



Rui Pedro dos  
Santos Oliveira

**Sistema de monitorização e controlo da produção  
de Salicornia na Ria de Aveiro**

**Monitorization and control system of the  
production of Salicornia in the Ria de Aveiro**

# **DOCUMENTO PROVISÓRIO**







Rui Pedro dos  
Santos Oliveira

**Sistema de monitorização e controlo da produção  
de Salicornia na Ria de Aveiro**

**Monitorization and control system of the  
production of Salicornia in the Ria de Aveiro**

# **DOCUMENTO PROVISÓRIO**





Rui Pedro dos  
Santos Oliveira

**Sistema de monitorização e controlo da produção  
de Salicornia na Ria de Aveiro**

**Monitorization and control system of the  
production of Salicornia in the Ria de Aveiro**

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Joaquim Manuel Henriques de Sousa Pinto, Professor Associado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro e do Doutor José Alberto Gouveia Fonseca, Professor Associado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.



## **o júri / the jury**

presidente / president

**ABC**

Professor da Universidade de Aveiro

vogais / examiners committee

**Doutor Joaquim Manuel Henriques de Sousa Pinto**

Professor Auxiliar da Universidade de Aveiro (orientador)

**GHI**

Professor associado da Universidade J (co-orientador)

**KLM**

Professor Catedrático da Universidade N (arguente)



## **agradecimentos / acknowledgements**

Ao meu orientador, Professor Joaquim Sousa Pinto, quero agradecer pelo acompanhamento, disponibilidade manifestada sempre acompanhada de boa disposição. Ao professor Joaquim Aberto Fonseca, pela ideia e contribuições no projeto. À Professora Helena Silva do Departamento de Biologia da Universidade de Aveiro, pela disponibilização de material referente Salicornia. Ao Sr. José M. G. Pereira pelas fotografias originais de *Salicornia ramosissima* na Ria de Aveiro

A todo o pessoal do IEETA que me acompanhou durante esta jornada, em especial à Madalena, ao Gabriel e à Sara pela companhia e camaradagem durante grande parte do semestre.

Aos meus amigos!

À Magda, minha namorada, por toda a paciência e por sempre acreditar em mim. Obrigado por todas as sugestões!

Por último mas não menos importante, quero agradecer a toda a minha família, em especial aos meus pais, irmãos, cunhados e sobrinhos, que ao longo destes cinco anos sempre estiveram ao meu lado e sempre me fizeram acreditar no alcance desta etapa da minha vida.



**palavras chave**

Salicórnia, sistema de informação, plataforma *web*, monitorização, atuação remota, API REST, simulação em *hardware*, sistema de video-vigilância

**resumo**

A evolução tecnológica é algo que sempre esteve presente na vida do ser humano desde os seus primórdios até aos dias de hoje, numa relação que cresceu e continua a crescer a um ritmo alucinante. Atualmente, o paradigma que atravessa qualquer atividade económica consiste em otimizar os recursos com objetivo de maximizar a produção através da evolução tecnológica. Na produção agrícola isto não é exceção e, por esse motivo os mecanismos de monitorização de parâmetros que influenciam a rentabilidade e a qualidade da produção começam a ser indispensáveis e preponderantes no sucesso do negócio. Desta forma, no cultivo da Salicórnia, uma planta que cresce na zona da Ria de Aveiro, também é essencial a criação de um sistema que permita monitorizar e ajudar a controlar as condições ideais de cultivo da espécie.

Esta dissertação tinha como principal objetivo a projeção e implementação de um sistema de informação para o controlo e monitorização do cultivo da Salicórnia, em colaboração com uma empresa da região de Aveiro e o Departamento de Biologia da Universidade de Aveiro. Para isso, fez-se a modelação dos requisitos da empresa e planeou-se a arquitetura do sistema. Seguidamente, desenvolveu-se uma aplicação web e uma API que permitem monitorizar a temperatura, a luminosidade e o nível da água nas leiras de cultivo de Salicórnia. Mais ainda, é possível atuar remotamente controlando válvulas de admissão de água. Para além disso, projetou-se um protótipo em *hardware* para simulação deste cenário e, incorporou-se um sistema de videovigilância que permite a observação dos campos de cultivo.

O sistema desenvolvido vai de encontro aos requisitos do cliente, para além disso, é um solução de baixo custo e eficaz na aquisição, processamento e armazenamento de dados. Adicionalmente, este sistema encontra-se estruturado para poder ser aplicado noutros contextos para além do cultivo da Salicórnia.



**keywords** Salicornia, information system, web platform, monitoring, remote control, REST API, hardware simulation, video surveillance system

**abstract** falta verificar se está tudo bem para traduzir ...



# Conteúdo

<b>Lista de Figuras</b>	<b>v</b>
<b>Lista de Tabelas</b>	<b>ix</b>
<b>Acrónimos</b>	<b>xi</b>
<b>1 Introdução</b>	<b>1</b>
1.1 Motivação . . . . .	2
1.2 Objetivos . . . . .	2
1.3 Metodologia de trabalho . . . . .	3
1.4 Organização do documento . . . . .	4
<b>2 O <i>Internet of Things</i> no cultivo da Salicornia</b>	<b>7</b>
2.1 Características da planta . . . . .	7
2.2 Condições ideais de cultivo da Salicornia . . . . .	8
2.3 Importância da planta . . . . .	9
2.4 Evolução tecnológica: o <i>Internet of Things</i> . . . . .	9
2.5 Considerações finais . . . . .	11
<b>3 Estado da arte</b>	<b>13</b>
3.1 Conceitos tecnológicos . . . . .	13
3.2 Sistema de Gestão de Base de Dados . . . . .	14
3.2.1 MySQL . . . . .	14
3.2.2 SQL Server . . . . .	14
3.2.3 PostgreSQL . . . . .	14
3.2.4 Comparação e solução adotada . . . . .	14
3.3 Desenvolvimento <i>web</i> . . . . .	15
3.3.1 ASP.net . . . . .	16
3.3.2 Flask . . . . .	16
3.3.3 Django . . . . .	16

3.3.4	Conclusões e solução adotada . . . . .	17
3.4	Desenvolvimento <i>mobile</i> . . . . .	18
3.4.1	Plataformas nativas . . . . .	18
3.4.2	Multi-plataforma . . . . .	18
3.4.3	Conclusões e solução adotada . . . . .	19
3.5	REST <i>Frameworks</i> . . . . .	20
3.5.1	Django Rest Framework . . . . .	20
3.5.2	Restlet . . . . .	21
3.5.3	Flask-RESTful . . . . .	21
3.5.4	Conclusões e solução adotada . . . . .	21
3.6	Microcontroladores . . . . .	22
3.6.1	Arduino Nano . . . . .	22
3.6.2	Raspberry Pi 3 . . . . .	23
3.7	Sensores . . . . .	24
3.7.1	Sensor de temperatura . . . . .	24
3.7.2	Sensor de luminosidade . . . . .	25
3.7.3	Sensor de salinidade . . . . .	26
3.8	Tecnologias de comunicação . . . . .	26
3.8.1	Zigbee . . . . .	26
3.8.2	Bluetooth (BLE) . . . . .	26
3.8.3	Wi-Fi (IEEE 802.11) . . . . .	27
3.8.4	Sigfox . . . . .	27
3.8.5	Comparação entre tecnologias de comunicação . . . . .	27
<b>4</b>	<b>Sistema de controlo e monitorização: arquitetura e modelação</b>	<b>29</b>
4.1	Descrição global do sistema . . . . .	29
4.2	Componentes . . . . .	31
4.2.1	<i>Sensor Module</i> . . . . .	31
4.2.2	<i>Controller Module</i> . . . . .	32
4.3	Análise de requisitos . . . . .	32
4.3.1	Requisitos funcionais . . . . .	33
4.3.2	Requisitos não funcionais . . . . .	35
4.4	Modelação . . . . .	35
4.4.1	Entidades envolvidas . . . . .	35
4.4.2	Casos de uso . . . . .	36
4.4.3	Modelo de dados . . . . .	40
4.5	Arquitetura lógica . . . . .	43
4.6	Arquitetura física . . . . .	44
4.6.1	Sistema de informação . . . . .	45

Aplicação web . . . . .	45
API REST . . . . .	47
Documentação interativa . . . . .	49
Implementação do sistema . . . . .	49
Aplicação mobile . . . . .	50
4.6.2 Simulação em <i>hardware</i> . . . . .	51
Sensores utilizados . . . . .	51
Comunicação . . . . .	53
4.6.3 Sistema de videovigilância . . . . .	55
Biblioteca para processamento de imagem: OpenCV . . . . .	55
4.7 Diagrama de componentes . . . . .	57
4.8 Considerações finais . . . . .	58
<b>5 Implementação</b>	<b>59</b>
5.1 Sistema de informação . . . . .	59
5.1.1 Sistema de registo e autenticação . . . . .	60
5.1.2 Geração de alarmes . . . . .	62
5.1.3 Visualização dos dados e cálculos estatísticos . . . . .	63
5.1.4 API . . . . .	64
5.1.5 Documentação da API . . . . .	65
5.1.6 Aplicação <i>web</i> . . . . .	66
5.1.7 <i>Deploy</i> do projecto . . . . .	67
5.2 Simulação em <i>hardware</i> . . . . .	69
5.2.1 Arduino Nano . . . . .	69
5.2.2 Raspberry Pi 3 . . . . .	70
5.3 Sistema de videovigilância . . . . .	71
5.3.1 Algoritmo de deteção de intrusos . . . . .	72
5.4 Considerações finais . . . . .	74
<b>6 Testes e resultados</b>	<b>75</b>
6.1 Testes funcionais . . . . .	75
6.1.1 <i>Application Programming Interface (API) Representational State Transfer (REST)</i> . . . . .	75
6.1.2 Comunicação via Bluetooth . . . . .	76
6.1.3 Deteção de intrusos . . . . .	77
6.2 Resultados . . . . .	78
6.3 Considerações finais . . . . .	83

<b>7 Conclusões e trabalho futuro</b>	<b>85</b>
7.1 Conclusões . . . . .	85
7.2 Problemas encontrados . . . . .	86
7.3 Trabalho futuro . . . . .	86
<b>A Mockup da aplicação mobile</b>	<b>93</b>
<b>B Implementação do trigger SQL</b>	<b>95</b>
<b>C Application Programming Interface REST</b>	<b>97</b>
<b>D Interligação de componentes</b>	<b>101</b>

# **Lista de Figuras**

1.1	Fases de desenvolvimento de um <i>software</i> . . . . .	3
2.1	Coloração da planta <i>Salicornia ramosissima</i> na primavera (à esquerda) e no outono (à direita) . . . . .	8
2.2	Esquema representativo do ciclo de vida da <i>Salicornia ramosissima</i> . . . . .	8
2.3	Evolução da Internet em cinco fases . . . . .	10
2.4	Pirâmide do conhecimento: modelo DIKW . . . . .	11
3.2	Arduino Nano . . . . .	23
3.3	Identificação dos diferentes pinos num Arduino Nano . . . . .	23
3.4	Raspberry Pi 3 . . . . .	24
3.5	Principais componentes de um Raspberry Pi 3 . . . . .	24
3.6	Sensibilidade luminosa das plantas durante a fotossíntese . . . . .	25
4.1	Ilustração dos principais componentes do sistema . . . . .	29
4.2	Ilustração da distribuição dos módulos em duas leiras . . . . .	30
4.3	Esquema de componentes e respetiva comunicação entre três <i>Sensor Module</i> (SM) e um <i>Controller Module</i> (CM) . . . . .	31
4.4	Casos de uso para a aplicação <i>web</i> ( <i>dashboard</i> ) . . . . .	36
4.5	Casos de uso para a aplicação <i>mobile</i> . . . . .	37
4.6	Esquema relacional da estrutura da base de dados . . . . .	40
4.7	Esquema representativo da arquitetura lógica do sistema . . . . .	43
4.8	Arquitetura física (blocos principais) . . . . .	44
4.9	Arquitetura do sistema de informação ( <i>dashboard</i> , base de dados e API) . . . . .	46
4.10	Processo de autenticação em HTTP através de <i>token</i> . . . . .	48
4.11	Arquitetura da aplicação <i>mobile</i> com respetivas tecnologias . . . . .	50
4.12	Sensor TTC 104 NTC . . . . .	51
4.13	Esquema eletrónico da ligação do sensor de temperatura . . . . .	51
4.14	Sensor de foto-resistência GL5528 . . . . .	52
4.15	Esquema eletrónico da ligação do sensor de luminosidade . . . . .	52

4.16	<i>Water Level Switch Liquid Level Sensor Plastic Ball Float</i>	53
4.17	Esquema eletrónico da ligação do sensor de nível líquido	53
4.18	<i>Light Emitting Diode (LED)</i>	53
4.19	Esquema eletrónico da ligação do LED	53
4.20	Comunicação entre componentes da simulação em <i>hardware</i>	54
4.21	Módulo Bluetooth HC-06	55
4.22	Esquema eletrónico da ligação do módulo Bluetooth	55
4.23	Raspberry Pi Camera Board V2 8MP 1080p	56
4.24	Arquitetura do sistema de videovigilância e deteção de intrusos	57
4.25	Diagrama final de componentes do sistema	58
5.1	Painel administrativo da <i>framework</i> Django	60
5.2	Diagrama de atividades do processo de registo e autenticação	61
5.3	Diagrama de fluxo para geração de alarmes	62
5.4	Documentação da API REST com a ferramenta Swagger	65
5.5	Lógica de implementação do <i>script</i> para o <i>Controller Module</i>	71
5.6	Exemplo da aplicação do parâmetro <b>winStride</b>	73
5.7	Pirâmide de imagens para diferentes <b>scale</b>	73
6.1	Resultado da utilização da ferramenta <i>Advanced REST client</i>	76
6.2	Resultado da interação com a aplicação <i>Bluetooth Terminal HC-05</i>	77
6.3	<i>Breadboard</i> com ligação dos diferentes componentes, destacando-se a activação do LED	77
6.4	Imagen original (frame1)	77
6.5	Resultado obtido (frame1)	77
6.6	Imagen original (frame2)	78
6.7	Resultado obtido (frame 2)	78
6.8	Imagen original (frame3)	78
6.9	Resultado obtido (frame 3)	78
6.10	Interface para novo registo, destacando o <i>feedback</i> dado ao utilizador	79
6.11	Interface para validação de utilizadores, com <i>feedback</i> após validação	79
6.12	Interface para visualizar sensores	79
6.13	Interface para adicionar sensores	79
6.14	Visualização dos <i>Sensor Modules</i> associados a um <i>Controller Module</i>	79
6.15	Interface para adição de um novo <i>Sensor Module</i>	79
6.16	Visualização de dados destacando a filtragem por data	80
6.17	Visualização tabular de dados, destacando a exportação para CSV	80
6.18	Visualização gráfica dos dados adquiridos por um sensor de luminosidade durante quatro dias, destacando os valores máximos, mínimos e médios	80
6.19	Processo de atuação remota	81

6.20 Resultado do processo de atuação remota . . . . .	81
6.21 Interface principal após <i>login (dashboard)</i> . . . . .	81
6.22 Sistema de videovigilância incorporado na <i>dashboard</i> . . . . .	82
6.23 Resultado da interface responsiva da plataforma <i>web</i> . . . . .	82
7.1 Esquema resumo do trabalho desenvolvido . . . . .	85
A.1 <i>Mockup</i> da aplicação <i>mobile</i> . . . . .	93
A.2 <i>Mockup</i> da aplicação <i>mobile</i> (continuação) . . . . .	94
D.1 Protótipo da solução em <i>hardware</i> desenvolvida . . . . .	101



# **Lista de Tabelas**

3.1	<i>Ranking BD-Engines</i> para popularidade dos Sistema de Gestão de Base de Dados (SGBD) estudados . . . . .	15
3.2	Comparação entre MySQL, SQL Server e PostgreSQL . . . . .	15
3.3	Comparação entre <i>frameworks</i> REST estudadas. . . . .	21
3.4	Características principais do Arduino Nano . . . . .	23
3.5	Comparação entre algumas características da versão 2 e 3 do Raspberry Pi . .	24
3.6	Comparação entre as tecnologias de comunicação abordadas . . . . .	28
4.1	Especificação das tabelas da base de dados existentes no sistema . . . . .	41
4.2	Especificação das tabelas da base de dados existentes no sistema (continuação)	42
4.3	<i>Endpoints</i> da API REST e respetivos métodos a implementar . . . . .	49
4.4	Características do sensor TTC 104 . . . . .	52
4.5	Características do sensor GL5528 . . . . .	52
4.6	Características do módulo bluetooth HC-06 . . . . .	55
4.7	Características do módulo bluetooth HC-06 . . . . .	56
5.1	Estrutura do ficheiro do tipo CSV possível de exportação . . . . .	64



# Acrónimos

<b>API</b>	<i>Application Programming Interface</i>	<b>I/O</b>	<i>Input/Output</i>
<b>CM</b>	<i>Controller Module</i>	<b>IDE</b>	<i>Integrated Development Environment</i>
<b>CPU</b>	<i>Central Processing Unit</i>	<b>IIS</b>	<i>Internet Information Services</i>
<b>CSI</b>	<i>Camera Serial Interface</i>	<b>IP</b>	<i>Internet Protocol</i>
<b>CSS</b>	<i>Cascading Style Sheets</i>	<b>ISM</b>	<i>Industrial, Scientific, Medical</i>
<b>CSV</b>	<i>Comma-Separated Values</i>	<b>IoT</b>	<i>Internet of Things</i>
<b>DIKW</b>	<i>Data-Information-Knowledge-Wisdom</i>	<b>JSON</b>	<i>JavaScript Object Notation</i>
<b>DOM</b>	<i>Document Object Model</i>	<b>JS</b>	<i>JavaScript</i>
<b>DOM</b>	Modelo de Objeto de Documento	<b>LDR</b>	<i>Light Dependent Resistor</i>
<b>EDR</b>	<i>Enhanced Data Rate</i>	<b>LED</b>	<i>Light Emitting Diode</i>
<b>FK</b>	<i>Foreign Key</i>	<b>MTV</b>	<i>Model-Template-View</i>
<b>FTP</b>	<i>File Transfer Protocol</i>	<b>MVC</b>	<i>Model-View-Controller</i>
<b>GPS</b>	<i>Global Positioning System</i>	<b>NTC</b>	<i>Negative Temperature Coefficient</i>
<b>HATEOAS</b>	<i>Hypermedia As The Engine Of Application State</i>	<b>ORM</b>	<i>Object Relational Mapper</i>
<b>HTML</b>	<i>HyperText Markup Language</i>	<b>ORM</b>	<i>Object-Relational Mapping</i>
<b>HTTP</b>	<i>HyperText Transfer Protocol</i>	<b>PANs</b>	<i>Wireless personal area networks</i>
		<b>PAR</b>	<i>Photosynthetically Active Radiation</i>

<b>PK</b>	<i>Primary keys</i>	<b>SQL</b>	<i>Structured Query Language</i>
<b>QR</b>	<i>Quick Response</i>	<b>SVM</b>	<i>Support Vector Machine</i>
<b>RAM</b>	<i>Random Access Memory</i>	<b>TCP</b>	<i>Transmission Control Protocol</i>
<b>REST</b>	<i>Representational State Transfer</i>	<b>UI</b>	<i>User Interface</i>
<b>ROM</b>	<i>Read-Only Memory</i>	<b>URL</b>	<i>Uniform Resource Locator</i>
<b>RTMP</b>	<i>Real-Time Messaging Protocol</i>	<b>USB</b>	<i>Universal Serial Bus</i>
<b>SDLC</b>	<i>Systems Development Life Cycle</i>	<b>VPS</b>	<i>Virtual Private Server</i>
<b>SGBD</b>	Sistema de Gestão de Base de Dados	<b>WSGI</b>	<i>Web Server Gateway Interface</i>
<b>SMTP</b>	<i>Simple Mail Transfer Protocol</i>	<b>WSGI</b>	<i>Web Server Gateway Interface</i>
<b>SM</b>	<i>Sensor Module</i>	<b>WWW</b>	<i>World Wide Web</i>
<b>SOAP</b>	<i>Simple Object Access Protocol</i>	<b>XML</b>	<i>Extensible Markup Language</i>

# Introdução

O contínuo crescimento da população mundial trouxe consigo um massivo aumento da exploração dos recursos naturais do planeta Terra, tais como plantas, animais e minerais. No caso das plantas, estas são usadas pelos seres humanos desde a antiguidade tanto para a alimentação como para o tratamento de diversas doenças. Muitas das espécies que surgem no mundo, numa primeira fase são consideradas pragas, contudo e após alguns estudos intensivos são descobertas verdadeiras pérolas devido às propriedades que apresentam. Um exemplo disso é a planta que servirá de contexto à elaboração desta dissertação, denominada por *Salicornia ramosissima*.

A Salicórnia é uma planta suculenta adaptada a ambientes salinos que se desenvolve maioritariamente em ambientes aquáticos com elevado teor de sal, crescendo ao longo dos estuários e sapais (salinas) costeiros do Mediterrâneo. Esta planta é utilizada para os mais diversos fins nomeadamente, como substituto do sal marinho.

Nos dias de hoje, nos processos de agricultura intensiva, há necessidade de otimizar os mecanismos de produção controlando certos aspectos para obter o maior proveito dos recursos e, ao mesmo tempo, a garantia da qualidade dos produtos. Para tal, torna-se fulcral conhecer as condições ideais para o melhor desenvolvimento da espécie. Para entender quais os fatores que influenciam esse desenvolvimento, é necessária a existência de equipamentos sensoriais que recolhem e transmitem os dados obtidos, a fim de serem analisados e processados. Por outro lado, é essencial garantir que esses parâmetros se encontram dentro dos valores ótimos, sendo necessário mecanismos inteligentes capazes de atuar e desencadear processos de correção ao sistema.

Uma vez que o cultivo da Salicórnia é relativamente recente, existe um grande interesse, por parte dos biólogos e dos agricultores, em entender de que forma certos parâmetros podem ou não influenciar o crescimento da espécie, com o propósito de potencializar a produtividade e rentabilidade do seu cultivo. Com este propósito e tirando partido da evolução tecnológica,

pretende-se desenvolver um sistema que permita a introdução da sensorização no cultivo da Salicórnia.

## 1.1 Motivação

O incentivo desta dissertação consiste em incluir sensorização na produção de Salicórnia, com o propósito de, através da consecutiva monitorização, oferecer dados reais para que os investigadores do Departamento de Biologia da Universidade de Aveiro possam identificar as condições ideais de certos parâmetros favoráveis ao crescimento da Salicórnia. Adicionalmente, esta solução servirá de embrião para um sistema de controlo e monitorização idealizado por uma empresa da região de Aveiro que se dedica à produção de Salicórnia, denominada por "Horta dos Peixinhos"<sup>1</sup>. Este sistema permitirá agilizar o mecanismo de transferência de águas, automatizando todo o processo de irrigação e cultivo desta espécie.

## 1.2 Objetivos

O objetivo geral do trabalho prático desta dissertação é desenvolver um sistema de informação que permita monitorizar e controlar o cultivo da Salicórnia. Para além disso, pretende-se desenvolver um protótipo em *hardware* que permita simular um cenário desejado. Idealiza-se que a solução encontrada esteja inserida no universo do *Internet of Things* (IoT), tanto quanto possível, isto é, que apresente alguma escalabilidade, baixo custo, autonomia total de funcionamento e capacidade de comunicação sem fios. Face aos objetivos supracitados, estabeleceram-se os seguintes objetivos específicos:

1. Recolher os requisitos do cliente e definir as funcionalidades com base na respetiva modelação;
2. Definir uma arquitetura para o cenário apresentado, tornando-o apto a ser adaptado a outros contextos;
3. Desenvolver uma API que disponibilize os serviços deste sistema, possibilitando a criação de novas aplicações;
4. Criar uma plataforma *web* que permita a interação simples e intuitiva com o utilizador para disponibilizar a leitura de dados dos diferentes sensores;
5. Permitir que o utilizador do sistema atue remotamente na ativação/desativação de atuadores;
6. Gerar alarmes quando os valores obtidos pelos sensores estão fora da gama dos valores ideais;

---

<sup>1</sup><http://hortadospeixinhos.com/>

7. Simular em *hardware* o cenário apresentado, utilizando alguns sensores ambientais e comunicação sem fios;
8. Criar um sistema de videovigilância que permita a deteção de intrusos e supervisionar os campos de cultivo.

### 1.3 Metodologia de trabalho

Para a elaboração do trabalho prático desta dissertação, foi adotada uma metodologia de investigação orientada à engenharia. Esta metodologia, assenta num conjunto de passos continuados pelos engenheiros que permite encontrar um conjunto de soluções para dar resposta aos problemas em questão. Os diversos passos, centram-se sobretudo na definição dos problemas e investigação do tema, especificação dos requisitos, *brainstorm* de soluções e escolha da melhor, desenvolvimento, protótipos e respetivos testes da solução. Por fim, se a solução não for de encontro com os requisitos estabelecidos, o processo é realizado novamente[1]. Este modelo é conhecido como metodologia de desenvolvimento iterativo e incremental.

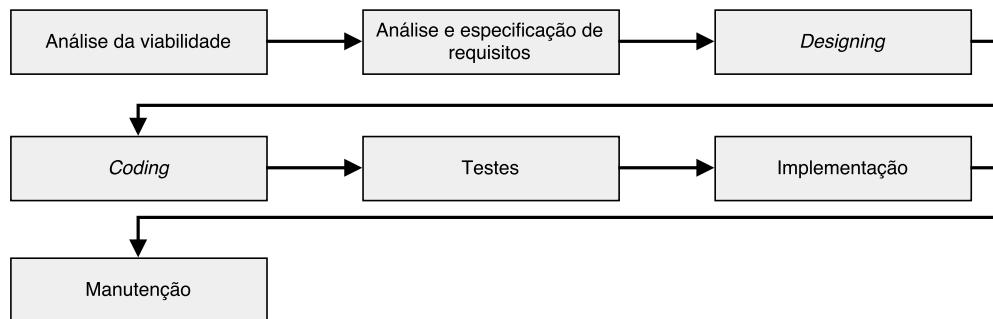


Figura 1.1: Fases de desenvolvimento de um *software* (Adaptado de [2])

No contexto do desenvolvimento desta solução, foram consideradas várias etapas do ciclo de vida de desenvolvimento de um *software* definidas por *Munish Kaur et al.*[2]. Este ciclo de vida, também conhecido como *Systems Development Life Cycle* (SDLC), é composto genericamente por quatro fases principais: conceção, projeção, criação e implementação. Antes do SDLC, o processo de desenvolvimento de *software* era tomado como uma atividade informal sem regras nem padrões. Isto pode dar origem a vários problemas, tais como o atraso no desenvolvimento, aumento de custos e baixa qualidade do *software* criado. Existem inúmeros modelos e visões que propõem alguns padrões e etapas necessárias ao desenvolvimento de um sistema com qualidade. Na figura 1.1 encontram-se as sete etapas consideradas por *Munish Kaur et al.*[2] que seguidamente serão descritas.

- **Análise da viabilidade:** nesta fase são analisados os dados de entrada e saída, processamento necessário, análise de custos e planeamento do projeto. É ainda incluída a viabilidade técnica em termos de *software*, *hardware* e pessoas qualificadas;

- **Análise e especificação de requisitos:** nesta fase são recolhidos e analisados os requisitos necessário para a elaboração do *software*. No final, pretende-se que sejam conhecidos os vários requisitos do *software* e seja também criado um documento que os especifique;
- **Designing:** consiste na tradução dos requisitos especificados para uma estrutura lógica. No final prevê-se que seja elaborado um documento de especificação do *design*;
- **Coding:** a programação real é elaborada nesta fase. O documento de *design* é traduzido para o código-fonte numa determinada linguagem de forma a que possa ser executado;
- **Testes:** o código-fonte gerado na fase anterior é testado usando vários cenários de teste com o objetivo de corrigir e avaliar o *software*;
- **Implementação:** o *software* desenvolvido é implementado para que possa ser disponibilizado ao utilizador para uso real. Pretende-se que o utilizador do sistema possa reportar erros ou problemas quando encontrados;
- **Manutenção:** o *software* poderá sofrer alterações para solucionar problemas que possam ter sido encontrados. Esta fase é responsável pela pós-implementação e manutenção do *software* para o seu bom funcionamento.

Adicionalmente, durante o desenvolvimento do trabalho prático desta dissertação foram criados alguns repositórios Git<sup>2</sup> que permitiram controlar as diferentes versões do código desenvolvido, permitindo ainda um *backup* sistemático e um armazenamento seguro. No contexto deste trabalho, a existência de um sistema de controlo de versões foi especialmente importante já que permitiu recuperar versões anteriores.

## 1.4 Organização do documento

A presente dissertação encontra-se dividida em sete capítulos. No primeiro capítulo, Introdução, é descrita e enfatizada a importância da monitorização e controlo do cultivo da Salicórnia, bem como os principais objetivos desta dissertação. Seguidamente, no capítulo dois, O IoT no cultivo da Salicórnia, será apresentada a Salicórnia destacando as suas características e importância. Para além disso, será exposta a evolução tecnológica até ao universo do IoT. De seguida, o capítulo três, Estado da arte, contém o enquadramento teórico onde são abordadas as tecnologias possíveis de utilização bem como a respetiva comparação. No quarto capítulo, Sistema de controlo e monitorização: arquitetura e modelação, são apresentados todos os requisitos do sistema e realizada a modelação do mesmo. Para além disso, é apresentada a arquitetura do sistema. Posteriormente, no capítulo quinto, Implementação,

---

<sup>2</sup><https://git-scm.com/>

é descrita a conceção e implementação do sistema e dos seus vários componentes. No sexto capítulo, Testes e resultados, são apresentados alguns testes funcionais e respetivos resultados do sistema desenvolvido. Finalmente, no capítulo sétimo, Conclusão e trabalho futuro, é analisado e refletido todo o trabalho realizado nesta dissertação por forma a concluir se os resultados obtidos satisfazem os objetivos previamente definidos. No fim, será mencionado e discutido todo o trabalho futuro que possa vir a ser realizado com intenção de aperfeiçoar a solução proposta.



# O Internet of Things no cultivo da Salicórnia

A *Salicornia ramosissima J. Woods (S. ramosissima)*[3] que impulsiona toda esta dissertação, é uma espécie do género *Salicornia L.*, pertencente à família das beterrabas denominada de *Chenopodiaceae*[4]. Neste capítulo será apresentada a planta, as suas principais características e propriedades, bem como as suas diferentes aplicações medicinais e alimentares. Este capítulo servirá ainda como uma pequena introdução ao conceito de IoT e a sua importância no contexto deste projeto.

## 2.1 Características da planta

A Salicórnia é uma espécie halófita, adaptada a viver em ambientes com elevado teor de sal[5], sendo uma das mais evoluídas da sua família. É uma planta anual de dimensão pequena, aparentemente sem folhas, ereta, com caules carnudos e suculentos, simples e/ou extremamente ramificados, segmentados por articulações[6], geralmente com menos de 30 cm de altura[7]. Esta planta tem uma coloração durante a maior parte do ano verde-escuro mas a sua ramagem torna-se verde-amarelado ou mesmo vermelho-púrpura no outono[6] (figura 2.1).

A *Salicornia ramosissima* desenvolve-se preferencialmente no litoral costeiro, em pântanos e sapais salgados ou em margens de salinas temporariamente alagadas. Encontra-se distribuída maioritariamente na parte oeste da Europa e a oeste da região do Mediterrâneo, sendo uma das espécies mais abundantes[8]. Em Portugal, onde é vulgarmente conhecida como erva-salada, sal verde e/ou espargos do mar[9], é encontrada frequentemente nas margens dos canais da Ria de Aveiro e Ria Formosa (Algarve)[9] sendo encontrada com menos frequência na região do Minho[6].



Figura 2.1: Coloração da planta *Salicornia ramosissima* na primavera (à esquerda) e no outono (à direita) (Fotografias por José M. G. Pereira)

Esta planta é pouco estudada pelos cientistas[8], sabendo-se apenas que possui um ciclo de vida anual bem definido (figura 2.2), com gerações discretas e as suas sementes são hermafroditas[11]. A Salicórnia cresce habitualmente entre março, início da sementeira (A da figura 2.2) com respetivo crescimento (B da figura 2.2) e novembro fechando assim o ciclo com a produção de sementes (E da figura 2.2). Entre maio e agosto decorre a colheita da planta[9] (C da figura 2.2) que pode ser utilizada para os mais diversos fins. A floração ocorre fundamentalmente no mês de outubro[8] (D da figura 2.2).

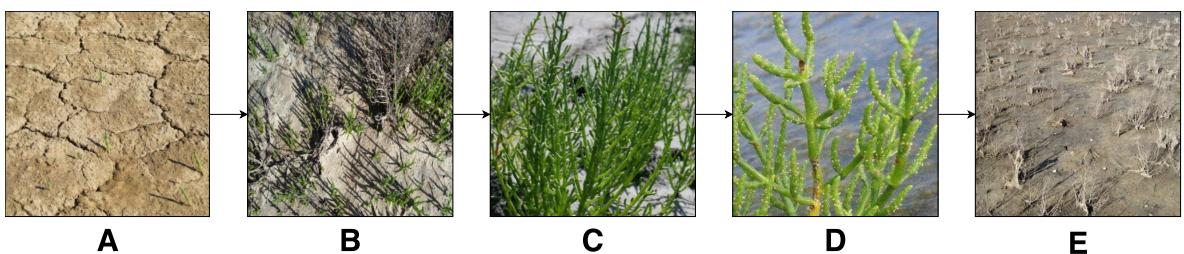


Figura 2.2: Esquema representativo do ciclo de vida da *Salicornia ramosissima*. A - semente incluída no sedimento; B - jovens plantas e plantas senescentes do ano anterior; C - planta no estado vegetativo, caule carnudo e articulado; D - planta no estado de floração; E - planta no estado senescente. (Fotografias por Helena Silva)

## 2.2 Condições ideais de cultivo da Salicórnia

O crescimento da *Salicornia ramosissima* é influenciado por diversos fatores ambientais que são atualmente estudados pelos biólogos, sendo a salinidade um dos mais importantes, já que influencia a distribuição, a abundância e a fisiologia da planta. Um estudo realizado por *Silva et al.*[11] comprova que esta planta halófita apresenta um crescimento ideal em salinidades baixas ou moderadas. Este estudo permite considerar ao contrário do que se

pensava, esta planta como uma halófita não obrigatória, já que o seu crescimento ideal não acontece em condições de salinidade elevada, tal como se esperava. Embora o crescimento ideal ocorra a baixas salinidades, a Salicórnia é capaz de tolerar níveis elevados de sais[12].

## 2.3 Importância da planta

Desde a antiguidade que as espécies do género *Salicornia L.* estão incluídas na alimentação humana. Normalmente consome-se crua, cozinhada ou seca. Quando crua é usada como acompanhamento das mais diversas refeições enquanto que seca ou triturada é usada como especiaria, para tempero na confeção de peixes, marisco ou carnes. O sal verde, como é conhecida, é um grande substituto do sal comum, pois é rico em substâncias depurativas e diuréticas. Os seus caules carnudos são bastante requisitados para cozinhas *gourmet*, não só pelo seu sabor salgado, mas também pelo seu elevado valor nutricional[13], nomeadamente pelos níveis de minerais e vitaminas antioxidantes, como a vitamina C e o  $\beta$ -caroteno. A Salicórnia é também uma fonte de proteínas e possui alto teor de ácidos gordos essenciais, destacando-se o ómega-3[14].

A nível medicinal, existem inúmeros estudos que revelam que as propriedades químicas da planta, tornam-na eficiente na prevenção e tratamento de algumas doenças, tais como, a hipertensão, cefaleias e escorbuto, diabetes, obesidade, cancro, entre outras[15].

Tendo em conta todas estas propriedades alimentares, medicinais e comerciais da Salicórnia, torna-se fulcral controlar o seu cultivo, a fim de otimizar a produção para tirar maior partido da sua importância biológica. Este controlo pode ser feito recorrendo à evolução tecnológica, tal como será descrito nas próximas secções deste capítulo.

## 2.4 Evolução tecnológica: o *Internet of Things*

Antes de descrever a importância e o conceito de IoT, é necessário entender as diferenças entre os termos Internet e Web (*World Wide Web (WWW)*), que são usados indistintamente pela sociedade. A Internet é a camada ou rede física composta por *switches*, *routers* e outros equipamentos[16]. A sua principal função é transportar informações de um ponto para outro de forma rápida, confiável e segura. Por outro lado, a Web pertence à camada de aplicações que opera sobre a Internet cuja principal função é oferecer uma interface que transforme as informações que fluem pela Internet em algo útil. Ao longo do tempo, a Web passou e continua a passar por várias etapas evolucionárias, identificadas como Web 1.0, Web 2.0 e Web 3.0.

- **Web 1.0 - passado:** esta primeira etapa foi inventada por Tim Berners Lee em 1989[17]. Nesta fase surgiram os principais conceitos que conhecemos da Internet atual: *Uniform Resource Locator (URL)*, *HyperText Markup Language (HTML)* e o *HyperText*

*Transfer Protocol* (HTTP). Mais tarde, em 1998 foi criado por Larry Page e Sergey Brin o Google que criou simplicidade nas pesquisas na Web[18].

- **Web 2.0 - presente:** a Web cresceu muito e muito rapidamente. Atualmente é considerada a versão mais próxima da visão de Tim Berners Lee (colaborativa), usado como meio de interação, comunicação global compartilhamento de informação.
- **Web 3.0 - futuro:** para o futuro prevê-se que os conteúdos *online* possam vir a estar organizados de forma semântica, muito mais personalizados para cada utilizador, sites, aplicações inteligentes e/ou publicidade baseada nas pesquisas e nos comportamentos.

A primeira evolução real da Internet foi o aparecimento do IoT, que a transformou em algo sensorial, através da medição de diferentes parâmetros, como por exemplo a temperatura, a pressão, as vibrações, a iluminação, a humidade, o *stress*, entre outras. Através destes parâmetros poderão surgir novas aplicações com elevado potencial para melhorar significativamente a forma como a sociedade vive, aprende, trabalha e se diverte.

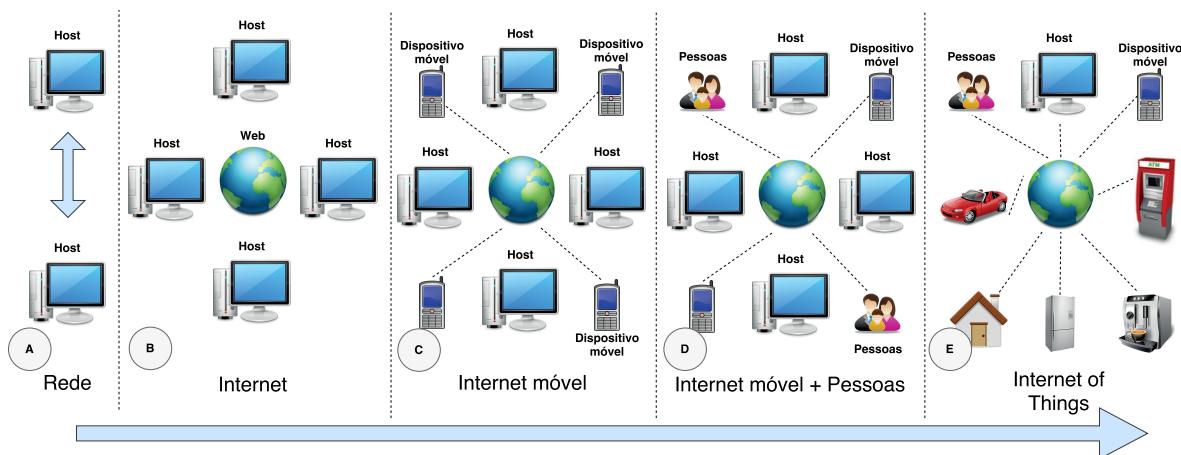


Figura 2.3: Evolução da Internet em cinco fases (Adaptado de [19])

A figura 2.3 representa a evolução da Internet em cinco fases. Inicialmente surgiu a conexão entre dois computadores que permite a criação de uma rede (A da figura 2.3), posteriormente nasce o conceito de WWW ligando um grande número de computadores entre si (B da figura 2.3). Seguidamente, surgiu a Internet móvel (C da figura 2.3) que permitiu ligar dispositivos móveis à Internet, possibilitando a ligação da sociedade através das redes sociais (D da figura 2.3). Finalmente, a Internet está a evoluir para o IoT, permitindo ligar objetos do quotidiano ao sistema global de redes de computadores (E da figura 2.3)[19].

Uma das principais vantagens do IoT é permitir ligar qualquer objeto eletrónico à Internet passando este a ser controlado e gerido através da Internet. O volume de dados gerado por este tipo de ligação pode ser interpretado pelo modelo *Data-Information-Knowledge-Wisdom* (DIKW)[20]. Este modelo, também conhecido como pirâmide do conhecimento (figura 2.4),

é uma hierarquia informacional utilizada especialmente nas áreas da ciência da informação e na gestão do conhecimento, onde cada camada acrescenta certos atributos sobre a anterior.

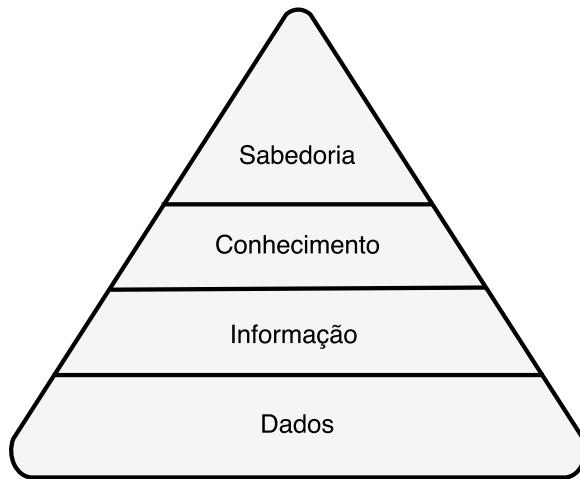


Figura 2.4: Pirâmide do conhecimento: modelo DIKW

A ligação dos objetos à Internet acarreta benefícios visíveis à nossa sociedade, possibilitando um maior controlo e entendimento de como os sistemas interagem entre si e proporcionando uma melhor qualidade de vida a todos. Embora as vantagens se sobreponham às desvantagens não nos podemos esquecer que existem alguns problemas a nível segurança, privacidade, legislação e identidade.

## 2.5 Considerações finais

Como vimos neste capítulo, as propriedades alimentares e terapêuticas da Salicórnia têm conduzido a um elevado interesse económico e ao aumento do seu desenvolvimento comercial. Existem inúmeras empresas a cultivar esta espécie para que possa ser comercializada para os mais diversos fins, sendo que grande parte já é exportada. Uma vez que a Salicórnia carece de um controlo minucioso de certos parâmetros durante o seu cultivo, sendo estes ainda estudados pelos biólogos, existe necessidade de criar um sistema de monitorização, permitindo assim melhorar e conhecer as melhores condições de produção desta espécie.

O universo do IoT pode ser aplicado neste contexto, uma vez que possibilitará a interligação de equipamentos eletrónicos que melhorem a eficiência de produção de Salicórnia através da colocação de sensores e atuadores no campo.



# Estado da arte

Neste capítulo, são apresentados os resultados da pesquisa bibliográfica sobre as ferramentas tecnológicas que poderão ser utilizadas no decorrer do trabalho prático desta dissertação. Serão apresentadas algumas das tecnologias mais populares e realizada uma comparação entre estas. Por fim, será apresentado e descrito motivo pelo qual serão utilizadas.

## 3.1 Conceitos tecnológicos

Nesta secção são apresentados alguns conceitos tecnológicos ou ferramentas utilizadas em toda a dissertação.

- HTML: consiste numa linguagem de marcação que permite estruturar uma página *web*;
- *Cascading Style Sheets* (CSS): permite personalizar os componentes existentes no HTML, por exemplo, personalizá-los com uma determinada cor, espaçamento ou fonte;
- *JavaScript* (JS): é uma linguagem de programação interpretada. Foi utilizada inicialmente para a criação de aplicações do lado do cliente, sendo que atualmente, já é bastante utilizada do lado do servidor, como por exemplo o node.js;
- Python: é uma linguagem de programação de alto nível, possuindo os seguintes paradigmas: orientação a objetos, programação imperativa, programação funcional;
- API: consiste numa interface de programação de aplicações, isto é, disponibiliza um conjunto de funções específicas que permitem ao programador aceder a determinadas funcionalidades de um sistema.

## 3.2 Sistema de Gestão de Base de Dados

Um Sistema de Gestão de Base de Dados (SGBD) consiste num *software* ou conjunto de *softwares* responsáveis pela manipulação de uma base de dados, permitindo a realização de um conjunto enorme de operações, como por exemplo, operações básicas de manipulação de registos e respetiva visualização, operações estatísticas e respetivos relatórios, automatização de funcionalidades entre outros. Seguidamente são apresentados três SGBD mais populares e a sua respetiva comparação.

### 3.2.1 MySQL

O MySQL é considerado o SGBD *open-source* mais popular do mundo, sendo líder em aplicações baseadas na *web*, tais como o Facebook, Twitter, YouTube, Yahoo! e muitos outros. Está comprovado que possui um elevado desempenho, fiabilidade e facilidade de utilização[21].

### 3.2.2 SQL Server

O SQL Server é um SGBD desenvolvido pela Microsoft com bastante popularidade, tendo suporte para as seguintes linguagens de programação: C++, PHP, Java, JS, Python entre outras. A última versão deste SGBD é o SQL Server 2016, que também está disponível para ambientes Linux[22].

### 3.2.3 PostgreSQL

O PostgreSQL é um SGBD do tipo objeto-relacional uma vez que permite um modelo de dados orientado a objetos, isto é, possibilita a manipulação de objetos, classes e heranças diretamente no esquemas da base de dados. Segundo o site oficial do PostgreSQL este é considerado um SGBD bastante poderoso e com desenvolvimento *open-source*[23].

### 3.2.4 Comparação e solução adotada

Na tabela 3.1 encontra-se a classificação atribuída pela *Ranking DB-Engines* para os SGBD estudados de acordo com a sua popularidade, nos meses de Junho e Julho de 2017 e Julho de 2016[24]. O MySQL é considerado o mais popular dos estudados, seguindo-se o SQL Server e por fim o PostgreSQL. Uma vez que o cliente que idealizou este sistema pretende que sejam utilizadas tecnologias sem qualquer custo adicional (sendo mais à frente referido nos requisitos não funcionais), excluiu-se desde logo o SQL Server da Microsoft. Restam então duas tecnologias estudadas, o PostgreSQL e o MySQL.

	<b>Julho 2017</b>	<b>Junho 2017</b>	<b>Julho 2016</b>
<b>MySQL</b>	2º lugar	2º lugar	2º lugar
<b>SQL Server</b>	3º lugar	3º lugar	3º lugar
<b>PostgreSQL</b>	4º lugar	4º lugar	5º lugar

Tabela 3.1: *Ranking BD-Engines* para popularidade dos SGBD estudados[24]

Embora as diferenças entre estes os dois SGBD não sejam significativas, podemos também ter em consideração a performance. Para tal, foram realizados alguns estudos que pretendem comparar estas duas ferramentas usando a *benchmark* TPC-H<sup>1</sup>, expondo que a performance do PostgreSQL é ligeiramente superior à do MySQL na maioria das *queries*[25]. A tabela 3.2 permite comparar algumas das características dos SGBD estudados, possibilitando concluir que todas usam o modelo relacional mas apenas o MySQL e o PostgreSQL são *open-source*.

	<b>MySQL</b>	<b>SQL Server</b>	<b>PostgreSQL</b>
<b>Modelo</b>	Relacional		
<b>Desenvolvido</b>	Oracle	Microsoft	PostgreSQL Global Development Group
<b>Lançamento</b>	1995	1989	1989
<b>Implementação</b>	C e C++	C++	C
<b>Licença</b>	Open-source	Comercial	Open-source
<b>Versão atual</b>	5.7.18 Abril 2017	SQL Server 2016 Junho 2016	9.6.3 Maio 2017
<b>Triggers</b>	Suporta		
<b>Stored Procedures e Functions</b>	Suporta		

Tabela 3.2: Comparação entre MySQL, SQL Server e PostgreSQL

### 3.3 Desenvolvimento web

No que diz respeito ao desenvolvimento *web* do sistema, permitindo que este seja adaptado ao cliente/empresa conforme as suas necessidades, poderão ser adotadas duas estratégias distintas:

- Por manipulação local utilizando JS que interage com o Modelo de Objeto de Documento (DOM). Exemplo de *frameworks*: Angular.JS, Angular 2, React.JS, Vue.JS;
- Por acesso ao servidor que servirá conteúdos criados em função dos pedidos do cliente. Exemplo de *frameworks*: ASP.net, Django, Flask, Play, Spring.

De modo a facilitar o desenvolvimento do sistema, optou-se que tanto a nível de *frontend* como de *backend* este seja realizado paralelamente. Posto isto, será utilizada uma *framework*

<sup>1</sup><http://www.tpc.org/tpch/>

em que o servidor disponibilizará os conteúdos em função das necessidade do cliente. Esta estratégia é mais eficiente e com tempos de resposta mais curtos, relativamente à primeira apresentada. De seguida, são apresentadas três *frameworks* bastantes populares e com características distintas.

### 3.3.1 ASP.net

ASP.net é uma tecnologia de *scripting* que corre do lado do servidor, permitindo colocar numa página *web*, *scripts* que irão ser executados por um determinado servidor. Uma aplicação desenvolvida nesta tecnologia é executada num servidor de Internet da Microsoft, conhecido por *Internet Information Services* (IIS), que pode ser desenvolvida recorrendo à linguagem C# ou Microsoft Visual Basic. Esta tecnologia oferece três modelos para a criação de aplicações *web*, ASP.NET Web Forms, ASP.NET Model-View-Controller (MVC), e ASP.NET Web Pages, podendo ser criadas aplicações *web* estáveis e maduras com qualquer um deles[26].

### 3.3.2 Flask

O Flask consiste numa *micro-framework* desenvolvida em Python baseada nas bibliotecas *Web Server Gateway Interface* (WSGI) Werkzeug e Jinja2<sup>2</sup>. Esta *framework* disponibiliza um modelo simples para desenvolvimento *web*, uma vez que permite economizar bastante tempo na sua conceção. O Pinterest e LinkedIn são exemplos de aplicações que utilizam esta *micro-framework*. As suas características principais são[27]:

- Possui um servidor para desenvolvimento e *debug*;
- Suporta testes unitários nativamente;
- Usa modelos da biblioteca Jinja2;
- Suporta *cookies* seguros (sessões do lado do cliente);
- É compatível com a primeira versão do WSGI;
- É baseado em unicode;
- Possui uma extensa documentação.

### 3.3.3 Django

A *framework* Django é uma ferramenta de desenvolvimento *web* de alto nível desenvolvida na linguagem Python, que possui uma arquitetura MVC[28], embora com algumas diferenças em relação a esta. No modelo MVC, a componente *Model* diz respeito à camada de acesso a dados, a *View* à parte do sistema responsável pela escolha e apresentação da informação e o

---

<sup>2</sup><http://jinja.pocoo.org/> (motor de modelos mais utilizado em Python)

*Controller* está encarregue de decidir que *views* escolher (figura 3.1). Os criadores do Django consideram a sua arquitetura com sendo *Model-Template-View* (MTV)[29].

- *Model*: é responsável por manipular, valiar, aceder e relacionar os dados. É considerada a fonte única e definitiva de informação (dados), possuindo os campos essenciais e os comportamentos sobre estes. Geralmente, cada *Model* mapeia uma única tabela na base de dados;
- *Template*: é responsável pela apresentação dos dados;
- *View*: é responsável pela lógica de negócio e respetiva manipulação, acedendo aos *Models* e encaminhando a informação pretendida para os *Templates*.

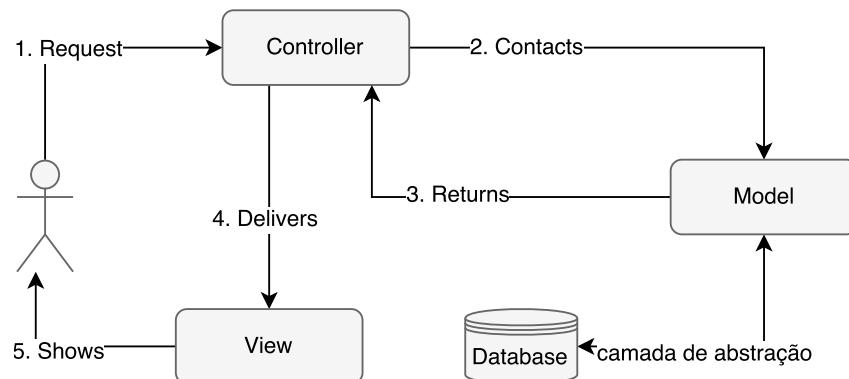


Figura 3.1: Modelo MVC (Adaptado de [30])

Um dos pontos mais interessantes no Django é sem dúvida a existência de *Object-Relational Mapping* (ORM). Consiste numa técnica de desenvolvimento que permite modelar os dados através de classes em Python. Através dela, é possível gerar tabelas na base de dados e manipulá-las sem a necessidade de interagir diretamente com *Structured Query Language* (SQL) (embora também seja possível).

De acordo com as descrições dos autores, esta ferramenta possui várias vantagens, entre as quais se destaca, uma boa documentação, facilidade, rapidez no desenvolvimento e respetivo *deployment*. Para além disso, é bastante estável e escalável[29].

### 3.3.4 Conclusões e solução adotada

Uma vez que o cliente do sistema pretende que não existam gastos adicionais, optou-se por utilizar ferramentas *open-source*. Esta opção permitiu desde logo excluir a tecnologia ASP.net.

A escolha para desenvolvimento *web* recaiu sobre o Django uma vez que esta *framework* recomenda a utilização do PostgreSQL como SGBD, indicando que permite alcançar um bom equilíbrio entre custo, características, rapidez e estabilidade[31].

## 3.4 Desenvolvimento *mobile*

Uma aplicação móvel, vulgarmente denominada por *app mobile*, consiste num determinado *software* utilizado para realizar funções específicas em dispositivos móveis, como *smartphones* ou *tablets*. Este tipo de *software* permite aos seus utilizadores facilitar o desempenho de determinadas atividades à distância de um simples toque no seu dispositivo.

Relativamente ao desenvolvimento de aplicações móveis, o programador pode recorrer a duas estratégias: desenvolvimento nativo ou multi-plataforma. Esta diferenciação centra-se na forma como o código é produzido, sendo seguidamente apresentadas algumas vantagens e desvantagens de cada um.

### 3.4.1 Plataformas nativas

Uma aplicação nativa é desenvolvida exclusivamente para ser utilizada por um dispositivo ou plataforma específica, como é o caso do iOS, Android ou Windows Phone. Nestes casos, são usadas ferramentas e linguagens específica suportadas pela plataforma, podendo interagir com as funcionalidade do sistema operativo ou aplicações existentes. Seguidamente são apresentadas algumas características de uma plataforma do tipo nativo[32].

- **Performance:** acarreta benefícios ao nível da performance, uma vez que as aplicações nativas são desenvolvidas em código específico e otimizado;
- **Desenvolvimento:** tem que ser desenvolvido de raiz para cada dispositivo, exigindo um maior investimento de tempo.
- **Interface:** podem ser utilizadas as funcionalidades da *User Interface* (UI) dos plataformas em questão.
- **Recursos disponíveis:** apenas são disponibilizados os recursos de cada plataforma, existindo APIs que permitem o seu acesso.
- **Plug-ins:** os *plug-ins* são específicos para cada plataforma, dificultando a sua integração para aplicações multi-plataforma.
- **Segurança:** recorre a APIs nativas, utilizando os protocolos de segurança conhecidos da plataforma.

### 3.4.2 Multi-plataforma

Uma aplicação multi-plataforma ou híbrida, é desenvolvida utilizando tecnologias *web*, como por exemplo o HTML, CSS e JS. Este tipo de aplicações utiliza uma funcionalidade denominada por *WebView* permitindo que o código *web* seja adaptado a uma aplicação respetiva para uma determinada plataforma. De seguida, são apresentadas algumas características desta estratégia de desenvolvimento[32].

- **Performance:** relativamente à performance, não são dados pontos positivos no que toca às aplicações híbridas, uma vez que estas usam WebViews e outros *plug-ins* que permitem abstrair o acesso às APIs nativas.
- **Desenvolvimento:** são utilizadas tecnologias *web* (HTML, CSS e JS), sendo mais eficazes em termos de custo uma vez que permite reutilizar código para diferentes dispositivos.
- **Interface:** para aceder à mesma UI do sistemas operativos é necessário utilizar *plug-ins* adicionais.
- **Recursos e Plug-ins:** existe muitos recursos disponíveis para aplicações multi-plataforma através de *plug-ins*, permitindo ter o mesmo efeito em todas as plataformas. Por outro lado, impõe-se a questão da performance.
- **Segurança:** recorrem a *plug-ins* externos para garantir maior segurança podendo ser potencialmente mais vulneráveis.

Existem inúmeras *frameworks* para desenvolvimento multi-plataformas, entre as quais destaco as três mais utilizadas no mercado: Ionic, PhoneGap e Xmarin.

- **Ionic:** consiste numa ferramenta para desenvolvimento *mobile*, sendo constituído por outro conjunto de *frameworks*: Cordova (permite a integração de recursos nativos), AngularJS (criação da *web app*) e Ionic Module e CLI (ferramenta e componentes disponibilizados pelo Ionic)[33].
- **PhoneGap:** tal como o Ionic, consiste numa *framework* *mobile* que utiliza o Cordava da Apache para aceder aos recursos nativos do sistema. É uma *framework open-source*, tendo compatibilidade com outras ferramentas.
- **Xmarin:** permite o desenvolvimento *mobile* a partir do .NET Framework, utilizando C#, possibilitando que o compilador gere código nativo disponível para qualquer tipo de plataforma. Uma das principais desvantagens do Xmarin prende-se com a necessidade de desenvolver a UI para todas as plataformas necessárias.

### 3.4.3 Conclusões e solução adotada

Um vez que o cliente pretende uma solução *mobile* multi-plataforma, isto é, possível de ser executada em iOS e Android optou-se por utilizar a *framework* Phonegap. Uma das principais desvantagens da utilização deste paradigma multi-plataforma é o seu baixo desempenho na utilização de recursos do dispositivo. Uma vez que não será necessária essa utilização optou-se por utilizar a *framework* mencionada anteriormente.

### 3.5 REST *Frameworks*

Com o evolução da tecnologia, houve a necessidade dos programadores exporem os seus serviços para serem utilizados por outras entidades. Neste sentido, foram criados métodos que permitem a comunicação entre diferentes sistemas, podendo estes serem alojados em máquinas ou redes diferentes. Para atingir este objetivo, existem atualmente algumas arquiteturas bastante populares, como é o caso do *Simple Object Access Protocol* (SOAP) e o REST.

- **SOAP:** consiste num protocolo aplicado a ambientes distribuídos e descentralizado, isto é, permite que dois processos possam comunicar estando a ser executados em máquinas ou redes diferentes[34]. A comunicação entre os diferentes componentes é realizada com base em mensagens *Extensible Markup Language* (XML) (formato *standard*), enviadas entre os intervenientes através HTTP.
- **REST:** é considerada uma alternativa à arquitetura SOAP, sendo menos formal relativamente a esta. Usa o protocolo HTTP como modo de funcionamento, tendo a possibilidade de definir o formato das mensagens a trocar, sendo normalmente utilizados os formatos *JavaScript Object Notation* (JSON) ou XML. Tem como grande vantagem a facilidade de implementação, escalabilidade, maior performance na execução e sem a complexidade dos *web-services* tradicionais [54].

Dadas as vantagens enumeradas relativamente à tecnologia REST, optou-se por utilizá-la. Seguidamente serão apresentadas algumas *frameworks* bastante populares no desenvolvimento de APIs REST e a sua respetiva comparação.

#### 3.5.1 Django Rest Framework

O Django REST Framework é considerado pelos seus autores uma ferramenta poderosa e flexível para a construção de APIs Web[35], sendo de fácil incorporação quando estamos perante um projeto desenvolvido com a *framework* Django. Esta ferramenta disponibiliza um conjunto de funcionalidade que permite estender as suas características de uma forma simples e intuitiva, tais como paginação de conteúdos e utilização de filtros, escolha dos formatos de dados, entre outras.

Uma das vantagens desta *framework* é a capacidade de construção do serviço através de vistas baseadas em classes, permitindo de um modo quase automático fazer a construção da API através dos modelos disponíveis, não sendo necessário para implementar a lógica de cada um dos *endpoints*<sup>3</sup> disponibilizados, existindo a possibilidade de personalizar a lógica de negócio das rotas. Além do mais, esta *framework* disponibiliza uma extensa documentação bem detalhada e explicativa, possuindo ainda uma grande comunidade que a apoia.

---

<sup>3</sup>Endereço ou ponto de conexão a um serviço web

### 3.5.2 Restlet

O Restlet consiste numa *framework* de desenvolvimento REST *open-source* para a linguagem Java, disponível para a implementação de servidores como de clientes, através da mesma API, reduzindo assim a curva de aprendizagem dos diferentes componentes do sistema.

Esta solução possui um mecanismo de extensões que permite incluir bibliotecas de Java externas com o objetivo de suportar propriedades das bases de dados, *templates*, segurança, e até outros tipos de serialização de conteúdos. Esta *framework* permite a autenticação através de diversos métodos, entre os quais, a autenticação básica em HTTP, Amazon S3 ou OAuth2.0.

### 3.5.3 Flask-RESTful

O Flask-RESTful é uma extensão para *microframework* Flask. É considerada simples e de fácil utilização, permitindo o desenvolvimento de serviços de forma rápida e leve através da linguagem Python. É ideal para o desenvolvimento de serviços simples e extensíveis, sem necessidade de configurações acrescidas de ORM, administração, autenticação, entre outros[36]. Adicionalmente, esta extensão permite abstrair as tarefas relativas ao protocolo HTTP. No entanto, o seu desenvolvimento é independente da lógica de negócio do serviço, possibilitando ao seu criador a liberdade de implementar o serviço tendo em conta as suas preferências e necessidades.

### 3.5.4 Conclusões e solução adotada

	Django Rest Framework	Restlet	Flask-RESTful
<b>Linguagem</b>	Python	Java	Python
<b>Serialização</b>	JSON, XML, YAML, HTML	JSON, XML, CSV, YAML	JSON
<b>Autenticação</b>	Básica, Token, Sessão, OAuth, OAuth2.0	Básica, Digest, AmazonS3, OAuth2.0	Não
<b>Cache</b>	Sim	Não	Não
<b>HATEOAS</b>	Sim	Não	Não
<b>Documentação</b>	Intuitiva e simples detalhada e com exemplos	Bastante extensa e detalhada	Detalhada e com alguns exemplos

Tabela 3.3: Comparação entre *frameworks* REST estudadas.

Existem inúmeras características que potenciam cada uma das ferramentas/*frameworks* abordadas anteriormente, entre as quais destaco a *Hypermedia As The Engine Of Application State* (HATEOAS). Esta característica determina se a API é navegável, isto é, a partir de um determinado recurso é possível descobrir quais os recursos associados e a sua respetiva

informação. Na tabela 3.3 apresenta-se uma comparação entre as ferramentas/*frameworks* discutidas anteriormente.

O Django REST Framework é a solução que apresenta uma maior maturidade, disponibilizando um conjunto de funções aos programadores, possibilitando boas práticas de desenvolvimento REST, aproveitando as vantagens da *framework* Django enumeradas anteriormente (secção 3.3.3). Para além disso, permite a autenticação da API, possibilitando que apenas alguns utilizadores tenham acesso aos serviços.

## 3.6 Microcontroladores

Um microcontrolador consiste numa solução integrada de um sistema computacional, num único dispositivo físico, sendo uma mistura de *hardware* com *software*. Possui vários módulos principais, um *Central Processing Unit* (CPU) onde é realizado todo o processamento, as memórias de instrução e dados (*Random Access Memory* (RAM) e *Read-Only Memory* (ROM)) e os portos de entrada e saída (*Input/Output* (I/O)), e ainda alguns periféricos (*Timer*, *Serial COM Port*).

Estes dispositivos são responsáveis pelo processamento de dados, tomada de decisões, controlo do funcionamento de alguns módulos, visualização e conversão da informação recolhida pelos sensores, entre outros. Com o objetivo de simular este sistema, serão utilizados dois microcontroladores bastante comuns no mercado, um Arduino Nano e um Raspberry Pi 3.

### 3.6.1 Arduino Nano

Como descrito no site oficial, um Arduino consiste numa plataforma *open-source* de prototipagem eletrónica composta por *hardware* e *software* flexíveis e com elevada facilidade de utilização. Esta plataforma é utilizada para projetos principalmente no contexto do IoT e da robótica educativa, sendo possível incorporar vários módulos, dependendo da tarefa que se quer executar[37].

O Arduino Nano possui um conjunto de pinos que podem ser programados para funcionarem como entradas ou saídas fazendo com que este microcontrolador interaja com o exterior para os mais diversos fins. Para além dos pinos de I/O existem pinos de alimentação que fornecem diversos valores de tensão que podem ser utilizados para transmitir energia elétrica aos diferentes componentes de um projeto[37]. Nas figuras 3.2 e 3.3 apresenta-se o Arduino utilizado e a identificação dos diferentes pinos existentes, respetivamente. Adicionalmente, na tabela 3.4 encontram-se as principais características desta versão do Arduino.

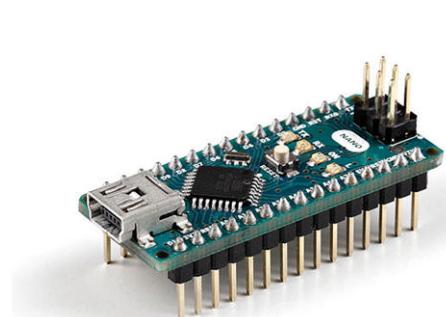


Figura 3.2: Arduino Nano

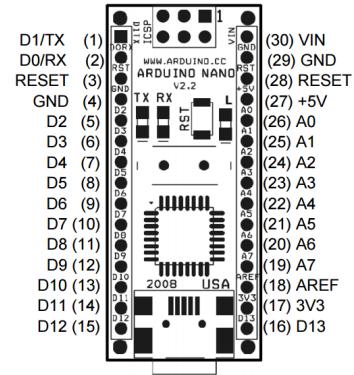


Figura 3.3: Identificação dos diferentes pinos num Arduino Nano (Retirado de [38])

<b>Microcontrolador</b>	ATmega328
<b>Tensão de operação</b>	5V
<b>Tensão de entrada</b>	7-12V
<b>Portas digitais</b>	14 (6 podem ser usadas como PWM)
<b>Portas analógicas</b>	8
<b>Corrente nos pinos I/O</b>	40mA
<b>Memória Flash</b>	32KB (2KB usado no bootloader)
<b>Memória RAM (SRAM)</b>	2KB
<b>EEPROM</b>	1KB
<b>Velocidade do Clock</b>	16MHz
<b>Dimensões</b>	45 x 18mm
<b>LED interno</b>	Pino digital 13
<b>Ligaçāo Universal Serial Bus (USB)</b>	Ligaçāo ao computador e alimentação

Tabela 3.4: Características principais do Arduino Nano (Adaptado de [39])

### 3.6.2 Raspberry Pi 3

Um Raspberry Pi (figura 3.4) é mais poderoso do que outros dispositivos de igual dimensão e incorpora alguns periféricos I/O, tal como um computador normal. Tem o tamanho de um cartão de crédito que possui um conjunto de *hardware* integrado que tal como o Arduino, possibilita uma interação com o exterior. Este dispositivo, desenvolvido no Reino Unido pela *Raspberry Pi Foundation*, pode ser ligado a um monitor através da saída HDMI, possuindo também uma saída de áudio e várias portas USB. O Raspberry Pi é compatível com sistemas operativos baseados em GNU/Linux, sendo que no trabalho prático desta dissertação será utilizada a versão 3 do Raspberry Pi com a versão Raspbian<sup>4</sup> instalada[40].

Na figura 3.5 encontram-se os principais componentes da versão utilizada e na tabela 3.5 é

<sup>4</sup>Distribuição Linux oficial do Raspberry Pi: <https://www.raspberrypi.org/downloads/raspbian/>

feita uma comparação entre a versão 2 e 3 do Raspberry Pi, permitindo concluir que a versão utilizada é mais poderosa.



Figura 3.4: Raspberry Pi 3

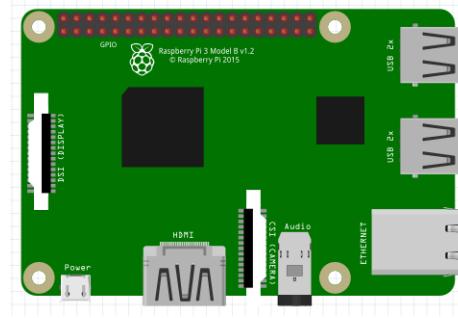


Figura 3.5: Principais componentes de um Raspberry Pi 3

	Raspberry Pi 2 Model B 1.2	Raspberry Pi 3 Model B
<b>CPU</b>	QUAD Core 900MHz	QUAD Core 1.2GHz
<b>RAM</b>	1GB SDRAM	1GB SDRAM
<b>Potência máxima</b>	5V/1.8A	5V/2.5A
<b>Armazenamento</b>	MicroSD	MicroSD
<b>USB 2.0</b>	4 x Portas USB	4 x Portas USB
<b>GPIO</b>	40 pinos	40 pinos
<b>Porta Ethernet</b>	Sim	Sim
<b>Wifi</b>	Incorporado (versão 802.11n)	Não
<b>Bluetooth LE</b>	Incorporado (versão 4.1)	Não

Tabela 3.5: Comparação entre algumas características da versão 2 e 3 do Raspberry Pi

## 3.7 Sensores

Esta secção tem como objetivo fazer um estudo comparativo entre as diferentes tecnologias usadas para a medição dos vários parâmetros ambientais necessários ao controlo e monitorização da produção de Salicornia. Serão descritos alguns sensores de temperatura, luminosidade e salinidade existentes no mercado.

### 3.7.1 Sensor de temperatura

Existem vários tipos de sensores de temperatura baseados em princípios de funcionamento distintos, nomeadamente os termopares, os termístores e os de circuito integrado.

- **Termopares:** são sensores de temperatura simples, robustos e de baixo custo, constituídos por duas partes diferentes de material condutor. Podem medir temperaturas entre os -200°C e os 2315°C, sendo usados em grande escala[41];

- **Termíster:** são sensores cuja resistência varia com a temperatura, sendo construídos a partir de materiais semicondutores. Um termíster detém uma maior sensibilidade, pois uma pequena variação de temperatura provoca uma grande variação na sua resistência, permitindo que a temperatura típica seja entre os -100°C e os 300°C. Estes sensores são frágeis, baratos e de dimensões reduzidas, sendo suscetíveis a problemas de auto aquecimento[42];
- **Circuito integrado:** são sensores construídos através de materiais semicondutores, o que possibilita que tenham uma gama de temperatura limitada, geralmente entre os -55°C e os 150°C. Estão disponíveis com saídas em tensão, ou corrente, linearmente proporcional à temperatura.

### 3.7.2 Sensor de luminosidade

Existe uma enorme variedade de sensores de luminosidade/radiação no mercado, contudo para monitorizar a luminosidade incidente numa planta é fundamental conhecer a radiação que é utilizada no processo de fotossíntese, denominada por *Photosynthetically Active Radiation* (PAR). A figura 3.6 ilustra o espectro de absorção de luz pelas plantas. É possível ver que estas são mais sensíveis à luz azul e vermelha (300-700 nm).

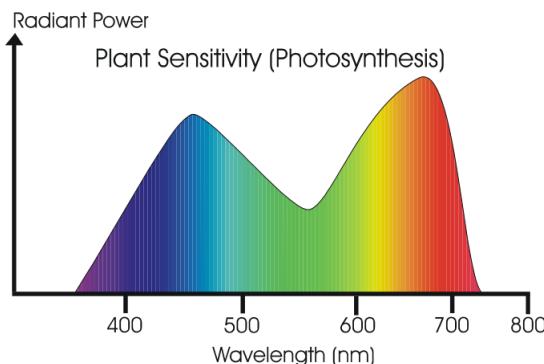


Figura 3.6: Sensibilidade luminosa nas plantas durante a fotossíntese (Retirado de [43])

Alguns dos sensores de luminosidade/radiação mais comuns no mercado são os sensores PAR, os piranómetros e os sensores optoelectrónicos, descritos de seguida.

- **Sensores PAR:** são desenhados especificamente para medir a PAR, sendo indicados para medir a luz incidente numa planta. Os custos associados a este tipo de sensores são elevados e por isso são utilizados maioritariamente na investigação hortícola;
- **Piranómetros:** estes sensores medem a radiação solar total (radiação ultravioleta, visível e infravermelha), devendo ser utilizados no exterior. Este tipo de sensores é particularmente útil em estações meteorológicas. No entanto, são bastante dispendiosos, podendo ser mais caros dos que os sensores PAR;

- **Sensores optoelectrónicos:** são dispositivos eletrónicos que se baseiam em fenómenos fotovoltaicos. Existem três sensores deste tipo, que possuem um elevado desempenho quando afetados pelo ruído e uma elevada sensibilidade. São considerados os sensores de luminosidade mais económicos.
  - **Fotodíodo:** pode ser visto como um diodo comum, gerando uma corrente ou tensão quando incide radiação sobre ele;
  - **Fototransístor:** é um transístor desenhado para receber luz, normalmente num módulo transparente;
  - **Light Dependent Resistor (LDR):** trata-se de um material semicondutor cuja resistência diminui com a incidência de luz. É um sensor económico e de fácil utilização.

### 3.7.3 Sensor de salinidade

Atualmente ainda não se encontra disponível nenhum sensor que permita medir exatamente a salinidade. Contudo, existem sensores que medem a condutividade elétrica num determinado meio, permitindo concluir o estado do meio relativamente à quantidade de sal existente. Um exemplo disso, é o sensor SAL-BTA da Vernier[44] que permite medir de forma fácil e precisa o teor de sal dissolvido numa solução aquosa.

## 3.8 Tecnologias de comunicação

Nesta secção serão apresentadas algumas das tecnologias de comunicação sem fios mais utilizadas em *Internet of Things* que permitem a troca de informação entre dispositivos. Serão abordados quatro tecnologias bastante populares atualmente: Zigbee, Bluetooth, Wi-Fi e SigFox. Por fim, é realizada uma comparação entre as tecnologias estudadas.

### 3.8.1 Zigbee

Zigbee é um padrão de rede sem fios destinado a aplicações de controlo e sensores remotos, adequado para operações em ambientes isolados. Esta tecnologia é de curto alcance, baixa complexidade e consumo de energia, possuindo ainda uma baixa taxa de dados. Permite uma topologia de rede *mesh* (malha) permitindo elevados níveis de fiabilidade e maior alcance de cobertura, fornecendo mais do que um caminho através da rede para qualquer ligação[45].

### 3.8.2 Bluetooth (BLE)

Bluetooth (BLE) é uma marca característica da versão 4.0 do Bluetooth, sendo projetado para aplicações de baixa potência, utilizada em comunicações de curta distância. Esta tecnologia de comunicação bastante conhecida, consiste numa especificação de rede sem fios

de âmbito pessoal, também conhecida por *Wireless personal area networks* (PANs), sendo de baixo custo e consumo de energia. O Bluetooth surgiu com a intenção de eliminar o elevado número de cabos existentes na comunicação entre dispositivos, sendo que este padrão funciona na banda de frequência 2,4GHz, conhecida como banda *Industrial, Scientific, Medical* (ISM)<sup>5</sup>, disponível mundialmente sem a necessidade de licenças[46][47].

### 3.8.3 Wi-Fi (IEEE 802.11)

O Wi-Fi é uma tecnologia de recepção de transmissão de dados sem fios, baseado no *standard* IEEE 802.11. Com o evoluir da tecnologia, têm surgido diferentes variações do mesmo *standard*, sendo a variante 802.11g a mais comum. Seguidamente, são apresentadas as características principais desta variante[48][49]:

- Banda de frequência de 2.4 GHz;
- Taxa de bits máxima de 54 Mbps na camada física, transferência de dados máxima de 24.7 Mbps;
- Topologia em estrela (infraestrutura) e ponto-a-ponto (ad-hoc);
- Vários mecanismos de encriptação e autenticação.

### 3.8.4 Sigfox

A tecnologia Sigfox permite a comunicação sem fios entre dispositivos possuindo um baixo consumo de energia, utilizando a banda ISM, tal como o Bluetooth. De acordo com o site oficial, o Sigfox é considerado um protocolo leve, permitindo trocas de pequenas mensagens possibilitando um consumo de bateria bastante reduzido[50]. Esta tecnologia é totalmente virada para o IoT devido ao seu baixo consumo.

### 3.8.5 Comparação entre tecnologias de comunicação

Na tabela 3.6 é apresentada uma comparação entre algumas características das tecnologias anteriormente estudadas. De acordo com os dados apresentados na tabela, concluímos que o Zigbee e o Sigfox são tecnologias de baixo consumo, contudo o alcance destas é inferior relativamente às tecnologias Bluetooth e Wi-Fi.

---

<sup>5</sup>Faixa de frequência compartilhada por dispositivos, como forno micro-ondas e Wi-Fi

	<b>Zigbee</b>	<b>Bluetooth</b>	<b>Wi-fi</b>	<b>Sigfox</b>
<b>IEEE</b>	802.15.4	802.15.1	802.11n	N/A
<b>Banda de frequência</b>	868/915 MHz 2.4 GHz	2,4-5,5 GHz	2,4-5 GHz	800 Hz (europa)
<b>Topologia da rede</b>	Mesh	Estrela	Estrela	P2P
<b>Energia (consumo)</b>	Muito baixo	Baixo	Médio	Muito baixo
<b>Alcance</b>	10m	aprox. 50m	100m	Rural: 10-15m Urbano: 3-5m
<b>Bateria (duração)</b>	Meses-anos	Dias	Horas	10/15 anos

Tabela 3.6: Comparaçāo entre as tecnologias de comunicação abordadas (Adaptado de [45])

# Sistema de controlo e monitorização: arquitetura e modelação

Este capítulo tem como principal objetivo a descrição do sistema que resultou do trabalho prático desta dissertação. Cada elemento do sistema é caracterizado de acordo com as suas funções, especificidades e arquitetura, bem como a forma como os elementos interagem entre si. Para além disso, é apresentado todo o processo de modelação do sistema tendo por base os requisitos do cliente.

## 4.1 Descrição global do sistema

Este sistema tem como objetivo a supervisão remota da produção de Salicórnia, permitindo não só a monitorização dos dados adquiridos pelos sensores, como também a atuação remota de determinados comandos. Neste contexto, também é possível a aquisição de imagens que possibilita a deteção de intrusos nas quintas onde se realiza a produção desta espécie. O esquema da figura 4.1 ilustra de um modo geral todos os componentes e as diferentes plataformas com que o cliente pode interagir.

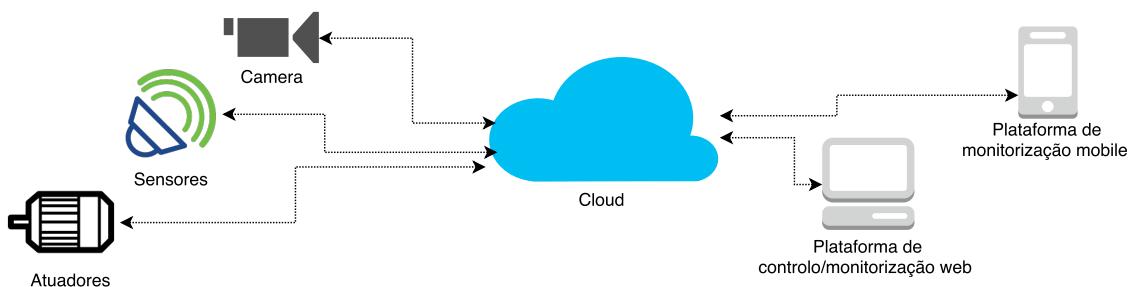


Figura 4.1: Ilustração dos principais componentes do sistema

Como vimos no capítulo 3, uma plantação de Salicórnia carece de um controlo relativamente fino de certos parâmetros ambientais sobretudo da salinidade do terreno, que depende, das chuvas, da salinidade da água dos canais da ria, entre outros. Nas quintas onde se cultiva Salicórnia, a produção faz-se numa espécie de leiras limitadas por pequenos canais de irrigação que podem ser cheios de água salgada proveniente dos esteiros que rodeiam a quinta. Esta operação implica a abertura de válvulas de admissão de água, medida do nível da maré nos canais, monitorização da qualidade e salinidade da água exterior.

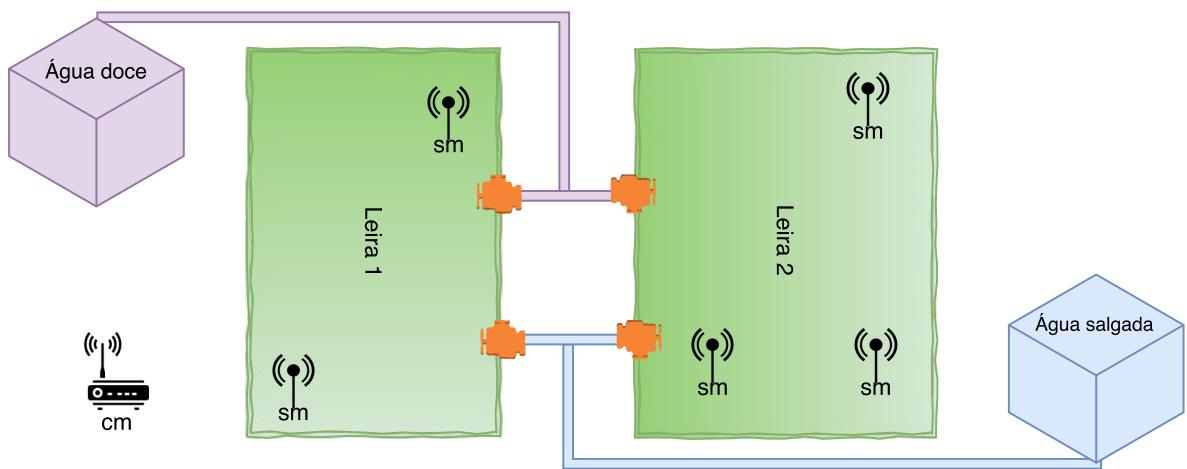


Figura 4.2: Ilustração da distribuição dos módulos em duas leiras

Tal como ilustrado na figura 4.2, foram colocados módulos com sensores distribuídos estrategicamente por cada leira. Cada um desses módulos, irá comunicar com um módulo central originando uma topologia de rede em estrela. Por sua vez, cada um destes módulos centrais irá comunicar diretamente com o servidor que receberá todos os dados adquiridos tanto pelos sensores como por atuadores, permitindo que estes sejam guardados numa base de dados específica. Os atuadores, permitirão despoletar ações que autorizam a ativação ou desativação de bombas e/ou válvulas para transferências de água de modo a melhorar as condições de cultivo da Salicórnia. Os módulos centrais têm acesso à camada protocolar *Transmission Control Protocol (TCP)/Internet Protocol (IP)* (Internet) para possibilitar a utilização da API REST desenvolvida para o efeito.

No que diz respeito às plataformas de interação com o cliente, existe uma *dashboard* (plataforma *web*) e uma aplicação *mobile*. A *dashboard* disponibiliza uma interface que apresenta as informações mais importantes para o utilizador de forma apelativa, tornando mais fácil a sua interação e respetiva leitura, possibilitando ainda a gestão de todo o sistema e realização de operações de controlo remoto. Por outro lado, a aplicação *mobile* permite apenas a monitorização do cultivo da Salicórnia e receção de alertas quando estes ocorrem.

## 4.2 Componentes

No contexto desta dissertação é necessário reter dois conceitos principais, são eles:

- **Sensor Module (SM):** consiste num módulo responsável pela aquisição de dados provenientes dos mais diversos tipos de sensores;
- **Controller Module (CM):** consiste num módulo responsável pela receção dos dados/estados dos sensores do *Sensor Module* e respetivo envio para a *cloud*.

O cenário da figura 4.3 ilustra três *Sensor Modules* que comunicam com um *Controller Module*. Cada um desses *Sensor Module* possui um conjunto específico de sensores, podendo estes ser atuadores ou câmaras. Para a comunicação com o *Controller Module*, cada *Sensor Module* possui um determinado módulo de comunicação que permite a transferência dos dados adquiridos pelos sensores. Posteriormente, o *Controller Module* possui também um determinado módulo de comunicação (TCP/IP) que permite a utilização da API e respetivo envio ou atualização dos dados adquiridos pelo sistema.

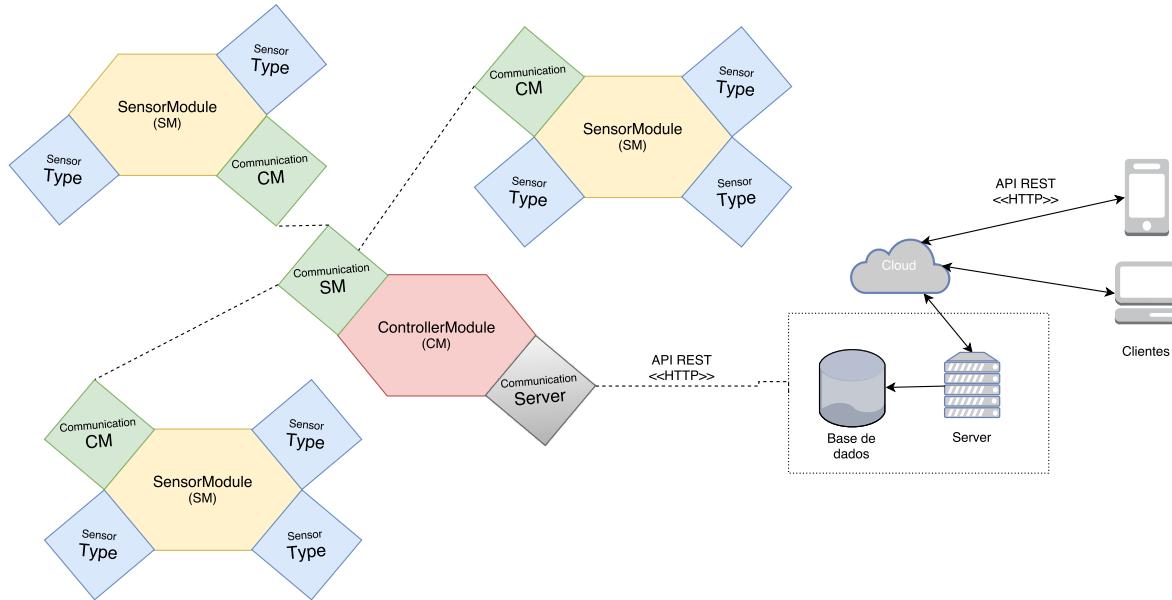


Figura 4.3: Esquema de componentes e respetiva comunicação entre três SM e um CM

Seguidamente serão especificados todos os detalhes de cada módulo, uma vez que serão considerados na modelação de todo o sistema.

### 4.2.1 Sensor Module

Um *Sensor Module* consiste num microcontrolador responsável pela aquisição de dados provenientes dos mais diversos tipos de sensores, podendo estes ser atuadores ou câmaras.

No caso de se tratar de um atuador, isto é, válvulas, bombas, contadores, pás ou cancelas, apenas serão lidos valores binários. Caso se trate de uma câmara, todo o processamento é feito internamente nesta, sendo que o sistema apenas irá receber o IP da mesma ou um URL para um servidor de *Real-Time Messaging Protocol* (RTMP).

Tal como referido anteriormente, cada *Sensor Module* terá que utilizar um determinado módulo de comunicação para possibilitar a transferência dos dados adquiridos para o módulo central. Para além disso, pretende-se que o *Sensor Module* em condições extremas, possa tomar decisões de atuação, isto é, caso seja lido um valor fora do padrão e que seja necessário a ativação de um atuador, este deverá ser auto suficiente em tomar esta decisão, sem necessidade de intervenção do utilizador.

Pretende-se que este módulo seja identificado por um determinado nome, possua uma bateria que permita a sua mobilidade, tenha um ou vários módulos de comunicação acoplados que permitam comunicar com um módulo central. Tenha uma determinada memória e um módulo *Global Positioning System* (GPS) que permita aceder à sua localização, identificando-o em caso de furto. Para além disso, um *Sensor Module* terá que possuir obrigatoriamente um ou vários sensores.

#### 4.2.2 Controller Module

Um *Controller Module* consiste num microcontrolador responsável pela receção dos dados provenientes dos vários *Sensor Modules*. Pretende-se que este módulo envie ou receba informações para os *Sensor Module* quando solicitados pelo utilizador. Após a receção dos dados, estes são enviados para um servidor em *cloud* através de uma API criada para o efeito. Como a tecnologia REST opera sob o protocolo de comunicação HTTP, este componente tem que necessariamente estar ligado à rede Internet (TCP/IP).

É essencial que este módulo possua alguma capacidade de processamento, uma vez que poderá ter vários *Sensor Modules* a si associados e com necessidade de constante envio e receção de dados. Para além disso, pretende-se que o *Controller Module* seja identificado por um determinado nome, tenha um módulo de comunicação que possibilite o envio de dados para um servidor e outros para comunicação com os diferentes *Sensor Modules*. Tal como acontece com os *Sensor Modules*, existe necessidade de acoplado um módulo GPS que permita localizar o microcontrolador em caso de robo.

### 4.3 Análise de requisitos

Após o entendimento da descrição geral do sistema bem como todos os componentes envolvidos, segue a fase de efectuar o levantamento dos requisitos do sistema, levando-nos a entender o que o cliente deseja possibilitando a criação dos processos de negócio necessários ao sistema. Seguidamente apresentam-se os requisitos funcionais e não funcionais deste sistema.

Relativamente à metodologia apresentada na secção 1.3, esta etapa refere-se às duas primeiras fases: Análise da viabilidade e Análise e especificação de requisitos.

#### 4.3.1 Requisitos funcionais

Os requisitos funcionais descrevem os critérios que devem ser usados para avaliar as funções específicas ou os comportamentos de um determinado sistema. Os requisitos funcionais que de seguida se apresentam foram propostos pelos clientes (empresa/investigadores) no contexto deste projeto para as duas plataformas disponíveis: *web (dashboard)* e *mobile*.

##### Aplicação web (*dashboard*)

- A interface do sistema deve permitir que o utilizador, seja ele qual for, entre ou faça *login* no sistema;
- A interface do sistema deve permitir que o utilizador, seja ele qual for, saia ou faça *logout* no sistema;
- O *dashboard* deverá permitir que qualquer utilizador possa recuperar a sua chave de acesso ao sistema (*password*);
- O sistema deve permitir que o utilizador possa adicionar e/ou gerir um ou vários módulos de sensores;
- A aplicação *web* deve permitir visualizar graficamente os dados que cada sensor obtém;
- O sistema deve permitir visualizar tabularmente (*dataset*) os dados que cada sensor obtém;
- Em cada uma das visualizações anteriormente descritas, deve ser possível efetuar uma filtragem por data;
- O sistema deverá permitir a exportação dos dados obtidos pelos sensores em formato *Comma-Separated Values* (CSV);
- O sistema deve ter a capacidade de gerar alarmes quando algum valor lido pelos sensores esteja fora do previsto.

De modo a tornar o sistema genérico foram impostos os seguintes requisitos adicionais:

- O sistema deve permitir que qualquer utilizador se possa registar no sistema, embora tenha que estar obrigatoriamente associado a uma empresa;
- O utilizador comum só terá acesso à sua área reserva após a validação por parte da empresa;

- A *dashboard* deverá permitir ao administrador a adição de novas empresas e a gestão de todos os utilizadores;
- O sistema deve permitir que qualquer utilizador possa adicionar, editar ou remover:
  - Tipos de sensores;
  - Tipos de comunicação;
  - *Controller Module* com as respetivas especificações e características;
  - Tipo de comunicação a um *Controller Module* que possibilite a sua comunicação como o servidor;
  - *Sensor Modules* a um determinado *Controller Module* com as suas respetivas especificações e características;
  - Um ou vários tipos de comunicação a um *Sensor Module* que permita a sua comunicação com um *Controller Module*;
  - Um ou vários sensores a um *Sensor Module* em que cada sensor poderá ser de um determinado tipo.
- O sistema deverá disponibilizar uma API que permita a criação de novas aplicações com base nesta;
- Consultar a documentação da API de modo interativo;
- Gerar e consultar *token* de autenticação para utilização da API;
- Alterar configurações base do utilizador (primeiro nome, último nome, email, etc.).

### **Aplicação mobile**

- A interface da aplicação *mobile* deve permitir que o utilizador, seja ele qual for, entre ou faça *login* no sistema;
- A interface da aplicação *mobile* deve permitir que o utilizador, seja ele qual for, saia ou faça *logout* no sistema;
- A aplicação *mobile* deve permitir visualizar graficamente os dados de cada sensor, sendo possível filtrá-los por data;
- A aplicação deve permitir receber alarmes quando um determinado valor lido pelo sensor esteja fora do pretendido.

### 4.3.2 Requisitos não funcionais

Os requisitos não funcionais são todos os requisitos da aplicação relacionados com performance, escalabilidade, segurança, disponibilidade e usabilidade. Estes não são necessariamente requeridos pelo cliente.

- O sistema deverá executar em qualquer plataforma, tanto web como mobile. No caso de aplicação *mobile* pretende-se que esta seja executada em iOS e Android;
- A base de dados deve ser protegida para acesso apenas a utilizadores autorizados;
- O sistema deverá disponibilizar uma API para que possam ser criados novos produtos com base neste;
- Deverá ser usada, na medida do possível, tecnologias sem qualquer custo para o cliente (*open-source*);
- Pretende-se que o sistema possa ser adaptado a qualquer outro contexto, não sendo especificamente restrito ao contexto da produção da Salicórnia.

## 4.4 Modelação

Tal como vimos no início da secção 4.3, a conceção de uma arquitetura envolve o estudo e respetiva modelação dos componentes que são fundamentais para a realização da mesma, bem como a análise dos casos de uso e respetiva especificação, criação do modelo de dados, diagramas de fluxos, entre outros. Toda esta modelação será descrita neste secção permitindo entender melhor como emparelhar toda a estrutura pretendida. Relativamente à metodologia apresenta na secção 1.3, esta etapa refere-se à fase de *Designing*.

### 4.4.1 Entidades envolvidas

As entidades envolvidas ou atores são os utilizadores do sistema, podendo ser humanos ou entidade máquina, que interagem com o sistema para executar uma determinada ação significativa. No contexto do sistema descrito, existem três entidades distintas que são importantes descrever:

- **General user:** este ator poderá registar-se e associar-se a uma determinada empresa existente no sistema. Após a validação por parte da empresa, este utilizador poderá aceder à sua área reservada através da *dashboard* ou da aplicação *mobile*.
- **Company user:** este utilizador tem a possibilidade de gerir todos os *general users* que se possam associar à sua empresa. Deste modo, este utilizador poderá validar ou remover os *general users* que a si se associam.

- **Administrador:** vulgarmente denominado por *admin*. Pretende-se que apenas exista uma único administrador. Este ator tem a possibilidade de adicionar novas empresas ao sistema, isto é, criar novos utilizadores com permissões específicas.

#### 4.4.2 Casos de uso

Nas figuras 4.4 e 4.5 encontram-se representados os esquemas dos casos de uso da *dashboard* e da aplicação *mobile*, respetivamente. Seguidamente serão descritos cada um dos casos de uso para as duas plataformas. De notar que todos os casos de uso na aplicação *mobile* também se encontram disponíveis através da *dashboard*.

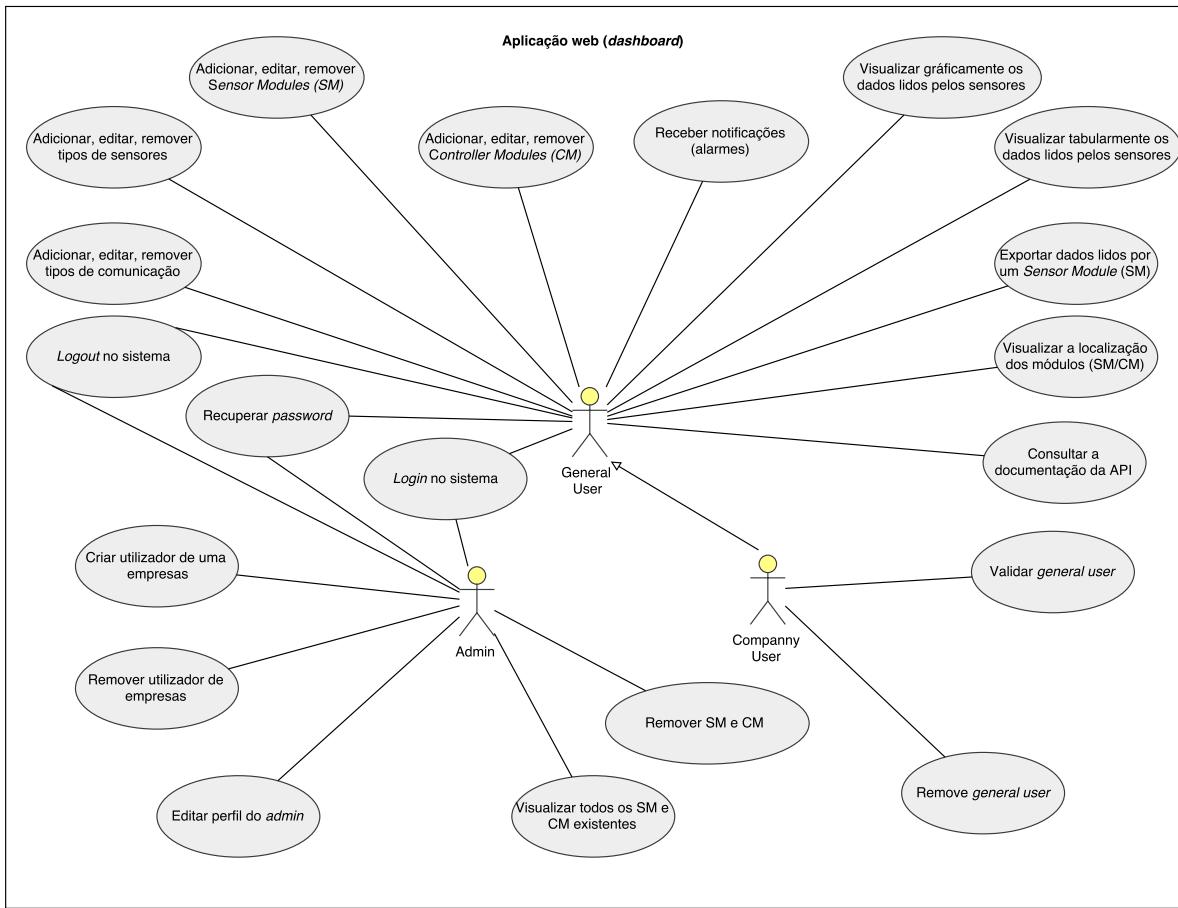
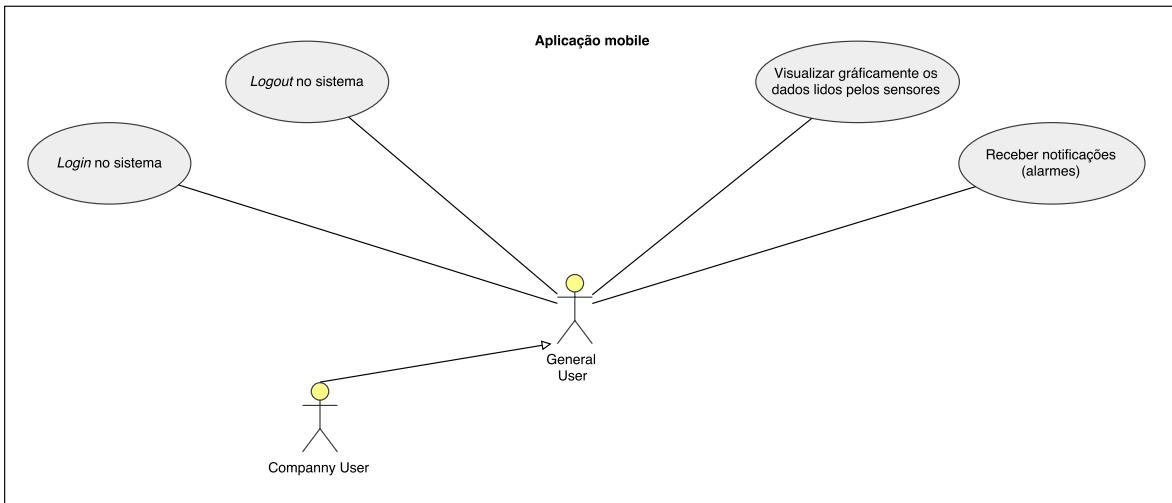


Figura 4.4: Casos de uso para a aplicação web (*dashboard*)

Figura 4.5: Casos de uso para a aplicação *mobile*

- **Login no sistema:** qualquer utilizador (*general user*, *company user* ou *admin*) poderá aceder ao sistema tendo para isso que possuir uma conta registada e validada no caso de ser um *general user*. Após a validação das suas credenciais (*username* e *password*) o utilizador poderá aceder à página inicial da *dashboard* e a todo o seu conteúdo. Este caso de uso encontra-se disponível para os dois tipos de plataformas;
- **Logout no sistema:** qualquer utilizador (*general user*, *company user* ou *admin*) poderá sair da sistema tendo que para isso que estar obrigatoriamente logado no sistema. Após o *logout* ser-lhe-á apresentado novamente a página de *login*. Este caso de uso encontra-se disponível para os dois tipos de plataformas;
- **Recuperar password:** qualquer utilizador (*general user*, *company user* ou *admin*) poderá recuperar a sua *password* de acesso ao sistema, para isso, terá que introduzir o seu email e posteriormente ser-lhe-á enviado o acesso que possibilita a redefinição da mesma;
- **Adicionar, editar, remover tipos de comunicação:** qualquer *general user* ou *company user* poderá adicionar, editar ou remover os tipos de comunicação existentes no sistema. Ao adicionar, o utilizador terá que introduzir um nome, um caminho relevante para o tipo de comunicação e selecionar um icon ilustrativo. No caso da remoção, esta ação apenas é possível caso o tipo de comunicação não se encontre em utilização por nenhum *Controller Module* ou *Sensor Module*;
- **Adicionar, editar, remover tipos de sensores:** qualquer *general user* ou *company user* poderá adicionar, editar ou remover os tipos de sensores existentes no sistema. Ao adicionar, o utilizador terá que introduzir um nome que o identifique, uma fonte de dados ou a escala dos mesmos e um icon que identifique o sensor. Para além disso, o

utilizador terá que escolher uma cor que identifique o tipo de sensor na *dashboard*. No caso da remoção, esta ação apenas é possível caso o tipo de sensor não se encontre em utilização por nenhum *Sensor Module*;

- **Adicionar, editar, remover um *Controller Module*:** qualquer *general user* ou *company user* poderá adicionar, editar ou remover um *Controller Module* existente no sistema. Neste caso, pressupõe-se que o utilizador possua um microcontrolador com as seguintes características que serão adicionadas: nome para identificação, valor da memória RAM em MBytes , estado inicial (ativo ou desativo) e um módulo de comunicação que permita comunicar com o servidor. Estas características são possíveis de alteração. Pretende-se que um *Controller Module* possa ser removido tendo *Sensor Modules* a sí associado, sendo estes também removidos automaticamente do sistema;
- **Adicionar, editar, remover um *Sensor Module*:** qualquer *general user* ou *company user* poderá adicionar, editar ou remover um *Sensor Module* existente no sistema. Pressupõe-se o utilizador possua um determinado microcontrolador com um ou mais sensores. Para adicionar o *Sensor Module* à plataforma o utilizador terá que atribuir um nome que o identifique, definir qual o seu estado inicial (ativo ou desativo), escolher os tipos de comunicação presentes e definir o intervalo de tempo para o qual pretende receber os dados lidos pelos sensores. Para além disso, permite associar ao *Sensor Module* os sensores presentes e definir o valor máximo e mínimos para o qual são gerados alarmes e respetivas mensagens para notificação. Todos estes dados são possíveis de edição;
- **Receber notificações (gerar alarmes):** sempre que um valor lido por um determinado sensor sai fora dos limites impostos, isto é do valor máximo e do valor mínimo, é gerado um alarme para o utilizador de modo a que este possa intervir. Este caso de uso encontra-se disponível para os dois tipos de plataformas;
- **Visualizar gráficamente os dados lidos pelos sensores:** o utilizador poderá interagir com as duas plataformas a fim de visualizar graficamente os dados lidos pelos diferentes tipos de sensores existentes, tendo a possibilidade de selecionar as datas de início e fim da consulta. Para além disso, poderá guardar os gráficos em diferentes formatos de imagem;
- **Visualizar tabularmente os dados lidos pelos sensor:** analogamente ao caso de uso anterior, o utilizador poderá neste caso visualizar tabularmente os dados obtidos, tendo também a possibilidade de filtrar os dados disponíveis por data;
- **Exportar dados lidos de um *Sensor Module*:** qualquer *general user* ou *company user* poderá exportar os dados lido pelos sensores, numa data específica, para um ficheiro do tipo CSV;

- **Visualizar a localização dos módulos (*Sensor Module/Controller Module*):** qualquer *general user* ou *company user* ao aceder à *dashboard* ser-lhe-á apresentado um mapa para cada *Controller Module* com a sua localização e dos seus respetivos *Sensor Modules*;
- **Consultar a documentação da API:** qualquer *general user* ou *company user* poderá aceder à *dashboard* para consultar a documentação iterativa da API existente;
- **Consultar *token* de autenticação:** qualquer *general user* ou *company user* poderá aceder ao *token* de autenticação para utilização da API;
- **Validar *general user*:** qualquer *company user* tem a possibilidade de validar os *general users* que a si se associam. Sempre que um novo *general user* é registado no sistema o *company user* é notificado via email. Posteriormente, caso o *general user* seja validado este também receberá um email de confirmação;
- **Remover *general user*:** qualquer *company user* tem a possibilidade de remove os *general users* a si associados;
- **Remover *Sensor Modules* e *Controller Modules* :** o *admin* do sistema tem a possibilidade de remover os *Sensor Modules* e/ou *Controller Modules* existentes no sistema;
- **Visualizar todos os *Sensor Modules* e *Controller Modules*:** o *admim* do sistema tem a possibilidade de visualizar todos os *Sensor Modules* e *Controller Modules* existentes no sistema de modo a alertar os clientes em caso de anomalias;
- **Criar um novo *company user*:** o *admim* tem a possibilidade de adicionar novos *company user* ao sistema;
- **Remover um *company user*:** o *admim* tem a possibilidade de remover qualquer *company user* registado no sistema.

#### 4.4.3 Modelo de dados

Depois da análise dos requisitos desenhou-se um modelo de dados que permitisse modelar todo o sistema descrito, obtendo-se assim o seguinte esquema relacional apresentado na figura 4.6.

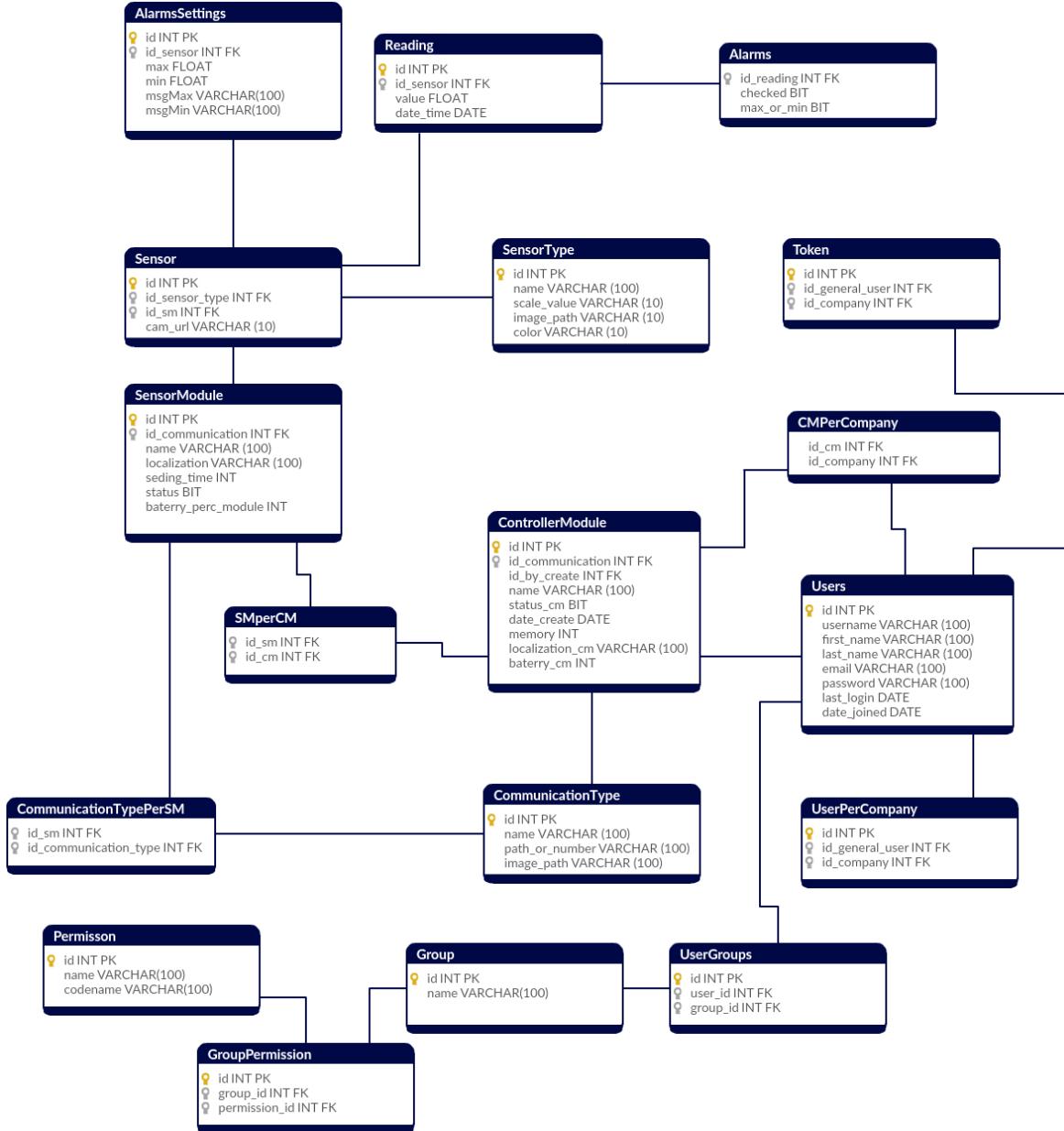


Figura 4.6: Esquema relacional da estrutura da base de dados

Nas tabelas 4.1 e 4.2 são descritas cada uma das entidades de dados existente neste sistema, evidenciando as chaves primárias (*Primary keys (PK)*) e estrangeiras (*Foreign Key (FK)*) de cada uma.

Nome da tabela	Chave primária (PK)	Chave estrangeira (FK)	Descrição
User	id (auto-incrementado)	N/A	Identifica cada um dos utilizadores inseridos no sistema
Token	auth_token_token_pkey (character varying(40))	user_id	Possui o <i>token</i> de autenticação do utilizador para a API
Group	id (auto-incrementado)	N/A	Possui todos os grupos existentes: <i>general user</i> , <i>company user</i> e <i>admin</i>
UserGroups	id (auto-incrementado)	user_id group_id	Permite associar um utilizador a um determinado grupo
Permissos	id (auto-incrementado)	N/A	Possui todas as permissões existentes (escrita, leitura, delete...)
GroupPermission	id (auto-incrementado)	group_id permission_id	Associa a cada grupo determinadas permissões
UserPerCompany	id (auto-incrementado)	company_id general_user_id	Associa cada <i>general user</i> a um <i>company user</i>
CommunicationType	id (auto-incrementado)	N/A	Identifica cada um dos tipos de comunicação inseridos no sistema
ControllerModule	id (auto-incrementado)	N/A	Identifica cada um dos <i>Controller Module</i> inseridos no sistema

Tabela 4.1: Especificação das tabelas da base de dados existentes no sistema

Nome da tabela	Chave primária (PK)	Chave estrangeira (FK)	Descrição
<b>CMPerCompany</b>	id (auto-incrementado)	cm.id company.id	Associa todos os <i>Controller Module</i> a um determinado <i>company user</i>
<b>SensorModule</b>	id (auto-incrementado)	N/A	Identifica cada um dos <i>Sensor Module</i> inseridos no sistema
<b>SMperCM</b>	id (auto-incrementado)	cm.id sm.id	Associa os <i>Sensor Modules</i> a um determinado <i>Controller Module</i>
<b>CommunicationTypePerSM</b>	id (auto-incrementado)	communication_type.id sm.id	Permite associar a um <i>Sensor Module</i> um ou várias tipos de comunicação
<b>SensorType</b>	id (auto-incrementado)	N/A	Identifica cada um dos tipos de sensores inseridos no sistema
<b>Sensor</b>	id (auto-incrementado)	sensor_type.id sm.id	Permite identificar um determinado sensor
<b>AlarmsSettings</b>	id (auto-incrementado)	sensor.id	Permite guardar as configurações para um determinado sensor (valor máximo, mínimo, mensagem)
<b>Reading</b>	id (auto-incrementado)	sensor.id	Permite guardar as leituras de um determinado sensor
<b>Alarms</b>	id (auto-incrementado)	reading.id	Armazena os alarmes que são gerados, associados a uma leitura

Tabela 4.2: Especificação das tabelas da base de dados existentes no sistema (continuação)

## 4.5 Arquitetura lógica

Nesta secção é apresentada a arquitetura lógica do sistema, indicando as camadas que contém, especificando quais as relações de dependência que estas possuem entre si. Seguidamente, é explicado como implementar esta lógica com os respetivos componentes físicos.

Normalmente este tipo de arquitetura é composto por três camadas com intenções diferentes: camada de apresentação, camada de lógica de negócios e camada de acesso a dados. Pretende-se que com a descrição desta arquitetura seja facilitada a manutenção, a portabilidade e a escalabilidade, fatores importantes quando queremos partilhar funcionalidades e informação entre aplicações de diferentes tipos.

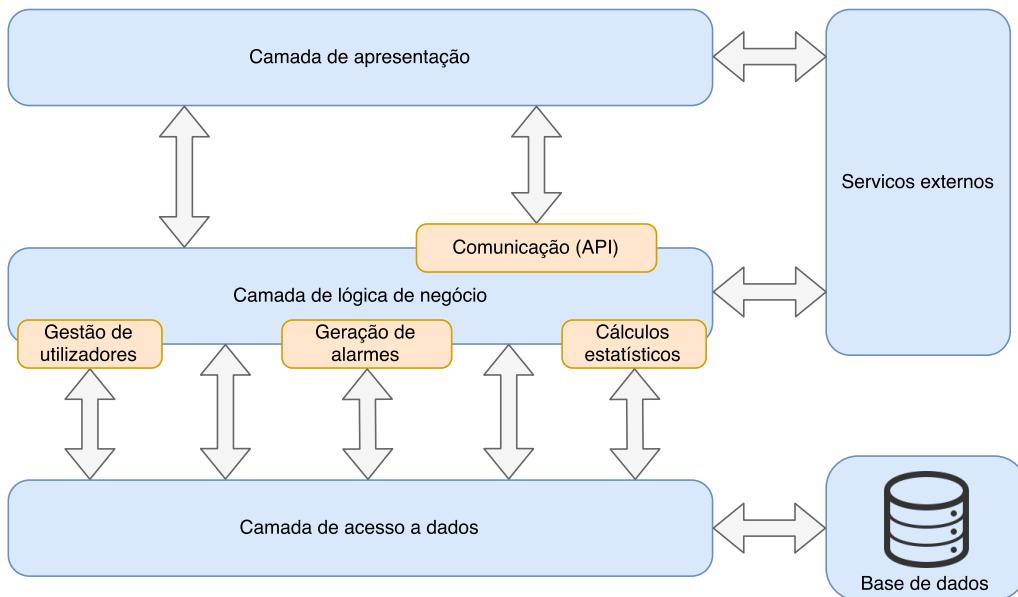


Figura 4.7: Esquema representativo da arquitetura lógica do sistema

A camada de apresentação é responsável pela comunicação entre os utilizadores e a aplicação, sendo ela *web* ou *mobile*, exibindo informações aos utilizadores, abrangendo uma interface que permite solicitações ao sistema. Esta camada tem uma relação de dependência com a camada de lógica de negócios e tira partido do acesso a serviços de informação externos, que fornecem diversas funcionalidades. A interface do utilizador foi desenvolvida em HTML e CSS, fazendo uso de jQuery e JS.

A camada de lógica de negócios diz respeito à camada onde é realizado todo o processamento dos dados adquiridos pelos sensores ou introduzidos pelos utilizadores do sistema através da camada de apresentação. Tal como apresentado na figura 4.7, existem quatro sub-camadas principais, que enfatizam os principais conceitos existentes nesta camada, a seguir descritos.

- **Gestão de utilizadores:** todas as operações realizadas por cada utilizador devem

ser medidas pelas permissões que estes possuem. Nesta camada é verificado se um determinado utilizador viola ou não essas permissões.

- **Geração de alarmes:** todos os alarmes são gerados conforme a verificação automática realizada a cada valor que chega ao sistema.
- **Cálculos estatísticos:** este componente da camada de lógica de negócio ganha especial relevância na altura em que se pretende determinar os valores médios, máximos e mínimos de um conjunto de medições para um determinado intervalo de tempo.
- **Comunicação via API:** este componente é fundamental para que se possa aceder, atualizar ou adicionar dados ao sistema de um modo simples.

Relativamente à camada de acesso a dados, deverão estar presentes as funcionalidades de criação, edição, remoção ou simples visualização dos dados, sendo responsável pelas operações de persistência e consulta de dados solicitados pela camada de lógica de negócio.

## 4.6 Arquitetura física

Enquanto que a arquitetura lógica se concentra nos diferentes níveis de abstração do sistema a arquitetura física foca-se na estrutura de alto nível. Seguidamente são apresentados todos os componentes físicos, tanto a nível de *software* com de *hardware*, que são fundamentais para ter um maior percepção do real funcionamento de todo o sistema. A figura 4.8 representa genericamente os blocos principais do sistema proposto que seguidamente serão descritos.

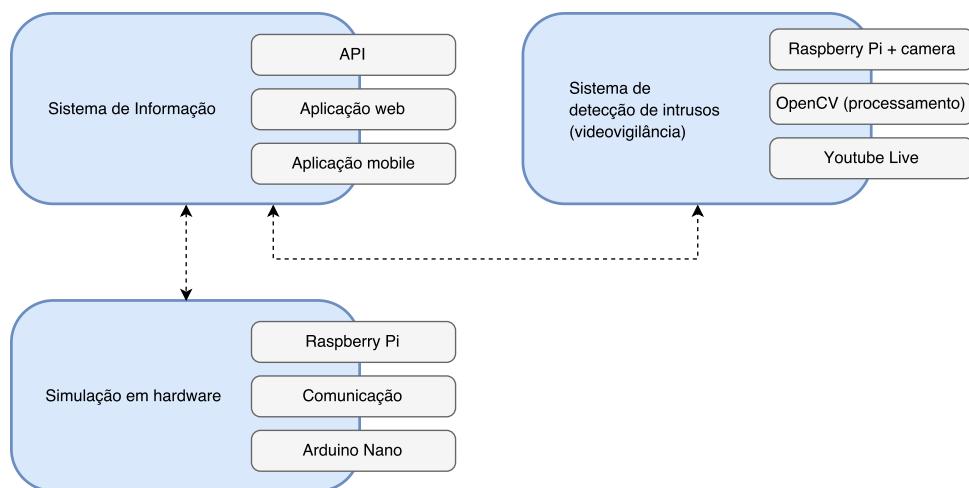


Figura 4.8: Arquitetura física (blocos principais)

#### 4.6.1 Sistema de informação

Segundo *Laudon et al.*[51], um sistema de informação define-se como sendo uma inter-relação de múltiplos componentes podendo estes ser equipamentos, telecomunicações, *software*, bases de dados e outras tecnologias de transformação de informação. Todos estes componentes permitem a recolha, processamento, armazenamento e distribuição de informação que possibilita a tomada de decisões e controlo para uma determinada organização, ou até mesmo para a sociedade, de modo a torná-la mais acessível e útil.

Dada a elevada complexidade de um sistema de informação, é possível identificar algumas funcionalidades comuns aos diversos sistemas existentes, são elas[52]:

- **Recolha de dados:** sequência de tarefas que possibilitam a adição de novos dados ao sistema;
- **Organização e armazenamento de dados:** é imprescindível uma boa organização da estrutura de dados possibilitando uma fácil e rápida localização;
- **Processamento de dados:** qualquer funcionalidade que permita a produção de resultados mais úteis do que os dados em bruto;
- **Distribuição de informação:** após o processamento de dados é fundamental a distribuição destes a quem precise deles;
- **Utilização da informação:** por si só, a informação não tem qualquer valor, a sua utilização em contexto adequado possibilita a extração de determinadas conclusões para que possam ser tomadas decisões.

A figura 4.9 ilustra a arquitetura do sistema de informação incluindo especificamente a aplicação web (*dashboard*), base de dados, API e respetiva implementação do sistema.

#### Aplicação web

A aplicação *web* é um componente essencial, enquadrando-se tanto na camada de lógica de negócio como também na camada de apresentação, permitindo a interação por parte do utilizador (*frontend*) como também no processamento lógico (*backend*).

Tal como vimos na secção 3.3.4 do Estado de Arte, a tecnologia para o desenvolvimento *web* recaiu sobre a *framework* Django, mais precisamente na versão 1.11 para python 2.7, sendo que como *Integrated Development Environment* (IDE) foi utilizada a verão 2016.3.3 do PyCharm. Dada a facilidade com que esta *framework* tem em manipular *views* e *templates*, optou-se que ambas as componentes (*frontend/backend*) fossem desenvolvidas recorrendo ao Django. Para além disso, optou-se por criar uma API REST que permitisse a manipulação dos dados existentes na base de dados, sendo esta também desenvolvida paralelamente com a aplicação *web*. Seguidamente será explicada a sua arquitetura.

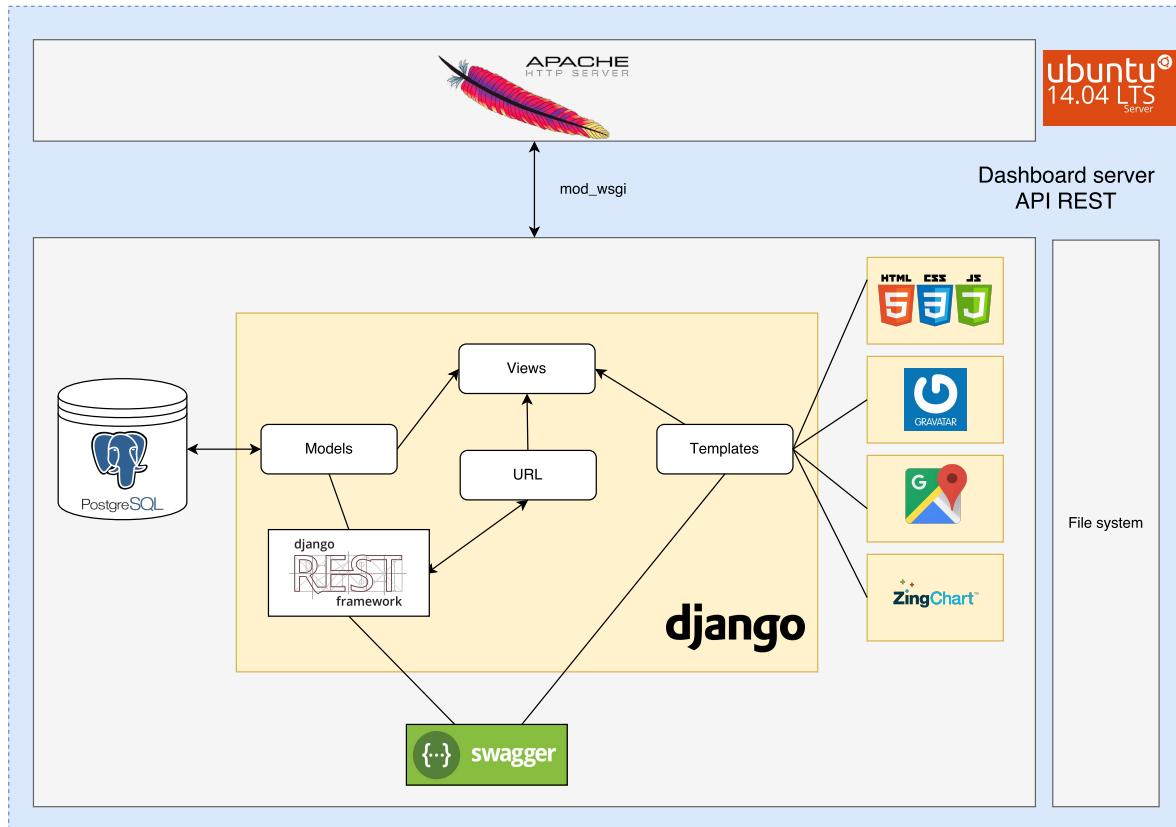


Figura 4.9: Arquitetura do sistema de informação (*dashboard*, base de dados e API)

Como descrito na secção 3.3.3, o Django permite modelar os dados através de classes, possibilitando gerar tabelas na base de dados de uma forma quase direta, sem recorrer ao SQL. A verificação dos valores lidos pelos sensores e possível geração de alarmes é averiguado por um *trigger* desenvolvido em SQL, no capítulo 5 será explicada a sua implementação.

Relativamente à escolha do SGBD recaiu sobre o PostgreSQL, mais concretamente na versão 9.3.16. Como ferramenta gráfica para administração deste SCBD foi utilizado o PgAdmin III<sup>1</sup> na versão 1.22. Este *software* com interface gráfica tem inúmeras funcionalidades desde a possibilidade de ligação a base de dados remotas até à adição, edição, remoção e consultas em tabelas. Esta ferramenta é *open-source* e encontra-se disponível para Windows e UNIX. Como vimos no capítulo 3, este SGBD permite uma grande facilidade de incorporação a um projeto Django.

No que diz respeito à camada de apresentação na plataforma *web* foram utilizadas as seguintes linguagens/bibliotecas/frameworks:

- **HTML, CSS e JS:** o ponto de partida para a criação da interface *web* assentou no *template* denominado por AdminLTE<sup>2</sup> sendo este baseado em Bootstrap 3<sup>3</sup>. Neste

<sup>1</sup>[www.pgadmin.org/](http://www.pgadmin.org/)

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

<sup>3</sup><http://getbootstrap.com/>

*template* prevalecem as seguintes características: *design* responsivo, interface leve e aperfeiçada, existência de múltiplos *plugins*, compatibilidade entre navegadores entre outros.

- **Gravatar:** serviço que disponibiliza um avatar que esteja associado a um endereço de email registado. O Gravatar disponibiliza uma API que pode ser utilizada nas mais diversas linguagens de programação[53].
- **API Google Maps:** consiste num serviço de visualização de mapas e imagens de satélite, desenvolvido pela Google. É usado na visualização da localização dos módulos de sensores.
- **ZingChart:** biblioteca em JS que permite receber dados a apresentá-los em formato gráfico. Esta solução *open-source* disponibiliza mais de 35 tipos e módulos de gráficos.

## API REST

Os métodos desta API permitiram executar funções do tipo REST. A tecnologia REST foi apresentada por Roy Fielding na Universidade de Califórnia no ano de 2000, cujo o título da sua dissertação era ”Architectural Styles and the Design of Network-based Software Architectures”. Roy estudou um conjunto de arquiteturas de *software* que usam a *web* como uma plataforma para computação distribuída[54]. Esta tecnologia define um conjunto de princípios que possibilitam desenhar serviços *web* com base nos recursos do sistema, considerando a forma com os recursos são coordenados e transferidos através de HTTP para vários clientes nas mais diversificadas linguagens.

Os métodos da API permitem executar as funções REST usando métodos HTTP explicitamente. Assim, torna-se fundamental perceber estes métodos para ter um melhor conhecimento do funcionamento da API. Seguidamente são descritos os métodos mais importantes que dão suporte a cada uma das funções REST.

- **GET:** permite efetuar operações de leitura;
- **POST:** permite realizar operações de escrita, possibilitando criar novos recursos ao sistema;
- **PUT:** permite criar ou atualizar um novo objeto ao sistema;
- **DELETE:** permite apagar objetos ou recursos ao sistema.

Como vimos no capítulo 3, a escolha da *framework* para construção desta API REST recaiu sobre o *Django REST framework*. Esta ferramenta possui uma extensa documentação e um elevado apoio da comunidade que a utiliza, sendo uma das mais utilizadas e incorporadas em projetos Django.

Relativamente à autenticação desta API, optou-se por utilizar um esquema de autenticação fundamentado no mecanismo de autenticação HTTP baseado em *tokens*[55]. Neste mecanismo, o cliente primeiramente troca as suas credenciais (*username* e *password*) por um *token*, seguidamente, em vez de enviar essas credenciais a cada requisição, o cliente apenas enviará o *token* inicialmente recebido, permitindo assim o acesso aos conteúdos pretendidos, com respetiva autenticação e autorização dos mesmos (figura 4.10).

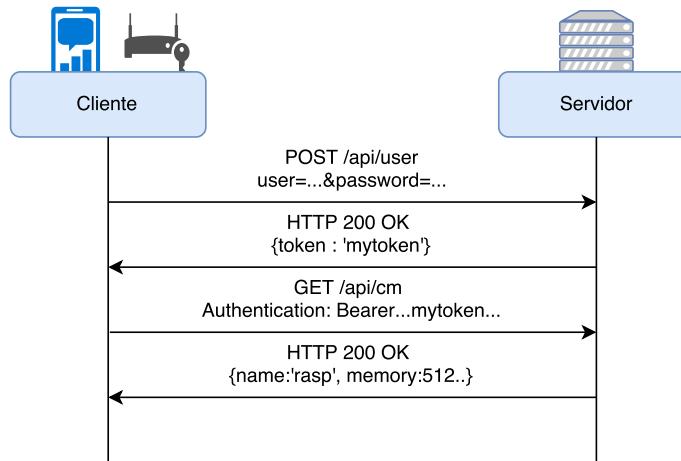


Figura 4.10: Processo de autenticação em HTTP através de *token* (Adaptado de [56])

Resumidamente e com o objetivo de explicar o esquema da figura 4.10, são enumerados os passos de autenticação em HTTP via *token*.

1. O cliente envia as credenciais para um servidor;
2. O servidor valida e autentica essas credenciais gerando consequentemente um *token*;
3. O servidor envia o *token* para o cliente;
4. O cliente guarda o *token* e este é enviado sempre que existe uma requisição através do cabeçalho do protocolo HTTP;
5. O servidor, em cada requisição verifica se o *token* é válido ou não. Caso seja, o servidor aceita a requisição, caso contrário esta é rejeitada;
6. O servidor pode ter um *endpoint* que renova o *token* sempre que necessário.

Na tabela 4.3 encontram-se todos os *endpoints* e respetivamente funções (POST/GET/-PUT) a implementar. Seguidamente será descrito como foi implementada a documentação interativa desta API.

<b>Endpoints da API REST</b>	<b>POST</b>	<b>GET</b>	<b>PUT</b>	<b>DELETE</b>
/api/user/	✓	✓		
/api/user/{pk_or_username}/		✓	✓	✓
/api/smpercm/		✓		
/api/smpercm/{pk_or_name_cm}	✓	✓		
/api/sm/	✓	✓		
/api/sm/{pk_or_name}/		✓	✓	✓
/api/sensortype/	✓	✓		
/api/sensortype/{pk_or_name}		✓	✓	✓
/api/sensorpersm/{id_sm_or_name_sm}	✓	✓		
/api/sensor/		✓		
/api/sensor/{pk_or_sensor_type}	✓	✓		
/api/reading/{id_sensor}/{date_start}/{date_end}	✓	✓		
/api/communication/{pk_or_name}/		✓	✓	✓
/api/cm/	✓	✓		
/api/cm/{pk_or_name}/		✓	✓	✓
/api/alarmssettings/{id_sensor}	✓	✓		
/api/alarms_sensor/{id_sensor}	✓	✓		
/api/alarms_reading/{id_reading}	✓	✓		

Tabela 4.3: *Endpoints* da API REST e respetivos métodos a implementar

## Documentação interativa

Para a geração automática da documentação da API utilizou-se o Swagger. Tal como descrito no site oficial desta *framework*[57], o Swagger é considerado a ferramenta de APIs mais popular e completa de todo o mundo permitindo facilmente o desenvolvimento do ciclo de vida de uma API, desde o *design*, documentação, testes e também implementação, tendo a grande vantagem de ser *open-source*. Neste contexto, apenas será utilizada como documentação, de modo a facilitar a interpretação da API criada. O Swagger possui uma interface apelativa e intuitiva, permitindo interagir com a API de modo que os seus utilizadores tenham uma ideia geral de como esta responde aos pedidos para diversos parâmetros e opções.

## Implementação do sistema

Para implementação do projeto anteriormente descrito e respetiva API, foi-me disponibilizada uma máquina virtual com uma distribuição Linux (Ubuntu 14.04.5) com as seguintes características:

- CPU: Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30GHz
- RAM: 2 GB
- Disco: 10.7 GB

Para o processo de *deployment* do projeto optou-se pela utilização do servidor Apache juntamente com o pacote mod\_wsgi<sup>4</sup>. Este pacote fornece uma WSGI compatível para o alojamento de aplicações *web* em Python sob o servidor HTTP Apache. O Apache é um servidor *web open-source* mais utilizado em todo o mundo[58].

### Aplicação mobile

Após a análise de requisitos da aplicação *mobile*, optou-se por elaborar um protótipo desta antes de proceder à sua real implementação. O *mockup* da aplicação apresenta-se no Apêndice A.

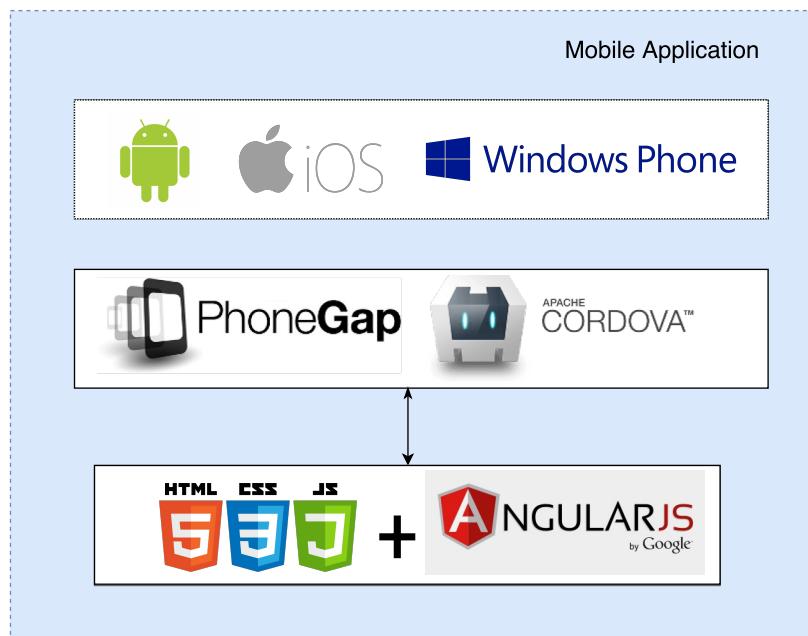


Figura 4.11: Arquitetura da aplicação *mobile* com respetivas tecnologias

Tal como descrito no capítulo 3 (Estado da Arte), para o desenvolvimento da plataforma *mobile* optou-se pela utilização de um paradigma multi-plataforma, mais concretamente a *framework* Phonegap. Esta framework utilizada a tecnologia Cordova da Apache que permite a integração com recursos nativos dos dispositivos. Através dela, é possível desenvolver aplicações móveis utilizando simplesmente HTML, CSS e JS sem a necessidade de depender de APIs específicas. De modo a facilitar a manipulação do JS optou-se por utilizar a *framework* AngularJS. Esta biblioteca *open source* mantida pela Google desde 2010, permite abstrair a manipulação do DOM e segue uma arquitetura MVC[59].

<sup>4</sup><https://modwsgi.readthedocs.io/en/develop/>

### 4.6.2 Simulação em hardware

Após o desenvolvimento da API, simulou-se o sistema num contexto real. Para tal, pretendia-se encontrar *hardware* que encaixa-se no contexto deste projeto. Foram utilizados dois microcontroladores (Arduino Nano e Raspberry Pi 3) e alguns sensores. Para este cenário, assume-se que o Arduino Nano é considerado um *Sensor Module* que possui um conjunto de sensores enquanto que o Raspberry Pi 3 é um *Controller Module* que recebe os dados provenientes do *Sensor Module* enviando-os para o servidor.

Seguidamente, são apresentados os sensores utilizados bem como os tipos de comunicação.

#### Sensores utilizados

Nesta secção apresentam-se os sensores utilizados na simulação e as suas principais características. Todos os sensores foram escolhidos tendo em conta o seu enquadramento no projeto e a sua disponibilidade em laboratório, sendo que serão ligados ao Arduino Nano.

#### Temperatura

Como sensor de temperatura foi utilizado um termíster do tipo *Negative Temperature Coefficient* (NTC). Como vimos no capítulo 3, um termíster é um semicondutor sensível à temperatura, ou seja, quando o coeficiente de variação da resistência com a temperatura é negativa, então a temperatura sobe e consequentemente a resistência diminui. Na figura 4.13 encontra-se o esquema de ligação deste componente e na tabela 4.4 as suas propriedades principais[60].



Figura 4.12: Sensor TTC 104 NTC

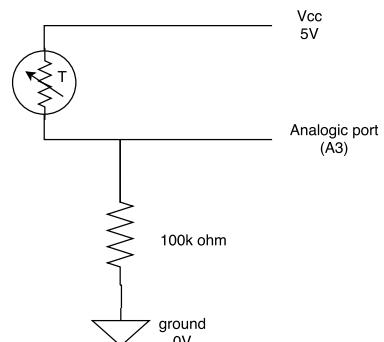


Figura 4.13: Esquema eletrónico da ligação do sensor de temperatura

<b>Dimensão</b>	5mm
<b>Resistência</b>	100 KΩ
<b>Valor máximo</b>	+125 °C
<b>Valor mínimo</b>	-30 °C
<b>Nível de confiança</b>	± 10%
<b>Preço</b>	0.35 €/unidade

Tabela 4.4: Características do sensor TTC 104 (Adaptado de [60])

### Luminosidade

Para simular a luminosidade incidente foi utilizado um sensor do tipo foto-resistência, neste caso o GL5528 (figura 4.14). Este sensor, também conhecido como LDR, não é mais do que uma resistência variável cujo o seu valor varia conforme a intensidade da luz que incide sobre ele, isto é, à medida que a intensidade da luz aumenta, a sua resistência diminui. Este sensor tem múltiplas aplicações, entre as quais se destaca a monitorização solar, indicador da posição do sol (*up/down*), alarmes anti-roubo, alarme para abertura/fecho de portas entre outras. Como vimos no capítulo 3 é um sensor de baixo custo e de fácil utilização. Na figura 4.15 encontra-se o esquema de ligação do componente e na tabela 4.5 são apresentadas as principais características do sensor utilizado.

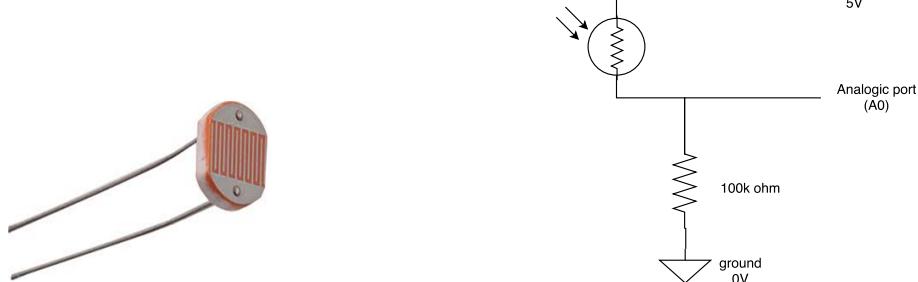


Figura 4.14: Sensor de foto-resistência GL5528

Figura 4.15: Esquema eletrónico da ligação do sensor de luminosidade

<b>Diâmetro</b>	5 mm
<b>Tensão máxima</b>	150 VDC
<b>Potência máxima</b>	100 mW
<b>Tensão de operação</b>	-30 °C a 70 °C
<b>Espectro</b>	540 nm
<b>Comprimento com terminais</b>	32 mm
<b>Resistência na luz</b>	10-20 KΩ (Lux 10)
<b>Material</b>	Carbono
<b>Preço</b>	0.22 €/unidade

Tabela 4.5: Características do sensor GL5528 (Adaptado de [61])

### Sensor para verificação do estado do nível de água

Este sensor, denominado por *Water Level Switch Liquid Level Sensor Plastic Ball Float* (figura 4.16), não é mais do que um interruptor que é ativo sempre que um determinado líquido ultrapassa o mesmo, isto é, sempre que algum líquido atingir o pedaço de plástico este irá subir ativando assim o circuito. Na figura 4.17 encontra-se o esquema da ligação deste sensor.



Figura 4.16: *Water Level Switch Liquid Level Sensor Plastic Ball Float*

Figura 4.17: Esquema eletrónico da ligação do sensor de nível líquido

### Simulador de válvula para transferências de águas

Para a simulação de uma válvula que permitirá a transferência de água doce e/ou água salgada foi utilizado um LED. Este possibilita facilmente identificar através da sua ativação se a válvula se encontra ativa ou não. Na figura 4.19 encontra-se o esquema de ligação deste componente.



Figura 4.18: LED

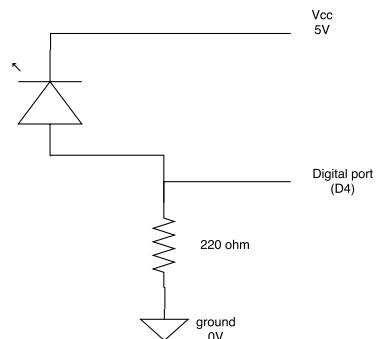


Figura 4.19: Esquema eletrónico da ligação do LED

## Comunicação

Nesta secção, apresentam-se os tipos de comunicação para o cenário escolhido. Pretendia-se que cada um dos módulo fique isolado, o que implicou o estudo e respetiva escolha de algumas tecnologias de comunicações sem fios (secção 3.8 do capítulo 3).

De acordo com o verificado na tabela 3.6, o Zigbee e o Sigfox apresentam características que melhor se adaptam ao universo do IoT. No entanto, para o contexto deste trabalho, privilegia-se o alcance das tecnologias Bluetooth e Wi-Fi dado que, pretende-se tirar partido desta característica e distribuir os *Sensor Modules* pela máxima distância possível. Para além disso, estas eram as duas únicas tecnologias disponíveis no laboratório.

- **Bluetooth:** utilizado para a comunicação entre o Arduino Nano e o Raspberry Pi 3. No Arduino, foi utilizado um módulo Bluetooth HC-06 e no caso do Raspberry Pi 3 foi utilizado o seu módulo interno (versão 4.1).
- **Wi-Fi:** utilizado para a comunicação entre o Raspberry Pi 3 e o servidor *web*, sendo utilizado o seu módulo interno (802.11.g).

O esquema da figura 4.20 ilustra os tipos de comunicação envolvidos nesta simulação para cada um dos componentes.

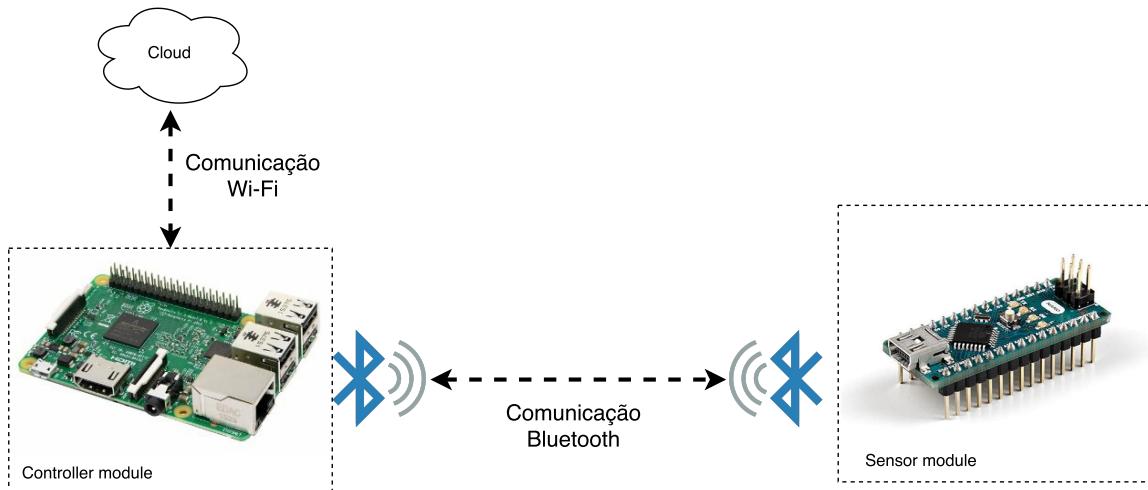


Figura 4.20: Comunicação entre componentes da simulação em *hardware*

### Módulo Bluetooth HC-06

Este módulo Bluetooth oferece uma forma simples de envio e receção de informações remotamente, podendo ser adquirido a um custo reduzido. Este componente funciona apenas em modo *slave*, isto é, apenas permite que outros dispositivos se liguem a si, mas não permite que ele próprio se ligue a outros. Para além disso, possui um LED que permite indicar se algum dispositivo está emparelhado. É um dos módulos Bluetooth mais comuns para o microcontrolador Arduino, possuindo um alcance máximo de aproximadamente 10 metros[62].

Na figura 6.3 encontra-se o esquema de ligação deste módulo e na tabela 4.6 são apresentadas as suas principais características.

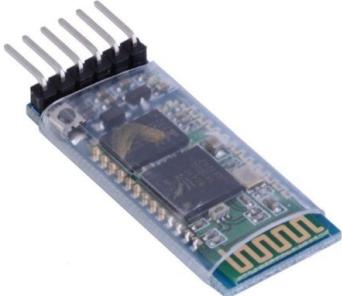


Figura 4.21: Módulo Bluetooth HC-06

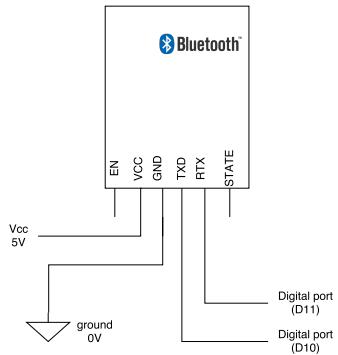


Figura 4.22: Esquema eletrónico da ligação do módulo Bluetooth

<b>Versão Bluetooth</b>	v2.0 com <i>Enhanced Data Rate (EDR)</i>
<b>Frequência</b>	2,4GHz Banda ISM
<b>Segurança</b>	Autentificação (PIN) e Encriptação
<b>Tensão</b>	Aconselhada 3,3v (2,7v - 4.2v)
<b>Alcance</b>	10 metros
<b>Dimensões</b>	26,9 x 13 x 2,2mm
<b>Peso</b>	9,6g
<b>Temperatura (funcionamento)</b>	-25C +75C
<b>Preço</b>	5.26 €/unidade

Tabela 4.6: Características do módulo bluetooth HC-06 (Adaptado de [62])

#### 4.6.3 Sistema de videovigilância

No contexto desta dissertação surgiu a necessidade de implementar um sistema de videovigilância que permitisse detetar intrusos, maioritariamente pessoas ou animais de grande porte, que possam invadir as quintas onde se produz Salicórnia. Esta necessidade prende-se essencialmente com elevado custo do *hardware* do sistema de monitorização e também de eventuais instrumentos de elevado custo necessários ao cultivo desta espécie, como por exemplo, geradores, máquinas elétricas para poda, entre outros.

#### Biblioteca para processamento de imagem: OpenCV

O OpenCV, também conhecido por *Open Source Computer Vision Library*, é uma biblioteca de *software* de visão por computador de código *open-source*. Esta biblioteca possui mais de 2500 algoritmos otimizados, que inclui um conjunto abrangente de algoritmos clássicos e avançados de visão computacional bem como algoritmos de *machine learning*. Esses algoritmos podem ser utilizados para os mais diversos fins, como por exemplo, para detetar e reconhecer rostos, identificar objetos, classificar ações humanas em vídeos, detetar movimentos numa câmara, seguir um objetos em movimento, entre outros. Esta biblioteca é

amplamente utilizada em empresas e grupos de investigação, tendo interfaces nas mais diversas linguagens: C++, C, Python, Java e MATLAB, embora seja nativamente escrita na linguagem C. OpenCV tem mais de 47 mil pessoas na sua comunidade e excede os 7 milhões de *downloads*, tendo suporte para Windows, Linux e Mac OS[63].

Desde logo, a escolha da tecnologia para processamento de imagem recaiu sobre o OpenCV não apenas por ser uma biblioteca bastante popular e possuir bastantes algoritmos implementados mas também por eu próprio possuir já algum *background* e projetos desenvolvidos nesta área. Pretendia-se que este processamento fosse implementado em material já adquirido sem necessidade de gastos. Optou-se então por utilizar um *Raspberry Pi 3* que juntamente com um *Raspberry Pi camera module* (figura 4.23) permitirá a aquisição de imagem e servirá também como *Controller Module* ao sistema de aquisição de dados. Na tabela 4.7 apresentam-se algumas características deste módulo para o Raspberry Pi 3.

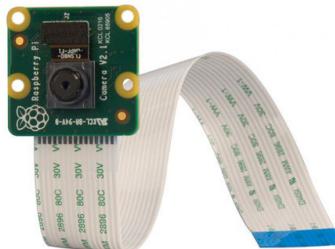


Figura 4.23: Raspberry Pi Camera Board V2 8MP 1080p

<b>Sensor</b>	8 megapixels Sony IMX219
<b>Resolução de exibição</b>	1080p, 720p e 640x480p para vídeo
<b>Ligaçāo à placa</b>	Cabo fita curta (branca)
<b>Dimensões</b>	25 x 23 x 9 mm
<b>Peso</b>	aproximadamente 3 g
<b>Compatibilidades</b>	última versão do Raspbian
<b>Preço</b>	26,67 €/unidade

Tabela 4.7: Características do Raspberry Pi camera module

Inicialmente, após incorporar o *Raspberry Pi camera module* no *Raspberry Pi 3* através do *Camera Serial Interface* (CSI) (imagem 3.5 da secção 3.6.2), testou-se a captura de vídeo através do raspivid. Esta ferramenta de linha de comando, permite testar o funcionamento do módulo, e para além disso, definir em que formato será guardado o vídeo adquirido, bem como o seu comprimento e dimensões[64].

Posteriormente, procedeu-se ao envio da imagem capturada, sem qualquer processamento, para o Youtube Live através da ferramenta FFmpeg. O Youtube Live permite que qualquer

utilizador possa realizar *streaming* de vídeo, possibilitando escolher entre dois modos: alta latência e baixa latência, isto é, boa ou má qualidade. Uma das grandes vantagens desta ferramenta é manter a *stream* transmitida permitindo consultá-la quando desejado. O Youtube Live utiliza o protocolo RTMP, pertencente à camada protocolar TCP/IP, mantendo uma conexão persistente com um servidor que permite a comunicação em tempo real de vídeo ou áudio. Por outro lado, o FFmpeg é uma ferramenta líder, capaz de codificar ou descodificar qualquer tipo de conteúdo, possibilitando a criação ou conversão de *stream* de vídeo/áudio em qualquer formato. É uma ferramenta *open-source* sendo que a sua interação é concretizada através de linha de comandos[65].

Na figura 4.24 encontra-se a arquitetura do sistema de videovigilância anteriormente descrito. No próximo capítulo será apresentado o algoritmo de deteção de intrusos disponibilizado pela ferramenta OpenCV.

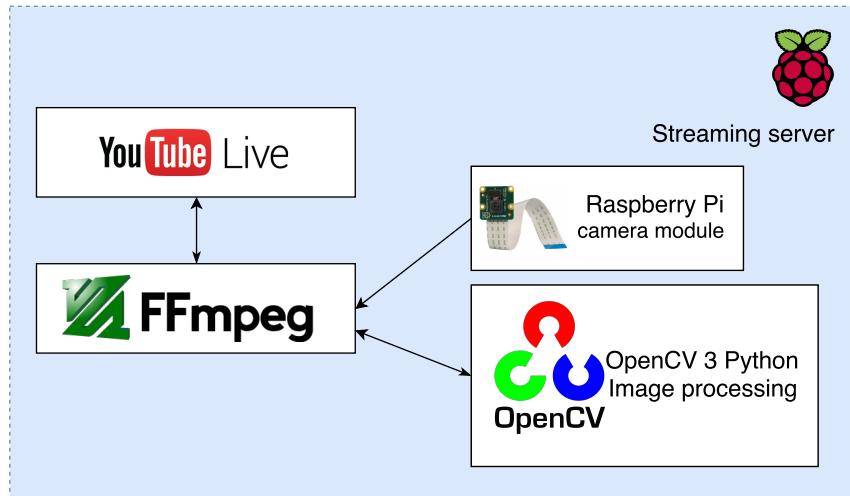


Figura 4.24: Arquitetura do sistema de videovigilância e deteção de intrusos

## 4.7 Diagrama de componentes

Para o cenário de simulação em *hardware* e respetivo sistema de informação, em que se inclui a *dashboard*, a API, aplicação *mobile* e o sistema de videovigilância, obteve-se o seguinte diagrama de componentes (figura 4.25) com as respetivas tecnologias utilizadas.

No que toca à comunicação entre os diferentes componentes, esta baseia-se essencialmente na API REST que foi criada, embora no caso do sistema de videovigilância seja utilizado um serviço externo (Youtube Live). Por fim, para a comunicação entre *Controller Module* e o *Sensor Module* no cenário de simulação é utilizada a tecnologia Bluetooth.

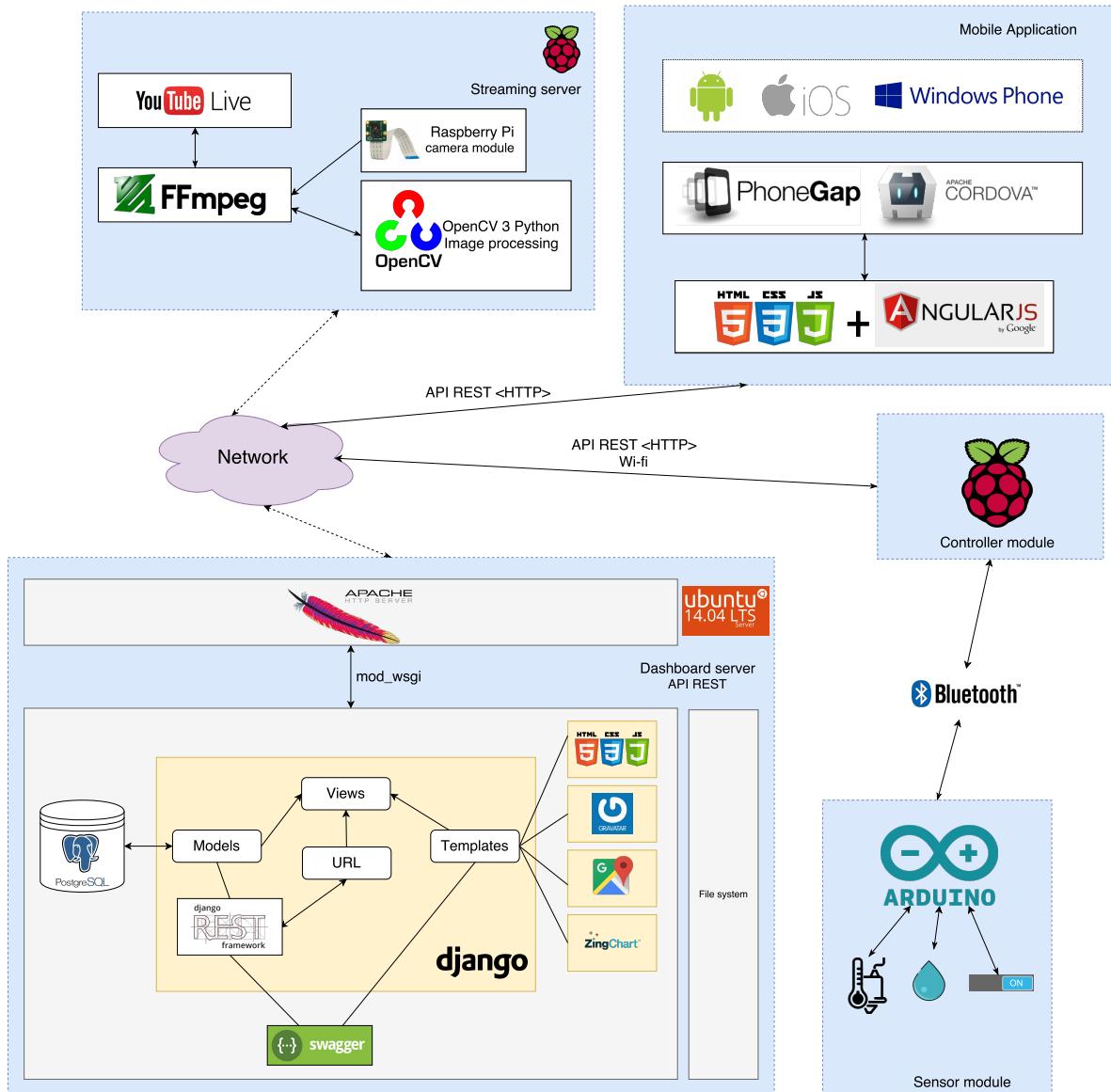


Figura 4.25: Diagrama final de componentes do sistema

## 4.8 Considerações finais

Neste capítulo é projetada toda a arquitetura do sistema, tendo por base os requisitos do cliente e respetiva modelação. Para além disso, a arquitetura anteriormente apresentada foi concebida tendo como garantia que poderia ser adaptada a quaisquer outros cenários distintos do apresentado. Esta arquitetura é apoiada através da exposição de diagramas de arquitetura, casos de uso e modelo de dados.

# Implementação

Neste capítulo, são explicados os detalhes da implementação deste projeto, incluindo especificamente o projeto Django que inclui o sistema de informação com a aplicação web (*frontend/backend*) e API REST. Para além disso, é descrita a implementação da simulação em *hardware* para o cenário descrito no capítulo anterior bem como o sistema de videovigilância para deteção de intrusos.

Segundo o ciclo de vida de um *software*, estabelecido por *Munish Kaur et al.*[2] que se apresentou na secção 1.3, este capítulo irá incluir, em parte, a fase de *designing* mas sobretudo a fase de *coding*.

## 5.1 Sistema de informação

Relativamente ao projeto Django, numa primeira fase procedeu-se à incorporação do SGBD PostgreSQL através do psycopg2<sup>1</sup>. O psycopg2 consiste num adaptador entre o PostgreSQL com a linguagem de programação Python, que permite executar de forma eficiente qualquer *script* em SQL. É de notar que nesta fase inicial, encontram-se instaladas algumas aplicações nativas do Django entre as quais, o `django.contrib.auth` e o `django.contrib.sessions`. Através delas é possível verificar o total funcionamento da base de dados uma vez que permitem criar automática das tabelas associadas ao utilizadores, grupos, permissões e respetivos conteúdos, administração, sessões entre outros. Estas tabelas são fundamentais ao bom funcionamento do sistema, sendo consideradas no modelo de dados descrito na secção 4.4.3.

Posteriormente, procedeu-se à criação dos diferentes `Models` conforme a nomenclatura apresentada nas tabelas 4.1 e 4.1 da secção 4.4.3. O excerto de código seguinte pretende exemplificar a criação de um `Model` associado à tabela que representa a estrutura *Sensor*

---

<sup>1</sup><http://initd.org/psycopg/docs/>

*Module*, com o respetivo identificador e atributos. Após a criação de cada *Model* procedeu-se à migração dos dados para o SGBD onde foi possível verificar que as tabelas tinham sido criadas, através da utilização da ferramenta gráfica pgAdmin III. De modo a testar a estrutura criada, procedeu-se à introdução de dados através da zona administrativa do Django (figura 5.1). Através dos dados introduzidos e descrevendo um cenário realista foi possível validar a estrutura e proceder à implementação *web* e criação da API REST.

```

1 class SensorModule(models.Model):
2     id = models.AutoField(primary_key=True)
3     name = models.CharField(max_length=128)
4     seding_time = models.IntegerField()
5     status_sm = models.BooleanField(default=True)
6     baterry_sm = models.IntegerField()
7     localization_sm = models.CharField(max_length=128)
8
9     def __str__(self):
10        return "#" + str(self.id) + " " + self.name

```

The screenshot shows the Django Admin interface. At the top, there's a blue header bar with the text 'Administração do Django'. Below it, a dark blue bar says 'Administração do site'. The main area contains several sections:

- AUTENTICAÇÃO E AUTORIZAÇÃO**: Lists 'Grupos', 'Permissões', and 'Utilizadores' with 'Adicionar' and 'Modificar' buttons.
- AUTH TOKEN**: Lists 'Tokens' with 'Adicionar' and 'Modificar' buttons.
- SALIAPP**: Lists 'Alarms settingss', 'Alarms', 'Cm per companies', and 'Communication type per sms' with 'Adicionar' and 'Modificar' buttons.
- Recent actions**: A sidebar listing recent actions:
  - SM: 812 Type: 2humidity | Valor lido:8.0 as 2017-05-31 09:59:20+00:00 Reading
  - SM: 812 Type: 2humidity | Valor lido:8.0 as 2017-05-31 09:59:33+00:00 Reading
  - SM: 812 Type: 2humidity | Valor lido:10.0 as 2017-05-31 09:56:25+00:00 Reading
  - 1modul1 Sensor module
  - 3cam Sensor module

Figura 5.1: Painel administrativo da framework Django

### 5.1.1 Sistema de registo e autenticação

Primeiramente, procedeu-se à adaptação do *template* AdminLTE de modo a criar as páginas de autenticação e registo dos utilizadores. Como vimos anteriormente irão existir utilizadores distintos, assim de modo a identificá-los e a definir as suas permissões houve necessidade de criar grupos específicas. Foram então criados dois grupos: *company*, que identifica uma empresa e *general*, que identificada um utilizador comum. O administrador do sistema é um utilizador que tem o estado de *superuser* ativo, isto é, possui todas as permissões

sem que estas lhe sejam atribuídas explicitamente.

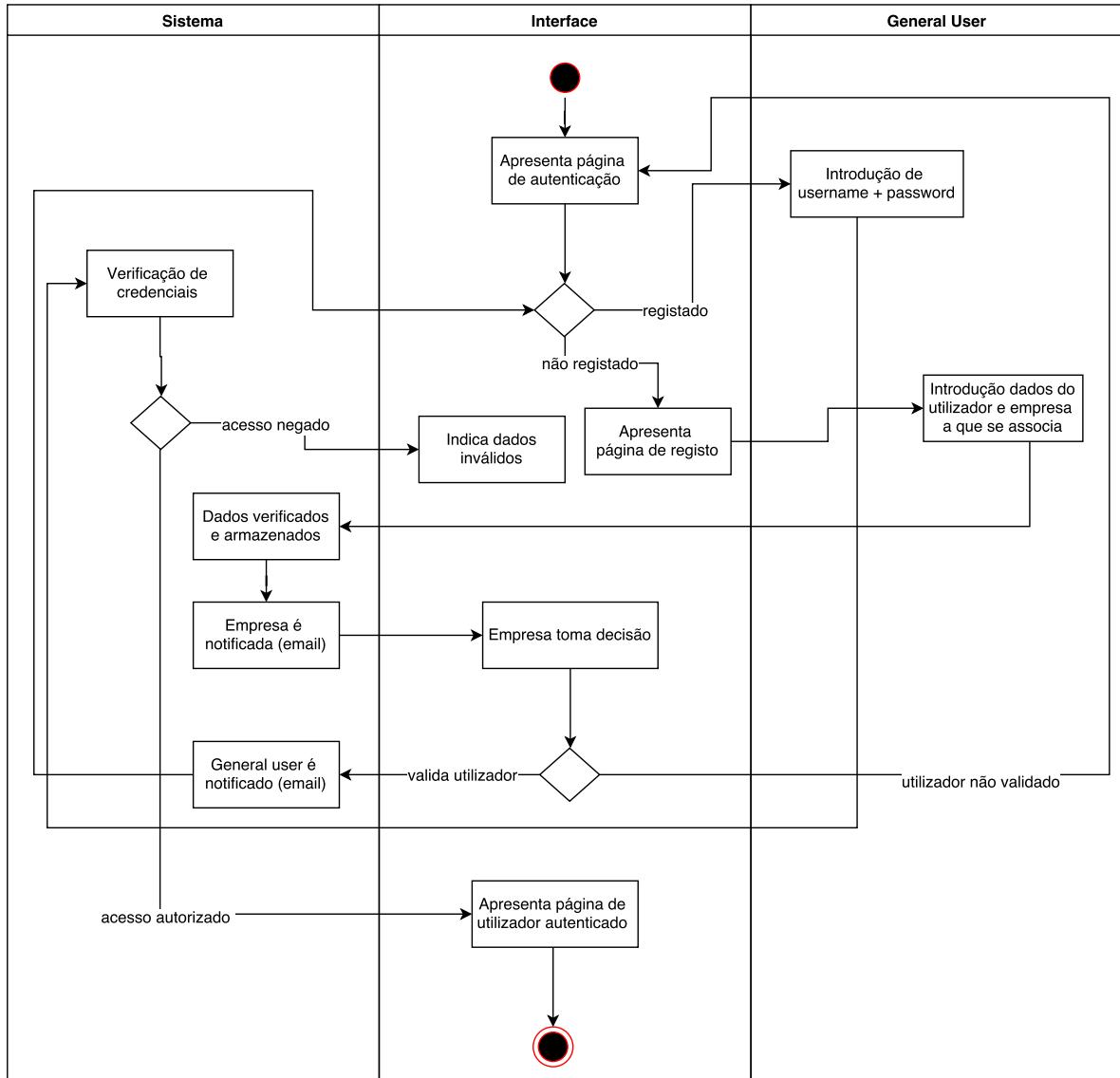


Figura 5.2: Diagrama de atividades do processo de registo e autenticação

Como vimos anteriormente, o *company user* apenas poderá ser adicionado ao sistema pelo administrador, tendo este também permissões de gerir todos os utilizadores registados. Por outro lado, os *company user* tem a possibilidade de validar os *general user* que se associam à sua empresa, tendo posteriormente acesso aos dados/conteúdos desta. Após o registo de um novo *general user* e a validação por parte do *company user*, existem notificações que são enviadas via email. Estas mensagens são enviadas recorrendo ao método `send_mail`<sup>2</sup> existente nativamente no Django pelo pacote `django.core.mail`, sendo construído através do módulo

<sup>2</sup><https://docs.djangoproject.com/en/1.11/topics/email/>

`smtplib`<sup>3</sup>. Na figura 5.2 apresenta-se o diagrama de atividades que ilustra o processo de registo de um *general user* que envolve o utilizador responsável pela empresa (*company user*).

### 5.1.2 Geração de alarmes

No modelo de dados definido, cada sensor tem que ter obrigatoriamente associado uma entrada na tabela `AlarmsSettings` que permite definir o valor máximo e mínimo para o qual são gerados alarmes bem como as mensagens de notificação associadas ao utilizador.

A geração de alarmes é condicionada pela comparação do valor lido pelos sensores com o valor máximo e mínimo definidos para o sensor em questão. Uma vez que é necessária esta verificação para cada leitura, decidiu-se criar um *trigger* em SQL que execute este procedimento, sendo este executado a nível da SGBD tornando o processo mais eficiente. Um *trigger* permite que uma determinada sequência de comandos sejam executadas sempre que um determinado evento ocorre, neste caso uma adição. No Apêndice B encontra-se um *stored procedure* (função) que implementa a sequência de comandos bem como a implementação do *trigger* na linguagem SQL. Na figura 5.3 encontra-se um diagrama de fluxo que permite ilustrar a lógica do *stored procedure* implementado.

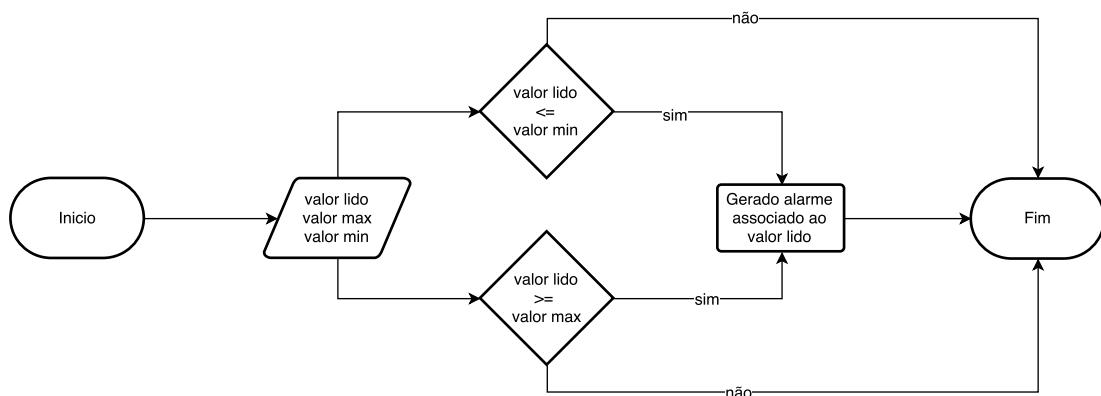


Figura 5.3: Diagrama de fluxo para geração de alarmes

O utilizador através da *dashboard* e da aplicação *mobile* poderá analisar os alarmes gerados bem como se o valor lido excede ou não atingiu o valor mínimo ou máximo, respetivamente. Para além disso, é apresentada a mensagem pré-definida (notificação) para que o utilizador possa tomar uma decisão.

<sup>3</sup>Módulo que define objetos para sessões *Simple Mail Transfer Protocol* (SMTP) <https://docs.python.org/3/library/smtplib.html#module-smtplib>

### 5.1.3 Visualização dos dados e cálculos estatísticos

O cliente do sistema ao aceder à *dashboard* e visualizar os detalhes de um determinado *Controller Module* ser-lhe-á apresentada uma lista de todos os *Sensor Modules* que este possui. Juntamente com esta lista são apresentadas as características principais de cada um, destacando os sensores existentes, o intervalo de tempo em que são enviadas dados, percentagem de bateria, coordenadas da localização do módulo e tipos de comunicação com o *Controller Module*. Complementarmente são apresentados quatro botões com icons ilustrativos que permitem realizar tarefas distintas.

- (editar): possibilita editar as características enumeradas anteriormente para cada *Sensor Module*.
- (remove): permite remover um determinado *Sensor Module* de um *Controller Module*.
- (visualização gráfica e atuação remota): é possível visualizar graficamente os dados obtidos pelos diferentes sensores existentes num *Sensor Module* bem como atuar remotamente num determinado atuador.
- (visualização tabular): disponibilizada em vista tabular uma visualização dos dados obtidos pelos sensores bem como a respetiva exportação para um ficheiro CSV.

Em ambas as visualizações anteriormente descritas é possível obter uma filtragem por data para os dados obtidos dos sensores de cada *Sensor Module*. Esta filtragem pode ser efetuada através de sete formas diferentes: do dia, do dia anterior, dos últimos sete dias, dos últimos trinta dias, deste mês, do último mês e selecionando a data de início e de fim especificamente.

No que toca à visualização gráfica toda a representação foi concebida recorrendo à biblioteca ZingChart em JS, sendo que toda a manipulação de dados é realizada a nível de *backend* e posteriormente apresentada através dos *templates* do Django. Esta biblioteca permite ainda que o cliente possa guardar os gráficos obtidos em formato de imagem (PNG, JPEG ou PDF). A análise estatística para o intervalo de data considerado permitirá ao cliente visualizar o valor máximo lido, o valor mínimo lido bem como o valor médio, sendo também todo o processamento realizado a nível de *backend*. Para além disso, a página de visualização gráfica também possibilitará a ativação/desativação remota dos atuadores existentes no terreno.

Por fim, relativamente à visualização tabular os dados são apresentados em três campos distintos que representam o tipo de sensor, data e hora da leitura, valor lido e a escala. A tabela contém paginação sendo que é possível escolher o número de entradas que serão apresentadas (10, 25, 50 ou 100). Complementarmente, existe uma caixa de texto que permite ao cliente pesquisar pelos campos apresentados, para além disso é possível ordenar alfabeticamente ou numericamente esses. O cliente também poderá exportar os dados apresentados num ficheiro do tipo CSV com a estrutura apresentada na tabela 5.1. Para a implementação

desta funcionalidade foram utilizados os métodos `writer` e `writerow()` disponíveis no pacote `csv` do Python.

ID	Sensor type	Scale	Date time	Value
579	Water valve	0/1	2017-05-31 09:30:14.762099+00:00	1
580	Temperature	C	2017-05-31 09:33:15.451236+00:00	25
581	Water level	0/1	2017-05-31 09:33:15.505198+00:00	1

Tabela 5.1: Estrutura do ficheiro do tipo CSV possível de exportação

#### 5.1.4 API

Tal como referido no capítulo anterior, para a implementação desta API do tipo REST foi utilizado o *Django REST framework*. Para tal, instalou-se a aplicação `'rest_framework'` no ficheiro de configuração do projeto Django, em `settings.py`. Para iniciar esta implementação procedeu-se à realização do *quickstart* disponível no site oficial da *framework*.

Primeiramente, procedeu-se à criação de um novo módulo `serializers.py` onde foram instanciadas todas as classes serializáveis<sup>4</sup> para cada `Model` existente bem como para os campos a considerar que possam ser usados na representação dos dados em formato JSON ou XML. De seguida, desenvolveu-se o módulo `apiViews.py` onde são implementadas todas as classes baseadas na `APIView`. A classe `APIView` é uma subclasse da `View` (abordada no capítulo 3) tendo algumas diferenças:

- Na classe `View` são retornados objetos do tipo `HttpRequest` ou `HttpResponse`, enquanto que na `APIView` são objeto do tipo `Request` e `Response`;
- A classe `APIView` permite a implementação dos seguintes métodos HTTP: `get()`, `post()`, `put()` e `delete()`;
- Através da classe `APIView` podem ser implementadas várias políticas da API.

A título de exemplo, o excerto de código seguinte encontra-se a implementação do *endpoint* `api/cm/{pk_or_name}` para o método GET.

```

1 #in serializers.py
2 class ControllerModuleSerializer(serializers.HyperlinkedModelSerializer):
3     id_communication = CommunicationTypeSerializer()
4     id_by_create = UserSerializer()
5
6     class Meta:
7         model = ControllerModule
8         fields = ('id', 'name', 'id_communication', 'id_by_create', 'baterry_cm',
9             'status_cm', 'date_create', 'memory', 'localization_cm')
```

<sup>4</sup>Consiste num processo de tradução de uma estruturas de dados ou de um objeto que pode ser armazenado (num arquivo ou *buffer* de memória) e reconstruído posteriormente.

```

9
10 #in apiViews.py
11 class ControllerModule_param(APIView):
12     def get_object(self, pk_or_name):
13         if pk_or_name.isdigit():
14             try:
15                 return ControllerModule.objects.get(pk=pk_or_name)
16             except ControllerModule.DoesNotExist:
17                 raise Http404
18         else:
19             try:
20                 return ControllerModule.objects.get(name__iexact=pk_or_name)
21             except ControllerModule.DoesNotExist:
22                 raise Http404
23
24     def get(self, request, pk_or_name, format=None):
25         cm = self.get_object(pk_or_name)
26         serializer = ControllerModuleSerializer(cm)
27         return Response(serializer.data)
28
29 #in url.py
30 url(r'^api/cm/(?P<pk_or_name>[-\w]+)/$', views.ControllerModule_param.as_view())

```

Como descrito anteriormente, a autenticação desta API usa um método de autenticação via *token*. Para tal, foi necessário a instalar a aplicação '`rest_framework.authtoken`' que fornece a estrutura Token, possibilitando a criação e modificação do *token* quando pretendido. Para que a autenticação seja possível é necessário incluir no módulo de configuração do Django o método pretendido, neste caso '`rest_framework.TokenAuthentication`'.

No anexo C encontram-se as especificações de cada *endpoint* existente nesta API REST.

### 5.1.5 Documentação da API

Para a utilização da documentação interativa da API REST foi utilizada a ferramenta Swagger. Para a sua utilização foi necessário incluir a aplicação '`rest_framework_swagger`' no ficheiro `settings.py` do Django e seguindamente definir o URL que lhe permitirá o acesso. Na figura 5.4 encontra-se um exemplo ao consultar a documentação desta API.

Figura 5.4: Documentação da API REST com a ferramenta Swagger

### 5.1.6 Aplicação web

O utilizador da *dashboard* após fazer a autenticação ser-lhe-à apresentado todos os *Controller Modules* a que tem acesso bem como as seguintes informações para cada um:

- Mapa onde é possível localizar todos os módulos associados a um *Controller Module*;
- Alarmes gerados pelos sensores que os *Sensor Modules* possuem e respetivas notificações;
- Últimos valores lidos pelo sensores dos *Sensor Modules*.

Cada um dos mapas interativo permitirá localizar o *Controller Module* bem como os *Sensor Modules* a si associados, sendo estes identificados por marcadores<sup>5</sup> de diferentes cores, no caso do *Controller Module*, a vermelho e nos *Sensor Modules*, a azul. A implementação destes mapas recorre à API do Google Maps em JS, permitindo receber a localização dos módulos em coordenadas GPS. Ao clicar sob um determinado marcador este irá direcionar o utilizador para a página de detalhes do módulo em questão.

Quando ocorre um alarme, o utilizador poderá verificar o motivo pelo qual este foi gerado, consultar a mensagem resultante bem como valor lido, tendo ainda a possibilidade de verificar/validar o alarme para que este não seja mais apresentado em destaque. Para além disso, o utilizador ao aceder à *dashboard* ser-lhe-ão apresentados os últimos valores lidos por cada sensor distribuídos por cada *Sensor Module*. Adicionalmente, na página principal da plataforma são apresentados alguns valores estatísticos da mesma: número de utilizadores registados, número de *Sensor Modules*, número de *Controller Modules* e número de leituras que já foram recolhidas pelos diferentes sensores.

A interface gráfica da plataforma permite ainda que o utilizador comum possa:

- Aceder à página de perfil possibilitando ao utilizador alterar as suas informações básicas e redefinir a sua *password*. Para além disso, também pode consultar o *token* de autenticação para a API;
- Consultar, adicionar ou editar os detalhes dos *Controller Modules* e dos *Sensor Modules*;
- Adicionar e consulta dos tipos de sensores e comunicações existentes;
- Alterar o avatar existente através da ferramenta *Gravatar* disponível em <http://pt.gravatar.com/>.

Sempre que o utilizador executa uma determinada ação na plataforma este é notificado informando-o se a ação foi bem sucedida ou não. A implementação deste mecanismo recorre à estrutura de mensagens<sup>6</sup> do Django, também conhecida por mensagens de *flash*.

<sup>5</sup>Um marcador identifica uma localização no mapa.

<https://developers.google.com/maps/documentation/javascript/markers?hl=pt-br>

<sup>6</sup><https://docs.djangoproject.com/en/1.11/ref/contrib/messages/>

### 5.1.7 Deploy do projeto

Para implementar o projeto Django juntamente com o servidor *web* Apache 2.0 num *Virtual Private Server* (VPS) Linux (Ubuntu 14.X), foram seguidos os seguintes passos:

1. Instalação do Python na versão 2.7 bem como pip<sup>7</sup> e respetivo *upgrade* para a versão mais recente

```
sudo apt-get install python-pip python-dev build-essential  
sudo pip install --upgrade pip
```

2. Instalação do PostgreSQL na ultima versão e criação de uma base de dados com o nome salibd

```
sudo apt-get install postgresql postgresql-contrib  
createdb salibd
```

3. Instalação do apache2 e do mod\_wsgi que fará a ligação do projeto Python com o servidor apache. Seguidamente dar reset ao apache2.

```
sudo apt-get install apache2  
sudo apt-get install libapache2-mod-wsgi
```

4. Configuração do virtualenv<sup>8</sup>, que consiste numa ferramenta para criar ambientes Python isolados. Por outro lado o virtualenvwrapper<sup>9</sup> permite uma fácil gestão do virtualenv, sendo que ao instalá-lo através do pip o virtualenv será instalado automaticamente.

```
pip install virtualenvwrapper
```

5. Criar um ambiente virtual (virtualenv) para o projeto. Para sua criação foi utilizada a opção `--system-site-packages` uma vez que a nossa aplicação terá que aceder a recursos fora do ambiente virtual, neste caso o PostgreSQL.

```
mkvirtualenv exampleenv --system-site-packages
```

6. Adicionar projeto e instalar dependências. Primeiramente é necessário ativar o virtualenv. Seguidamente procedeu-se à instalação do Django e instalação dos requisitos da aplicação (através do ficheiro requirements.txt disponibilizado pelo IDE).

```
workon exampleenv  
pip install Django  
cd /var/www
```

---

<sup>7</sup>Sistema de gestão de pacotes usado para instalar e gerir pacotes de software escritos na linguagem Python.

<sup>8</sup><https://virtualenv.pypa.io/en/stable/>

<sup>9</sup><https://virtualenvwrapper.readthedocs.io/en/latest/>

```
git clone https://github.com/ruipoliveira/ThesisSalicornia-web.git
pip install -r requirements.txt
```

7. Criação do virtual host. Terá que estar no directório `/etc/apache2/sites-available/` e criar um ficheiro `.conf` com o seguinte conteúdo.

```
1 <VirtualHost *:80>
2   ServerAdmin webmaster@mydomain.com
3   ServerName 192.168.160.20
4   ServerAlias http://192.168.160.20
5   WSGIScriptAlias / /var/www/example.wsgi
6
7   Alias /static/ /var/www/ThesisSalicornia-web/saliDashboard/saliapp/
8     static/
9     <Location "/static/">
10    Options -Indexes
11  </Location>
12
13  <Directory /var/www/ThesisSalicornia-web/saliDashboard>
14  Order deny,allow
15  Allow from all
16 </Directory>
17 </VirtualHost>
```

Por fim, é necessário ativar esta configuração através do comando

```
a2ensite example.conf
```

8. Criação do ficheiro `wsgi` no directório `/var/www/`

```
1 #file name 'example.wsgi'
2 import os
3 import sys
4 import site
5 # Add the site-packages of the chosen virtualenv to work with
6 site.addsitedir('/var/www/.virtualenvs/exampleenv/local/lib/python2.7/
7   site-packages')
8 # Add the app's directory to the PYTHONPATH
9 sys.path.append('/var/www/ThesisSalicornia-web/saliDashboard')
10 sys.path.append('/var/www/ThesisSalicornia-web/saliDashboard/
11   saliDashboard')
12 os.environ['DJANGO_SETTINGS_MODULE'] = 'saliDashboard.settings'
13 # Activate your virtual env
14 activate_env=os.path.expanduser("/var/www/.virtualenvs/exampleenv/bin/
15   activate_this.py")
16 execfile(activate_env, dict(__file__=activate_env))
17 from django.core.wsgi import get_wsgi_application
18 application = get_wsgi_application()
```

9. Sincronizar da base de dados através do comando

```
python manage.py migrate
```

## 5.2 Simulação em *hardware*

Nesta secção explica-se a implementação a nível de *software* no contexto desta simulação para cada um dos microcontroladores.

### 5.2.1 Arduino Nano

No que diz respeito ao Arduino Nano (*Sensor Module*), numa fase inicial, procedeu-se à ligação dos diversos componentes apresentados na secção 4.6.2 a uma placa branca (*breadboard*) tal como se apresentada no Anexo D. Para auxiliar o desenvolvimento de *software* foi utilizada a versão 1.8.1 do IDE do próprio Arduino<sup>10</sup>. Seguidamente apresenta-se a implementação necessária a nível de sensores e de comunicação.

Foram desenvolvidos os seguintes métodos que permitem aceder aos valores lidos de cada um dos sensores. Para além disso, foi criado um método que permite alterar o estado de ativação do LED, permitindo simular o estado da válvula para transferência de águas.

- **int readTemperature(int port):** é efetuada uma leitura no porto analógico através do método `analogRead` e seguidamente realizada uma conversão para °C (graus Celsius);
- **long readLuminosity(int port):** é efetuada uma leitura ao porto analógico e posteriormente é realizada uma conversão para percentagem (%);
- **int readWaterValve(int port):** é efetuada uma leitura no porto digital através do método `digitalRead` disponibilizado pelo Arduino.
- **int readWaterLevel(int port):** é realizada uma leitura no porto digital através do método `digitalRead`.
- **void setWaterValve(int port, int state):** se a variável `state` for 1 então o porto é colocado a HIGH (1) através do método `digitalWrite`, caso contrário é colocado a LOW (0)

Inicialmente procedeu-se à leitura de cada sensor de forma individual de modo a garantir o seu total funcionamento. Sempre que é enviado um pedido de leitura dos sensores pelo *Controller Module* os valores são enviados no formato apresentado em 5.1.

<temperatura>;<nível\_água>;<luminosidade>;<estado\_válvula> (5.1)

<sup>10</sup><https://www.arduino.cc/en/Main/Software>

Numa primeira fase, procedeu-se à comunicação entre o *Sensor Module* e *Controller Module* através de porta série. Seguidamente, optou-se por incorporar o módulo Bluetooth de modo a tornar os dois módulos independentes. Foi utilizado o *package SoftwareSerial.h* disponibilizado pelo Arduino, que permite interagir facilmente com o módulo de comunicação utilizado. Posteriormente, decidiu-se quais os *inputs* que o Arduino irá receber e que ações iria executar, concluindo-se que podiam ser recebidos valores entre 0 e 2.

- **0:** ativação do LED (válvula para transferência de água);
- **1:** desativação do LED (válvula para transferência de água);
- **2:** requisitar dados obtidos pelos sensores existentes no formato definido em 5.1.

Antes de proceder à implementação de envio e receção de dados por Bluetooth no Raspberry Pi 3, testou-se esta funcionalidade isoladamente. Para isso, utilizou-se uma aplicação existente na *Play Store* denominada de *Bluetooth Terminal HC-05*<sup>11</sup>, que permitiu facilmente validar este mecanismo. Os resultados deste teste funcional serão apresentados no próximo capítulo.

### 5.2.2 Raspberry Pi 3

Nesta simulação, o *Controller Module* recebe apenas os dados adquiridos por um *Sensor Module*, e envia as ações que este terá que executar. Para a comunicação entre os dois módulos, foi utilizado o módulo interno Bluetooth 4.1 disponível no *hardware* do Raspberry Pi 3. De modo a conseguir receber os dados adquiridos e enviá-los para o servidor através da API, desenvolveu-se um *script* em Python que permite o seguinte:

1. Verificação dos dispositivos Bluetooth disponíveis. Para aceder a este recurso, foi utilizada uma extensão do Python denominada de *pybluez*<sup>12</sup>.
2. Estabelecer uma ligação com módulo HC-06 através de um *socket* de comunicação (variável global na figura 5.5). Para tal, foi utilizado o *package socket* disponibilizado pelo Python.
3. Foram criados dois *threads* que permitem realizar funcionalidades distintas (figura 5.5):
  - **Controlar estado da válvula:** permite aceder à API de modo a verificar o estado do LED (válvula de admissão) e atualizá-lo sempre que necessário no *Sensor Module*, através do método *send()*. No caso de ser enviado o dígito '1' o LED será ligado (válvula aberta), enquanto que se for enviado o dígito '0' o LED será desligado (válvula fechada).

<sup>11</sup><https://play.google.com/store/apps/details?id=project.bluetoothterminal>

<sup>12</sup><https://github.com/karulis/pybluez>

- **Receção de dados:** no caso de ser enviado o dígito '2', o *socket* ficará a aguardar a receção dos dados lidos pelos sensores no formato definido, utilizando para isso o método `recv()`. Após a receção dos dados, é efetuado algum processamento para que estes possam ser enviado pela API, com um atraso (*seding time*) igual ao definido pelo utilizador na *dashboard*.

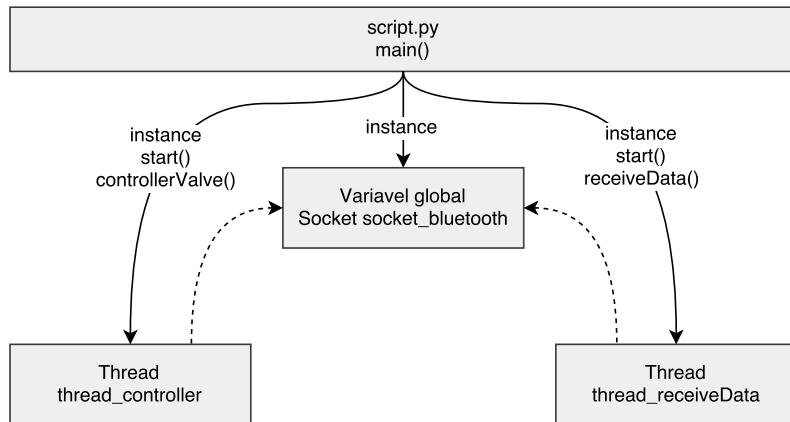


Figura 5.5: Lógica de implementação do *script* para o *Controller Module*

### 5.3 Sistema de videovigilância

Tal como descrito na secção 4.6.3, foram utilizadas as ferramentas FFMpeg e raspivid para envio de vídeo sem qualquer tipo de processamento para o servidor RTMP do Youtube. Para tal, foi executado o seguinte comando no terminal do Raspberry Pi 3, sendo possível observar o resultado na conta do Youtube.

```

1 raspivid -o - -t 0 -vf -hf -fps 30 -b 6000000 | ffmpeg -re -ar 44100 -ac 2 -
  acodec pcm_s16le -f s16le -ac 2 -i /dev/zero -f h264 -i - -vcodec copy -
  acodec aac -ab 128k -g 50 -strict experimental -f flv rtmp://a.rtmp.
  youtube.com/live2/ux4a-v8x7-3yws-635b
  
```

Seguidamente, são descritos alguns dos parâmetros mais importantes presentes no comando anterior[66].

- `-o -`: grava o vídeo para que possa ser utilizado pelo FFMpeg;
- `-t 0`: grava e transmite o vídeo até que seja parado manualmente;
- `-vf -hf`: inverte a imagem (horizontal/vertical) para permitir uma correta visualização;
- `-fps 30` : define o numero de frames por segundo, neste caso 30.

- **-ar 44100 -ac 2 -acodec pcm\_s16le -f s16le -ac 2 -i /dev/zero:** adiciona um canal de áudio preenchido a zeros, uma vez que o YouTube rejeita fluxos sem um canal de áudio;
- **-f h264:** diz ao FFmpeg que irá receber entrada h264 ( padrão para compressão de vídeo).

Para além do descrito anteriormente, foi criado um *script* em python que permite a aquisição de imagem proveniente do Raspberry Pi através do package **picamera**<sup>13</sup>. Este pacote, fornece uma interface em Python (disponível para qualquer versão) para o módulo de câmera Raspberry Pi, permitindo uma fácil interação entre a aquisição da imagem e respetivo processamento. Seguidamente será explicado um algoritmo de deteção de intrusos disponibilizado pelo OpenCV.

### 5.3.1 Algoritmo de deteção de intrusos

Para a implementação do algoritmo de deteção de intrusos comecei por estudar o módulo **peopledetect.py**<sup>14</sup> disponível no repositório github do OpenCV. Este algoritmo permite detetar pessoas recorrendo a algoritmos específicos do OpenCV já treinados para o efeito, que serão descritos seguidamente. Complementarmente, foi analisado um artigo[67] que permitiu entender melhor a implementação apresentada, evidenciando o processo de deteção de características, algoritmos de treino (neste caso *Support Vector Machine* (SVM)) e respetivos testes. No excerto de código seguinte encontra-se a implementação base fornecida pelo OpenCV para deteção de pessoas, que seguidamente será descrita.

```

1 hog = cv2.HOGDescriptor()
2 hog.setSVMClassifier(cv2.HOGDescriptor_getDefaultPeopleDetector())
3
4 (rects, weights) = hog.detectMultiScale(img, winStride=(8,8),
5 padding=(32,32), scale=1.05)

```

- **HOGDescriptor():** esta classe implementa um histograma de gradientes orientados[67] que tem como objetivo a deteção de objetos;
- **setSVMClassifier():** permite definir os coeficientes para um classificador SVM linear. O SVM é um algoritmo de *machine learning* usado para classificação e análise de regressões;
- **getDefaultPeopleDetector():** retorna um classificador pré-treinado para deteção de pessoas (para tamanho de janela padrão)[68];

<sup>13</sup><http://picamera.readthedocs.io/en/release-1.13/>

<sup>14</sup><https://github.com/npinto/opencv/blob/master/samples/python2/peopledetect.py>

- `detectMultiScale()`: permite detetar objetos de diferentes tamanhos. Este método possui vários parâmetros que permitem ajustar o tamanho de deteção de um determinado objeto, como por exemplo o `winStride`, `padding` ou a `scale`;
  - `winStride`: define o ”tamanho do passo” na posição x e y de uma janela deslizante (figura 5.6). Este parâmetro é opcional;
  - `padding`: este parâmetro é opcional e define o preenchimento. Consiste num tuplo que indica o número de pixels, na direção x e y, tal como o `winStride`. Os valores típicos são (8, 8), (16, 16), (24, 24) e (32, 32);
  - `scale`: este parâmetro controla o fator com que a imagem é redimensionada em cada camada da pirâmide de imagens (figura 5.7), influenciando o número de níveis existentes. É um parâmetro opcional.

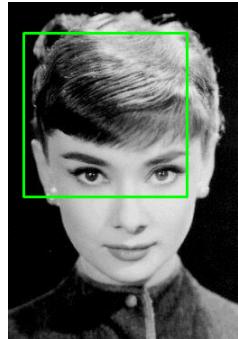


Figura 5.6: Exemplo da aplicação do parâmetro `winStride`

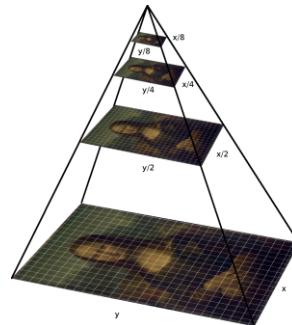


Figura 5.7: Pirâmide de imagens para diferentes `scale`

O algoritmo apresentado permite a deteção de intrusos (humanos), quando estes surgem em vídeos ou imagens. Pretende-se incorporar este algoritmo no sistema de videovigilância (Youtube Live), possibilitando a deteção de indivíduos quando estes são capturados pela câmara, gerando alarmes para a plataforma. Para incorporar o algoritmo apresentado com o Youtube Live é necessário entender o mecanismo de *streaming* disponibilizado pela API do YouTube Live Streaming<sup>15</sup>.

Como vimos anteriormente, existem vários parâmetros para o método `detectMultiScale()`, que permitem ajustar o tamanho da deteção. Dependendo da disposição exata da câmara é possível ajustar estes parâmetros permitindo uma deteção o mais eficaz possível. No próximo capítulo serão apresentados alguns exemplos em que é possível testar o algoritmo apresentado.

<sup>15</sup><https://developers.google.com/youtube/v3/live/getting-started>

## 5.4 Considerações finais

Neste capítulo foram apresentadas algumas soluções de implementação que achei serem mais relevantes. Seguidamente encontram-se algumas conclusões finais da implementação de cada componente e no próximo capítulo serão mostrados os testes e resultados.

- **Sistema de informação:** após a modulação do sistema de informação foi possível implementá-lo de forma a permitir ilustrar o cenário pretendido, embora também seja possível adaptá-lo a outros quadros. O sistema disponibiliza uma interface *web* intuitiva e apelativa, sendo que toda a comunicação com os componentes de *hardware* seja realizada através da API REST. Esta API permite a criação de novos cliente com base no sistema, um exemplo disso é a aplicação *mobile* que infelizmente não houve tempo para ser desenvolvida, sendo apenas apresentado um *mockup* desta.
- **Simulação em hardware:** simulou-se o sistema recorrendo a *hardware* disponível em laboratório, tanto a nível de microcontroladores, sensores ou módulos de comunicação. No entanto, não houve possibilidade de testar nenhum sensor de salinidade apesar de ser bastante importante no contexto do projeto, uma vez que é um dos parâmetros principais necessários de monitorização no cultivo da Salicórnia. Relativamente à comunicação desta simulação, numa primeira fase, toda a comunicação entre o *Controller Module* e *Sensor Module* foi realizada através de porta série (cabo físico), posteriormente optou-se por utilizar uma tecnologia de comunicação sem fios.
- **Sistema de videovigilância:** relativamente ao sistema de videovigilância foi possível testar o serviço do Youtube Live para este cenário, ficando a falta a incorporação do algoritmo de deteção com a API do Youtube Live.

# Testes e resultados

## 6.1 Testes funcionais

Nesta secção são apresentados alguns testes a nível de funcionalidades do sistema. Estes testes permitem averiguar se determinados blocos, que sejam possíveis de testar isoladamente, se encontram em total funcionamento.

### 6.1.1 API REST

Após a criação da API REST foram utilizadas duas ferramentas, uma gráfica e outra por linha de comandos, que permitiram testar e personalizar os cabeçalhos num pedido HTTP, sendo cada uma delas descrita de seguida.

- *Advanced REST client*<sup>1</sup>: consiste numa ferramenta gráfica (extensão para o Google Chrome) que permite auxiliar os programadores *web* na criação e testes de pedidos HTTP personalizados. É o único cliente REST que faz a conexão diretamente no *socket*, fornecendo controlo total sobre os cabeçalhos de ligação e solicitações/resposta.
- CURL<sup>2</sup>: é uma biblioteca (libcurl) e ferramenta de linha de comandos (cURL) para transferências de dados através do URL. Esta ferramenta suporta uma variedade de protocolos comuns da Internet com por exemplo HTTP, *File Transfer Protocol* (FTP), SMTP entre outros.

Estas duas ferramentas permitiram testar e validar o funcionamento da API REST através da utilização dos métodos GET, PUT, POST e DELETE para cada *endpoint*, quando aplicado. De notar que para todos os testes foi necessário incorporar o campo *Authorization*

<sup>1</sup><https://advancedrestclient.com/>

<sup>2</sup><https://curl.haxx.se/>

para autenticar a utilização da API através de um *token* fornecido. A figura 6.1 e o comando abaixo apresentado permitem ilustrar um teste para o método GET no endpoint `api/sm` através da ferramenta gráfica e na de linha de comandos, respetivamente.

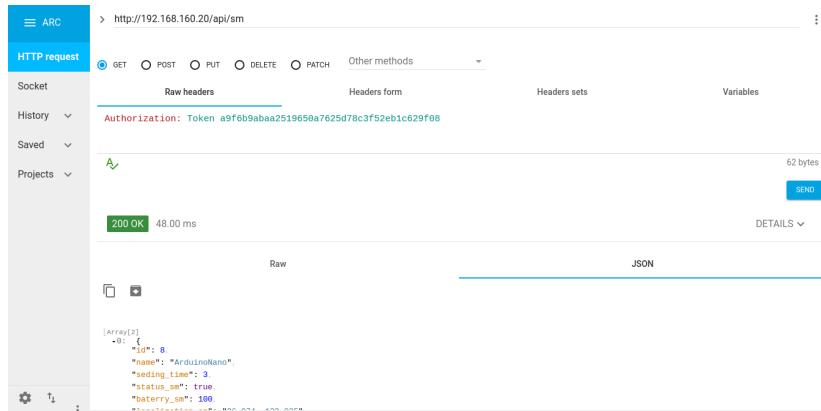


Figura 6.1: Resultado da utilização da ferramenta *Advanced REST client*

```

1 $ curl -X GET -H "Authorization: Token 79
2   e546740afe1aa4fb8d09a897146763e9f1b835" http://192.168.160.20/api/cm/
3   [{"id":4,"name":"Rasp3","id_communication":{"id":5,"name":"wireless","path_or_number":""}, "image_path": "earth-grid.png"}, {"id":12,"username":"josesilva","first_name":"Jose","last_name":"silva","email":"ruipedrooliveira@ua.pt","last_login":"2017-07-12T15:34:01.669706Z","date_joined":"2017-05-29T16:07:33.102064Z"}, {"battery_cm":100,"status_cm":true,"date_create":"2017-05-31T09:07:10.300203Z,"memory":512,"localization_cm":36.964,-122.015"}]
  
```

### 6.1.2 Comunicação via Bluetooth

Para testar o módulo Bluetooth HC-06 foi utilizada a aplicação *Bluetooth Terminal HC-05*. Esta possibilitou enviar vários *inputs* descritos, ou seja, os algarismos zero (0), um (1) e dois (2) e observar o seu resultado.

Caso seja enviado o 1 através da caixa de texto "Enter Command" ou através do botão "ON val" previamente criado, é ativado o LED. Por outro lado, caso seja enviado o 0, o LED desliga-se. Isto mimetiza a abertura e o fecho de um válvula para transferência de águas nas leiras de produção de Salicórnia, respetivamente enviando 1 e 0. Na situação de ser enviado o algarismo 2, o módulo Bluetooth encarrega-se de enviar a informação recolhida pelos sensores no formato descrito na secção 5.2.1. A figura 6.8 demonstra a manipulação do estado da válvula (LED) e respetiva resposta de confirmação (ON/OFF OK!), e por fim o envio dos valores lidos pelos sensores: temperatura a 28°C, sensor do nível de água ativo, luminosidade a 60% e a válvula ativa. Na figura 6.3 é possível observar o resultado após o envio do algarismo 1 com a activação do LED (E da figura 6.3). Para além disso, é possível observar a ligação dos

diferentes componentes na *breadboard*: o Arduino Nano (A da figura 6.3), módulo Bluetooth HC-06 (B da figura 6.3), sensor de luminosidade (C da figura 6.3) e o de temperatura (D da figura 6.3).

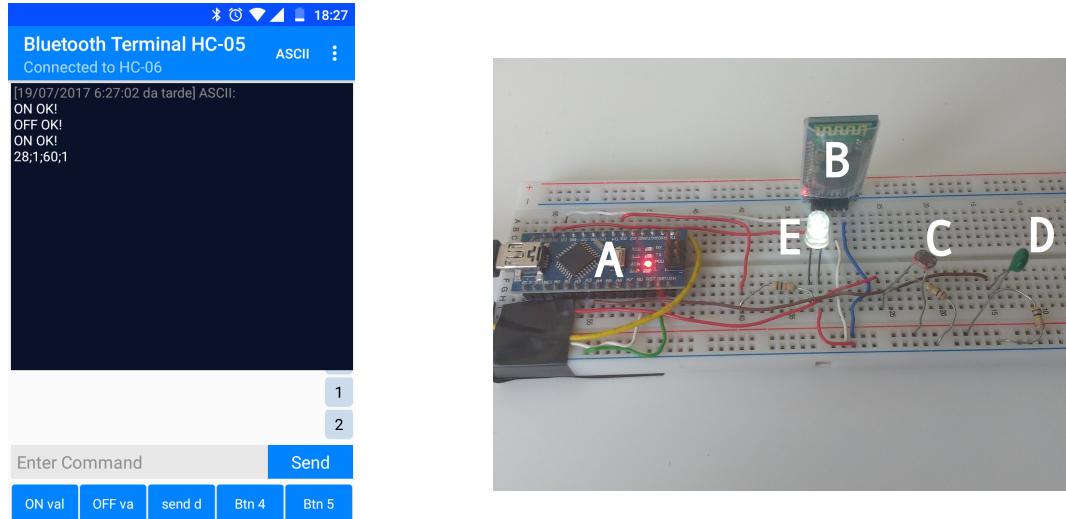


Figura 6.2: Resultado da interação com a aplicação *Bluetooth Terminal HC-05*

Figura 6.3: *Breadboard* com ligação dos diferentes componentes, destacando-se a ativação do LED

### 6.1.3 Deteção de intrusos

Relativamente ao algoritmo de deteção de intrusos, este foi testado em três cenários distintos, representados nas figuras abaixo (frame1, frame2 e frame3). É possível averiguar que em nenhum deles os valores atribuídos aos parâmetros referidos na secção 5.3.1 são totalmente iguais.



Figura 6.4: Imagem original (frame1)

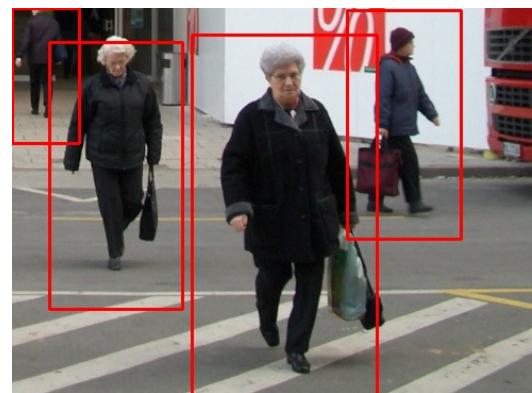


Figura 6.5: Resultado obtido (frame1) em que  $winstride = (4, 4)$ ,  $padding = (8, 8)$  e  $scale = 1.1$



Figura 6.6: Imagem original (frame2)

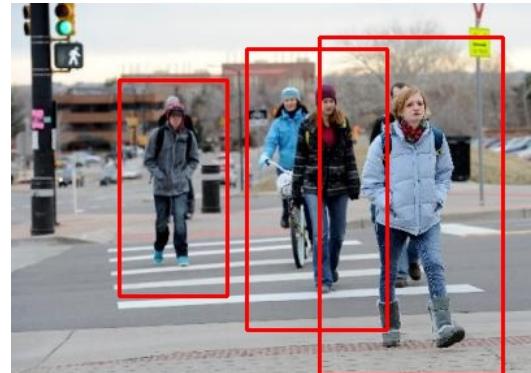
Figura 6.7: Resultado obtido (frame 2) em que  $winstride = (4,4)$ ,  $padding = (24,24)$  e  $scale = 1.3$ 

Figura 6.8: Imagem original (frame3)

Figura 6.9: Resultado obtido (frame 3) em que  $winstride = (4,4)$ ,  $padding = (8,8)$  e  $scale = 0.7$ 

Como podemos ver, para os três casos de teste não existe nenhum conjunto de parâmetros que satisfaça todos os cenários, o que torna impossível saber quais serão usados no contexto da câmara de videovigilância, uma vez que depende da localização e do ângulo da mesma no terreno.

## 6.2 Resultados

Nesta secção são apresentados os principais resultados obtidos no que toca ao trabalho prático desta dissertação. É apresentado o sistema de registo dos utilizadores e respetiva validação por parte da empresa, a adição de um novo tipo de sensor na *dashboard* bem como a adição de um novo *Sensor Module*. Para além disso, é mostrado o resultado do envio contínuo de dados para o cenário de simulação em *hardware* apresentado. Por fim, é apresentado o mecanismo de atuação remota numa válvula e o respetivo resultado obtido.

Figura 6.10: Interface para novo registo, destacando o *feedback* dado ao utilizador

Figura 6.11: Interface para validação de utilizadores, com *feedback* após validação

Na figura 6.10 é apresentada a interface *web* para o registo de um novo utilizador no sistema. Este ao registar-se terá que escolher a empresa a que se associa, sendo esta notificada por email da existência de um novo utilizador. A empresa terá que aceder à sua área privada para validar ou remover o utilizador associado (figura 6.11).

Figura 6.12: Interface para visualizar sensores

Figura 6.13: Interface para adicionar sensores

A figura 6.12 apresenta a página que permite verificar todos os tipos de sensores existentes no sistema, por outro lado, a figura 6.13 exemplifica a adição de um sensor de salinidade com a especificação das suas características. Este dois passos são análogos ao processo de visualização e adição de tipos de comunicação.

Figura 6.14: Visualização dos *Sensor Modules* associados a um *Controller Module*

Figura 6.15: Interface para adição de um novo *Sensor Module*

Na figura 6.14 é possível observar os *Sensor Modules* associados ao *Controller Module* denominado por Raspi. São visíveis todas as características de cada *Sensor Module*, bem como os botões que permitem a adição, edição, remoção ou visualização dos dados adquiridos. Por outro lado, na figura 6.15 é apresentada a interface para adição de um novo *Sensor Module* e respetivos *feedbacks* para o utilizador. O processo de visualização e adição de um *Controller Module* é idêntico.

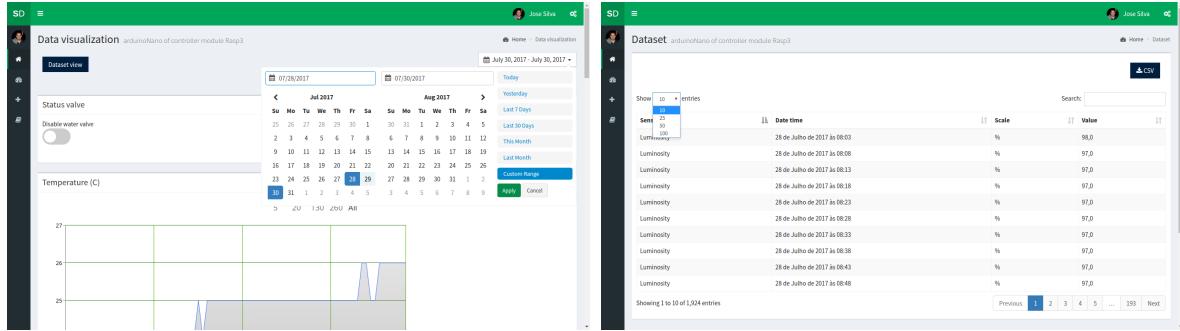


Figura 6.16: Visualização de dados destacando a filtragem por data

Figura 6.17: Visualização tabular de dados, destacando a exportação para CSV



Figura 6.18: Visualização gráfica dos dados adquiridos por um sensor de luminosidade durante quatro dias, destacando os valores máximos, mínimos e médios

Na figura 6.16 é apresentada a interface de visualização gráfica dos dados adquiridos pelos diferentes sensores de um *Sensor Module* ou por outros componentes existentes, isto é, atuação remota ou sistema videovigilância. Nesta figura é destacada a filtragem por data, a atuação

remota e o botão "Dataset View" que permite visualizar os dados em formato tabular (figura 6.17). Estes dados são passíveis de serem ordenados pelos diferentes campos bem como a sua exportação para um ficheiro do tipo CSV através de um botão existente.

Seguidamente, na figura 6.18 é apresentado o resultado dos dados adquiridos para o sensor de luminosidade durante quatro dias, sendo que eram recebidos dados dos sensores de 5 em 5 minutos. Na mesma interface é possível observar o número de leituras efetuadas nesta data, o valor máximo e mínimo medido e o valor médio. Os valores 5, 20, 130 e 250 são botões que permitem alterar o numero de leituras apresentadas no gráfico.

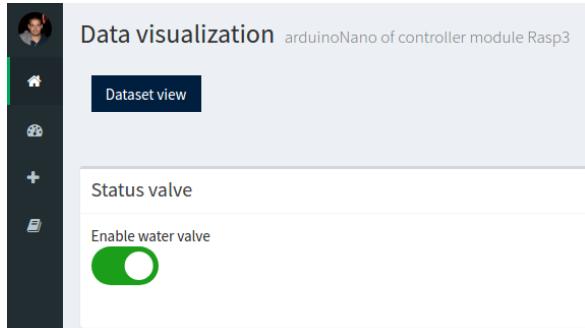


Figura 6.19: Processo de atuação remota



Figura 6.20: Resultado do processo de atuação remota

Na figura 6.19 é possível observar o processo de atuação remota através de ativação do *switch*, sendo apresentada a mensagem "Enable water valve". A figura 6.20 apresenta o resultado da atuação remota onde é possível observar a alteração do estado de abertura da válvula (de zero para um, fechado e aberto, respetivamente).

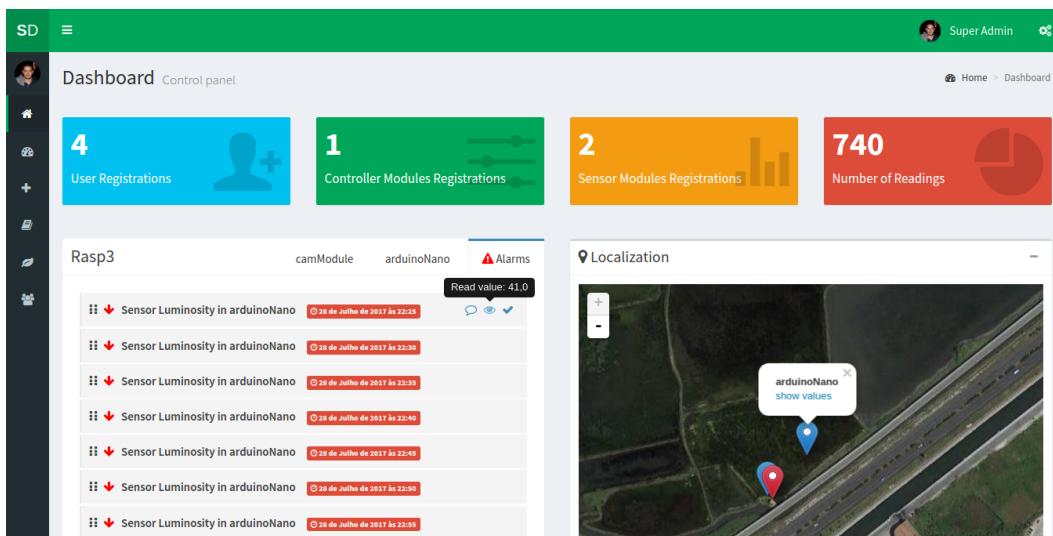


Figura 6.21: Interface principal após *login* (*dashboard*)

Na figura 6.21 é apresentada a *dashboard* do sistema após a validação bem sucedida das

credenciais do utilizador. Nesta página são apresentados alguns dados estatísticos relacionadas com o sistema, como por exemplo, o número de utilizadores registado, números de *Controller Modules* e *Sensor Modules* e número total de valores obtidos pelos sensores no sistema. Para cada *Controller Module* é apresentado um mapa onde é possível localizar cada um dos módulos identificados através de um marcador de diferentes cores (a vermelho no caso do *Controller Module* e a azul nos *Sensor Modules*). No separador "Alarms" é possível consultar todos os alarmes ocorridos destacando o valor que gerou o alarme bem como a mensagem gerada. É possível verificar o alarme para que este não volte a ser apresentado. No separador "arduinoNano", são apresentados os últimos valores obtidos pelos sensores deste módulo.

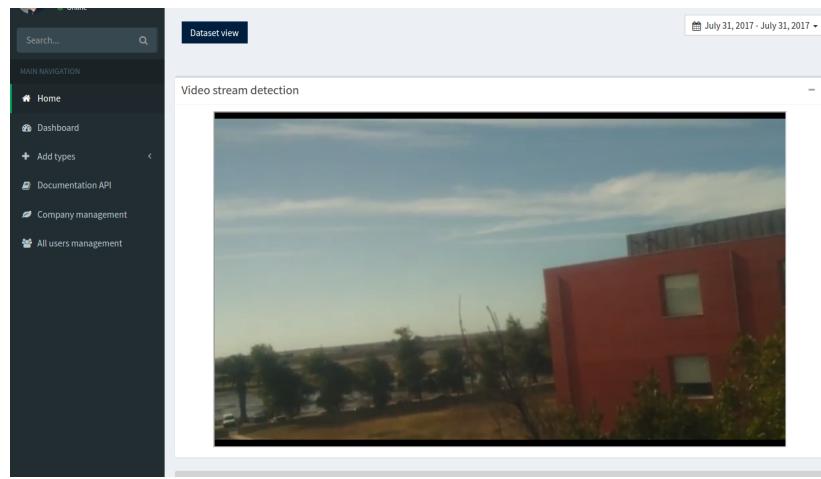


Figura 6.22: Sistema de videovigilância incorporado na *dashboard*

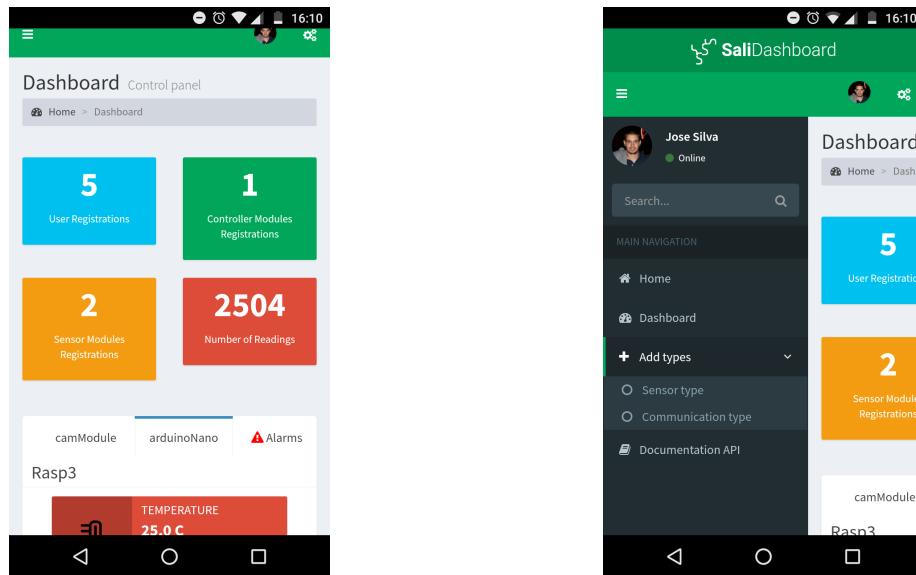


Figura 6.23: Resultado da interface responsiva da plataforma *web*

Na figura 6.22 é apresentada a incorporação do sistema de videovigilância na plataforma *web* do sistema. Por fim, na figura 6.23 é apresentado o acesso à *dashboard* através de um dispositivo *mobile*, sendo possível constatar o seu *design* totalmente responsivo, permitindo utilizar esta interface como ponto de partida para criação da aplicação *mobile* planeada.

### 6.3 Considerações finais

Neste capítulo são apresentados alguns dos testes funcionais a alguns módulos do sistema, bem como os resultados dos diferentes componentes do sistema integrados. A plataforma desenvolvida permite que o seu utilizador possa consultar os alarmes gerados e verificar a evolução dos dados lidos pelos diferentes sensores. Complementarmente o utilizador poderá atuar remotamente em atuadores de forma a melhorar as condições de cultivo da Salicórnia. Adicionalmente, este sistema disponibiliza uma API que permite a criação de novas aplicações. De forma a simular este cenário, foi criado um protótipo em *hardware* com diferentes sensores e microcontroladores, utilizando protocolos de comunicação sem fios. Por fim, foi criado um sistema de videovigilância e testado um algoritmo de deteção de intrusos.



# Conclusões e trabalho futuro

## 7.1 Conclusões

O objetivo deste trabalho consistiu em desenhar e desenvolver um sistema de informação que permitisse o armazenamento dos dados provenientes de um sistema de sensores para monitorizar e controlar o cultivo da Salicórnia. O trabalho prático desta dissertação foi elaborado tendo por base este objetivo geral e pode-se afirmar que este foi cumprido com sucesso. Este sistema disponibiliza uma plataforma *web* que permite aos utilizadores consultar os dados obtidos pelos sensores e atuar remotamente permitindo melhorar as condições de cultivo. Para além disso, é disponibilizada uma API que permite o acesso a serviços do sistema, possibilitando a criação de novas aplicações. Para simular e testar o quadro pretendido, foi criado um protótipo de *hardware*. Adicionalmente, foi criado um sistema de videovigilância para incorporar nas quintas onde se faz a produção desta planta. Todas estas funcionalidades vão de encontro aos objetivos específicos apresentados na secção 1.2, à exceção da incorporação do sistema de videovigilância com o algoritmo de deteção de intrusos. Contudo, este algoritmo foi apresentado e testado para alguns cenários, permitindo concluir que os parâmetros utilizados dependem do ângulo e da posição da câmara. Na figura 7.1 encontra-se um esquema que permite resumir todo o trabalho realizado nesta dissertação.

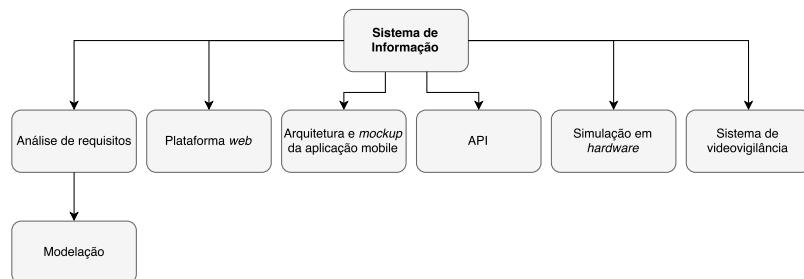


Figura 7.1: Esquema resumo do trabalho desenvolvido

Toda a modelação do sistema vai de encontro aos requisitos inicialmente especificados pelo cliente, bem como aos definidos durante o desenvolvimento deste trabalho. Desta forma, é permitido que o sistema seja genérico e passível de ser aplicado a qualquer cenário, seguindo a arquitetura definida. Adicionalmente aos objetivos desta dissertação, planeou-se a arquitetura e criou-se um *mockup* de uma aplicação mobile, estando esta prevista pelos requisitos do cliente.

O sistema de informação criado poderá ser utilizado como ponto de partida para qualquer objetivo, desde que respeite a arquitetura inicialmente definida, isto é, composta por *Controller Modules* e *Sensor Modules*.

## 7.2 Problemas encontrados

Durante o desenvolvimento e implementação deste sistema surgiram alguns problemas, tanto pontuais e de correção simples, como problemas estruturais, que levaram a algumas mudanças. Alguns dos problemas estruturais estão relacionados com o modelo de dados, em que foram adicionados novos campos às tabelas existentes, quer para o suporte de novas funcionalidades, quer para aumentar o comportamento dinâmico do sistema.

Tal como referido anteriormente, não foi possível incorporar o sistema de videovigilância com o algoritmo de deteção de intrusos, sendo que para tal, pretendia-se utilizar a API do Youtube. Esta utilização não foi possível devido à reduzida documentação da API que dificultou a sua implementação. Para além disso, existem poucos exemplos que permitem entender eficazmente a utilização da API.

Um outro obstáculo na realização deste trabalho, foi o facto de o projeto não ser financiado por parte do cliente, impossibilitando assim, a compra de um sensor de salinidade(condutividade), sendo este um dos parâmetros mais importante de monitorizar no controlo do cultivo da Salicornia. Adicionalmente, pretendia-se adquirir uma câmara de maior qualidade e outra térmica (infravermelhos), permitindo a deteção de intrusos durante a noite.

## 7.3 Trabalho futuro

Como trabalho futuro, propõe-se realizar alguns testes de usabilidade à aplicação *web* permitindo verificar o grau de facilidade/dificuldade de utilização deste *software*. Outra situação que poderia ser interessante pretende-se com automatizar o registo dos módulos através da leitura de um código *Quick Response* (QR) por um aplicação *mobile* criada para o efeito. Por fim, pretende-se criar um circuito impresso do protótipo de *hardware* desenvolvido.

# Bibliografia

- [1] *The Engineering Design Process*. [Online]. Available: <https://www.sciencebuddies.org/engineering-design-process/engineering-design-process-steps.shtml>
- [2] P. T. Hiep, H. Noi, V. Nam, N. H. Hoang, H. Noi, and V. Nam, “A Review of Open Source Software Development Life Cycle Models,” *International Journal of Multimedia and Ubiquitous Engineering*, vol. 9, no. 5, pp. 391–402, 2014. [Online]. Available: <http://dx.doi.org/10.14257/ijseia.2014.8.3.38>
- [3] João Silva, “Sal verde, National Geographic.” [Online]. Available: <https://nationalgeographic.sapo.pt/23-arquivo/as-nossas-historias/298-sal-verde> [Accessed: 2017-02-01]
- [4] S. Beer and O. Demina, “A new species of *Salicornia* (Chenopodiaceae) from European Russia,” pp. 253–257, 2005.
- [5] M. Ferri and N. Menezes, *Glossário Ilustrado de Botânica*, 1st ed., Livraria Nobel, Ed., Brasil, 1981.
- [6] M. H. A. Silva, “Aspectos morfológicos e ecofisiológicos de algumas halófitas do sapal da Ria de Aveiro,” Ph.D. dissertation, Universidade de Aveiro, 2000. [Online]. Available: <http://ria.ua.pt/handle/10773/925>
- [7] V. Isca, A. Seca, D. Pinto, and A. Silva, *An overview of Salicornia genus: the phytochemical and pharmacological profile*, natural pr ed., V. Gupta, Ed. Daya Publishing House, New Delhi, 2014.
- [8] E. Figueroa, J. Jimenez-Nieva, J. Carranza, and C. Gonzalez Vilches, “Distribucion y Nutricion Mineral de *Salicornia ramosissima* J. Woods, *Salicornia europaea* L. y *Salicornia dolichostachya* Moss. en el estuario de los rios Odiel y Tinto (Huelva, SO España),” *Limnetica*, vol. 3, no. 2, pp. 307–310, 1987.
- [9] R. Pinto, “Expresso — A planta que é uma alternativa ao sal: antes era uma praga, agora é uma erva gourmet,” 2015. [Online]. Available: <http://bit.ly/1PR7KAG> [Accessed: 2017-02-01]

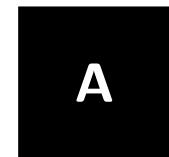
- [10] A. J. Davy, G. F. Bishop, and C. S. B. Costa, “*Salicornia* L. (*Salicornia pusilla* J. Woods, *S. ramosissima* J. Woods, *S. europaea* L., *S. obscura* P.W. Ball & Tutin, *S. nitens* P.W. Ball & Tutin, *S. fragilis* P.W. Ball & Tutin and *S. dolichostachya* Moss),” *Journal of Ecology*, vol. 89, no. 4, pp. 681–707, 2001.
- [11] H. Silva, G. Caldeira, and H. Freitas, “*Salicornia ramosissima* population dynamics and tolerance of salinity,” *Ecological Research*, vol. 22, no. 1, pp. 125–134, 2007.
- [12] A. Rubio-Casal, J. Castillo, C. Luque, and M. Figueroa, “Influence of salinity on germination and seeds viability of two primary colonizers of Mediterranean salt pans,” *Journal of Arid Environments*, vol. 53, no. 2, pp. 145–154, feb 2003. [Online]. Available: <http://linkinghub.elsevier.com/retrieve/pii/S0140196302910426>
- [13] M. Filomena, D. J. Raposo, R. Manuel, S. Costa, A. Maria, and M. Bernardo, “Controlled atmosphere storage for the preservation of *Salicornia ramosissima*,” no. October 2016, 2009.
- [14] Y. Ventura, W. A. Wuddineh, M. Myrzabayeva, Z. Alikulov, I. Khozin-Goldberg, M. Shpigel, T. M. Samocha, and M. Sagi, “Effect of seawater concentration on the productivity and nutritional value of annual *Salicornia* and perennial *Sarcocornia* halophytes as leafy vegetable crops,” *Scientia Horticulturae*, vol. 128, no. 3, pp. 189–196, apr 2011. [Online]. Available: <http://linkinghub.elsevier.com/retrieve/pii/S0304423811000537>
- [15] Q. Z. Wang, X. F. Liu, Y. Shan, F. Q. Guan, Y. Chen, X. Y. Wang, M. Wang, and X. Feng, “Two new nortriterpenoid saponins from *Salicornia bigelovii* Torr. and their cytotoxic activity,” *Fitoterapia*, vol. 83, no. 4, pp. 742–749, jun 2012. [Online]. Available: <http://www.ncbi.nlm.nih.gov/pubmed/22414316> <http://linkinghub.elsevier.com/retrieve/pii/S0367326X12000640>
- [16] D. Evans, “A Internet das Coisas Como a próxima evolução da Internet está mudando tudo,” pp. 5–7, 2011.
- [17] B. Getting, “Basic Definitions: Web 1.0, Web 2.0, Web 3.0 — Practical Ecommerce.” [Online]. Available: <http://www.practicalecommerce.com/articles/464-Basic-Definitions-Web-1-0-Web-2-0-Web-3-0> [Accessed: 2017-02-20]
- [18] J. Lovato, “Google’s evolution in 10 years,” 2014. [Online]. Available: <http://www.mediavisioninteractive.com/blog/search-enginenews/looking-back-moving-forward-google-evolution/> [Accessed: 2017-02-20]
- [19] T. Our, “Resume : Context Aware Computing for The Internet of Things : A Survey Article 2013,” pp. 1–5, 2013.

- [20] J. Rowley, “The wisdom hierarchy: representations of the DIKW hierarchy,” *Journal of Information Science*, vol. 33, no. 2, pp. 163–180, 2007. [Online]. Available: <http://alturl.com/7qike>
- [21] MySQL, “MySQL :: About MySQL,” 2011. [Online]. Available: <http://www.mysql.com/about/> [Accessed: 2017-07-13]
- [22] “SQL Server 2017 on Windows and Linux — Microsoft,” 2017. [Online]. Available: <https://www.microsoft.com/en-us/sql-server/sql-server-2017> [Accessed: 2017-07-17]
- [23] The PostgreSQL Global Development Group, “PostgreSQL: About,” 2012. [Online]. Available: <https://www.postgresql.org/about/> [Accessed: 2017-05-29]
- [24] DB-Engines, “Historical Trend of the Popularity Ranking of Database Management Systems,” 2014. [Online]. Available: <http://db-engines.com/en/ranking{ }trend> [Accessed: 2017-07-13]
- [25] I. V. López and G. B. Gutiérrez, “El Benchmark TPC-H en MySQL y PostgreSQL,” *Ingenieria.Lm.Uasnet.Mx*, pp. 1–7, 2009. [Online]. Available: <http://ingenieria.lm.uasnet.mx/sitio/congreso/documentos/iso15.pdf>
- [26] Microsoft, “ASP.NET Overview,” p. 1, 2016. [Online]. Available: <https://msdn.microsoft.com/en-us/library/4w3ex9c2.aspx> [Accessed: 2017-07-17]
- [27] Flask, “Welcome — Flask (A Python Microframework),” 2014. [Online]. Available: <http://flask.pocoo.org/> [Accessed: 2017-07-17]
- [28] J. Deacon, “Model-View-Controller (MVC) Architecture,” *JOHN DEACON Computer Systems Development, Consulting & Training*, pp. 1–6, 2009. [Online]. Available: <http://www.jdl.co.uk/briefings/index.html{ #}mvc>
- [29] D. S. Foundation, “Django Documentation,” pp. 1–1172, 2012. [Online]. Available: <https://docs.djangoproject.com/en/1.3/> [Accessed: 2017-07-17]
- [30] J. Sugrue, *Beginning Backbone.js*, 2013.
- [31] A. Holovaty and J. Kaplan-Moss, *The Definitive Guide to Django: Web Development Done Right*, 2009.
- [32] Ibm, “Native, web or hybrid mobile-app development,” *Thought Leadership White Paper*, pp. 0–7, 2012.
- [33] Ionic, “Welcome to Ionic - Ionic Framework,” 2016. [Online]. Available: <http://ionicframework.com/docs/guide/preface.html> [Accessed: 2017-07-19]

- [34] I. S. Kang, R. Moraes, and W. Santos, “Padrão TISS: Um Estudo de Caso sobre o uso de web services para o seu desenvolvimento,” *12th CONTECSI International Conference on Information Systems and Technology Management*, pp. 1–17, 2015. [Online]. Available: <http://www.contecsi.fea.usp.br/envio/index.php/contecsi/12CONTECSI/paper/view/2185>
- [35] A. Holmes, “Reviewing Django REST Framework,” 2014. [Online]. Available: <http://blog.isotoma.com/2014/03/reviewing-django-rest-framework/> [Accessed: 2017-07-20]
- [36] Flask-RESTful, “Flask-RESTful — Flask-RESTful 0.2.1 documentation,” 2017. [Online]. Available: <http://flask-restful-cn.readthedocs.io/en/0.3.5/> [Accessed: 2017-07-29]
- [37] M. Banzi, D. Cuartielles, T. Igoe, G. Martion, and D. Mellis, “Arduino - Introduction,” 2012. [Online]. Available: <http://arduino.cc/en/Guide/Introduction> [Accessed: 2017-05-25]
- [38] “Arduino Nano - User Manual.” [Online]. Available: <https://www.arduino.cc/en/uploads/Main/ArduinoNanoManual23.pdf>
- [39] J. Melorose, R. Perroy, and S. Careas, “Arduino Nano,” 2015. [Online]. Available: <http://www.farnell.com/datasheets/1682238.pdf> [Accessed: 2017-07-17]
- [40] Raspberry Pi Foundation, “Raspberry Pi Foundation - About Us,” 2012. [Online]. Available: <https://www.raspberrypi.org/about/> [Accessed: 2017-07-17]
- [41] REOTEMP Instrument Corporation, “Thermocouple-Thermocouples-What is a thermocouple-Types of thermocouples.” [Online]. Available: <http://www.thermocoupleinfo.com/index.htm> [Accessed: 2017-07-17]
- [42] “Temperature Sensors Watlow Educational Series.” [Online]. Available: [https://kontrolotomasyon.files.wordpress.com/2016/10/dt{\\_-}temperaturesensors{\\_-}thewatlow{\\_}25ekim2016.pdf](https://kontrolotomasyon.files.wordpress.com/2016/10/dt{_-}temperaturesensors{_-}thewatlow{_}25ekim2016.pdf)
- [43] Argus, “Light and lighting contril in greenhouses,” no. August, pp. 1–29, 2010. [Online]. Available: <http://www.arguscontrols.com/resources/Light-and-Lighting-Control-in-Greenhouses.pdf>
- [44] “Salinity Sensor (Order Code SAL-BTA) Collecting Data with the Salinity Sensor.” [Online]. Available: <https://www.vernier.com/files/manuals/sal-bta.pdf>
- [45] A. B. A. Rahman, “Comparison of Internet of Things ( IoT ) Data Link Protocols,” pp. 1–21, 2015. [Online]. Available: <http://www.cse.wustl.edu/{~}jain/cse570-15/index.html>

- [46] R. Bruno, M. Conti, and E. Gregori, “Bluetooth : Architecture , Protocols and Scheduling Algorithms,” *Cluster Computing*, vol. 5, no. 2, pp. 117–131, 2002. [Online]. Available: <http://link.springer.com/10.1023/A:1013989524865>
- [47] Bluetooth(TM), “Bluetooth Specification,” *Specification of the Bluetooth System*, vol. 1, p. 1084, 2001. [Online]. Available: <http://www.bluetooth.com>
- [48] W. Paper, “IEEE 802.11g The New Mainstream Wireless LAN Standard,” 2005. [Online]. Available: <http://www.broadcom.com/products/index>
- [49] “IEEE 802.11g — Wi-Fi WLAN — Tutorial - Radio-Electronics.Com.” [Online]. Available: <http://www.radio-electronics.com/info/wireless/wi-fi/ieee-802-11g.php> [Accessed: 2017-07-18]
- [50] “Sigfox Technology Overview — Sigfox.” [Online]. Available: <https://www.sigfox.com/en/sigfox-iot-technology-overview> [Accessed: 2017-07-18]
- [51] Laudon, C. Kenneth, Laudon, and P. Jane, *Management Information Systems New Approaches to Organization & Technology*. Prentice Hall, 1998.
- [52] E. Turban, *Information technology for management : improving quality and productivity*. Wiley, 1996. [Online]. Available: [https://books.google.pt/books?id=FqxzQgAACAAJ&redir\\_esc=y&hl=pt-PT](https://books.google.pt/books?id=FqxzQgAACAAJ&redir_esc=y&hl=pt-PT)
- [53] “Gravatar - Globally Recognized Avatars.” [Online]. Available: <http://pt.gravatar.com/site/implementhttps://secure.gravatar.com/> [Accessed: 2017-06-10]
- [54] R. T. Fielding, “UNIVERSITY OF CALIFORNIA, IRVINE Architectural Styles and the Design of Network-based Software Architectures,” 2000. [Online]. Available: <http://jpkc.fudan.edu.cn/picture/article/216/35/4b/22598d594e3d93239700ce79bce1/7ed3ec2a-03c2-49cb-8bf8-5a90ea42f523.pdf>
- [55] T. Christie, “Django REST framework TokenAuthentication,” 2016. [Online]. Available: <http://www.django-rest-framework.org> [Accessed: 2017-07-03]
- [56] Ado Kukic, “Cookies vs Tokens: The Definitive Guide,” 2016. [Online]. Available: <https://auth0.com/blog/cookies-vs-tokens-definitive-guide> [Accessed: 2017-07-03]
- [57] SmartBear Software, “Swagger – The World’s Most Popular Framework for APIs.” 2017. [Online]. Available: <http://swagger.io/> [Accessed: 2017-06-07]
- [58] The Apache Software Foundation, “Foundation Project,” 2016. [Online]. Available: <https://www.apache.org/foundation/http://apache.org/foundation/> [Accessed: 2017-06-12]

- [59] Google, “AngularJS — Superheroic JavaScript MVW Framework,” 2015. [Online]. Available: <https://angularjs.org/> [Accessed: 2017-07-29]
- [60] “Datasheet, NTC Thermistor TTC05 Series, Disc Type for Temperature Sensing/Compensation.” [Online]. Available: <http://extra-parts.com/datasheets/TTC.pdf>
- [61] L. LIDA OPTICAL&ELECTRONIC CO., “Datasheet, CdS Photoconductive cells, GL5528,” p. 1. [Online]. Available: <https://pi.gate.ac.uk/pages/airpi-files/PD0001.pdf> [Accessed: 2017-05-24]
- [62] L. Guangzhou HC Information Technology Co ., “HC06 Datasheet,” no. 13, pp. 1–17, 2011.
- [63] Itseez, “About - OpenCV library.” [Online]. Available: <http://opencv.org/about.html> [Accessed: 2017-05-19]
- [64] “raspivid - Raspberry Pi Documentation.” [Online]. Available: <https://www.raspberrypi.org/documentation/usage/camera/raspicam/raspivid.md> [Accessed: 2017-07-25]
- [65] FFmpeg, “About FFmpeg,” 2015. [Online]. Available: <https://www.ffmpeg.org/about.html> [Accessed: 2017-07-25]
- [66] “Stream Live with Raspberry Pi Camera.” [Online]. Available: <https://www.digikey.com/en/maker/blogs/streaming-live-to-youtube-and-facebook-using-raspberry-pi-camera/969a7932d47d42a79ba72c81da4d9b66> [Accessed: 2017-07-29]
- [67] N. Dalal and B. Triggs, “Histograms of Oriented Gradients for Human Detection,” *Proc. Int. Conf. Computer Vision and Pattern Recognition*, pp. 886–893, 2005. [Online]. Available: <http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf>
- [68] “Feature Detection — OpenCV 2.4.13.2 documentation [ONLINE].” [Online]. Available: [http://docs.opencv.org/2.4/modules/gpu/doc/object{\\_-}detection.html](http://docs.opencv.org/2.4/modules/gpu/doc/object{_-}detection.html) [Accessed: 2017-07-10]



## Mockup da aplicação mobile

Nas figuras A.1 e A.2 são apresentados os *mockups* da aplicação *mobile* prevista.

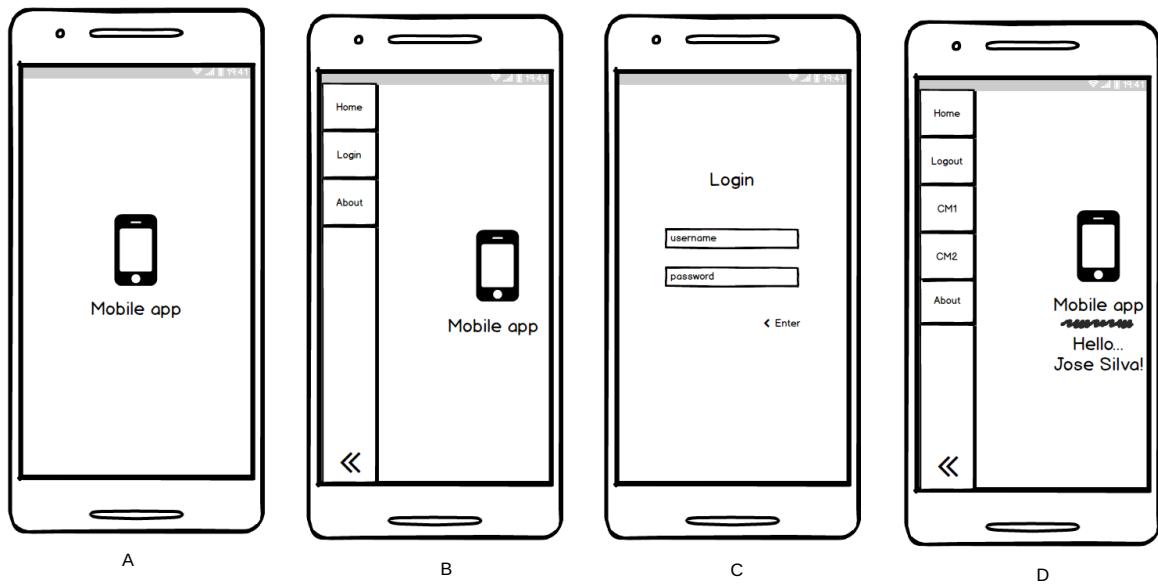


Figura A.1: *Mockup* da aplicação *mobile*

- **A:** página inicial da aplicação *mobile*;
- **B:** menu lateral deslizante (*sidebar*) onde são apresentados os diferentes botões para as diferentes funcionalidades sem autenticação do utilizador;
- **C:** página de *login* na aplicação *mobile*;
- **D:** página inicial e *sidebar* após efecutar o *login* do utilizador.

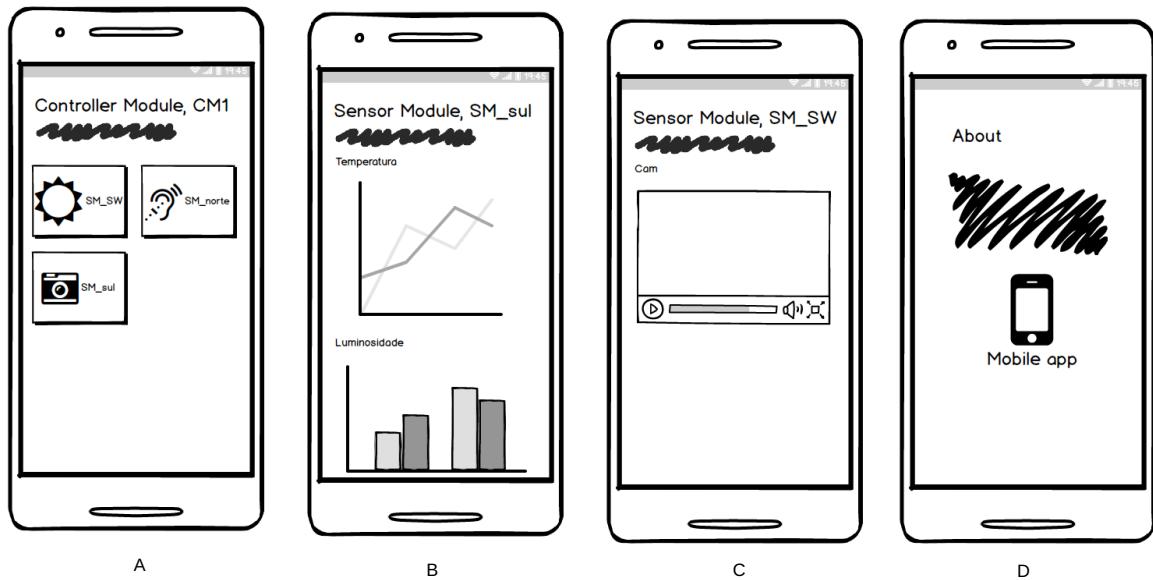


Figura A.2: *Mockup* da aplicação mobile (continuação)

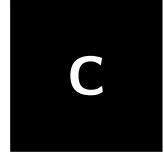
- **A:** página de detalhes de um *Controller Module*, onde são apresentados os botões para cada *Sensor Module* existente;
- **B:** página de detalhes de um *Sensor Module*, onde são apresentados os dados adquiridos pelos sensores em modo gráfico;
- **C:** página para visualização do sistema de video-vigilância;
- **D:** página de informações da aplicação.

# Implementação do trigger SQL

Seguidamente encontra-se o *script* SQL da implementação do *stored procedure* e respetivo *trigger*. Os dois últimos comandos permitem eliminar a *stored procedure* e o *trigger*, respetivamente.

```
1 CREATE OR REPLACE FUNCTION alarm_occurred() returns trigger as $alarm$  
2 DECLARE  
3     varmax FLOAT;  
4     varmin FLOAT;  
5 BEGIN  
6  
7     varmax := (select max from saliapp_alarmssettings where id_sensor_id= new.  
8         id_sensor_id);  
9     varmin := (select min from saliapp_alarmssettings where id_sensor_id= new.  
10        id_sensor_id);  
11  
12    IF (new.value >= varmax) THEN  
13        insert into saliapp_alarms (id_reading_id, checked, max_or_min) VALUES (new.id  
14            , 'f' , 't');  
15    RETURN new;  
16    END IF;  
17    IF (new.value <= varmin) THEN  
18        insert into saliapp_alarms (id_reading_id, checked, max_or_min) VALUES (new.id  
19            , 'f' , 'f');  
20    RETURN new;  
21    END IF;  
22  
23    RETURN NULL;  
24    END  
25    $alarm$  
26 LANGUAGE plpgsql;
```

```
24 create trigger trigger_alarm_occurred after insert on saliapp_reading
25 for each row execute procedure alarm_occurred();
26
27 DROP FUNCTION alarm_occurred();
28
29 DROP TRIGGER trigger_alarm_occurred ON saliapp_reading;
```



C

# Application Programming Interface REST

Seguidamente encontram-se descritos cada um dos *endpoint* da API REST desenvolvida e os métodos que este permite.

- /api/user/
  - **Métodos disponíveis:** POST, GET
  - **Descrição:** retorna os utilizadores registados no sistema, distinguindo o seu id, username, primeiro e ultimo nome, email, data do registo e do último acesso. É também indicado o tipo de utilizador a que se refere.
- /api/user/{pk\_or\_username}/
  - **Métodos disponíveis:** GET, PUT, DELETE
  - **Descrição:** permite aplicar os métodos a um determinado utilizador registado no sistema, sendo este identificado pelo seu identificador ou pelo username. É retornado o primeiro e último nome, email, data do registo e do último acesso.
- /api/smpercm/
  - **Métodos disponíveis:** GET
  - **Descrição:** permite visualizar todos os *Sensor Modules* que os *Controller Modules* possuem.

- /api/smpercm/{pk\_or\_name\_cm}
  - **Métodos disponíveis:** GET, POST
  - **Descrição:** permite visualizar todos os *Sensor Modules* que um determinado *Controller Module* possui, sendo este identificado por um nome ou pelo seu id no sistema. É também possível adicionar um novo *Sensor Module* ao *Controller Module* em questão.
- /api/sm/
  - **Métodos disponíveis:** GET, POST
  - **Descrição:** são apresentadas todas as características dos *Sensor Modules* existentes no sistema. Permite ainda adicionar um novo *Sensor Module*.
- /api/sm/{pk\_or\_name}/
  - **Métodos disponíveis:** GET, PUT, DELETE
  - **Descrição:** são apresentadas todas as características de um *Sensor Module*, sendo este identificado pelo seu nome ou pelo seu id. É possível atualizar as suas características ou eliminar o *Sensor Module*.
- /api/sensortype/
  - **Métodos disponíveis:** GET, POST
  - **Descrição:** são apresentados todos os tipos de sensores existentes no sistema e os seus respetivos atributos. Permite também adicionar novos tipos de sensores ao sistema.
- /api/sensortype/{pk\_or\_name}
  - **Métodos disponíveis:** GET, PUT, DELETE
  - **Descrição:** são apresentadas as características de um tipo de sensor existente, sendo este identificado por um nome ou id. É possível atualizar ou eliminar este tipo de sensor.
- /api/sensorpersm/{id\_sm\_or\_name\_sm}
  - **Métodos disponíveis:** GET, POST
  - **Descrição:** são apresentados todos os sensores existentes num determinado *Sensor Module*, sendo este identificado por um nome ou id. É possível adicionar novos sensores a um *Sensor Module*.

- /api/sensor/
  - **Métodos disponíveis:** GET
  - **Descrição:** são retornados todos os sensores registados no sistema.
- /api/sensor/{pk\_or\_sensor\_type}
  - **Métodos disponíveis:** GET, POST
  - **Descrição:** são retornadas as características de um sensor, sendo este identificado pelo tipo de sensor ou id.
- /api/reading/id\_sensor/{date\_start}/{date\_end}
  - **Métodos disponíveis:** GET, POST
  - **Descrição:** são apresentados todas as leituras de um determinado sensor, identificado por um id, sendo possível definir a data de início e fim das leituras apresentadas. É também possível adicionar novas leituras ao sistema.
- /api/communication/{pk\_or\_name}
  - **Métodos disponíveis:** GET, PUT, DELETE
  - **Descrição:** são retornados todos os tipos de comunicação, sendo estes identificados pelo seu nome ou id. Para além disso, é possível atualizar os seus dados ou eliminá-lo.
- /api/cm/
  - **Métodos disponíveis:** GET, POST
  - **Descrição:** são retornados todos os *Controller Modules* existentes no sistema. Para além disso, é possível adicionar um novo *Controller Module*.
- /api/cm/{pk\_or\_name}
  - **Métodos disponíveis:** GET, PUT, DELETE
  - **Descrição:** são retornadas as características de um determinado *Controller Module*, sendo este identificado pelo seu nome ou id. É possível atualizar os seus dados ou eliminá-lo.
- /api/alarmssettings/{id\_sensor}
  - **Métodos disponíveis:** GET, POST
  - **Descrição:** são apresentadas as configurações de alarmes para um determinado sensor, sendo este identificado pelo seu id.

- /api/alarms\_sensor/{id\_sensor}
  - **Métodos disponíveis:** GET, POST
  - **Descrição:** são apresentados todos os alarmes gerados para um determinado sensor, sendo este identificado pelo seu id.
- /api/alarms\_reading/{id\_reading}
  - **Métodos disponíveis:** GET, POST
  - **Descrição:** permite verificar se uma determinada leitura, identificada pelo seu id, foi ou não alvo de um alarme.

D

## Interligação de componentes

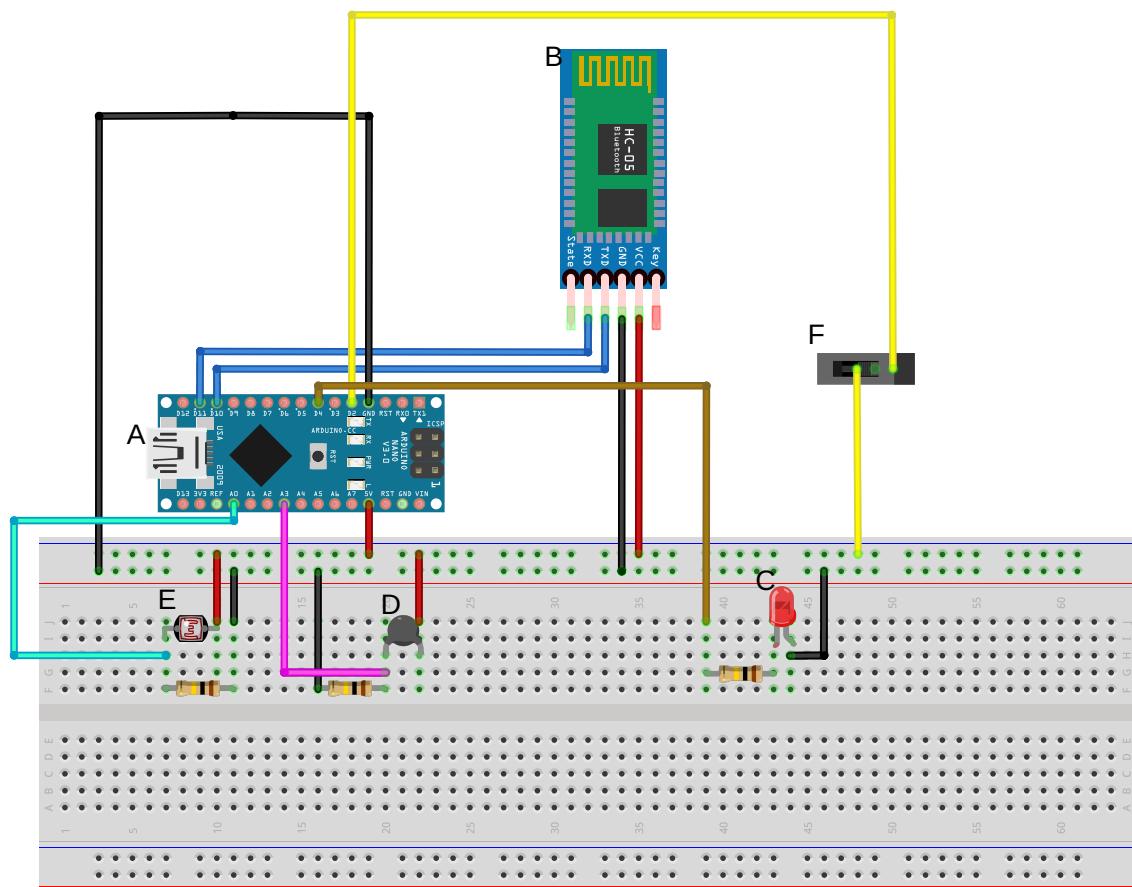


Figura D.1: Protótipo da solução em *hardware* desenvolvida

Na figura D.1 encontra-se o esquema da interligação dos diferentes componentes de *hardware* numa placa de prototipagem (placa branca).

**Legenda:**

- **A:** Arduino Nano
- **B:** Módulo Bluetooth HC-06
- **C:** LED
- **D:** Sensor de temperatura TTC 104 NTC
- **E:** Sensor de luminosidade GL5528 (foto-resistência)
- **F:** Sensor para nível de água