



Rui Pedro dos
Santos Oliveira

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

**Monitorization and control system for 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 for 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 for 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 da Universidade de Aveiro (orientador)

GHI

Professor associado da Universidade J (co-orientador)

KLM

Professor Catedrático da Universidade N (arguente)

**agradecimentos /
acknowledgements**

palavras chave	cultivo da salicórnia, sensores, monitorização, atuadores, atuação remota, plataforma web, plataforma mobile, sistema de vídeo-vigilância
resumo	A evolução da Internet tem proporcionado o crescimento de novos modelos de negócio suportados por tecnologias de informação e comunicações e tem possibilitado o desenvolvimento de novos modelos de serviços baseados no cloud computing. A exploração de novas descobertas na área da percepção visual, nomeadamente no que se refere à apreciação de obras de arte geniais, . . .

keywords Cultivo da salicórnia, irrigação, sensores, atuadores, web, monitorização, atuação remota.

abstract Nowadays, it is usual to evaluate a work ...

Conteúdo

Lista de Figuras	v
Lista de Tabelas	ix
Acrónimos	xi
1 Introdução	1
1.1 Motivação	2
1.2 Objetivos	3
1.3 Organização do documento	3
2 O Internet of Things (IoT) no cultivo da Salicornia	5
2.1 Características da planta	5
2.2 Condições ideais de cultivo da Salicornia	6
2.3 Importância da planta	7
2.4 Evolução tecnológica: o IoT	7
2.5 Considerações finais	9
3 Estado da arte	11
3.1 Conceitos tecnológicos	11
3.2 Sistema de Gestão de Base de Dados	12
3.2.1 MySQL	12
3.2.2 SQL Server	12
3.2.3 PostgreSQL	12
3.2.4 Comparação e solução adotada	12
3.3 Desenvolvimento web	13
3.3.1 ASP.net	14
3.3.2 Flask	14
3.3.3 Django	14
3.3.4 Conclusões e solução adotada	15

3.4	Desenvolvimento mobile	16
3.4.1	Plataformas nativas	16
3.4.2	Multi-plataforma	16
3.4.3	Conclusões e solução adotada	17
3.5	REST Frameworks	18
3.5.1	Django Rest Framework	18
3.5.2	Restlet	19
3.5.3	Flask-RESTful	19
3.5.4	Conclusões e solução adotada	19
3.6	Microcontroladores	20
3.6.1	Arduino Nano	20
3.6.2	Raspberry Pi 3	21
3.7	Sensores	22
3.7.1	Sensor de temperatura	22
3.7.2	Sensor de luminosidade	23
3.7.3	Sensor de salinidade	24
3.8	Tecnologias de comunicação	24
3.8.1	Zigbee	24
3.8.2	Bluetooth (BLE)	24
3.8.3	Wi-Fi (IEEE 802.11)	25
3.8.4	Sigfox	25
3.8.5	Comparação entre tecnologias de comunicação	25
4	Sistema de controlo e monitorização: arquitetura e modelação	27
4.1	Descrição global do sistema	27
4.2	Componentes	28
4.2.1	<i>Sensor Module</i>	29
4.2.2	<i>Controller Module</i>	30
4.3	Análise de requisitos	30
4.3.1	Requisitos funcionais	32
4.3.2	Requisitos não funcionais	33
4.4	Modelação	34
4.4.1	Entidades envolvidas	34
4.4.2	Casos de uso	34
4.4.3	Modelo de dados	39
4.5	Arquitetura lógica	42
4.6	Arquitetura física	43
4.6.1	Sistema de informação	44
	Aplicação web	44

API REST	46
Documentação interativa	48
Implementação do sistema	48
Aplicação mobile	49
4.6.2 Simulação em <i>hardware</i>	50
Sensores utilizados	50
Comunicação	52
4.6.3 Sistema de vídeo-vigilância	54
Biblioteca para processamento de imagem: OpenCV	54
4.7 Diagrama de componentes	57
4.8 Considerações finais	58
5 Implementação	59
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 Application Programming Interface (API)	64
5.1.5 Documentação da API	65
5.1.6 Aplicação web	66
5.1.7 <i>Deploy</i> do projecto	67
5.1.8 Aplicação mobile	69
5.1.9 Considerações finais	69
5.2 Simulação em <i>hardware</i>	70
5.2.1 Arduino Nano	70
Sensores	70
Comunicação	71
5.2.2 Raspberry Pi 3	71
5.2.3 Considerações finais	72
5.3 Sistema de vídeo-vigilância	73
5.3.1 Algoritmo de deteção de intrusos	73
5.3.2 Testes	74
5.3.3 Implementação	75
5.3.4 Considerações finais	75
6 Testes e resultados	77
6.1 Testes funcionais	77
6.1.1 API REST	77
6.1.2 Comunicação via Bluetooth	78

6.1.3	Deteção de intrusos	79
6.2	Cenário de teste	80
6.3	Interface mobile	82
6.4	Simulação em hardware	82
6.5	Sistema de deteção de intrusos	83
6.6	Considerações finais	83
7	Conclusão e trabalho futuro	85
7.1	Conclusão	85
7.2	Trabalho futuro	85
7.3	Considerações finais	85
A	Application Programming Interface REST	93
B	Mockups da aplicação mobile	97
C	Trigger SQL	99
D	Interface gráfica	101
E	Descrição formal dos casos de uso gerais	103
F	Interligação de componentes	105

Lista de Figuras

1.1	Salicórnia na Ria de Aveiro (Fotografia por José M. G. Pereira)	1
2.1	Coloração da planta <i>Salicornia ramosissima</i> na primavera (à esquerda) e no outono (à direita) (Fotografia por José M. G. Pereira)	6
2.2	Esquema representativo do ciclo de vida da <i>Salicornia ramosissima</i>	6
2.3	Evolução da internet em cinco fases	8
2.4	Pirâmide do conhecimento: modelo DIKW	9
3.1	Arduino Nano	20
3.2	Identificação dos pinos num Arduino Nano	20
3.3	Raspberry Pi 3	21
3.4	Principais componentes no Raspberry Pi 3	21
3.5	Sensibilidade luminosa das plantas durante a fotossíntese)	23
4.1	Ilustração dos principais componentes do sistema	27
4.2	Ilustração da distribuição dos módulos em duas leiras	28
4.3	Esquema de componentes e respetiva comunicação entre três <i>Sensor Module</i> (SM) e um <i>Controller Module</i> (CM)	29
4.4	Fase de desenvolvimento de um software	31
4.5	Casos de uso para a aplicação web (dashboard)	35
4.6	Casos de uso para a aplicação mobile	36
4.7	Esquema relacional da estrutura da base de dados	39
4.8	Arquitetura lógica	42
4.9	Arquitetura física (blocos)	43
4.10	Arquitetura do sistema de informação (<i>dashboard</i> , base de dados e API)	45
4.11	Processo de autenticação em HyperText Transfer Protocol (HTTP) através de token	47
4.12	Arquitetura da aplicação mobile	49
4.13	Sensor TTC 104 NTC	50
4.14	Esquema eletrotécnico da ligação do sensor de temperatura	50

4.15 Sensor foto-resistência GL5528	51
4.16 Esquema eletrotécnico da ligação do sensor de luminosidade	51
4.17 <i>Water Level Switch Liquid Level Sensor Plastic Ball Float</i>	52
4.18 Esquema eletrotécnico da ligação do sensor de nível líquido	52
4.19 Light Emitting Diode (LED)	52
4.20 Esquema eletrotécnico da ligação do LED	52
4.21 Comunicação entre componentes da simulação em <i>hardware</i>	53
4.22 Módulo Bluetooth HC-06	54
4.23 Esquema eletrotécnico da ligação do módulo Bluetooth	54
4.24 Raspberry Pi Camera Board V2 8MP 1080p	55
4.25 Arquitetura do sistema de video stream	56
4.26 Diagrama final de componentes do sistema	57
 5.1 Painel administrativo do 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	66
5.5 Threads e ddd... implementacao	72
5.6 Sensor TTC 104 NTC	74
5.7 Piramide ..Retirado de	74
 6.1 Documentação da API REST com a ferramenta Swagger	78
6.2 Resultado da interação com a aplicação <i>Bluetooth Terminal HC-05</i>	79
6.3 <i>Breadboard</i> com ligação dos diferentes componentes, destacando-se a ativação do LED	79
6.4 Imagem original (frame 1)	79
6.5 Resultado obtido (frame 1)	79
6.6 Imagem original (frame 1)	81
6.7 Resultado obtido (frame 1)	81
6.8 Imagem original (frame 1)	81
6.9 Resultado obtido (frame 1)	81
6.10 Imagem original (frame 1)	81
6.11 Resultado obtido (frame 1)	81
6.12 Imagem original (frame 1)	82
6.13 Resultado obtido (frame 1)	82
6.14 Imagem original (frame 1)	82
6.15 Resultado obtido (frame 1)	82
6.16 Imagem original (frame 1)	82
6.17 Resultado obtido (frame 1)	82

D.1 Pirâmide do conhecimento: modelo DIKW	101
D.2 Pirâmide do conhecimento: modelo DIKW	102
F.1 Protótipo de montagem de componentes eletrotécnicos	105

Lista de Tabelas

3.1	<i>Ranking BD-Engines</i> para popularidade dos Sistema de Gestão de Base de Dados (SGBD) estudados	13
3.2	Comparação entre MySQL, SQL Server e PostgreSQL	13
3.3	Comparação entre frameworks REST estudadas.	19
3.4	Características do Arduino Nano	21
3.5	Comparação entre versão 2 e 3 do Raspberry Pi	22
3.6	Comparação entre tecnologias de comunicação	25
4.1	Especificação das tabelas existentes no sistema	40
4.2	Especificação das tabelas existentes no sistema (continuação)	41
4.3	Endpoints da API REST e respetivos métodos a implementar	48
4.4	Características do sensor TTC 104	51
4.5	Características do sensor GL5528	51
4.6	Características do módulo bluetooth HC-06	54
4.7	Características do módulo bluetooth HC-06	55
5.1	Estrutura do ficheiro do tipo Comma-Separated Values (CSV) possível de exportação	64
B.1	Um nome qualquer	97

Acrónimos

API	Application Programming Interface	REST	Representational State Transfer
CM	<i>Controller Module</i>	SDLC	Systems Development Life Cycle
CSS	Cascading Style Sheets	SGBD	Sistema de Gestão de Base de Dados
CSV	Comma-Separated Values	SM	<i>Sensor Module</i>
DOM	Document Object Model	SQL	Structured Query Language
FK	Foreign Key	UI	User Interface
GPS	Global Positioning System	URL	Uniform Resource Locator
HTML	HyperText Markup Language	WSGI	Web Server Gateway Interface
HTTP	HyperText Transfer Protocol	WWW	World Wide Web
I/O	Input/ Output	JS	JavaScript
IDE	Integrated Development Environment	ORM	Object-Relational Mapping
IoT	<i>Internet of Things</i>	CPU	Central Processing Unit
LDR	Light Dependent Resistor	RAM	Random Access Memory
NTC	Negative Temperature Coefficient	DIKW	Data-Information-Knowledge-Wisdom
ORM	Object Relational Mapper	ISM	Industrial, Scientific, Medical
PK	Primary keys	LED	Light Emitting Diode
REST	Representational State Transfer		

IP	Internet Protocol	IIS	Internet Information Services
EDR	Enhanced Data Rate	MVC	Model-View-Controller
SMTP	Simple Mail Transfer Protocol	MTV	Model-Template-View
TCP	Transmission Control Protocol	ROM	Read-Only Memory
JSON	JavaScript Object Notation	USB	Universal Serial Bus
VPS	Virtual Private Server	PANs	Wireless personal area networks
SVM	Support Vector Machine	HATEOAS	Hypermedia As The Engine Of Application State
FTP	File Transfer Protocol	SOAP	Simple Object Access Protocol
DOM	Modelo de Objeto de Documento	CSI	Camera Serial Interface
PAR	Photosynthetically Active Radiation	RTMP	Real-Time Messaging Protocol
ASP	Active Server Pages		

Introdução

Os recursos naturais, nomeadamente, plantas, animais e minerais, são utilizados desde a antiguidade pelo ser humano, não apenas como fonte de alimentos mas também para o tratamento de diversas doenças [1]. Muitas das espécies que nascem em todo o mundo inicialmente são consideradas pragas, contudo e após alguns estudos intensivos à espécie são descobertas verdadeiras pérolas. Um exemplo disso é a salicornia.



Figura 1.1: Salicornia na Ria de Aveiro (Fotografia por José M. G. Pereira)

<http://eusougourmet.blogspot.pt/2011/09/compre-o-que-e-nosso-salicornia.html>

* O gênero salicornia inclui cerca de 117 espécies, sendo *Salicornia herbacea*, *Salicornia bigelovii*, *Salicornia europea*, *Salicornia prostata*, *Salicornia mmosissima* e *Salicornia virginica* aquelas com maior ocorrência.

A que serve de mote a esta dissertação ...

A salicornia é a planta que iremos dar destaque durante este projeto. Esta planta é por vezes utilizada como substituta do sal marinho [1] e utilizada para os mais diversos fins. Iremos

abordar alguns deles mais à frente.

A salicornia nasce e cresce naturalmente ao longo dos estuários e sapais (salinas) costeiras do Mediterrâneo].

Esta é uma planta suculenta adaptada a ambientes salinos (halófita) que se desenvolve maioritariamente em ambientes aquários com elevado teor de sal.]

Existem mais de de as mais comuns são:

Existem cerca de uma centena de espécies do género *Salicornia* L.], as mais comum encontram-se destacadas de seguida:

Salicornia virginica: é uma planta com flor e pode ser encontrada na região mediterrânica

Salicornia europea: resce em várias zonas de entre-marés salinas

Salicornia maritima: *Salicornia bigelovii*: *Salicornia perennis*: *Salicornia ramosissima*:

A evolução tecnológica é algo que sempre esteve presente na vida do ser humano desde os seus primórdios até aos dias atuais, sendo que se tem verificado um aumento desta relação com o humano e principalmente com o ritmo da própria evolução. As tecnologias, de uma maneira geral, são todas as invenções produzidas pelo homem, para aumentar a sua atividade no planeta e simplificar o modo de vida que quem o habita [1]. O conceito de “Internet das coisas” (do inglês “Internet of Things”, IoT) é fruto desta evolução tecnológica, já que permite a ligação dos mais diversos dispositivos eletrónicos à Internet.

1.1 Motivação

A motivação desta dissertação consiste na introdução da sensorização no cultivo da salicornia, a fim de, através da constante monitorização, oferecer dados suficientes para que os biólogos possam identificar as condições ótimas de salinidade para o crescimento da planta. Ao mesmo tempo, esta solução poderá servir de embrião para um futuro sistema que interaja com os mecanismos de rega afim de automatizar todo o processo de irrigação e cultivo.

O cenário de cultivo de salicornia representa outros desafios, como a inexistência de locais de fornecimento de energia ou infraestruturas de comunicação com fios. Estas características, aliadas ao facto de estes sensores servirem apenas para a aquisição e transmissão de informação pouco frequente e de baixo volume, tornam este cenário claramente num cenário que encaixa no universo da Internet Of Things (IoT).

Cliente que faz produção Horta dos Peixinhos empresa que produz salicornia

Horta dos Peixinhos

Aquicultura em águas salgadas e salobras Cultura e comercialização de Salicornia.

Departamento de biologia para disponibilização dos dados lidos pelo sensor para realização de estudos. para estudos..

1.2 Objetivos

Face aos objetivos supracitados, estabeleceram-se os seguintes objetivos específicos:

O trabalho prático destes tem como objetivo o desenvolvimento

2) Definição das funcionalidades da plataforma e dos alarmes; 3) Implementação de um protótipo de plataforma; 4) Desenvolvimento das Apps para comunicação com a plataforma e com o terreno;

- Criação de uma plataforma web que permita:

- Disponibilizar a leitura dos mais diversos sensores de sensores (temperatura, salinidade...)
- Permitir gerar alarmes de inundação, sendo este enviados via SMS ou email para o cliente.
- Atuar remotamente para drenagem de água em excesso existente nas leiras
- Sistema de transmissão de vídeo disparada por eventos gerados pelos sensores

- Criação de uma aplicação móvel que permita receber alarmismos de situações anómalias.

1.3 Organização do documento

A presente dissertação encontra-se dividida em 7 capítulos:

O presente documento encontra-se estruturado da seguinte forma, contendo sete capítulos: Introdução, O IoT no cultivo da Salicórnia, Estado da arte, Sistema de controlo e monitorização: arquitetura e modelação Implementação, Testes e resultados, e por fim, Conclusão e trabalho futuro.

No primeiro capítulo, Introdução,

Seguidamente, no capítulo dois, O IoT no cultivo da Salicórnia

De seguida, no capítulo três, O IoT no cultivo da Salicórnia

No quarto capítulo, Sistema de controlo e monitorização: arquitetura e modelação,

Posteriormente, no capítulo quinto, Implementação, é descrito...

No sexto capítulo, Testes e resultados, é...

Finalmente, no capítulo sétimo, Conclusão e trabalho futuro,

todo o trabalho realizado nesta dissertação é analisado e refletido por forma a concluir se os resultados obtidos satisfazem os objetivos previamente definidos. Na última parte é mencionado e discutido todo o trabalho futuro a realizar para que a solução proposta possa ser aperfeiçoada

O IoT no cultivo da Salicórnia

A *Salicornia ramosissima J. Woods (S. ramosissima)*[1] que impulsionará toda esta dissertação, é uma espécie do género *Salicornia L.*, pertencente à família das beterrabas denominada de *Chenopodiaceae*[2]. Neste capítulo será apresentada a planta, as suas principais características e respetivas 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 respetiva 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 sais[3], sendo uma das mais evoluídas da sua família. É uma planta anual de dimensão pequena, aparentemente sem folhas, ereta, os seus caules são carnudos e suculentos, simples e/ou extremamente ramificados, segmentados por articulações[4], geralmente com menos de 30 cm de altura[5]. 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[4] (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[6]. Em Portugal, onde é vulgarmente conhecida como erva-salada, sal verde e/ou espargos do mar[7], é encontrada frequentemente nas margens dos canais da Ria de Aveiro e Ria Formosa, no Algarve[7] sendo encontrada com menos frequência na região do Minho[4]. Na Inglaterra, a salicórnia é conhecida como *purple glasswort*, podendo este nome estar na origem desta pigmentação caraterística[8].

Esta planta é uma das menos estudadas pelos cientistas[6], sabendo-se apenas que pos-



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

sui um ciclo de vida anual bem definido, com gerações discretas e as suas sementes são hermafroditas[9]. 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[7] (C da figura 2.2) que pode ser utilizada para os mais diversos fins. A floração ocorre fundamentalmente no mês de outubro[6] (D da figura 2.2).

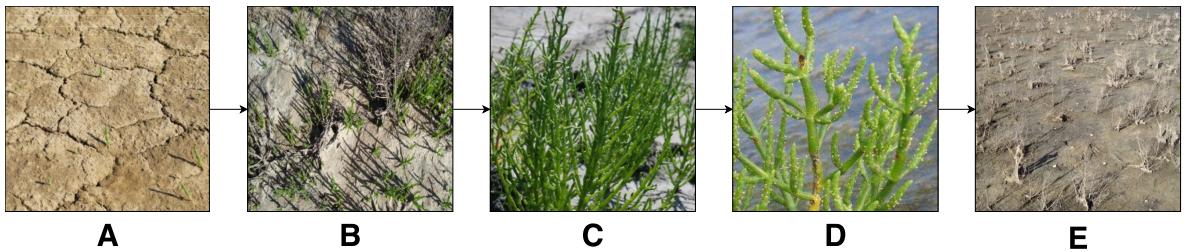


Figura 2.2: Esquema representativo do ciclo de vida da *Salicornia ramosissima*. A - semente incluída no sedimento; B - jovens plantas e plantas senescentes do ano anterior; C - planta no estado vegetativo, caule carnudo e articulado; D - planta no estado de floração; E - planta no estado senescente.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

2.2 Condições ideais de cultivo da Salicórnia

O crescimento da *Salicornia ramosissima* é influenciado por diversos fatores ambientais, 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.*[9] comprova que esta planta halófita apresenta um crescimento ideal em salinidades baixas ou moderadas. Este estudo permite considerar esta planta como uma halófita não obrigatória, já que o seu crescimento

ideal não acontece em condições de salinidade elevada. Embora o crescimento ideal ocorra a baixas salinidades, a *Salicornia* é capaz de tolerar níveis elevados de salinidade no meio de cultivo[10].

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 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[11], nomeadamente pelos níveis de minerais e vitaminas antioxidantes, como a vitamina C e o β -caroteno. A Salicornia é também uma fonte de proteínas e possui alto teor de ácidos gordos, destacando-se o ómega-3[12].

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[13].

Tendo em conta todas estas propriedades alimentares e medicinais da Salicornia, 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, nomeadamente ao conceito de IoT, tal como será descrito nas próximas secções deste capítulo.

2.4 Evolução tecnológica: o IoT

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[14]. 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, explicadas nas próximas secções.

- **Web 1.0 - passado:** esta primeira etapa foi inventada por Tim Berners Lee em 1989[15]. Nesta fase surgiram os principais conceitos que conhecemos da Internet atual: Localizador Uniforme de Recursos (do inglês Uniform Resource Locator (URL)), Linguagem de Marcação de Hipertexto (do inglês HyperText Markup Language (HTML))

e Protocolo de Transferência de Hipertexto (do inglês HTTP). Ainda nesta primeira fase, mas mais tarde, em 1998 foi criado por Larry Page e Sergey Brin o Google que criou simplicidade nas pesquisas na Web[16].

- **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 já transformou a Internet em algo sensorial, através da medição de diferentes características, como por exemplo a temperatura, a pressão, as vibrações, a iluminação, a humidade, o *stress*, entre outras. No futuro, ao desenvolvimento de aplicações revolucionárias com potencial para melhorar significativamente a forma como a sociedade vive, aprende, trabalha e se diverte.

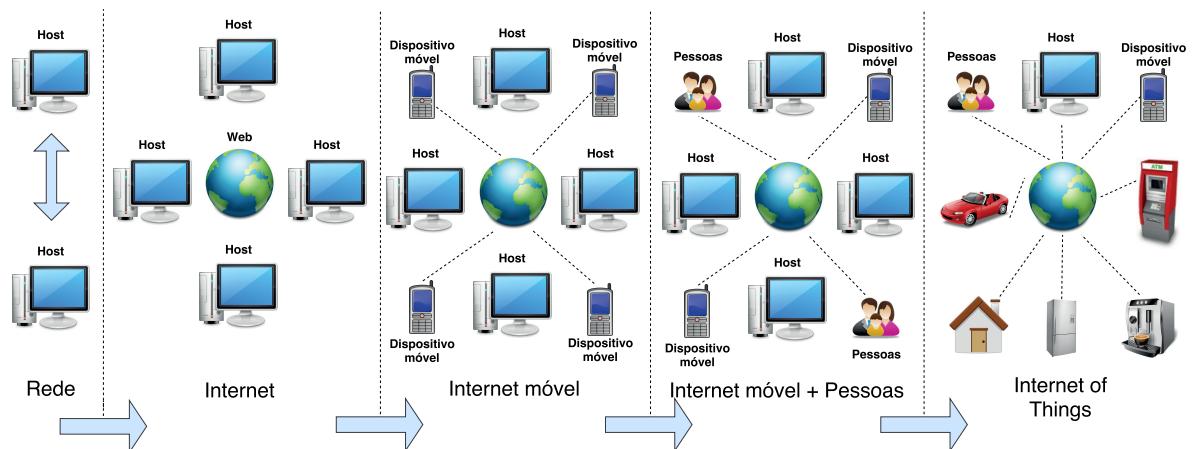


Figura 2.3: Evolução da internet em cinco fases (Adaptado de [17])

A figura 2.3 representa a evolução da Internet em cinco fases. Inicialmente surge a conexão entre dois computadores que permite a criação de uma rede, posteriormente nasce o conceito de World Wide Web (WWW) ligando um grande número de computadores entre si. Seguidamente, surgiu a Internet móvel que permitiu conectar dispositivos móveis à Internet, possibilitando a ligação da sociedade através das redes sociais. Finalmente, a internet está a evoluir para o IoT, permitindo ligar objetos do quotidiano ao sistema global de redes de computadores[17].

Uma das principais vantagens do IoT é a sua ligação evidente a todos os objetos, o que por si só é uma ideia avassaladora. O volume de dados gerado por este tipo de ligação pode ser interpretado pelo modelo Data-Information-Knowledge-Wisdom (DIKW)[18]. 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.

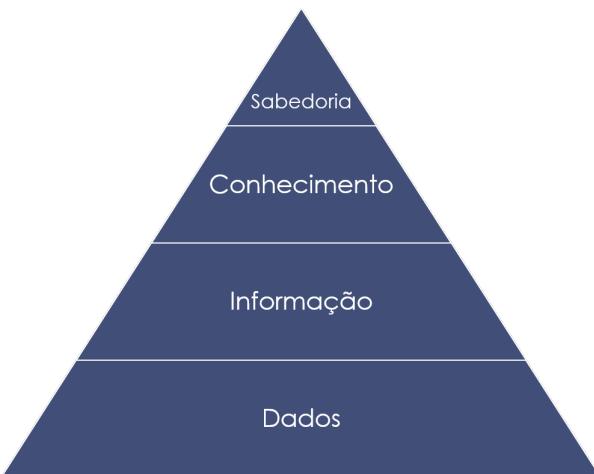


Figura 2.4: Pirâmide do conhecimento: modelo DIKW

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 Salicornia 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 salicornia carece de um controlo minucioso de certos parâmetros durante o seu cultivo, existe necessidade de criar um sistema que monitorização de forma a melhor as condições de produção desta espécie.

O conceito de IoT pode ser aplicado neste contexto, uma vez que possibilitará a interligação de equipamentos eletrónicos que melhorem a eficiência de produção da espécie através da colocação de sensores, atuadores e respetiva atuação remota.

Estado da arte

Neste capítulo, são apresentados os resultados da pesquisa bibliográfica sobre as ferramentas e funcionalidades que poderão estar presentes no sistema a desenvolver. Apresenta-se de forma geral todas as tecnologias possíveis de utilização e respetiva comparação.

3.1 Conceitos tecnológicos

Nesta secção são apresentados alguns conceitos tecnologias utilizados em toda a dissertação. Com o objetivo de não os repetir, estes serão descritos apenas aqui.

- HTML: consiste numa linguagem de marcação que permite estruturar uma página web.
- Cascading Style Sheets (CSS): permite personalizar os componentes existentes HTML, isto é, 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.
- 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.
- Web Service: é um tipo de API que trabalha com HTTP, ou Simple Mail Transfer Protocol (SMTP).

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 respectiva visualização, operações estatísticas e respetivos relatórios, automatização de funcionalidades entre outros. Seguidamente são apresentados três SGBD e a sua respetiva comparação.

3.2.1 MySQL

O MySQL é considerado o SGBD *open source* mais popular de todo o mundo, sendo líder em aplicações baseadas na web, incluindo o Facebook, Twitter, YouTube, Yahoo! e muitos outros. Está comprovado que possui um elevado desempenho, fiabilidade e facilidade de utilização[19].

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 ultima versão deste SGBD é o SQL Server 2016, estando também disponível para ambiente Linux[20].

3.2.3 PostgreSQL

O PostgreSQL é um sistema de gestão de base de dados 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 sources*[21].

3.2.4 Comparação e solução adotada

A tabela 3.1 encontra-se a classifica atribuída pela *Ranking DB-Engines* para os SGBD estudados de acordo com a sua popularidade, nos meses de Julho de 2017, Junho de 2017 e Julho de 2016[22]. O MySQL é considerado o mais popular dos estudados, posteriormente o SQL Server e por fim o PostgreSQL.

Uma vez que o cliente deste sistema pretende que sejam utilizadas tecnologias sem qualquer custo adicional, excluiu-se desde logo o SQL Server da Microsoft. Restam então duas tecnologias estudadas: PostgreSQL e MySQL. Relativamente aos requisitos não funcionais do sistema, estes serão abordados mais à frente nesta dissertação.

	Julho 2017	Junho 2017	Julho 2016
MySQL	2º lugar	2º lugar	2º lugar
SQL Server	3º lugar	3º lugar	3º lugar
PostgreSQL	4º lugar	4º lugar	5º lugar

Tabela 3.1: *Ranking BD-Engines* para popularidade dos SGBD estudados[22]

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-H¹, expondo que a performance do PostgreSQL é ligeiramente superior à do MySQL na maioria das queries[23]. A tabela 3.2 permite comparar algumas das características dos SGBD estudados.

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

Tabela 3.2: Comparação entre MySQL, SQL Server e PostgreSQL

3.3 Desenvolvimento web

No que toca 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

¹<http://www.tpc.org/tpch/>

em que o servidor servirá 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. A sigla ASP significa Active Server Pages. Uma aplicação desenvolvida nesta tecnologia corre num servidor de Internet da Microsoft, conhecido por Internet Information Services (IIS), podendo ser desenvolvido 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[24].

3.3.2 Flask

O Flask consiste numa micro-framework desenvolvida em Python baseado nas bibliotecas Web Server Gateway Interface (WSGI) Werkzeug e Jinja2². Esta framework disponibiliza um modelo simples para desenvolvimento web, um vez que permite economizar bastante tempo na sua conceção. De seguida, são apresentadas algumas das suas principais características[25].

- Possui um servidor para desenvolvimento e *debug*
- Suporta testes unitários nativamente
- Usa modelos da biblioteca Jinja2
- Suporta *cookies* seguros (sessões do lado do cliente)
- Compatível com a primeira versão do WSGI
- Baseado em unicode
- Extensa documentação

3.3.3 Django

A framework Django consiste numa ferramenta de desenvolvimento web de alto nível desenvolvida na linguagem Python, possuindo uma arquitetura MVC[26], 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

²<http://jinja.pocoo.org/>

informação e o *Controller* está encarregue de decidir que *views* escolher. Os criadores do Django consideram a sua arquitetura com sendo Model-Template-View (MTV)[27].

- *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 ao *Model* e encaminhando o pretendido para o *Template*.

De acordo com as descrições dos autores, esta ferramenta possui as seguintes vantagens[27]:

- Boa documentação;
- Facilidade e rapidez de desenvolvimento e *deployment*;
- Escalabilidade;
- Estabilidade.

3.3.4 Conclusões e solução adotada

Uma vez que o cliente do sistema não pretende que existam gasto adicionais, optou-se por utilizar ferramentas *open-source* que 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, indicando que permite alcançar um bom equilíbrio entre custo, características, rapidez e estabilidade[28].

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*. Pretende-se que este tipo de *software* permita ao seus utilizadores facilitar o desempenho de determinadas atividades à distância de um simples toque no seu dispositivo.

No que toca 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 específicas 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[29].

- **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[29].

- **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 numerosas *frameworks* para desenvolvimento multi-plataformas, entre as quais destaco três das 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 nativo), AngularJS (criação da *web app*) e Ionic Module e CLI (ferramenta e componentes disponibilizados pelo ionic)[30].
- **PhoneGap:** tal como o Ionic, consiste numa framework mobile que utilizada o Cordava da Apache para aceder aos recursos nativos do sistema. Vantagens desvantagens?? pq é melhor
- **Xamarin:** 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 Xamarin prende-se com a necessidade de desenvolver a UI para as plataformas necessárias.

3.4.3 Conclusões e solução adotada

Um vez que o cliente pretende uma solução mobile multiplataforma, 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 multiplataforma é o seu baixo desempenho na utilização de recursos do dispositivo. Uma 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 que os programadores pudessem expor os seus serviços para que possam ser utilizados por outras entidades. Neste sentido, foram criados métodos que permitem a comunicação entre diferentes sistemas, podendo estes estarem 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 Representational State Transfer (REST).

- SOAP: consiste num protocolo aplicado a ambientes distribuídos e descentralizado.
formato standard formato obrigado XML
- REST: ...
possibilidade de definir o formato das mensagens a trocar, sendo normalmente utilizados os formatos JSON ou XML

Dada as vantagens enumeradas relativamente à tecnologia REST, optou-se por a utilizar. 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 [31],

que pode ser usada juntamente com a framework de desenvolvimento de aplicações Web Django, que quando integrada no desenvolvimento de um determinado *backend* permite a implementação de serviços do tipo REST.

A API navegável Web é uma vitória usabilidade enorme para os desenvolvedores.

Políticas de autenticação , incluindo pacotes para OAuth1a e OAuth2 .

Serialização que suporta tanto ORM e não ORM fontes de dados.

Customizável todo o caminho - basta usar vistas regulares baseadas na função , se você não precisar dos mais poderosos recursos .

Extensa documentação , e grande apoio da comunidade .

Utilizado e confiável por empresas internacionalmente reconhecidas, incluindo Mozilla , Red Hat , Heroku , e Eventbrite .

3.5.2 Restlet

3.5.3 Flask-RESTful

é mais simples e minimalista, permitindo desenvolver serviços REST de uma forma rápida e leve, possuindo apenas as funcionalidades mais básicas necessárias para o desenvolvimento deste tipo de serviços.

3.5.4 Conclusões e solução adotada

Hypermedia As The Engine Of Application State (HATEOAS)

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

Tabela 3.3: Comparação entre frameworks REST estudadas.

Django REST Framework é aquela que apresenta um grau de maturidade maior, apresentando um grande conjunto de funcionalidades aos programadores, permitindo a utilização de boas práticas de desenvolvimento de serviços REST, aproveitando todas as vantagens da plataforma Django, nomeadamente o ORM definido pela própria plataforma, autenticação, gestão de base de dados, de rotas, entre outras.

com autenticação via token

app mobile microcontroladores - i controller modulers

documentação com swagger

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 protipagem 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[32].

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[32]. Nas figuras 3.1 e 3.2 apresenta-se o Arduino utilizado e a identificação dos diferentes pinos existentes, respectivamente. Adicionalmente, na tabela 3.4 encontram-se as principais características desta versão do Arduino.

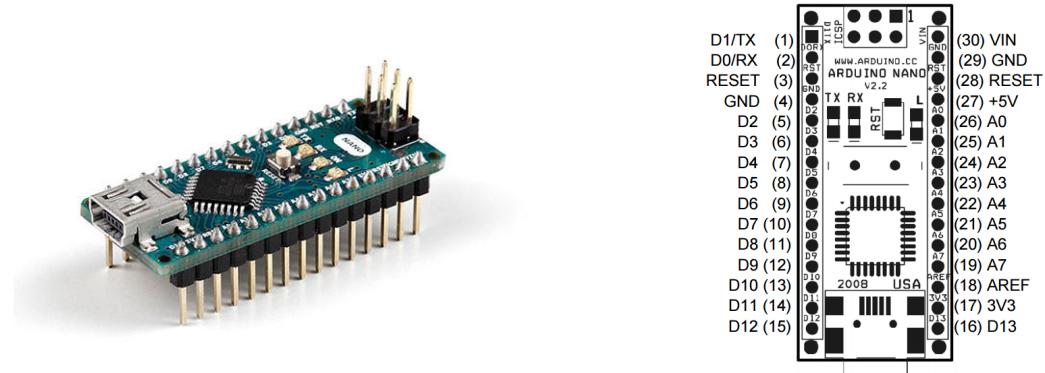


Figura 3.1: Arduino Nano

Figura 3.2: Identificação dos pinos num Arduino Nano (Retirado de [33])

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

Tabela 3.4: Características do Arduino Nano (Adaptado de [34])

3.6.2 Raspberry Pi 3

Um Raspberry Pi (figura 3.3) 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 Raspbian³ instalada[35]..

Na figura 3.4 encontram-se os principais componentes da versão utilizada e na tabela 3.5 é feita uma comparação entre a versão 2 e 3 do Raspberry Pi, permitindo concluir que a versão utilizada é mais poderosa.



Figura 3.3: Raspberry Pi 3

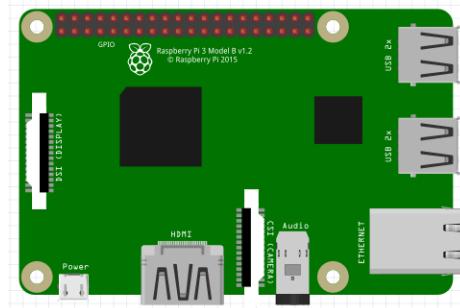


Figura 3.4: Principais componentes no Raspberry Pi 3

³Distribuição Linux oficial do Raspberry Pi: <https://www.raspberrypi.org/downloads/raspbian/>

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

Tabela 3.5: Comparação entre 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 Salicórnia. 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[36].
- **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 aquecimento[37].
- **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 3.5 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).

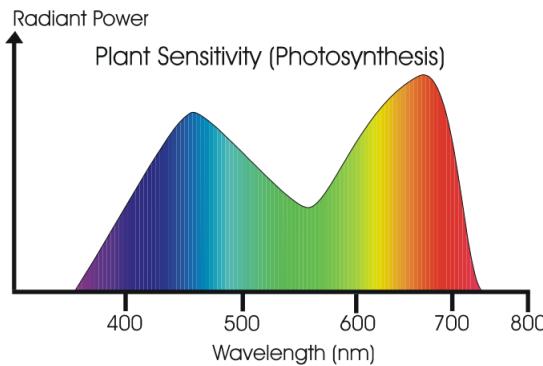


Figura 3.5: Sensibilidade luminosa nas plantas durante a fotossíntese (Retirado de [38])

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.
 - **Fototransistor:** 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. Trata-se de um sensor económico e de fácil utilização.

3.7.3 Sensor de salinidade

Atualmente ainda não há disponível nenhum sensor que permite 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[39] 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 utilizados em *Internet of Things* que permitem a troca de informação entre dispositivos. Serão abordados quatro tecnologias bastante populares comunicação: 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 taxa de dados baixa. 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[40].

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 fio 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)⁴, disponível mundialmente sem a necessidade de licenças[41][42].

⁴Faixa de frequência compartilhada por dispositivos, como forno micro-ondas e Wi-Fi

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[43][44]:

- 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[45]. 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.

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

Tabela 3.6: Comparação entre tecnologias de comunicação (Adaptado de [40])

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.

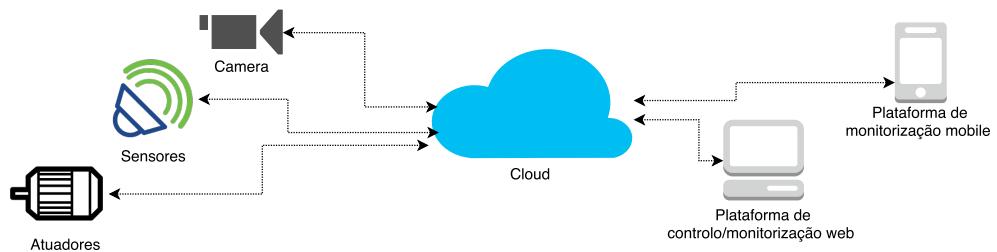


Figura 4.1: Ilustração dos principais componentes do sistema

Como vimos no capítulo 3, uma plantação de Salicórnia carece de um controlo relativamente

mente 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 Salicornia, 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.

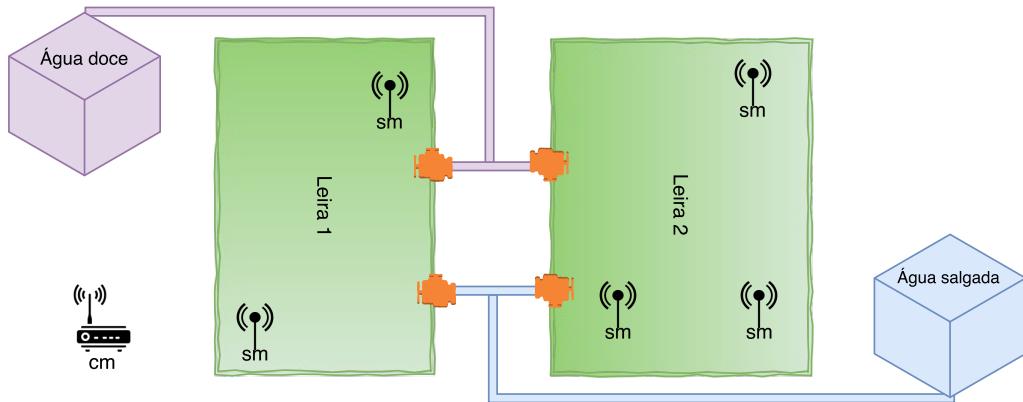


Figura 4.2: Ilustração da distribuição dos módulos em duas leiras

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 águas de modo a melhorar as condições de cultivo da Salicornia. Os módulos centrais têm acesso à camada protocolar TCP/IP (Internet) de modo a conseguirem a utilização da API REST via HTTP desenvolvida para o efeito.

No que diz respeito às plataformas de interação com o cliente, existe uma *dashboard* 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 Salicornia e receção de alertas quando estes ocorrem.

4.2 Componentes

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

- **Sensor Module:** consiste num módulo responsável pela aquisição de dados provenientes dos mais diversos tipos de sensores.
- **Controller Module:** consiste num módulo responsável pela receção dos dados/estados 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 dado adquiridos pelos sensores. Posteriormente, o *Controller Module* possui um determinado protocolo de comunicação (TPC/IP) que permite a utilização da API e respetivo envio ou atualização dos dados adquiridos pelo sistema.

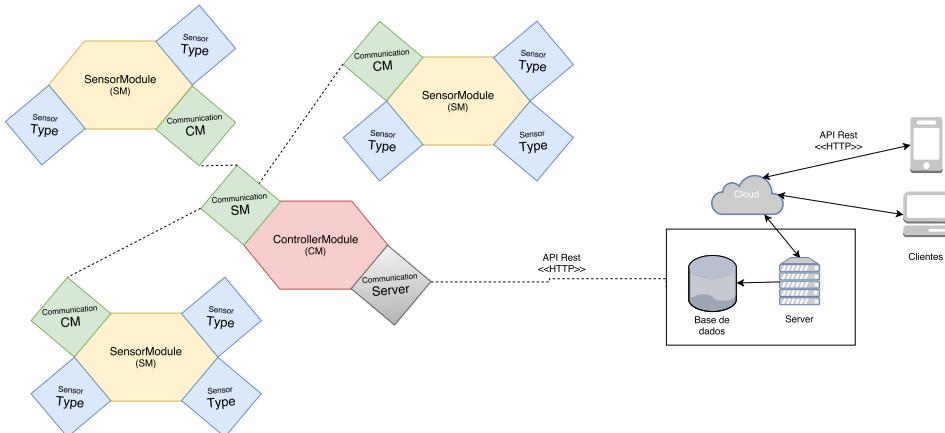


Figura 4.3: Esquema de componentes e respetiva comunicação entre três SM e um CM

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 micro-controlador 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.

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, uma 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 micro-controlador 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 REST criada para o efeito. Sendo que a tecnologia REST opera sob o protocolo de comunicação HTTP, este componente tem que necessariamente estar ligado à rede Internet via Transmission Control Protocol (TCP)/Internet Protocol (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 micro-controlador em caso de robo.

4.3 Análise de requisitos

Durante o desenvolvimento de *software* pressupõe-se que os seus intervenientes sigam determinadas metodologias para que o seu sistema possa revolucionar a vida de um grupo em específico ou até mesmo da sociedade.

O ciclo de vida do desenvolvimento de um software, também conhecido como Systems Development Life Cycle (SDLC), é composto genericamente por quatro fases principais: conceção, projeto, criação e implementação. Antes do SDLC, o processo de desenvolvimento do *software* foi tomado como atividade informal sem regras nem padrões formais. Este facto poderá originar 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 4.4 encontram-se as várias etapas consideradas por *Munish Kaur et al.*[46].

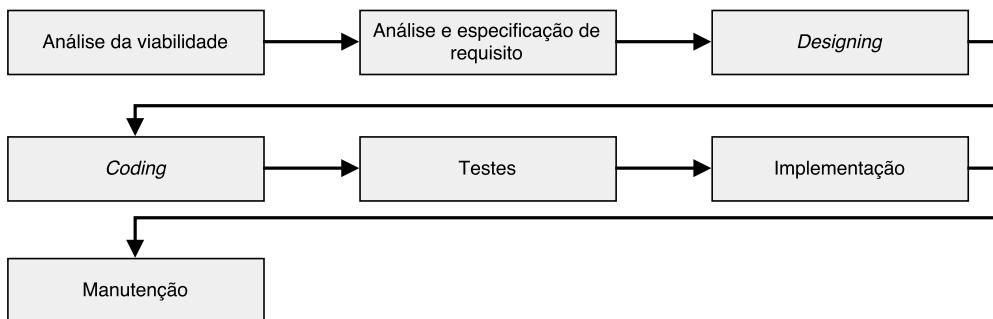


Figura 4.4: Fase de desenvolvimento de um software (Adaptado de [46])

- **Análise de 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 testes. São usadas várias técnicas de teste para 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 utilizado do sistema possa reportar erros ou problemas quando encontrados.
- **Manutenção:** o *software* poderá sofrer alterações para solucionar problemas que tenham ocorrido. Esta fase é responsável pela pós-implementação e manutenção do *software* para o seu bom funcionamento.

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 ou acredita precisar possibilitando a criação dos processos de negócio necessários ao sistema. Seguidamente apresentam-se os requisitos funcionais e não funcionais deste sistema.

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 pelo cliente 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.
- 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 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 e *standalone* 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 de utilizador

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.
- 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 pedidos pelo cliente, embora grande parte exista com devido

- 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 (*opensource*).
- Pretende-se que o sistema possa ser adaptado a qualquer outro contexto, não sendo especificamente restrito ao contexto da produção da Salicornia.

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.

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 4.5 e 4.6 encontram-se representados os esquemas dos casos de uso da *dashboard* (denominada por *saliDashboard*) e da aplicação *mobile* (denominada por *saliMobile*),

respectivamente. 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*.

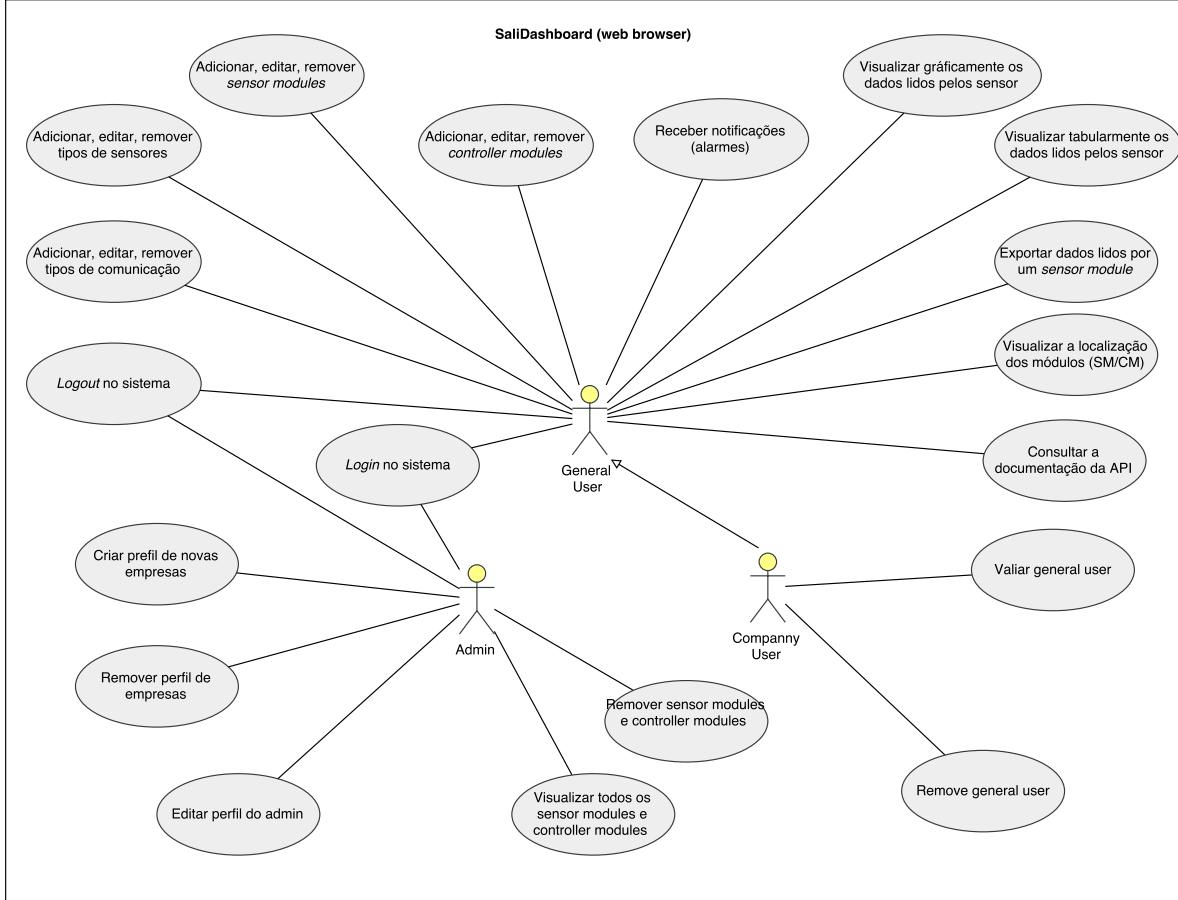


Figura 4.5: Casos de uso para a aplicação web (dashboard)

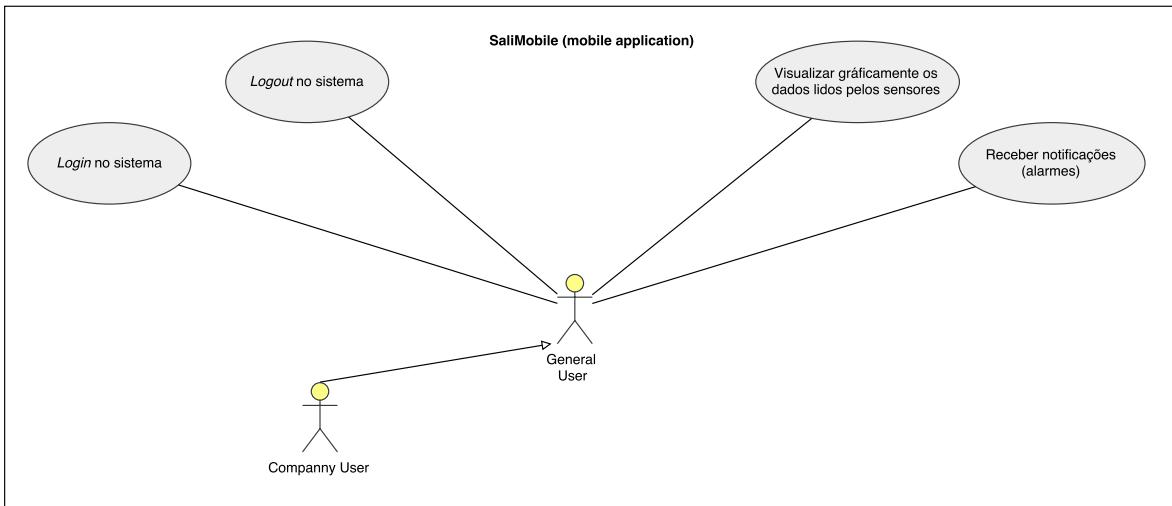


Figura 4.6: Casos de uso para a aplicação mobile

- **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 do general user. Após a validação das suas credenciais (username e password) o utilizador poderá aceder à pagina 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 tenha 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 envia 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 CM ou SM.
- **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 dados 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 SM.

- **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 micro-controlador com as seguintes características que podem ser adicionadas: determinado nome que o identifique, valor da memória RAM em MB, estado inicial (ativo ou desativo) e um modulo de comunicação que permita comunicar com o servidor web. Estas características são possíveis de alteração. Pretende-se que o controller module possa ser removido tendo SM a sí associado, sendo estes também removidos dos sistema.
- **Adicionar, editar, remover um sensor modules:** qualquer *general user* ou *company user* poderá adicionar, editar ou remover um sensor module existente no sistema. Pressupõe-se o utilizador possui um determinado micro-controlador com um ou mais sensores. Para adicionar o SM à plataforma o utilizador terá que atribuir um nome que o identifique, definir qual o seu estado inicial (ativo ou desativo), escolher que tipos de comunicação possam estar presentes e definir o intervalo de tempo para o qual pretende receber os dados lidos pelos sensores. Para além disso, permite associar ao SM os sensores presentes e definir o valor máximo e mínimos para o qual são gerados alarmes e respectivas mensagens informativas. Todos os dados são possíveis de edição.
- **Receber notificações (gerar alarmes):** sempre que um valor lido por um determinado sensor sai fora do limite imposto (valor máximo e 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:**
Este caso de uso encontra-se disponível para os dois tipos de plataformas.
 - Visualizar tabularmente os dados lidos pelos sensor
- **Exportar dados lidos por um sensor module:** qualquer *general user* ou *company user* poderá exportar os dados lido pelos sensores para um ficheiro do tipo CSV para uma data especificada.
- **Visualizar a localização dos módulos (SM/CM):** qualquer *general user* ou *company user* ao aceder à dashboard ser-lhe-à apresentado um mapa para cada CM com a sua localização e dos seus respetivos SM.

- **Consultar a documentação da API:** qualquer *general user* ou *company user* poderá aceder à dashboard para consultar a documentação 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 um novo general user é registado no sistema o comapny user é notificado via email. Posteriormente, caso o general user seja validado este também receberá um email de confirmação.
- **Remove general user:** qualquer company user tem a possibilidade de remove os general users que a si estão associados.
- **Remover sensor modules e controller modules :** o admin do sistema tem a possibilidade de remover os sensores 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 sensores 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, sendo para isso necessário
- **Remover um company user:** o admim tem a possibilidade de remover qualquer company user registrado no sistema.

4.4.3 Modelo de dados

Depois da análise de requisitos desenhou-se um modelo de dados que permitisse modelar todo o sistema descrito, obtendo-se assim o seguinte esquema relacional apresentado na figura 4.7.

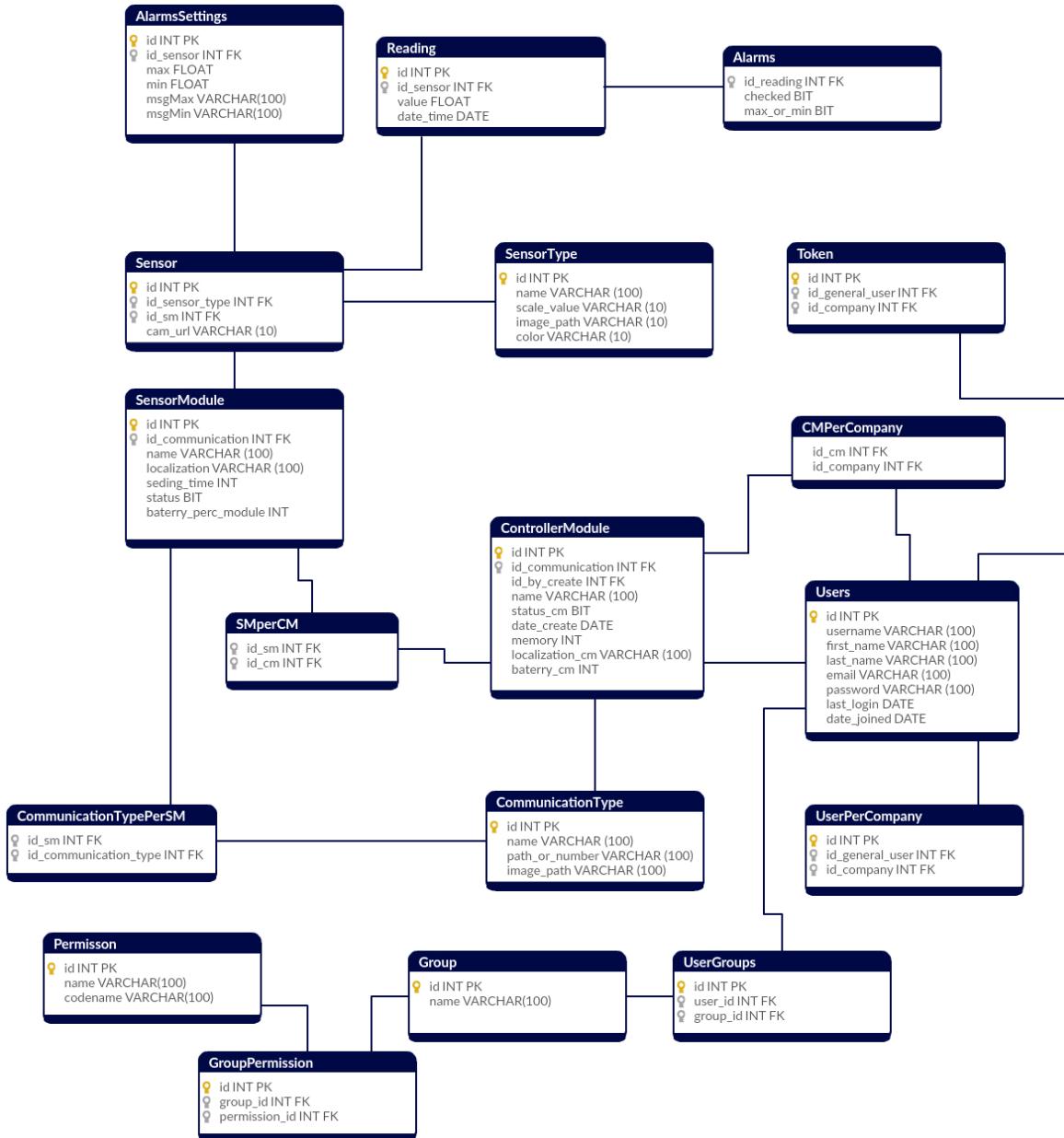


Figura 4.7: Esquema relacional da estrutura da base de dados

Nas tabelas 4.1 e 4.2 são descritos cada uma das entidades de dados existente neste sistema, evidenciando as chaves primárias e estrangeiras 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 token de autenticação do utilizador para a API
Group	id (auto-incrementado)	N/A	Possui todos os grupos existentes: general user, company user e admin
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 general user a um company user
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 controller module inseridos no sistema

Tabela 4.1: Especificação das tabelas existentes no sistema

Nome da tabela	Chave primária (PK)	Chave estrangeira (FK)	Descrição
CMPerCompany	id (auto-incrementado)	cm_id company_id	Associa todos os controller module a um determinado company user
SensorModule	id (auto-incrementado)	N/A	Identifica cada um dos sensor module inseridos no sistema
SMperCM	id (auto-incrementado)	cm_id sm_id	Associa os sensor modules a um determinado controller module
CommunicationTypePerSM	id (auto-incrementado)	communication_type_id sm_id	Permite associar a um sensor module um ou várias tipos de comunicação
SensorType	id (auto-incrementado)	N/A	Identifica cada um dos tipos de sensores inseridos no sistema
Sensor	id (auto-incrementado)	sensor_type_id sm_id	Permite identificar um determinado sensor
AlarmsSettings	id (auto-incrementado)	sensor_id	Permite guardar as configurações para um determinado sensor (max,min,...)
Reading	id (auto-incrementado)	sensor_id	Permite guardar as leituras de um determinado sensor
Alarms	id (auto-incrementado)	reading_id	Armazena os alarmes que são gerados

Tabela 4.2: Especificação das tabelas 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.

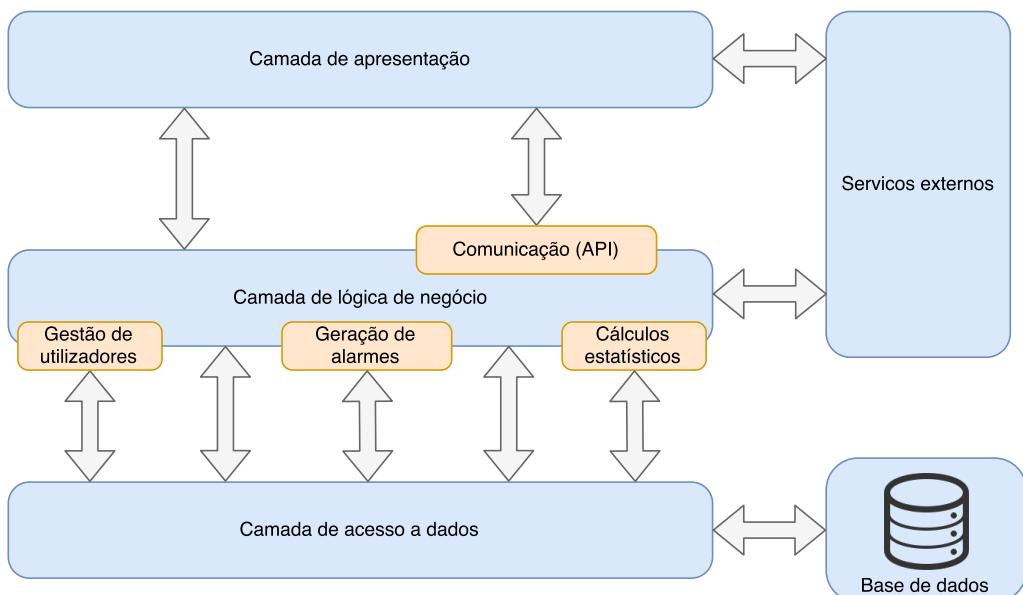


Figura 4.8: Arquitetura lógica

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 Javascript.

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 4.8, 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 de 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 4.9 representa genericamente os blocos principais do sistema proposto que seguidamente serão descritos.

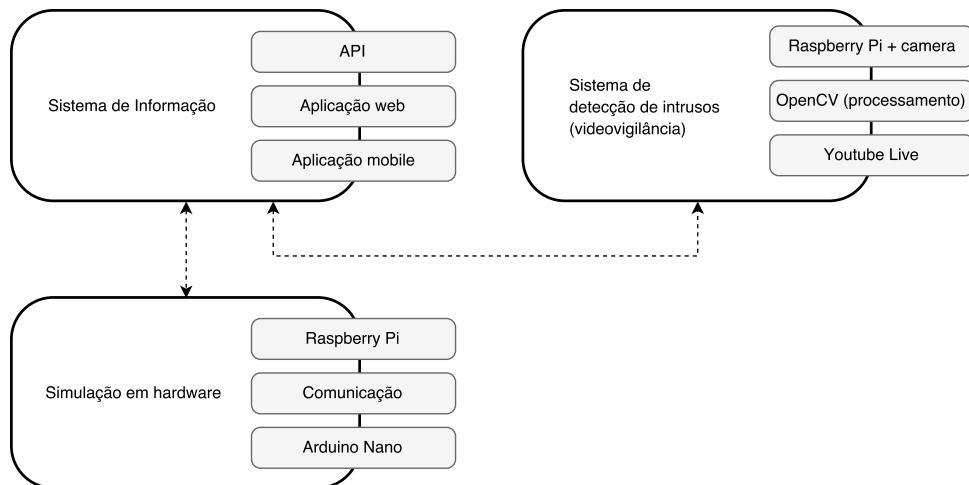


Figura 4.9: Arquitetura física (blocos)

4.6.1 Sistema de informação

Segundo *Laudon et al.*[47] 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[48]:

- **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 na 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 4.10 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 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 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.

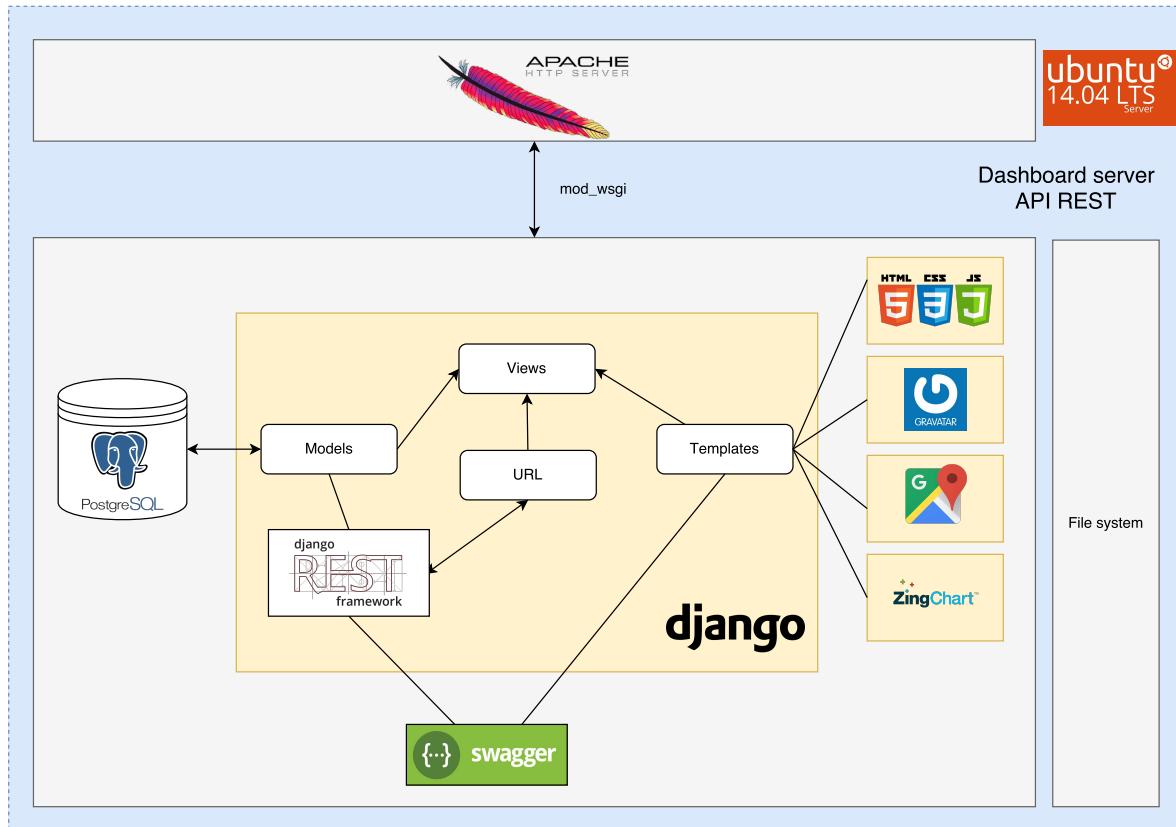


Figura 4.10: Arquitetura do sistema de informação (*dashboard*, base de dados e API)

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). 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 xxx. Como ferramenta gráfica para administração deste SGBD foi utilizado o PgAdmin III¹ na versão 1.22. Este software gráfico 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 *opensource* e encontra-se disponível para Windows e UNIX. Como vimos no capítulo 3, este SGBD permite uma grande facilidade de incorporação a um projeto em Django.

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

¹www.pgadmin.org/

- **HTML, CSS e JS:** o ponto de partida para a criação da interface web assentou no template denominado por AdminLTE² sendo este baseado em Bootstrap 3³. Neste template prevalecem as seguintes características: design responsivo, interface leve e atractiva, existência de múltiplos plugins, compatibilidade entre navegadores entre outros.
- **Gravatar:** serviço que disponibiliza um avatar que esteja associado a endereços de email registado. O Gravatar disponibiliza uma API que pode ser utilizada nas mais diversas linguagens de programação[49].
- **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 *opensource* disponibiliza mais de 35 tipos e módulos de gráficos.

API REST

Os métodos desta API permitiram execuções 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[50]. 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 efectuar operações de leitura
- **POST:** permite realizar operações de escrita, permitindo criar novos recursos ao sistema.
- **PUT:** permite criar ou atualizar um novo objecto ao sistema
- **DELETE:** permite apagar objecto ou recurso ao sistema.

²<https://adminlte.io/>

³<http://getbootstrap.com/>

Como vimos no capítulo 3, a escolha da *framework* para construção desta API REST recaiu sobre o *Django REST framework*. Esta framework 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*[51]. 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 4.11).

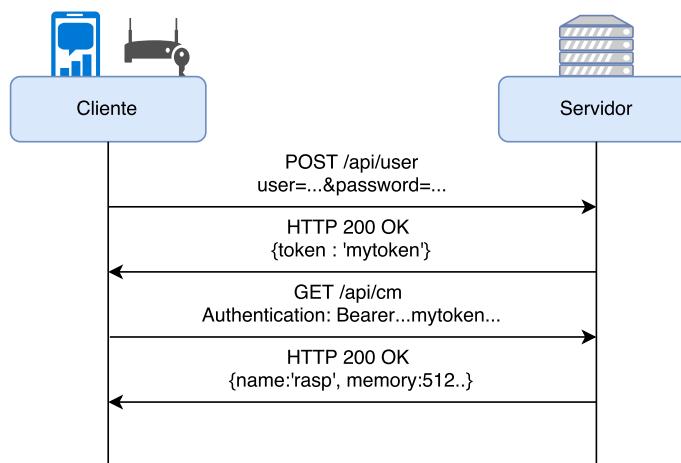


Figura 4.11: Processo de autenticação em HTTP através de token (Adaptado de [52])

Resumidamente e com o objetivo de explicar o esquema da figura 4.11, 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 respectivas funções (POST/GET/PUT) a implementar. Seguidamente será descrito como foi implementada a documentação desta API.

Endpoints da API REST	POST	GET	PUT	DELETE
/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[53], 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 *opensource*. 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 fornecida uma máquina com um 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* deste projeto optou-se pela utilização do servidor Apache juntamente com o pacote mod_wsgi⁴. 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 *opensource* mais utilizado em todo o mundo[54].

Aplicação mobile

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

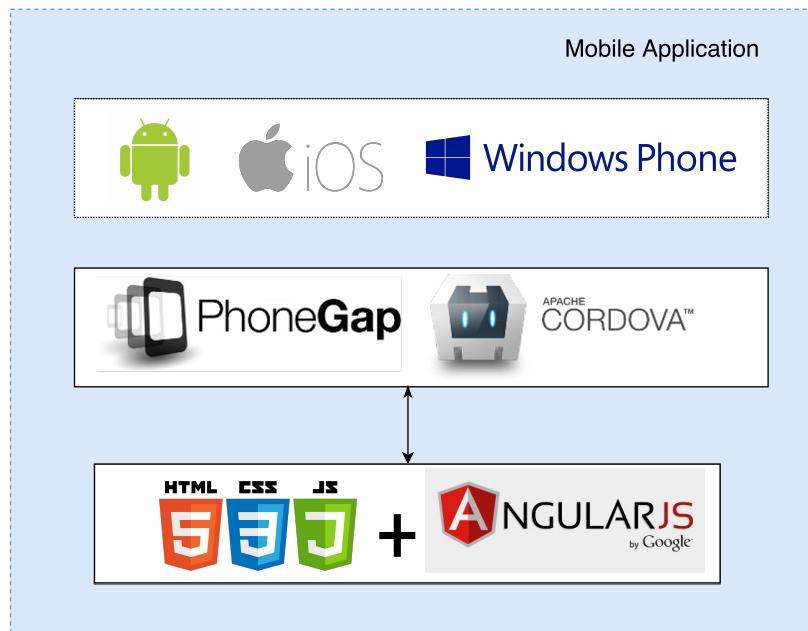


Figura 4.12: Arquitetura da aplicação mobile

Tal como descrito no capítulo 3, para o desenvolvimento do aplicativo 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 framework *opensource* mantida pela google desde 20XX

VAMOS USAR ANGULAR 2

Consumir a API REST

Consumir a API REST Consumir a API REST

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

4.6.2 Simulação em hardware

Após a desenvolvimento da API, simulou-se o sistema num contexto real. Para tal, pretendia-se encontrar *hardware* que encaixasse 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 5.7 encontra-se o esquema de ligação deste componente e na tabela 4.4 as suas propriedades principais[55].



Figura 4.13: Sensor TTC 104 NTC

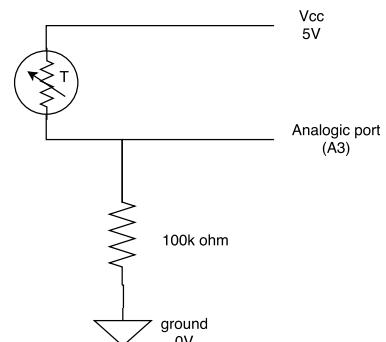


Figura 4.14: Esquema eletrotécnico da ligação do sensor de temperatura

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

Tabela 4.4: Características do sensor TTC 104 (Adaptado de [55])

Luminosidade

Para simular a luminosidade incidente foi utilizado um sensor do tipo foto-resistência, neste caso o GL5528 (figura 4.15). 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 4.16 encontra-se o esquema de ligação do componente e na tabela 4.5 são apresentadas as principais características do sensor utilizado.

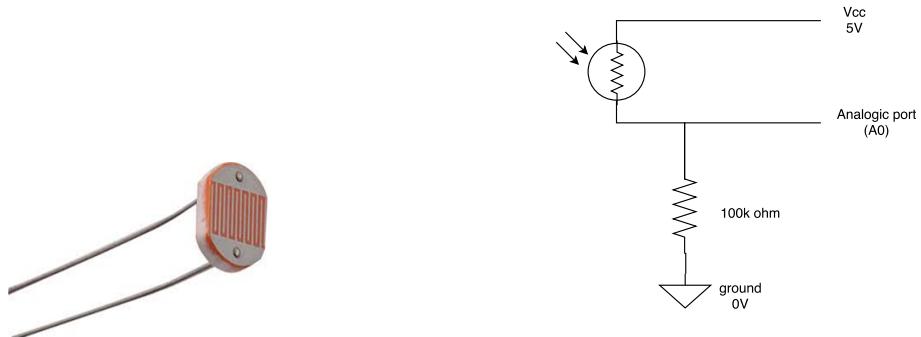


Figura 4.15: Sensor foto-resistência GL5528

Figura 4.16: Esquema eletrotécnico da ligação do sensor de luminosidade

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

Tabela 4.5: Características do sensor GL5528 (Adaptado de [56])

Sensor para verificação do estado do nível de água

Este sensor, denominado por *Water Level Switch Liquid Level Sensor Plastic Ball Float* (figura 4.17), 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 4.18 encontra-se o esquema da ligação deste sensor.



Figura 4.17: *Water Level Switch Liquid Level Sensor Plastic Ball Float*

Figura 4.18: Esquema eletrotécnico da ligação do sensor de nível líquido

Simulador de válvula para transferências de águas

Para a simulação de uma válvula que permitirá as transferências 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 4.20 encontra-se o esquema de ligação deste componente.



Figura 4.19: LED

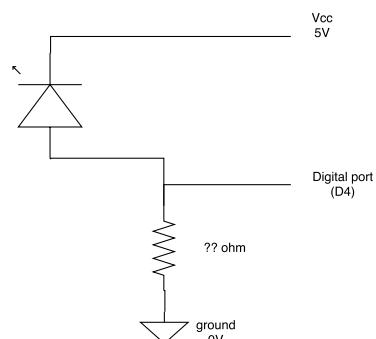


Figura 4.20: Esquema eletrotécnico da ligação do LED

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 conceito de IoT. No entanto, para o contexto deste trabalho, privilegia-se o alcance das tecnologias Bluetooth e Wifi 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 4.21 ilustra os tipos de comunicação envolvidos nesta simulação para cada um dos componentes.

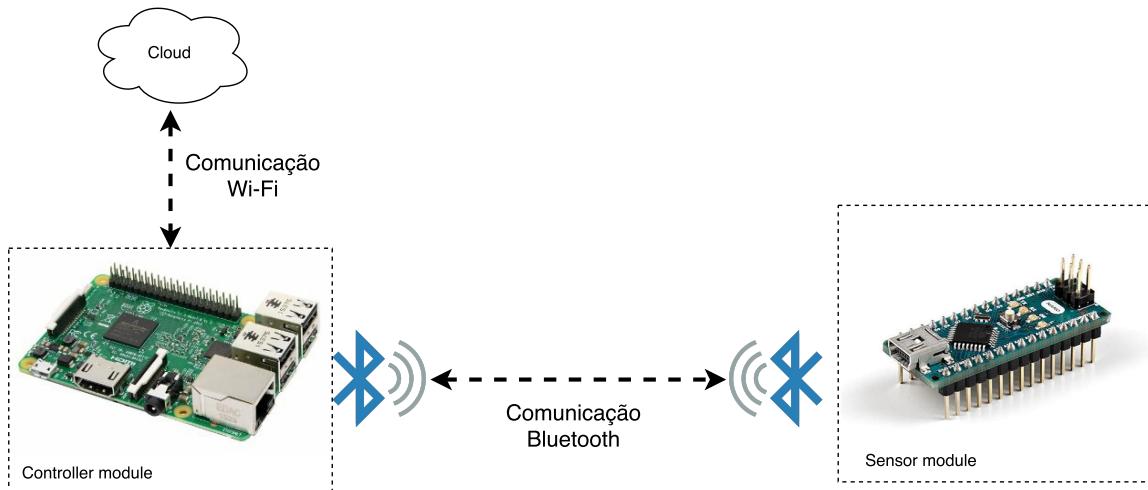


Figura 4.21: Comunicação entre componentes da simulação em *hardware*

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 no microcontrolador Arduino, possuindo um alcance máximo de aproximadamente 10 metros[57].

Na figura 6.17 encontra-se o esquema de ligação deste módulo e na tabela 4.6 são apresentadas as suas principais características.



Figura 4.22: Módulo Bluetooth HC-06

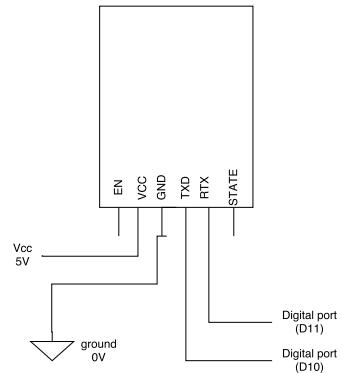


Figura 4.23: Esquema eletrotécnico da ligação do módulo Bluetooth

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

Tabela 4.6: Características do módulo bluetooth HC-06 (Adaptado de [57])

4.6.3 Sistema de vídeo-vigilância

No contexto desta dissertação existiu a necessidade de implementar um sistema de vídeo vigilância que permitisse detetar intrusos, maioritariamente pessoas ou animais de grande porte, que possam invadir as quintas onde se produz Salicornia. 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 detectar

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[58].

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 4.24) 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.

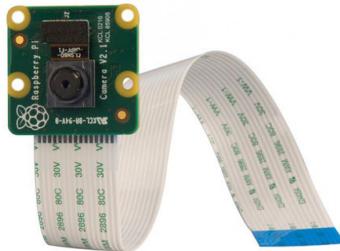


Figura 4.24: Raspberry Pi Camera Board V2 8MP 1080p

Sensor	8 megapixels Sony IMX219
Resolução de exibição	1080p, 720p e 640x480p para vídeo
Ligaçāo à placa	Cabo fita curta (branca)
Dimensões	25 x 23 x 9 mm
Peso	aproximadamente 3 g
Compatibilidades	última versão do Raspbian
Preço	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 X da secção XX), 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[59].

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 o utilizador escolher entre dois modos: baixa latência e alta latência, isto é, com boa ou má qualidade, respectivamente. O Youtube Live utiliza o protocolo Real-Time Messaging Protocol (RTMP), sendo este um servidor [60] ...

Numa primeira fase, pretendeu-se adquirir a imagem proveniente da câmara e proceder à stream de video para o Youtube Live, através da ferramenta FFmpeg, sem qualquer processamento.

Youtube Live

FFmpeg

Posteriormente, foram estudados alguns algoritmos de deteção de intrusos disponibilizados pelo OpenCV, que serão abordados no próximo capítulo.

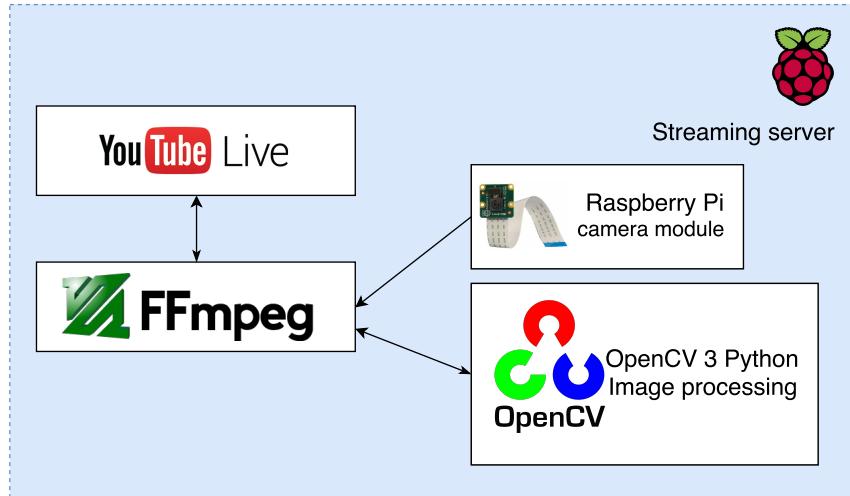


Figura 4.25: Arquitetura do sistema de video stream

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 sistema de deteção de intrusos, obtive o seguinte diagrama de componentes com as respetivas tecnologias utilizadas.

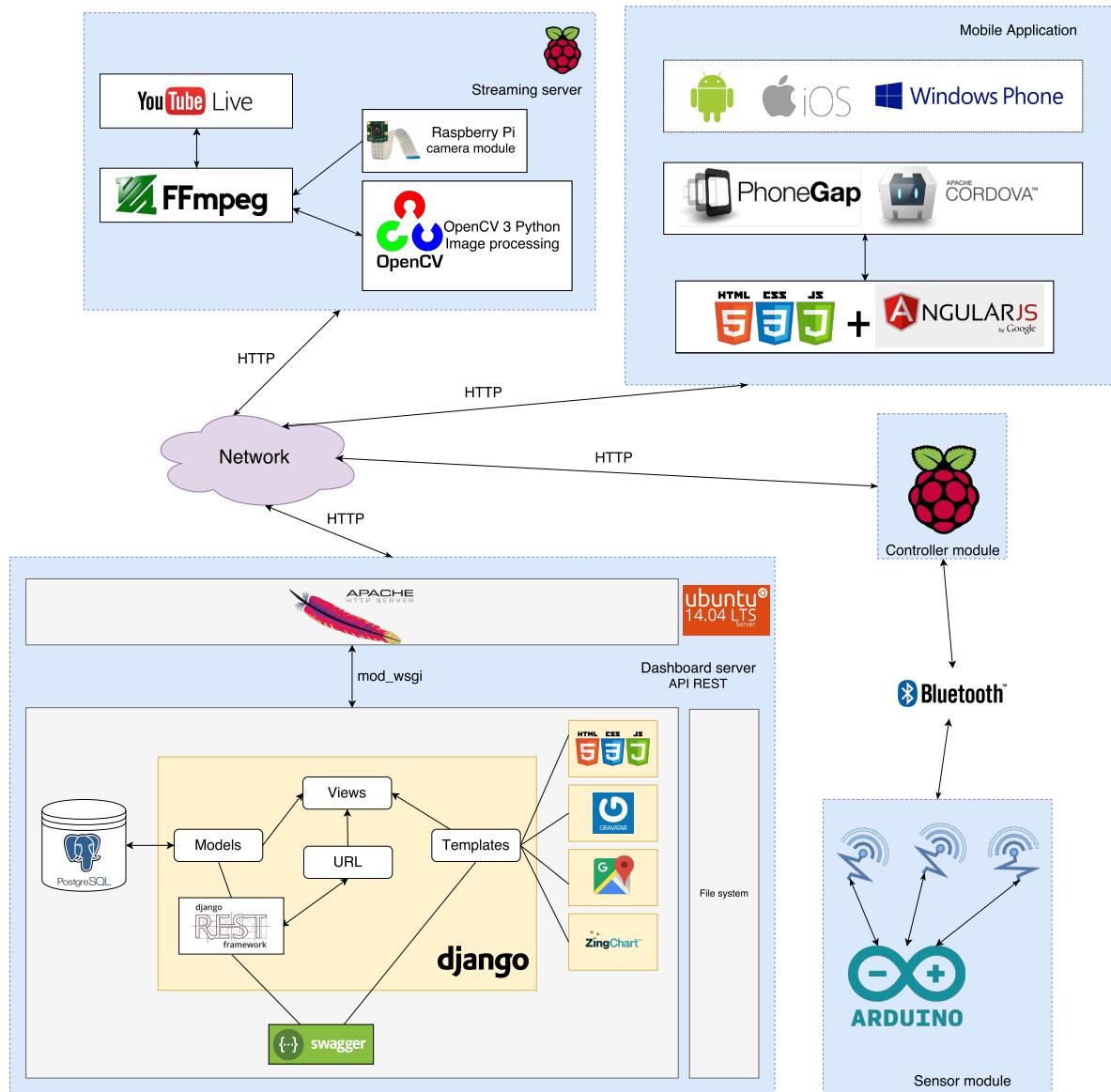


Figura 4.26: Diagrama final de componentes do sistema

4.8 Considerações finais

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 e também a aplicação mobile em Phonegap. Para além disso, é descrita a implementação da simulação em *hardware* para o cenário descrito no capítulo anterior bem como o sistema de deteção de intrusos.

Segundo o ciclo de vida de um software, estabelecido por *Munish Kaur et al.*[46] que analisámos na secção X.X, 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 psycopg2¹. 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 instalados 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 XX.

Posteriormente, procedeu-se à criação dos diferentes `Models` conforme a nomenclatura apresentada nas tabelas X e X da secção XX. O excerto de código seguinte pretende ilustrar a criação do `Model` associado à tabela que representa a estrutura *Sensor Module*, com o

¹<http://initd.org/psycopg/docs/>

respetivo identificador e atributos. Após a criação de cada `Models` procedeu-se à migração dos dados para o SGBD onde era 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 da aplicação 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 content area has three main sections:

- AUTENTICAÇÃO E AUTORIZAÇÃO**: Contains links for "Grupos", "Permissões", and "Utilizadores", each with "Adicionar" and "Modificar" buttons.
- AUTH TOKEN**: Contains a "Tokens" section with "Adicionar" and "Modificar" buttons.
- SALIAPP**: Contains sections for "Alarms settingss", "Alarms", "Cm per companies", and "Communication type per sms", each with "Adicionar" and "Modificar" buttons.

On the right side, there's a sidebar titled "Recent actions" which lists recent activities:

- 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 09:50:33+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 do 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, uma vez que, de modo a identificá-los e a definir a sua 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.

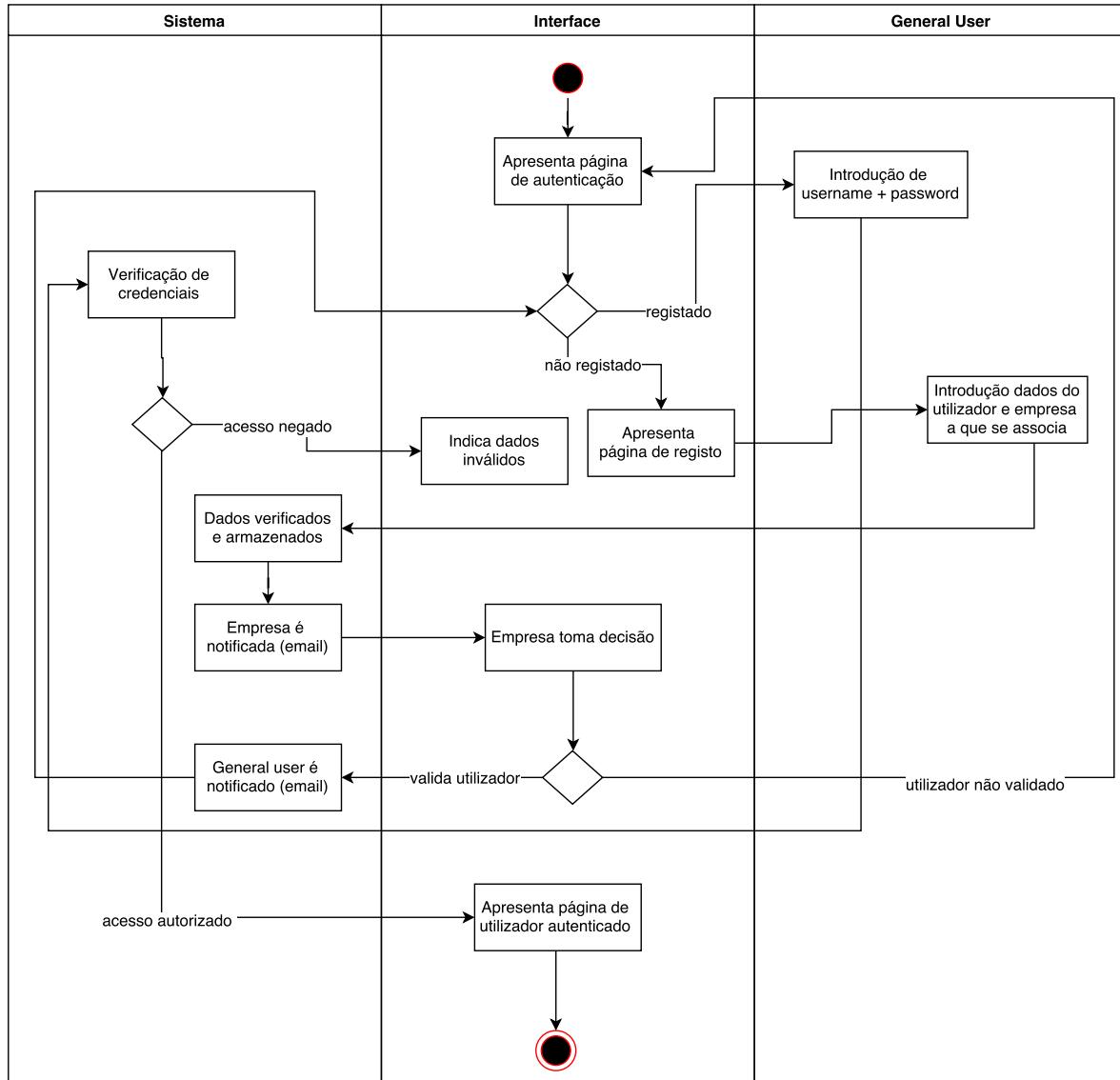


Figura 5.2: Diagrama de atividades do processo de registo e autenticação

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 da empresa. 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`² existente nativamente no Django pelo pacote `django.core.mail`, sendo construído através do módulo

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

`smtplib`³. 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 associada uma entrada na tabela `AlarmsSettings` que permite definir o valor máximo e mínimo para o qual são gerados alarmes bem como as mensagens associadas que permitem informar o 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 anexo X 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*.

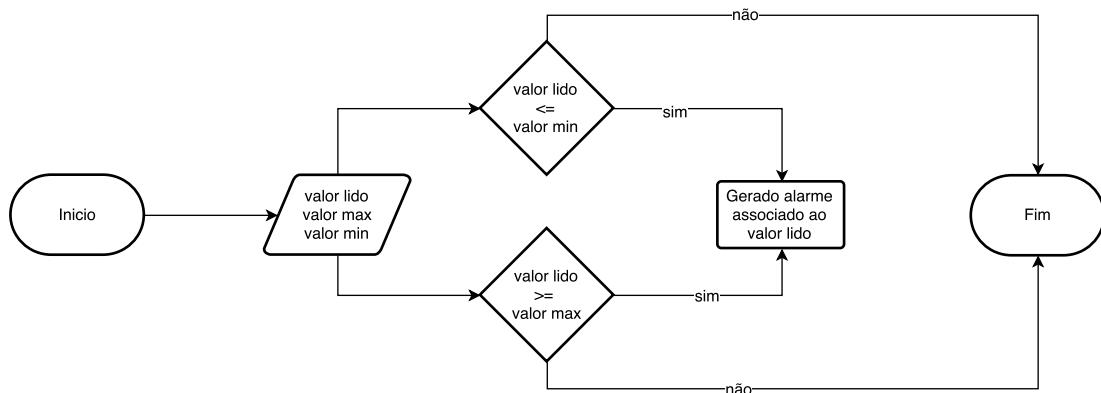


Figura 5.3: Diagrama de fluxo para geração de alarmes

Seguidamente são apresentadas algumas alternativas à implementação apresentada, pretendendo que estas sejam mais eficientes e igualmente competitivas às apresentadas pelas soluções apresentadas em XX.

- Análise do valor médio de cinco medições e respetivo desvio padrão
- Análise dos dados, também conhecida por data analist..... falta desenvolver data mining

³Módulo que define objetos para sessões SMTP <https://docs.python.org/3/library/smtplib.html#module-smtplib>

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 pretendido. Para além disso, é apresentada a mensagem pré-definida para que o utilizador possa tomar uma decisão.

5.1.3 Visualização dos dados e cálculos estatísticos

O cliente do sistema ao aceder à *dashboard* e pretender 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 sugestivos que permite realizar tarefas distintas.

- (editar): possibilita editar as características enumeradas anteriormente para cada *Sensor Module*.
- (remove): permite remover o *Sensor Module* do CM.
- (visualização gráfica): é possível visualizar graficamente os dados lidos pelos diferentes sensores existentes no *Sensor Module*.
- (visualização tabular): facilita em vista tabular uma visualização dos dados lidos pelos sensores bem como a respetiva exportação em 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 ultimo mês e selecionando a data de inicio 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. A análise estatística para o intervalo de data considerado permitirá ao cliente obter 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, esta página também possibilitará a ativação/desativação remota dos atuadores existentes no terreno.

Por fim, relativamente à visualização tabular os dados são apresentados em três campos distintas 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 serializáveis⁴ para cada `Model` existente bem como para os campos a considerar que possam ser usados na representação dos dados em formato JavaScript Object Notation (JSON). 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 `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}` com 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

```

⁴Consiste num processo de tradução de uma estrutura de dados ou de um objeto que pode ser armazenado (num arquivo ou buffer de memória) e reconstruído posteriormente

```

8     fields = ('id', 'name', 'id_communication', 'id_by_create', 'battery_cm',
9               'status_cm', 'date_create', 'memory', 'localization_cm')
10
11 #in apiViews.py
12 class ControllerModule_param(APIView):
13     def get_object(self, pk_or_name):
14         if pk_or_name.isdigit():
15             try:
16                 return ControllerModule.objects.get(pk=pk_or_name)
17             except ControllerModule.DoesNotExist:
18                 raise Http404
19         else:
20             try:
21                 return ControllerModule.objects.get(name__iexact=pk_or_name)
22             except ControllerModule.DoesNotExist:
23                 raise Http404
24
25     def get(self, request, pk_or_name, format=None):
26         cm = self.get_object(pk_or_name)
27         serializer = ControllerModuleSerializer(cm)
28         return Response(serializer.data)
29
30 #in url.py
31 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.authentication.TokenAuthentication`' ■■■

No anexo A 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 é 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.

The screenshot shows the Swagger UI for a REST API. At the top, there's a green header bar with the Swagger logo and links for 'Django Login' and 'Authorize'. Below the header, the title 'API documentation' is centered. The main content area is organized into sections based on API endpoints:

- alarms_reading**: Includes 'GET /api/alarms_reading/{id_reading}' and 'POST /api/alarms_reading/{id_reading}'.
- alarms_sensor**: Includes 'GET /api/alarms_sensor/{id_sensor}' and 'POST /api/alarms_sensor/{id_sensor}'.
- alarmssettings**: A collapsed section indicated by a minus sign.
- cm**: Includes 'GET /api/cm/' (highlighted in blue), 'POST /api/cm/' (highlighted in green), 'DELETE /api/cm/{pk_or_name}/' (highlighted in red), 'GET /api/cm/{pk_or_name}/' (highlighted in blue), and 'PUT /api/cm/{pk_or_name}/' (highlighted in orange).
- cperusers**: A collapsed section indicated by a minus sign.
- communication**: A collapsed section indicated by a minus sign.

Each endpoint row has 'Show/Hide', 'List Operations', and 'Expand Operations' buttons at the top right.

Figura 5.4: Documentação da API REST com a ferramenta Swagger

5.1.6 Aplicação web

O utilizador da dashboard após autenticar-se ser-lhe-á 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;
- 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 marcadores⁵ 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

⁵Um marcador identifica uma localização no mapa.

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

cada sensor distribuídos por cada *Sensor Module*. Adicionalmente, na página principal da plataforma são apresentados alguns valores estatísticos da mesma: numero de utilizadores registados, numero de *Sensor Modules*, numero de *Controller Modules* e numero 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 que permite alterar as suas informações básicas e a password mas também consultar o token de autenticação na API.
- Consultar, adicionar ou editar os detalhes dos *Controller Modules* e dos *Sensor Modules*
- Adicionar e respetiva consulta dos tipos de sensores e comunicação existentes
- Alterar o avatar existente através da ferramenta *Gravatar* disponível em <http://pt.gravatar.com/>

gravatar

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 mensagens⁶ do Django, também conhecida por mensagens de flash.

5.1.7 Deploy do projeto

Para implementar o projeto Django juntamente com um 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 pip⁷ 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
```

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

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

4. Configuração do virtualenv⁸, que consiste numa ferramenta para criar ambientes Python isolados. Por outro lado o virtualenvwrapper⁹ 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
```

```
git clone https://github.com/ruioliveira/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>
```

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

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

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/
    site-packages')
7 # Add the app's directory to the PYTHONPATH
8 sys.path.append('/var/www/ThesisSalicornia-web/saliDashboard')
9 sys.path.append('/var/www/ThesisSalicornia-web/saliDashboard/
    saliDashboard')
10 os.environ['DJANGO_SETTINGS_MODULE'] = 'saliDashboard.settings'
11 # Activate your virtual env
12 activate_env=os.path.expanduser("/var/www/.virtualenvs/exampleenv/bin/
    activate_this.py")
13 execfile(activate_env, dict(__file__=activate_env))
14 from django.core.wsgi import get_wsgi_application
15 application = get_wsgi_application()
```

9. Sincronizar da base de dados através do comando

```
python manage.py migrate
```

5.1.8 Aplicação mobile

5.1.9 Considerações finais

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 F. Para auxiliar o desenvolvimento de *software* foi utilizada a versão 1.8.1 do IDE do próprio Arduino¹⁰. Seguidamente apresenta-se a implementação necessária a nível de sensores e de comunicação.

Sensores

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 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)

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

Comunicação

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. Depois, 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*¹¹, 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*¹².
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

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

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

será ligado (válvula aberta), enquanto que se for enviado o dígito '0' o LED será desligado (válvula fechada).

- **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*.

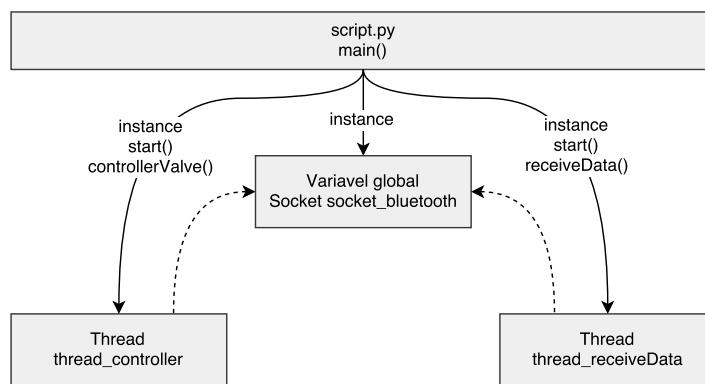


Figura 5.5: Threads e ddd... implementacao

5.2.3 Considerações finais

Simulou-se o sistema recorrendo a *hardware* disponível em laboratório, tanto a nível de microcontroladores, sensores ou módulos de comunicação. No entanto, não houve oportunidade 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.

Numa primeira fase, toda a comunicação foi realizada recorrendo ao método mais simples, por porta série. Todos os dados recebidos eram enviados para o servidor através da API criada para o efeito. Posteriormente, optou-se por isolar os módulos sem necessidade de cabos, utilizando para isso um módulo Bluetooth.

5.3 Sistema de vídeo-vigilância

Tal como referido anteriormente, para o
raspivid

Foi feito video stream com ffmpeg para o youtube

Para testar o algoritmo de opencv que a seguir se apresenta foi utilizado o

.....

No que toca ao desenvolvimento do sistema de deteção de intrusos, optou-se por utilizar o package `picamera`¹³ para aquisição da imagem proveniente do Raspberry Pi. 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. Neste contexto optou-se obviamente por utilizar a interface Python da biblioteca do 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`¹⁴ 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[61] 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[61] 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)[62].

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

¹⁴<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, entre os quais destaco:
 - `winStride`: define o ”tamanho do passo” na posição x e y da janela deslizante. 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, na qual a janela deslizante é ”acolchoado” antes da extração do recurso HOG. Os valores típicos para preenchimento 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, influenciando o número de níveis existentes. É um parâmetro opcional.

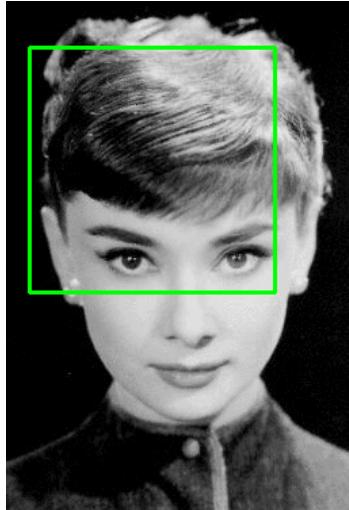


Figura 5.6: Sensor TTC 104 NTC

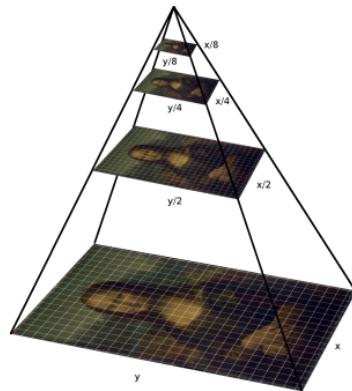


Figura 5.7: [63]

apenas foi

5.3.2 Testes

Pretende-se instalar uma câmara na quinta onde se produz salicornia, de modo a detectar grande parte da área existente, sobretudo onde existe os módulos sensoriais mas também algumas máquinas utilizadas no cultivo desta espécie. Para tal, tenciona-se colocar uma câmara, numa primeira fase uma *Raspberry Pi camera module*, mas posteriormente poderá ser substituída por uma câmara IP de maior qualidade. Esta câmara irá coloca-se no topo de um armazém existente virada para o centro da quinta a aproximadamente 3 metros de altura.

Com o objetivo de determinar quais os valores a atribuir aos parâmetros `winStride`, `padding` e `scale` foram realizados alguns testes com algumas imagens de um dataset onde seja possível detetar pessoas assumindo a mesma perspetiva, isto é, mesma altura e ângulo.

Foram considerados quatro frames de um dataset disponibilizado pela Universidade de Oxford utilizado para diferentes estudos a nível do desempenho em processamento de imagem envolvendo sistemas de vídeo vigilância, mais precisamente para detecção e tracking de pessoas[64]. O vídeo disponibilizado chama-se `TownCentreXVID.avi`.

Os resultados obtidos, isto é o tempo de processamento, numero de pessoas detetadas bem como as imagens consideradas encontram-se no anexo X. Foram considerados os seguintes valores possíveis para os parâmetros estudos:

- `winStride`: (2, 2), (4, 4), (8, 8)
- `padding`: (8,8), (16, 16), (24, 24)
- `scale`: 0.5, 1.0, 1.5

5.3.3 Implementação

Após concluir quais os melhores valores a utilizar como parâmetros no método `detectMultiScale()` procedeu-se à criação de uma aplicação em Flask que possibilita o acesso à imagem capturada pelo módulo câmara e realização do respetivo processamento. Pretende-se que seja instalado um servidor web NGINX com o objetivo de ser incorporado na dashboard.

Caso sejam detetadas pessoas são gerados alarmes através da API REST.

5.3.4 Considerações finais

incorporar o processamento
utilizando a API do youtube para realizar o stream
e respetiva geração de alarmes através da API.

6

Testes e resultados

Neste capítulo são apresentados alguns testes a nível de funcionalidades em alguns componentes bem como a apresentação de um cenário de teste com os respetivos resultados.

6.1 Testes funcionais

Nesta secção são apresentados alguns testes a nível de funcionalidades do sistema. Estes testes permitem averiguar se determinados blocos do sistema, 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 deles descrita de seguida.

- *Advanced REST client*¹: 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.
- CURL²: consiste numa 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.

¹<https://advancedrestclient.com/>

²<https://curl.haxx.se/>

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

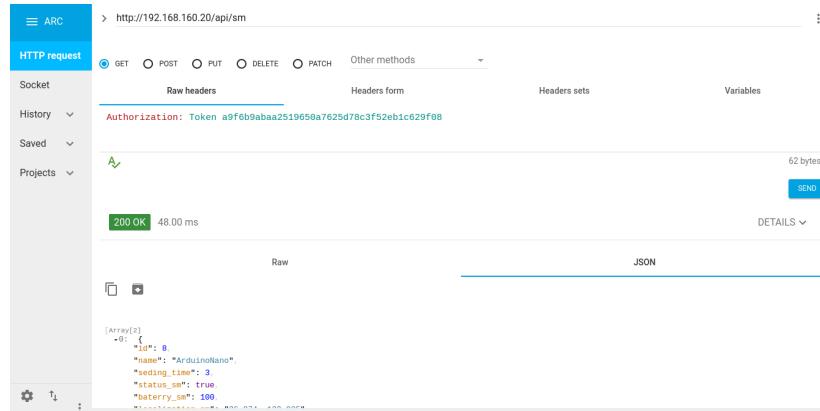


Figura 6.1: Documentação da API REST com a ferramenta Swagger

```

1 $ curl -X GET -H "Authorization: Token 79
e546740afe1aa4fb8d09a897146763e9f1b835" http://192.168.160.20/api/cm/
2 [{"id":4,"name":"Rasp3","id_communication":{"id":5,"name":"wireless","path_or_number":"","image_path":"earth-grid.png"},"id_by_create":{"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"}]
3
  
```

6.1.2 Comunicação via Bluetooth

Para testar o módulo Bluetooth HC-06 foi utilizada a aplicação *Bluetooth Terminal HC-05*. Através desta foi possível enviar os 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 Salicornia. 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 6.16 demonstra a manipulação do estado do 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 6.17 é possível observar o resultado após o envio do algarismo 1 com a ativação do LED (em E). Para além disso, é possível observar a ligação dos diferentes componentes na breadboard: o Arduino Nano (em A), módulo Bluetooth HC-06 (em B), sensor de luminosidade (em C) e o de temperatura (em D).



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

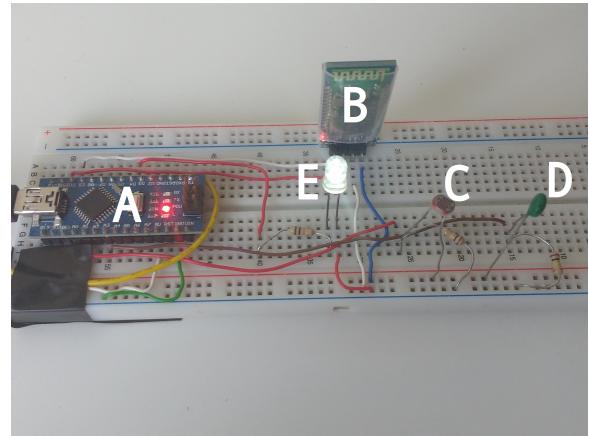


Figura 6.3: *Breadboard* com ligação dos diferentes componentes, destacando-se a ativação do LED

6.1.3 Deteção de intrusos

Foram realizados alguns testes aos algoritmo apresentado, obtendo-se os seguintes resultados



Figura 6.4: Imagem original (frame 1)



Figura 6.5: Resultado obtido (frame 1)

winstride (4, 4) padding (8, 8) scale 1.1

A figura seguinte são apresentados os
Dashboard home
Add novo cm e visualizacao dos existentes
add novo sm e visualizar os associados a esse CM
Visual graficamente e tabularmente os dados lidos... exportar por CSV;

6.2 Cenário de teste

1. Criação de um *Controller Module* com apenas um *Sensor Module*
2. O *Sensor Module* possui os seguintes sensores com as seguintes especificações:
 - (a) Sensor de temperatura:
 - (b) Sensor de luminosidade:
 - (c) Nível do tanque de água doce:
 - (d) Bomba para transferência de água doce:
3. Para o cenário apresentado, pretende-se que sejam enviados dados para o sistema durante 24 horas.
4. Os valores adquiridos pelos sensores são enviados para o sistema de 5 em 5 minutos nas primeiras 12 horas e de 10 em 10 minutos nas restantes.



Figura 6.6: Imagem original (frame 1)

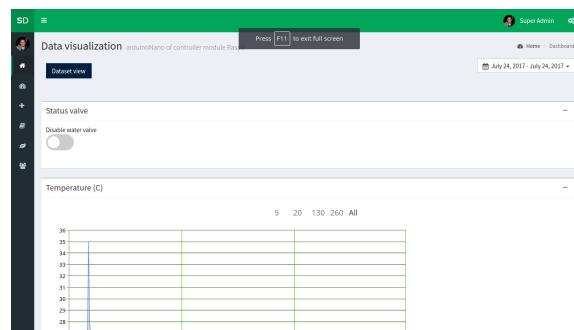


Figura 6.7: Resultado obtido (frame 1)



Figura 6.8: Imagem original (frame 1)

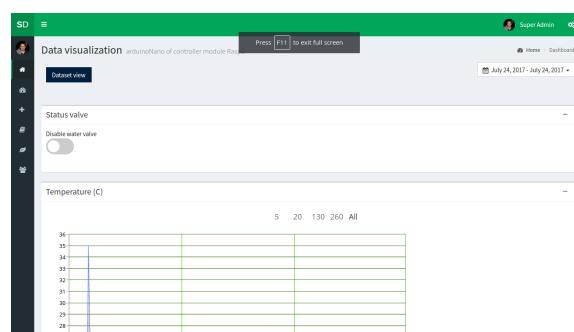


Figura 6.9: Resultado obtido (frame 1)



Figura 6.10: Imagem original (frame 1)

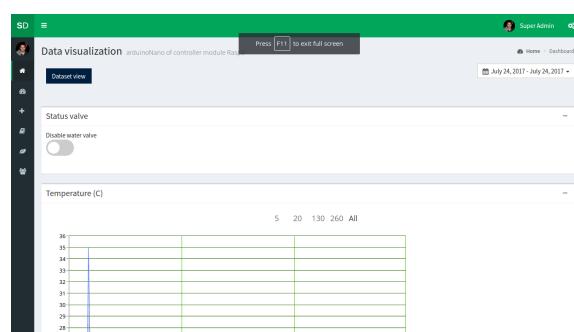


Figura 6.11: Resultado obtido (frame 1)



Figura 6.12: Imagem original (frame 1)

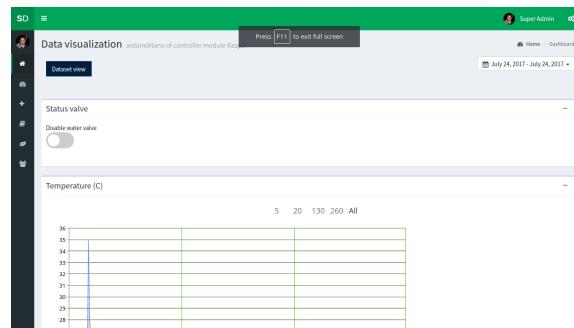


Figura 6.13: Resultado obtido (frame 1)



Figura 6.14: Imagem original (frame 1)

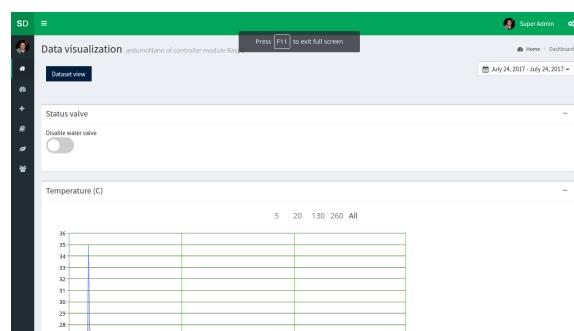


Figura 6.15: Resultado obtido (frame 1)



Figura 6.16: Imagem original (frame 1)

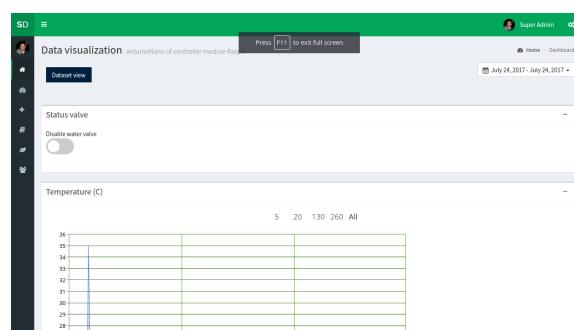


Figura 6.17: Resultado obtido (frame 1)

6.3 Interface mobile

receber notificações aspeto final da app; gráficos

6.4 Simulação em hardware

testar o envio de comandos para modulo bluetooth e verificar resultados enviados...
verificar ativação de uma valvula quando

6.5 Sistema de deteção de intrusos

- Exemplo em que os parametros testados funcionam bem e detectam pessoas numa imagem... desenhar rectangulos
- incorporação streaming na dashboard

6.6 Considerações finais

Conclusão e trabalho futuro

7.1 Conclusão

projeto sem financiamento por não foram utilizados sensores de salinidade:
justificar o falta fazer se é estável pode ser usado como ponto de partida para
O que podia ser feito: testes de usabilidade, medir tempos de resposta do web site;

7.2 Trabalho futuro

utilizar um sensor de salinidade para o cenário e implementar
geração de alarmes não ser com base se um determinado valor sai do range mas sim por
existir
fazer soldadura dos componentes
teste de usabilidade à interface gráfica e mobile
cuidado com os adjetivos na escrita, isto é suposto ser um documento crítico de engenheiraria, não é para vender um produto

Que novidades traz e que limitações
O que fazia de outra forma...?

7.3 Considerações finais

Bibliografia

- [1] João Silva, “Sal verde, National Geographic.” [Online]. Available: <https://nationalgeographic.sapo.pt/23-arquivo/as-nossas-historias/298-sal-verde> [Accessed: 2017-02-01]
- [2] S. Beer and O. Demina, “A new species of *Salicornia* (Chenopodiaceae) from European Russia,” pp. 253–257, 2005.
- [3] M. Ferri and N. Menezes, *Glossário Ilustrado de Botânica*, 1st ed., Livraria Nobel, Ed., Brasil, 1981.
- [4] 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>
- [5] 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.
- [6] 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.
- [7] 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]
- [8] 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.
- [9] 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.

- [10] 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>
- [11] 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.
- [12] 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>
- [13] Q. Z. Wang, X. F. Liu, Y. Shan, F. Q. Guan, Y. Chen, X. Y. Wang, M. Wang, and X. Feng, "Two new norriterpenoid 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>
- [14] D. Evans, "A Internet das Coisas Como a próxima evolução da Internet está mudando tudo," pp. 5–7, 2011.
- [15] 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]
- [16] 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]
- [17] T. Our, "Resume : Context Aware Computing for The Internet of Things : A Survey Article 2013," pp. 1–5, 2013.
- [18] 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>
- [19] MySQL, "MySQL :: About MySQL," 2011. [Online]. Available: <http://www.mysql.com/about/> [Accessed: 2017-07-13]

- [20] “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]
- [21] The PostgreSQL Global Development Group, “PostgreSQL: About,” 2012. [Online]. Available: <https://www.postgresql.org/about/> [Accessed: 2017-05-29]
- [22] 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]
- [23] 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>
- [24] Microsoft, “ASP.NET Overview,” p. 1, 2016. [Online]. Available: <https://msdn.microsoft.com/en-us/library/4w3ex9c2.aspx> [Accessed: 2017-07-17]
- [25] Flask, “Welcome — Flask (A Python Microframework),” 2014. [Online]. Available: <http://flask.pocoo.org/> [Accessed: 2017-07-17]
- [26] 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>
- [27] D. S. Foundation, “Django Documentation,” pp. 1–1172, 2012. [Online]. Available: <https://docs.djangoproject.com/en/1.3/> [Accessed: 2017-07-17]
- [28] A. Holovaty and J. Kaplan-Moss, *The Definitive Guide to Django: Web Development Done Right*, 2009.
- [29] Ibm, “Native, web or hybrid mobile-app development,” *Thought Leadership White Paper*, pp. 0–7, 2012.
- [30] Ionic, “Welcome to Ionic - Ionic Framework,” 2016. [Online]. Available: <http://ionicframework.com/docs/guide/preface.html> [Accessed: 2017-07-19]
- [31] A. Holmes, “Reviewing Django REST Framework,” 2014. [Online]. Available: <http://blog.isotoma.com/2014/03/reviewing-django-rest-framework/> [Accessed: 2017-07-20]
- [32] 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]

- [33] “Arduino Nano - User Manual.” [Online]. Available: <https://www.arduino.cc/en/uploads/Main/ArduinoNanoManual23.pdf>
- [34] J. Melorose, R. Perroy, and S. Careas, “Arduino Nano,” 2015. [Online]. Available: <http://www.farnell.com/datasheets/1682238.pdf> [Accessed: 2017-07-17]
- [35] Raspberry Pi Foundation, “Raspberry Pi Foundation - About Us,” 2012. [Online]. Available: <https://www.raspberrypi.org/about/> [Accessed: 2017-07-17]
- [36] REOTEMP Instrument Corporation, “Thermocouple-Thermocouples-What is a thermocouple-Types of thermocouples.” [Online]. Available: <http://www.thermocoupleinfo.com/index.htm> [Accessed: 2017-07-17]
- [37] “Temperature Sensors Watlow Educational Series.” [Online]. Available: <https://kontrolotomasyon.files.wordpress.com/2016/10/dt{ }temperaturesensors{ }thewatlow{ }25ekim2016.pdf>
- [38] 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>
- [39] “Salinity Sensor (Order Code SAL-BTA) Collecting Data with the Salinity Sensor.” [Online]. Available: <https://www.vernier.com/files/manuals/sal-bta.pdf>
- [40] 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>
- [41] 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>
- [42] Bluetooth(TM), “Bluetooth Specification,” *Specification of the Bluetooth System*, vol. 1, p. 1084, 2001. [Online]. Available: <http://www.bluetooth.com>
- [43] W. Paper, “IEEE 802.11g The New Mainstream Wireless LAN Standard,” 2005. [Online]. Available: <http://www.broadcom.com/products/index>
- [44] “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]
- [45] “Sigfox Technology Overview — Sigfox.” [Online]. Available: <https://www.sigfox.com/en/sigfox-iot-technology-overview> [Accessed: 2017-07-18]

- [46] 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>
- [47] Laudon, C. Kenneth, Laudon, and P. Jane, *Management Information Systems New Approaches to Organization & Technology*. Prentice Hall, 1998.
- [48] 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
- [49] “Gravatar - Globally Recognized Avatars.” [Online]. Available: <http://pt.gravatar.com/site/implementhttps://secure.gravatar.com/> [Accessed: 2017-06-10]
- [50] A. Rodriguez, “RESTful Web services: The basics,” no. February, pp. 1–11, 2015. [Online]. Available: <https://pdfs.semanticscholar.org/ca66/561d3602f65aef1301145e4e2689681b1967.pdf> <http://www.ibm.com/developerworks/webservices/library/ws-restful/>
- [51] T. Christie, “Django REST framework TokenAuthentication,” 2016. [Online]. Available: <http://www.djangoproject-rest-framework.org> [Accessed: 2017-07-03]
- [52] Ado Kukic, “Cookies vs Tokens: The Definitive Guide,” 2016. [Online]. Available: <https://auth0.com/blog/cookies-vs-tokens-definitive-guide> [Accessed: 2017-07-03]
- [53] SmartBear Software, “Swagger – The World’s Most Popular Framework for APIs.” 2017. [Online]. Available: <http://swagger.io/> [Accessed: 2017-06-07]
- [54] The Apache Software Foundation, “Foundation Project,” 2016. [Online]. Available: <https://www.apache.org/foundation/http://apache.org/foundation/> [Accessed: 2017-06-12]
- [55] “Datasheet, NTC Thermistor TTC05 Series, Disc Type for Temperature Sensing/Compensation.” [Online]. Available: <http://extra-parts.com/datasheets/TTC.pdf>
- [56] 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]
- [57] L. Guangzhou HC Information Technology Co ., “HC06 Datasheet,” no. 13, pp. 1–17, 2011.
- [58] Itseez, “About - OpenCV library.” [Online]. Available: <http://opencv.org/about.html> [Accessed: 2017-05-19]

- [59] “raspivid - Raspberry Pi Documentation.” [Online]. Available: <https://www.raspberrypi.org/documentation/usage/camera/raspicam/raspivid.md> [Accessed: 2017-07-25]
- [60] H. Parmar and E. M. Thornburgh, “Adobe’s Real Time Messaging Protocol Specification,” pp. 1–52, 2012. [Online]. Available: <http://www.adobe.com/devnet/rtmp.html>
- [61] 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>
- [62] “Feature Detection — OpenCV 2.4.13.2 documentation [ONLINE].” [Online]. Available: <http://docs.opencv.org/2.4/modules/gpu/doc/object{ }detection.html> [Accessed: 2017-07-10]
- [63] A. Rosebrock, “HOG detectMultiScale parameters explained,” 2015. [Online]. Available: <http://www.pyimagesearch.com/2015/11/16/hog-detectmultiscale-parameters-explained/> [Accessed: 2017-07-11]
- [64] “Coarse Gaze Estimation.” [Online]. Available: <http://www.robots.ox.ac.uk/ActiveVision/Research/Projects/2009bbenfold{ }headpose/project.html{ #}datasets> [Accessed: 2017-07-11]



A

Application Programming Interface

REST

- /api/user/
 - Métodos disponíveis:
 - Descrição:
- /api/user/
 - Métodos disponíveis:
 - Descrição:
- /api/user/{pk_or_username}/
 - Métodos disponíveis:
 - Descrição:
- /api/smpercm/
 - Métodos disponíveis:
 - Descrição:
- /api/smpercm/{pk_or_name_cm}
 - Métodos disponíveis:
 - Descrição:
- /api/sm/

- Métodos disponíveis:
 - Descrição:
- /api/sm/{pk_or_name}/
 - Métodos disponíveis:
 - Descrição:
- /api/sensortype/
 - Métodos disponíveis:
 - Descrição:
- /api/sensortype/{pk_or_name}
 - Métodos disponíveis:
 - Descrição:
- /api/sensorpersm/{id_sm_or_name_sm}
 - Métodos disponíveis:
 - Descrição:
- /api/sensor/
 - Métodos disponíveis:
 - Descrição:
- /api/sensor/{pk_or_sensor_type}
 - Métodos disponíveis:
 - Descrição:
- /api/reading/id_sensor/{date_start}/{date_end}
 - Métodos disponíveis:
 - Descrição:
- /api/communication/{pk_or_name}
 - Métodos disponíveis:
 - Descrição:
- /api/cm/

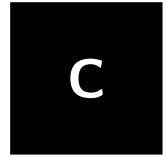
- Métodos disponíveis:
- Descrição:
 - /api/cm/{pk_or_name}
 - Métodos disponíveis:
 - Descrição:
 - /api/alarmssettings/{id_sensor}
 - Métodos disponíveis:
 - Descrição:
 - /api/alarms_sensor/{id_sensor}
 - Métodos disponíveis:
 - Descrição:
 - /api/alarms_reading/{id_reading}
 - Métodos disponíveis:
 - Descrição:

B

Mockups da aplicação mobile

Tabela B.1: Um nome qualquer

Posição	País	IDH
1	Noruega	.955
2	Austrália	.938
3	EUA	.937
4	Holanda	.921
5	Alemanha	.920



C

Trigger SQL

```
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.  
     id_sensor_id);  
8 varmin := (select min from saliapp_alarmssettings where id_sensor_id= new.  
     id_sensor_id);  
9  
10 IF (new.value >= varmax) THEN  
11 insert into saliapp_alarms (id_reading_id, checked, max_or_min) VALUES (new.id  
     , 'f' , 't');  
12 return new;  
13 END IF;  
14 IF (new.value <= varmin) THEN  
15 insert into saliapp_alarms (id_reading_id, checked, max_or_min) VALUES (new.id  
     , 'f' , 'f');  
16 return new;  
17 END IF;  
18  
19 RETURN NULL;  
20 END  
21 $alarm$  
22 LANGUAGE plpgsql;  
23  
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;
```

D

Interface gráfica

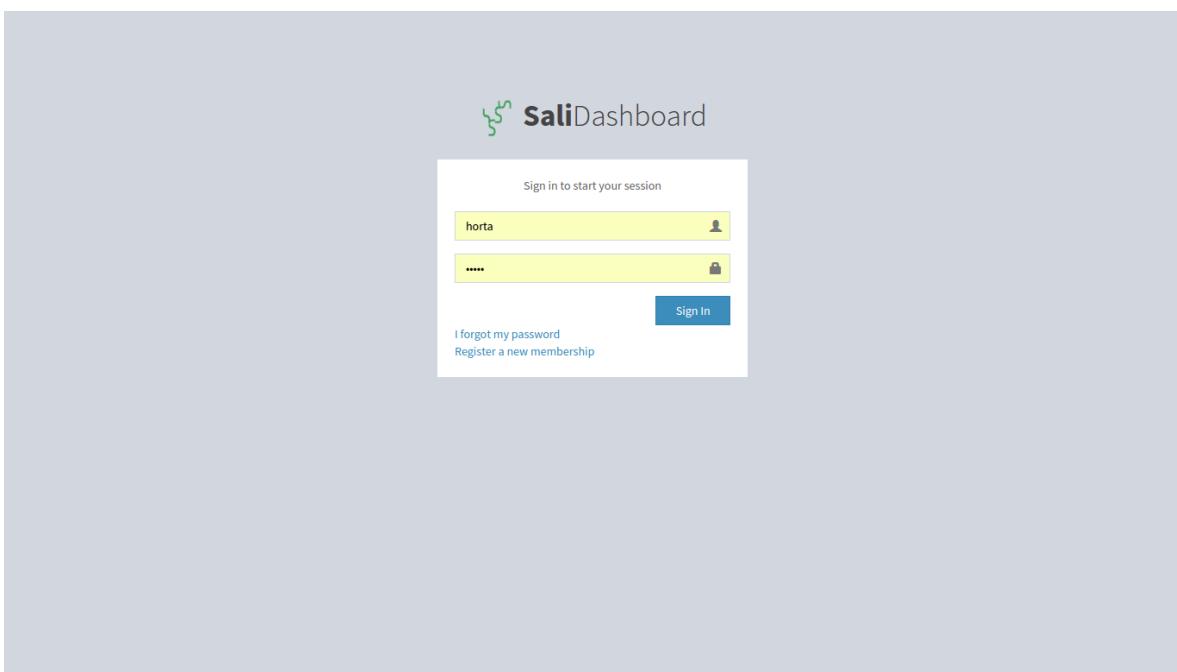


Figura D.1: Pirâmide do conhecimento: modelo DIKW

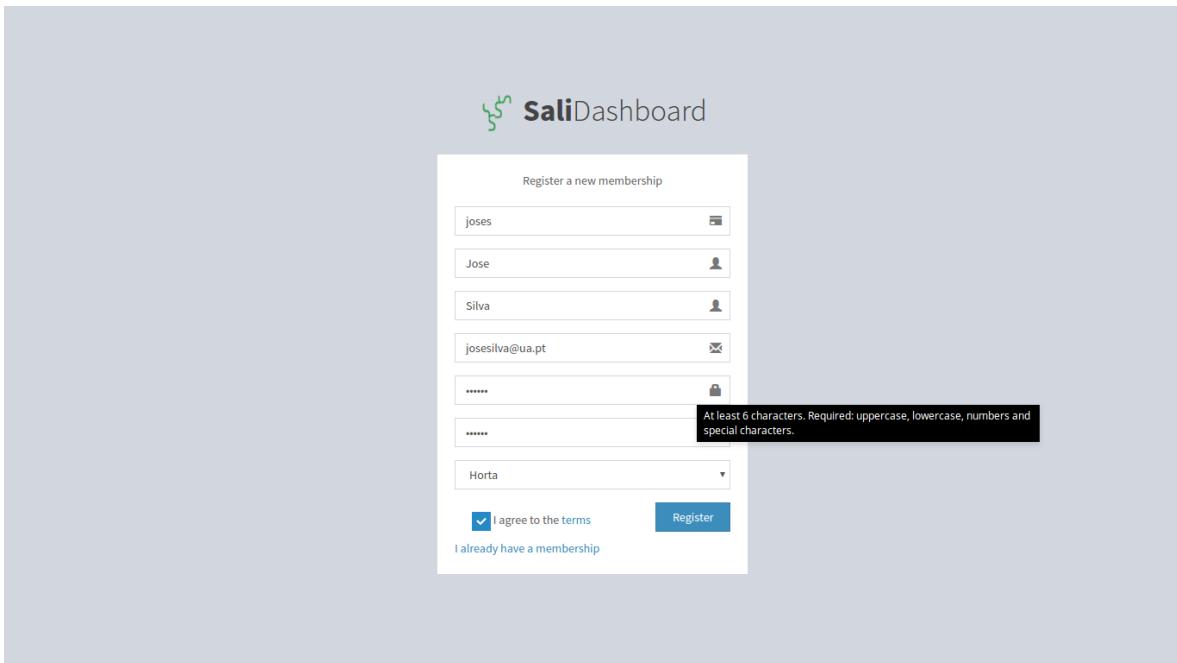


Figura D.2: Pirâmide do conhecimento: modelo DIKW

E

Descrição formal dos casos de uso gerais

F

Interligação de componentes

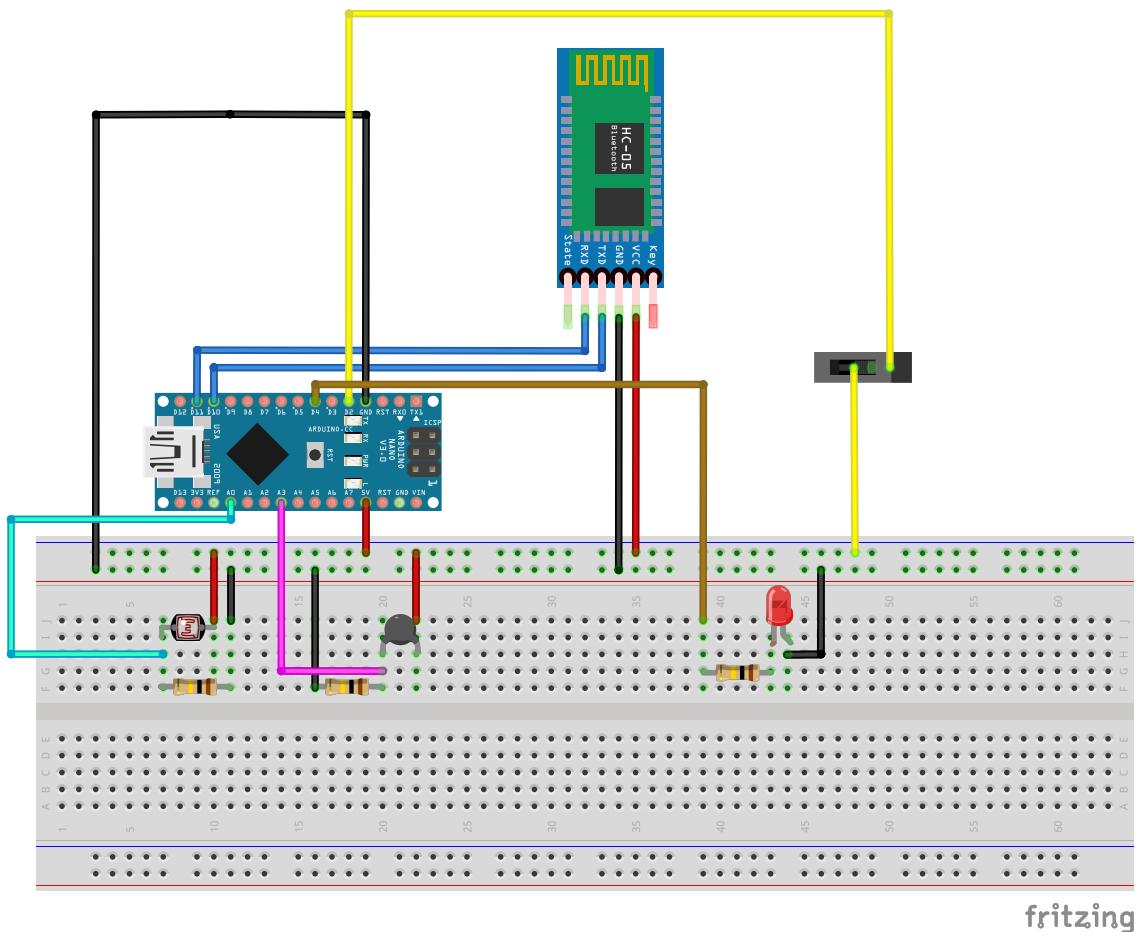


Figura F.1: Protótipo de montagem de componentes eletrotécnicos

