

## **Vídeo porteiro low-cost (IoT)**

Licenciatura em Engenharia Informática

2181454 – João Rebelo dos Santos

2181459 – Rúben Daniel Custódio Garcia

Leiria, julho de 2022

## **Vídeo porteiro low-cost (IoT)**

Licenciatura em Engenharia Informática

2181454 – João Rebelo dos Santos

2181459 – Rúben Daniel Custódio Garcia

Trabalho de Projeto da unidade curricular de Projeto Informático realizado sob a orientação do Professor Doutor David Ferreira Safadinho, do Professor Paulo Sampaio Abreu Madeira e do Professor Doutor António Manuel de Jesus Pereira.

Leiria, julho de 2022



# **Agradecimentos**

Dedicamos este trabalho ao Professor Doutor David Ferreira Safadinho e ao Professor Paulo Sampaio Abreu Madeira, que se disponibilizaram para nos auxiliar durante o desenvolvimento deste projeto e nos inspiraram a apontar sempre mais alto.

Agradecemos também às nossas famílias por todo o suporte ao longo dos anos e apoio moral durante a realização deste projeto e do curso. Estes agradecimentos estendem-se ainda aos amigos que nos incentivaram todos os dias e ofereceram apoio nos momentos críticos.

Gostaríamos ainda de dedicar este trabalho a George Kusunoki Miller, que foi e continua a ser uma inspiração para todos nós.

# Resumo

Com o aparecimento da pandemia de COVID-19 houve um elevado crescimento no interesse em formas de limitar o contacto com outras pessoas, ou seja, respeitar o distanciamento social.

Deste modo foi planeado o desenvolvimento de uma campanha *low-cost* que visa permitir que os utilizadores consigam saber quem está à porta remotamente a qualquer altura, bem como fazerem uma vídeo chamada com eles por meio da campanha, podendo abri-lhes a porta se o desejarem. A campanha a desenvolver deve ainda poder detetar movimento e alertar os utilizadores sempre que isto acontecer ou sempre que alguém tocar à campanha. Deve ainda estar disponível numa página *web* todas as imagens capturadas pela campanha, bem como informação estatística relativa ao número de vezes que foi detetado movimento num certo dia.

Assim, foi desenvolvida uma campanha à base do microcontrolador ESP32-CAM, sendo esta capaz de capturar imagem através de uma câmara, detetar movimento através de um *passive infrared sensor* (PIR) e abrir/fechar a porta através de um relé. Foram desenvolvidos dois servidores a correr em paralelo, um responsável por interagir com as campanhas através de Sockets, o outro por publicar a página *web* onde os utilizadores conseguem aceder a informações relevantes sobre as suas campanhas e interagir com estas, fazem ambos os servidores recurso a uma base de dados dedicada a guardar informações das campanhas e dos utilizadores. Os dois servidores correm em *threads* diferentes, pelo que partilham informações apenas quando necessário.

Para desenvolver o servidor *web* foi utilizado Flask como framework, utilizando diversas bibliotecas para facilitar a implementação das funcionalidades desejadas, utilizando ainda Jinja como renderizador *web* para as diversas páginas html.

O backend foi ainda colocado numa *virtual machine* (VM) na *cloud*, permitindo que o serviço seja utilizado a qualquer altura e em qualquer lugar. Foram utilizados exclusivamente free tiers pelo que o uso desta VM não causou despesas.

A nossa campanha diferencia-se das alternativas disponíveis por manter um custo extremamente baixo enquanto possui as principais funcionalidades que as restantes têm.

O *hardware* utilizado tem limitações no que toca a qualidade de imagem, que é bastante mais baixa do que a das alternativas para uso comercial, mas mantém uma qualidade alta o suficiente para o propósito para o qual foi desenvolvida. A campanha funciona ligada à corrente, pois não foi implementado um suporte para bateria, ainda assim facilmente se consegue implementar e testar essa funcionalidade.

# Abstract

With the appearance of the COVID-19 pandemic there was a large rise in interest in ways to limit contact with other people, in other words, ways to respect social distancing.

Bearing that in mind, we planned and developed a low-cost *doorbell* with the intent to allow its users to remotely check who is at their door, from anywhere at any time, as well as allowing them to do video calls with said people through the *doorbell*, having the ability to open the door for them, if they feel so inclined. Furthermore, the envisioned *doorbell* must also be able to detect movement and alert the user whenever movement is detected, or someone presses it. In addition, all pictures taken by the *doorbell*, as well as statistical data on the amount of time movement was detected in a particular day are to be made available in a web page.

To this end, a *doorbell* was developed based on an ESP32-CAM microcontroller, being capable of taking pictures with its camera, detect movement through a passive infrared sensor (PIR) and open/close the door through the use of a relay. Additionally, two servers were developed to run in parallel, one responsible for the interacting with the *doorbells* through Sockets, the other for publishing the web page where the users can access relevant information about their *doorbells* and interact with them. Both servers make use of a database dedicated to storing information related to the *doorbells* and the users. Also, both servers are running on different threads, thus sharing information strictly when necessary.

In order to develop the web server, we made use of Flask as a framework, employing many libraries in order to facilitate the implementations of the desired functionalities, also, Jinja was utilized as web renderer for several html pages.

The backend was then placed in a virtual machine (VM) on the cloud, allowing for the service to be used anywhere at any time. As a means to keep the VM from having financial costs we made use of only free tiers.

Our *doorbell* distinguishes itself from the alternatives by keeping an extremely low price whilst also having the main features the alternatives do.

Finally, the hardware used has limitations when it comes to image quality, it being considerably lower than the alternatives available for commercial use, but it maintains a high

enough quality for purpose for which it was developed. The *doorbell* functions plugged in, since we did not develop it with battery support as a concern, but this feature can easily be implemented and tested.



# Índice

<b>Agradecimentos.....</b>	<b>ii</b>
<b>Resumo.....</b>	<b>iii</b>
<b>Abstract.....</b>	<b>v</b>
<b>Lista de Figuras.....</b>	<b>x</b>
<b>Lista de tabelas .....</b>	<b>xi</b>
<b>Lista de siglas e acrónimos .....</b>	<b>xii</b>
<b>1. Introdução .....</b>	<b>1</b>
<b>2. Trabalhos Relacionados .....</b>	<b>2</b>
<b>2.1. IoT Doorbell (1) .....</b>	<b>3</b>
<b>2.2. Smart Doorbell System using IoT (2) .....</b>	<b>4</b>
<b>2.3. DIY Smart Home Doorbell (3) .....</b>	<b>5</b>
<b>2.4. Automatic Smart Door Bell with IoT (4).....</b>	<b>6</b>
<b>2.5. SS4H-SD Smart Doorbell (5).....</b>	<b>7</b>
<b>2.6. Ring Video Doorbell (6) .....</b>	<b>8</b>
<b>2.7. Smart Doorbell/Video Intercom System (7).....</b>	<b>9</b>
<b>2.8. DIY Video-Doorbell with ESP32 (8).....</b>	<b>10</b>
<b>2.9. DIY Doorbell (9) .....</b>	<b>11</b>
<b>2.10. DIY Ring-Doorbell (10).....</b>	<b>12</b>
<b>2.11. Comparações dos Trabalhos Relacionados.....</b>	<b>13</b>
<b>2.12. Síntese .....</b>	<b>15</b>
<b>3. Proposta de Solução .....</b>	<b>16</b>
<b>3.1. Descrição geral da Solução .....</b>	<b>16</b>

<b>3.2.</b>	<b>Requisitos .....</b>	<b>17</b>
<b>3.3.</b>	<b>Síntese.....</b>	<b>18</b>
<b>4.</b>	<b>Implementação .....</b>	<b>19</b>
<b>4.1.</b>	<b>Campainha desenvolvida.....</b>	<b>19</b>
4.1.1.	Microcontrolador (ESP32-CAM).....	19
4.1.2.	Câmara .....	19
4.1.3.	Sensor de movimento .....	19
4.1.4.	Botão .....	20
4.1.5.	Relê.....	20
<b>4.2.</b>	<b>Tecnologias utilizadas .....</b>	<b>21</b>
4.2.1.	Python Virtual Environments.....	21
4.2.2.	Flask .....	21
4.2.3.	Werkzeug .....	21
4.2.4.	Jinja .....	22
4.2.5.	Flask-Mail .....	23
4.2.6.	JWT .....	23
4.2.7.	OpenCV.....	23
4.2.8.	Página Web .....	24
4.2.9.	Base de Dados .....	25
4.2.10.	Serviço de Notificações via email.....	27
4.2.11.	Doorbell WiFIManager .....	27
4.2.12.	Backend base.....	28
4.2.13.	Socket.....	28
4.2.14.	Stream.....	29
4.2.15.	FCGI.....	31
4.2.16.	Plataforma na cloud.....	31
<b>4.3.</b>	<b>Configuração da plataforma .....</b>	<b>32</b>
<b>4.4.</b>	<b>Inicialização e configuração da Doorbell .....</b>	<b>34</b>
<b>4.5.</b>	<b>Síntese.....</b>	<b>37</b>
<b>5.</b>	<b>Testes e Resultados.....</b>	<b>38</b>

<b>5.1.</b>	<b>Testes realizados .....</b>	<b>38</b>
5.1.1.	Teste às capturas e ao sistema de alertas .....	39
5.1.2.	Teste à base de dados.....	40
5.1.3.	Teste ao relê.....	41
<b>5.2.</b>	<b>Análise dos resultados obtidos.....</b>	<b>41</b>
<b>5.3.</b>	<b>Síntese .....</b>	<b>41</b>
<b>6.</b>	<b>Conclusão .....</b>	<b>42</b>
<b>6.1.</b>	<b>Funcionalidades futuras.....</b>	<b>42</b>
	<b>Referências.....</b>	<b>45</b>
	<b>Glossário .....</b>	<b>48</b>

# Lista de Figuras

Figura 1 Componentes da 1ª campanha <sup>3</sup> .....	3
Figura 2 Diagrama do circuito da 2ª campanha <sup>4</sup> .....	4
Figura 3 Componentes da 3ª Campanha <sup>5</sup> .....	5
Figura 4 PCB montado <sup>8</sup> .....	7
Figura 5 Utilização da 7ª Campanha <sup>10</sup> .....	9
Figura 6 Arquitetura do Projeto .....	16
Figura 7 Dependências dos <i>templates</i> html .....	22
Figura 8 Esquema da base de dados .....	26
Figura 9 Esquema completo sobre o funcionamento do backend .....	30
Figura 10 Esquema para comunicação da Stream. ....	30
Figura 11- Configuração do FCGI na Cloud .....	31
Figura 12 – Exemplo de configuração da plataforma.....	33
Figura 13 Doorbell modo hotspot Wi-Fi .....	34
Figura 14 Doorbell modo hotspot Hotspot.....	35
Figura 15 Esquema completo da conexão da Doorbell ao servidor .....	36
Figura 16 Email recebido no teste .....	39
Figura 17– Capturas realizadas no teste .....	40
Figura 18– Comportamento do relê durante o teste .....	41

# Lista de tabelas

Tabela 1 – Comparação das várias soluções descritas anteriormente.....	14
--	----

# Lista de siglas e acrónimos

2FA	Two-Factor Authentication
AP	Access Point
AWS	Amazon Web Services
AWS SNS	Amazon Simple Notification Service
CCTV	Closed-Circuit Television
CGI	Common Gateway Interface
DC-DC	Direct Current to Direct Current
DIY	Do it yourself
DNS	Domain Name Service
E.I.	Engenharia Informática
FCGI	Fast Common Gateway Interface
FHD	Full High Definition
GPIO	General Purpose Input/Output
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IoT	Internet of Things
IP	Internet Protocol
IPL	Instituto Politécnico de Leiria
IR	Infrared Radiation
JST	Japan Solderless Terminal
JWT	JSON Web Token
LAN	Local Area Network
LCD	Liquid Crystal Display
LED	Light-Emitting Diode
MAC	Media Access Control
OTP	One-Time Password
PCB	Printed Circuit Board
PIR	Passive Infrared Sensor
PSRAM	Pseudo Static RAM
RFID	Radio-Frequency Identification

SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
T.I.	Tecnologias de Informação
TLS	Transport Layer Security
UUID	Universally Unique Identifier
VM	Virtual Machine
VPN	Virtual Private Network
YAML	Yet Another Markup Language





# 1. Introdução

Este documento tem como objetivo a explicação do planeamento e passos tomados no desenvolvimento de um projeto no âmbito da unidade curricular de Projeto Informático do 3º ano do curso de Engenharia Informática do Instituto Politécnico de Leiria.

Este projeto surge do impacto que as tecnologias de informação (TI) têm no nosso dia a dia, estando cada vez mais presente nas nossas casas, locais de trabalho e/ou lazer. A massificação da microeletrónica tem contribuído significativamente para a generalização do conceito de *internet of things* (IoT), e consequentemente, para o aparecimento de novas soluções low-cost que prometem tornar o nosso quotidiano mais inteligente e tecnológico a um preço acessível a qualquer pessoa.

Com base nisto este projeto é dedicado ao estudo e implementação de um vídeo porteiro utilizando um microcontrolador de baixo custo (ESP32-CAM), este último além de baixo preço também apresenta capacidade de captura de imagem e conexão à internet via Wi-Fi.

O vídeo permite o utilizador monitorizar remotamente a(s) sua(s) porta(s), ou seja, que tenha acesso a uma transmissão ao vivo da câmara a qualquer hora, de qualquer lugar, desde que este tenha uma conexão à internet, sendo-lhe também possível a capacidade de abrir a porta remotamente.

Para esse efeito foi necessário desenvolver uma página *web* a ser publicada pela placa ESP32-CAM que deve permitir a configuração do sistema, bem como identificar as pessoas que são detetadas com mais frequência. Deve ainda conter um sistema de alertas de email ou SMS para que o utilizador seja alertado de que alguém, está a tocar à campainha ou quando o movimento é detetado.

De seguida no capítulo 2 serão exploradas as alternativas disponíveis no mercado, que serão então comparadas ao projeto desenvolvido, no capítulo 3 será apresentada a solução proposta. O capítulo 4 diz respeito ao protótipo. Por fim no capítulo 5 serão apresentados os testes efetuados à solução desenvolvida.

## 2. Trabalhos Relacionados

Neste capítulo serão exploradas as alternativas presentes no mercado e compará-las com a solução implementada, tanto em termos de componentes como em termos de funcionalidades, mas primeiro é necessário mencionar e explicar algumas das ferramentas usadas.

Home Assistant <sup>1</sup> - Este projeto surge da necessidade de gerir os dispositivos inteligentes numa única *app*, facilitando a criação de uma casa inteligente. Este aplicativo é gratuito, *open-source* e *community-driven development*, estando assim a ser mantido pela comunidade e seguro em relação a privacidade do mesmo. Sendo este *open-source* qualquer pessoa pode sugerir, adicionar ou editar o código existente tornando esta uma ferramenta para todo o tipo de dispositivos e utilizações.

ESPHome <sup>2</sup> - Esta solução é uma extensão para o Home Assistant, trata-se de um sistema simples para controlar placas Esp, através de um ficheiro de configuração tipo *yet another markup language* (YAML), isto criar uma comunicação entre o Home Assistant e o Esp com esforço mínimo, com este tipo de sistema a configuração dos aparelhos torna-se rápida e simples.

---

<sup>1</sup> <https://www.home-assistant.io/>

<sup>2</sup> <https://esphome.io/>

## 2.1.IoT Doorbell <sup>3</sup> (1)

IoT Doorbell visa reinventar a campainha de modo que os utilizadores recebam alertas e saibam quem está à porta a partir de qualquer lugar. Esta trata-se de uma solução *low-cost* e *do it yourself* (DIY), optando por uma abordagem minimalista e simples de executar.

Utiliza um Raspberry Pi 1 como microcontrolador, um cartão MicroSD, uma câmara que se conecta via USB, um botão e um serviço de Amazon *simple notification service* (AWS SNS) que é usado para poder enviar os alertas ao utilizador.

Este sistema funciona com base num botão, quando este último é pressionado o Raspberry faz um pedido ao AWS SNS para este enviar uma mensagem ao utilizador sobre a tal ação decorrida.

Na Figura 1 podemos observar alguns dos componentes usados nesta campainha.



**Figura 1** Componentes da 1ª campainha <sup>3</sup>

---

<sup>3</sup> <https://www.hackster.io/taiyuk/iot-doorbell-fae18>

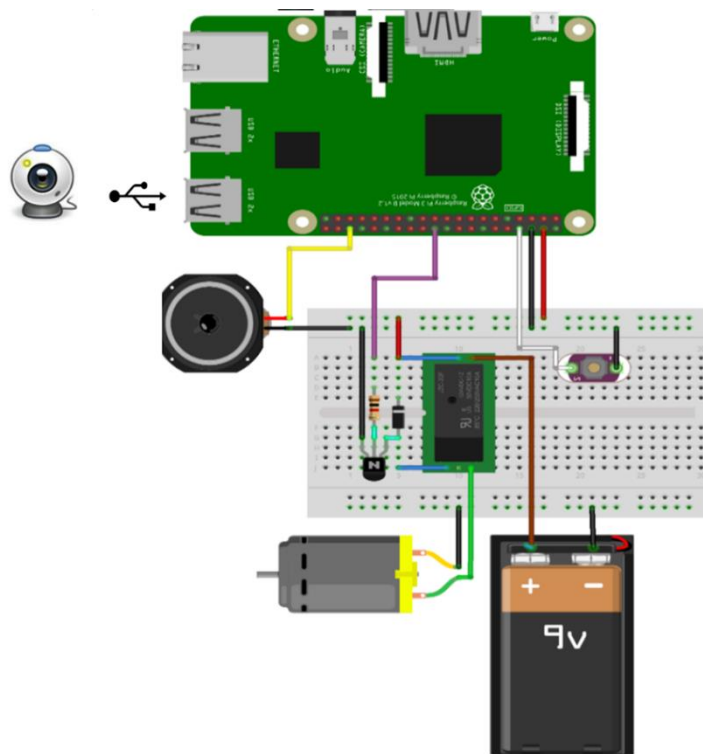
## 2.2. Smart Doorbell System using IoT <sup>4</sup> (2)

A Smart Doorbell System using IoT trata-se de um produto comercial que surge do desejo de eliminar as falhas de segurança existentes num dos sistemas atuais: *radio-frequency identification (RFID) security system*. Para este efeito utiliza reconhecimento facial como modo de identificação dos seus utilizadores.

Neste caso é usado um Raspberry Pi como microcontrolador, um cartão MicroSD, um PIR, uma câmara, um botão, e um servomotor associado a um circuito relê.

Quando alguém toca à campainha é lhe tirada uma fotografia que vai ser comparada com outras existentes na base de dados através de reconhecimento facial usando a biblioteca Open-CV, se a pessoa que pressionou a campainha for reconhecida a porta é lhe aberta, caso contrário o utilizador recebe um alerta por email com a fotografia e *one-time password (OTP)*. A porta será aberta quando quem tocou à campainha inserir o código OTP enviado para o utilizador, sendo então a sua fotografia de reconhecimento adicionada à base de dados.

Na Figura 2 é apresentado o diagrama do circuito.



**Figura 2** Diagrama do circuito da 2ª campainha <sup>4</sup>

<sup>4</sup> <https://www.pantechsolutions.net/smart-doorbell-system-using-iot>

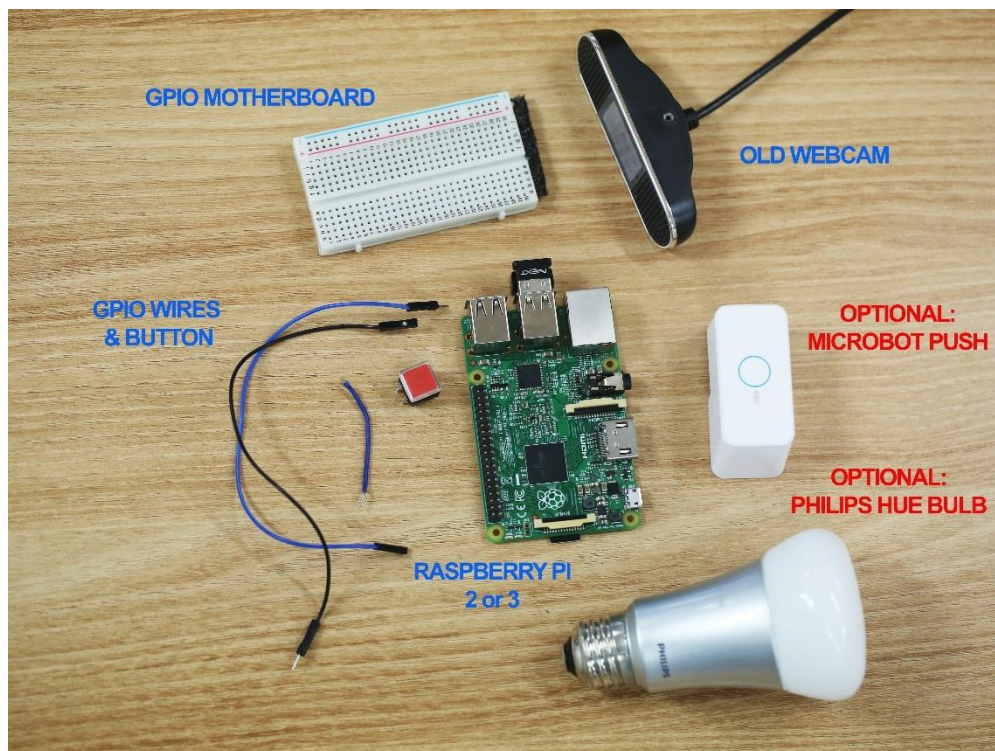
### 2.3.DIY Smart Home Doorbell <sup>5</sup> (3)

Esta solução visa disponibilizar uma alternativa DIY barata (abaixo de 40€) às *smart doorbells* disponíveis no mercado. Uma solução que serve para informar os utilizadores quando alguém está à porta através de alertas, dandos-lhe a opção de abrir a porta remotamente.

Utiliza um Raspberry Pi 3 (ou um Raspberry Pi 2 com adaptadores Bluetooth & Wi-Fi) usado como microcontrolador, um cartão MicroSD, uma *general-purpose input/output* (GPIO) *breadboard* e cabos, um botão, uma câmara, uma lâmpada e um *microbot push*, sendo estes dois últimos opcionais.

Quando alguém toca à campainha, é enviado uma notificação ao utilizador através de um *bot* na *app* Telegram<sup>6</sup>, após receber esta notificação o utilizador pode enviar um comando ao *bot* a pedir para este abrir a porta remotamente.

Na Figura 3 é mostrado o material que este projeto utiliza incluindo os opcionais.



**Figura 3** Componentes da 3ª Campainha <sup>5</sup>

<sup>5</sup> <https://www.instructables.com/DIY-Smart-Home-Doorbell-for-Less-Than-40/>

<sup>6</sup> <https://telegram.org/>

## 2.4. Automatic Smart Door Bell with IoT <sup>7</sup> (4)

Este projeto surge no âmbito de uma vasta pesquisa que foi feita em sistemas de segurança, tendo como objetivo desenvolver um aparelho eletrónico que deteta atividade humana e ativa a campainha automaticamente de modo a alertar os residentes. Tem ainda como segundo objetivo estar conectado à internet de forma a conseguir enviar dados recolhidos para armazenamento em *cloud*.

O material utilizado é um Arduino Uno como microcontrolador, um modulo Wi-Fi, uma câmara, um PIR, um *piezo buzzer* e um servomotor.

Este sistema não utiliza um botão para tocar a campainha, ao contrário dos projetos analisados anteriormente, para tal ação acontecer basta ser detetado movimento pelo PIR. Quando o movimento é detetado o *buzzer* é acionado reproduzindo um som de campainha, ativando também a câmara que começa a enviar, em tempo real, uma *live stream* para uma plataforma *cloud* (ThinkSpeak), o utilizador pode aceder a esta stream através de uma *app* Android, através desta dita *app* o utilizador pode controlar a porta. Posteriormente todos os dados de utilização são enviados e armazenados na *cloud*.

---

<sup>7</sup> <https://www.skyfilabs.com/project-ideas/automatic-smart-door-bell-using-iot>

## 2.5.SS4H-SD Smart Doorbell <sup>8</sup> (5)

Este projeto tem como objetivo criar uma campainha inteligente DIY barata e capaz de competir com as alternativas existentes no mercado. Esta foi desenhada no intuito de ser integrada num sistema de Home Assistant.

Utiliza um ESP32 como microcontrolador, uma antena externa do ESP32, um *printed circuit board* (PCB), um botão, um relê, um *direct current to direct current* (DC-DC) converter para energia, um *pseudo static* RAM (PSRAM) para melhor resolução de câmara e uma caixa para proteger o circuito montado.

Este projeto opta por não utilizar alimentação por bateria devido ao impacto no desempenho causado por esta, levando demasiado tempo a acordar do modo *deep sleep* e a conectar-se à internet. Esta solução também faz uso da aplicação ESPHome em junção com Home Assistant que gere os aparelhos conectados e como sistema de alertas para o utilizador. Uma configuração padrão destes dois serviços não permite que o utilizador receba notificações em qualquer parte do mundo, sendo necessário configurar uma *virtual private network* (VPN) ou fazer *port forwarding* para esse efeito, apesar de não ser mencionado o sistema aparenta funcionar apenas a nível *local area network* (LAN).

Quando o botão é premido a câmara captura uma foto que posteriormente é enviada ao utilizador através do Home Assistant, dando-lhe a opção de ignorar o alerta ou de abrir a porta remotamente.

Na Figura 4 podemos observar a campainha montada sem a sua caixa.



Figura 4 PCB montado <sup>8</sup>

<sup>8</sup> <https://smartsolutions4home.com/ss4h-sd-smart-doorbell/>



## 2.6. Ring Video Doorbell <sup>9</sup> (6)

A Ring Video Doorbell trata-se de um produto comercial vendido a um nível global, existem vários modelos de campainha vendidos, oferecendo uma diversidade de funcionalidades, no entanto, visto que estamos a comparar soluções *low-cost*, optámos por explorar apenas o modelo mais baratos que a Ring Video Doorbell disponibiliza.

Todos os seus modelos têm as seguintes características em comum:

- Conectam-se à rede Wi-Fi e enviam alertas quando movimento é detetado, o movimento é detetado até 9 metros de distância.
- Têm a capacidade de fazer vídeo chamadas, permitindo que o utilizador fale com quem estiver na porta.
- Podem gravar de dia e de noite através de infravermelhos e permitem a visualização de uma *live stream* através da sua *app*.
- Permite que os vídeos capturados sejam guardados na *cloud* através de uma subscrição.

O modelo mais barato disponível à data (9/7/2022) é o modelo Ring Video Doorbell Wired estando disponível a 59\$. Este modelo é ligado diretamente à corrente onde a campainha original está, substituindo-a. O campo de visão da câmara é de 155° horizontalmente e 90° verticalmente, com resolução *full high definition* (FHD), movimento customizável e zonas de privacidade.

---

<sup>9</sup> <https://www.lifewire.com/how-ring-doorbell-works-4583925>



## 2.7.Smart Doorbell/Video Intercom System <sup>10</sup> (7)

Este projeto foi desenvolvido como uma forma de facilitar interações sociais mantendo o distanciamento social devido a pandemia de COVID-19, tendo como principais objetivos permitir que o utilizador consiga interagir com quem está à porta remotamente, bem como fazer vídeo chamadas de qualquer lugar a qualquer altura.

É necessário um Raspberry Pi 3 como microcontrolador, um cartão MicroSD, um Raspberry Pi LCD (liquid Crystal display) Screen, uma câmara Raspberry Pi, um microfone USB, um alto-falante STEMMA com amplificador, parafusos, cabos e um conector *Japan solderless terminal* (JST) 3.

Quando alguém toca à campainha é criada uma reunião no Jitsi Meet<sup>11</sup>, sendo o convite dessa reunião enviado para o utilizador via email. Quando o utilizador clica no link recebido este entra na reunião, estando em vídeo chamada com quem está à porta.

Na Figura 5 podemos observar a utilização da campainha.



**Figura 5** Utilização da 7ª Campainha <sup>10</sup>

<sup>10</sup> <https://www.hackster.io/hackershack/smart-doorbell-video-intercom-system-e5aa61>

<sup>11</sup> <https://meet.jit.si/>

## 2.8.DIY Video-Doorbell with ESP32 <sup>12</sup> (8)

Este projeto trata-se de uma campainha que visa ter um melhor desempenho, pois muitas disponíveis no mercado não contam com este fator em consideração. Este foi desenvolvido para ser integrado num sistema de Home Assistant.

Utiliza um ESP32-CAM como microcontrolador, um *piezo buzzer*, um sensor de toque, um PIR e um LCD.

Tal como foi explorado na subsecção 2.5, esta solução utiliza ESPHome em conjunto com Home Assistant como sistema de alertas, pelo que também consideramos que funciona em qualquer lugar com as limitações referidas também na opção anterior mencionada.

Quando alguém toca à campainha uma fotografia é obtida e enviada para o utilizador através do Home Assistant, a partir desta o utilizador pode abrir a porta remotamente, este projeto também tem suporte para modo *delivery* que permite a porta ser aberta automaticamente sempre que alguém toca a campainha.

---

<sup>12</sup> <https://ei23.com/smarthome/awesome-diy-video-doorbell-based-on-an-esp32/>

## 2.9.DIY Doorbell <sup>13</sup> (9)

A DIY Doorbell tem como objetivo criar uma campainha inteligente a nível *low-cost*, fazendo uso do sistema *Home Assistant*.

E necessário um ESP32-CAM como microcontrolador, um sensor de toque e uma câmara.

Tal como explorado anteriormente nas subsecções 2.5 e 2.8, a solução utiliza em ESPHome em junção com Home Assistant como sistema de alertas. Quando a campainha é pressionada é enviada automaticamente uma foto ao utilizador, sendo-lhe disponível uma opção para abrir remotamente a porta e assistir ao uma *live stream*. Para a abertura da porta é necessária uma integração com o Home Assistant em conjunto com um modulo de abertura que funcione com este sistema.

---

<sup>13</sup> [https://youtu.be/6zoaxnAxR\\_c](https://youtu.be/6zoaxnAxR_c)

## 2.10. DIY Ring-Doorbell <sup>14</sup> (10)

Este projeto pretende criar uma campainha *low-cost* e DIY que permita ver uma *live stream* da câmara integrada na campainha, bem como reconhecimento facial, sendo possível abrir a porta remotamente.

Foi utilizado um Raspberry Pi 4 como microcontrolador, um cartão MicroSD, uma câmara, um servomotor e a caixa para a campainha.

Esta solução implementa uma aplicação *web* usando Flask que se encontra hospedado no Raspberry Pi, este sistema faz uso constante da câmara para detetar movimento e posteriormente reconhecer se uma pessoa está à porta, se a pessoa for reconhecida e autorizada este abre a porta automaticamente, sendo também possível aceder a *stream* e destrancar a porta remotamente.

---

<sup>14</sup> [https://youtu.be/9bJFWIVm\\_Fo](https://youtu.be/9bJFWIVm_Fo)

## 2.11. Comparações dos Trabalhos Relacionados

De seguida encontra-se a Tabela 1 que compara as características dos diversos projetos explorados com o projeto desenvolvido, bem como as tecnologias utilizadas.

‘Alertas’ -> Indica o método de envio de alertas utilizado.

‘Comercial/DIY’ -> Indica se o sistema foi desenvolvido para uso comercial ou é DIY.

Nas próximas colunas a serem mencionadas a cor **Verde** significa que é uma vantagem e a cor **Vermelha** uma desvantagem:

- ‘Reconhecimento facial’ -> Indica se a solução tem suporte a reconhecimento facial onde seja possível identificar a pessoa que está na campanha
- ‘Guarda capturas’ -> Indica se o projeto guarda as diversas capturas tiradas durante a utilização para consultas futuras.
- ‘Video chamada’ -> Permite vídeo chamadas através da campanha.
- ‘Deteta movimento’ -> Capacidade de detetar movimento.
- ‘Abrir porta’ -> Indica se a porta pode ser aberta remotamente.
- ‘Low-cost’ -> Se o preço foi considerado como sendo de baixo custo (abaixo de 50€).
- ‘Mensalidade’ -> Se o funcionamento necessita de uma subscrição.
- ‘Bateria’ -> Indica se o sistema tem suporte para uso de bateria.
- ‘Anytime Anywhere’ -> Tecnologia que permite o utilizador aceder a campanha de qualquer lugar a qualquer hora.

**Tabela 1** – Comparação das várias soluções descritas anteriormente.

	Alertas	Reconhecimento Facial	Guarda Capturas	Video Chamadas	Deteta Movimento	Abrir Porta	Low-Cost	Comercial/DIY	Mensalidade	Bateria	Anytime Anywhere
1 (2.1)	Email/SMS	Não	Sim	Planeado	Planeado	Não	Sim	DIY	Não	Não	LAN 1*
2 (2.2)	Email	Sim	Sim	Não	Sim	Sim	Não	Comercial	Não	Não	LAN 1*
3 (2.3)	App	Não	Não	Não	Não	Sim	Sim	DIY	Não	Não	LAN 1*
4 (2.4)	App	Não	Sim	Não	Sim	Sim	Sim	DIY	Não	Não	LAN 1*
5 (2.5)	App	Não	Sim	Não	Não	Sim	Sim	DIY	Não	Não	LAN 1*
6 (2.6)	App	Sim	Sim	Sim	Sim	Sim	Não	Comercial	Sim	Não	Cloud
7 (2.7)	Email	Sim	Sim	Sim	Não	Sim	Sim	DIY	Não	Não	Cloud
8 (2.8)	App	Sim	Sim	Não	Sim	Sim	Sim	DIY	Não	Não	LAN 1*
9 (2.9)	App	Não	Não	Não	Não	Sim	Sim	DIY	Não	Sim	LAN 1*
10 (2.10)	---	Sim	Não	Não	Não	Sim	Sim	DIY	Não	Não	LAN 1*
11 (Solução proposta)	Email	Planeado	Sim	Planeado	Sim	Sim	Sim	Por definir	Não	Por desenvolver	Cloud 2*

1\* Apenas funciona em *Local Area Network*(LAN) não sendo possível migrar para a *cloud*, sendo necessário uma VPN ou *port forwarding* para acesso fora da LAN.

2\* Pode ser hospedado pelo utilizador (LAN) ou na *cloud*.

## 2.12. Síntese

Foram exploradas diversas abordagens com variados métodos para alertar o utilizador, mas com um sistema de campanha base semelhante, assim, torna-se claro que devemos optar por uma abordagem que engloba as mais valias dos diversos projetos no que toca à estrutura da campanha, mas expandir as funcionalidades oferecidas pelo *backend* de modo a que a nossa solução se distinga das outras.

### 3. Proposta de Solução

Neste capítulo é apresentada a solução proposta, explicando as responsabilidades de cada componente, fazendo recurso a esquemas e figuras para ajudar a ilustrar o seu funcionamento.

Também serão descritos os requisitos funcionais e não funcionais.

#### 3.1.Descrição geral da Solução

Pretendemos desenvolver um sistema com quatro módulos: utilizador (com acesso *web*), visitante (quem toca a campainha), campainha (também referenciada como *doorbell*) e uma plataforma na *cloud* (*socket* e *web*). (Figura 6)

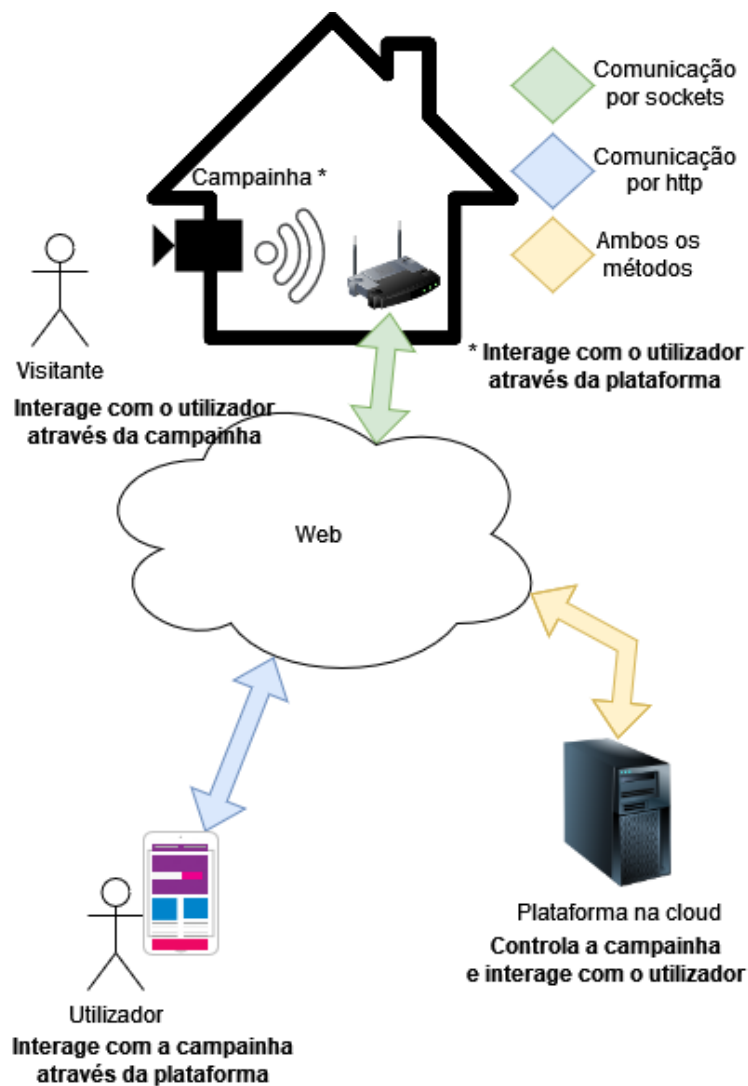


Figura 6 Arquitetura do Projeto



### 3.2.Requisitos

Pretendemos que, através da plataforma disponível *online*, o utilizador consiga interagir com as suas *doorbells*, podendo assim abrir a sua porta remotamente tal como assistir a uma *live stream*, de modo a alcançar este objetivo, planeamos que a *doorbell* a desenvolver tenha a capacidade de se conectar à internet, conseguindo assim comunicar com a plataforma disponibilizada.

A *doorbell* deve ainda ter a capacidade de tirar fotos e/ou gravar vídeos além de abrir a porta para possíveis visitantes, acrescentado a isto deve ainda poder enviar um alerta caso alguém pressione o botão da mesma ou caso seja detetado movimento.

A plataforma a ser desenvolvida deve permitir que um utilizador consiga:

- Abrir a porta remotamente
- Pedir e receber uma imagem/vídeo capturado pela *doorbell* em tempo real.
- Visualizar as captura antigas de uma *doorbell*, devendo assim a plataforma guardar todas as capturas recebidas das *doorbells*.
- Fazer uma vídeo chamada com quem está a tocar a campainha.
- Reconhecimento facial quando alguém toca a campainha, devendo identificar a pessoa e posteriormente uma ação pré-configurada pelo utilizar.
- Permitir o utilizador receber um alerta a qualquer momento.
- Apresentar dados estatísticos sobre a atividade detetada, realçando dias onde houve maior atividade do que a esperada (podendo indicar um risco de segurança).

Além das funcionalidades propostas foi posteriormente sugerido o desenvolvimento de um sistema onde um utilizador seja possibilitado de ter várias *doorbells* em simultâneo e que esta plataforma também suporte múltiplos utilizadores.

### 3.3.Síntese

Neste capítulo foi abordado o nosso planeamento para desenvolver uma solução ao desafio proposto, tendo sido explorado o funcionamento pretendido de cada parte do sistema, bem como as suas interações com os utilizadores.

Uma plataforma deverá ser desenvolvida, permitindo que os utilizadores interajam com as suas *doorbells* de modo remoto, bem como que tenham acesso a dados recolhidos pelas mesmas e a capacidade de alterar o seu comportamento.

A campanha a ser desenvolvida deve conseguir conectar-se à internet, e, através dela, à plataforma. Deve ainda conseguir abrir a porta, detetar movimento e capturar imagens e vídeos.

## 4. Implementação

Neste capítulo é apresentada a implementação da solução proposta bem como as decisões tomadas durante o desenvolvimento até obtermos o produto final e as alternativas consideradas.

### 4.1. Campanha desenvolvida

Iremos fazer uma breve explicação do material utilizado e o porque dessas escolhas.

#### 4.1.1. Microcontrolador (ESP32-CAM)

O microcontrolador deve ter a capacidade de se conectar à internet por Wi-Fi, bem como suporte a uma câmara e capacidade para acrescentar sensores para alcançar as funcionalidades previstas no capítulo anterior. Este deve ter também um custo reduzido visto que o projeto é *low-cost*.

Apesar de estar inicialmente planeado ser utilizado um ESP32-CAM, após a exploração dos projetos semelhantes que foram descritos no capítulo 2 concluímos que utilizar um Raspberry Pi 2/ 3 também seria uma opção viável. Uma vez que tanto o ESP32 como o Raspberry Pi têm a capacidade de se conectar à internet e de utilizarem câmaras entre outros componentes necessários, a escolha entre estes ficou reduzida a uma comparação entre preços e desempenho. O Raspberry Pi tem um desempenho consideravelmente melhor, no entanto também tem um preço mais elevado (7 a 8 vezes mais caro do que um ESP32). Uma vez que este projeto tem como objetivo ser *low-cost* optamos por prosseguir com a opção mais barata e tomar possíveis restrições a nível de desempenho como um desafio, elegendo assim o ESP32-CAM como o nosso microcontrolador.

#### 4.1.2. Câmara

Existem vários modelos de câmara que podem ser utilizados num ESP32-CAM, o modelo com o qual nos trabalhamos foi o Ov2640, sendo este o único que captura uma imagem verticalmente, pelo que foi necessário aplicar uma rotação de 90° às capturas, com isto em mente o *backend* fica responsável pela tal rotação de imagem.

#### 4.1.3. Sensor de movimento

Para que a *doorbell* desenvolvida neste projeto consiga detetar movimento seria necessária uma constante captura de imagem tal como uma análise a está, mas este tipo de abordagem

poderia causar um grande impacto o desempenho e possivelmente no tempo de vida do produto. Tendo como base esta problemática, optamos por utilizar um PIR, que deteta movimento de forma passiva através de infravermelhos (IR), isto é, o sensor deteta radiação IR que é muito próxima ao tipo de radiação emitida pela temperatura emitida por animais de sangue quente.

#### **4.1.4. Botão**

Foi utilizado um botão de pressionar para que seja possível acionar a *doorbell*, criando assim o efeito tradicional de tocar à campainha.

#### **4.1.5. Relê**

De modo que os utilizadores consigam abrir a porta remotamente foi necessário incluir um relê, que fica responsável por trancar/destrancar a porta. Foi pensado o uso de um servomotor ou um relê, mas chegou-se à conclusão que um relê seria mais prático pois dá para implementar em diversas portas com fechaduras eletrônicas existentes.

## 4.2. Tecnologias utilizadas

Para o desenvolvimento do *backend* do projeto foi decidido utilizar Python<sup>15</sup> devido à sua facilidade de escrita além de que tínhamos conhecimento que este suporta tanto conexões *socket* para comunicação com a *doorbell*, tal como a existência de diversas *frameworks* para web. A *framework web* que decidimos utilizar foi o Flask<sup>16</sup> pois esta aparenta ter bastante documentação e era algo novo em que nenhum de nos tinha utilizado anteriormente, tendo sido interessante explorar o seu funcionamento, o Flask é baseado em Werkzeug<sup>17</sup> e em Jinja<sup>18</sup>.

### 4.2.1. Python Virtual Environments

É comum que diferentes projetos em Python necessitem de diferentes versões da mesma biblioteca, de modo a alcançar a compatibilidade entre as diversas bibliotecas instalados, é usado um *virtual environment* quando existem múltiplos projetos na mesma máquina. O uso deste último permite que cada projeto individual tenha apenas as bibliotecas necessárias com a respetiva versão compatível. Uma vez que não foi necessário utilizar este sistema, acabou por não ser implementado no projeto desenvolvido.

### 4.2.2. Flask

Flask é um *micro web framework* criado em Python. Este não contém uma camada de abstração do banco de dados, verificação de formulários, ou qualquer outro componente em que as bibliotecas de terceiros fornecem funções comuns. Sendo assim o Flask está desenhado para quem começa um projeto de raiz consiga rápida e facilmente implementar algo básico, mas mantendo a sua escalabilidade necessária para o desenvolvimento de projetos mais complexos. Podemos então afirmar que o Flask se distingue das alternativas pelo seu baixo nível de entrada (sem perder escalabilidade) e pelo nível de customização oferecido.

### 4.2.3. Werkzeug

Werkzeug é uma WSGI *toolkit* como tal tem suporte a vários pedidos *hypertext transfer protocol* (HTTP), suportando pedidos, resposta e funções de utilidade. Isto permite que uma *framework web* seja construída sobre esta biblioteca tal como aconteceu com o Flask que faz uma implementação direta desta internamente. Como o Flask foi construído sobre esta

---

<sup>15</sup> <https://www.python.org/>

<sup>16</sup> <https://flask.palletsprojects.com/en/2.1.x/>

<sup>17</sup> <https://werkzeug.palletsprojects.com/en/2.1.x/>

<sup>18</sup> <https://jinja.palletsprojects.com/en/3.1.x/>

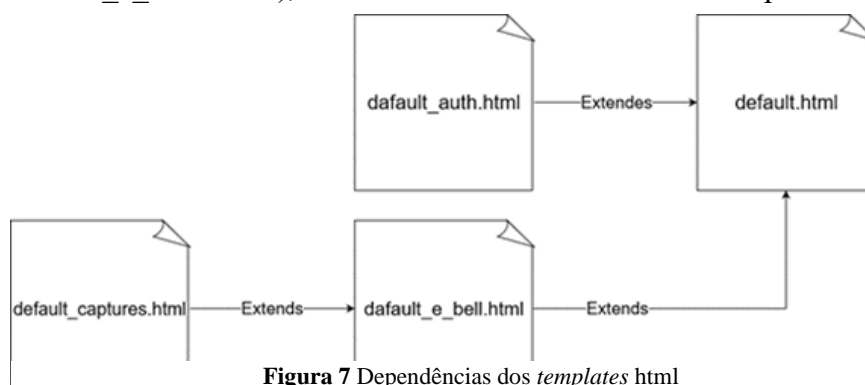
ferramenta, quando o Flask entra em modo de hospedagem ele usa o Werkzeug também na sua hospedagem.

#### 4.2.4. Jinja

Jinja é um *web template engine* popular para Python, este tipo de tecnologia combina um *template web* como uma fonte de dados específica para renderizar uma página *web* dinamicamente do lado do servidor (sendo esta renderização a nível de *backend*, qualquer alteração a fonte de dados depois da renderização não será aplicada).

Através do uso de Jinja foi possível passar variáveis do servidor *web* para as páginas *hypertext markup language* (HTML) o que facilitou a construção dinâmica das mesmas com o uso de operações como declarações “*for*” e “*if*”. Permitiu ainda o desenvolvimento de quatro *templates* padrão dos quais as páginas herdam as características. (Figura 7)

- O ficheiro “default.html” tem o formato base de todas as páginas devem seguir(*layout*), sendo as suas características herdadas por todas as páginas incluído os outros *templates*.
- O ficheiro “default\_auth.html” que tem o formato base das páginas relacionados com a autenticação do usuário, sendo estas o login e o registo, contento o *layout* padrão para este tipo de páginas.
- O ficheiro “default\_e\_bell.html” define a estrutura base para as restantes páginas que não estão associadas com a autenticação, esta página adiciona algumas características sobre o *layout* que faltam no “default.html”, esta página também exporta as diversas *scripts* JavaScript necessárias para o funcionamento e por fim implementa a demonstração de novos alertas em tempo real.
- O ficheiro “default\_captures.html” herda o ficheiro referido anteriormente (“default\_e\_bell.html”), este contem a estrutura básica para as páginas que



**Figura 7** Dependências dos *templates* html

tenham capturas, tal como o JavaScript para a atualização automática das novas capturas.

#### 4.2.5. Flask-Mail

Com a necessidade de envio de alertas para o utilizador, foi decidido que isto iria ser feito através do email, para tal e após uma pesquisa breve, foi descoberta uma biblioteca chamada Flask-Mail, sendo esta uma implementação para o Flask ao qual é usado um servidor email usado por terceiros para enviar um email. Como este serviço foi usado com Flask é possível enviar um email contendo uma ‘mini’ página *web* embutida, estando assim disponível o uso de Jinja durante a renderização da mesma.

#### 4.2.6. JWT<sup>19</sup>

Após a criação dos alertas via email, foi pensando em enviar um *link* para o utilizador poder aceder diretamente ao alerta recebido na página, este tipo de *link* iria necessitar de um *token* para o utilizador se autenticar automaticamente. Com isto em mente foi usada uma biblioteca chamada PyJWT, que contém funcionalidades básicas para um *json web token* (JWT).

Apesar de não ter sido implementado nos alertas o tal sistema *token*, este foi usado para autenticação interna do utilizador enquanto este navega na nossa página.

Um JWT apresenta algumas vantagens em relação a um *token* normal pois este tem capacidade de conter um *payload* de dados que são criptografados, podem ser assinados através de um certificado e também permite introduzir um limite de vida para este *token*, com estas funcionalidades existem diversas vantagens a nível de segurança em utilizar este mecanismo.

Existem duas bibliotecas parecidas para o JWT, uma delas é chamada de ‘PyJWT’ e a outra apenas de ‘JWT’, caso ambas estas bibliotecas tenham sido instaladas durante o seu importe para Python existe uma ambiguidade em qual é importada.

#### 4.2.7. OpenCV<sup>20</sup>

Foi utilizada a biblioteca OpenCV para algumas alterações às imagens captadas pela câmara que posteriormente foram enviadas ao servidor, esta biblioteca também fornece um

---

<sup>19</sup> <https://jwt.io/>

<sup>20</sup> <https://opencv.org/>

subcomponente chamado de ‘*VideoWriter*’ que permite criar um vídeo a partir de diversos *frames*.

OpenCV também é bastante conhecido e usado para computação visual em tempo real, o que teria sido a base para o reconhecimento facial. Uma outra vantagem é que apesar de ser muito completa é extremamente rápida quando comparada a outras possíveis alternativas (a nível de tratamento de imagem).

#### **4.2.8. Página Web**

A plataforma *web* disponibiliza várias páginas para os utilizadores poderem visualizar os diversos dados recolhidos pelas *doorbells*, bem como *live stream* das mesmas e gerenciar as configurações de alertas.

Quando o utilizador entra pela primeira vez na página *web* este é redirecionado para uma zona de ‘Login’, sendo-lhe dado também uma opção para Registar, após o ‘Login’ ou ‘Registo’ do utilizador, este é redirecionado para o ‘Manage Doorbells’ que mostra todas as *doorbells* associadas tal como algumas informações básicas sobre cada uma.

As atuais páginas existentes são: ‘Manage Doorbells’, ‘All Streams’, ‘Captures’ e ‘Alerts’.

‘Manage Doorbells’ permite ver todas as *doorbells* que estão registadas á conta do utilizador, bem como a última foto tirada por esta e o seu estado atual, esta página é a primeira a ser apresentada após o ‘Login’/‘Registo’.

‘All Streams’ disponibiliza uma vista com as diversas lives de todas as *doorbells* associadas ao utilizador, semelhante aos sistemas *closed-circuit television* (CCTV) com diversas câmaras onde é possível ver todas ao mesmo tempo

‘Captures’ é a página que mostra todas as capturas de todas as *doorbells* do utilizador, contendo uma breve descrição da ação que criou a captura e ainda a data de quando esta foi tirada. Ao clicar numa captura esta é ampliada para melhor visualização da parte do utilizador.

‘Alerts’ permite visualizar todos os alertas do sistema, estes podem ser anúncios do servidor, registo de uma nova *doorbell*, movimento detetado ou quando alguém toca à campainha. Quando algum dos alertas mencionados anteriormente acontece é enviado um email ao utilizador sobre esse alerta com um *link* anexado para a página de alertas.



Existe ainda mais uma página que está associada a cada *doorbell* individualmente, esta apresenta uma *live stream* da câmara, uma opção para guardar a imagem atual da câmara, abrir a porta e mudar as configurações básicas como nome da *doorbell* e emails de alertas associados a mesma (sendo necessário confirmar a *password*).

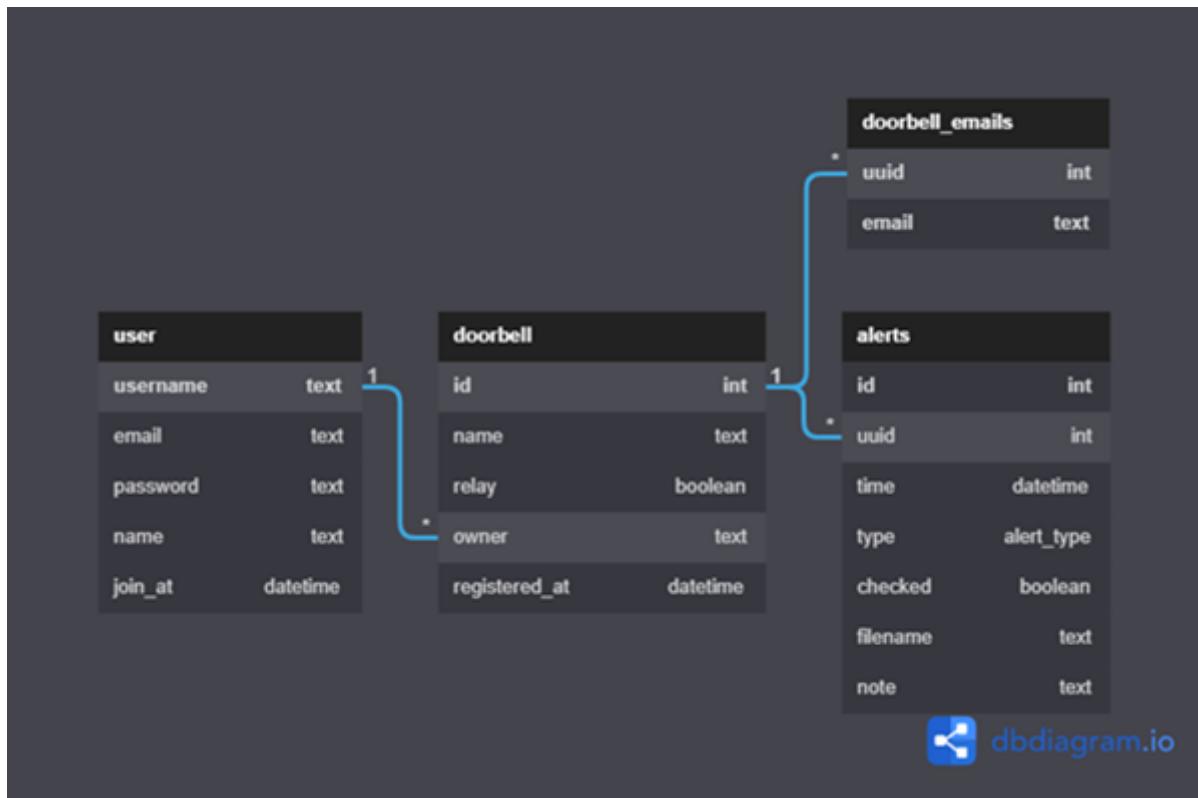
De modo que utilizadores recebam o alerta em tempo real durante a utilização da página *web* foi adicionado um ícone em formato de sino na *navbar*, este mostra o número de alertas que ainda não foram vistos, quando o utilizador clica no ícone abre um menu *dropdown* mostrando uma pequena mensagem sobre os alertas.

Para evitar usarmos mais um servidor decidimos não utilizar WebSocket entre a página *web* e a backend, em vez disso o JavaScript faz um pedido de 5 em 5 segundos para atualização de informações mostradas na página (Exemplo: número de alertas).

#### **4.2.9. Base de Dados**

De modo a minimizar os recursos necessários para operar o *backend* optamos por utilizar o SQLite como base de dados. SQLite distingue-se de outras alternativas por ter um melhor desempenho em termos de velocidade de operações de escrita e leitura, só carregando os dados necessários em memória e tem ainda como mais-valia o facto de a escrita no ficheiro ser incremental, o que, em caso de falha, a informação perdida é minimizada.

A base de dados é composta por 4 tabelas como podemos observar pela Figura 8.



**Figura 8** Esquema da base de dados

A tabela ‘user’ é responsável por guardar as informações básicas do utilizador, tais como informação de login, email, o nome e a data de registo.

A tabela ‘doorbell’ guarda as informações relativas a cada *doorbell*, *universally unique identifier* (UUID) representado por ‘id’ na tabela, o nome que por padrão é igual ao uuid, se contém relê ou não (indicado durante o registo), o dono da *doorbell* que corresponde a um utilizador e a data em que foi registada.

A tabela ‘doorbell\_emails’ guarda os emails a que cada *doorbell* vai enviar um alerta, estando como padrão o email de registo do utilizador.

Por fim a tabela ‘alerts’ guarda informações que correspondem a alertas e capturas, esta tem o campo ‘id’ que é uma chave primaria incremental, ‘uuid’ da *doorbell* a que esta associado, ‘time’ que indica a data e hora ao qual o ocorreu a ação (captura ou alerta), ‘type’ para guardar o tipo de alerta que é representado (Descrito no próximo paragrafo), ‘checked’ que indica se a foi vista ou não, o ‘filename’ que corresponde ao um ficheiro associado caso exista e por fim ‘note’ que são notas ou uma mensagem extra que é guardada caso necessário.

Tipo de alertas: System (1), NewBell (2), Bell (3), Movement (4), UserPicture (5) e OpenDoor (6). ‘System’ corresponde a uma mensagem do sistema, apesar de no projeto não ter sido usado decidimos criá-la caso seja necessário futuramente, ‘NewBell’ indica que uma nova *doorbell* foi registada, ‘Bell’ sinaliza que alguém pressionou na campainha, caso seja detetado movimento este é representado por ‘Movement’, a pedido do utilizador temos o ‘UserPicture’ quando este pede uma foto e quando abre a porta é registado como ‘OpenDoor’.

#### 4.2.10. Serviço de Notificações via email

Inicialmente foi pensado utilizar os serviços do Gmail para enviar emails automaticamente, mas a Google alterou o método de autenticação para aumentar a segurança das suas contas (30 de Maio de 2022), então o login da conta Gmail passa a ser feito através de uma *password* de aplicativo, isto é, uma *password* que é gerada automaticamente pela entidade em que permite login na conta e ao qual deve ser criada uma nova *password* para cada aplicação, estas *passwords* podem ser revogadas a qualquer momento não comprometendo assim a segurança da conta e não revelando a *password* principal a outras aplicações. Porém para ativar este mecanismo também é necessário ativar o *two-factor authentication* (2FA) na conta usada.

Infelizmente por padrão os emails recebidos pelo nosso sistema de notificações são considerados como *spam*, assim sendo vão para ao email de lixo, isto implica que as imagens não serão carregadas automaticamente nem o user irá receber notificações na sua *app* do email, como tal o utilizador tem de marcar a origem como legítima/segura.

Email oficial: <mailto:ebellsalertsystem@gmail.com>

#### 4.2.11. Doorbell WiFiManager<sup>21</sup>

Durante o desenvolvimento do código para o ESP32 foi utilizado uma biblioteca chamada WiFiManager cujo seu objetivo é facilitar a configuração do ESP32 para se conectar a um *access point* (AP), gerido através de um *hotspot* e usando *captive portal*. Esta biblioteca também permite pedir alguns *inputs* de configuração ao utilizador.

---

<sup>21</sup> <https://github.com/tzapu/WiFiManager>

#### 4.2.12. Backend base

O *backend* desenvolvido é constituído por um servidor *socket*, um servidor *web* e uma base de dados, onde os servidores correm simultaneamente em *threads* diferentes, comunicando entre eles apenas quando necessário, fazendo também recurso à base de dados por ambos (*socket* e *web*).

Durante o arranque o servidor vai correr com a *thread* principal, ao qual cria uma segunda *thread* que irá ficar responsável pelo servidor *socket*, após a criação da segunda *thread* a principal irá iniciar o Flask e fica responsável pelos pedidos HTTP.

#### 4.2.13. Socket

O servidor *socket* é um servidor *transmission control protocol* (TCP) que é responsável por aceitar novos pedidos de conexão enviados por clientes(*doorbells*) a tentarem-se conectar via TCP. Sempre que uma cliente se conecta e é aceite pelo servidor, este cria uma *thread* que fica exclusivamente associada àquela cliente, fazendo a leitura dos dados recebidos pela conexão TCP e posteriormente escrita para a mesma conexão. Após a criação desta nova *thread* o servidor envia um pedido ao cliente para esta se identificar através do endereço *media access control* (MAC) também referido como UUID.

O servidor aguarda uma resposta por parte do cliente, verificando se este está registado na base de dados, sendo-lhe devolvido uma confirmação. Caso a *doorbell* já se encontre registada, esta encontra-se pronta a ser utilizada, mas se este não for o caso, o cliente entrará em modo *hotspot* para este inserir o seu username ao qual irá ficar associada a *doorbell*.

Os dados de registo (username) são verificados com os utilizadores existentes na base de dados sendo sempre enviada uma resposta indicando se o cliente foi registado ou se o utilizador não existe.

A utilização de uma *thread* para cada *doorbell* surge da necessidade de executar diversos ciclos para leitura dos dados recebidos para comunicação, após uma breve avaliação, concluímos que o uso de apenas uma *thread* para as diversas *doorbells* podia criar problemas futuros em relação a escalabilidade do projeto.

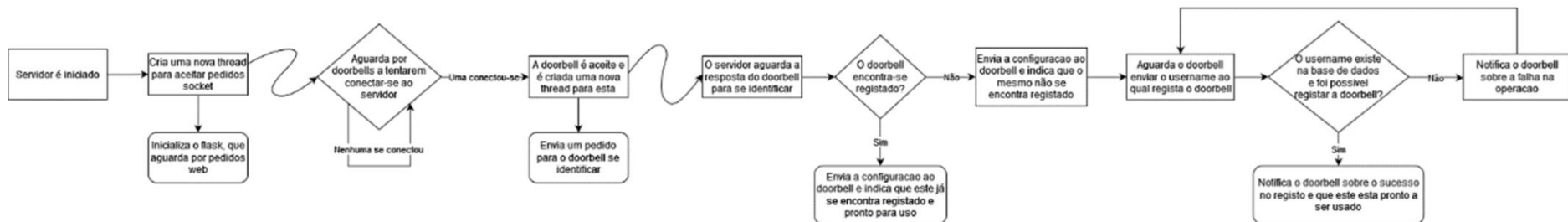
#### 4.2.14. Stream

Para obter um resultado com efeito de *live stream* foi concluído que é necessário existir uma comunicação constante entre a *doorbell* e a plataforma, então quando esta *live stream* é pedida pelo utilizador na página *web*, o servidor envia à *doorbell* um pedido para este começar a enviar os vários *frames* capturados pela câmara, este envio de frames ocorre a 20fps o que corresponde a 50ms, para tal é usado uma função de espera do lado do *doorbell* apesar de estar definida para 50ms caso tenha ocorrido um atraso no envio de dados para o *socket* este atraso é descontado ( $50 - \text{Atraso} = \text{Valor}$  ou 0 caso a subtração seja negativa).

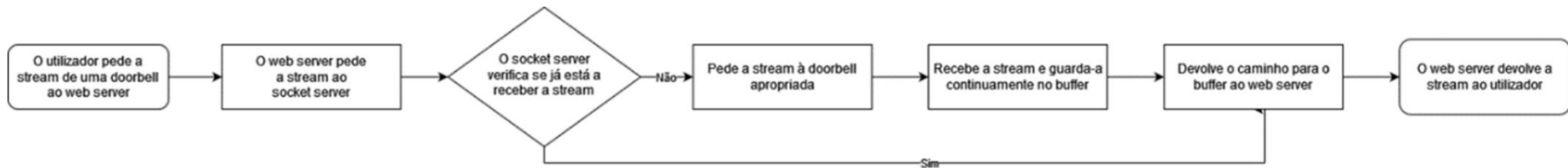
Quando o servidor *socket* recebe os dados da *frame*, este guarda numa variável temporariamente que é partilhada entre as várias *threads*, assim o Flask consegue aceder ao que o utilizador enviou para o *socket* e mostrá-lo ao utilizador.

Na Figura 9 iremos mostrar um esquema completo sobre o funcionamento do *backend*, desde o momento em que o servidor é iniciado até que a *doorbell* esteja pronta a ser utilizada.

Na Figura 10 demonstração do funcionamento do servidor quando um utilizador quer assistir uma *live stream*.



**Figura 9** Esquema completo sobre o funcionamento do backend



**Figura 10** Esquema para comunicação da Stream.

#### 4.2.15. FCGI

O fast common gateway interface (FCGI) é um protocolo binário desenvolvido para reduzir a sobrecarga relacionada à interface entre o servidor *web* e o programa common gateway interface (CGI), permitindo uma maior quantidade de pedidos por unidade de tempo.

#### 4.2.16. Plataforma na cloud

De modo que seja possível utilizar o serviço de forma *Anytime Anywhere* existiu uma necessidade de hospedar a plataforma na *cloud*.

A plataforma *cloud* que decidimos utilizar foi a *Amazon Web Services* (AWS) pois esta disponibiliza *free tiers* durante um ano, assim sendo não existiram quaisquer custos durante o desenvolvimento e *deployment* do projeto na *cloud*.

A máquina na *cloud* tem de ser capaz de correr Python 3.7 (versão mínima do projeto), também é recomendado o uso de um servidor *web* (desde que suporte FCGI) como por exemplo Apache ou Lighttp para um melhor desempenho.

No caso da *cloud* em vez de utilizarmos o padrão de hospedagem do Flask (Werkzeug), foi utilizado o FCGI que se conecta diretamente de forma binária ao Lighttp, no caso de termos usado o Werkzeug teríamos de usar um *proxy* em vez de FCGI.

No caso de ser utilizado o Lighttpd é necessário mudar uma configuração que não vem ativa por padrão para permitir *live streams*, caso não seja feito nenhuma *live stream* irá funcionar, a configuração mencionada é ‘server.stream-response-body = 2’ está precisa de ser adicionado ao ficheiro de configuração que em sistemas Linux encontra-se nesta localização ‘/etc/lighttpd/lighttpd.conf’.

Como neste projeto foi usado FCGI é preciso o configurar (Figura 11)

```
#FastCGI Conf
server.modules += ( "mod_fastcgi" )

fastcgi.server += ( "/proj" =>
    (
        "bin-path" => server_root + "/esp32cam/server.fcgi",
        "socket" => "/tmp/fcgi.socket",
        "max-procs" => 1,
        "check-local" => "disable",
    )
)
```

Figura 11- Configuração do FCGI na Cloud

Quando o utilizador acede a página *web* este é redirecionado para uma subsecção ‘/proj’ (Exemplo: <http://pagina.com/proj>) pois o FCGI não pode ser ativo no espaço padrão (‘/’), ao iniciar o Lighttpd este vai procurar pelo ‘bin-path’ onde se encontra o binário que vai iniciar a plataforma, por questões de segurança ‘check-local’ foi desativado pois caso esta opção tivesse ativa a *web* podia sofrer ataques direcionadas à leitura de ficheiros não autorizada pois teria acesso ao mapeamento onde os ficheiros da plataforma se encontram.

Para garantir uma maior segurança à página desenvolvida foi também adicionado suporte ao HTTP secure (HTTPS), utilizando o serviço Let’s Encrypt <sup>22</sup> que permite obter um certificado que é validado por uma entidade autorizada de forma grátis. A duração de este certificado disponível de forma gratuita é de 90 dias, mas este também pode ser renovado de forma completamente gratuita.

Os recursos utilizados pela VM vão depender maioritariamente do número de *doorbells* em simultâneos conectados. Como apenas foram realizados testes com até duas *doorbells* não foi possível ter uma noção da quantidade de recursos necessários, mas foi determinado que para apenas duas *doorbells* o *free tier* da AWS é mais do que suficiente, com esta informação em mente é possível que não sejam necessários muitos recursos.

### 4.3. Configuração da plataforma

Para o bom funcionamento da plataforma esta precisa de ser configurada, no diretório raiz da mesma contém uma pasta chamada ‘web’ e dentro dessa pasta é necessário existir um ficheiro chamado ‘flask.cfg’ que é carregado quando a plataforma inicializa.

O ficheiro necessita de algumas configurações obrigatórias tais como:

- ESP\_FILES\_DIR -> Caminho para a pasta onde irão ser guardadas as capturas.
- MAIL\_SERVER -> Endereço do servidor de email.
- MAIL\_PORT -> Porta do servidor de email.
- MAIL\_USERNAME -> *Username* para login no servidor de email.
- MAIL\_PASSWORD -> *Password* para login no servidor de email.
- MAIL\_USE\_TLS -> Se a conexão deve utilizar *transport layer security* (TLS). Esta opção vai depender do servidor de email.

---

<sup>22</sup> <https://letsencrypt.org/>



- MAIL\_USE\_SSL -> Se a conexão deve utilizar *secure sockets layer* (SSL). Esta opção vai depender do servidor de email.
- JWT\_SECRET\_KEY -> Chave secreta utilizada pelo JWT ao encriptar.
- SCHEMA\_FILE -> Ficheiro que contem o esquema da base de dados em formato SQLite.
- SECRET\_KEY -> Chave secreta utilizada pelo Flask para encriptar as *sessions*.
- SERVER\_NAME -> Endereço para o servidor, pode ser por endereço *internet protocol* (IP) ou *domain name system* (DNS), este é necessário para pode enviar emails com um *link* direto para o *website*.

Existem mais configurações, mas são opcionais, a lista completa de configurações pode ser encontrada aqui<sup>23</sup>.

A Figura 12 é um exemplo de um ficheiro de configuração para a plataforma.

```
DEBUG = False
ENV = 'deployment'
ESP_FILES_DIR = 'esp_files'
MAIL_SERVER = 'smtp.gmail.com'
MAIL_USERNAME = 'EBellsAlertSystem@gmail.com'
MAIL_PASSWORD = 'SUPER SECRET'
MAIL_PORT = 465
MAIL_USE_TLS = False
MAIL_USE_SSL = True
MAIL_DEFAULT_SENDER = 'E-Bells Alert System'
JWT_SECRET_KEY = 'SUPER SECRET'
RANDOM_SECRET_KEY = False
SCHEMA_FILE = 'schema.sql'
SECRET_KEY = 'SUPER SECRET'
SERVER_NAME = 'proj47-ip1-2022.duckdns.org'
SESSION_COOKIE_SECURE = True
```

**Figura 12** – Exemplo de configuração da plataforma

<sup>23</sup> <https://flask.palletsprojects.com/en/2.1.x/config/>

#### 4.4. Inicialização e configuração da Doorbell

Explicação passo a passo como configurar a *doorbell*.

1. O utilizador liga a *doorbell*, esta verifica a sua conexão à internet, caso não seja possível, a *doorbell* entra automaticamente em modo *hotspot*.
2. O utilizador conecta-se ao *hotspot* da *doorbell* e introduz os dados de acesso a um AP para esta se ligar à rede. (Figura 13)
3. A *doorbell* tenta-se conectar ao AP e estabelecer comunicação com o servidor *socket*, caso falhe volta a entrar em modo *hotspot*. (Figura 14)
4. Após a conexão ser estabelecida é verificado se o *doorbell* se encontra registada, caso esteja ela encontra-se pronta a utilizar. Se não estiver entra em modo *hotspot* pedindo pelos dados do utilizador e se contem um relê.
5. Por fim são enviados os dados ao servidor e aguarda a resposta sobre a validação dos dados. Entra de novo em modo *hotspot* caso falhe.

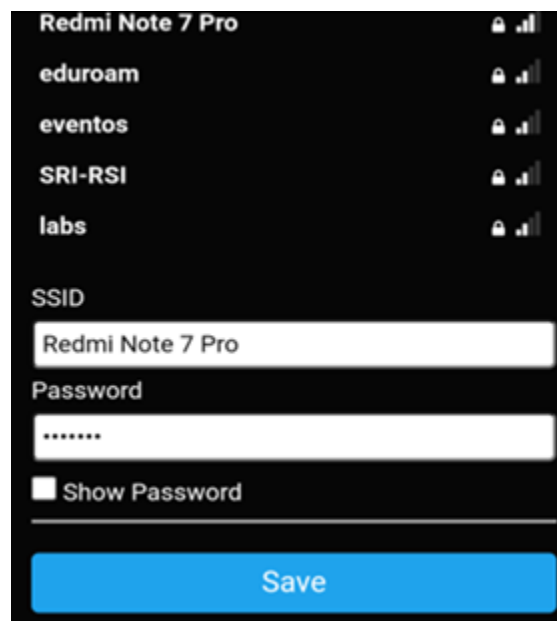
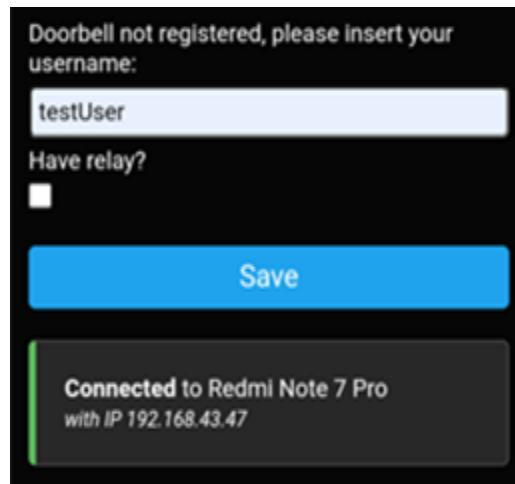


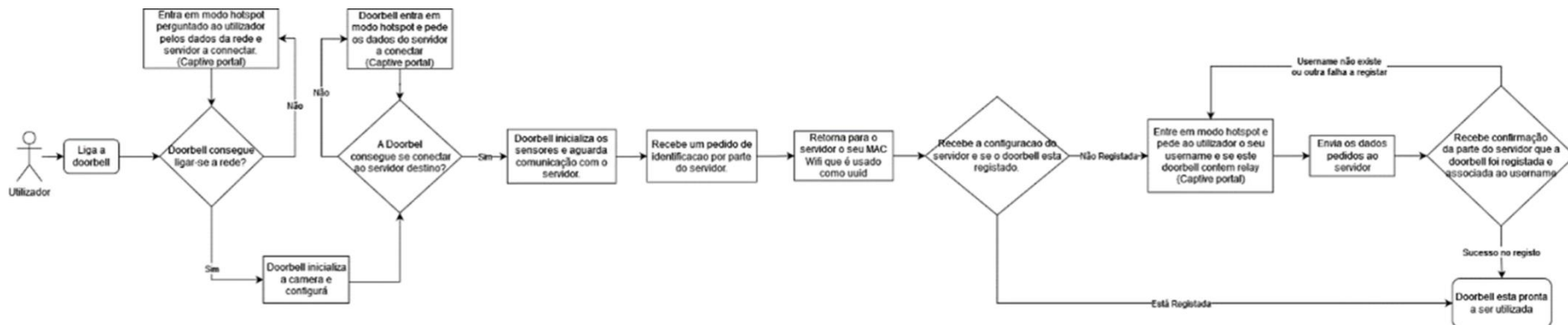
Figura 13 Doorbell modo hotspot Wi-Fi



**Figura 14** Doorbell modo hotspot Hotspot

Quando a *doorbell* é adicionada, esta fica registada com o email da conta do utilizador, sendo este email usado para os alertas.

Na Figura 15 fica o esquema completo da conexão da *Doorbell* ao servidor.



**Figura 15** Esquema completo da conexão da Doorbell ao servidor

## 4.5.Síntese

Neste capítulo foram exploradas as funcionalidades desenvolvidas e as que ficaram por desenvolver, bem como as tecnologias utilizadas, tanto a nível de *software* como a nível de *hardware*, pelo que apos a sua leitura deve ser claro o funcionamento do sistema desenvolvido e como o utilizar.

## 5. Testes e Resultados

Neste capítulo serão apresentados os diversos testes efetuados de modo a verificar o bom funcionamento do sistema desenvolvido, no final do capítulo será feita uma análise aos resultados obtidos.

### 5.1. Testes realizados

Todos os seguintes testes foram realizados através da plataforma disponibilizada pela VM na *cloud* (AWS) no âmbito de verificar o bom funcionamento do serviço *Anytime Anywhere*.

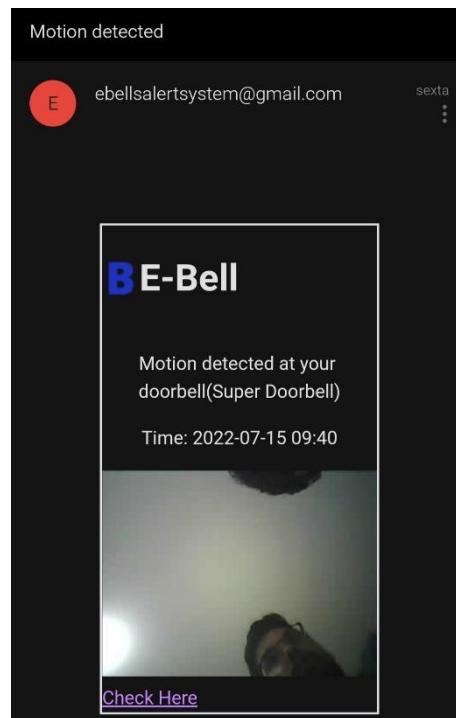
Nota: durante o uso prolongado do ESP32 este parece aquecer tornando-o um pouco mais lento, durante este uso prolongado foram usados 2 ESP's sendo que um deles ficou lento pois aquecia muito, no entanto, o outro ESP32 não parecia sofrer deste problema.

### 5.1.1. Teste às capturas e ao sistema de alertas

Este teste tem como objetivo testar o bom funcionamento do sistema de alertas via email, bem como se a *doorbell* está a detetar movimento e/ou quando o botão é pressionado.

Foram então colocados dois emails na lista de emails a serem alertados na página de uma *doorbell* ativa, no intuito de que ambos os emails recebam os alertas.

Exemplo de um email de aleta recebido na Figura 16



**Figura 16** Email recebido no teste

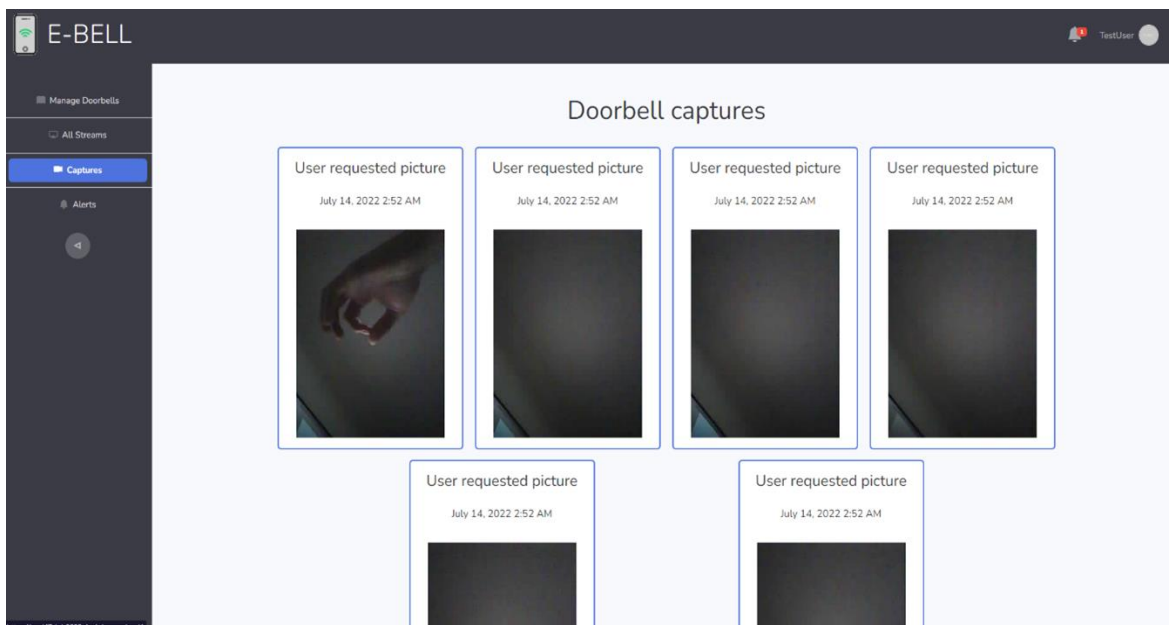
Tanto quando o botão foi premido como quando o movimento foi detetado foram enviados emails para cada email pretendido, estando o sistema a funcionar como esperado. Foi, no entanto, verificado que a gravação de vídeos não estava a funcionar. Após testado a nível LAN concluímos que este problema é exclusivo ao serviço disponível em *cloud*, as possíveis causas podem ser o uso do sistema Linux em vez de Windows por parte da *cloud* ou o uso de FCGI.

### 5.1.2. Teste à base de dados

Este teste tem como objetivo averiguar se as capturas realizadas pelas *doorbells* estão a ser guardadas e exibidas como pretendido.

Foi então comparada a página de capturas em dois momentos diferentes, no início deste teste, e após várias capturas terem sido realizadas a pedido do utilizador.

Página de capturas após o utilizador ter pedido para captura imagens da câmara  
Figura 17.



**Figura 17**– Capturas realizadas no teste

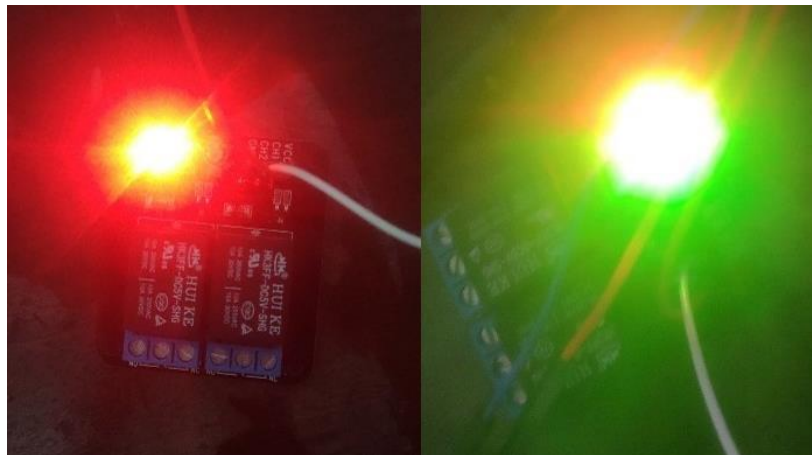
Verificamos que as novas capturas são apresentadas como esta previsto, tendo a sua origem e data sido também verificadas.



### 5.1.3. Teste ao relê

Com este teste pretendemos garantir o bom funcionamento do relê, que está responsável por trancar/destrancar a porta. Foi então realizado o acesso à página de um *doorbell* com um relê conectado, tendo assim o botão para abrir a porta disponível, este último foi pressionado e o comportamento foi verificado.

Figura 18 demonstra o relê a ser aberto a pedido do utilizador.



**Figura 18**– Comportamento do relê durante o teste

Como previsto, foi possível ativar o relê através da página *web*, sendo que o *light-emitting diode* (LED) indicado o estado do relê, vermelha indica fechado e verde indica aberto.

## 5.2. Análise dos resultados obtidos

Uma vez que estas funcionalidades foram testadas repetidamente ao longo do desenvolvimento do projeto, à parte da não gravação de vídeos, já era esperado que estes resultados fossem obtidos. Ainda assim não foi possível identificar a causa do problema na gravação de vídeos (como mencionado anteriormente, suspeita de ser em sistema Linux ou uso de FCGI).

## 5.3. Síntese

Através dos diversos testes realizados foi possível confirmar o funcionamento do sistema desenvolvido, garantindo assim o seu bom funcionamento tal como pretendido. Nos casos de testes todos foram validados com sucesso com exceção da gravação do vídeo.

## 6. Conclusão

Em conclusão, a solução proposta utiliza uma campainha que deteta movimento, tranca/destranca a porta e captura fotos e vídeos, estando conectada à internet via Wi-Fi, comunicando com uma plataforma. As páginas *web* disponibilizadas permitem que os utilizadores tenham acesso às capturas realizadas pelas suas *doorbells* e que visualizem em tempo real as suas *streams*, podendo fazer pedidos de fotografia ou controlar a porta a qualquer momento. Os utilizadores podem alterar os emails de alerta, podendo ser utilizados múltiplos ao mesmo tempo. Deste modo a nossa solução distingue-se da maioria das alternativas em termos de funcionalidades disponíveis aos utilizadores mantendo o preço baixo e fácil configuração. Com isto em mente a nossa solução permite que uma maior percentagem da população passasse a utilizar smart *doorbells*, normalizando-as, dando assim um passo em frente no ajuste da população às tecnologias moderna.

Seria, no entanto, necessário a utilização de mais recursos do que os utilizados atualmente pela VM no servidor da Amazon, pelo que seria necessário cobrar mensalidades. Vale a pena mencionar também que esta VM sofreu múltiplos ataques pelo que seira necessário também implementar mais medidas de segurança à VM. Deveria também ser utilizado um sistema de envio de emails diferente, visto que a utilização da Google como serviço de emails não é recomendado para uma operação de maior magnitude.

### 6.1. Funcionalidades futuras

Foram consideradas diversas funcionalidades durante o planeamento do projeto, enquanto as principais foram implementadas, algumas das que ficaram por implementar são demasiado pertinentes para não incluir neste relatório.

Reconhecimento facial é uma funcionalidade que pretendíamos desenvolver uma vez que a maioria dos projetos explorados no 2º capítulo não a tem, tratando-se assim de uma mais-valia que distingue o nosso projeto dos restantes. Devido ao elevado consumo de recursos durante a captura de imagens, foi previsto que seria necessário a utilização de uma segunda placa ESP32 para transmissão de áudio para possibilitar as chamadas de voz, com isto a campainha passaria a conter dois microcontroladores um para o áudio e outro para o vídeo.

Apesar do reconhecimento facial ser uma das funcionalidades previstas no enunciado do projeto, estando planeado ser utilizado para reconhecer os visitantes mais usuais, esta não chegou a ser implementada. Contudo esta funcionalidade foi planeada e estudada fazendo uso da biblioteca OpenCV que permite criar o modelo de reconhecimento facial. (Exemplo<sup>24</sup>)

A apresentação de dados estatísticos foi planeada da seguinte forma: deverá ser contado o número de vezes que o movimento é detetado num dia, sendo os dias organizados por dias da semana. Deste modo seria possível detetar possíveis alterações na rotina esperada e enviar um alerta ao utilizador. Consideramos que o seu desenvolvimento teria sido outra mais-valia, explorando mais as funcionalidades de sistema de segurança.

Também foi implementado um sistema para o nome e foto do usuário fosse automaticamente usado quando fosse necessário, no entanto ficou a faltar uma página *web* para editar este tipo de informações (foto, *username*, *password*).

O projeto também implementa o envio de configurações tais como tempo de gravação ou duração do relê aberto, a configuração seria individual para cada ESP32, no entanto, este sistema necessita de alterações. Primeiro este tipo de configurações precisa de ser guardado na base de dados, é necessário também só enviar esta configuração no fim de registar o *doorbell*, com a versão atual a configuração é enviada em conjunto com a confirmação de que a *doorbell* está registada. Por fim, faltou desenvolver uma página *web* para o utilizador poder editar estas configurações para cada ESP32.

Tal como mencionado no capítulo 4, na implementação do serviço de alertas, não foi implementado o sistema pretendido, pois o *link* enviado ao utilizador não faz login automaticamente usando um JWT, mas consideramos que é uma funcionalidade que seria bem recebida caso seja implementada futuramente.

Por fim falta apontar uma falha de segurança que deve ser considerada para implementação futura, que é o uso de SSL em vez de *sockets*, pois estes tornam a comunicação entre o servidor e o ESP32 segura usando encriptação e autenticação da mesma, serão necessários testes de carga caso seja implementado pois encriptação

---

<sup>24</sup> <https://expoleet.medium.com/flask-opencv-face-recognition-b9cbc3d1d280>

geralmente usa bastante poder de processamento e o ESP32 poderá não ter capacidade para tal em simultâneo com as diversas funcionalidades que tem atualmente.

## Referências

[Home Assistant] - <https://www.home-assistant.io/>

[ESPHome] - <https://esphome.io/>

[Telegram] - <https://telegram.org/>

[Jitsi Meet] - <https://meet.jit.si/>

[IoT Doorbell] - <https://www.hackster.io/taiyuk/iot-doorbell-faeel8>

[Smart Doorbell System using IoT] - <https://www.pantechsolutions.net/smart-doorbell-system-using-iot>

[DIS Smart Home Doorbell] - <https://www.instructables.com/DIY-Smart-Home-Doorbell-for-Less-Than-40/>

[Automatic Smart Door Bell with IoT] - <https://www.skyfilabs.com/project-ideas/automatic-smart-door-bell-using-iot>

[SS4H-SD Smart Doorbell] - <https://smartsolutions4home.com/ss4h-sd-smart-doorbell/>

[Ring Video Doorbell] - <https://www.lifewire.com/how-ring-doorbell-works-4583925>

[Smart Doorbell/Video Intercom System] - <https://www.hackster.io/hackershack/smart-doorbell-video-intercom-system-e5aa61>

[DIY Video Doorbell with ESP32] - <https://ei23.com/smarthome/awesome-diy-video-doorbell-based-on-an-esp32/>

[DIY Doorbell] - [https://youtu.be/6zoaxnAxR\\_c](https://youtu.be/6zoaxnAxR_c)

[DIY Ring-Doorbell] - [https://youtu.be/9bJFWIVm\\_Fo](https://youtu.be/9bJFWIVm_Fo)

[Flask] - <https://flask.palletsprojects.com/en/2.1.x/>

[Jinja] - <https://jinja.palletsprojects.com/en/3.1.x/>

[WiFiManager] - <https://github.com/tzapu/WiFiManager>

[FCGI] - <https://flask.palletsprojects.com/en/1.1.x/deploying/fastcgi>

[SQLite] - <https://www.sqlite.org/index.html>

[AWS] - <https://aws.amazon.com>

[JWT] - <https://jwt.io/>

[PyJWT] - <https://pyjwt.readthedocs.io/en/stable/>

[Python] - <https://www.python.org/doc/>

[Python Virtual Environments] - <https://docs.python.org/3/tutorial/venv.html>

[Werkzeug] - <https://werkzeug.palletsprojects.com/en/2.1.x/>

[Flask-Mail] - <https://flask-mail.readthedocs.io/en/latest/>

[OpenCV] - <https://opencv.org/>

[Flask-Session] - <https://www.educba.com/flask-session/>

[WiFiClient] - <https://docs.espressif.com/projects/arduino-esp32/en/latest/>

[Server Socket] - <https://docs.python.org/3/library/socket.html>

[ESP32-CAM]

<https://docs.platformio.org/en/latest/boards/espressif32/esp32cam.html#ai-thinker-esp32-cam>

[ESP32 Camera Driver] - <https://github.com/espressif/esp32-camera>

[Let's Encrypt] - <https://letsencrypt.org/>

[email-validator] - <https://pypi.org/project/email-validator/>

[bcrypt] - <https://pypi.org/project/bcrypt/>

[NumPy] - <https://numpy.org/>

[Python time] - <https://docs.python.org/3/library/time.html>

[Python threading] - <https://docs.python.org/3/library/threading.html>

[Python selectors] - <https://docs.python.org/3/library/selectors.html>

[Bootstrap] - <https://getbootstrap.com/>

[Bootstrap Icons] - <https://icons.getbootstrap.com/>

[Python Regular expression operations] - <https://docs.python.org/3/library/re.html>

[Python Logging] - <https://docs.python.org/3/library/logging.html>

[Python Logging handlers] - <https://docs.python.org/3/library/logging.handlers.html>

[Moment.js] - <https://momentjs.com/>

[Python base64] - <https://docs.python.org/3/library/base64.html>

[Python struct] - <https://docs.python.org/3/library/struct.html>

[Python multiprocessing] - <https://docs.python.org/3/library/multiprocessing.html>

## Glossário

Biblioteca – “coleção de subprogramas utilizados no desenvolvimento de software” <sup>25</sup>

Framework – “Abstração que une códigos comuns entre vários projetos de software provendo uma funcionalidade genérica.” <sup>26</sup>

Thread – “Tarefa que um determinado programa realiza” <sup>27</sup>

---

<sup>25</sup> [https://pt.wikipedia.org/wiki/Biblioteca\\_\(computa%C3%A7%C3%A3o\)](https://pt.wikipedia.org/wiki/Biblioteca_(computa%C3%A7%C3%A3o)) (17-07-2022)

<sup>26</sup> <https://pt.wikipedia.org/wiki/Framework> (17-07-2022)

<sup>27</sup> [https://pt.wikipedia.org/wiki/Thread\\_\(computa%C3%A7%C3%A3o\)](https://pt.wikipedia.org/wiki/Thread_(computa%C3%A7%C3%A3o)) (17-07-2022)