Casa Inteligente

Eduardo Gonçalves

Jorge Grabovschi

Trabalho de Projeto da unidade curricular de Tecnologias de Internet

Leiria, junho de 2023

Lista de Figuras

Lista de Tabelas

Lista de siglas e acrónimos

API	*Application Programming Interface*
ESTG	Escola Superior de Tecnologia e Gestão
IoT	*Internet of Things*
IPLeiria	Instituto Politécnico de Leiria
SBC	*Single Board Computer*

Índice

1. Introdução

Numa altura de elevado crescimento a nível tecnológico tem-se tentado usá-la a nosso favor para poupar tempo e/ou recursos. Por isso, surge o conceito de Casa Inteligente, uma casa ideal que se aproveita destas inovações.

O objetivo deste trabalho é criar uma solução IoT para uma Casa Inteligente, utilizando o conhecimento que nos foi lecionado na unidade curricular de Tecnologias de Internet. Com esta solução pretendemos monitorizar certas propriedades físicas e despoletar ações que alteram o meio físico da casa, como ligar uma ventoinha, ou até mesmo abrir uma porta, através de eventos. Todas estas ações poderão ser vistas através de uma plataforma (sítio web) dinâmica e usável por uma pessoa comum, que se encontra sempre atualizada e acessível *Anytime & Anywhere*, havendo sempre uma comunicação entre os vários dispositivos e coisas (sensores e atuadores) através de uma API. Esta plataforma tem diferentes níveis de privilégios e possibilidade de registo. Com isto também temos o objetivo de explorar as diferentes tecnologias por detrás deste projeto.

Para esta solução foram usados fundamentos básicos de desenvolvimento web (HTML, CSS, PHP e JavaScript) para criar o sítio web e a API. Foram utilizados dispositivos com capacidade de comunicação *WiFi* e de monitoramento de sensores e atuadores. Foram também usados sensores e atuadores em cenário virtual no Cisco Packet Tracer.

Neste trabalho temos sensores e atuadores ligados aos controladores com interface de rede que comunicam via *WiFi* com a API. A plataforma web também comunica com a API para obter informações de forma dinâmica e para despoletar certas ações nos diferentes atuadores através de botões.

2. Arquitetura

Com isto, queremos monitorizar e controlar uma Casa Inteligente num quarto e numa sala de estar.

O quarto tem um sensor de temperatura, de humidade e um sensor de presença de luz. Além disso também tem um LED, uma luz, um candeeiro e uma ventoinha.

A sala de estar tem um sensor de chamas, um *buzzer* (emite um sinal sonoro), uma luz, uma câmara e uma porta que é possível ser controlada através da plataforma.

Todos estes dispositivos estão ligados a controladores com interface de rede que comunicam via *WiFi* com a API (servidor na *cloud*) através da Internet, como consta a Figura 1.

As ligações sensor/atuador – controlador são as seguintes:

Controlador 1	Controlador 2	Controlador 3	Controlador 4	Controlador 5
• Sensor de chamas • Luz (sala) • Led • Câmara*	• Candeeiro • *Buzzer* (sinal sonoro) • Sensor de temperatura e de humidade • Luz (quarto)	• Ventoinha	• Sensor de presença de luz	• Porta

Tabela 1 – Lista de sensores e de controladores aos quais estão ligados

*A câmara não está ligada ao controlador com fios, mas comunica com esta através de comunicação *WiFi*. O controlador apenas processa a imagem recebida através da comunicação para enviá-la para o servidor.

Resumindo a figura da arquitetura IoT usada no trabalho é a seguinte:

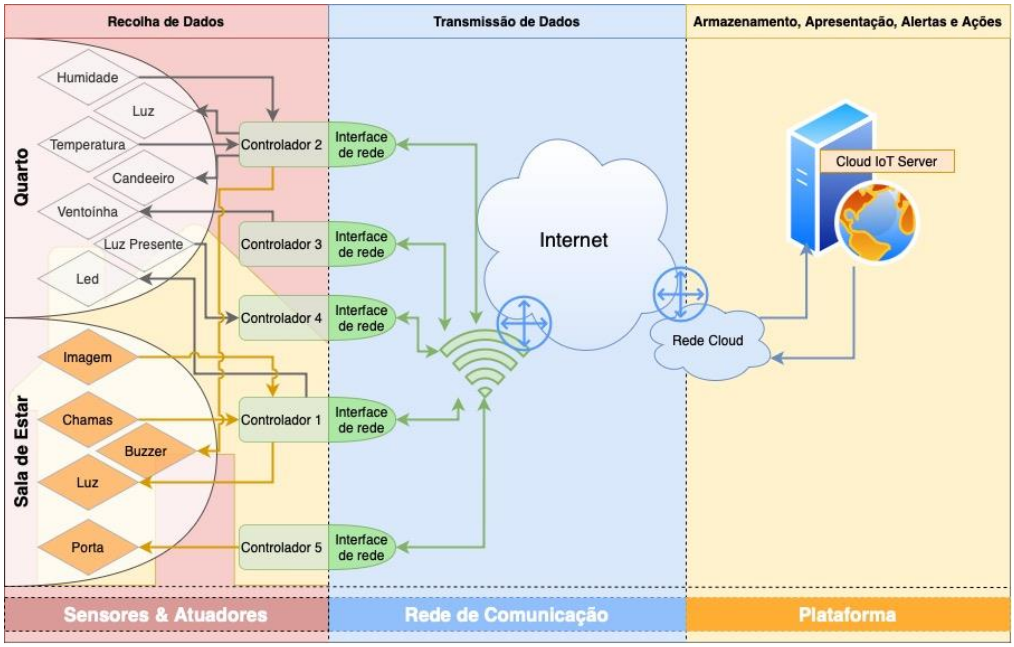

Figura 1 - Arquitetura IoT usada no projeto

3. Implementação

Primeiramente, como foi dito no Introdução, a plataforma web (*dashboard*) permite o registo de novos utilizadores e tem 4 níveis de utilizador (RESTRITO, NORMAL, ADMIN E META_ADMIN), sendo que cada um dos níveis tem diferentes privilégios, os quais lhes permitem ter acesso a funcionalidades diferentes. O nível RESTRITO apenas tem acesso à página *about*. O nível NORMAL tem acesso à *dashboard* e às paginas das divisões (para além dos privilégios do RESTRITO). O nível ADMIN tem acesso aos históricos e pode gerir os níveis dos utilizadores de nível inferior (para além dos privilégios do NORMAL), ou seja, consegue tornar um utilizador RESTRITO num NORMAL, por exemplo. O nível META_ADMIN é igual ao ADMIN mas ainda pode alterar o nível de um ADMIN. Apenas pode existir um META_ADMIN. A plataforma segue uma decisão representada pelo diagrama de fluxos seguinte:

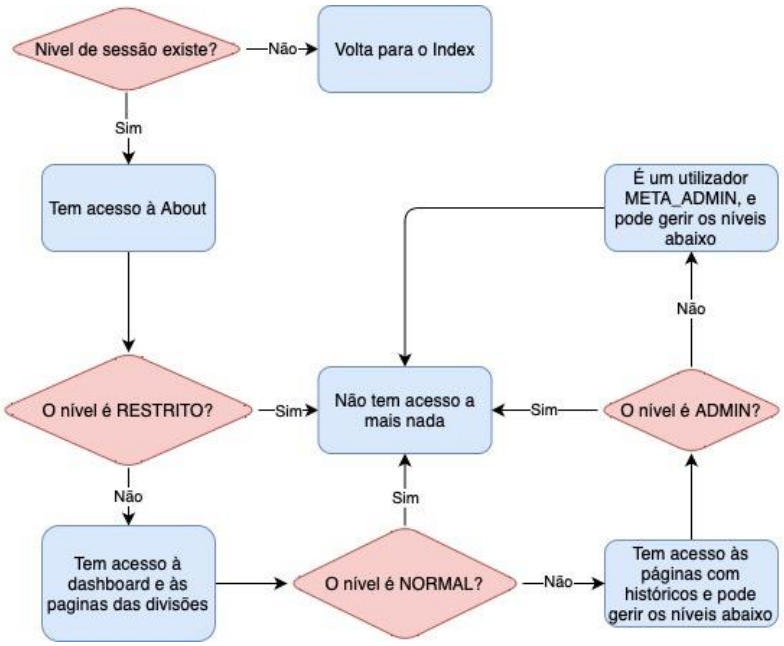

Figura 2 - Diagrama de fluxos das decisões da plataforma quanto ao nível de utilizador

A plataforma web permite obter a informação sobre os diferentes sensores e atuadores de forma dinâmica (através de pedidos GET à API em JavaScript e da substituição da informação obtida numa *template* já definida, na diretoria *templates*, e com estruturas de decisão implementadas num ficheiro de "comportamentos", *comportamentos.js*) e permite

eventos que despoletam ações (através de pedidos POST à API) em dispositivos, tal como será descrito a seguir.

Assim, temos os eventos seguintes:

- O sensor de temperatura que está ligado ao controlador 2 lê a temperatura do quarto e faz um POST desta à API de 5 em 5 segundos, através do seu código em C++. Por sua vez, o controlador 1 está a pedir o valor da temperatura do quarto à API e liga o LED caso a temperatura seja menor que 28 graus, através do seu código em Python, e faz um pedido POST à API com a informação do estado do LED de 5 em 5 segundos. Além disso, o controlador 4 também fica a pedir à API a temperatura do quarto e liga uma ventoinha se esta for superior a 25 graus, e faz um POST do estado da ventoinha de 5 em 5 segundos, através do seu código em JavaScript.

- O sensor de chamas que está ligado ao controlador 1 verifica a presença de chamas na sala de estar e faz um POST desta à API de 5 em 5 segundos, através do seu código em Python. Por sua vez, o controlador 2 está a pedir à API o valor da presença de chamas na sala de estar, de 5 em 5 segundos, e despoleta uma ação de tocar um alerta sonoro se for detetado a presença das chamas, através do seu código em C++.

- O sensor de presença de luz que está ligado ao controlador 3 verifica a presença de luz no quarto e faz um POST à API de 5 em 5 segundos, através do seu código em JavaScript. Por sua vez, o controlador 2 está a pedir à API se há presença de luz (de 5 em 5 segundos) e acende um candeeiro se não houver presença desta, através do seu código em C++.

- A informação de todos os sensores e atuadores (exceto o do alerta sonoro) é apresentado na *dashboard* (através de POST's à API) e altera a sua estrutura (imagens e "pills" – sinais de aviso/alerta). Se a temperatura exceder os 25 graus, a imagem muda e a "pill" muda de aspeto e se a temperatura for inferior a 15 graus a "pill" também muda. Se a humidade passa dos 60% a imagem e "pill" muda e se a humidade for inferior a 15 a "pill" muda. O estado das luzes, do LED, do sensor de chamas, do sensor de presença de luz, da porta, da ventoinha

e do candeeiro altera as imagens e as "pills". Além disso, a imagem da câmara é atualizada na plataforma. Todos os valores e detalhes dos dispositivos são enviados à API por pedidos POST através do código desenvolvido nos controladores aos quais estão associados de 5 em 5 segundos. A plataforma faz pedidos GET à API de 5 em 5 segundos, através do seu código em JavaScript, para apresentar tudo de maneira atualizada. É de notar que a API só aceita valores e detalhes se os seus parâmetros estiverem corretos, e só aceita imagens com menos de 1000KB com extensão ".png" ou ".jpg".

- Finalmente, existem interruptores na *dashboard* que despoletam ações nos diferentes dispositivos. É possível ligar ou desligar a luz do quarto e da sala e abrir ou fechar a porta da sala através desses interruptores. Se ligarmos ou desligarmos o interruptor, a plataforma faz imediatamente um POST à API com o novo estado do dispositivo. O controlador 1 pede à API o estado da luz da sala de 5 em 5 segundos e liga ou desliga a luz dependendo do estado recebido, no seu código em Python. O controlador 2 pede à API o estado da luz do quarto de 5 em 5 segundos e liga ou desliga a luz dependendo do estado recebido, através do seu código em C++. O controlador 5 pede à API o estado da porta da sala de 5 em 5 segundos e abre ou fecha a porta dependendo do estado recebido, através do seu código em JavaScript.

Para implementar esta solução, como será explicado no capítulo Cenário de Teste, usámos um *Raspberry Pi 3 Model B* (controlador 1 do capítulo anterior), um *Arduino MKR1000* (controlador 2 do capítulo anterior) e 3 MCU's no Cisco Packet Tracer (controladores 3, 4 e 5 do capítulo anterior):

	Controlador	Linguagem de Programação
Controlador 1	Raspberry Pi 3 Model	Python
Controlador 2	Arduino MKR1000	C++
Controlador 3	MCU (Cisco Packet Tracer)	JavaScript
Controlador 4		
Controlador 5		

Tabela 2 - Lista de Controladores e Linguagens de Programação associadas

A *dashboard* permite também aceder aos históricos de todos os seus dispositivos (exceto o histórico de imagens da câmara). Com isso implementámos todos os eventos e a *dashboard*, e assim podemos ter uma solução IoT ideal e seguir para o cenário de testes.

4. Cenário de Teste

Com tudo o explicado em mente, temos uma *dashboard* (website, que foi desenvolvida na primeira fase do projeto e melhorada no segundo) e uma API (ficheiro *api.php* em desenvolvido na primeira fase e também alterada na segunda para aceitar imagens). O servidor IoT foi corrido num dos nossos computadores pessoais com a ajuda de um serviço web local através de um software com Apache, e todos os controladores quer físicos ou virtuais estavam conectados à mesma rede *WiFi* do computador que estava a correr o serviço web local.

Para este cenário de teste, em termos de controladores foram utilizados:

1. Um Arduino MKR1000 (ligado ao computador por um cabo USB);
2. Um Raspberry Pi 3 Model B;
3. Três MCU's no Cisco Packet Tracer.

Em termos de coisas (sensores e/ou atuadores) foram usados:

1. Um DHT11 (sensor de temperatura/humidade do quarto) ligado ao Arduino (pino "+" ao pino "Vcc", pino "data" ao pino "0" e pino "-" ao pino GND);
2. Dois LED's (luz do quarto e led do quarto) ligados ao Raspberry Pi (no "led" a parte positiva ao pino GPIO2 e a negativa num GROUND, na "luz" a parte positiva ao pino GPIO5 e a negativa noutro ground);
3. Sensor de chamas ligado ao Raspberry Pi (pino "+" a um "Vcc", pino "D0" ao GPIO3 e pino "G" a um pino ground);
4. Buzzer ativo ligado ao Arduino (pino "+" ao pino "7" e o pino "-" no ground);
5. LED *buit-in* do Arduino (luz da sala);
6. LED RGB (candeeiro do quarto) ligado ao Arduino (pino "R" ao pino "11", pino "G" ao pino "10", pino "B" ao pino "9" e pino "-" ao ground);
7. A câmara de um *smartphone*, usando a aplicação *DroidCam*.
8. Um sensor de presença de luz ligado a um MCU do ambiente do Packet Tracer (pino "D0");
9. Um motor (ventoinha) ligado a um segundo MCU do ambiente do Packet Tracer (pino "A0");
10. Uma porta ligada a um terceiro MCU do ambiente do Packet Tracer (pino "D0");

Nota: o ground de cada sensor/atuador do Arduino estava ligado a um *spliter* que por sua vez estava ligado ao ground do Arduino, isto porque o Arduino apenas tem um pino ground.

Depois de ligados estes sensores e atuadores, procedemos ao desenvolvimento do código para implementar os eventos nos diferentes controladores. O código do Raspberry Pi, num ficheiro *proj.py*, foi escrito usando o Visual Studio Code (editor de texto da nossa escolha, que também foi usado para desenvolver a *dashboard*); o código do Arduino, num ficheiro *proj.ino*, foi escrito usando o *software* Arduino IDE; e o código dos restantes MCU's, em JavaScript, foi escrito num editor integrado com o Packet Tracer (aba *programming* de cada MCU).

Para utilizar o Raspberry Pi, tivemos de instalar o Sistema Operativo que nos foi dado pelos professores da unidade curricular. Após a intalação, tínhamos acesso às diretorias do SBC utilizando o *samba*. Depois para executarmos o código desenvolvidos (programa que corre num *loop* infinito), utilizámos o utilitário *ssh* na linha de comando do nosso computador pessoal (que estava configurado para se conectar à mesma rede WiFi em que o Raspberry Pi estava) para ter acesso ao terminal do Raspberry Pi. Posteriormente, executámos o comando *python proj.py* e tínhamos o código a correr. Também é neste código que se realiza uma captura da imagem (através do modulo cv2) da câmara do *smartphone* que é guardada no diretório corrente do ficheiro em *Python*.

Para usar o Arduino, ligámos o controlador ao computador pessoal com um cabo USB, e após desenvolver o código, fazíamos *upload* do código ao Arduino, utilizando o *software* Arduino IDE, e o código já estava a correr no controlador. Neste código foi necessário efetuar todas as configurações para este se conectar à mesma rede de todos os outros dispositivos através da tecnologia de comunicação *WiFi*.

Quanto aos MCU's no ambiente do Cisco Packet Tracer (ficheiro projeto.pkt), o código em JavaScript foi desenvolvido na aba "Programming" de cada MCU e depois foi apenas necessário clicar em *Run*. Com isso já tínhamos os MCU's a funcionar.

A *dashboard* e a API foram desenvolvidas em *HTML*, *CSS*, *PHP* e *JavaScript* usando como editor de texto o Visual Studio Code e o IntelliJ IDEA. A *dashboard* atualiza dinamicamente os seus "cards" e tabelas fazendo pedidos GET às informações dos dispositivos que se encontram no ficheiro *estrutura.txt* da diretoria da API.

Com tudo desenvolvido, montado e com o serviço web local ligado, conseguíamos começar a testar a nossa solução IoT. Todos o funcionamento da comunicação entre dispositivos e *dashboard* através da API pode ser representado pelo seguinte diagrama:

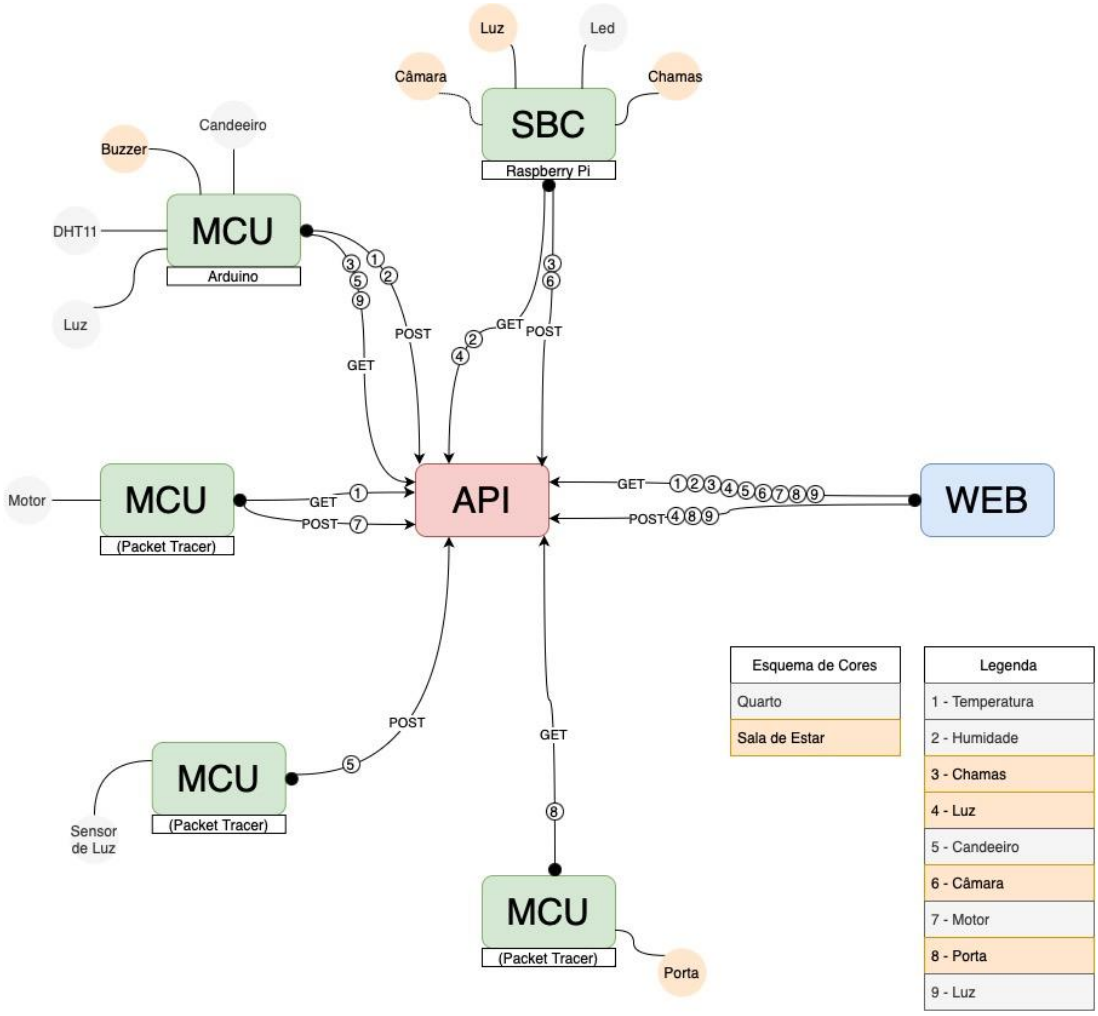

Figura 3 - Diagrama das comunicações entre dispositivos e web

5. Resultados obtidos

Com todas as montagens feitas seguimos para a observação dos resultados. Fizemos uma tabela para esclarecer os eventos que nós conseguimos ver no cenário de testes:

Origem	Destino	Sucesso/Insucesso
Pedido POST à API de 5 em 5 segundos de todos os valores dos sensores e atuadores (exceto o buzzer) ligados a todos os controladores	Pedido GET de 5 em 5 segundos em JavaScript na *dashboard* para apresentar todos estes valores e alterar a estrutura (imagens e "pills") de forma dinâmica sem a necessidade de refrescar a página	Sucesso
Pedido POST à API usando JavaScript para alterar o estado da luz da sala (*dashboard*)	Pedido GET de 5 em 5 segundos do estado da luz da sala no script Python do SBC, e observação da alteração do estado do LED em função da informação recebida pelo GET	Sucesso
Pedido POST à API usando JavaScript para alterar o estado da luz do quarto (*dashboard*)	Pedido GET de 5 em 5 segundos do estado da luz da sala no script em C++ do Arduino, e observação da alteração do estado do LED em função da informação recebida pelo GET	Sucesso
Pedido POST à API usando JavaScript para alterar o estado da porta da sala (*dashboard*)	Pedido GET de 5 em 5 segundos do estado da luz da sala no script JavaScript de um MCU do ambiente Packet Tracer, e observação da alteração do estado da porta em função da informação recebida pelo GET	Sucesso

Pedido POST à API de 5 em 5 segundos usando JavaScript num MCU do ambiente do Packet Tracer com o valor da presença de luz no quarto	Pedido GET de 5 em 5 segundos usando C++ no Arduino para obter o valor da presença de luz no quarto. Quando não há presença de luz, o LED RGB acende (candeeiro do quarto)	Sucesso
Pedido POST à API de 5 em 5 segundos usando C++ no Arduino da temperatura do quarto	Pedido GET de 5 em 5 segundos usando o script Python no SBC para obter a temperatura do quarto. O LED associado liga caso a temperatura seja menor que 28 graus,	Sucesso
	Pedido GET de 5 em 5 segundos usando JavaScript num MCU no ambiente do Packet Tracer para obter a temperatura do quarto. Caso a temperatura do quarto seja maior que 25 graus o motor liga	Sucesso
Pedido POST à API de 5 em 5 segundos usando Python no SBC do valor de presença de chamas na sala	Pedido GET de 5 em 5 segundos usando C++ no Arduino para obter o valor da presença de chamas na sala. Caso as chamas estejam presentes o buzzer liga.	Sucesso

Tabela 3 - Tabela dos eventos observados no cenário de testes

Como é possível ver, conseguimos alcançar todos os objetivos a nível de eventos. Além disso foi possível observar que a *dashboard* funciona como era intencionado funcionar, com o seu refrescamento dinâmico (“cards”, tabelas, “pills” e histórico) e com os seus privilégios e possibilidade de registo (criação de novos utilizadores). A câmara (com o auxílio do *DroidCam*) também foi um sucesso. O script em Python conseguia tirar uma foto e fazer o POST desta de 5 em 5 segundos.

6. Conclusão

Em suma, foi possível realizar uma solução IoT com uma plataforma web com vários níveis de permissão, possibilidade de registo, páginas dinâmicas que se atualizam "sozinhas" e com históricos para cada sensor/atuador.

Foi possível realizar uma montagem com vários sensores/atuadores (quer em cenário físico quer em cenário virtual) que se comunicam entre eles e com a plataforma web através de pedidos HTTP (GET's e POST's) a uma API. Esta é uma possível solução que poderia ser implementada numa Casa Inteligente real para se monitorizar e/ou controlar duas divisões (quarto e sala de estar).

Finalmente, também é possível concluir que conseguimos atingir os nossos objetivos propostos. Mesmo assim não ficámos satisfeitos com a responsividade de todos os dispositivos, porque os pedidos HTTP aconteciam de 5 em 5 segundos, e achamos que seria melhor se tudo acontecesse instantaneamente. Apesar de tudo, achamos que é uma excelente montagem.

7. Bibliografia

- Diagramas feitos em:
  - https://www.drawio.com/
- Tutoriais de funções de PHP em:
  - https://www.php.net/
  - https://www.phptutorial.net/
- Bootstrap:
  - https://getbootstrap.com/
  - https://bootstrapbay.com/
  - https://bootstrapious.com/
- Tutoriais de HTML, CSS, PHP e JavaScript:
  - https://www.w3schools.com/
- Questões gerais sobre programação ou montagem de dispositivos:
  - https://chat.openai.com/
  - https://stackoverflow.com/
  - PowerPoint’s Teóricos da UC