, inclusive o espaço e as dimensões são as mesmas que no caso anterior.

Segundo o gráfico da **figura 7.3**, as variações das medições são as mesmas que no **caso 1**, levando à mesma conclusão que nesse mesmo caso.

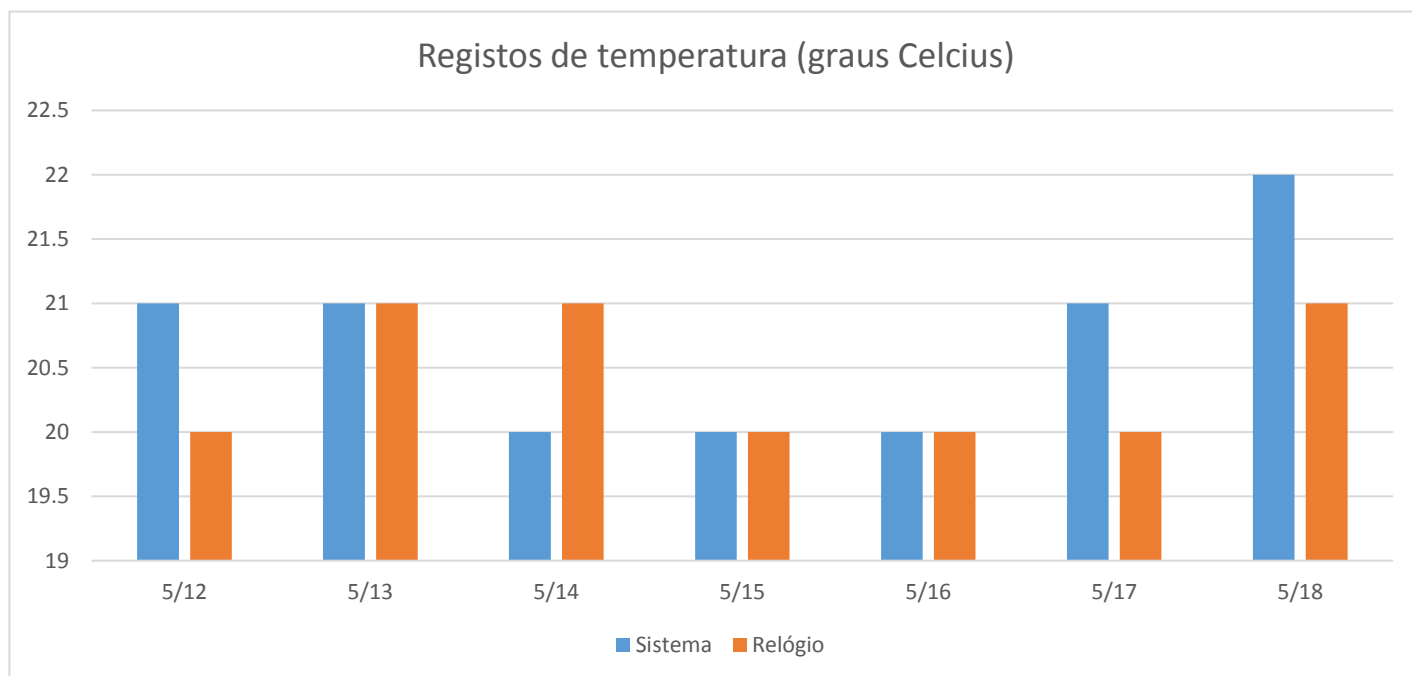


Figura 7.3 – Registo das medições de temperatura e respetivas comparações.

Em relação à humidade as mesmas variações foram detetadas.

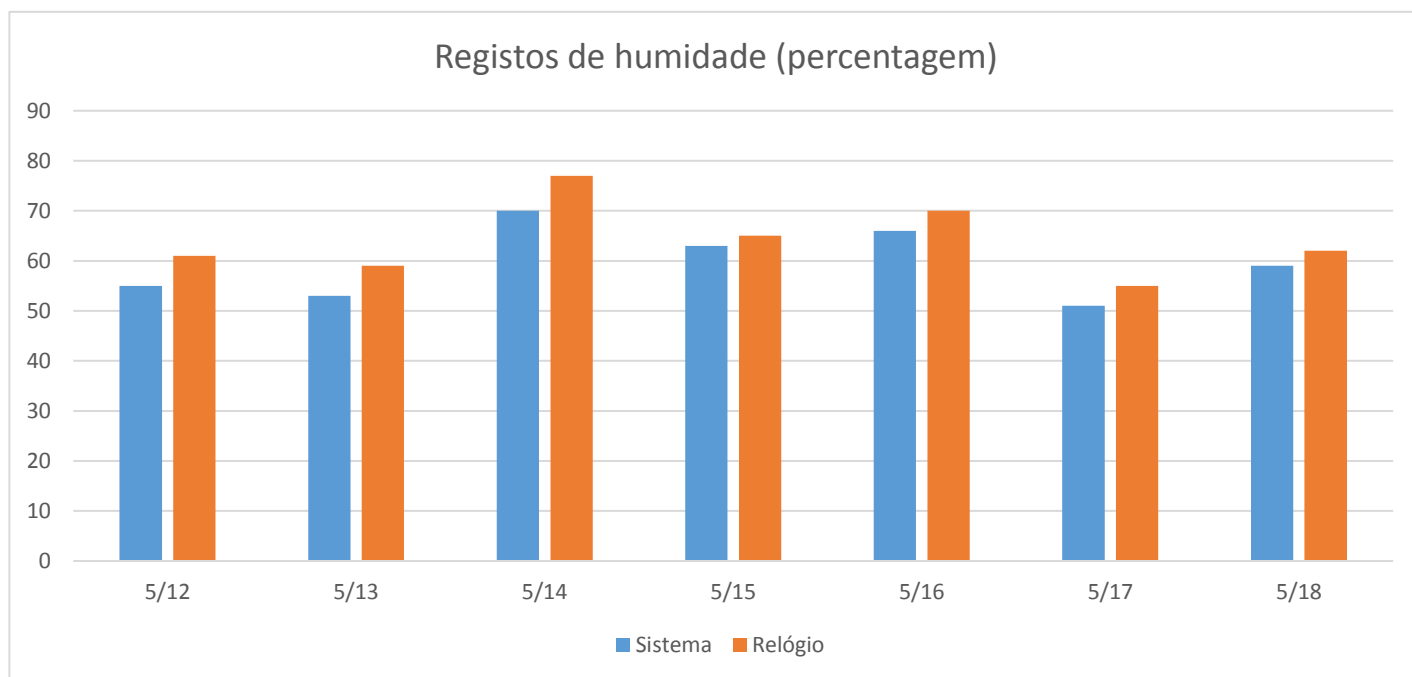


Figura 7.4 – Registo das medições de humidade e respetivas comparações.

Caso 3: IURD

Por fim o sistema ficou durante **2 dias** em sessão de testes contínuos neste último caso. Foi cedida por parte da entidade principal do estabelecimento a utilização do sistema no salão de reuniões, e usaram-se os mesmos equipamentos de teste nos casos 1 e 2, juntamente com um protocolo de **3 medições manuais por dia**.

No que diz respeito às medições de temperatura e humidade podemos concluir, como nos **casos 1 e 2**, que o sistema possui **bons graus de precisão**. Não estão presentes gráficos nesta secção por conta dos dados serem poucos e liderarem a mesma conclusão que nos casos anteriores.

Infelizmente o teste de medição de pessoas **mostrou-se muito instável** e registou valores exacerbados, tendo muitas das vezes demonstrado imprecisão nos horários de maior fluxo de pessoas.

Conclusões finais

O sistema **demonstrou níveis satisfatórios de precisão** em todos os casos nas medições registadas. De referir que as restantes medições onde não foi possível efetuar a comparação devem ser posteriormente avaliadas de forma a verificar se correspondem de igual forma às medições reais.

Conclusão

Com o avanço do IoT, objetos do nosso dia-a-dia, vários deles presentes desde gerações anteriores, estão cada vez mais automatizados e capazes de interagir de uma forma tecnologicamente avançada com os utilizadores humanos.

Através da elaboração deste projeto consegui aprofundar e consolidar conhecimentos de programação na linguagem *Python*. Todos estes conhecimentos ser-me-ão valiosos para um possível futuro como profissional na área de informática e computação, com o cumprimento de todos os objetivos e especificações propostos pela docente orientadora.

Foram sentidas algumas dificuldades durante a elaboração deste projeto tendo em conta outras unidades curriculares, pois o tempo de conciliação era pouco e por conta deste ser meu último semestre como estudante universitário, a sensação de pressão era constante. Mesmo assim, consegui ultrapassar certos limites pessoais e como não tinha outra hipótese, decidi dar o meu melhor. Este projeto não foi feito para fins profissionais.

Desde muito jovem que tive fascínio por tecnologia. Sempre tive curiosidade em saber como é que determinadas ferramentas tecnológicas funcionam, e além disso, queria saber como poder implementar esses conhecimentos em algo realista. Anos na universidade me proporcionaram conhecer um pouco sobre esse mundo tecnológico, e vi neste projeto uma oportunidade de por as mãos à obra em algo que posso dizer ser *meu*, e algo que posso utilizar para o meu dia-a-dia.

De uma forma pessoal, aprecio de uma forma positiva a elaboração deste projeto, e com este o interesse pela Internet das Coisas foi aumentando, e como referido anteriormente, a experiência adquirida ditará um futuro mais preenchido como Engenheiro Informático, sem problemas ou restrições em englobar-me na área de IoT.

Referências

- [1] K. Baras e L. L. Brito, "Introduction to the Internet of Things," p. 3.
- [2] Logitia Solution pvt Ltd, Web and Application Development Company, "What is internet of things and its uses?," Quora, 29 4 2017. [Online]. Available: <https://www.quora.com/What-is-internet-of-things-and-its-uses>. [Acedido em 22 5 2018].
- [3] C. McClelland, "Internet of Things Examples and Applications," Iot for all, 8 12 2016. [Online]. Available: <https://www.iotforall.com/internet-of-things-examples-applications/>. [Acedido em 22 5 2018].
- [4] Carnegie Mellon University, "CMUSphinx Open Source Speech Recognition Toolkit," [Online]. Available: <https://cmusphinx.github.io/>.
- [5] J. Mitchel, "A Unique Slug Generator for Django," Coding for Entrepreneurs, 3 2 2017. [Online]. Available: <https://www.codingforentrepreneurs.com/blog/a-unique-slug-generator-for-django/>. [Acedido em 22 5 2018].
- [6] Arduino, "Relógio "Perfeito" - Ajuda," 11 6 2011. [Online]. Available: <https://forum.arduino.cc/index.php?topic=65209.0>. [Acedido em 6 6 2018].