Smart Hotel

Alexandre Leal Marques Batista Jerónimo

Rafael Saramago Amaral

Trabalho de Projeto da unidade curricular de Tecnologias de Internet

Leiria, junho de 2021

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 (Internet das Coisas)
IPLeiria	Instituto Politécnico de Leiria
MCU	Microcontrolador
SBC	Single Board Computer

Índice

1. Introdução

O tema escolhido para a realização do nosso projeto, no âmbito da Unidade Curricular de Tecnologias de Internet, consiste no desenvolvimento de um sistema de IOT que permita criar um hotel inteligente.

Um hotel inteligente – ou "*Smart Hotel*" – foca-se em, por um lado, proporcionar o máximo conforto ao cliente e, por outro, reduzir ao mínimo o trabalho do seu gestor. Isto é possível através de sensores e atuadores que formam um sistema que se controla sozinho, como por exemplo, um sensor de temperatura que, detetando que a temperatura atingiu um certo limite definido pelo gestor, aciona o ar condicionado de modo a baixar a temperatura, tornando o ambiente mais agradável e economizando, assim, recursos energéticos.

O projeto será desenvolvido utilizando tecnologias *web* e simulado num ambiente virtual através do Cisco Packet Tracer. Com as tecnologias *web* HTML, PHP, CSS, JavaScript e MySQL será criada uma aplicação ligada a uma base de dados, que permite auxiliar a gestão e visualização de todos os sensores e atuadores existentes no hotel e serão ainda desenvolvidos *scripts* em Python de modo a controlar remotamente determinados atuadores.

Esta aplicação, conhecida como *Dashboard*, estará interligada com uma API desenvolvida especificamente para a concretização do projeto. Esta API será responsável por processar todos os pedidos do Dashboard e do ambiente virtual e assim modificar o estado do sensor ou atuador relevante na base de dados.

2. Arquitetura

De seguida, temos um esquema básico das relações entre a API, os scripts em Python, o ambiente virtual do hotel e o respetivo Dashboard.

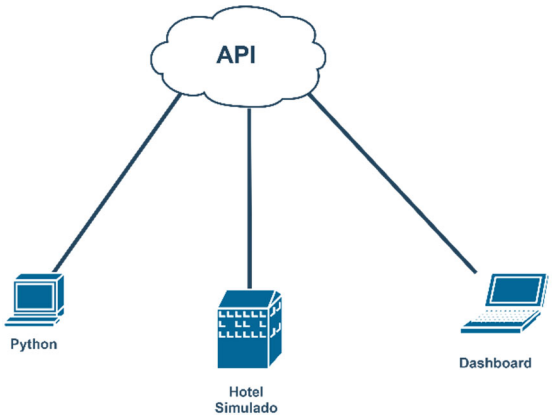

Figura 1 – Relações da API

Como o esquema indica, a API é responsável por interpretar todos os pedidos, quer sejam dos *scripts* Python, do Dashboard ou do ambiente virtual. No ambiente virtual, API estará ligada ao servidor do hotel, que por sua vez estará ligado a um router *wireless* ligado a todos os atuadores, microcontroladores e computadores.

Este próximo esquema representa uma ideia do ambiente simulado em maior detalhe.

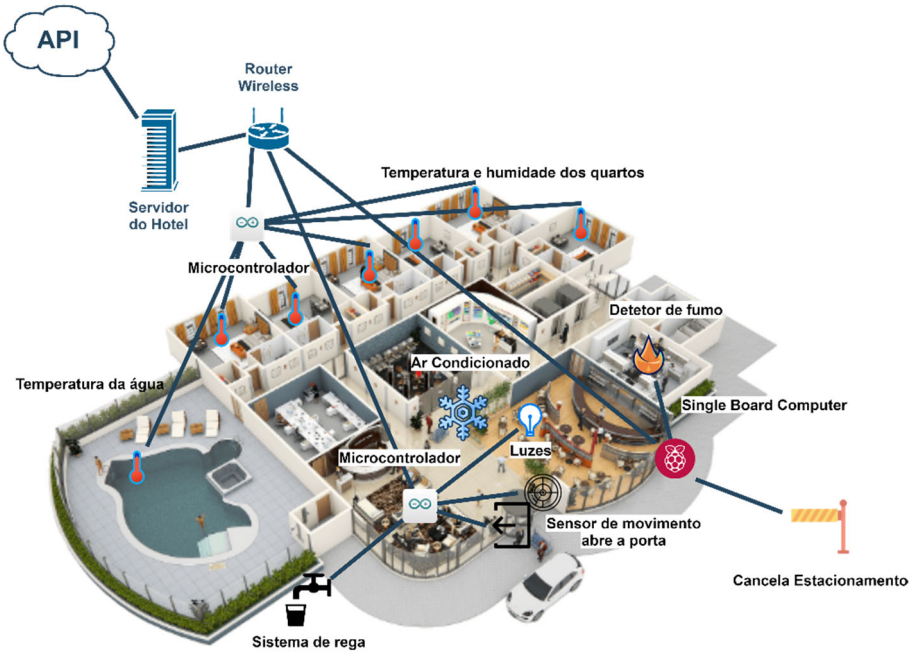

Figura 2 – Protótipo do ambiente virtual

Como ilustrado, todos os sensores e atuadores estão ligados a um respetivo microcontrolador ou Single Board Computer que, por sua vez, estão ligados ao servidor IOT do hotel. Os microcontroladores e computadores são responsáveis por realizar pedidos HTTP GET e POST à API do servidor.

Cada quarto do hotel terá sensores de temperatura e humidade ligados a um microcontrolador que envia os dados para a API, que, por sua vez, atualiza os respetivos campos na base de dados. Haverá ainda um sensor de temperatura na piscina, um detetor de fumo na cozinha, um sensor de movimento na porta principal. Todos estes sensores serão controlados pelos microcontroladores e computadores, também eles controlados pela API através dos pedidos HTTP.

Em relação aos atuadores, haverá uma cancela de estacionamento, um sistema de rega automática, luzes, uma porta automática e ar condicionado, também controlados pelos MCU e SBC.

3. Implementação

A API e o Dashboard do projeto foram implementados com recurso às *framework* Laravel, Livewire e Tailwind, que integram as tecnologias *web* HTML, CSS, PHP, JavaScript e MySQL. Os *scripts* de ativação de atuadores e recolha de dados foram realizados com a linguagem Python e o ambiente virtual foi simulado através do Cisco Packet Tracer.

A *framework* Laravel permitiu um desenvolvimento mais rápido e simples da API e do Dashboard, facilitando ainda a melhoria da segurança e proteção de dados, por exemplo através de um sistema de *login*, vedando assim todas as páginas do Dashboard a utilizadores não autorizados pelo hotel (à exceção da página de login), e ainda do *hashing* das palavras-passe guardadas na base de dados, de modo que, mesmo que alguém consiga aceder à base de dados, as palavras-passe sejam impossíveis de decifrar. Como anteriormente a este projeto não tínhamos ainda trabalhado com esta tecnologia, fizemos uma pesquisa exaustiva e uma vasta análise à sua documentação, assim como a fóruns e vídeos sobre a mesma (StackOverflow e Laracasts).

A "*sub-framework*" Livewire, uma *framework* do Laravel, permitiu a criação de pedidos assíncronos do Dashboard, ou seja, a informação do Dashboard está sempre atualizada em tempo real, sem ser necessário "refrescar" a página.

A *framework* de CSS e Javascript "Tailwind" tornou o desenvolvimento de um *design* e *layout* apelativo muito mais rápido. É ainda uma framework extremamente leve e rápida, algo muito importante no desenvolvimento de uma página *web*. Foram utilizados alguns ícones do "icon-set" Font Awesome no Dashboard.

Foram realizados ainda dois *scripts* em Python. O *script* "toggle_devices.py" mostra ao utilizador um menu com vários atuadores e as suas respetivas teclas. Ao pressionar essa tecla, o estado do atuador é alterado. Por exemplo, quando o utilizador carrega na tecla 'L', as luzes da entrada do hotel passam do estado "desligadas" para "ligadas" e vice-versa.

O *script* "parking_webcam.py" deteta se a cancela de estacionamento abriu. Se a cancela abrir, o programa tira uma fotografia ao veículo que está a transpor a cancela. Como se trata de uma simulação, foi utilizada uma *webcam* de um computador para tirar as fotografias. As imagens são, então, enviadas para a API que, por sua vez, as armazena na diretoria

"/public/storage/parking" do projeto. O Dashboard mostra estas fotografias por ordem da mais recente à mais antiga e cada imagem com um tamanho máximo de 1000kB.

O ambiente virtual, desenvolvido com o Cisco Packet Tracer, é constituído por um servidor IOT ligado a um *switch*, que por sua vez está ligado a um *Access Point*. Todos os atuadores, microcontroladores e computadores estão ligados a este *Access Point*. Quando um estado de um atuador é alterado, os microcontroladores são responsáveis por enviar um pedido POST à API, que por sua vez atualiza a base de dados. Os microcontroladores recebem ainda pedidos GET, feitos também à API, e atualizam então os respetivos atuadores. Estes programas desenvolvidos em JavaScript correm periodicamente, pois o estado do atuador pode ter sido alterado na base de dados através de eventos realizados no Dashboard ou nos programas Python.

A base de dados, desenvolvida em MySQL, possui dez tabelas (ou entidades); no entanto, só cinco delas são cruciais ao funcionamento da aplicação, pois as outras correspondem a tabelas geradas automaticamente pelo Laravel, que tratam das suas migrações. As migrações, ou "migrations", são o método utilizado pelo Laravel para criar a base de dados a partir do zero, de modo que seja idêntica em qualquer computador em que seja utilizada.

As entidades principais da base de dados são as tabelas “actuators” (ou atuadores), “logs”, “regions” (ou regiões), “sensors” (ou sensores) e “users”. Estas entidades relacionam-se de acordo com o seguinte esquema Entidade-Relação.

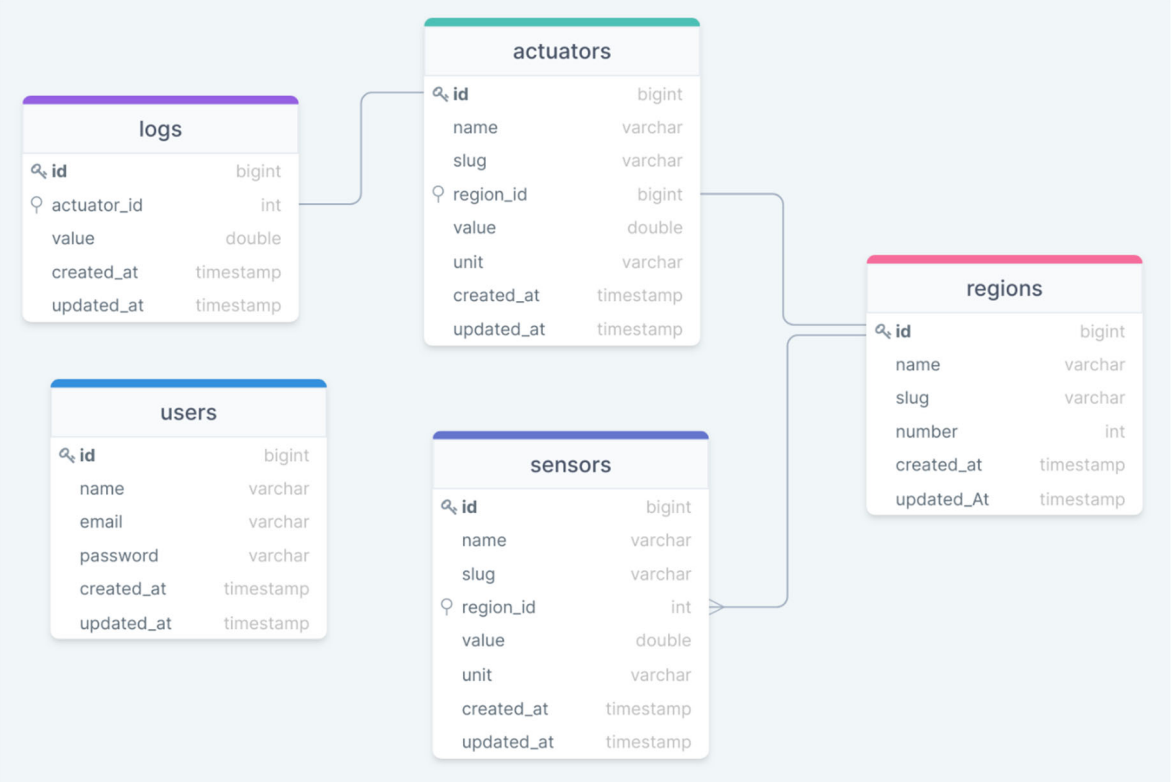

Figura 3 – Modelo ER da Base de Dados

3.1. Descrição de cada Entidade

Entidade	**Objetivo**
Regions	Identificação de cada região do hotel. (Ex: id: 5, name: room, slug: Room, number: 3)
Actuators	Armazena os dados de cada atuador. (Exemplo de atuador pertencente ao “room 3” – name: barrier, slug: Barrier, region_id: 5, value: 0, unit: c_o)
Sensors	Armazena os dados de cada sensor. (Exemplo de sensor pertencente ao “room 3” – name: temperature, slug: Temperature, region_id: 5, value: 14, unit: ºC)
Logs	Armazena dados relevantes do respetivo atuador sempre que o estado (coluna “value”) deste é alterado.
Users	Entidade independente das acima referidas. Armazena o nome, email e password (em *hash*) do utilizador.

Tabela 1 - Entidades Relacionais

Pontos importantes:

- As colunas "region_id" são chaves estrangerias (ou *foreign keys)* que representam a coluna "id" da respetiva região. A coluna "actuator_id", seguindo a mesma lógica, representa o "id" do respetivo atuador na tabela dos Logs.
- A coluna "created_at" é automaticamente preenchida com a data e hora da sua criação, e a coluna "updated_at" é alterada sempre que uma outra coluna seja modificada (Ex: a coluna "value").
- A coluna "slug" é responsável por guardar o nome "apresentável" da entidade, por exemplo, o atuador do Ar Condicionado tem o nome de código (name) "air_conditioner", mas a sua "slug" é conhecida como "Ar condicionado". Esta "slug" é o nome que é apresentado ao utilizador no Dashboard.
- A coluna "unit" guarda uma *string* com a unidade de medida/estado de cada sensor e atuador. No caso dos atuadores, neste projeto foi utilizada a convenção "c_o" para simbolizar os estados "Closed" e "Open" e "off_on" para simbolizar "OFF" e "ON". Estas distinções são relevantes tanto no código do Dashboard como nos microcontroladores.

Como a API foi desenvolvida com recurso à *framework* Laravel, esta está configurada por caminhos ou "routes" definidos no ficheiro "/routes/api.php". Estes caminhos são responsáveis por interpretar os pedidos GET e POST, que serão validados e executados de acordo com as regras impostas nos seus respetivos "controllers" (SensorController.php, ActuatorController.php, UploadController.php), devolvendo o código HTTP adequado.

No anexo 8.15 encontra-se um exemplo de um caminho (*route*) do ficheiro "api.php".

Estas linhas de código são responsáveis por redirecionar todos os pedidos à API. No exemplo em anexo, este código redireciona pedidos GET relativos aos sensores para o método "show" do controlador "SensorController.php". Como o código se encontra no ficheiro "api.php", o Laravel transforma o caminho "/sensor" em "/api/sensor".

Por exemplo, para fazermos um pedido GET à API, ao sensor de temperatura do quarto 1, temos de passar um URL como:

http://localhost/api/sensor?region_name=room&region_number=1&name=temperature

O método "show" do SensorController vai então validar o pedido GET, pois este tem de possuir os parâmetros obrigatórios (name e region_name) e valida ainda se o sensor e a região existem e se os tipos de dados passados são os corretos. Caso falhe alguma destas validações, a API retorna o código 400 (Bad Request). Se o pedido for feito com sucesso, retorna o código 200 (OK).

Foi anexado, no ponto 8.16, um exemplo de uma validação do nome e número da região.

O Dashboard é todo construído através de "Blade components", componentes que permitem a reutilização de código repetido, como a estrutura principal do Dashboard (Sidebar, Topbar e Footer). É ainda utilizada a "*sub-framework*" Livewire, uma *framework* do Laravel que possibilita, como referido anteriormente, a realização de pedidos assíncronos em determinados componentes. Estes componentes assíncronos são utilizados na página principal do Dashboard, de modo a manter os valores dos atuadores e sensores atualizados em tempo real.

São ainda disponibilizados três botões (*toggles*) na página principal do Dashboard, responsáveis por ligar e desligar as luzes da entrada do hotel e o sistema de rega e, ainda, abrir e fechar a cancela do parque de estacionamento. O Dashboard possui ainda um botão que permite ao utilizador trocar entre modo escuro e claro, por motivos de acessibilidade ou simplesmente por preferência.

É possível também criar novas contas de utilizador; no entanto, apenas um utilizador previamente registado tem permissões para o fazer. O utilizador inicial é criado automaticamente quando se cria a base de dados através das migrações do Laravel. Isto porque foram ainda implementados "seeders" de base de dados por motivos de teste. Estes "seeders" semeiam a base de dados com registos de exemplo. Esta funcionalidade torna-se especialmente útil na criação instantânea de uma grande quantidade de *logs* de exemplo.

3.2. Eventos

De seguida, apresenta-se uma tabela com alguns dos eventos que ocorrem entre os diversos ambientes implementados.

De	Para	Evento
Python	Packet Tracer	Prime tecla B (POST) → Fecha/Abre a cancela do estacionamento
Python	Dashboard	Prime tecla B (POST) → Dashboard atualiza o estado da cancela (Fechada/Aberta)
Dashboard	Packet Tracer	Botão de desligar/ligar luzes (POST) → Luzes apagam e acendem.
Dashboard	Python	Abre-se a cancela (POST) → É tirada uma fotografia (webcam) ao carro a entrar (Python).
Packet Tracer	Dashboard	Alarme de incêndio ON (IOT Server) → Ativa os extintores de incêndio (POST)
Packet Tracer	Python	Cancela é aberta → tira fotografia ao carro a entrar (Python).

Tabela 2 - Eventos

3.3. Funcionalidades Opcionais

Nas funcionalidades complementares, optámos por implementar 7 funcionalidades.

- Pedidos assíncronos
- Segurança dos dados aplicacionais
- Utilização de um número de sensores e atuadores superior aos mínimos exigidos (12 sensores e 7 atuadores)
- Utilização do modelo MVC (Framework Laravel)
- Utilização de base de dados para armazenamento
- Criação de uma página com histórico das últimas imagens recebidas
- Controlo das imagens recebidas através de código PHP de modo a aceitar apenas as imagens com um tamanho máximo de 1000kB e com a extensão .jpg ou .png

Como anteriormente referido, os pedidos assíncronos são realizados com a *framework* Livewire e a segurança dos dados consiste no *hashing* das palavras-passe na base de dados.

O controlo do tamanho das imagens é feito no ficheiro "routes/web.php", código que se encontra anexado no ponto 8.17 do último capítulo.

4. Cenário de Teste

Como referido anteriormente, o cenário de teste foi desenvolvido num ambiente virtual realizado no Cisco Packet Tracer. A imagem seguinte mostra o ambiente simulado com todos os sensores e atuadores nas respetivas divisões/regiões do hotel.

Figura 4 - Ambiente Simulado

Como a figura demonstra, foram utilizados 5 MCU e 2 SBC, todos ligados à rede do hotel e fundamentalmente controlados pela API.

4.1. Atuadores e Sensores

Alguns sensores e atuadores ficaram inutilizados por falta de tempo na conclusão do ambiente virtual (como a luz de emergência), no entanto, estes não foram contabilizados nas contagens de sensores e atuadores previamente feitas.

O MCU0, na região da entrada do hotel (conhecida como "foyer" em inglês e na base de dados), está ligado a um sensor de temperatura na porta A0, um sensor de movimento na porta D0 e à porta da entrada do hotel na porta D1.

Este MCU é responsável pela abertura da porta principal sempre que o sensor de movimento deteta movimento, e recolhe ainda dados da temperatura ambiente da região.

Os MCU1, 2, 3 e 4 encontram-se no quarto respetivo ao seu número. Cada um destes MCU está ligado a um sensor de temperatura e um sensor de humidade nas portas A0 e A1, respetivamente. Estes MCU atualizam o valor dos sensores na base de dados periodicamente.

O SBC0, na entrada do hotel, está ligado ao ar condicionado na porta D0 e às luzes de entrada na porta D1.

Este SBC é responsável pela ativação do ar condicionado sempre que a temperatura atinja pelo menos 15ºC e a ativação ou desativação das luzes de entrada, controladas pelo Dashboard e pelos programas Python.

O SBC1, na zona do estacionamento e jardim (conhecido em inglês e na base de dados como "parking" e "garden", respetivamente), está ligado à cancela de estacionamento na porta D0, a um sistema de rega na porta D1, ao LED da cancela de estacionamento na porta D2 e a um sistema de extinção de incêndios na porta D3.

Este SBC abre e fecha a cancela e liga e desliga o LED quando a cancela é aberta/fechada externamente através do Dashboard ou do *script* "toggle_devices.py". Liga e desliga também o sistema de rega quando este é ativado ou desativado externamente e recebe ainda o estado atual do extintor de incêndios. Este extintor de incêndios é ativado pelo servidor IOT sempre que o detetor de fumo dispara.

Segue-se agora uma lista com os atuadores e outra com os sensores utilizados e implementados.

Atuadores:

- Porta principal do hotel
- Luzes de entrada, representadas por um candeeiro
- Ar condicionado da entrada
- Sistema de rega
- Extintor de incêndio da cozinha
- Cancela do parque de estacionamento, representada por uma porta

Sensores:

- Um sensor de temperatura por cada quarto (total de 4 quartos)
- Um sensor de humidade por cada quarto (total de 4 quartos)
- Um sensor de temperatura na entrada do hotel
- Sensor de movimento na entrada do hotel
- Detetor de fumo na cozinha

As seguintes tabelas mostram as interações, portas e modos que os SBC e MCU têm com cada sensor e atuador, e ainda os endereços de IP, rede, SSID e palavra-passe da rede do hotel, a que estão conectados.

Microcontrolador: MCU0			
IP: 192.168.0.50 Server Address: 192.168.0.2 SSID: HotelGateway Password: smarthotel			
Sensores			
Nome	Porta	Modo	Descrição
Motion Sensor	D0	INPUT	HIGH quando deteta movimento, LOW após 5 segundos sem movimento.
Temperature Sensor	A0	INPUT	Envia dados da temperatura para o MCU.
Atuadores			
Nome	Porta	Modo	Descrição
Smart Door Foyer	A1	OUTPUT	Porta abre quando o sensor de movimento deteta movimento.

Tabela 3 - Especificações do MCU0

Microcontrolador: MCU1, 2, 3 e 4			
IP: 192.168.0.51, 192.168.0.52, 192.168.0.53, 192.168.0.54, respetivamente Server Address: 192.168.0.2 SSID: HotelGateway Password: smarthotel			
Sensores			
Nome	Porta	Modo	Descrição
Temperature Sensor	A0	INPUT	Envia dados da temperatura para os MCU.
Humidity Sensor	A1	INPUT	Envia dados da humidade para os MCU.

Tabela 4 - Especificações dos MCU1, 2 3 e 4

Single Board Computer: SBC0			
IP: 192.168.0.60 Server Address: 192.168.0.2 SSID: HotelGateway Password: smarthotel			
Atuadores			
Nome	Porta	Modo	Descrição
Air Conditioner Foyer	D0	OUTPUT	ON se a temperatura da entrada do hotel for superior a 15ºC. OFF se a temperatura for menor ou igual a 15ºC.
Lights Foyer	D1	OUTPUT	ON/OFF se o utilizador ligar/desligar as luzes externamente (no Python ou Dashboard).

Tabela 5 - Especificações do SBC0

Single Board Computer: SBC1			
IP: 192.168.0.61 Server Address: 192.168.0.2 SSID: HotelGateway Password: smarthotel			
Atuadores			
Nome	Porta	Modo	Descrição
Parking Barrier	D0	OUTPUT	Abre/fecha se o utilizador abrir/fechar a cancela externamente (no Python ou Dashboard)
Sprinkler	D1	OUTPUT	ON/OFF se o utilizador ligar/desligar o sistema de rega externamente (no Python ou Dashboard)
Parking Barrier Light	D2	OUTPUT	ON quando a cancela está aberta. OFF quando a cancela está fechada.
Fire Sprinkler	D3	OUTPUT	ON quando o detetor de fumo dispara. OFF se o detetor de fumo não detetar nada.

Tabela 6 - Especificações do SBC1

5. Resultados obtidos

No cenário virtual, a variável da temperatura ambiente varia de 6ºC a 24ºC da meia-noite às 12h00. Das 12h00 às 18h00 a temperatura volta a baixar até aos 6ºC. Os MCU1, 2, 3 e 4 foram configurados para, a cada 5 segundos, enviar os dados dos sensores a que estão ligados para a API.

O MCU0 deteta a cada 300ms se o sensor de movimento detetou algo e envia a temperatura atual da entrada do hotel para a API. Caso o sensor tenha detetado movimento, o MCU abre a porta durante 5 segundos e envia o estado da porta para a API (estado "aberta" durante 5 segundos e depois "fechada").

O SBC0 verifica a cada 5 segundos se a temperatura é ou não superior a 15ºC e confirma também através da API se foi feito um pedido para acender as luzes. Ativa então o ar condicionado se a temperatura for, de facto, superior a 15ºC e acende ou apaga as luzes se foi feito o pedido.

O SBC1, a cada 3 segundos deteta, se a cancela foi aberta/fechada e se o sistema de rega e de extinção de incêndio foram ativados/desativados. Caso afirmativo, realiza então as ações adequadas. O *script* Python pede a cada 5 segundos o estado da cancela de estacionamento. Quando deteta que a cancela foi aberta, é tirada uma fotografia com a *webcam*, que por sua vez é enviada para a API, que a guarda no local adequado (diretoria "/public/storage/parking").

6. Conclusão

Consideramos que o projeto Smart Hotel foi concretizado com sucesso e encontra-se funcional. Todas as funcionalidades cruciais – o Dashboard, os *scripts* em Python e o ambiente simulado – estão operacionais. Alguns sensores e atuadores no ambiente virtual ficaram desligados e inutilizados por falta de tempo; porém, estes eram apenas complementares ao projeto e não uma necessidade, daí termos estabelecido prioridades. As funcionalidades opcionais listadas anteriormente foram todas implementadas corretamente.

Como não tínhamos trabalhado com a *framework* Laravel, tivemos de pesquisar e analisar documentação da mesma, tanto em diversos fóruns como na documentação oficial do Laravel, o que encarámos como um desafio. Aprendemos também a fazer pedidos assíncronos através da *sub-framework* Livewire, algo muito útil no desenvolvimento de um Dashboard, e também a desenvolver o *design* e *layout* de páginas *web* com a *framework* Tailwind.

De futuro, seria interessante adicionar funcionalidades como gráficos que apresentam a variação dos dados dos sensores ao longo do tempo, e ainda a possibilidade de personalização do Dashboard por conta e tipo de utilizador.

7. Bibliografia

- Documentação do Laravel 8: https://laravel.com/docs/8.x
- Documentação do Laravel Livewire: https://laravel-livewire.com/
- Documentação do Tailwind: https://tailwindcss.com/docs
- Laracasts: https://laracasts.com/
- StackOverflow: https://stackoverflow.com/
- Font Awesome: https://fontawesome.com/

8. Anexos

8.1. Código do Microcontrolador 0

```
/* Opens the door when motion is detected.
 * Also responsible for sending temperature data to the server
 */

var url_post_sensor = "http://projeto-ti.test/api/sensor";
var url_post_actuator = "http://projeto-ti.test/api/actuator";
var url_get = "http://projeto-
ti.test/api/actuator?name=door&region_name=foyer";

var motionSensor = 0;
var tempSensor = A0;
var door = 1;
var isClosed = 1;

// Converts the motion sensor status into 0 and 1 (from 0 and 1023)
function detectMotion(slot) {
    if (analogRead(slot)) return 1;
    return 0;
}

// Connvert the temperature into celsius
function readTemp(slot) {
    var value = analogRead(slot);
    return (value / 1023) * 200 - 100;
}

function setup() {
    pinMode(motionSensor, INPUT);
    pinMode(tempSensor, INPUT);
    pinMode(door, OUTPUT);
}

function loop() {
    var values;
    var hasMotion = detectMotion(motionSensor);
    var temp = Math.round(readTemp(tempSensor));


    values = {
        name: "temperature",
        value: temp,
        region_name: "foyer",
    };

    // POST temperature
    RealHTTPClient.post(url_post_sensor, values);

    // Door POST payload
    values = {
        name: "door",
        value: isClosed ? 0 : 1,
        region_name: "foyer",
    };

    // If the motion sensor detects movement
```

```
    if (hasMotion) {
        // GET door status
        RealHTTPClient.get(url_get, function (status, data) {
            if (status != 200) return Serial.println("Error!");
            isClosed = 0;
            if (data == 1) return;
            isClosed = 1;
        });

        // POST door's status (0 - Closed / 1 - Open)
        if (isClosed) {
            isClosed = 0;
            customWrite(door, "1,0");

            values = {
                name: "door",
                value: 1,
                region_name: "foyer",
            };

            RealHTTPClient.post(url_post_actuator, values);
        }
    } else {
        // If the door is open, closes the door and sends a POST
        if (!isClosed) {
            isClosed = 1;
            customWrite(door, "0,0");

            values = {
                name: "door",
                value: 0,
                region_name: "foyer",
            };

            RealHTTPClient.post(url_post_actuator, values);
        }
    }

    delay(300);
}
```

8.2. Código do Microcontrolador 1, 2, 3, 4

A única diferença no código dos microcontroladores 1, 2, 3 e 4 é a variável "roomNumber" que é igual ao número do respetivo quarto. Por exemplo, no quarto 4, a variável "roomNumber" é igual a 4.

```
/* Sends the room's sensors' data to the server */

var url = "http://projeto-ti.test/api/sensor";

var tempSensor = A0;
var humiSensor = A1;
var roomNumber = 1;

// Converts the temperature to Celsius an returns it
function readTemp(slot) {
    var value = analogRead(slot);
    return (value / 1023) * 200 - 100;
}

// Converts the humidity to percentage and returns it
function readHumi(slot) {
    var value = analogRead(slot);
    return (value * 100) / 1023;
}

function setup() {
    pinMode(A0, INPUT);
    pinMode(A1, INPUT);
}

function loop() {
    var temp = Math.round(readTemp(tempSensor));
    var humi = Math.round(readHumi(humiSensor));
    var values;

    values = {
        name: "temperature",
        value: temp,
        region_name: "room",
        region_number: roomNumber,
    };

    // POST of the room's temperature
    RealHTTPClient.post(url, values);

    values = {
        name: "humidity",
        value: humi,
        region_name: "room",
        region_number: roomNumber,
    };

    // POST of the room's humidity
    RealHTTPClient.post(url, values);

    delay(5000);
}
```

8.3. Código do SBC 0

```
/* Controls the Air Conditioner */

var url_post = "http://projeto-ti.test/api/actuator";
var url_get_temp = "http://projeto-
ti.test/api/sensor?name=temperature&region_name=foyer";
var url_get_lights = "http://projeto-
ti.test/api/actuator?name=lights&region_name=foyer";
var ac = 0;
var lights = 1;
var isOn = 0;

function setup() {
    pinMode(ac, OUTPUT);
}

function loop() {
    // GET foyer's temperature
    RealHTTPClient.get(url_get_temp, function (status, data) {
        if (status != 200) return Serial.println("Error!");
        if (data > 15) {
            isOn = 1;
            return digitalWrite(ac, HIGH);
        }
        isOn = 0;
        digitalWrite(ac, LOW);
    });

    // GET light's state (off/on)
    RealHTTPClient.get(url_get_lights, function (status, data) {
        if (status != 200) return Serial.println("Error!");
        if (data == 1) return customWrite(lights, 2);
        customWrite(lights, 0);
    });

    // POST temperature
    var values = {
        name: "air_conditioner",
        value: isOn,
        region_name: "foyer",
    };

    RealHTTPClient.post(url_post, values);

    delay(5000);
}
```

8.4. Código do SBC 1

```
/* Toggles the sprinklers and parking barrier and turns the LED ON */

var url_get_barrier = "http://projeto-
ti.test/api/actuator?name=barrier&region_name=parking";
var url_get_sprinklers = "http://projeto-
ti.test/api/actuator?name=sprinklers&region_name=garden";
var url_post = "http://projeto-ti.test/api/actuator";
var barrier = 0;
var sprinklers = 1;
var lights = 2;
var fireSprinkler = 3;

// Converts the motion sensor status into 0 and 1 (from 0 and 1023)
function detectMotion(slot) {
    if (analogRead(slot)) return 1;
    return 0;
}

function setup() {
    pinMode(barrier, OUTPUT);
    pinMode(sprinklers, OUTPUT);
    pinMode(lights, OUTPUT);
    pinMode(fireSprinkler, OUTPUT);
}

function loop() {
    var values;


    // GET the barrier's state
    RealHTTPClient.get(url_get_barrier, function (status, data) {
        if (status != 200) return Serial.println("Error!");
        // If the barrier is open
        if (data == 1) {
            // Turns the light ON and opens the barrier
            digitalWrite(lights, HIGH);
            return customWrite(barrier, "1,0");
        }

        // Turns the light OFF and closes the barrier
        digitalWrite(lights, LOW);
        customWrite(barrier, "0,0");
    });

    // GET the sprinklers' state and turns them OFF/ON
    RealHTTPClient.get(url_get_sprinklers, function (status, data) {
        if (status != 200) return Serial.println("Error!");
        if (data == 1) return customWrite(sprinklers, 1);
        customWrite(sprinklers, 0);
    });

    // POST the fire sprinkler's state (ON if Fire Alarm is ON)
    values = {
        name: "fire_sprinkler",
        value: customRead(fireSprinkler),
        region_name: "kitchen",
    };

    RealHTTPClient.post(url_post, values);

    delay(3000);
}
```

8.5. Código Python do ficheiro “parking_webcam.py”

```
# This script takes a photo (webcam) every 5 seconds
# as long as the parking barrier is open

from datetime import datetime
from os import remove
from time import sleep

from cv2 import VideoCapture, destroyAllWindows, imwrite
from requests import get, post


def post_file(file):
    url = "http://projeto-ti.test/api/upload"

    # Opens the image
    f = open(file, "rb")
    image = {"image": f}

    # Sends the image to the server
    r = post(url, files=image, data={"location": "parking"})

    # If the POST wasn't successful
    if r.status_code != 200:
        print("POST error!\n" + r.text)
        return

    # Closes and deletes the file after a successful POST
    f.close()
    remove(file)


def upload_picture_when_open(camera):
    r = get("http://projeto-
ti.test/api/actuator?name=barrier&region_name=parking")

    # If the GET wasn't sucessful
    if r.status_code != 200:
        return

    # Creates a string with the current time (e.g. "11-06-2021 19_02_26")
    now = datetime.now().strftime("%d-%m-%Y %H_%M_%S")

    # Creates a string with the filename (e.g "11-06-2021 19_02_26.png")
    file = now + ".png"

    # Stores the barrier's state returned by the GET request (0 - Closed
/ 1 - Open)
    is_open = r.json()

    # If the barrier is open (0 - Closed / 1 - Open)
    if is_open:
        print("Picture taken: " + now)
        ret, pic = camera.read()
        imwrite(file, pic)
        post_file(file)


try:
    # Readies the camera
```

```
    camera = VideoCapture(0)
    print("This program is responsible for taking a picture every time
the parking barrier opens.")
    print("Press CTRL+C to exit the program.")
    while True:
        upload_picture_when_open(camera)
        sleep(5)
except KeyboardInterrupt:
    # Shuts the camera down and exits
    camera.release()
    destroyAllWindows()
    print("Program exited by user.")
```

8.6. Código Python do ficheiro “toggle_devices.py”

```
# This script shows the user a menu with multiple actuators
# When the corresponding key is pressed, the actuator is toggled (ON / OFF ; Open / Closed)

from msvcrt import getch, kbhit

from requests import get, post


def post_to_api(data):
    # Creates the payload of necessary data for the POST request
    payload = {
        "name": data[0],
        "region_name": data[1],
        "value": data[2],
    }

    # Sends the payload to the server
    r = post("http://projeto-ti.test/api/actuator", data=payload)

    # If the POST wasn't successful
    if r.status_code != 200:
        print("Request error!" + r.text)


def get_from_api(actuator_name, region_name):
    r = get("http://projeto-ti.test/api/actuator?name=" + actuator_name + "&region_name=" + region_name)

    # If the GET request wasn't successful
    if r.status_code != 200:
        return print("Request error!")

    # Returns the device's current state (0 - OFF/Closed | 1 - ON/Open)
    is_toggled = r.json()
    return 0 if is_toggled else 1


try:
    print("Usage:\n[B] Toggle barrier\n[L] Toggle lights\n[S] Toggle sprinklers\n[CTRL+C] Exit")
    while True:
        if kbhit():
            # Converts the input character to uppercase for easier validation
            key = getch().upper()

            # Checks the user input and toggles the device (0 - OFF/Closed | 1 - ON/Open)
            if key == b"B":
                toggled = get_from_api("barrier", "parking")
                if toggled:
                    print("Barrier opened")
                else:
                    print("Barrier closed")
                post_to_api(["barrier", "parking", toggled])
            elif key == b"L":
                toggled = get_from_api("lights", "foyer")
                if toggled:
```

```
                    print("Lights ON")
                else:
                    print("Lights OFF")
                post_to_api(["lights", "foyer", toggled])
            elif key == b"S":
                toggled = get_from_api("sprinklers", "garden")
                if toggled:
                    print("Sprinklers ON")
                else:
                    print("Sprinklers OFF")
                post_to_api(["sprinklers", "garden", toggled])
            else:
                print("Invalid option!")
except KeyboardInterrupt:
    print("Program terminated by user.")
```

8.7. Código do ficheiro "/routes/api.php"

```
Route::get('/sensor', [SensorController::class, 'show']);

Route::post('/sensor', [SensorController::class, 'update']);

Route::get('/actuator', [ActuatorController::class, 'show']);

Route::post('/actuator', [ActuatorController::class, 'update']);

Route::post('/upload', [UploadController::class, 'upload']);

Route::middleware('auth:sanctum')->get('/user', function (Request
$request) {
    return $request->user();
});
```

8.8. Código do ficheiro "/routes/web.php"

```
Route::group(['middleware' => ['auth:sanctum']], function () {
    Route::get('/', function () {
        $sensors = Sensor::orderBy('region_id')->get(); // Receives every sensor
from the database
        $actuators = Actuator::orderBy('region_id')->get(); // Receives every
actuator from the database
        $regions = Region::orderBy('name')->orderBy('number')->get(); // Receives
every region from the database
        return view('dashboard/index', compact('sensors', 'actuators',
'regions')); // Shows the dashboard with the sensors', actuators' and regions'
data
    })->name('dashboard');

    Route::get('/logs/{region}/{number?}', function ($regionName, $number = null)
{
        // Receives every region from the database
        $regions = Region::orderBy('name')->orderBy('number')->get();

        // Equals to true if the region exists
        $exists = Region::where('name', $regionName)->where('number', $number)-
>exists();

        // Returns an error if the region couldn't be found
        if (!$exists) return 'ERROR: Region not found!';

        // Finds the region
        $region = Region::where('name', $regionName)->where('number', $number)-
>first();

        // Stores the sensors and actuators from the specified region
        $sensors = Sensor::where('region_id', $region->id)->get();
        $actuators = Actuator::where('region_id', $region->id)->get();

        // Creates an array with the ID of every actuator belonging to the
specified region
        $actuatorsIdArray = $actuators->pluck('id')->toArray();

        // Stores the logs from every actuator from the specified region
        $logs = Log::whereIn('actuator_id', $actuatorsIdArray)-
>orderBy('created_at', 'desc')->get();

        // Shows the dashboard with the sensors' data
        return view('dashboard/logs', compact(
            'sensors',
            'actuators',
            'regions',
            'region',
            'logs'
        ));
    })->name('logs');

    Route::get('/regions/{region}/{number?}', function ($regionName, $number =
null) {
        // Receives every region from the database
        $regions = Region::orderBy('name')->orderBy('number')->get();

        // Equals to true if the region exists
        $exists = DB::table('regions')
            ->where('name', $regionName)
            ->where('number', $number)
            ->exists();

        // Returns an error if the region couldn't be found
        if (!$exists) return 'ERROR: Region not found!';
```

```
        // Finds the region
        $region = DB::table('regions')
            ->where('name', $regionName)
            ->where('number', $number)
            ->first();

        // Stores the sensors and actuators from the specified region
        $sensors = Sensor::where('region_id', $region->id)->get();
        $actuators = Actuator::where('region_id', $region->id)->get();

        // Creates an array with the ID of every actuator belonging to the
specified region
        $actuatorsIdArray = $actuators->pluck('id')->toArray();

        // Stores the logs from every actuator from the specified region
        $logs = DB::table('actuators')
            ->whereIn('id', $actuatorsIdArray)
            ->orderBy('created_at', 'desc')
            ->get();

        return view('dashboard/regions', compact('sensors', 'actuators',
'regions', 'region')); // Shows the dashboard with the sensors' data
    })->name('regions');

    Route::get('/user', function (Request $request) {
        return $request->user();
    });

    Route::get('/profile', function () {
        $regions = Region::orderBy('name')->orderBy('number')->get();
        return view('profile/show', compact('regions'));
    })->name('profile');

    Route::get('/camera', function () {
        // Receives every region from the databse, ordered by name and number
        $regions = Region::orderBy('name')->orderBy('number')->get();
        // Stores the images location in a variable
        $path = public_path('storage/parking');
        // Get all files from the path
        $images = File::allFiles($path);
        // Checks if each image is bigger than 100kib or the file type isn't
either .png or .jpg
        foreach ($images as $key => $image) {
            if ($image->getSize() > 1024000 || ($image->getExtension() != 'png'
&& $image->getExtension() != 'jpg')) {
                unset($images[$key]);
            }
        }
        return view('dashboard/camera', compact('regions', 'images'));
    })->name('camera');

    Route::get('/register', function () {
        return view('auth/register');
    })->name('register');
});
```

8.9. Código Blade da View "/resources/views/dashboard/index.blade.php"

```
<x-dashboard.master title="Dashboard" :regions="$regions" isRoot="true">

    <!-- Highlighted Actuators -->
    <div class="grid grid-cols-1 gap-5 mt-6 sm:grid-cols-2 lg:grid-cols-
3">

        <livewire:dashboard.highlights :actuators="$actuators"
region="foyer" actuatorName="lights" svg="lightbulb" />

        <livewire:dashboard.highlights :actuators="$actuators"
region="garden" actuatorName="sprinklers" svg="faucet" />

        <livewire:dashboard.highlights :actuators="$actuators"
region="parking" actuatorName="barrier" svg="parking" />

    </div>

    <!-- Actuators Table -->
    <h3 class="mt-6 text-xl dark:text-light">Actuators</h3>
    <livewire:dashboard.table :devices="$actuators" isRoot=true />

    <!-- Sensors Table -->
    <h3 class="mt-6 text-xl dark:text-light">Sensors</h3>
    <livewire:dashboard.table :devices="$sensors" isRoot=true />

</x-dashboard.master>
```

8.10. Código Blade do componente "/resources/views/components/dashboard/master.blade.php"

```
<!DOCTYPE html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="csrf-token" content="{{ csrf_token() }}">

    <title>Dashboard - Hotel Inteligente</title>

    <!-- Styles -->
    <link rel="stylesheet" href="/css/styles.css" />
    <link rel="stylesheet" href="/css/app.css" />
    @livewireStyles
    <script
src="https://cdn.jsdelivr.net/gh/alpinejs/alpine@v2.8.2/dist/alpine.min.js"
defer></script>
</head>

<body>
    <div>
        <div class="flex h-screen overflow-y-hidden bg-white dark:bg-dark">

            <!-- Sidebar e Topbar -->
            <x-dashboard.sidebar :regions="$regions" />

            <!-- Main content -->
            <main class="flex-1 max-h-full p-5 overflow-hidden overflow-y-
scroll">

                <!-- Main content header -->
                <div class="flex flex-col items-start justify-between @if
(!$isRoot) pb-6 space-y-4 border-b @endif dark:border-teal-700 dark:text-light
lg:items-center
                    lg:space-y-0 lg:flex-row">
                    <div class="text-2xl font-semibold whitespace-nowrap">
                        {{ $title }}
                        @if ($numberedRegion && $numberedRegion->number)
                            <x-region-dropdown :regions="$regions"
:numberedRegion="$numberedRegion" />
                        @endif
                    </div>
                </div>

                {{ $slot }}

            </main>

            <!-- Footer -->
            <x-dashboard.footer />

        </div>
    </div>

    <script src="https://kit.fontawesome.com/6bed2cc76e.js"
crossorigin="anonymous"></script>
    <script src="/js/script.js"></script>

    @livewireScripts
</body>

</html>
```

8.11. Código PHP do componente "/app/View/Components/Dashboard/Master.php"

```
class Master extends Component
{
    public $title;
    public $regions;
    public $numberedRegion;
    public $isRoot;

    public function __construct($title, $regions, $numberedRegion = null,
$isRoot = false)
    {
        $this->title = $title;
        $this->regions = $regions;
        $this->numberedRegion = $numberedRegion;
        $this->isRoot = $isRoot;
    }

    public function render()
    {
        return view('components.dashboard.master');
    }
}
```

8.12. Código Blade da View "/resources/views/dashboard/regions.blade.php"

```
<x-dashboard.master title="Actuators and Sensors - {{ !$region->number ?
$region->slug : '' }}" :regions="$regions"
    :numberedRegion="$region">

    <div class="mt-6">
        <h3 class="inline text-xl dark:text-light">Actuators</h3>
        <livewire:dashboard.table :devices="$actuators"
:regions="$regions" />
    </div>
    <div class="mt-6">
        <h3 class="inline text-xl dark:text-light">Sensors</h3>
        <livewire:dashboard.table :devices="$sensors" :regions="$regions"
/>
    </div>
</x-dashboard.master>
```

8.13. Código do Controller "/app/Http/Controllers/SensorController.php"

O algoritmo do ActuatorController.php é idêntico a este, mudando apenas o objeto "sensor" para "actuator".

```
class SensorController extends Controller
{
    /**
     * Display the specified resource.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function show(Request $request)
    {
        // Checks if the necessary parameters exist
        if (!$request->name || !$request->region_name)
            return response('ERROR: Invalid parameters! Must contain
[name] [region_name]', 400);

        // Equals to true if the region exists and has a number (e.g.
Room 3)
        $exists = DB::table('regions')
            ->where('name', $request->region_name)
            ->where('number', $request->region_number)
            ->exists();

        // Returns an error if the region couldn't be found
        if (!$exists) return response('ERROR: Region not found!', 400);

        // Finds the region_id
        $region_id = DB::table('regions')
            ->where('name', $request->region_name)
            ->where('number', $request->region_number)
            ->first()->id;

        // Finds the sensor
        $sensor = DB::table('sensors')
            ->where('region_id', $region_id)
            ->where('name', $request->name)
            ->first();

        // Returns an error if the sensor couldn't be found
        if (!$sensor) return response('ERROR: Sensor not found!', 400);

        // Returns the sensor's data
        return $sensor->value;
    }

    /**
     * Update the specified resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function update(Request $request)
    {
        // Validate the Request
```

```
        if (!$request->name || !$request->region_name || $request->value
=== null)
            return response('ERROR: Invalid parameters! Must contain
[name] [region_name] [value]', 400);

        if (!is_numeric($request->value)) return response('ERROR: [value]
parameter must be a number!', 400);

        // Equals to true if the region exists
        $exists = DB::table('regions')
            ->where('name', $request->region_name)
            ->where('number', $request->region_number)
            ->exists();

        if (!$exists) return response('ERROR: Region not found!', 400);

        // Finds the region_id
        $region_id = DB::table('regions')
            ->where('name', $request->region_name)
            ->where('number', $request->region_number)
            ->first()->id;

        // Finds the sensor id
        $sensor = DB::table('sensors')
            ->where('region_id', $region_id)
            ->where('name', $request->name)
            ->first();

        if (!$sensor) return response('ERROR: Sensor not found!');

        // Stores the sensor's data from the database
        $sensor = Sensor::find($sensor->id);

        // Updates the sensor's value with the request's value
        $sensor->value = $request->value;

        // Saves the new value into the database
        return $sensor->save();
    }
}
```

8.14. Código do Controller "/app/Http/Controllers/UploadController.php"

```
class UploadController extends Controller
{
    public function upload(Request $request)
    {
        if (!$request->location) return response('ERROR: Must have a
[location] parameter!', 400);

        $image = $request->file('image');
        $allowedMimeTypes = ['image/jpeg', 'image/png', 'image/bmp']; //
Allowed file formats

        // If the image wasn't found
        if (!$image) return response('ERROR: Image not found!', 400);

        // If the file sent was a forbidden file type
        if (!in_array($image->getMimeType(), $allowedMimeTypes)) return
response('ERROR: Invalid file type!', 400);

        // Rename and move the file to the storage directory
        return $image->move('storage/' . $request->location, $image-
>getClientOriginalName());
    }
}
```

8.15. Código de uma "route" do ficheiro api.php

```
Route::get('/sensor', [SensorController::class, 'show']);
```

8.16. Código de uma validação de uma região do ficheiro ActuatorController.php

```
        // Equals to true if the region exists and has a number
        // (e.g. Room 3)
        $exists = DB::table('regions')
            ->where('name', $request->region_name)
            ->where('number', $request->region_number)
            ->exists();

        // Returns an error if the region couldn't be found
        if (!$exists) return response('ERROR: Region not found!', 400);
```

8.17. Controlo do tamanho das imagens

```
        // Stores the images location in a variable
        $path = public_path('storage/parking');
        // Get all files from the path
        $images = File::allFiles($path);
        // Checks if each image is bigger than 100kib or the file type
isn't either .png or .jpg
        foreach ($images as $key => $image) {
            if ($image->getSize() > 1024000 || ($image->getExtension() !=
'png' && $image->getExtension() != 'jpg')) {
                unset($images[$key]);
            }
        }
```