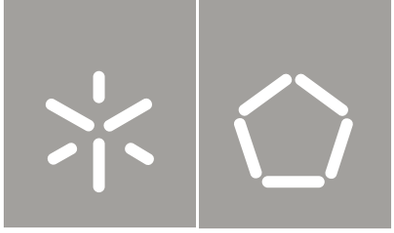
Universidade do Minho
Escola de Engenharia

Carlos Gabriel Ferreira Pinto

Gestão e Instalação de Redes de Sensores Sem Fios (Domótica)

outubro de 2013

Gestão e Instalação de Redes de Sensores Sem Fios (Domótica)
Carlos Gabriel Ferreira Pinto
UMinho | 2013



Universidade do Minho
Escola de Engenharia

Carlos Gabriel Ferreira Pinto

Gestão e Instalação de Redes de Sensores Sem Fios (Domótica)

Tese de Mestrado
Ciclo de Estudos Integrados Conducentes ao Grau de
Mestre em Engenharia Eletrónica Industrial e Computadores

Trabalho efetuado sob a orientação do
Professor Doutor Jorge Cabral

outubro de 2013

DECLARAÇÃO

Nome: Carlos Gabriel Ferreira Pinto

Correio electrónico: a52706@alunos.uminho.pt

Tel./Tlm.: 962650051

Número do Bilhete de Identidade13536872

Título da dissertação:

Gestão e Instalação de Redes de Sensores Sem Fios (Domótica)

Ano de conclusão: 2013

Orientador:

Professor Dr. Jorge Cabral

Designação do Mestrado

Ciclo de Estudos Integrados Conducentes ao Grau de Mestre em Engenharia Industrial Electrónica e Computadores

Guimarães, ___/___/_____

Assinatura: _____

Agradecimentos

Em primeiro lugar, o maior agradecimento é, sem dúvida, destinado à minha mãe Ana Maria Pinto, à minha irmã Ana Gabriela Pinto, à Maria Alves e Sara Pereira por todo o apoio e suporte durante a minha vida pessoal e académica.

Ao meu orientador, Professor Doutor Jorge Cabral pelo apoio e confiança depositada em mim para a realização deste projeto e pelas oportunidades proporcionadas.

Aos meus colegas de curso e amigos, principalmente os que partilhei casa Pedro, Hélder, Filipe, Vasco, Paulo, Arlindo pela amizade, confiança e acompanhamento do meu percurso académico.

Por fim, um grande agradecimento ao Paulo Gonçalves pela orientação prestada durante o desenvolvimento desta dissertação e também para o pessoal do *Embedded System Research Group*, por todos os conhecimentos disponibilizados e por terem proporcionado todas as condições para a realização desta dissertação e acima de tudo pelos momentos bem passados.

A todos, muito obrigado!

Resumo

A integração de redes de sensores sem fios com as plataformas móveis e a *Internet* traduz-se em inúmeros benefícios para os utilizadores. Assim, é possível o controlo e monitorização do sistema num único ponto, sendo fornecida a possibilidade de atuação remota, através das aplicações desenvolvidas para diversas plataformas. Esta integração foi provocada pela aumento das funcionalidades e do número de periféricos dos dispositivos móveis, entre os quais *Bluetooth*, *Wi-Fi*, permitindo uma maior interação com os sistemas dos ambientes de domótica.

A utilização das *Wireless Sensor Networks* fornece maior robustez aos sistemas e mais eficiência ao nível do controlo de ambientes. Contudo, veio dificultar o seu processo de instalação, mas também o processo de monitorização do correto funcionamento do sistema, provocado pelo grande número de componentes que compõe a rede.

Neste sentido, a presente dissertação visa o desenvolvimento e implementação de uma arquitetura que permite auxiliar o instalador no processo de instalação e monitorização de *WSNs*. O sistema utiliza as tecnologias *QR Code* e *NFC* através dos periféricos dos dispositivos móveis para identificação dos nós da rede, facilitando o seu processo de identificação e manutenção. A rede é parte integrante do projeto *Eco-smart Heat Pump* composto por um sistema de controlo de uma bomba de calor, apresentando-se como uma excelente solução para controlo de temperatura e humidade de ambientes habitacionais. Este projeto resulta de uma colaboração entre a empresa Pinto Brasil, Fábrica de Máquinas Industriais S.A. e a Universidade do Minho.

Assim, nesta dissertação, apresenta-se o desenvolvimento de um sistema para instalação e gestão de *WSNs* para plataformas móveis, *Web* e um conjunto de serviços Web utilizando computação em nuvem, assim como a arquitetura de solução de *hardware*.

Palavras-chave: Sistemas Embebidos, Cross-Platform, Smartphone, Graphical User Interface, Android, Cloud Computing, Mobile applications, QR Code, NFC.

Abstract

The integration of Wireless Sensor Networks with the mobile platforms and the internet results in several benefits for the users. So, it is possible to control and monitor the system from a single point, being provided the possibility of remote actuation through the aplications developed for several platforms. This integration was caused by the increase of the number of functionalities and peripherals in the mobile devices, Bluetooth and Wi-Fi, allowing a bigger interaction with the systems of domotic environments.

The use of *WSNs* gives a bigger reliability and efficiency at the level of control of environments. Althoug, it dificults the process of the system instalation, but also in the process of monitoring its correct behavior, caused by the large number of components that composes the network.

In this context, the present thesis proposes the development and implementation of an architecture that assists the installer in the process of instalation and monitorization of the *WSN*. So, the system uses *QR Code* and NFC tecnologies through mobile devices peripherals to identify the networks nodes, facilitating the process of identification and maintenance of the system. The *WSN* is a part of the research project *Eco-smart Heat Pump* that allows the control of an *Heat-Pump*, representing an excelent solution to control the temperature and humidity of living environments. This project is the result of a partnership between Minho University and the *SME* Pinto Brasil, Fábricas de Máquinas Industriais S.A., funded by QREN.

So, this thesis, describes the development of a GUI to install and manage a group WSNs of an heat pump for mobile and Web platforms and also a set of Web services using Cloud Computing, as well as the hardware's architecture solution.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

API	*Application Programming Interface*
CRUD	*Create, Read, Update, Delete*
CSS	*Cascading Style Sheets*
DAO	*Data Access Object*
DRY	*Don't Repeat Yourself*
DVM	*Dalvik Virtual Machine*
ESRG	*Embedded System Research Group*
GPS	*Global Positioning System*
GUI	*Graphical User Interface*
HTML	*Hyper Text Markup Language*
HTTP	*Hyper Text Transfer Protocol*
HVAC	*Heating, Ventilation, and Air Conditioning*
IDE	*Integrated Development Environment*
iOS	*iPhone Operating System*
ISO	*International Standards Organization*
JSON	*JavaScript Object Notation*
JVM	*Java Virtual Machine*
MOS	*Mobile Operating System*
MVC	*Model-View-Controller*
NFC	*Near Field Communication*

ORM	*Object-Relational Mapping*
PDA	*Personal digital assistant*
PDF	*Portable Document Format*
PHP	*Hypertext Preprocessor*
POO	*Programação Orientada a Objetos*
QRCode	*Quick Response Code*
QREN	*Quadro de Referência Estratégica Nacional*
RFID	*Radio Frequency Identification*
SDK	*Software Development Kit*
SGBD	*Sistema de Gestão de Base de Dados*
SMS	*Short Message Service*
SMTP	*Simple Mail Transport Protocol*
SQL	*Structured Query Language*
UML	*Unified Modeling Language*
URI	*Uniform Resource Identifier*
URL	*Uniform Resource Locator*
WSN	*Wireless Sensor Network*
WWW	*World Wide Web*
XML	*eXtensible Markup Language*

Capítulo 1

Introdução

Nos dias de hoje, as redes de sensores sem fios (WSNs) integradas em arquiteturas de sistemas de domótica são cada vez mais comuns. A evolução das tecnologias móveis, *smartphones* e *tablets*, permitiu a integração de diversos periféricos, tais como *Wi-fi*, *Bluetooth*, *Câmera*, nestes dispositivos. Com isto, estes dispositivos cada vez mais são parte integrante destas redes, permitindo o seu controlo e monitorização através de *GUIs* nativas, aplicações *Web* ou *Desktop* desenvolvidas para este intuito. As *GUIs* associadas a *WSNs* para controlo de ambientes de *habitações* permitem a atuação sobre diversos parâmetros dos sistemas, sendo que através de uma ligação à *internet* fornecem a possibilidade de atuação remota.

A complexidade das *WSNs* provoca diversos problemas no seu processo de instalação e manutenção devido à dificuldade na identificação dos componentes da rede, nomeadamente os nós. A utilização de *GUIs* e de serviços *Web* permitem auxiliar o instalador no âmbito da identificação do componente da rede que se encontra com problemas, uma vez que o seu estado é modificado através de serviços *Web* e monitorizado nas interfaces gráficas.

O projeto *Eco-smart Heat Pump* visa, entre outras funcionalidades, o desenvolvimento de uma rede de sensores sem fios para controlo de um bomba de calor, ar e água, para instalação em habitações, sendo uma excelente solução o controlo da temperatura do ambiente do domicilio e águas sanitárias. O projeto resulta de uma colaboração entre a Universidade do Minho e a empresa Pinto Brasil, Fábrica de Máquinas Industriais S.A.

Neste contexto, a presente dissertação é parte integrante da arquitetura do projeto *Eco-smart Heat Pump*, no sentido em que foi desenvolvido um sistema de gestão e definido o processo de instalação desta *WSN* utilizando um *Website*,

aplicação *Android* e um conjunto de serviços *Web*. Para isso, o processo de identificação e instalação dos componentes da rede é baseado na utilização das tecnologias *QR Code* e *NFC*.

1.1 Objetivos

A presente dissertação tem como objetivo o desenvolvimento de um sistema de monitorização e instalação da rede de sensores sem fios do projeto *Eco-Smart Heat Pump*, sendo possível dividir os objetivos por quatro componentes. Assim, o primeiro visa a implementação de uma base de dados do tipo *MySQL*. O segundo componente consiste no desenvolvimento de um conjunto de serviços *Web*, ou seja, um *middleware*, de forma a fornecer uma interface de comunicação com a base de dados por parte da aplicação *Android* e o nó coordenador da própria *WSN*. O terceiro é composto pela implementação de um *Website* na linguagem *PHP*, para monitorização e gestão da *WSN*. Por sua vez, no quarto e último objetivo é proposto o desenvolvimento de uma aplicação móvel para a plataforma *Android* que possua funcionalidades de leitura de *QR Code* e *NFC* para identificação dos componentes da rede, mas que também seja possível consultar e alterar o estados dos nós e restantes componentes do sistema.

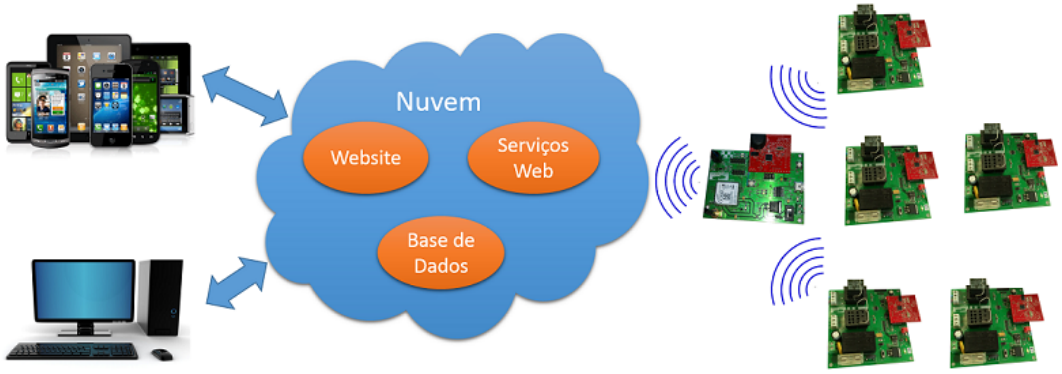


Figura 1.1: Arquitetura do Sistema

1.2 Estrutura da dissertação

No capítulo 1 é apresentada uma breve contextualização do trabalho, sendo de seguida especificados os objetivos. Por fim, é exposta a estrutura da presente dissertação.

No segundo capítulo é apresentado o estado da arte e o enquadramento teórico da dissertação, mais precisamente, são descritas as abordagens existentes de sistemas de monitorização e instalação de *WSNs* e apresentados os conceitos teóricos e tecnologias utilizadas neste trabalho.

No capítulo 3 é descrita a arquitetura do projeto *Eco-smart Heat Pump* e o enquadramento da presente dissertação na estrutura deste sistema, sendo especificada a arquitetura deste trabalho. De seguida, são especificados os quatro módulo constituintes do sistema e descrito o processo de instalação da rede utilizando as componentes deste sistema de instalação.

No capítulo 4 é descrita a fase de implementação da arquitetura e módulos especificados no capítulo 3, assim como as decisões mais importantes no processo de desenvolvimento das componentes do sistema.

No capítulo 5 são apresentados os resultados da implementação do sistema, sendo expostas as principais funcionalidades que compõe a arquitetura implementada.

Por fim, no sexto e último capítulo, são apresentadas as principais conclusões do trabalho realizado. Além disso, são definidas algumas sugestões para o trabalho futuro, de forma a melhorar e ampliar o sistema desenvolvido.

Capítulo 2

Estado da Arte

Neste capítulo é apresentada uma visão geral das abordagens existentes de sistemas de climatização baseados em *WSNs* (redes de sensores sem fios), tendo como foco o tema desta dissertação, os *GUIs* de instalação e manutenção de *WSNs* através de dispositivos móveis e suporte *Web*. De seguida é feita uma descrição dos conceitos, técnicas e tecnologias utilizadas no desenvolvimento das três componentes deste trabalho: i) Serviços Web e arquitectura SOA; ii) Técnicas de desenvolvimento de aplicações *Web* e iii) Desenvolvimento de aplicações para plataformas móveis.

2.1 Sistemas de climatização

Atualmente os sistemas de domótica baseados em redes de sensores sem fios inteligentes são compostos por uma grande diversidade de dispositivos para controlo e monitorização do ambiente da habitação, sendo possível controlar a temperatura, humidade, iluminação entre outras componentes. A conectividade com a *Internet* e rede *Wi-Fi* fornece grandes vantagens, permitindo a gestão destes sistemas e a potencialidade de atuação remota [6]. A interação com os dispositivos móveis também fornece grandes benefícios para estes sistemas, uma vez que, através da utilização das tecnologias *Bluetooth*, *NFC* e *Wi-fi* presentes nos *smartphones*, é possível interagir com o sistema e seus componentes, levando a que diversas empresas apresentassem as suas soluções de domótica, geridas através de *GUIs* baseados em aplicações móveis e/ou aplicações *Web*.

A solução *RedLink*$^{\text{TM}}$ *Wireless System* da empresa *Honeywell* [7] representa

uma *WSN* composta por diversos nós distribuídos por várias zonas do domicílio. O sistema contém uma central de controlo *TrueZONE Panel* com adaptador sem fios e diversos dispositivos para controlo da temperatura e humidade, como pode ser visualizado nas figuras 2.1 e 2.2. Cada zona contém um termóstato sem fios conetado ao painel de controlo para alteração e visualização dos parâmetros (2.4) das componentes controladas, neste caso temperatura.

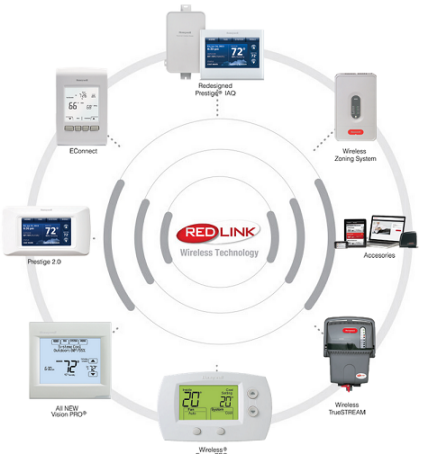


Figura 2.1: Sistema *Honeywell RedLink*^{TM} *Wireless System*

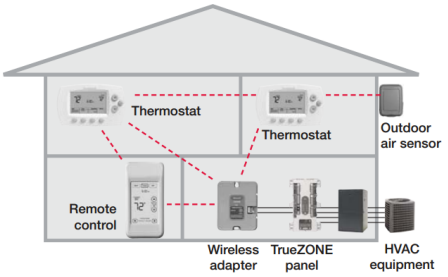


Figura 2.2: Exemplo de uma instalação do Sistema *RedLink*^{TM} num domícilio

Através do *Remote Control*(2.3), o utilizador tem a possibilidade de consultar e alterar os valores de temperatura de cada zona com um único dispositivo, e ainda, com aplicação móvel desenvolvida para as plataformas móveis *iOS* e *Android*(2.5), é fornecida a possibilidade de atuação remota. A empresa também disponibiliza uma aplicação *Desktop* para interação com o termostato e um *website*(2.6) para gestão dos vários sistemas associados ao cliente.

Figura 2.3: *RedLink*^{TM} *Remote Control*

Figura 2.4: *RedLink*^{TM} Thermostat

A instalação e conexão de cada nó do sistema é feita manualmente através da

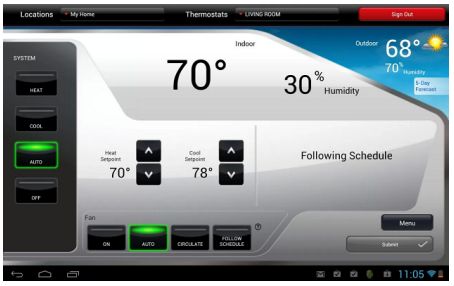


Figura 2.5: *RedLink*^{TM} *Website*

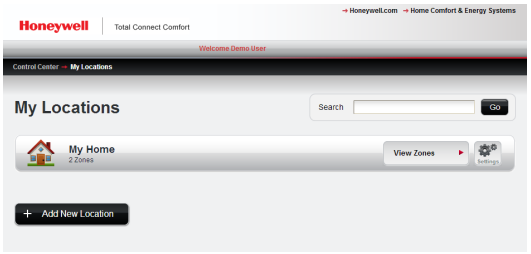


Figura 2.6: *RedLink*^{TM} Android App

opção de configuração presente no painel, figura 2.7 e 2.8, sendo necessário definir a zona e nome de localização do dispositivo. A identificação do dispositivo é feita através do próprio ecrã ou através da navegação do menu do *Remote Control*(2.3).

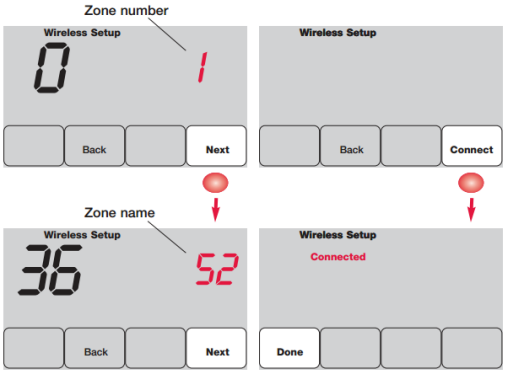


Figura 2.7: Exemplo da configuração da coneção sem fios da Consola do termostato ao Painel de Controlo

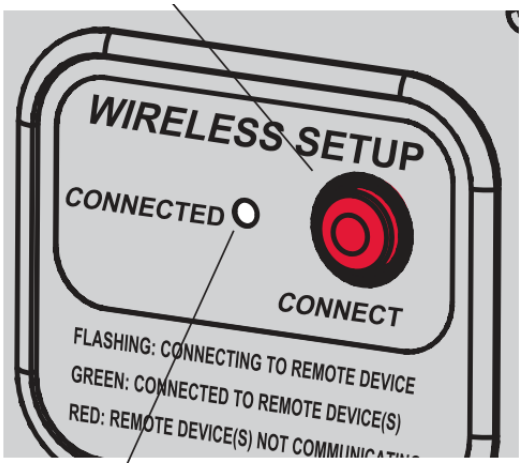


Figura 2.8: Parte traseira da consola do termostato - Botão e Led de sinalização de Conetividade

O pacote de *software HomeSeer HS2*[8] fornece ao utilizador a integração dos vários dispositivos da habitação, permitindo o seu controlo através de *GUIs* instalados no PC *Desktop*, aplicações móveis e também na *internet*, através de um *Website*, figura 2.9. Todos os dispositivos do sistema da *HomeSeer* funcionam com módulos baseados no protocolo de comunicação sem fios *Z-Wave*[9], permitindo o controlo e monitorização de temperatura, humidade, iluminação e ainda a interação com o sistema de segurança e telefónico. O controlo da temperatura é feita através do acionamento dos dispositivos *HVAC* conetados a termostatos através de uma rede baseada em tecnologias sem fios.

As aplicações permitem a configuração de todo o sistema, remotamente e localmente. Através de listagens é possível visualizar estados, eventos, *Logs*, dispositivos instalados e respetivas propriedades.

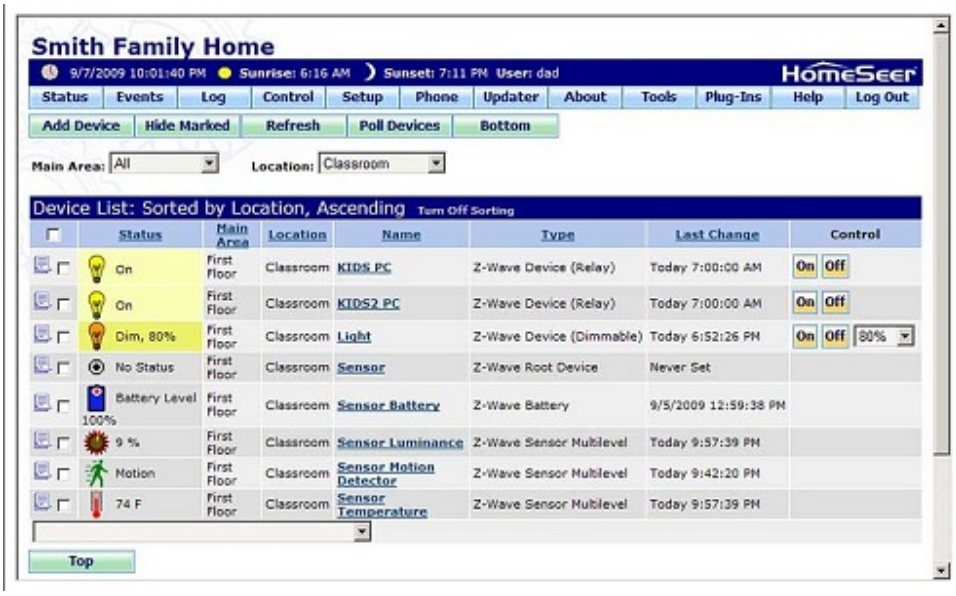


Figura 2.9: Website para monitorização do sistema *HomeSeer*.

O sistema da empresa *Control4* [10] tem um conjunto de aplicações gráficas direcionadas ao utilizador que permitem a interface com os dispositivos fornecidos, sendo possível controlar e monitorizar a climatização, iluminação e segurança da casa, estando disponíveis para *Android* (2.10), *iOS* e para *Desktop*.

Figura 2.10: Aplicação *Android* orientada para *Tablet* do sistema *Control4* que permite a correspondente monitorização.

Figura 2.11: Conjunto de dispositivos que disponíveis para utilização das aplicações do sistema *Control4*.

Devido à complexidade das redes de sensores sem fios e ao seu elevado número de nós e dispositivos conetados, o processo de instalação e manutenção por parte do fabricante e do instalador do equipamento, representa um processo árduo, nomeadamente na identificação dos nós e componentes da *WSN*. Por isso, em [1], o autor propõe um sistema de identificação e registo de componentes da *WSN* através da utilização de uma *tag* com um *Quick Response Code* anexado, como representado na Figura 2.12. Esta solução utiliza a *URI* do serviço de registo do

nó na base de dados contido no *QR Code* para indicação do número de série e o ID do respetivo nó (Figura 2.12).

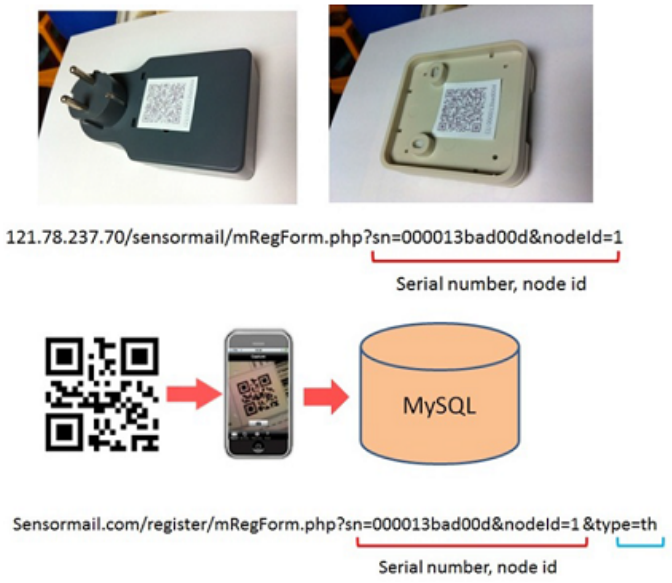


Figura 2.12: Implementação presente em [1] de um sistema de identificação de componentes utilizando *QR Codes*.

Concluindo, nos pacotes das soluções de domótica de controlo do ambiente das habitações, na generalidade, as empresas incluem *GUIs* para plataformas móveis(*Android* e *iOS*), *Desktop* e aplicações *Web* para gestão e monitorização tanto local como remota. Com estas aplicações é possível a configuração, edição e visualização dos parâmetros do sistema. A instalação das *WSNs* utilizando *QR Codes*, mostra-se como um modo muito útil para o instalador no processo de identificação e manutenção dos componentes da rede.

2.2 Plataformas móveis

Desde o lançamento dos primeiros *PDAs (Personal Digital Assistant)* com o sistema operativo móvel *Palm OS* em 1996, o mundo das tecnologias móveis tem sido uma grande aposta por parte das empresas devido à grande oportunidade de negócio pelo facto da constante procura deste tipo de dispositivos por parte da população. A ligação à *internet* surgiu nas versões seguintes com a possibilidade de consulta do *email*. Com o lançamento do *Windows Mobile* por parte da *Microsoft*, surgiram novas funcionalidades no campo da reprodução de conteúdos multimédia e de troca de mensagens, o *Messenger*. A evolução da tecnologia per-

mitiu a inclusão de cada vez mais periféricos nos dispositivos móveis, tais como câmara fotográfica, GPS, NFC, Bluetooth, Wi-fi, ligação de dados, não limitando os telemóveis apenas à tarefa de efetuar chamadas e o envio de mensagens escritas. Assim, surgiu o conceito de *Smartphone* e este tornou-se um dispositivo essencial para o dia a dia, tendo a partir do lançamento do *iPhone* uma massificação a nível Global, substituindo o comum telemóvel.

Apesar do desenvolvimento de aplicações para dispositivos móveis ser um conceito com vários anos, este teve o seu maior crescimento e notoriedade com a criação do mercado de aplicações *online* da Apple, a *App Store*, sendo um ótimo sistema para distribuição de aplicações para o sistema operativo móvel do *smartphone iPhone*, o *iOS* [11]. Com base neste conceito, várias empresas de desenvolvimento de sistema operativos para *smartphones*, também criaram as suas lojas de aplicações, onde os programadores podem desenvolver as suas aplicações e coloca-las nas *Application Stores*, necessitando apenas que estas cumpram regras definidas em relação ao seu desenvolvimento e inserção na loja. Para isto, cada empresa de desenvolvimento dos sistemas operativos, disponibiliza de forma livre, as suas próprias ferramentas de desenvolvimento nativas (*SDK*) [12], o que provocou um forte aumento do número e variedade de aplicações disponíveis nas lojas. Os *SDKs* mais utilizados são o *Android SDK* [13] da *Google*, que pode ser utilizado como extensão no *Eclipse IDE* [14], e o Xcode [15] da *Apple*. Estas ferramentas possuem um ambiente de desenvolvimento, um emulador dos smartphones e um conjunto de *application programming interfaces(API)* muito poderoso para tirar partido de todas as funcionalidades dos sistemas operativos e dos dispositivos móveis.

O *Android* é uma plataforma de desenvolvimento mais flexível, em comparação com o *iOS*, devido à variedade de sistemas operativos em que o ambiente de desenvolvimento pode ser utilizado, *Windows, Linux, MacOS* e de *IDEs*, tais como o *Eclipse, Netbeans, Android Studio*, etc. Por outro lado, o desenvolvimento para *iOS* é mais restrito, visto que o ambiente de desenvolvimento apenas pode ser utilizado no *SO MacOS* e o *IDE* é limitado ao *XCode*.

	Android	iOS
Multiplataforma	Sim	Não
Multi-IDE	Sim	Não
SO	Windows, Linux, MacOS	MacOS
IDE	Eclipse, Android Studio, Netbeans	XCode

Tabela 2.1: Características dos ambientes de desenvolvimento para as plataformas *Android* e *iOS*.

Segundo o relatório da empresa *Millennial Media*, no primeiro trimestre de 2013, o sistema operativo *Android* encontra-se com 52% e o *MOS iOS* com 39%, figura 2.13, do mercado dos sistemas operativos para *smartphones* [2].

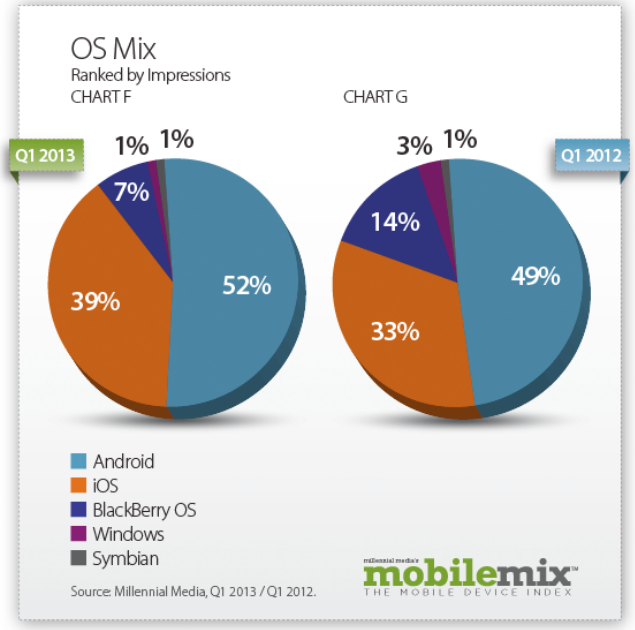


Figura 2.13: Gráfico da distribuição dos *MOS* no mercado dos *smartphones* [2]

2.2.1 Android

O sistema operativo móvel *Android*, desenvolvido pela *Google*, teve a sua primeira versão em 2008, o *Android 1.0*, lançado no primeiro *smartphone* do mercado com este *SO*, o *HTC Dream*. A versão 1.0 do Android fornecia suporte para câmara fotográfica, continha um navegador Web e suporte completo para páginas *HTML e XHTML*. A *Google* também desenvolveu uma loja de aplicações *online*, permitindo atualizações constantes das aplicações. O *Android* tem como base o *Kernel* do *Linux* e tem recebido constantes atualizações, chegando à versão *4.3 Jelly Bean*, baseada na versão 3.0.31 do *Linux* [3].

O sistema operativo móvel *Android* é uma *stack* de *software* dividida em cinco camadas, figura 2.14. A primeira, o *Kernel* do *Linux* é a camada de abstração entre a *hardware* e o *software*, contendo os drivers essenciais para interação com o *hardware*, foi modificado pela *Google* devido às necessidades especiais de gestão de memória e energia.

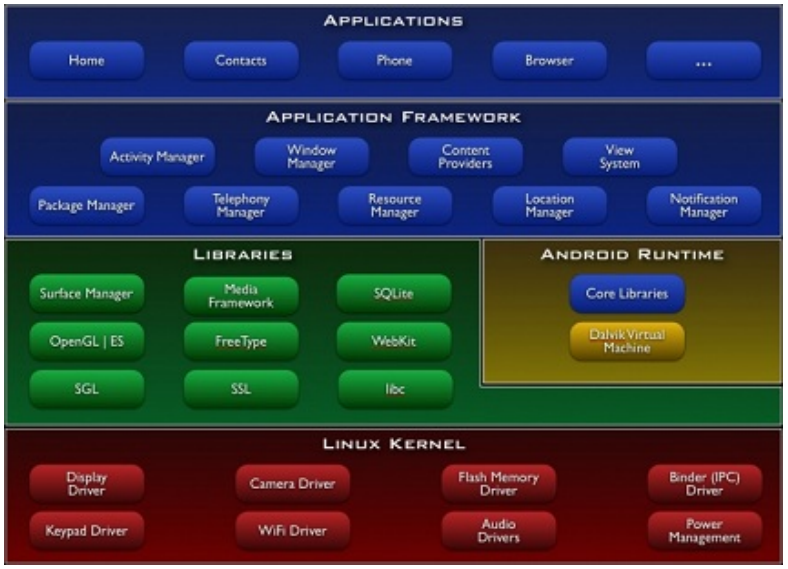


Figura 2.14: Arquitetura do sistema operativo *Android* [3]

Inclui os serviços essenciais do sistema, tais como, segurança de ficheiros, *threads*, gestão de processos e protocolos de rede. A seguinte camada contém as bibliotecas dos componentes do sistema, escritas em C e C++, também bibliotecas multimédia, visualização de camadas 2D e 3D, funções para navegadores *Web* e funções para acesso à base de dados *SQlite*. A camada *Runtime* contém as bibliotecas nativas *java* e a máquina virtual *Dalvik*. Este tipo de *JVM (Java Virtual Machine)* é usada para executar aplicações e está otimizada para ambientes com baixa capacidade de memória e processamento. A camada da *Framework* de aplicações é escrita em *Java* e fornece uma camada de abstração para interação com a DVM(*Dalvik Virtual Machine*). Simplifica a reutilização dos componentes, através do uso das *APIs* disponibilizadas, agilizando o processo de desenvolvimento por parte do programador. Contém:

- Acitivity Manager: gere o ciclo de vida das aplicações.
- Content Providers: para permitir às aplicações acederem aos dados de outras aplicações.
- Resource Manager: fornece acesso as recursos tais como *strings* e ficheiros de *layout*.

Na camada de topo encontram-se todas as aplicações, escritas em *Java* que fornecem clientes *email* e *SMS*, contatos e todo o tipo de funcionalidades que o utilizador necessitar.

Componentes de uma aplicação

Os componentes de aplicação são blocos essenciais de uma aplicação *Android*. Existem quatro tipos que servem um propósito distinto e um ciclo de vida distinto que define como o componente é criado e destruído [16]. Os quatro tipos são:

- **Activities**: Um ecrã com a interface com o utilizador de uma aplicação é representado por uma *Activity*, sendo que aplicação pode ser composta por diversas *Activities*. A atividade é implementada através da subclasse *Activity*.

- **Services**: O serviço é uma componente que é executado em *background* e realiza processos de longa duração ou operações para processos remotos. O serviço é implementado através da subclasse *service* e não disponibiliza interface de utilizador.

- **Content Providers**: O *content provider* gere um conjunto de dados partilhados da aplicação. Através do *content provider* as aplicações podem aceder às localizações de fontes armazenamento de dados persistentes. Por exemplo, o *MOS Android* fornece um *content provider* que gere os contatos do utilizador. Desta forma, é possível a sua consulta por parte de qualquer aplicação desde que o *content provider* o permita. Este componente é implementada através da subclasse *ContentProvider*, devendo implementar um conjunto de *APIs standard* de forma a permite que outras aplicações realizem transações.

- **Broadcast receivers**: É uma componente que responde aos anúncios globais do sistema, muitos são originados pelo próprio sistema, por exemplo, quando uma *SMS* é recebida, o ecrã é desligado ou uma fotografia é capturada. As aplicações também podem iniciar *Broadcasts*, por exemplo, permitindo que outras aplicações saibam que certa informação tenha sido descarregada e está disponível para utilização. Um *Broadcast receiver* é considerado como um *gateway* para outras componentes e é utilizado para realizar uma quantidade mínima de trabalho.

Os tipos componentes *activities, services* e *broadcast receivers* são ativados por uma mensagem assíncrona chamada de *Intent*. Assim, são criados através de objetos da classe *Intent* que definem a mensagem para ativar certos componentes ou tipos de componente e os ligam, de forma a requisitar uma ação de outros componentes, independentemente se pertencem ou não à mesma aplicação.

Antes do *MOS Android* iniciar um componente de uma aplicação, deve saber se este existe através da leitura do ficheiro da aplicação *AndroidManifest.xml*. Todas os componentes de uma aplicação devem ser declarados neste ficheiro e deve estar localizado na raiz da aplicação (*root*). Para além de conter os componentes da aplicação, também deve:

- Identificar qualquer permissão de utilizador que a aplicação necessite, tais como, acesso à internet ou leitura dos contatos;

- Declarar o nível mínimo necessário para a aplicação, definido como *API Level*, com base nas *APIs* que a aplicação utililiza;

- Declarar as funcionalidades de *Hardware* e *Software* necessárias para a aplicação, por exemplo, a câmera fotográfica, serviços *bluetooth* ou ecrã *multitouch*.

- Identificação de bibliotecas que a aplicação necessita ou outras *APIs* para além das da *framework Android*, por exemplo a biblioteca *Google Maps*.

Todos os elementos da interface de utilizador de uma aplicação são construídos utilizando objetos *view* ou *ViewGroup*. A *view* é um objeto que desenha qualquer coisa no ecrã que permite interação com o utilizador. A *ViewGroup* é um objeto que contém um conjunto de *Views* ou *ViewGroups* de forma a definir o *layout* da interface. A *framework* do *Android* fornece uma coleção tanto de classes de *Views* e *ViewGroups* que oferecem um conjunto de controlos de entrada (botões e caixas de texto) e modelos de *layouts* (*Linear e Relative Layout*). Os *Layouts* podem ser declarados através da instanciação de objetos *view* através de código, mas a forma mais fácil e efetiva, é através da definição do *layout* num ficheiro *XML*.

Numa aplicação *Android*, o *developer* deve definir externamente os recursos da aplicação, sendo definidos por imagens e *strings*. Assim, é possível fornecer recursos alternativos que suportem configurações específicas do dispositivo, tais como tamanhos de ecrã ou suporte para diferentes linguagens. Os recursos estão localizados na pasta
res do projeto da aplicação e devem estar organizados conforme o tipo e configuração. Quando a aplicação é compilada pelo *aapt tool*, é gerada a classe *R* com todos os *IDs* dos *resources* da aplicação presentes na secção
res.

2.2.2 iOS

O *iOS*, chamado de *iPhone OS* até à sua versão 4, é sistema operativo móvel utilizado em todos os dispositivos móveis da *Apple*, *iPhones, iPads e iPods*. A primeira versão foi anunciada em 2007 tendo sofrido constantes evoluções até à presente versão 7. Entre as principais encontra-se o suporte às capacidades *multitoutch* na primeira versão, sendo que no lançamento e suporte da loja de aplicações, *App Store*, no *iPhone OS 2* mudou a história dos *smartphones* em 2008. Nas versões seguintes foram adicionadas funcionalidades de *multitasking* de aplicações, centro de aplicações e a *Siri*.

O desenvolvimento de aplicações para *iOS* é realizado através do *iOS SDK* e do *IDE XCode* e através da linguagem *Objective-C*. A *framework* define conjunto fundamental de técnicas e *design patterns* para o desenvolvimento de aplicações de forma a fornecer uma estrutura critica da aplicação, uma vez que na maioria dos casos é a única forma de acesso ao *hardware*. Os *design patterns* fundamentais que o *developer* deve saber são:

- **Model-View-Controller**: Define a estrutura global da aplicação, permitindo separar o código das *views*, *controllers* e *models*. É utilizada a *framework Cocoa Touch* para gestão das interações de utilizador, permitindo a utilização de diversos componentes, entre eles botões, *table lists* e transições de páginas.

- **Delegation**: Facilita a transferência de informação e dados de um objeto para outro.

- **Target-Action**: Traduz interações do utilização com botões e controlos em código que a aplicação possa executar.

- **Block objects**: Utilizados para implementar *callbacks* e código assíncrono.

- **Sandboxing**: Todas as aplicações *iOS* são colocadas em *sandboxes* para proteção do sistema e outras aplicações. A estrutura do *sandbox* afeta a localização dos ficheiros das aplicações e contém implicações de *backups* de dados e funcionalidades das *app's*.

2.3 Web Services

Os serviços *Web* permitem a interação *machine-to-machine* numa rede, ou seja, fornece meios de interoperabilidade entre aplicações e sistemas de diferentes plataformas. São componentes de *software* que comunicam utilizando *standards* baseados nas tecnologias *Web HTTP* e mensagens baseadas em *XML*, definindo uma interface que descreve um conjunto de operações acessíveis por meio de uma rede [17].

O seu desenvolvimento tem como objetivo a comunicação entre várias aplicações e fonte de dados persistentes, podendo variar em termos de complexidade, desde simples operações, como verificação de estados, até às mais complexas como execução de processos em grande escala por parte de entidades financeiras.

Como são baseados no protocolo *HTTP* e nos protocolos *Web* baseados em *XML*, incluindo *SOAP* e *WSDL*, os *Web services* são independentes do sistema operativo, linguagem de programação e *hardware*. Isto significa que aplicações de linguagens e plataformas diferentes podem trocar dados sem restrições pela *Internet* ou redes locais (*LANs*).

Os serviços *Web* são descritos por *standards* que descrevem o modo de utilização, isto é, são compostos pelo formato da mensagem, protocolos de transporte e sua localização. As localizações dos recursos são identificadas por *URIs (Uniform Resource Identifiers)*, sendo que através destes, o cliente pode executar pedidos ao servidor [17][18][19].

Nas secções seguintes será apresentada uma breve descrição das duas principais implementações dos serviços *Web*: o *REST* e o *SOAP*.

2.3.1 SOAP

Simple Object Access Protocol ou simplesmente SOAP, é um protocolo baseado em *XML* que permite a troca de informação através da *internet*, fornecendo uma via de comunicação entre aplicações escritas em linguagens diferentes e executadas em plataformas distintas [19][20]. As mensagens trocadas são codificadas conforme o protocolo utilizado, normalmente *HTTP* e é utilizado o *WSDL (Web Services Description Language)* para descrever o serviço.

O *SOAP* define o formato, a arquitetura da mensagem e um elemento *XML* chamado de envelope que contém um cabeçalho (*header*) e um corpo (*body*). O cabeçalho é um recipiente extensível para a infraestrutura da informação da ca-

mada da mensagem e pode ser usado para *routing* e configuração da qualidade do serviço(transações, segurança e fiabilidade). O corpo da mensagem contém o *payload* da mensagem. A estrutura da mensagem é descrita usando a estrutura *XML* para que o protocolo *SOAP* nos dois pontos de interação possa ler e escrever o seu conteúdo de modo a direciona-la para a implementação apropriada. [20]. No excerto de código presenta em 2.1, é apresentado um exemplo de uma mensagem utilizando o protocolo *SOAP*. No corpo da mensagem está presente o caminho da função referente ao pedido "*http://www.example.org/stock*", o pedido *GetStockPrice* e um parâmetro *StockName*. O protocolo *HTTP* é definido no início da mensagem através da definição do método, do *host*, do tamanho e tipo da mensagem.

Listagem 2.1: Exemplo do conteúdo de uma Mensagem de um pedido utilizando o *SOAP*

```
1  POST /InStock HTTP/1.1
2  Host: www.example.org
3  Content-Type: application/soap+xml; charset=utf-8
4  Content-Length: nnn
5
6  <?xml version="1.0"?>
7  <soap:Envelope
8   xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
9   soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
10
11   <soap:Header>
12     <m:Trans xmlns:m="http://www.w3schools.com/transaction/"
13     soap:mustUnderstand="1">234
14     </m:Trans>
15   </soap:Header>
16
17   <soap:Body xmlns:m="http://www.example.org/stock">
18     <m:GetStockPrice>
19    <m:StockName>IBM</m:StockName>
20     </m:GetStockPrice>
21   </soap:Body>
22  </soap:Envelope>
```

2.3.2 REST

REST é um estilo arquitetural definido por Fielding [18], sendo que o termo é normalmente usado em conjunto com o protocolo *HTTP*. O *REST* foca-se no conceito de recursos e utiliza as potencialidades do *HTTP* para fornecer representações destes recursos em vários estados. Uma *URL* única identifica cada recurso e pode ser utilizada através dos comandos/métodos *HTTP*: *GET*, *POST*, *DELETE*

e *PUT* [18][20][17][21].

O termo *REST* é baseado num conjunto princípios fundamentais [18][20]:

Identificação de recursos através de *URIs*: Um serviço *REST* oferece um conjunto de recursos que são definidos por *URIs* para possível identificação por parte do cliente, fornecendo espaço de endereçamento para o recurso e um método para serem descobertos.

Uniform interface: Os recursos são manipulados utilizando o conjunto de operações *CRUD* (Create, Read, Update, Delete): *GET*, *POST*, *DELETE* e *PUT*.

Self-descriptive messages: Os recursos são abstraídos da sua representação para que o seu conteúdo possa ser acedido numa grande variedade de formatos, sendo necessário descrever o tipo de conteúdo do pedido através do cabeçalho da mensagem. São definidos no parâmetro *Content-type* e podem ser: *JSON (application/json)*, *XML (application/xml)* e *XHTML (application/xhtml+xml)*, entre outros.

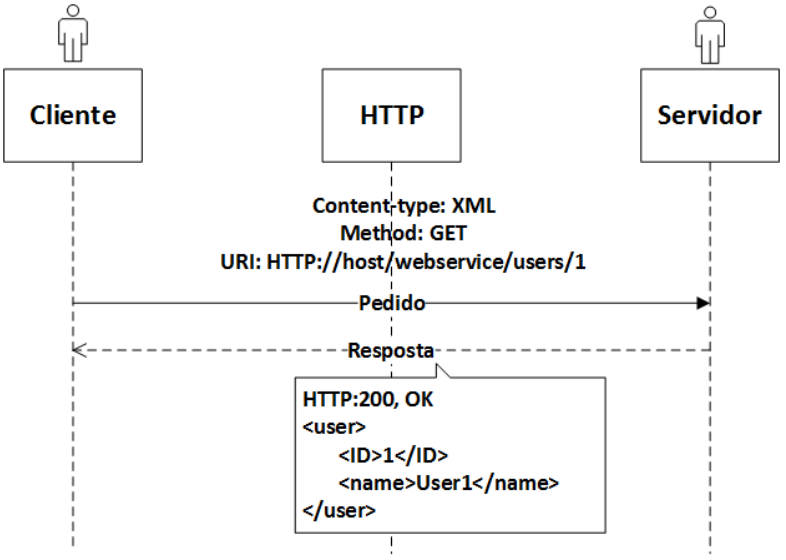


Figura 2.15: Exemplo de pedido e resposta num sistema Cliente-Servidor de um Serviço *Web*

A figura 2.15 representa um exemplo de um pedido ao serviço por parte do Cliente, é utilizado o método GET e o tipo de dados requisitado é *XML*.

2.4 Código QR

O *QR Code* 2.16 é um símbolo bidimensional, normalmente referido como parte da família dos códigos de barras, teve como primeira aplicação a ajuda ao

controlo de produção de componentes automóveis. Foi desenvolvido em 1994 pela empresa *Denso*, uma das maiores do grupo *Toyota*, vindo a ser aprovado como um *Standard* internacional ISO (ISO/IEC18004) em junho de 2000. Apesar de ser desenvolvido tendo como objetivo o setor automóvel, o código QR rapidamente se expandiu para várias áreas de aplicação tais como a gestão de stocks e inventários. Mas, devido à evolução dos *smartphones* e das suas câmaras fotográficas, a presença dos códigos QR em revistas e vários tipos de publicidade tornou-se uma prática comum, contendo endereços de *Internet* para aplicações móveis e sítios Web.

Figura 2.16: Código QR com a mensagem: "QR Code"

Nas vários usos dos *Quick Response Codes*, destacam-se o armazenamento de informações de contatos (*Email*, telefone), datas de eventos, conteúdo de SMS e a codificação de *URLs*. A sua massificação deveu-se às seguintes razões:

- Maior densidade e suporta uma maior gama de caracteres especiais.
- Suporta até um máximo de 7089 carateres numéricos e 4296 carateres alfanuméricos.
- Pode ser usado de forma livre devido à patente ter sido libertada para domínio público pela *Denso*.
- A maioria dos *smartphones* são equipados com uma câmara que permite a leitura dos códigos QR.
- Custo de produção bastante baixo visto que podem ser imprimidos em qualquer suporte em papel, podendo ser utilizados em embalagens, revistas, livros ou qualquer local publicitário.

Comparados com os códigos de 1D e 2D, os *QR Codes* têm maior capacidade num espaço mais reduzido devido ao seu método de *error-correction* e características únicas que permitem uma leitura mais fiável e com maior velocidade em

comparação com outros tipos de códigos de barras, o que pode ser observado na tabela 2.2.

Tabela 2.2: Características dos códigos de barras bidimensionais existentes

		QR Code	PDF417	DataMatrix	MaxiCode
					
Developer		DENSO Wave	Symbol Technologies	RVSI Acuity CiMatrix	UPS
Type		Matrix	Stacked barcode	Matrix	Matrix
Data capacity	Numeric	7,089	2,710	3,116	138
	Alphanumeric	4,296	1,850	2,355	93
	Binary	2,953	1,018	1,556	-
	Japanese, Chinese or Korean characters	1,817	554	778	-
Main features		Large capacity, small size, high-speed scanning	Large capacity	Small size	High-speed scanning
Main applications		All categories	Office automation	Factory automation	Logistics
Standards		AIM, JIS, ISO	AIM, ISO	AIM, ISO	AIM, ISO

O código *QR* contém níveis de correção de erro de forma a permitir a sua leitura com poucos preocupações em relação ao ângulo e foco da câmera e mesmo em cada de dano no código ou mesmo existência de ruído. Assim, foram definidos quatro níveis, cujas percentagens de espaço do código correspondem a dados de correção de erros:

- Nível L (Low) 7%
- Nível M (Medim) 15%
- Nível Q (Quartile) 25%
- Nível H (High) 30%

Ao contrário de outros tipos de códigos de barras de 1-D, que representam informação através da posição, tamanho e distância das barras num único eixo (horizontal), os *QR Codes* organizam a informação através dos elementos "claros" e "escuros", chamados de módulos [4], tanto na vertical como na horizontal. Cada módulo escuro ou claro de um código representa 0 ou 1 respetivamente. Os módulos fornecem diversas funções, alguns representam dados, enquanto outros compõem padrões que melhoram a qualidade de leitura, permitem alinhamento dos símbolos, correção de erros e compensação da distorção do *QR Code*.

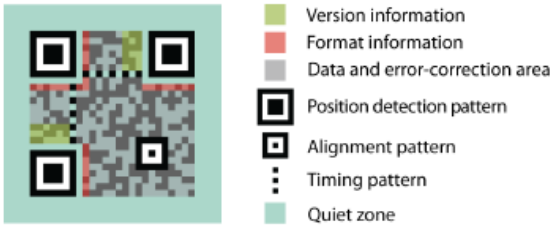


Figura 2.17: Estrutura de um *QR Code* [4]

2.5 NFC

No início década de 90, as capacidades da tecnologia RFID foram amplamente exploradas na tentativa de conectar o mundo físico com o mundo virtual. Em 2004, grande empresas como *Sony*, *Philips* e *Nokia* formaram uma aliança para criar *standards* no uso da tecnologia *RFID (Radio Frequency IDentification)*, na área das aplicações de consumo, resultando na criação do *NFC Forum*[22] e do protocolo de comunicação sem fios de curta distância, o *NFC (Near-Field-Communication)*. Esta tecnologia utiliza indução por campo magnético para comunicação sem fios entre dispositivos eletrónicos, com uma distância máxima de utilização de 20cm, operando numa frequência de 13.56MHz com velocidades entre os 106kbit/s e 424kbit/s e tem como base os *standards ISO*: *ISO/IEC 18092, ISO/IEC 14443 e ISO/IEC 15693*[23][24]. Os dispositivos móveis (*Smartphones* e *Tablets*) representam a área onde a tecnologia *NFC* tem maior incidência, onde oferece inúmeras funcionalidades, tais como, pagamento rápidos, rápido acesso à informação de cartazes e *posters*, cobrança de bilhetes, troca de dados com outro dispositivo.

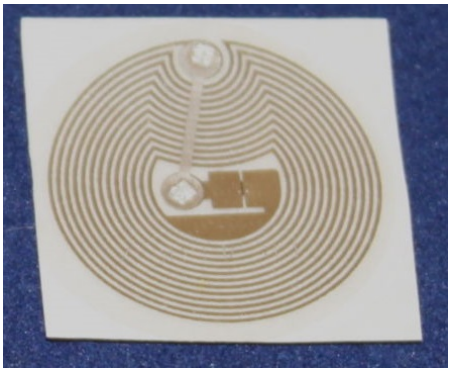


Figura 2.18: TAG NXP NTAG203 - 144 bytes, *Standard* ISO 14443A e Frequência de operação 13.56MHz

As *TAGs NFC*, figura 2.18, são uma ferramenta muito útil para armazenar

endereços de *Websites* e de *download* de aplicações utilizando a mensagem NDEF *(NFC Data Exchange Format)*. Cada mensagem NDEF pode estar dividida em vários registos NDEF. Estes registos contém o tamanho e o *RTD (Record Type Definition)*, ou seja, o tipo de informação que descreve a sua função. Os tipos de *RTD* mais usados são:

- Texto simples: permite escrever *strings* simples usando *ASCII* ou *unicode*.
- *Unique resource identifier*: permite a escrita de *URIs*, contendo o *header "http://www."*. A aplicação que receber este tipo NDEF pode passar automaticamente para outra para processamento, por exemplo um *Web Browser*.
- *Generic control*: Contém um *URL* para ações para iniciar aplicações.
- *Telephony*: *Tags* com este tipo de *RTD* permitem iniciar automaticamente a chamada telefónica, através a inicialização do *Dial* e a passagem do número por parâmetro.

2.6 Conceitos de Engenharia de software

Nesta secção são apresentados os conceitos mais importantes desta dissertação relacionados com a Engenharia de *Software*. Será feita uma breve descrição do conceito de *framework* e do modelo arquitetural *MVC (Model-view-controller)* no desenvolvimento de aplicações.

2.6.1 Conceito de Framework

Desde sempre, um dos principais objetivos da engenharia de software é a reutilização de *software*. Com este conceito é possível a diminuição do *Time-to-market* das aplicações através da reutilização de código, reduzindo o esforço necessário por parte do *Developer*. Com o surgimento da paradigma POO e o aumento da sua popularidade, a reutilização de grandes componentes tornou-se uma realidade, resultando na definição das *frameworks* orientadas a objetos.

Framework pode ser definido como um conjunto de classes abstratas que descrevem o modelo dos componentes das bibliotecas, tendo como a possibilidade de extensão destes componentes, um dos seus principais pontos fortes [25]. Representa um *design* abstrato de um tipo de aplicação e consiste num número de classes que que podem ser retiradas de uma biblioteca ou podem ser específicas

de uma aplicação, oferecendo métodos e funcionalidades para a reutilização das classes através dos conceitos de herança e polimorfismo [25]. Devido ao grande número de definições para *Framework* Orientada a Objetos, o autor de [26], após de várias definições analisadas, define que:

> "*A (generative) architecture designed for maximum reuse, represented as a collective set of abstract and concrete classes; encapsulated potential behaviour for subclassed specializations.*"

O conceito de *Framework* é considerado como a base de desenvolvimento de aplicações e fornece grandes vantagens neste processo. Existem diversas definições para este conceito, mas pode-se afirmar que um *Framework* engloba um conjunto de funções, rotinas e classes que evitam que o processo de desenvolvimento de projetos seja feito do zero e a cada novo projeto.

2.6.2 *Model-View-Controller (MVC)*

O *pattern*[1] *Model View Controller (MVC)* foi introduzido por *Trygve Reenskaug* na *framework* para a plataforma *Smalltalk 80* na década de 70, sendo que se tem tornado um termo comum, tendo grande influência na maioria das *frameworks* com interface de utilizador e no processo de decisão nas arquiteturas das aplicações gráficas [27][28]. Este conceito visa a separação da código da interface gráfica do código da lógica de negócio e controlo da aplicação, fazendo-a mais flexível, com mais facilidade de migração e potenciando a reutilização de código. Assim, é possível a alteração drástica da componente gráfica sem afetar as estruturas de dados e a lógica da aplicação, permitindo também interfaces diferentes como diversas linguagens ou um conjunto de permissões de utilizador [29].

A arquitetura *MVC*, figura 2.19, é definida em três partes: o *model*, a *View* e o *Controller*. O *Model* representa um conjunto de classes e a informação que modelam e suportam a aplicação, não possuindo qualquer visibilidade para outras componentes. A *View* contém a interface gráfica, mostrando janelas e informação ao utilizador. Por último, o *Controller* processa os eventos e ações do utilizador através da manipulação do *model* e da *view*. O autor, em [28], defende que o *pattern MVC* faz a separação entre a componente da interface e *model* e também a separa o *controller* da *view*, devido a:

[1] Em [21], um *Design pattern* é uma descrição de objetos e classes que são personalizadas para resolver um problema geral num contexto particular.

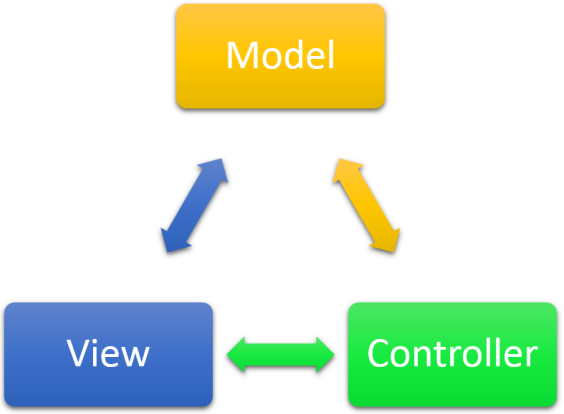


Figura 2.19: Arquitetura *Model-View-Controller*

- No ato de desenvolvimento da componente gráfica, o *developer* pensa sobre os mecanismos da interface com o utilizador, sendo que com o *model* o pensamento recai sobre as componentes de negócio, normalmente interações com a base de dados. Por isto, são utilizadas diferentes bibliotecas, envolvendo cada componente.
- Normalmente, o utilizador pretende ver a informação dos modelos de várias formas. Esta divisão vai tornar este processo mais flexível, permitindo a reutilização de código.
- Facilidade de teste das componentes, evitando processos de teste complexos, tendo a possibilidade de testar cada componente separadamente.

2.6.3 *Data Access Object (DAO)*

Diversas aplicações utilizam um arquitetura que contém uma componente de dados persistentes (ex: base de dados). A prática comum nas aplicações é a inclusão da lógica de acesso às base de dados na lógica de negócio, como pode ser visto na figura 2.21. Isto significa que o objeto de negócio conhece os seus dados mas também como e onde acede-los. Esta solução faz com que cada vez que a fonte de dados persistentes seja alterada a lógica da aplicação também tem que mudar, de modo a aceder aos novos dados. De modo a evitar estes problemas, o *D. Mati, et al.* em [5], propõe a separação destas componentes, inserindo um camada intermédia de abstração, o chamada de *data acess object (DAO)*.

O *design pattern Data Access Object (DAO)* permite criar um camada de abstração entre a fonte de dados persistente e as componentes de negócio, en-

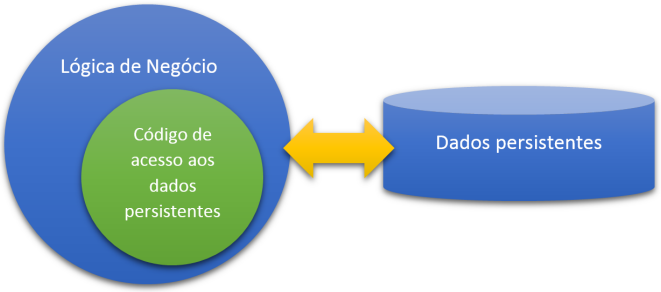


Figura 2.20: Exemplo de acesso à fonte de dados persistentes sem separação entre código de acesso do de lógica de negócio [5]

capsulando o seu acesso direto. Assim é possível controlar como a informação é acedida e se ocorrer uma migração da base de dados, apenas o código da camada DAO necessita de ser modificado. O DAO contém a lógica de acesso à fonte de informação e gere a conexão para obtenção e armazenamento de dados.

Figura 2.21: Exemplicação da utilização de uma camada de abstração *DAO* entre o código da lógica de negócio e da fonte de dados persistentes[5]

2.7 *Frameworks PHP*

O *PHP* é uma linguagem de *script* muito utilizada no desenvolvimento de aplicações *Web* devido à possível integração de outras linguagens com o *PHP*. É cada vez mais importante no mundo de desenvolvimento *Web* devido à sua facilidade de utilização, robustez, ser *open-source* e multiplataforma [30].

Atualmente, existem diversas *frameworks* em *PHP* para desenvolvimento *Web* que permitem o rápido desenvolvimento através da implementação de diversos *patterns*, entre os principais encontra-se o *MVC*, ORM, *Templates*, *DAO* e conexão a múltiplas base de dados. A maioria das *frameworks* mais utilizadas destacam-se pela implementação destes *patterns*, pela sua vasta comunidade de *developers* e documentação muito completa. Entre estas, destacam-se: *Yii, CodeIgniter, Zend, Cake PHP* e *Symfony*.

2.8 Cloud Computing

Cloud computing não é um conceito novo, mas apenas recentemente teve o seu maior crescimento. Com a evolução das tecnologias e a melhoria no processamento e armazenamento, permitiu o crecimento do conceito de computação em nuvem, onde estes recursos podem ser "alugados" pela *internet* conforme as necessidades dos utilizadores, sendo apenas cobrado o custo do tempo de utilização. Assim, recursos de alto desempenho estão disponíveis aos utilizadores e empresas sem necessidade de um elevado investimento na compra de *hardware* e evitando máquinas físicas e custos de manutenção [31]. Três aspetos relativos ao *hardware* e ao preço destacam-se na computação em nuvem:

- Os utilizadores não necessitam de planear os recursos necessários devido à disponibilidade "ilimitada" de recursos na *Cloud*.
- As empresas não necessitam de realizar compromissos a longo termo porque com a *Cloud* podem fazer a alteração dos recursos de *hardware* conforme as suas necessidades.
- A possibilidade de pagar os recursos conforme o tempo de utilização.

A definição de computação em nuvem refere-se como aplicações fornecidas como serviços pela *internet* e os sistemas de *hardware* e software nos data centers que fornecem estes serviços [32]. Estes serviços fornecidos são compostos por bases de dados, armazenamento, plataformas, segurança, entre outros, podendo ser geridos dinamicamente conforme as necessidades do utilizador [33].

2.9 Conclusões

Assim, para gestão e monitorização das soluções de domótica no controlo do ambiente da habitação, as empresas desenvolvem *GUIs* para diversas plataformas, sendo as aplicações *Web* e as aplicações desenvolvidas para os sistemas operativos móveis as mais utilizados. As aplicações para as plataformas *Android* e *iOS* e os *Websites*, são ótimas ferramentas para gestão remota, utilizando serviços *Web* para comunicação com os sistemas. As empresas também apresentam aplicações para sistemas *Desktop*, na maioria para o sistema operativo *Windows*, limitando a atuação localmente. As aplicações são compostas por listas de elementos necessários para a monitorização, tais como, eventos, *Logs*, parâmetros e configurações

do sistema.

A utilização dos *QR Codes* no processo de instalação e manutenção de componentes de soluções de *WSNs*, é uma ótima solução para redução do tempo e melhoria na facilidade da execução do processo por parte do instalador e fabricante do produto, permitindo a rápida identificação e visualização dos dados dos elementos da rede.

A tecnologia *NFC* também se enquadra numa boa solução para identificação dos nós da rede através da utilização de *TAGs* com a informação respetiva. O uso desta tecnologia na instalação de *WSNs* tem a desvantagem de ter um custo mais elevado em comparação à utilização de *QR Codes*, visto que os custos de impressão de um código são bem menores, em relação à compra de uma *TAG NFC*. A segunda desvantagem em relação ao uso do NFC mostra que apenas um número muito baixo de *smartphones* vêm equipados com esta tecnologia, enquanto que a maioria destes dispositivos contêm uma câmara instalada que pode ser usada na leitura e descodificação dos *QR Codes*.

As plataformas *Android* e *iOS* detêm a maior parte do mercado dos *smartphones* e *Tablets*. Uma das principais razões deste domínio é o vasto conjunto de aplicações disponibilizado nas lojas *Online*. Isto deve-se à disponibilização de (*SDKs*) de forma livre para o desenvolvimento de aplicações, sendo que os *developers* podem retirar dividendos da publicação e venda das aplicações. O *Android* tem ligeiras vantagens em relação ao ambiente de desenvolvimento em comparação com o *iOS*, devido à sua flexibilidade na utilização em diversos sistemas operativos e diversos *IDEs*. Isto porque, o desenvolvimento para *iOS* exige maiores gastos na compra de dispositivos de teste (*iPhone* e *iPAD*) e de plataformas *MacOS*, uma vez que os seus preços são bastante superiores em comparação com os dispositivos *Android* e aos PCs com os *SO Windows* ou *Linux*. Uma das grandes desvantagens da plataforma *iOS* é a falta de suporte à tecnologia *NFC*.

Os serviços *Web* são uma ferramenta muito útil no desenvolvimento de arquiteturas porque permitem a interface e troca de dados entre aplicações executadas em plataformas distintas e com linguagens diferentes. Ao contrário do *SOAP*, o *REST* é baseado e apenas suporta o protocolo *HTTP*, tirando partido de todas as suas potencialidades. Em [20], o autor defende que o *REST* tem melhor performance, é mais flexível e são necessárias menos decisões a nível arquitetural em comparação com o *SOAP*. A simplicidade do *REST* evita a tomada de uma série de decisões estruturais relacionadas com a pilha do protocolo *SOAP*. Mas se forem utilizadas as mesmas funcionalidades e tecnologias, as duas abordagens são consideradas similares. O *SOAP* pode ser considerado como uma melhor abordagem

ao nível empresarial, devido às características de *QoS (Quality-of-Service)*, fiabilidade e segurança a nível da mensagem. Enquanto que o *REST* é considerado mais flexível e melhor solução para integrações *ad hoc*.

Concluindo, após análise dos conceitos apresentados neste capítulo, é possível assumir que para a arquitetura da presente dissertação, o *Android* encontra-se como uma excelente plataforma para o desenvolvimento da aplicação móvel devido ao seu suporte de *NFC*, flexibilidade das plataformas de desenvolvimento e baixo custo dos dispositivos de teste. Também devido à sua flexibilidade ao nível da arquitetura, os serviços *REST* são uma excelente solução para implementação da interface entre a base de dados e as aplicações cliente. Ao nível da aplicação de gestão e monitorização, a utilização de uma *framework* em *PHP* fornece diversas vantagens ao nível de redução do tempo de implementação, robustez da estrutura da aplicação devido à implementa do *MVC*, mas também ao nível de flexibilidade da utilização da base de dados devido à implementação do *Data Acess Object*.

Capítulo 3

Especificação do Sistema

No capítulo anterior foi possível enquadrar e expor os principais conceitos relacionados com o tema da dissertação. Além disso, foi possível definir as melhores orientações e abordagens para a arquitetura do sistema. Portanto, neste capítulo apresenta-se a descrição da arquitetura do projeto de investigação "Eco-Smart Heat Pump", assim como o enquadramento da presente dissertação neste projeto. De seguida, será descrita a arquitetura da GUI de instalação e manutenção, ou seja, serão apresentados os seus requisitos, os casos de uso, bem como a especificação da funcionalidade de instalação da *WSN*. Esta especificação é composta pela apresentação dos seguintes módulos e respetivas estruturas:

- Base de dados;
- Aplicação *Web*;
- Serviços *Web*, definição das respetivas funções e estrutura das *URLs*;
- Aplicação *Android*.

3.1 Arquitetura *Eco-Smart Heat Pump*

O projeto "Eco-Smart Heat Pump" visa do desenvolvimento de um sistema para controlo e monitorização de uma bomba de calor de uma habitação, ar e água, representando uma otíma solução de domótica para aquecimento e arrefecimento de ambientes e de águas sanitárias, numa colaboração entre a Universidade do Minho e a empresa Pinto Brasil, Fábrica de Máquinas Industriais S.A..

O sistema é composto por uma rede de sensores sem fios definida por cinco

nós correspondendo a uma zona cada. Os nós (*Sensor Actuator Board*) são compostos por um sensor de temperatura, um de sensor de humidade e um atuador para o ventiloconvetor, como indicado na figura 3.1. A *WSN* contém um nó coordenador, o *Coordenator Gateway Board*. Como sendo o nó central da rede, é responsável pelo controlo dos nós e o *Hardware* da bomba de calor. Cada nó é composto por um microcontrolador *CC2530* da *Texas Instruments* e utilizado o protocolo RF *Simplicity* [34] para a gestão da camada da rede. A aplicação móvel *Android* representa outra componente do sistema e fornece uma interface ao utilizador para monitorizar e definir os parâmetros de temperatura e humidade das zonas. A comunicação entre a aplicação *Android* e o nó coordenador é feita através de *SMS*, para isso, é utilizado um módulo GSM no coordenador para permitir este tipo de comunicação.

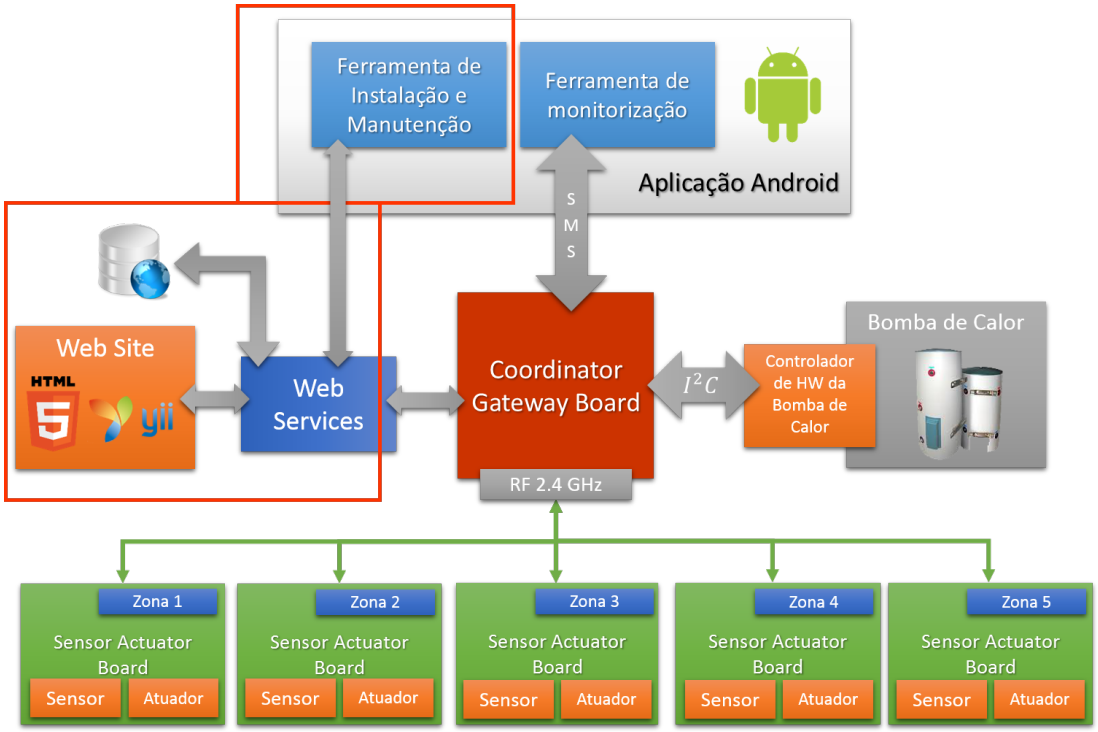


Figura 3.1: Arquitetura do Sistema *Eco-Smart Heat Pump*

A arquitetura do sistema é constituída por uma ferramenta de instalação e manutenção, representada na secção delimitada por a caixa a vermelho na figura 3.1. Esta componente contém um *Website*, uma aplicação *Android*, uma base de dados e um conjunto de serviços *Web* para interação entre as componentes da ferramenta, como pode ser visto na figura 3.2. Esta secção representa o tema desta dissertação. Assim o principal objetivo é composto pelo desenvolvimento e

implementação da arquitetura da ferramenta de instalação e manutenção da *WSN* do projeto "Eco-Smart Heat Pump".

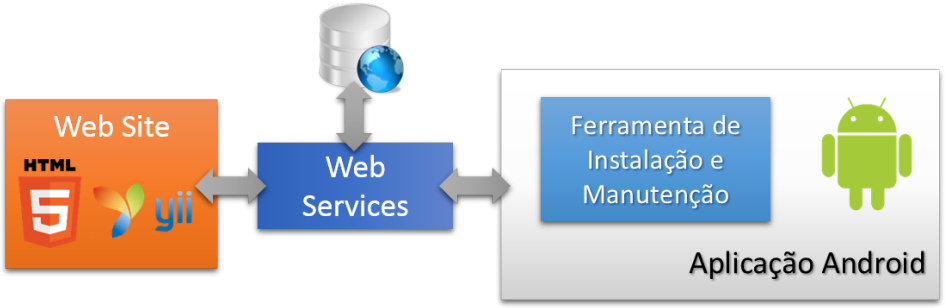


Figura 3.2: Arquitetura da ferramenta de instalação

3.2 Sistema de Gestão e Instalação

A presente dissertação tem como objetivo o desenvolvimento de um sistema para gestão e instalação do projeto "Eco-smart Heat Pump", de modo a auxiliar o instalador na identificação dos componentes da rede através da leitura da informação contida num *QR Code*. Desta forma, é utilizada a aplicação *Android* para visualizar os dados do nó e proceder à sua instalação ou manutenção, tirando partido das funcionalidades da câmara do *smartphone* para a leitura do código. Além disso, através do suporte para *NFC* do *smartphone*, também é possível a utilização de *TAGs NFC* para identificação do nó.

O sistema é composto por um conjunto de serviços *Web* que possibilitam a interface entre a aplicações *Android* e nó coordenador da rede com a base de dados do sistema. Através do *Website* será possível gerar os *QR Codes* e gerir todos os componentes do sistema, incluindo *WSNs* e respetivos nós, assim como eventos e utilizadores associados.

3.2.1 Requisitos do sistema

Como especificado no Capítulo 1, a arquitetura do sistema e os seus módulos devem cumprir os seguintes requisitos:

Requisitos funcionais

- A aplicação *Android* deve possuir uma funcionalidade de leitura dos *QR Codes* e consequente comunicação com a base de dados através dos serviços *Web*, de forma a possibilitar a recepção e visualização da informação dos componentes identificados pelo código.
- O *Website* tem que permitir a geração de *QR Codes* com informação referente aos nós da *WSN* e permitir a sua adição a um ficheiro *PDF*;
- Possibilidade de visualização e gestão das *WSNs* e respetivos componentes através do *Website*, com restrições, permissões e níveis de acesso aos diversos tipos de utilizadores;
- O conjunto de serviços *Web* deve satisfazer as necessidades das aplicações *Web*, *Android* e do nó coordenador da rede, através da requisição de recursos da base de dados recorrendo à utilização do protocolo *REST*.

Requisitos não funcionais

- A aplicação *Android* deve ser desenvolvida utilizando as funcionalidades da *framework* de desenvolvimento *Android SDK* e do *IDE Eclipse* usando a linguagem de programação *JAVA* e a linguagem de marcação *XML*.
- Para a implementação do *Website* e dos serviços *Web*, deve ser utilizada a *Framework Yii*.
- A base de dados deve ser relacional e utilizar o sistema de gestão de base de dados *MySQL*.

3.2.2 Arquitetura do sistema

Como referido anteriormente, a arquitetura do sistema é composta por quatro módulos: serviços *Web*, aplicação *Android*, aplicação *Web* e base de dados relacional. Esta arquitetura está representada no diagrama da figura 3.3.

Após a sua análise, é possível verificar a existência das duas aplicações cliente, a aplicação *Android* e o *Website*. Estes *GUIs* fornecem a possibilidade do utilizador interagir com o sistema, sendo que a aplicação *Android* é do uso exclusivo do ator

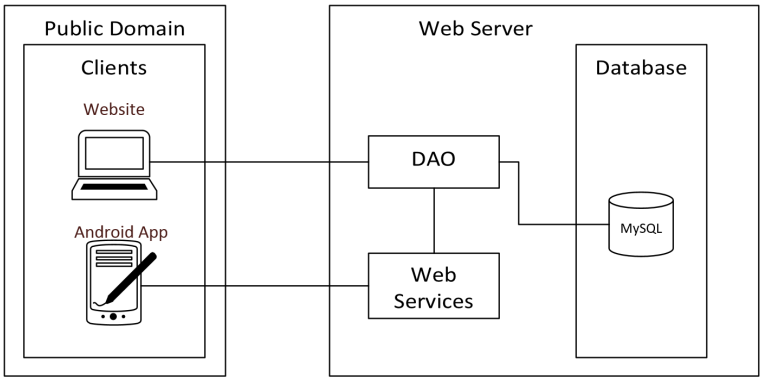


Figura 3.3: Arquitetura do Sistema

instalador da *WSN*. A aplicação *Web* tem como atores o administrador da rede, o fabricante, o instalador e o utilizador da rede.

Atores e contexto

O sistema tem como propósito disponibilizar um conjunto de ações e funcionalidades aos atores, fornecendo acesso a cada uma consoante o tipo e nível de acesso, figura 3.4. Assim, foi definido o nível "Administrador" para permitir o acesso a todas as funcionalidades do sistema, ou seja, o utilizador com este nível de autenticação tem acesso a todas as ações sobre os componentes do sistema, podendo ser definido como um *superutilizador*. A função deste tipo de utilizador é caraterizada por monitorizar o sistema e garantir o seu correto funcionamento, sendo garantidas permissões de acesso à gestão dos utilizadores e controlo dos seus níveis de autenticação. O utilizador "fabricante" é responsável pela gestão das *Tags NFC* e *QR Code*, sendo concedido acesso às informações dos nós para realizar o conjunto de ações associadas às *TAGs*. Este nível de autenticação apenas permite o acesso à aplicação Web. O "Instalador" utiliza a aplicação *Android* para realizar o processo de instalação e manutenção, através da leitura das *TAGs* que identificam os componentes da Rede. Este nível também permite autenticação no *Website* e monitorização das *WSNs* associadas a este tipo de utilizador. O "utilizador" da rede apenas tem acesso ao *Website* e um número restrito de ações referentes às redes que lhe pertencem. Para o *Access Point* da *WSN*, o *CGB (Coordinator Gateway Board)*, foi definido um nível de autenticação para acesso aos serviços *Web*.

A tabela 3.1 resume as permissões de acesso às componentes do sistema de cada utilizador.

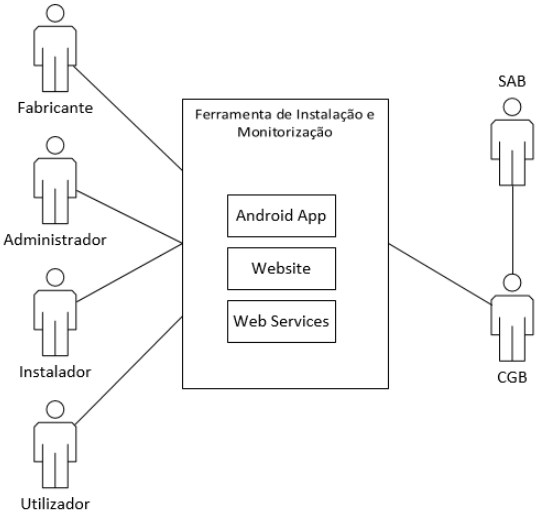


Figura 3.4: Representação em diagrama de casos de uso dos atores do sistema.

	Admin.	Instalador	Fabricante	Utilizador WSN	CGB
App. Android	x	x	-	-	-
App. Web	x	x	x	x	-
Serviços Web	x	-	-	-	x

Tabela 3.1: Tabela de permissões de utilização para as aplicações do sistema

Casos de Uso e descrição

Como referido na secção anterior, o sistema é composto por um conjunto de ações divididas pelos vários módulos, por sua vez, associadas aos cinco diferentes tipos de atores. Com permissões de acesso ao subsistema *Website*, é possível executar um conjunto de operações *CRUD* sobre os modelos existentes, tais como, as *WSNs*, nós e utilizadores. O diagrama de casos de uso na linguagem de modelação *UML* presente na figura 3.5 representa um visão geral das funcionalidades do sistema.

O nível de autenticação "administrador" permite acesso a todas as ações do sistema, possibilitando a manutenção do sistema de forma flexível e facilidade no acesso em caso de erros ou problemas no sistema. A principal funcionalidade associada ao "fabricante" é a possibilidade de geração dos *QR Codes* com respetivas informações do nó após o sua produção, sendo garantias permissões às ações associadas aos modelos *TAGs*. Para controlo do processo de instalação ou manutenção

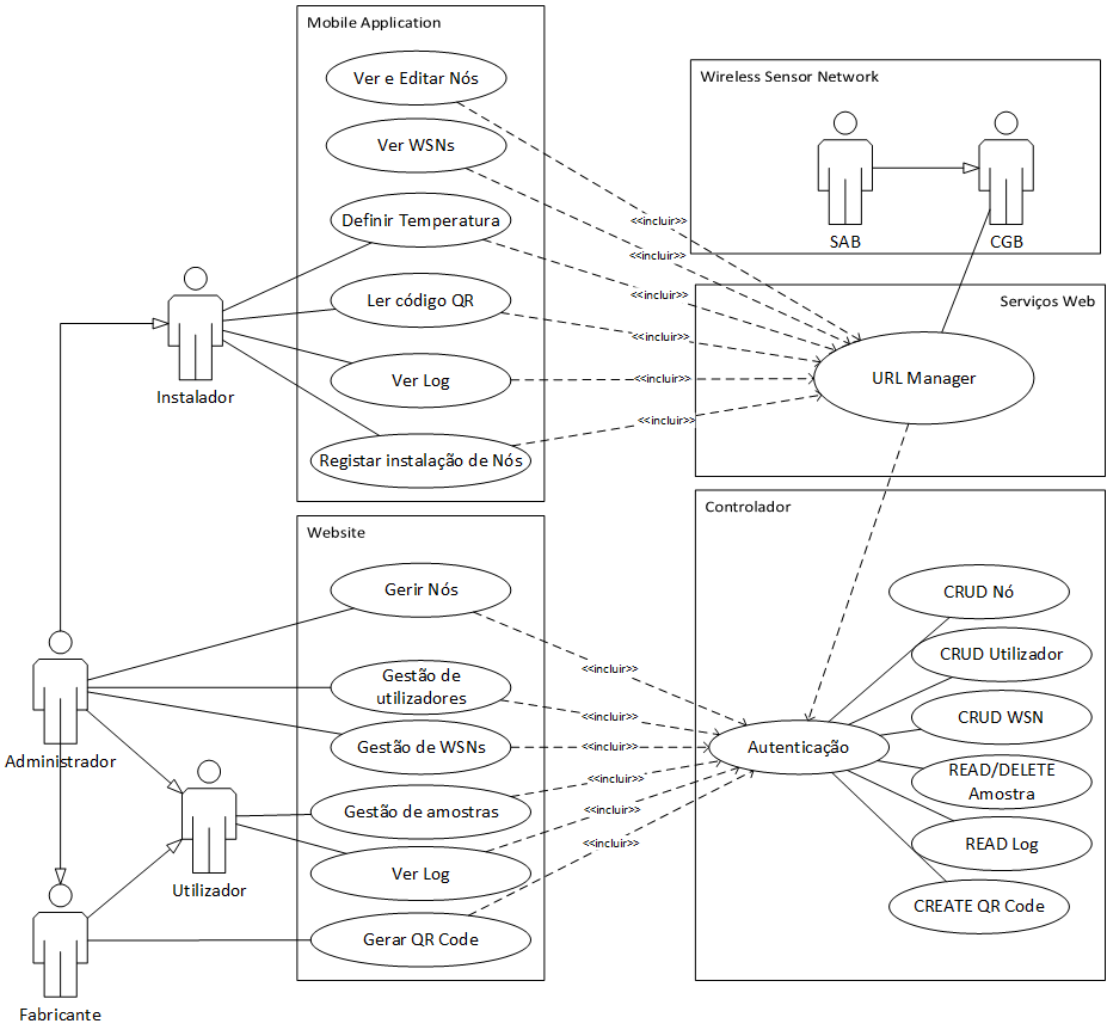


Figura 3.5: Diagrama de casos de uso do sistema.

da rede, o utilizador "instalador" tem permissão de acesso à aplicação *Android*. Com este GUI, é possível identificar os nós e consultar a informação associada através da leitura do *QR Code*. Ainda permite consulta da *WSN* com os respetivos nós com opção de alteração do seu estado de instalação. Por isso, a este nível de autenticação são dadas permissões de utilização das ações *CRUD* aos modelos *WSN* e nós.

O nível "utilizador da WSN" é atribuído ao proprietário da rede de sensores, fornecendo-lhe a possibilidade de consultar o estado de funcionamento, erros e monitorização da atuação do sistema, ou seja, operações *READ* para cada modelo associada à rede. O nó coordenador da rede de sensores (CGB) sem fios deve adicionar e atualizar os nós *End Device*, assim como registar amostras dos sensores através da utilização dos serviços *Web* disponibilizados pelo sistema. O nível definido para este tipo de utilizador foi "Coordinator", figura 3.6.

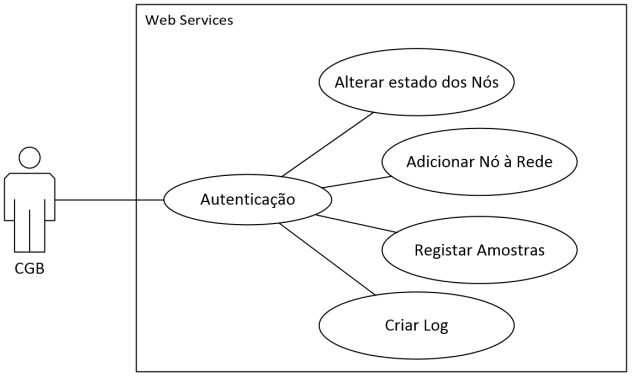


Figura 3.6: Diagrama de casos de uso que representa as ações possíveis para o ator *CGB*.

A tabela 3.2 resume as operações possíveis sobre os modelos do sistema associadas a cada tipo de utilizador, descritas nos parágrafos anteriores.

	Admin.	Instalador	Utilizador WSN	Fabricante	CGB
WSNs	CRUD	CRUD	R*	-	CRUD
Nós	CRUD	CRUD	R*	CRUD	CRUD
Utilizadores	CRUD	RUD	R*	-	-
Amostras	CRUD	R	R*	-	C
QR Codes	CRUD	R	-	CRUD	-

Tabela 3.2: Operações possíveis sobre os modelos do sistema associadas a cada tipo de utilizador. *Acesso apenas aos elementos associados ao utilizador

3.2.3 Base de Dados

Numa análise *bottom-up* da arquitetura, a camada mais baixa representa a base de dados. Esta camada fornece ao sistema uma fonte de dados persistentes permitindo armazenar e consultar os dados de forma rápida e segura. A *Framework Yii* contém uma camada *DAO* para realizar a conexão à base de dados, desta forma é possível abstrair ligação entre a base de dados e a camada superior, composta pelos serviços *Web* e o *Website*. Assim, para implementação desta camada, o sistema de gestão da base de dados*(SGBD) MySQL* apresenta-se como uma boa solução de implementação devido à sua vasta gama de funcionalidades na gestão de base de dados relacionais, mais precisamente, na manipulação de relações entre tabelas, evitando inconsistências de dados através da utilização de

foreign keys(chaves estrangeiras). Esta funcionalidade está presente no *MySQL* desde a versão 5.5 com a introdução do *InnoDB Storage Engine*, permitindo maior flexibilidade e desempenho nas pesquisas, através da utilização de relações entre as tabelas recorrendo a chaves estrangeiras.

O diagrama relacional simplificado da base de dados está representado na figura 3.7. Como pode ser observado, a tabela *WSN* contém os atributos da rede, como a sua localização e número de telefone. Esta tabela tem uma relação *many-to-many*(muitos para muitos) com a tabela utilizadores, ou seja, uma *WSN* pode pertencer a muitos utilizadores, enquanto que um utilizador pode estar associado a diversas *WSNs*. De forma a manter o sistema seguro, foram criadas restrições no acesso a funcionalidades, por isso, a tabela *nível de autenticação* contém os diferentes tipos de acesso, sendo que cada utilizador possui o respetivo nível.

Como referido na secção 3.2.1, cada nó deve conter uma *TAG QR CODE* com informação sobre o nó de forma a agilizar a sua identificação. Esta relação está representada entre as tabelas *Nó* e *QR Code*.

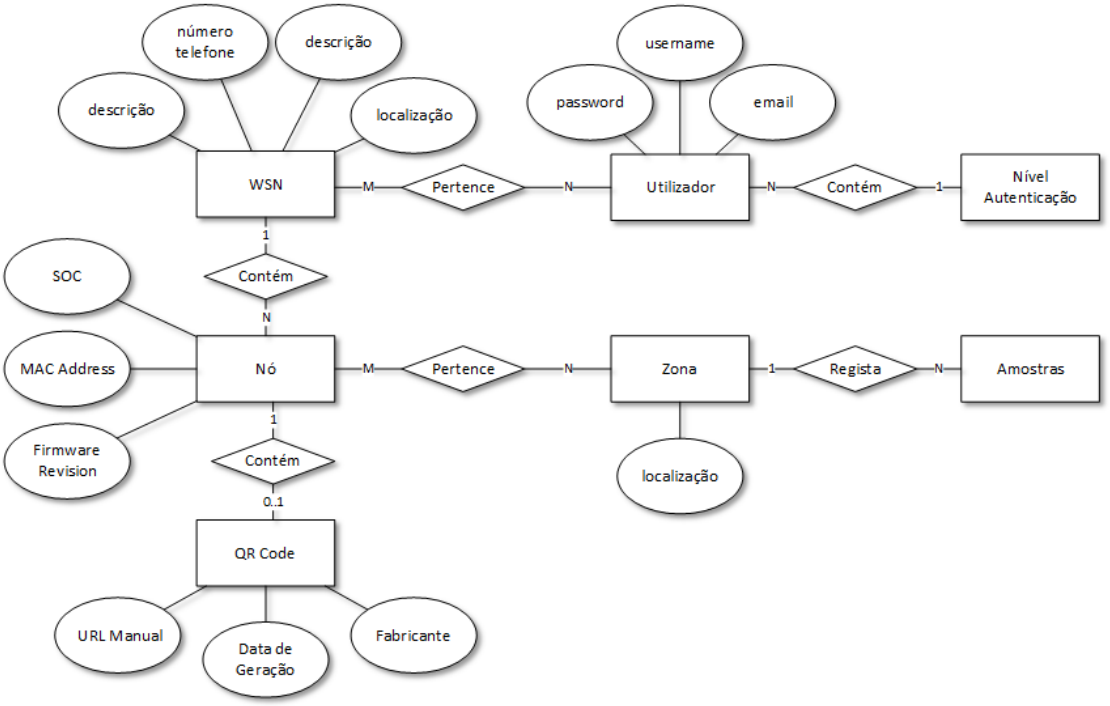


Figura 3.7: Diagrama relacional simplificado da base de dados do sistema na notação de Chen.

3.2.4 Website - GUI de Manutenção e Gestão

A aplicação *Web* representa uma GUI para acesso rápido ao conteúdo e parâmetros do sistema, de modo a permitir uma gestão eficaz. Como pode ser visualizado no diagrama da figura 3.5, o *Website* deve permitir realizar operações *CRUD* sobre os modelos existentes, através do acesso à base de dados do sistema.

O *Website* deve conter um conjunto de funcionalidades de modo a cumprir todos os requisitos propostos. Assim, de seguida são apresentadas as componentes necessárias a implementação do sistema:

- Autenticação: Nesta funcionalidade, deve ser fornecida um menu que permita ao utilizador realizar o *Login* com o seu *username* e *password*.
- Geração de *QR Codes*: O *Website* deve permitir a geração de códigos *QR* com informações relativas ao nó, com possibilidade de exportação para ficheiro *PDF*.
- Operações *CRUD*: O utilizador tem ao seu dispor um conjunto de operações *CRUD* sobre os modelos associados ao sistema, entre eles os utilizadores, *WSNs*, nós, *Logs* e as *TAGs*.
- Alteração de estado de instalação e de ativação da *WSN* e nós.

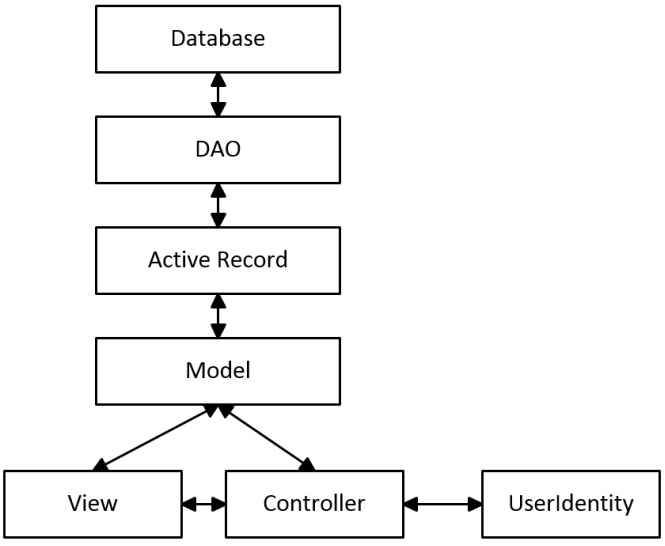


Figura 3.8: Diagrama de blocos do *Website*

A *Yii* é uma excelente *framework* para desenvolvimento de aplicações *Web* utilizando a linguagem *PHP*. Tem diversas vantagens em relação às restantes *frameworks* devido à implementação de diversos *patterns*, tais como *MVC*, *DAO* e

o *ORM*. Sendo que uma das maiores vantagens deve-se à grande comunidade de *developers*, o que se traduz numa excelente documentação e inúmeras extensões e correções de *Bugs*.

O *pattern MVC* é uma excelente arquitetura para desenvolvimento de aplicações *Web* de modo a realizar uma melhor estruturação do código. Este conceito visa a separação do código da interface gráfica do código da lógica de negócio e de controlo da aplicação. A *framework Yii* implementa este conceito de modo a facilitar a estruturação do código por parte do *developer*. Por isso, o *Website* terá um conjunto de modelos que representam as tabelas existentes na base de dados. De modo a otimizar a implementação destas classes, a *framework* utiliza a técnica de mapeamento de objetos relacionais (ORM), a classe *Active Record*, 3.9.

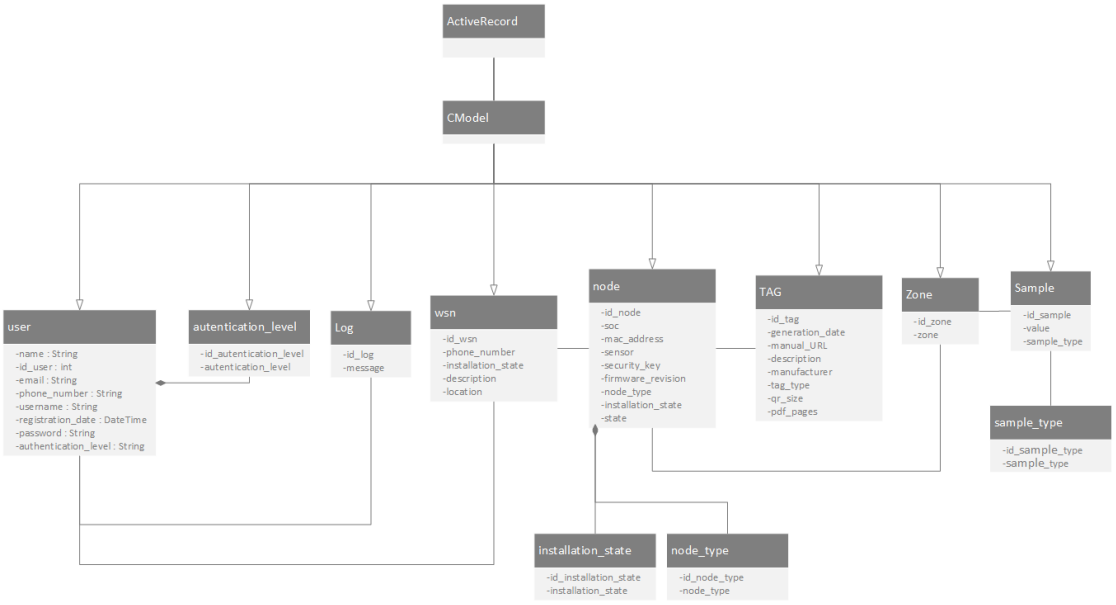


Figura 3.9: Diagrama de classes

Como especificado na documentação da [35], cada classe modelo herda a classe *Active Record*. Esta classe implementa a técnica de mapeamento relacional de objetos(ORM) sendo as operações *CRUD* são implementadas como métodos desta classe, figura 3.9. Como cada classe modelo representa um tabela na base de dados, estas contêm entidades relacionais. Através do *AR* é possível representar estas relações nas classes que herdam da classe *Active Record*. Para cada modelo do sistema (utilizadores, *WSNs*, nós, *Logs* e *TAGs*) é necessário implementar um modelo, um controlador com as ações associadas às operações *CRUD* e as diversas interfaces para cada ação, figura 3.10.

Os controladores de cada modelo contém várias ações correspondentes às operações *CRUD* especificadas, figura 3.11. Desta forma são definidas cinco ações

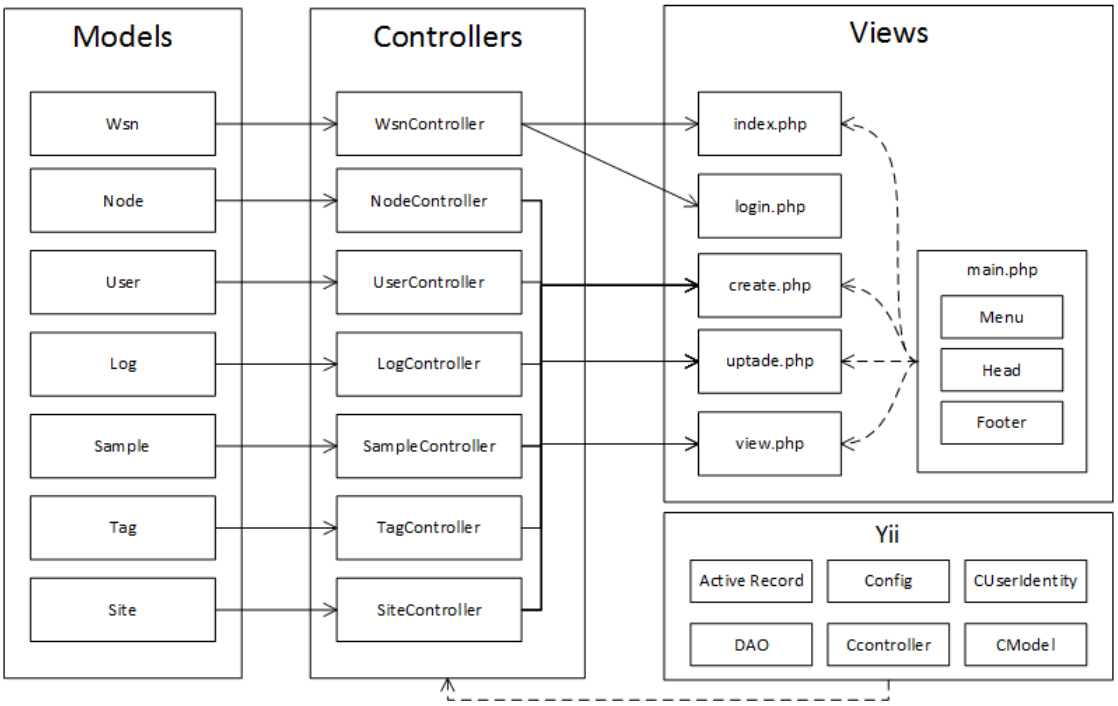


Figura 3.10: Diagrama do *Website* utilizando o *pattern MVC*

padrão:

- *ActionView*: Permite a visualização das propriedades de uma instância de um modelo.
- *ActionList*: Listagem de todas as instâncias de um modelo
- *ActionUpdate*: Ação de edição de uma instância.
- *ActionDelete*: Funcionalidade que permite remover uma instância de um modelo.
- *ActionCreate*: Ação que permite criar uma instância.

A *framework Yii* permite definir as regras de acesso a cada ação presente no controlador por parte de cada utilizador, através da função *accessRules*. Assim, neste controlador é possível implementar as regras especificadas na tabela 3.2.

A geração de *QR Codes* é uma das principais funcionalidades do *Website*. O código deve ser composto pelas informações presentas na tabela *TAG* da base de dados, tendo como um dos atributos o *ID* do nó. Desta forma é possível identificar o nó e recolher informação através dos serviços *Web*. Após a geração do código, a aplicação *Web* deve permitir a exportação dos códigos para ficheiros *PDF* com personalização do número de páginas e tamanho do *QR Code*. Na figura 3.12 está

presente o conteúdo de um código *QR* necessário para identificação do nó.

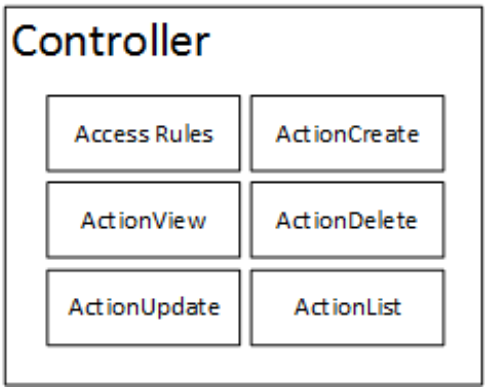


Figura 3.11: Representação do conjunto de funções presentes num controlador.

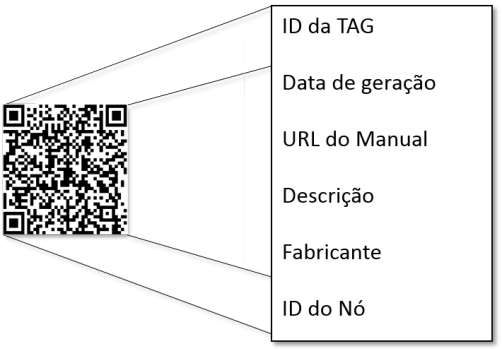


Figura 3.12: Conteúdo de um QR Code

3.2.4.1 Estrutura e Navegação

A estrutura do *layout* do *Website* é definida por três componentes: a *Header*, o *Body* e o *Footer*. A *Header* pode ser definido como o cabeçalho da página. Aqui deve ser implementado o *menu* de navegação onde é possível aceder aos *links* para as funcionalidades da aplicação *Web*. Nas duas primeiras representações de páginas da figura 3.13 pode ser visualizado o *menu* de navegação, parte superior, definido antes da autenticação do utilizador. Este é composto por *links* para as páginas principal, *login*, contatos e instruções de utilização. Na última componente da figura 3.13 está representado o *menu* após a autenticação, por isso, estão visíveis as ligações para as principais funcionalidades. A componente do *menu Management* é definido por uma *drop down list* que contém os acessos através de *URLs* às funcionalidades que permitem realizar operações *CRUD* aos modelos do sistema. Por outro lado, o *Footer* é composto pelos logotipos das instituições associadas ao projeto, a empresa Pinto Brasil e a Universidade do Minho.

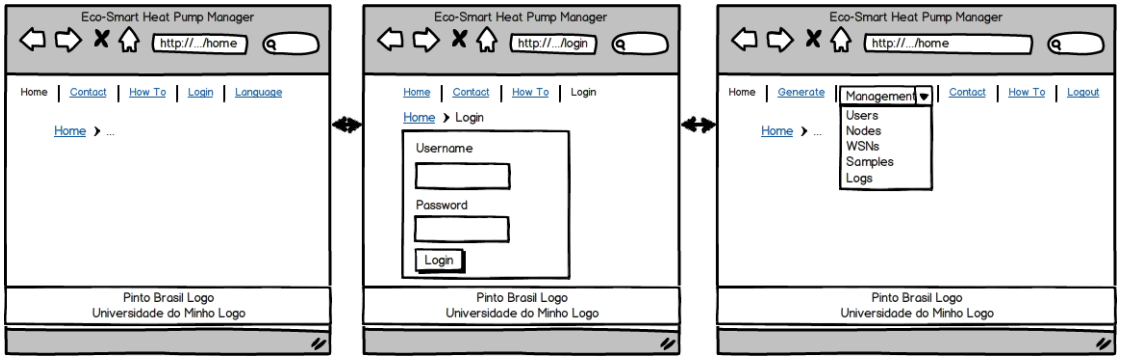


Figura 3.13: *Mockups* da estrutura das páginas do *Website* antes e depois da autenticação do utilizador

3.2.5 Serviços Web

Os serviços *Web* são uma ótima funcionalidade para realizar a interface *machine-to-machine*[18]. O padrão arquitetural *REST* é uma excelente implementação deste conceito devido à sua flexibilidade, desempenho e no aproveitamento das funcionalidades do protocolo *HTTP*, como foi analisado no Capítulo 2. De maneira a fornecer à arquitetura do sistema uma forma de comunicação entre as aplicações cliente e a base de dados, foi desenvolvido um conjunto de serviços *Web* utilizando o protocolo *REST*. Desta forma, é possível realizar operações *CRUD* sobre os recursos da fonte de dados persistentes, figura 3.14.

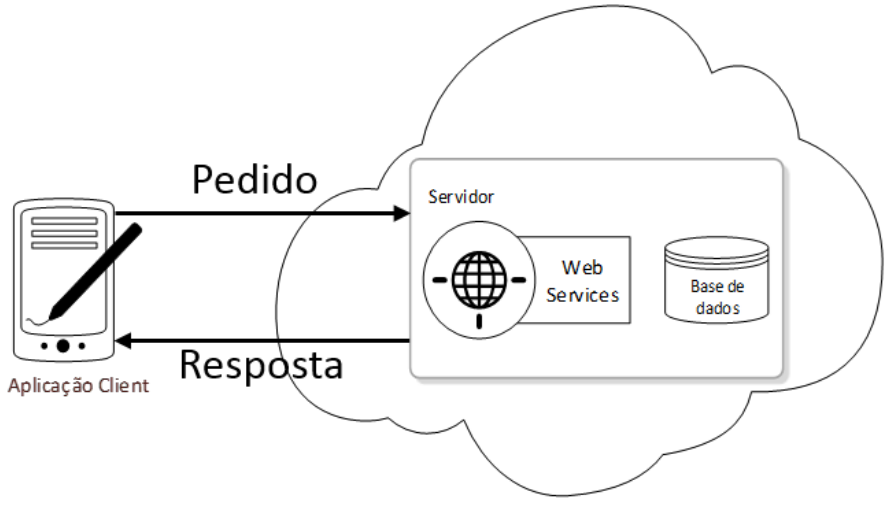


Figura 3.14: Representação da interface entre a aplicação cliente e a base de dados utilizando serviços Web

Assim, foram definidas cinco ações gerais para o conjunto de serviços *Web* para cada tipo de recurso, utilizando quatro métodos *HTTP*, *GET, POST, UP-*

DATE e DELETE. Os serviços fornecem ações ao cliente que consistem em criar, editar, apagar, ver e listar recursos da base de dados. São compostos por uma *URI* de forma a serem identificados pelo utilizador e vários *headers*, definidos por um conjunto de parâmetros:

- *Status* - Contém o tipo de resposta do servidor, sendo indicado através dos *HTTP status Codes*[36].

- *Content-type* - Indica o tipo da estrutura de dados da informação utilizada no serviço, por exemplo, para *JSON*(*application/json*) ou *XML(application/XML)*.

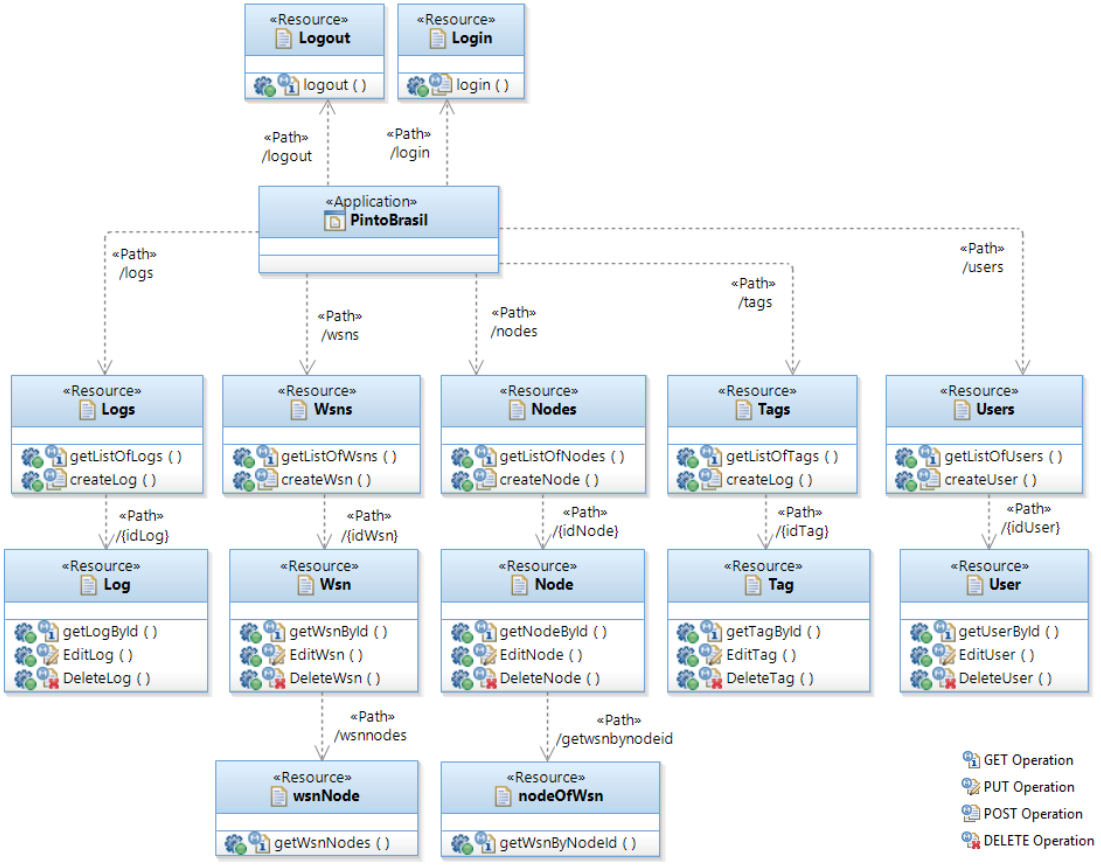


Figura 3.15: Diagrama de especificação dos serviços *REST*

A tabela 3.3 contém os cinco serviços que podem ser utilizados para cada recurso da base de dados.

URI	Método	Ação	Content-type
/recurso	GET	Listar	JSON
/recurso	POST	Criar	JSON
/recurso/{idRecurso}	PUT	Editar	JSON
/recurso/{idRecurso}	DELETE	Apagar	JSON
/recurso/{idRecurso}	GET	Ver	JSON
/wsn/{idWsn}/wsnnodes	GET	Ver Nós da WSN	JSON
/node/{idNode}/wsnbynodeid	GET	Ver WSN do Nó	JSON
/login	POST	Autenticação	JSON
/logout	GET	Fechar Sessão	JSON

Tabela 3.3: Tabela de ações generalizadas dos serviços *Web* utilizando os métodos *HTTP*

O serviço de listagem é identificado através da *URI "/recurso/list"* e do método *GET*. Quando este pedido é executado com sucesso, devolve o código "200", que corresponde ao estado *OK* e um *array* de recursos na estrutura de dados *JSON*. A interface entre o cliente e a fonte de dados persistentes realizada pelo serviço *Web*, está representada no diagrama sequencial da figura 3.16.

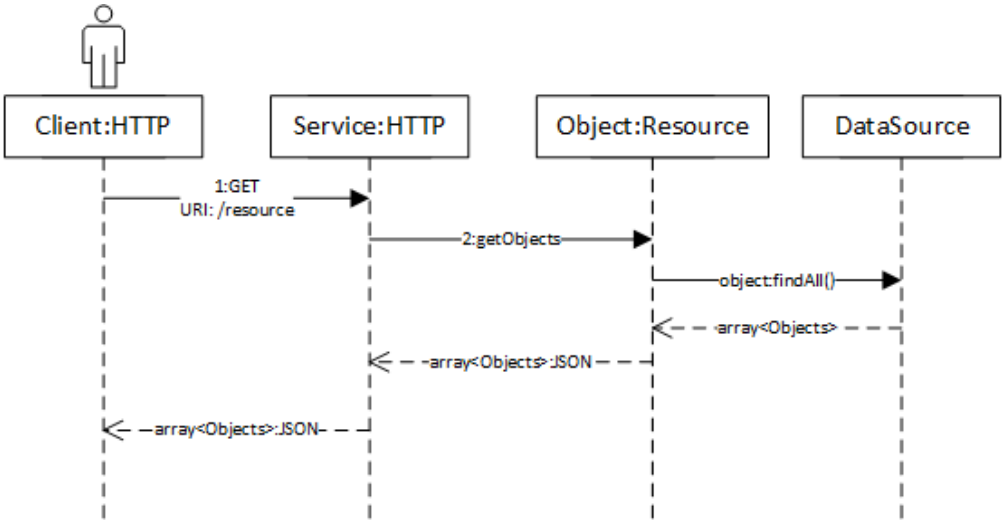


Figura 3.16: Diagrama Sequencial com representação de um pedido *REST* para listagem de recursos.

A pesquisa de um recurso é semelhante ao serviço anterior, apenas na *URI* é passado o seu *ID* como parâmetro e a ação de "list", sendo posteriormente devolvido como resposta um objeto *JSON* com o recursos requerido, figura 3.17.

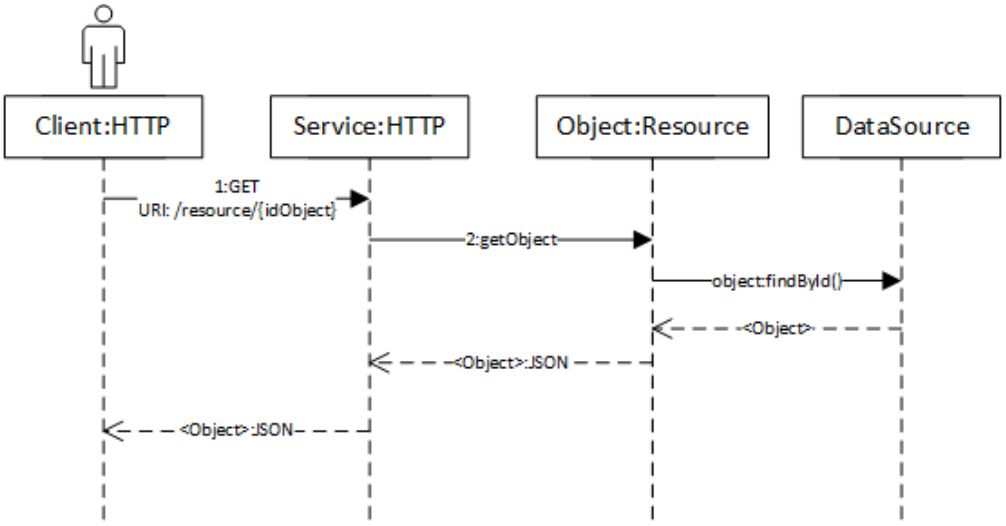


Figura 3.17: Diagrama Sequencial com representação de um pedido *REST* para pesquisa de uma instância de um recurso.

Para remover uma instância também feita a passagem do *ID* como parâmetro na *URI*, sendo executado o pedido *HTTP* com o método *DELETE*, figura 3.18.

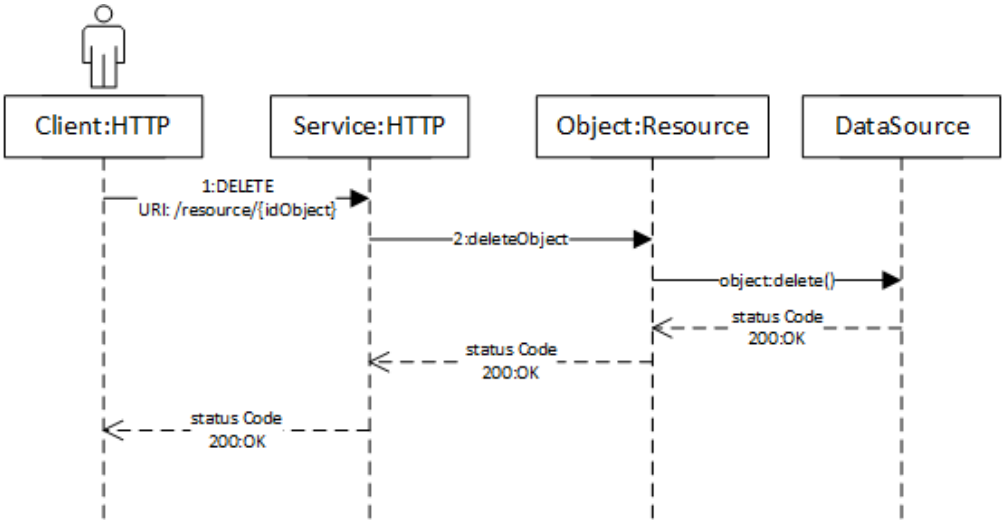


Figura 3.18: Diagrama Sequencial com representação de um pedido *REST* que permite remover uma instância de um recurso.

Da mesma forma, o serviço de edição utiliza a *URI* "*/recurso/update/idRecurso*", sendo necessário o método *PUT* e o envio de um objeto *JSON* com os atributos do recurso, figura 3.19.

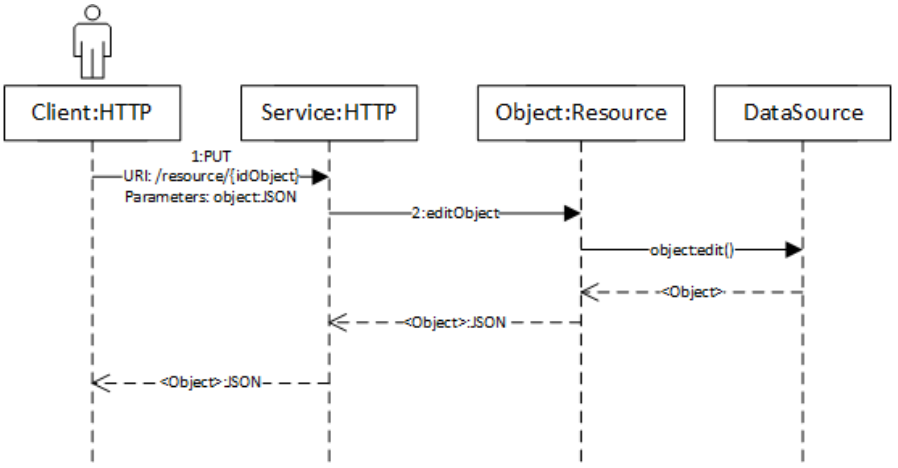


Figura 3.19: Diagrama Sequencial com representação de um pedido *REST* para edição de uma instância de um recurso.

Para a utilização do serviço que permite criar um recurso, é necessário realizar um pedido *HTTP* com o método *POST* e passar uma componente *JSON* com os atributos do recurso.

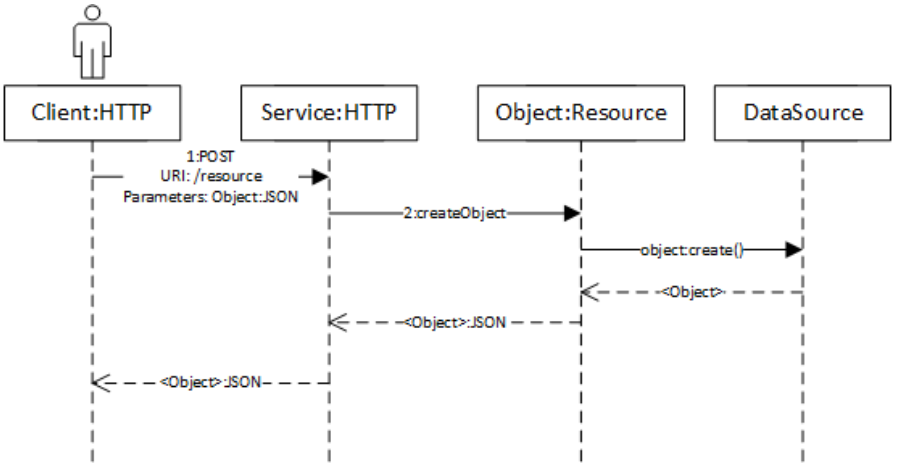


Figura 3.20: Diagrama Sequencial com representação de um pedido *REST* para criação de uma instância de um recurso.

O serviço identificado pela *URI* "*/wsn/{idWsn}/wsnnodes*" fornece à aplicação cliente uma via para obtenção de todos os nós de uma determinada rede. Ainda, foi desenvolvido um serviço que permite fornecer um objeto *JSON* com os atributos de uma *WSN* através do *ID* de um nó que a compõe.

O serviço de *login* deve permitir a autenticação do utilizador ao nível do servidor de forma a aumentar a segurança no acesso aos dados. Por outro lado, para fechar a sessão do utilizador, é usado o serviço de *logout*.

A framework *Yii* possui excelentes funcionalidades para implementação de serviços *Web* em *PHP* no protocolo *REST*. Através do seu *DAO* é possível estabelecer uma conexão à base de dados de forma a proceder às ações requisitadas. Estas ações serão geridas por um controlador, especificado na aplicação da framework, que com o auxílio da funcionalidade de gestão de *URLs* da *Yii*, permite redirecionar o pedido para a respetiva ação definida. Cada uma processa os parâmetros recebidos pelos pedidos *HTTP* através dos modelos mapeados das tabelas da base de dados, através do *ORM* da *framework*, e executam as funções associadas a cada serviço devolvendo os recursos correspondentes a cada serviço em objetos *JSON*.

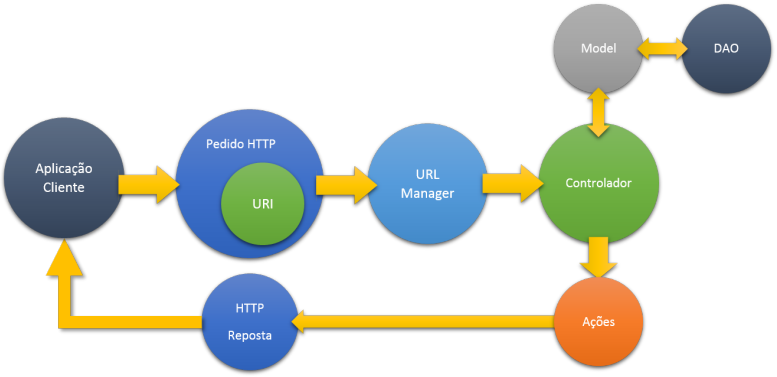


Figura 3.21: Fluxo de execução da Arquitetura dos serviços utilizando as funcionalidades da *Framework Yii*

Em cada função associada ao serviço, é verificada a autenticação do utilizador, ou seja, se existe uma sessão ativa no servidor do mesmo e de seguida é verificado o método *HTTP* correspondente ao ao serviço. Caso os parâmetros se encontrarem errados, serão enviados como resposta os respectivos erros, figura 3.22.

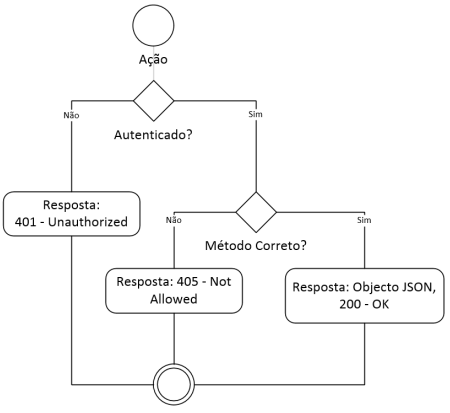


Figura 3.22: Representação de um pedido de Serviços *REST*

3.2.6 Aplicação Android - GUI de instalação

Os *smartphones* e *tablets* são excelentes dispositivos para interação com diversos dispositivos devido aos seus inúmeros periféricos, tais como *Bluetooth*, *Wi-fi*, *NFC*, câmera entre outros. A aplicação *Android* da presente dissertação tem como objetivo interagir com os nós da *WSN* através da leitura de *QR Codes* gerados com a informação sobre estes utilizando a câmera do *smartphone*. Este processo está representado na figura 3.23. Neste processo, o instalador utiliza a aplicação para a leitura do *TAG QR Code* e de seguida a aplicação requisita informação sobre o nó com recurso aos serviços *Web*, figura 3.23.

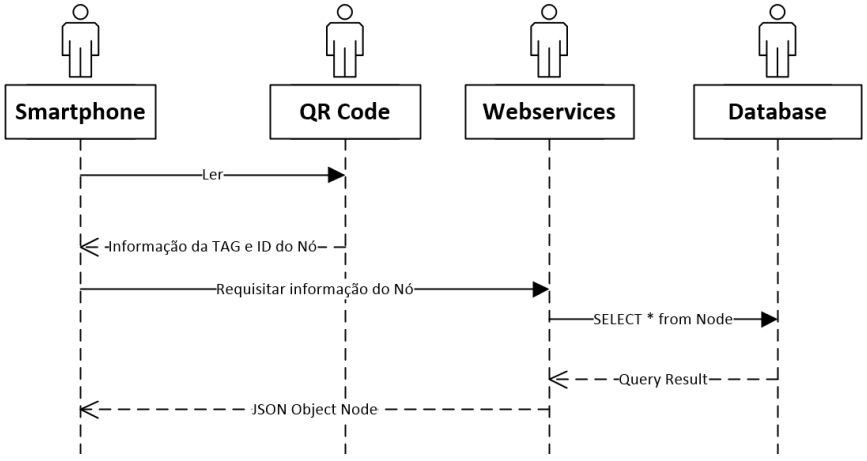


Figura 3.23: Diagrama sequencial do processo de leitura de um código *QR* e utilização dos pedidos *Web* para aquisição do respetivo nó

Na aplicação *Android* também se pode aplicar o *pattern MVC*, desta forma é possível separar o código da interface, da lógica de negócio e dos modelos. No desenvolvimento de uma aplicação *Android*, as interfaces são implementadas através da estrutura de dados *XML*. Os modelos da presente aplicação representam as tabelas da base de dados e são definidos por classes *Bean*[1]. As *Activities* da aplicação podem ser definidos como os controladores, uma vez que nestas classes está presente o processamento, onde são utilizados os modelos e as interfaces do sistema. As *Activities* e os modelos da aplicação estão definidos no diagrama de classes da figura 3.25.

[1]Em Java um Bean é uma classe que contém todos os seus atributos privados, mas que podem ser acedidos através dos seus únicos métodos *get* e *set* públicos.

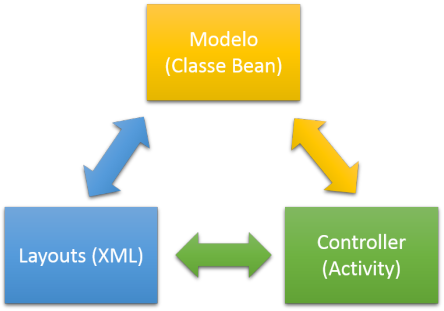


Figura 3.24: Diagrama do *Pattern MVC* na aplicação *Android*

As classes *Activity* herdam da classe *Activity* todos os métodos necessários para a criação e controlo da interface, estando associados à classe *R*, de forma a obterem acesso aos identificadores dos objetos especificados nas interfaces.

A primeira interface apresentada na aplicação é composta por um menu de autenticação, onde é necessário introduzir o nome de utilizador e a palavra-passe. De seguida é apresentado o *Menu* contendo as botões para acesso às funcionalidades da aplicação.

A primeira funcionalidade é representada pela classe *ScanActivity*, onde é implementado o processo de leitura de *QR Codes* e consequente processamento da informação recolhida.

O conjunto de *Activities*, *ViewNode*, *ViewWsn* e *ViewLog*, deve permitir a visualização das instâncias dos respetivos modelos existentes na base de dados. Também devem conter funcionalidades de edição e remoção destes componentes. A descrição da sequência de execução das atividades está presenta no diagrama da figura 3.26.

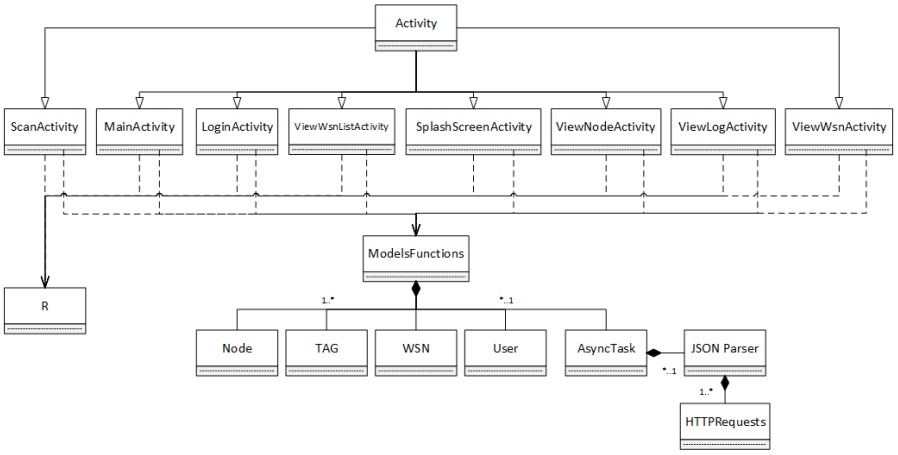


Figura 3.25: Diagrama de classes da aplicação *Android*

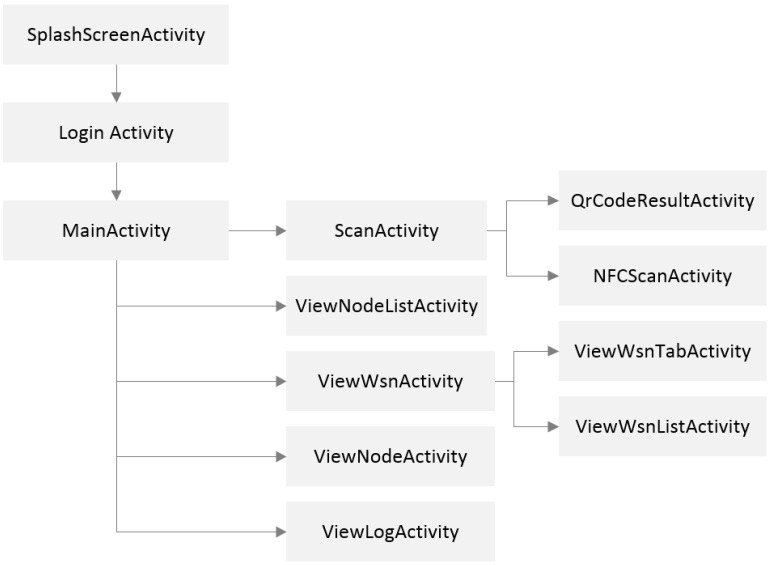


Figura 3.26: Fluxo de *Activities* na aplicação *Android*

Ferramentas de desenvolvimento

As aplicações para a plataforma *Android* são desenvolvidas com recurso a um conjunto de funcionalidades presentes na *framework Android SKD*. Este *SDK* é composto por o *Android Virtual Device (AVD)* e o *Android Developer Tools (ADT)*, simulador e ferramentas de desenvolvimento respetivamente. O *Eclipse* é o *IDE* mais comum entre a comunidade de *developers*. Através da inclusão do *Plugin ADT* é possível desenvolver aplicação para *Android* nesta plataforma. Para simulação e teste de aplicações pode ser usado um *smartphone, tablet* ou o emulador com a plataforma *Android* usando o *AVD*[13].

3.2.7 Processo de Instalação

O processo de instalação da rede de sensores sem fios é definido por duas componentes: a primeira é constituída pelo serviço de criação e ativação da *WSN* e nós por parte do nó coordenador da rede (*CGB*). A segunda componente é realizada pelo instalador e é composta pela leitura da informação dos nós com a aplicação *Android* e consequente verificação da correção do sistema e modificação do estado de instalação dos nós.

No início do processo de instalação, o utilizador "instalador" deve criar uma entrada da *WSN* que pretende instalar através do *Website* ou da aplicação *Android*. De seguida deve associar o nó coordenador à rede previamente criada e definir o

seu estado como "instalado", também utilizando uma das aplicações cliente. Após a ativação da rede, o nó coordenador deve mudar o seu estado para "Ativo" através dos serviços *Web*. O diagrama sequencial da figura 3.27 apresenta esta descrição.

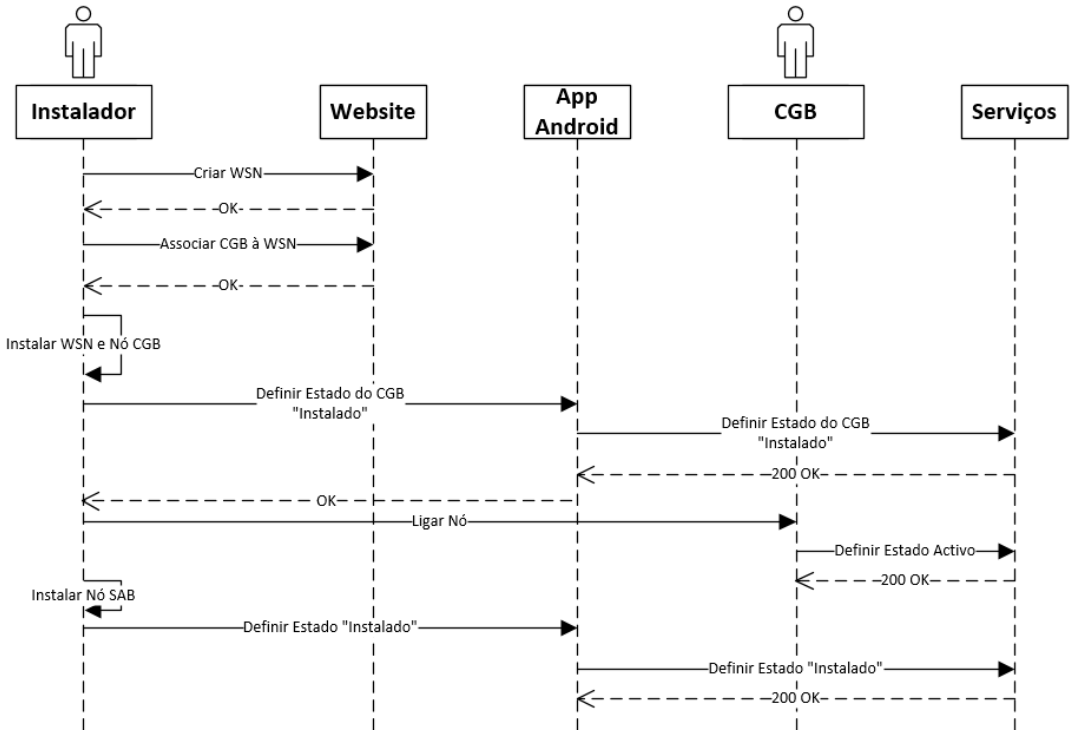


Figura 3.27: Diagrama sequencial do processo de instação de uma *WSN* e respectivos nós

O processo de instalação de um nó do tipo *Sensor Actuator Board* (SAB) na base de dados do sistema pode ser realizado pelo instalador ou mesmo pelo próprio *CGB*. O primeiro caso acontece quando o instalador associa o nó *SAB* à WSN, sendo definido o seu estado de instalação como "instalado" e estado do nó "desativado". Após a ativação do nó, este é identificado pelo *CGB* e associado à *WSN*. O segundo caso acontece quando o nó é identificado pelo *CGB*, mas não se encontra associado à *WSN*. Nesta situação, o nó coordenador tem a tarefa de associar o nó *SAB* à rede e definir o seu estado como "ativo", figura 3.29. O endereço *MAC* de cada nó permite a sua identificação por parte do nó coordenador e pelos utilizadores da *WSN*.

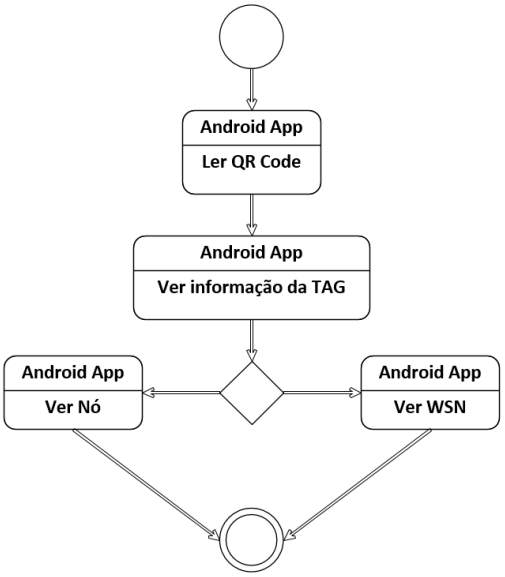


Figura 3.28: Processo de Leitura de um código QR e opções disponíveis

A identificação do nó por parte do instalador é feita através da utilização da funcionalidade de leitura dos *QR Codes*. Como o código contém o *ID* do nó, é possível selecioná-lo corretamente de modo a executar o pedido do serviço *Web* que devolve a respetiva instancia do nó e as componentes associados. Após a leitura do código *QR*, a aplicação de instalação deve disponibilizar opções de visualização da informação e de edição das propriedades do nó e também da *WSN* onde está instalado, figura 3.28.

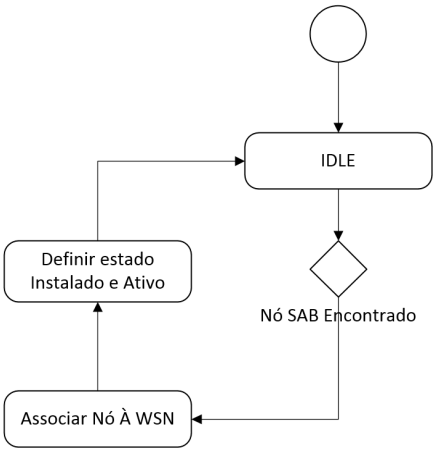


Figura 3.29: Diagrama assíncrono de deteção de nó SAB

Capítulo 4

Implementação

Neste capítulo é descrita a implementação dos componentes do sistema especificados no capítulo anterior. O capítulo anterior permitiu especificar os módulos do sistema de modo a compreender a arquitetura do sistema e as suas funcionalidades. Assim, no capítulo 3, foi definido modelo da base de dados, foram especificados os serviços *Web* para realizar interface entre a base de dados e as aplicações cliente. Também foram definidas as estruturas e funcionalidades necessárias para a implementação do *Website* e da aplicação *Android*. Por último foi especificado o processo de instalação da *WSN* utilizando as *GUIs* desenvolvidas nesta dissertação. Assim, neste capítulo é apresentado o trabalho desenvolvido na implementação das *GUIs* de instalação e manutenção do projeto *Eco-smart Heat Pump*. A apresentação dos módulos segue a sequência apresentada no capítulo anterior, sendo ainda descrita a implementação dos conceitos e funcionalidades mais relevantes da presente dissertação.

Processo de implementação

A implementação do sistema foi dividida em três fazes distintas. A primeira foi definida pelo desenvolvimento da base de dados e implementação dos serviços *Web*. A implementação das aplicações cliente, aplicação *Android* e *Website*, foi realizada durante a segunda fase de desenvolvimento. Nestas fases, os serviços *Web* e a base de dados foram desenvolvidos e instalados num servidor local, instalado na máquina de desenvolvimento.

Na última fase, foi configurado um servidor remoto instalado numa máquina, figura 4.1, fornecida pelos dos serviços *Web* de computação em nuvem da empresa

Amazon[37], de forma testar o sistema remotamente.

4.1 Arquitetura

Como foi especificado no capítulo anterior, o sistema é composto por duas aplicações cliente para instalação, monitorização e gestão do sistema, e um conjunto de serviços *Web* para realizar a interface entre as aplicações e a base de dados. Como pode ser visualizado na figura 4.1, é utilizada uma instância de uma máquina virtual fornecida pela *Amazon Web Services* para hospedar a base de dados, os serviços *Web* e o *Website*. Neste sistema foi instalado um servidor *MySQL* para fornecer suporte à base de dados. Os serviços *Web* e o *Website* foram desenvolvidos maioritariamente na linguagem *PHP*, para ser possível executá-lo, foi instalado na máquina um servidor *Apache*.

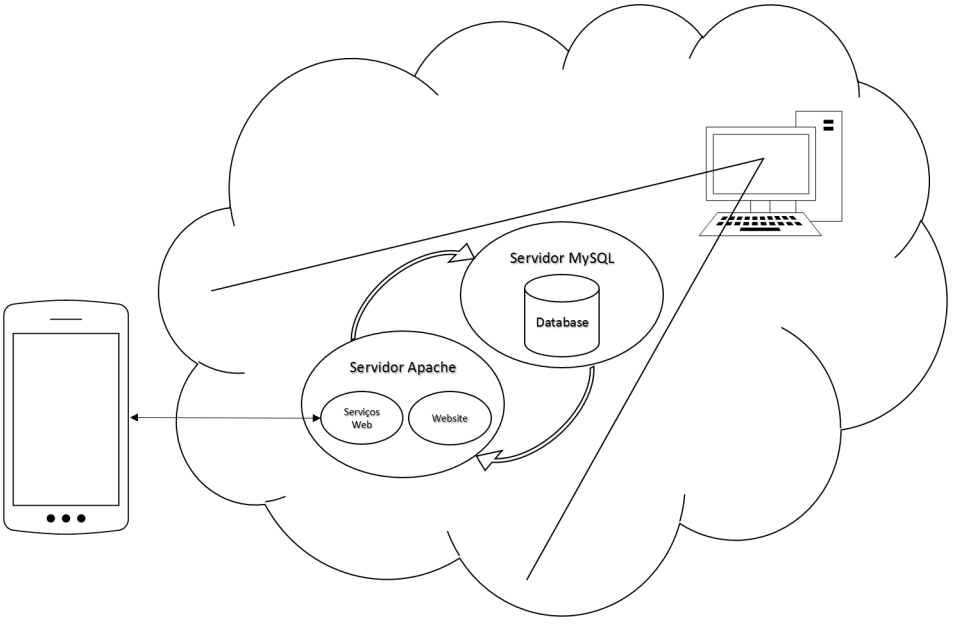


Figura 4.1: Arquitetura do sistema: servidores e aplicações

4.2 Base de Dados

A base de dados foi implementada com auxílio da ferramenta de modelação *MySQL Workbench 6.0* [38]. Desta forma, foi possível desenvolver o modelo da base de dados através da inclusão das tabelas e respetivas relações, figura 4.2 e consequente geração do código *MySQL* para implementação desta fonte de dados

persistentes.

As tabelas da secção da base de dados delimitada pela caixa a vermelho implementam a funcionalidade de armazenamento de utilizadores e níveis de autenticação. Assim, a tabela *user* representa o modelo de utilizadores, onde estão contidos as suas propriedades, tais como o nome, *username* e *password*. Este modelo também contem o atributo *authentication_level* que representa os tipos de utilizador e os vários níveis de permissões que têm no sistema, como foi especificado na secção 3.2.2.

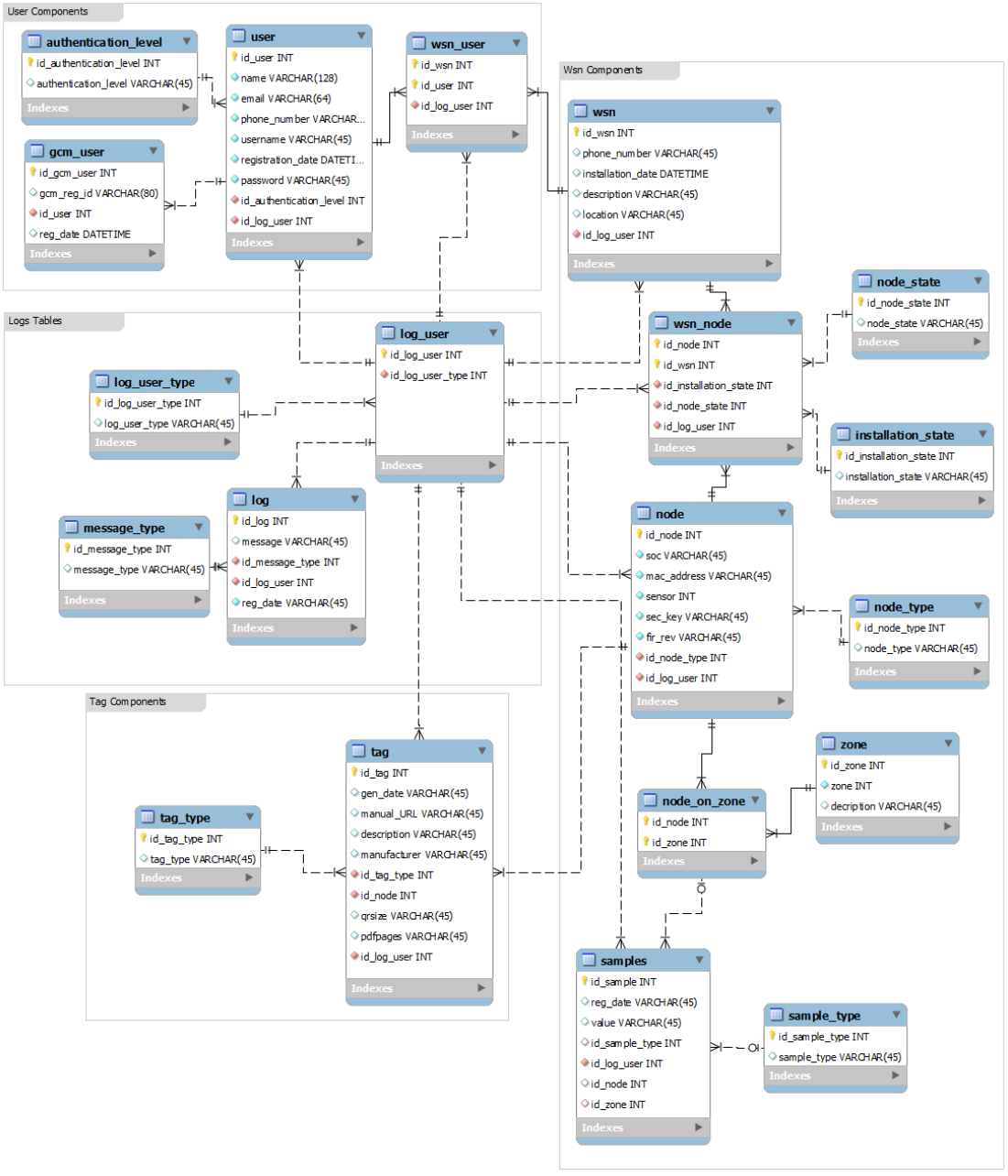


Figura 4.2: Diagrama de entidades e relacionamentos da Base de Dados

As relações entre as tabelas *user, wsn_user e wsn* representam a associação dos utilizadores às *WSNs*. Assim, foi definido uma relação de muitos para muitos entre as tabelas *wsn* e *user* onde um utilizador pode estar associado a várias *WSNs* e várias *WSNs* pertencem a um utilizador. De forma evitar a utilização do relacionamento *m:m*, foi criada uma tabela intermédia de junção *wsn_user* composta pelas duas chaves primárias (*foreign keys*) das duas tabelas, resultado em dois relacionamentos *1:m*. O conteúdo da caixa verde no diagrama de entidades e relacionamentos da figura 4.2 corresponde ao grupo de tabelas das *WSNs* e respetivos componentes. Neste conjunto, a tabela correspondente aos nós associa-se às *WSNs* num relacionamento *m:m*. Também associadas a cada instancia dos nós, são utilizadas tabelas estáticas com os diversos estados de instalação, ativação e tipo (tabela 4.1).

Ativação	Instalação	Tipo
Ativo	Instalado	CGB
Desativado	Não Instalado	SAB
		MPB

Tabela 4.1: Tabela de permissões de utilização para as aplicações do sistema

A arquitetura do sistema *Eco-Smart Heat Pump* define os vários nós *SAB* estão distribuídos por cinco zonas do local de instalação da *WSN*. Para isso, foi criada a tabela *zone* e a tabela de junção *node_on_zone* para representar esta componente do sistema. Como o nome dos atributos da tabela *zone* indicam, cada zona é composta pelo número de zona correspondente e a respetiva descrição. Os valores de temperatura e humidade amostrados pelos nós da *WSN* são armazenados pela tabela *samples*, sendo que para cada registo foram definidos atributos como a data da amostra, o respetivo valor, o nó e zona associados e ainda o tipo, representado na tabela *sample_type*.

As *TAGs* permitem a rápida identificação dos nós da *WSN* através da informação nelas contida. Esta informação está representada na base de dados através da tabela *tag* e dos seus atributos. Como foi definido no capítulo 3, esta tabela contém a data de geração da *TAG*, a *URL* do manual do nó e sensores associados, o fabricante e o tipo de *TAG*. No sistema foram definidos dois tipos de *TAG*: *QR Code* e *NFC*, permitindo assim a utilização de duas tecnologias distintas para a identificação do nó.

No arquitetura também foi definida uma funcionalidade de registo de eventos relevantes do sistema. Desta forma, é possível monitorizar o seu correto funcio-

namento recorrendo ao registo de *Logs* em todas as ações e modificações feitas no sistema, incluindo todas as aplicações clientes e execução de pedidos dos serviços *REST*. Na implementação da base de dados foi definido que cada uma das tabelas mais importantes (*WSN*, *node*, *tag* e *user*) tem um *ID* da tabela *log_user* de maneira a ser possível identificar o grupo de *logs* correspondentes a cada instância. A tabela *log_user* é composta pelo de tipo de *log_user*, tabela 4.2, de forma a ser possível identificar todos os *logs* correspondentes a cada tabela. As tabelas que representam as funções dos *Logs* estão contidas na caixa a verde do diagrama de *ER* da figura 4.2.

Tipo *Log_user*	Tipo de mensagem
WSN	ACTION
Node	ERROR
User	INFORMATION
Tag	WARNING
Wsn_user	STATUS
Wsn_node	CONFIRMATION
Sample	

Tabela 4.2: Tabela de tipo de *log_user*

4.3 Website

A aplicação *Web* permite aos utilizadores da *WSN* gerir e monitorizar todo o sistema num único ponto, com diferentes tipos de acesso às funcionalidades e propriedades, dependendo do nível de autenticação de cada um. Também é possível realizar operações *CRUD* sobre as instâncias das tabelas que representam componentes associados à *WSN*, assim como gerar *QR Codes* com as propriedades dos nós.

O *Website* foi desenvolvido utilizando as funcionalidades da *framework Yii*. A linguagem utilizada para as componentes *Server-side* foi *PHP* 5.4, permitindo definir páginas *Web* dinâmicas e interativas. A componente *frontend* foi desenvolvida através da linguagem *HTML*, mais precisamente a versão cinco e com recurso à linguagem de estilos *CSS3*, de forma a formatar o conteúdo estruturado pelos elementos *HTML* e melhorar o aspecto gráfico das páginas *Web*.

O *Website* desenvolvido é composto por três grandes blocos: a *framework Yii*, a *bootstrap* e a framework *TCPDF* para geração dos códigos *QR*. O primeiro passo do processo de implementação do *Website* é caraterizado pela criação de uma

nova aplicação através da estrutura de ficheiros da *Yii* e posterior modificação do ficheiro de configuração das respetivas propriedades, */protected/config/main.php*. Na figura C.1 do anexo C pode ser visualizada a estrutura de ficheiros de uma aplicação *Web* baseada na *Yii*.

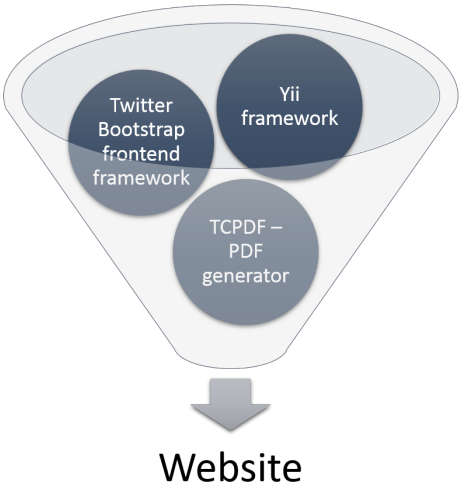


Figura 4.3: Representação das três grandes componentes que compõem o *Website*

A aplicação é composta por três grandes componentes: a *framework Yii*, a *bootstrap* e a *framework TCPDF*. Como definido anteriormente, a estrutura base da aplicação *Web* foi desenvolvida utilizando as componentes da *Yii*.

4.3.1 Bootstrap

A utilização da *Bootstrap* do *Twitter* na aplicação permite obter um aspeto mais agradável e profissional na aplicação, uma vez que é possível customizar cada objeto *HTML* através da definição da respetiva classe padronizada na *bootstrap*. A *bootstrap* é constituída por um conjunto de ficheiros *CSS*, sendo que as suas componentes definem as cores, sombras e estilos dos objetos e componentes *HTML*. Os objetos *HTML* são maioritariamente declarados na forma de *widgets*(4.1) de forma a uniformizar a sua utilização. Assim, foi utilizada uma extensão da *Yii*[39] composta por *widgets* baseados na *bootstrap* do *twitter*[40] de forma a otimizar e facilitar o desenvolvimento de aplicações *Web* baseadas nesta componente. Nas figuras 4.4 e 4.5 é possível visualizar as diferenças na utilização desta *framework*. A configuração é realizada através dos seguintes passos:

- Extrair ficheiros da *bootstrap* para a pasta */protected/extensions*

- Definir o *Path* da *bootstrap* no ficheiro */protected/config/main.php*: *Yii::setPathOfAlias('bootstrap', dirname(__FILE__).'/../bootstrap');*
- Adicionar o componente no ficheiro */protected/config/main.php*: *'bootstrap'=>array('class'=>'bootstrap.components.Bootstrap')*
- Definir os ficheiros *CSS* no *HEAD* da aplicação situado no ficheiro */protected/view/layouts/main.php*

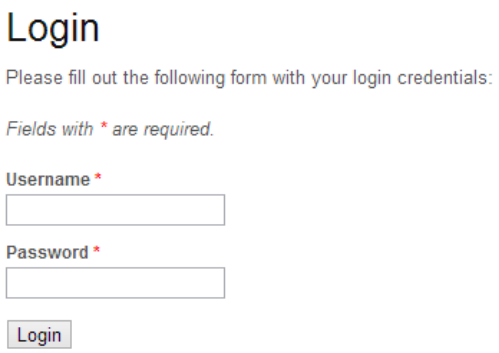


Figura 4.4: *Login Form* sem *bootstrap*

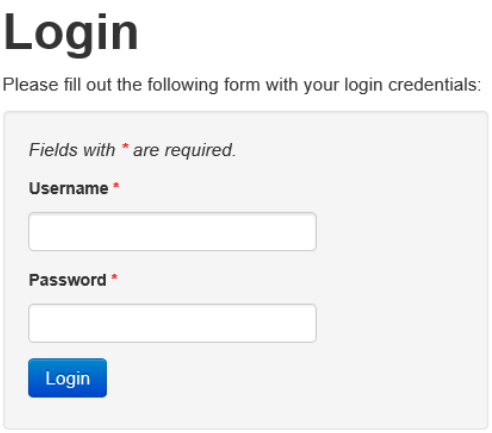


Figura 4.5: *Login Form* com *bootstrap*

Listagem 4.1: Declaração de um botão sob a forma de *widget*

```
1 <?php
2     $this->widget('bootstrap.widgets.TbButton',
3         array('buttonType'=>'submit',
4               'label'=>'Login',
5               'type'=>'primary'));
6 ?>
```

4.3.2 Autenticação

Na *Yii*, classe *CUserIdentity* representa uma identidade composta pelos atributos *username* e *password*, implementando o método *authenticate* que permite verificar a correção dos atributos, o método login que inicia uma sessão de autenticação, e o método *logout*. Os objetos de sessão mantêm o estado e a instância de *UserIdentity* nos diversos pedidos *HTTP* nas múltiplas páginas da aplicação *Web*. A *Yii* define que todas as classes que herdem esta classe devem implementar o método *authenticate*. Assim, foi definida uma classe *UserIdentity* que herda de *CUserIdentity* e implementa este método, como pode ser visualizado no código em B.1. A classe *CUserIdentity* permite definir estados persistentes de sessão que po-

dem ser acedidos através do componente *User* da *Yii*, desta forma, foram definidos três:

- *'fullname'*: contém o nome do utilizador
- *'isAdmin'*: contém o nível de autenticação
- *'UserID'*: define o ID do utilizador

Para o utilizador realizar a autenticação foi definido um formulário de *login*, figura 4.5. O modelo associado a esta *view*, *LoginForm*, é composto pela função de *login*, listagem 4.2, e pelos atributos *username e passoword*. Na função é declarada uma instância de *UserIdentity* e após a introdução dos parâmetros de autenticação, é executada a função *login* associada à componente *User* da *Yii*. Após isto, nesta componente é possível realizar operações de *logout* e ações definidas nos controladores que necessitem de autenticação.

Listagem 4.2: Função de *Login*

```
1  <?php
2  public function login()
3  {
4   if($this->_identity===null)
5   {
6    $this->_identity=new UserIdentity($this->username,$this->password);
7    $this->_identity->authenticate();
8   }
9   if($this->_identity->errorCode===UserIdentity::ERROR_NONE)
10  {
11   $duration=$this->rememberMe ? 3600*24*30 : 0; // 30 days
12   Yii::app()->user->login($this->_identity,$duration);
13   return true;
14  }
15  else
16   return false;
17 }
18 ?>
```

4.3.3 *MVC* e operações *CRUD*

No ficheiro de configuração da aplicação, é possível adicionar diversas funcionalidades entre as quais a ligação à base de dados, definições de *URLs* e a componente *Gii*. Maioritariamente, as operações *CRUD* as necessárias para implementar as funcionalidades do *Website* sobre os modelos existentes na base de dados, como pode ser analisado na figura 3.10. A ferramenta *Gii* da *framework Yii*, fornece a possibilidade de geração de código, sendo possível criar o modelo,

controlador e as *views* associados à tabela da base de dados, com as respetivas operações *CRUD* implementadas. As tabelas cujas operações e componentes foram geradas estão presentes na figura 3.10.

Model

No *Website*, as classes *model* utilizam a classe *Active Record* como extensão de forma a permitir a utilização dos seus métodos de *data access*(*find, findAll, delete, remove, etc.*). Assim, estas classes representam as tabelas da base de dados, as suas relações e regras dos respetivos atributos. As regras desta classe definem o tamanho e o tipo dos atributos. Além disso, as relações das tabelas também são definidas nesta classe, sendo que cada regra é composta por um identificador, o tipo de relação, a tabela relacionada e a chave que a constitui. Como pode ser analisado na listagem de código B.2, a função *relations()* contém as relações da tabela *Node*. A primeira relação implementada, representa a conexão desta tabela com a tabela *NodeType*. Assim, esta relação é identificada através de *'idNodeType'* e tem uma relação do tipo *'self :: BELONGS_TO'*.

Controller

O *Controller* é composto por diversas ações e as respetivas regras de acesso para o conjunto de utilizadores. As ações implementadas para cada controlador correspondem às cinco operações *CRUD* definidas na secção 3.2.4. A *Yii* permite definir no ficheiro controlador para cada modelo, o conjunto de utilizadores que possuem acesso às ações. A listagem de código 4.3 contém a implementação das regras de acesso às ações definidas no *NodeController*. Assim, foram definidos três conjuntos de utilizadores:

- *: Utilizadores não autenticados
- @: Utilizadores autenticados
- 'HasSiteCUDPermissions': Utilizadores que contém nível de autenticação que permite realizar operações *CRUD*.

Listagem 4.3: Conjunto de regras de acesso às ações utilizadas em todos os controladores

```
1 public function accessRules()
2 {
3  return array(
```

```
4     array('allow',  /* allow all users to perform 'index' and 'view' actions */
5      'actions'=>array('index','view', 'admin'),
6      'users'=>array('@'),
7     ),
8     array('allow', /* allow authenticated user to perform 'create' and 'update'
          actions */
9      'actions'=>array('create','update', 'deletewsnnode', 'createwsnnode', '
          delete', 'addusertowsn'),
10     'users'=>array('@'),
11     'expression'=>"User::model()->HasSiteCUDPermissions(Yii::app()->user->
          getState('isAdmin'))",
12    ),
13    array('deny',  /* deny all users */
14     'users'=>array('*'),
15    ),
16   );
17  }
```

Menu de navegação

A *Yii* e a *Boostrap* fornecem um menu de navegação que permite aceder às diversas funcionalidades e ações da aplicação, sendo que os *widgets TbNavbar* e *TbMenu* representam este tipo menu. Assim, foi definido um *menu* que permite aceder a todas as operações *CRUD* dos modelos, aceder ao formulário de *login* e função de *logout*, assim como a funcionalidade de geração de códigos *QR*. Como pode ser analisado na figura 4.6 e 4.7, no *menu* de navegação é possível navegar entre as funcionalidades e interfaces que permitem realizar operações sobre as tabelas da base de dados, especificadas no capítulo 3.

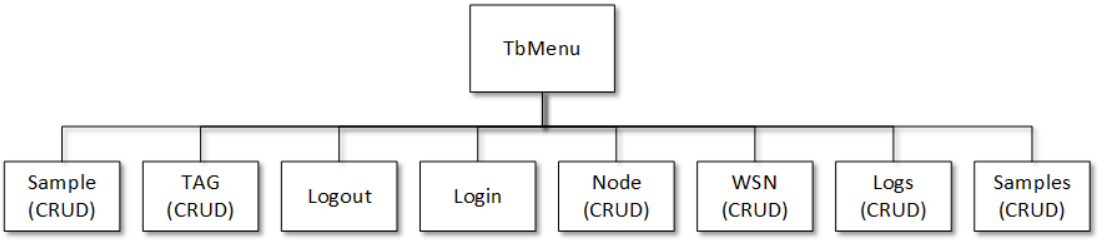


Figura 4.6: Componentes do menu de naveção *Menu* de navegação

4.3.4 Gerador de *QR Code* e *PDF*

A geração de códigos *QR* e consequente exportação para ficheiro *PDF* é uma funcionalidade essencial no processo de instalação da *WSN*. Para realizar esta operação, é utilizada a *framework TCPDF*, desenvolvida em *PHP*. Esta *framework*

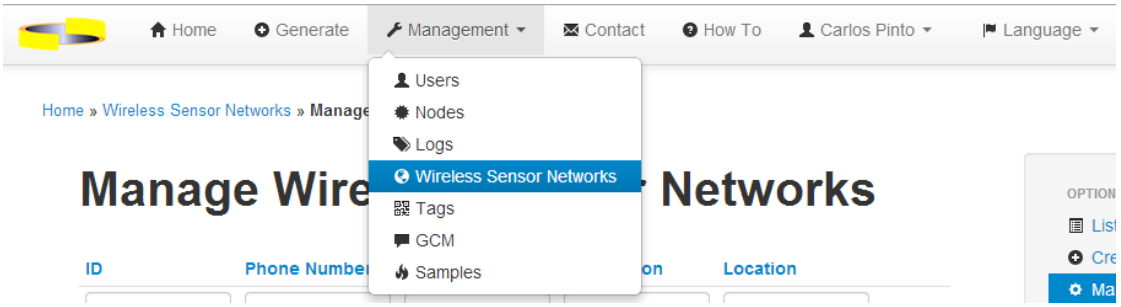


Figura 4.7: Implementação do *menu* de navegação utilizando os *widgets TbNavbar* e *Tbmenu* da extensão da *bootstrap* do *Twitter*.

fornece um conjunto de classes que permitem gerar ficheiros *PDF* e adicionar diversos tipos de conteúdo, entre os quais texto, imagens em diversos formatos, a geração de códigos de barras, etc.

De forma a utilizar foi definida uma função que recebe os parâmetros do código *QR* e as propriedades do *PDF* pelo método *HTTP POST* 4.4. Na função é criada uma instância da classe *TCPDF*. De seguida é adicionado o número de páginas recebidas pelo parâmetro. Para gerar o código *QR* é utilizado o método *write2DBarcode* que recebe como parâmetros a informação a inserir no código, o tamanho e o tipo de código de barras. Neste caso foi utilizado o *QR CODE* com o nível de correção de erros *H*.

Listagem 4.4: Funcao de geracao de codigos *QR* e ficheiros *PDF* utilizando o conjunto de classes *TCPDF*

```
1  <?php
2      error_reporting(E_ALL);
3
4      require_once('../assets/tcpdf/config/lang/eng.php');
5      require_once('../assets/tcpdf/tcpdf.php');
6
7      $getdata = $_GET['data'];
8      $qrsize = $_GET['size'];
9      $pdfpages = $_GET['pdfpages'];
10
11   $lobj_pdf = new TCPDF(); //Cria o objeto local TCPDF
12   for ($index = 0; $index < $pdfpages; $index++) {
13    $lobj_pdf->AddPage(); //Adiciona uma nova pagina ao objeto que estq sendo
          gerado.
14    $lobj_pdf->write2DBarcode( $getdata, 'QRCODE,H', '20', '20', $qrsize,
          $qrsize);
15   }
16
17   $lobj_pdf->Output( 'qr_code.pdf', 'I' ); // Grava em disco o novo documento
         gerado
18
19   echo"'$lobj_pdf'";
20  ?>
```

Na *View* onde se encontra o formulário de criação das *TAGs* referente ao código *QR* é definido um *script*4.5 que utiliza os campos associados ao formulário e os passa como parâmetro para a função de geração.

De forma a prevenir o acesso à informação contida nos *QR Codes* por parte de utilizadores não autorizados, o seu conteúdo foi codificado através do algorítmo *MD5*. Assim o conteúdo do código é composto pela seguinte *string*:

$$QrPbApp * md5(data) \tag{4.1}$$

Listagem 4.5: *Script* para geracao do *PDF* e codigo *QR*

```
1 var Frame = document.getElementById('Frame');
2 var qrdata = '<?php echo "{$model->id_tag}|{$model->gen_date}|{$model->
      manual_URL}|{$model->manufacturer}|{$model->description}|{$model->id_node
      }"; ?>';
3 var qrsize = '<?php echo "$model->qrsize"?>';
4 var pdfpages = '<?php echo "$model->pdfpages"?>';
5 var qrdataencoded = Base64.encode(qrdata);
6 Frame.src = '<?php echo Yii::app()->request->baseUrl;?>/assets/pdf.php?data=
      QrPbApp*' + qrdataencoded + '&size=' + qrsize + '&pdfpages=' + pdfpages;
```

4.4 Serviços Web

Os serviços *Web* foram desenvolvidos com o intuito de realizar uma interface entre a base de dados e as aplicações cliente do sistema. No caso da arquitetura desenvolvida na presente dissertação, os serviços serão utilizados pela aplicação *Android* e pelo nó coordenador da *WSN*.

A *framework Yii*, através das suas funcionalidades, permite a implementação de serviços *REST*. Para desenvolvimento dos serviços, foi definido o controlador *WsrestController* onde são especificadas as ações que definem os serviços. Desta forma, cada ação corresponde a cada modelo definido na figura 3.15 e é composta pelos cinco serviços padrão especificados na tabela 3.3.

No capítulo 2 foi analisada a componente de gestão de *URLs* da *Yii*. Tirando partido desta funcionalidade é possível definir as *URIs* referentes às ações para cada serviço e respetivos métodos *HTTP*. Assim, foram definidas as regras presentes em B.4. O serviço é identificado pela regra definida na componente *URLManager* do ficheiro de configuração da aplicação. Cada regra é composta pela *URI*, a ação, parâmetros passados na *URI* e o respetivo método 4.2.

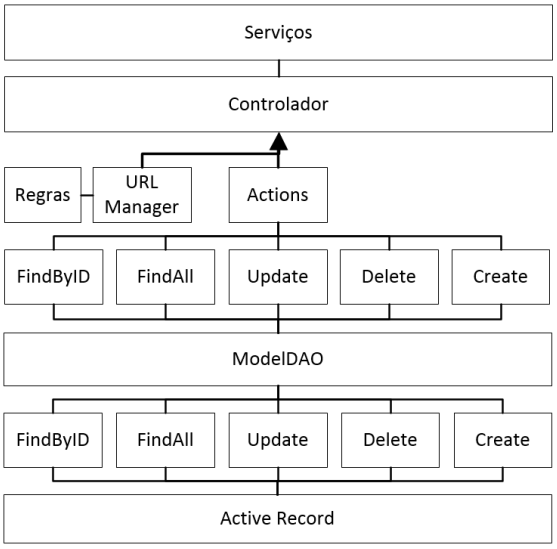


Figura 4.8: Representação das camadas do sistema dos serviços *Web* implementados utilizando as funcionalidades da *Yii*.

$$array('URL','pattern' => 'controller/action','verb' => 'method'), \qquad (4.2)$$

Em 4.6 estão presentes as regras definidas para os cinco serviços padrão no caso do modelo *Node*. Na primeira componente da regra, *'wsrest/node'*, define a *URL* que irá ser utilizada para realizar o pedido. No *pattern* é definido o controlador *wsrest* e a ação que é identificada pelo modelo do serviço. Também podem ser passados parâmetros através da *URI*, no caso do serviço de pesquisa e remoção é passado o *ID* da instância do modelo. Foi definida uma camada *DAO* que contem as funções necessárias pada cada método. Esta estrutura de classes está presente no diagrama de classes da figura 4.10. O fluxograma da figura 4.9 representa o corpo comum das cinco funções definidas. A primeira parte da função é composta pela verificação da autenticação do utilizador e se o nível de autenticação é suficiente. De seguida é verificada a correção do método, processado o pedido e enviada a respetiva resposta com o código de estado *HTTP*.

Listagem 4.6: Regras das *URIs* dos servicos *Web* correspondente ao modelo No

```
1 <?
2 //node services rules
3 array('wsrest/node', 'pattern'=>'wsrest/node', 'verb'=>'GET|POST|PUT'),
4 array('wsrest/node', 'pattern'=>'wsrest/node/<id:\d+>', 'verb'=>'GET|DELETE'),
5 ?>
```

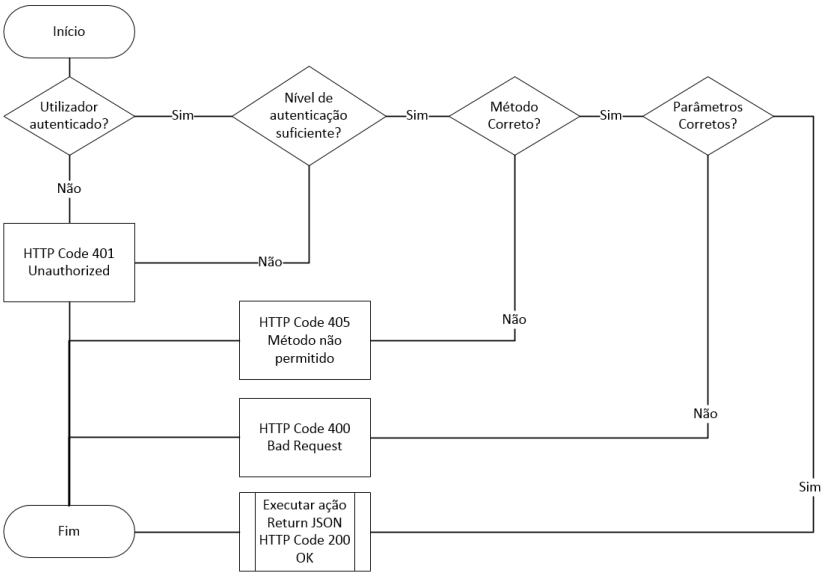


Figura 4.9: Fluxograma de representação genérica de um serviço *REST* implementado

Login e autenticação

A autenticação do utilizador é realizada através do serviço de *Login* de forma a permitir ao utilizador o acesso a todos serviços do sistema. Por questões de segurança, o serviço utiliza a sessão *HTTP*, classe *httpSession*, implementada pela classe *CWebUser*, que representa o utilizador atual. Assim, após a autenticação do utilizador, é criada uma sessão com os parâmetros *username*, *ID* e o *SessionID*. Assim é possível usar o método *Yii:app()->user->isGuest* da classe *CUser* herdado da classe *CWebUser*. Este método contém o *ID* da sessão associada ao utilizador. Quando é executado o pedido *HTTP*, este contém o *SessionID* da conexão existente e é verificado se a sessão do utilizador é válida e existente no servidor, figura 4.11.

A utilização deste conceito será exemplificado na secção da aplicação *Android*, onde é necessário aplicar o *pattern Singleton* para manter a sessão ativa e consequente *SessionID* válido numa única instância da classe que implementa a entidade *HTTP*. Desta forma evita-se a troca dos dados do utilizador em todos os pedidos dos serviços, restringindo a troca dos dados apenas ao serviço de *login*.

O serviço de login é utilizado através da *URL /wsrest/login* e do método *HTTP POST*, sendo necessário enviar os parâmetros *username* e *password* num objeto *JSON*. Após o pedido ser aceite, é devolvido o objeto *user* no formato *JSON* com todas as informações do respetivo utilizador 4.7.

Listagem 4.7: Resposta na estrutura de dados *JSON* de um objeto *user*

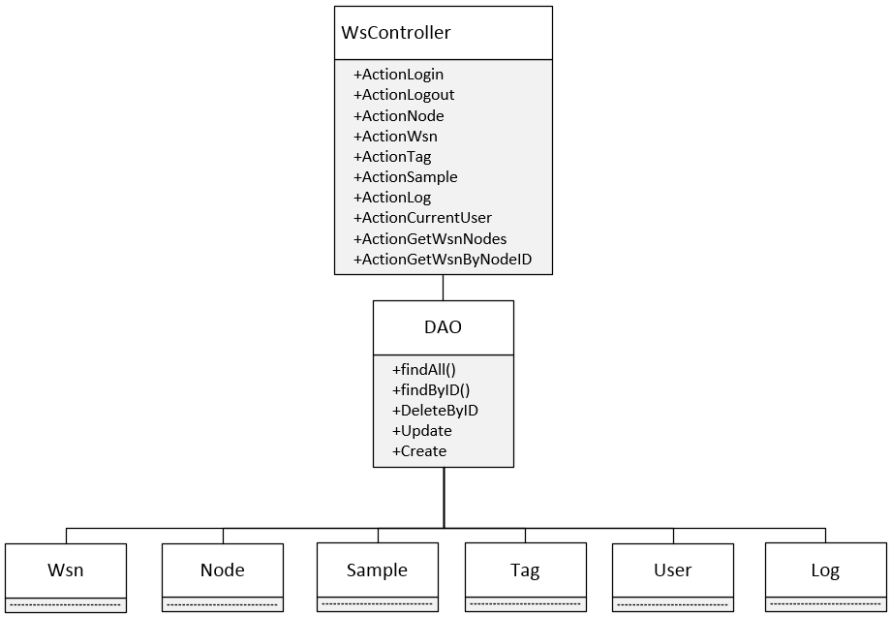


Figura 4.10: Diagrama de classes das componentes dos serviços *REST*

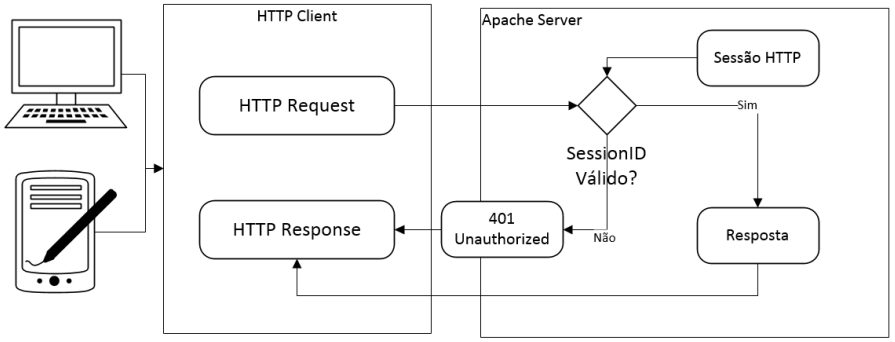


Figura 4.11: Representação de um *Http Request* e verificação da validade da sessão no servidor.

```
{
 "user": [
  {
    "id_user": "1",
    "name": "Carlos Pinto",
    "email": "adminwsninstaller@gmail.com",
    "phone_number": "900000000",
    "username": "admin",
    "registration_date": "2013-03-20 03:11:13",
    "authentication_level": "Administrator"
  }
 ]
}
```

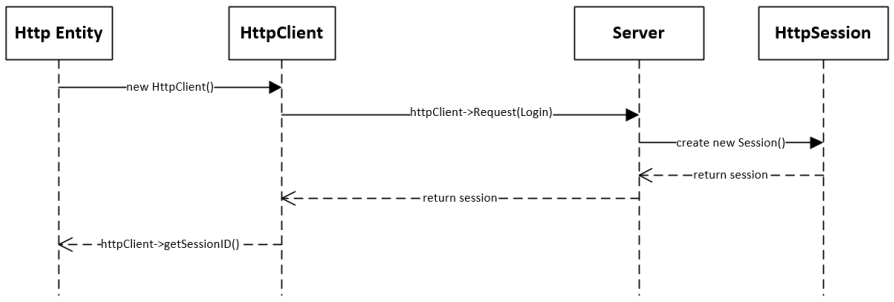


Figura 4.12: Diagrama sequencial do processo de criação de um sessão no servidor e respetivo cliente *http*

Utilização dos serviços

Os cinco serviços predefinidos, figura 3.3, permitem realizar operações *CRUD* aos modelos do sistema. Como estes serviços são comuns aos diversos modelos, de seguida é apresentado um exemplo da utilização dos serviços para o recurso/modelo *Node*.

O serviço que permite obter todos os elementos do recurso é identificado pela *URI wsrest/node* e pelo método *GET*. Se o pedido for executado com sucesso é devolvido um objeto *JSON* contendo um *array* de objetos *node* com a código de *status HTTP 200*, figura 4.8.

Listagem 4.8: Resposta na estrutura de dados *JSON* de um conjunto de nós do pedido de listagem de todos os nós

```
{
 "node": [
  {
    "id_node": "1",
    "soc": "CGB",
    "mac_address": "0:12:4b:0:1:47:4:de",
    "sensor": "Humidity",
    "sec_key": "1",
    "fir_rev": "1.01",
    "id_node_type": "1",
    "id_log_user": "7"
  },
  {
    "id_node": "2",
    "soc": "SAB",
    "mac_address": "0:12:4b:0:1:47:11:17",
    "sensor": "Temperature",
    "sec_key": "1",
```

```
19      "fir_rev": "1.01",
20      "id_node_type": "2",
21      "id_log_user": "8"
22    }
23   ]
24  }
```

4.5 Android

Nesta secção é apresentada a implementação do *GUI* de apoio à instalação de *WSNs*. A aplicação foi desenvolvida para as versões 4.0 ou superiores da plataforma *Android*. Para o seu desenvolvimento foi utilizada o *IDE Eclipse* e o framework *Android SDK*. Como foi especificado no capítulo anterior, a aplicação deve suportar a leitura e descodificação dos *QR Codes* gerados pelo *Website* do sistema, de forma a identificar o nó associado e a respetiva rede de sensores sem fios.

A aplicação *Android* utiliza os serviços *REST* para comunicar com a base de dados do sistema. Na plataforma *Android*, para ser executado um pedido *http* de um serviço *Web*, é necessário utilizar a classe *Asynktask* como subclasse e realizar o pedido *HTTP* de forma assíncrona, figura 4.13, no seu método *doInBackground*. A classe *HttpClient* é responsável pela execução dos serviços e contém o *Token* da sessão *Http* criada, sendo que identifica a sessão no servidor, sendo válida para execução dos restantes serviços após a execução do método de *login*. De forma a manter a sessão e a identificação da sessão aberta no servidor, é necessário que esta instância da classe se mantenha e seja utilizada para realizar os pedidos associados ao este utilizador. Para isso, foi implementado o *pattern Singleton* através do método *getHttpClient*, listagem 4.9.

Listagem 4.9: Implementação do *Pattern Singleton*

```
1  private static HttpClient getHttpClient() {
2   if (mHttpClient == null) {
3    mHttpClient = new DefaultHttpClient();
4    final HttpParams params = mHttpClient.getParams();
5    HttpConnectionParams.setConnectionTimeout(params, HTTP_TIMEOUT);
6    HttpConnectionParams.setSoTimeout(params, HTTP_TIMEOUT);
7    ConnManagerParams.setTimeout(params, HTTP_TIMEOUT);
8   }
9   return mHttpClient;
10 }
```

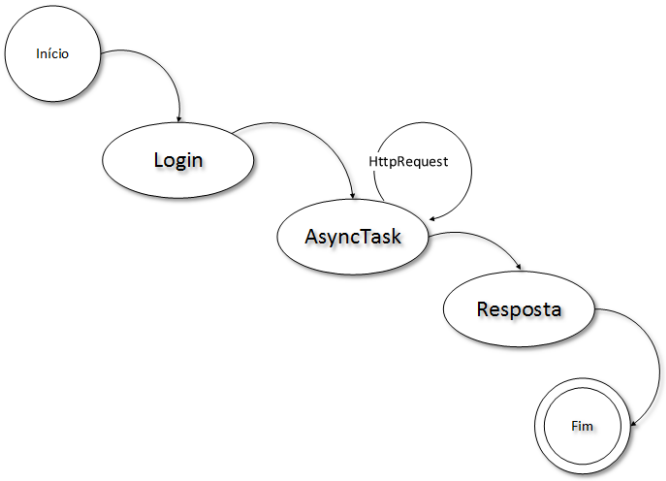


Figura 4.13: Diagrama de estados de um pedido *REST* para autenticação utilizando a classe *AsyncTask*.

Na estrutura da aplicação foram implementados as classes *Bean* dos modelos apresentados na figura 3.25. Na classe *ModelActions* foram definidas as funções necessárias para as operações *CRUD* para cada modelo. Cada função utiliza o método *execute* da classe *Asynctask* para obter o objeto *JSON* pretendido. Esta classe contém um construtor com os atributos necessários para cada pedido *HTTP*, sendo eles a *URI*, o método *HTTP*, os parâmetros na estrutura de dados *JSON*. As respostas dos pedidos são compostas por objetos *JSON* e pelos *Header* do serviço, que contém o *HTTP status Code* para verificação da validade do pedido. Cada método associado às operações *CRUD* de cada modelo, recebem o objeto *JSON* e realizam o *parser* para compor a resposta em objetos do modelo especificados em cada função, como pode ser visto nos métodos da classe genérica apresentada na figura 4.14.

A sequência de atividades da aplicação segue a especificação presente na figura 3.26. Assim, a primeira *Activity* apresentada na aplicação é composta por um ecrã de apresentação com os logotipos da empresa *Pinto Brasil* e da Universidade do Minho. De seguida, após dois segundos, é apresentada a *LoginActivity* que é composta por um formulário para preenchimento do nome de utilizador e *password*. Nesta atividade é utilizado o método de *Login* definido na classe *ModelActions*. Este método tem como parâmetros de entrada o objeto *JSON* com os atributos de *Login* e retorna um objeto *User* com os dados do utilizador autenticado se o início da sessão for realizada com sucesso. De forma a evitar que o utilizar realize o processo de *Login* sem conexão à *internet* foi implementado um *Broadcast* que desabilita o formulário de *Login*, prevenindo assim que o utilizador

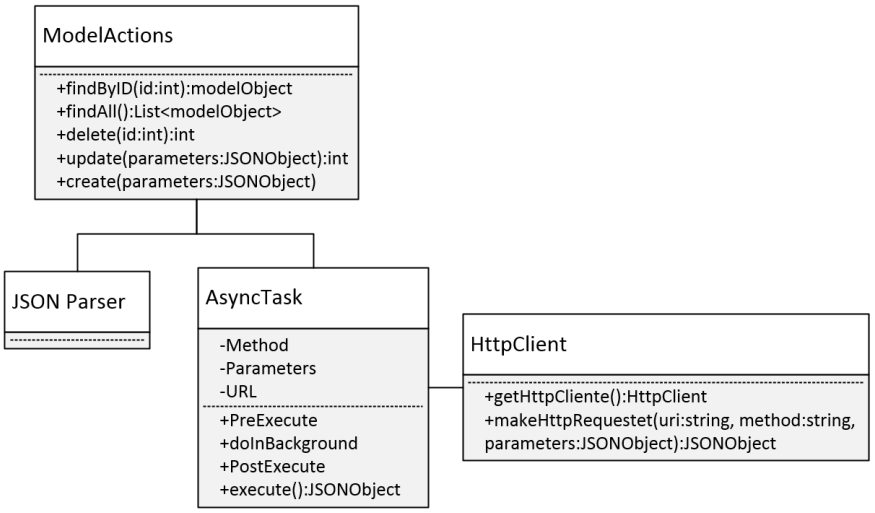


Figura 4.14: Diagrama que compõe as classes associadas às operações *CRUD* implementadas para a aplicação *Androdi*.

pressione o botão de *Login*, figura 4.16. Assim, no corpo do *broadcast* é detetada a mudança de estado do conexão à *internet* e executadas as respectivas intruções relatvias ao formulário de *Login*.

A *Activity* referente ao *Menu* da aplicação segue-se após a *Activity* de autenticação. Através deste *menu* é possível aceder a todas as funcionalidades do sistema representadas pelas *Activities* especificadas na figura 3.26 após a *MenuActivity*.

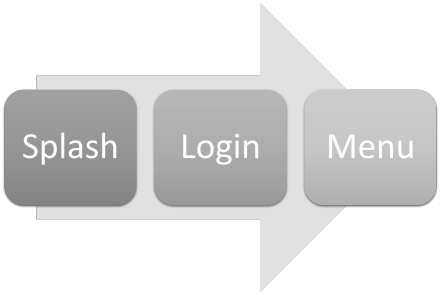


Figura 4.15: Sequência de atividades desde o início da aplicação até à *menu*.

4.5.1 Leitor de *NFC*, *QRCode* e biblioteca *ZXing*

O primeiro botão definido no *menu* permite ao utilizador aceder ao à funcionalidade de leitura de *TAGs QRCode* ou *NFC*. Para seleção destas opções é instanciado um *Dialog* que contém duas *CheckBoxs* que representam as duas possibilidades, figura 4.17.

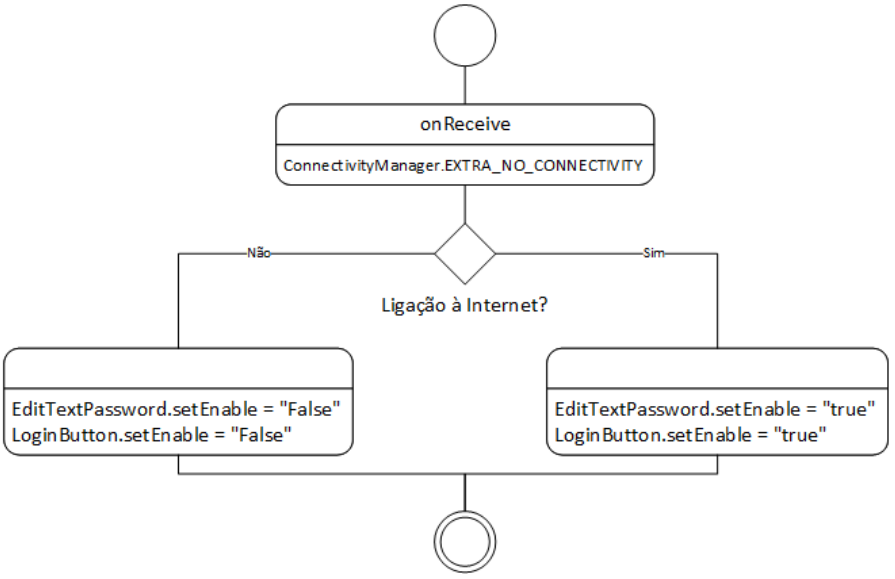


Figura 4.16: Representação do corpo do método *onReceive* do *BroadCast* para verificação das mudanças de estado na conexão à *internet*.

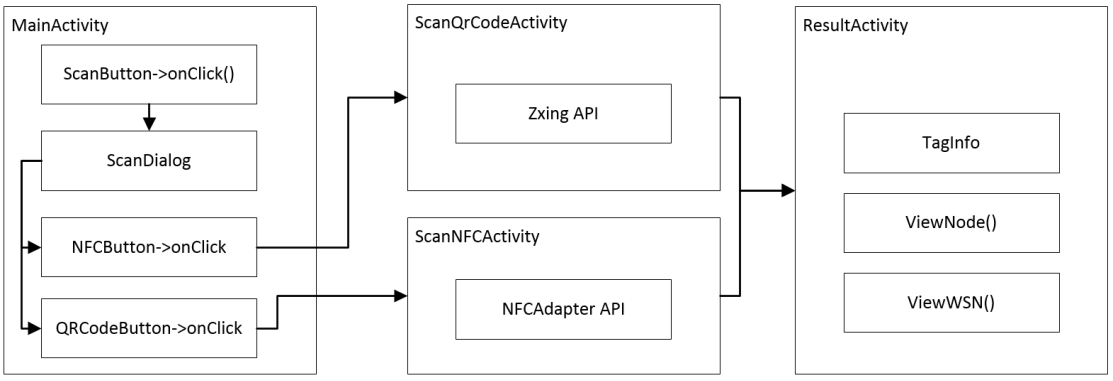


Figura 4.17: Arquitetura do sistema: servidores e aplicações

A leitura dos *QR Codes* nesta aplicação é realizada recorrendo à biblioteca *open-source Zxing*[41]. Esta biblioteca foi desenvolvida em *Java* e permite a leitura de diversos formatos e tipos de códigos de barras, inclusive *QR Codes*, utilizando a câmera embutida no *smartphone*. Após a seleção do modo de leitura dos *QR Codes*, é a instanciada a *Activity* de *Scan* forncida pela biblioteca. Através de *Intent* é passado o tipo de código que se pretende ler e o modo da *API*, neste caso o modo *SCAN_MODE*. Após ser devolvido o resultado este pode ter dois caminhos. Com esta funcionalidade é possível ler qualquer *QR Code* e apresentar o seu conteúdo. Se o código não cumprir as regras especificadas em 4.1, é mostrado no ecrã um *Dialog* com opções de pesquisa na *internet* ou abertura do conteúdo no *browser* do *Smartphone*. Pelo contrário, se o conteúdo do código corresponder

ao definido em 4.1, este é processado e de seguida apresentado o seu conteúdo na *QrResultActivity*. Como o conteúdo desta *TAG* contém a identificação do nó e os diversos parâmetros associados, nesta atividade foram criados dois botões que permitem navegar até à atividade que apresenta os dados do nó e à que representa a informação da *WSN*, figura 4.18.

Listagem 4.10: Utilização da biblioteca *ZXing* via *Intent* para leitura de *QR Codes*

```
1 Intent intentqrcode = new Intent("com.google.zxing.client.android.SCAN");
2       intentqrcode.putExtra("SCAN_MODE", "QR_CODE_MODE");
3       startActivityForResult(intentqrcode, 0);
4       dialog.dismiss();
```

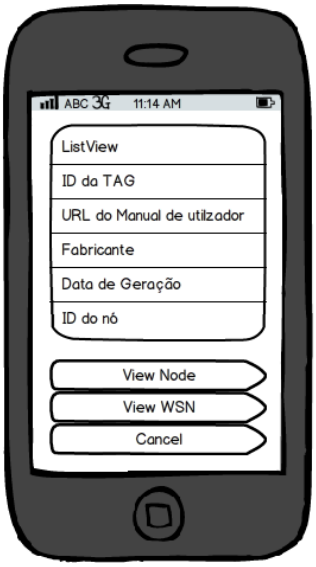


Figura 4.18: Layout da atividade que permite visualizar os parâmetros da *TAG*.

A leitura das *TAGs NFC* é realizada através da atividade *NFCScanActivity*. Como pode ser analisado no fluxograma da figura 4.19, nesta *activity* é verificado o estado do *adapter NFC*, se se encontrar ativo, é criado um *PendingIntent* que irá devolver os detalhes da *TAG* encontrada. Assim, é registada uma *Activity* com o filtro de *Intents* do tipo *MIME NFCDEF_DISCOVERED*.

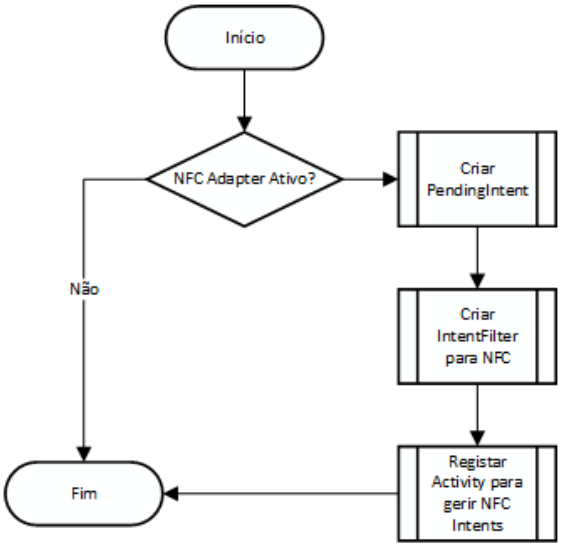


Figura 4.19: Fluxograma de registo do *adapter* e do *handler* da deteção de *TAGs NFC*.

Assim, após ser detetada um *TAG*, é executa a função *onNewIntent* que verifica e realiza o *parser* do conteúdo da mensagem da *TAG*, figura 4.20. Após isto, se a *TAG* for válida para a mensagem definida em 4.1, é devolvido o conteúdo para a atividade *QRREsultActivity*, figura 4.18.

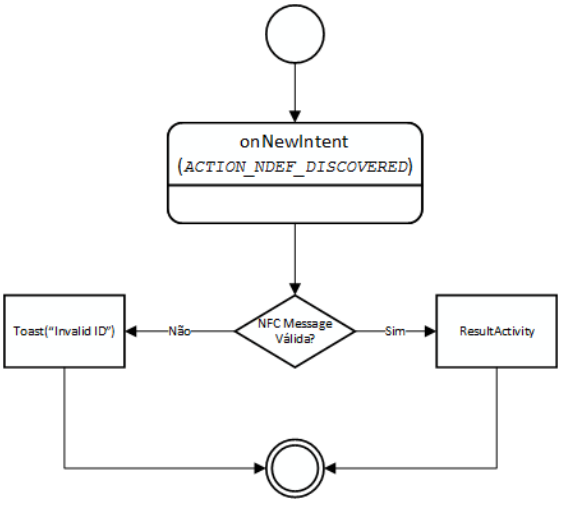


Figura 4.20: Fluxograma de representação do *Handler e parser* do conteúdo da mensagem da *TAG NFC*.

Capítulo 5

Testes e Resultados

No capítulo anterior foi apresentada a implementação dos módulos pertencentes à arquitetura do sistema. Primeiramente, foi apresentada a base de dados, seguindo-se do *Website*, dos serviços *Web* e aplicação *Android*. Por fim, foi apresentada a configuração do servidor e consequente instalação dos serviços *REST* e *Website* na *Cloud*.

No presente capítulo são apresentados os resultados da implementação da arquitetura do sistema, nomeadamente as aplicação *Android* e *Web*. Por fim é descrito o processo de instalação da *WSN* utilizando as ferramentas desenvolvidas.

5.1 Website

A aplicação *Web* foi desenvolvida com o propósito de fornecer uma plataforma que permite a gestão e monitorização das componentes do sistema. Através da *framework Yii* e a *bootstrap* do *Twitter* foi possível implementar um sistema robusto e *user-friendly*. Na página inicial, figura 5.1, pode ser visualizado o *menu* para navegação entre as funcionalidades do sistema. Nesta figura, o menu não disponibiliza os *links* de acesso às principais funcionalidades devido à falta de autenticação por parte do utilizador. Após o utilizador realizar o processo de *login*, é-lhe fornecido acesso às funcionalidades críticas conforme o seu nível de autenticação, disponibilizadas através do *menu* presente no cabeçalho da página.

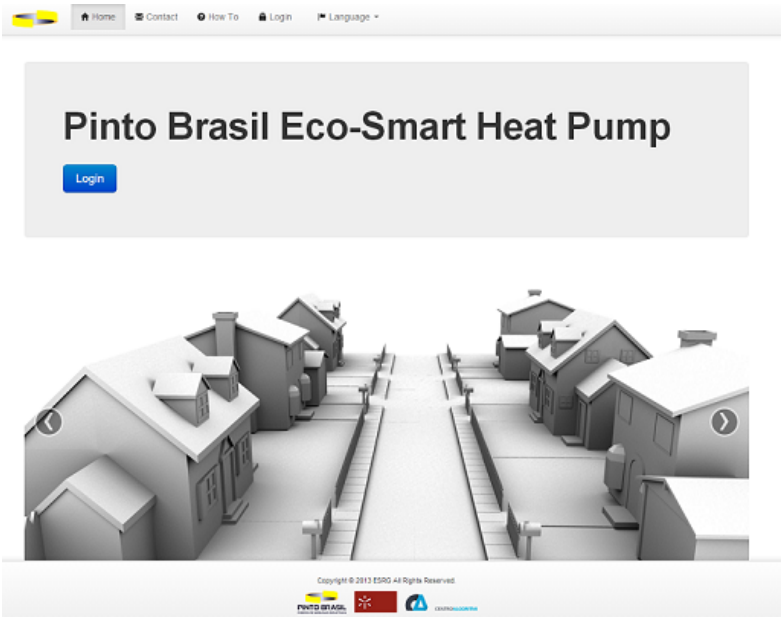


Figura 5.1: Página *Home* do *Website*.

A figura 5.2 contém a página de visualização do conteúdo de uma rede de sensores sem fios. Como pode ser analisado, é apresentada a informação relativa à *WSN*, assim como uma tabela com as informações dos nós que a compõem. Também foi implementada uma funcionalidade que permite associar utilizadores à *WSN*, de forma a ter acesso às operações e funcionalidades associadas.

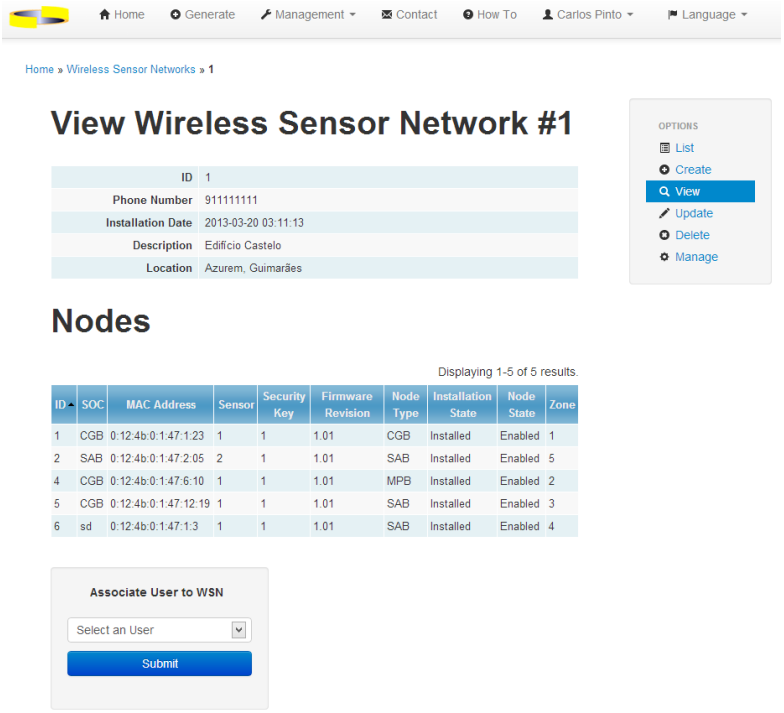


Figura 5.2: Página de visualização de *WSN*.

O *Website* disponibiliza a opção de visualização da lista de *WSNs* a que o utilizador autenticado tem acesso. Esta tabela, figura 5.3 contém as informações de cada rede e na última coluna é possível realizar três operações: ver, editar e remover.

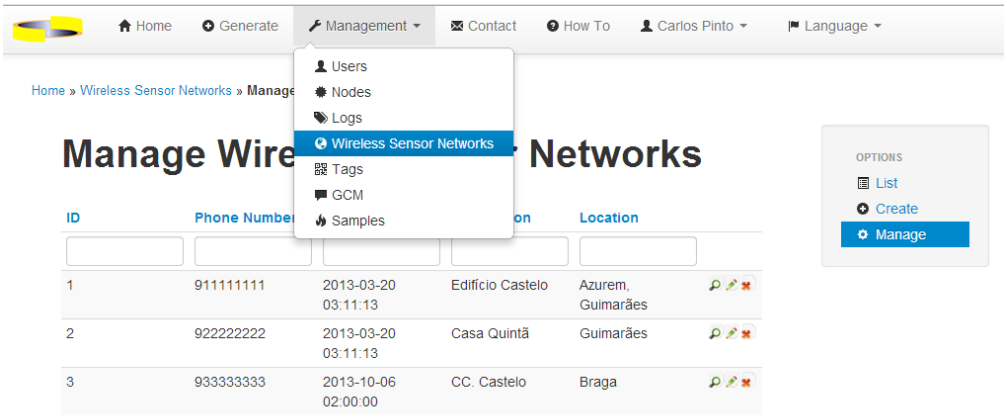


Figura 5.3: Página de visualização da lista de *WSNs* associadas ao utilizador autenticado.

A página da figura 5.4 define a funcionalidade de geração de *QR Codes*. Neste formulário, é possível introduzir todas as informações relativas à *TAG* e ainda definir o tamanho do código e o número de páginas para o ficheiro *PDF* a ser gerado.

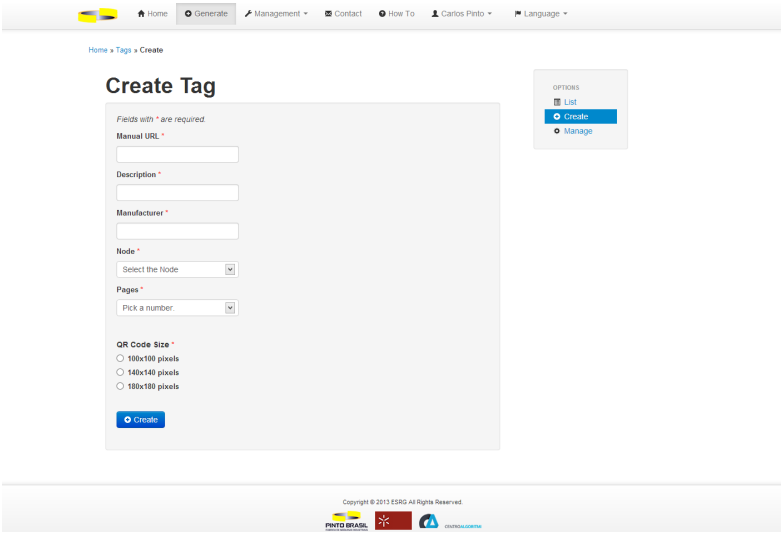


Figura 5.4: createtag

Após a geração do ficheiro *PDF* com o *QR Code*, o utilizador é redirecionado para a página de visualização da respetiva *TAG*, figura 5.6. Aqui é possível

consultar a informação da *TAG*, o ficheiro *PDF* e realizar o seu *download*, figura 5.6.

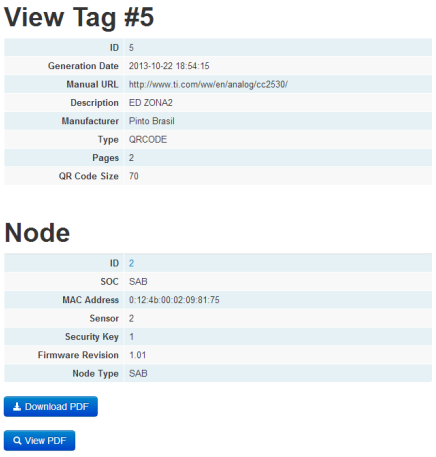


Figura 5.5: Página de visualização do conteúdo *TAG QRCode* e consulta e *download* do ficheiro *PDF* com o código *QR* (1).

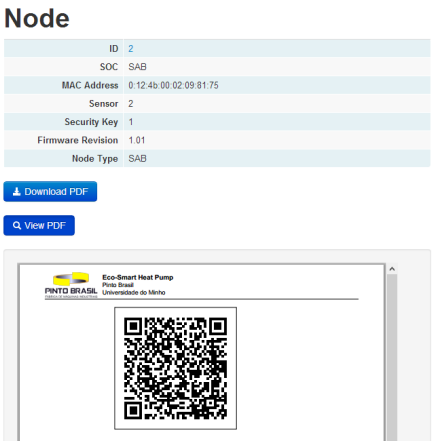


Figura 5.6: Página de visualização do conteúdo *TAG QRCode* e consulta e *download* do ficheiro *PDF* com o código *QR* (2).

O acesso ao ficheiro de instalação é realizado através da página de ajuda presente na figura 5.7. Deste modo, foram definidos dois modos para o *download* do ficheiro: um botão que contém a *URL* de acesso ao ficheiro ou através da leitura do código *QR* com o *smartphone* que contém a *URL* com o caminho para o ficheiro. Por último, no *menu* de navegação é possível definir a linguagem de apresentação do *Website*, como pode ser visto na figura 5.7.

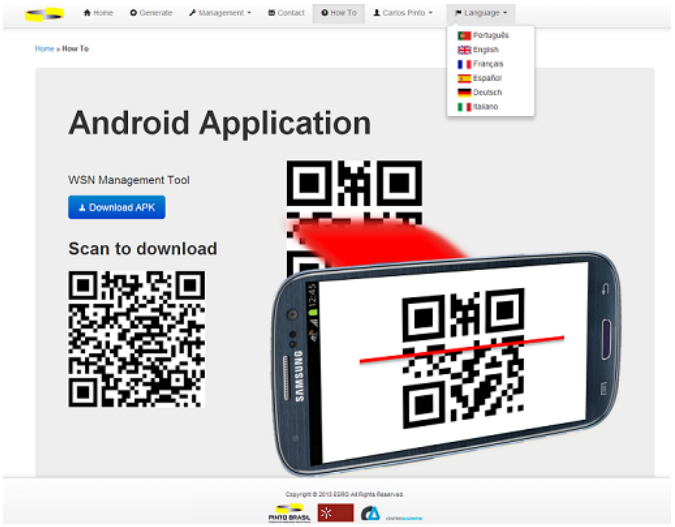


Figura 5.7: howtolanguage

5.2 Aplicação Android

Nesta secção são apresentados os resultados da implementação da aplicação *Android* e descritas as principais funcionalidades implementadas para este plataforma.

A aplicação de instalação exibe como primeira atividade uma animação *splash* com *fade effect* com duração de dois segundos, figura 5.8, composto com imagens dos colaboradores deste projeto. Ao tocar no ecrã do dispositivo ou pressionando o botão de *return*, é forçado o fim da animação e apresentada a atividade composta pelo funcionalidade de *Login*, figura 5.9.

O ecrã de login, figura 5.9 segue-se após o fim do *Splash* e restringe o acesso às funcionalidades da aplicação, uma vez que exige ao utilizador a sua autenticação. O formulário é composto por duas *EditText* e um botão, onde no seu *listener* do evento de *click*, é executado o pedido *REST* de autenticação e verificação do correto preenchimento dos parâmetros correspondentes ao *username* e *password*. Nesta *activity* foi implementado um *broadcast* que verifica a alteração do estado da ligação à *internet* e desabilita o formulário de *login* de forma a evitar a tentativa de autenticação sem conexão de dados.

Figura 5.8: Atividade com animação composta por *fade effect*

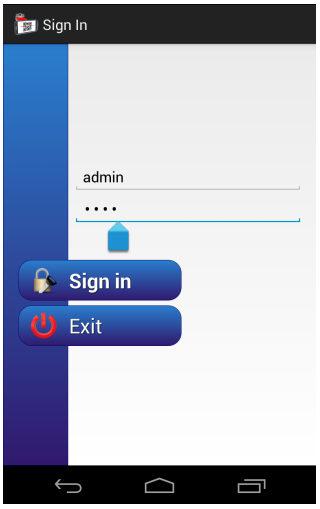


Figura 5.9: Atividade com o formulário de autenticação.

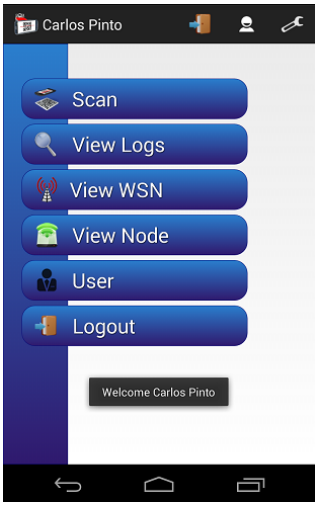


Figura 5.10: Ecrã com o *menu* principal para acesso às funcionalidades da app.

A figura 5.10 apresenta menu de navegação entre as funcionalidades da aplicação, sendo apresentado após ser efetuada uma autenticação com sucesso. Nesta

atividade também é utilizado o *broadcast* de verificação do estado da ligação à *internet*. Caso a conexão seja perdida, o *menu* é fechado e exibido o ecrã de *login* novamente.

As figuras 5.11, 5.13 e 5.12 compõe a sequência de leitura de um código *QR* para identificação de um nó. Assim, primeiro é lançado o *dialog* com a opção de escolha do método de leitura da *Tag*. Caso seja escolhido a opção *QR Code*, é iniciado o modo de leitura de códigos de barra utilizando a *API ZXing* de forma a ser inicializada a câmera do *smartphone*. Caso o *QR Code* identificado não corresponda aos gerados pelo *Website*, através da verificação da correspondência do conteúdo do código com componente 4.1, é mostrado um *dialog* com as opções de pesquisa no motor de busca *Google* ou de abertura da *URL* através do *browser* predefinido. Por outro lado, caso o código seja válido, é descodificada a mensagem do código e mostrado no ecrã o conteúdo da *TAG*. São ainda definidos dois botões para visualização do nó correspondente ou para acesso À *WSN* correspondente ao nó, figura 5.14.

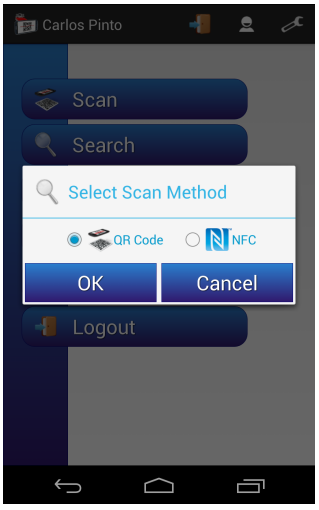


Figura 5.11: *Dialog* para seleção do modo de leitura das *TAGs*

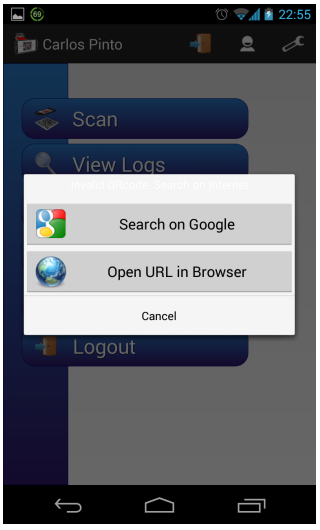


Figura 5.12: *Dialog* para escolha da pesquisa do conteúdo da *TAG* na *Web*

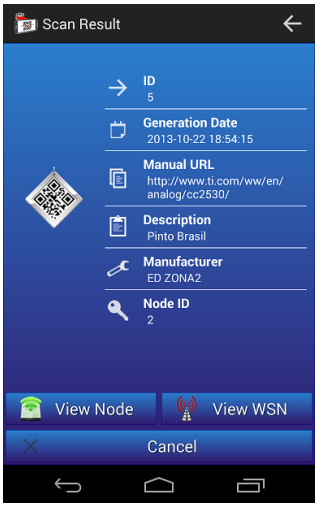


Figura 5.13: Atividade de visualização do conteúdo da *TAG QR Code*

Caso seja escolhida a opção de leitura das *TAGs NFC*, é iniciada uma *activity* que no seu método *OnCreate*, é implementada a verificação do estado do periférico *NFC* e definida a ação consoante o resultado. Se o periférico não estiver ligado, é iniciado um *dialog* com um botão para acesso às definições para ativar a componente *NFC*, figura 5.17. Pelo contrário, se o *NFC* se encontrar ativo, é apresentado o ecrã da figura 5.16, de modo a indicar ao utilizador para aproximar

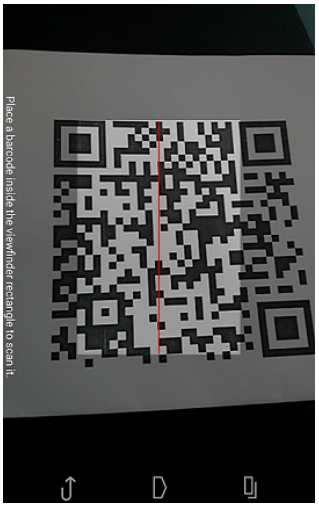


Figura 5.14: Ecrã de Leitura de um código *QR* utilizando a câmera do *smartphone*.

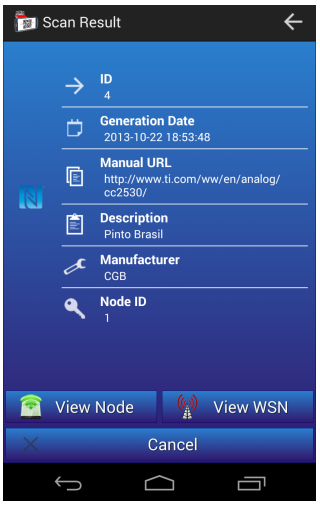


Figura 5.15: Atividade de visualização do conteúdo da *TAG NFC*

Figura 5.16: Atividade para leitura de *TAGs NFC*

a *TAG NFC* ao *smartphone* para a sua leitura. Se o conteúdo da *TAG* se encontrar correto, é apresentada a informação da figura 5.15, contendo as funcionalidades idênticas às da funcionalidade de leitura de *QR Codes*.

As *WSNs* podem ser consultadas de duas formas, 5.18: visualização de todas as *WSNs* através de um *Slide View*, contendo uma *WSN* por cada página de navegação ou através da visualização de uma *WSN* pesquisando por *ID* ou número de telefone, figura 5.23. A utilização da *Slide View* permite a rápida visualização das diversas *WSNs* de forma rápida através do movimento de *Swipe* horizontal no ecrã do *smartphone*. Em cada página de navegação é possível consultar as informações da rede correspondente, figura 5.22, e da legenda do estado dos nós, presente na figura 5.26.

No ecrã de visualização da rede, esta é composta por diversos nós dispersos na figura de representação de uma habitação e respetivas zonas. O utilizador pode selecionar cada nó através do toque no respetivo símbolo no ecrã da atividade. Desta forma, se o nó se encontrar instalado, é apresentado um *dialog*, figura 5.20, que permite a sua consulta, remoção ou alteração do estado de instalação e ativação. Pelo contrário, se não existir nó na zona selecionada, símbolo com cor cinzento, é apresentado um *dialog* com a opção de leitura de um código *QR* para adição do respetivo nó. A funcionalidade de pesquisa por parâmetro apenas disponibiliza uma *WSN* para visualização. Esta atividade é composta por um *menu* definido por três tabulações com a *WSN* (figura 5.23), as informações (figura 5.24)

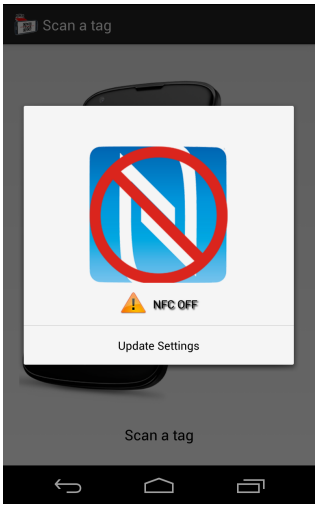


Figura 5.17: Ecrã com botão para acesso às definições de *NFC*

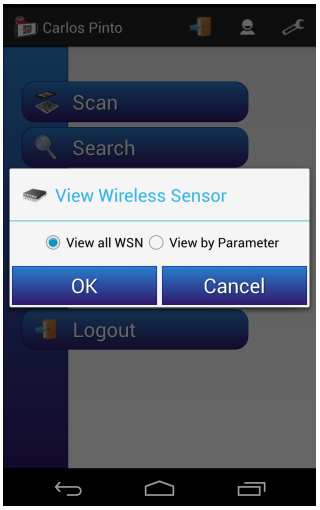


Figura 5.18: Ecrã com *Dialog* para seleção do modo de pesquisa de *WSNs*

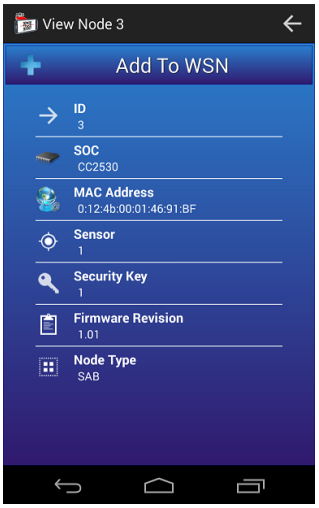


Figura 5.19: Ecrã de visualização do conteúdo de um nó com opção de associação a uma *WSN*

e os *Logs*.

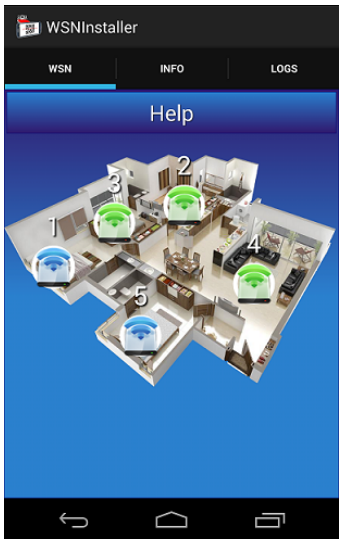


Figura 5.23: Atividade com menu com *Tabs* para visualização das componentes da *WSN*.

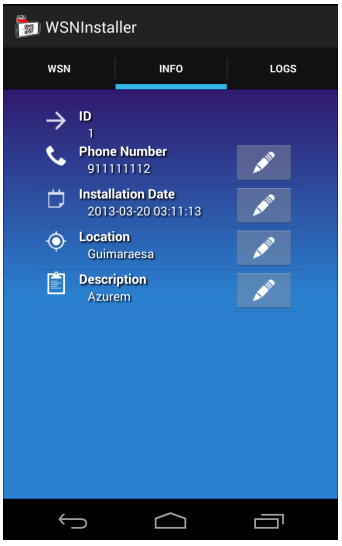


Figura 5.24: *Tab* que permite a consulta e edição da informação da *WSN*.

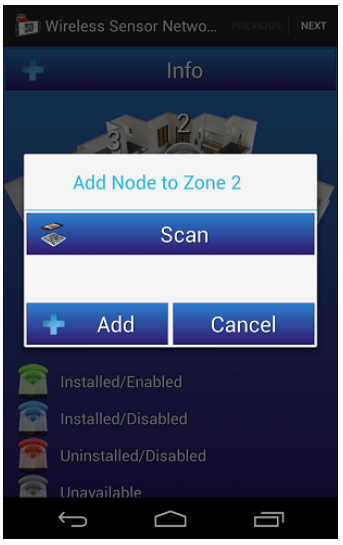


Figura 5.25: *Dialog que permite adicionar um nó à zona especificada da* WSN.

A figura 5.27 contém a atividade que permite utilizar a funcionalidade de pesquisa de nós do sistema através do *ID* correspondente. Uma vez que o *ID* seja válido e nível de autenticação permite a sua consulta, é apresentado a atividade de visualização da informação do nó presenta na figura 5.19.

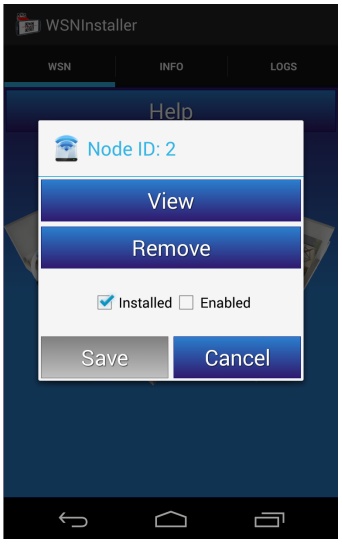


Figura 5.20: Atividade com *dialog* para realizar operações sobre o nó.

Figura 5.21: Atividade de visualização da representação de uma *Wsn*.

Figura 5.22: Atividade de visualização da representação de uma *Wsn* com respetiva informação vísivel.

A última funcionalidade disponibilizada no *menu* corresponde à listagem dos *Logs* do sistema. Esta lista apresenta os *Logs* por ordem decrescente por data de criação e cerca de dez *logs* de forma a facilitar a visualização por parte do utilizador. Se for realizado o *swipe* vertical até ao fim da lista, são adicionados os dez *logs* seguintes.

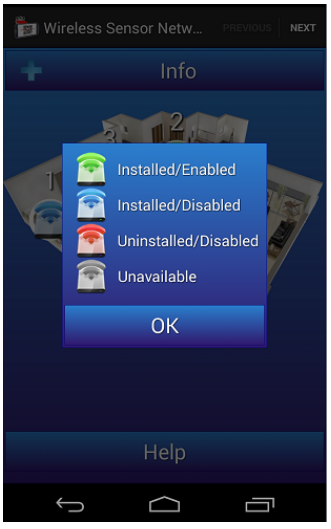


Figura 5.26: Ecrã que permite a consulta do *dialog* com os estados correspondentes dos nós.

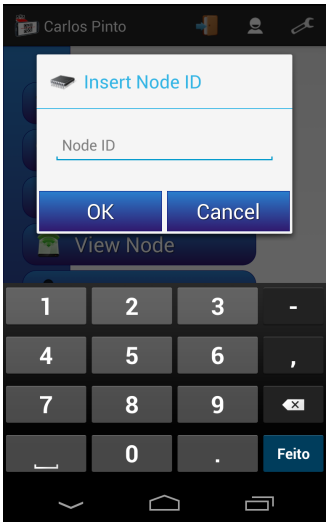


Figura 5.27: Funcionalidade de pesquisa de um nó através do seu *ID*.

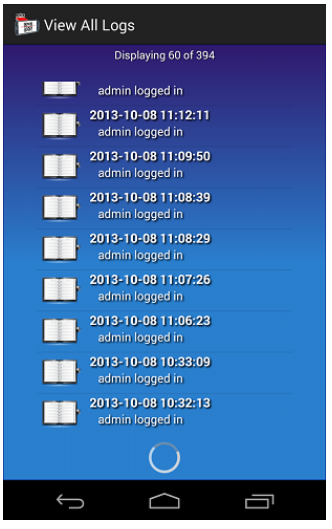


Figura 5.28: Funcionalidade de que permite a consulta de todos os *Logs* do sistema.

5.3 Processo de Instalação

Um dos objetivos principais da presente dissertação passa por definir o processo de instalação da *WSN* do projeto *Eco-smart Heat Pump* com o intuito de facilitar e auxiliar o instalador neste procedimento. Assim, de seguida é apresentado a metodologia do processo de instalação definido em 3.2.7 com recurso à utilização dos módulos implementados neste projeto.

O processo tem como finalidade a instalação da rede de sensores sem fios num ambiente de domótica de forma a manter o registo digital de todos os seus elementos para fácil monitorização e gestão de todo o sistema. O procedimento é definido por três passos: i) criação da WSN, ii) associação do Nó *CGB* e iii) deteção, adição e consequente ativação dos nós *SAB* à *WSN*.

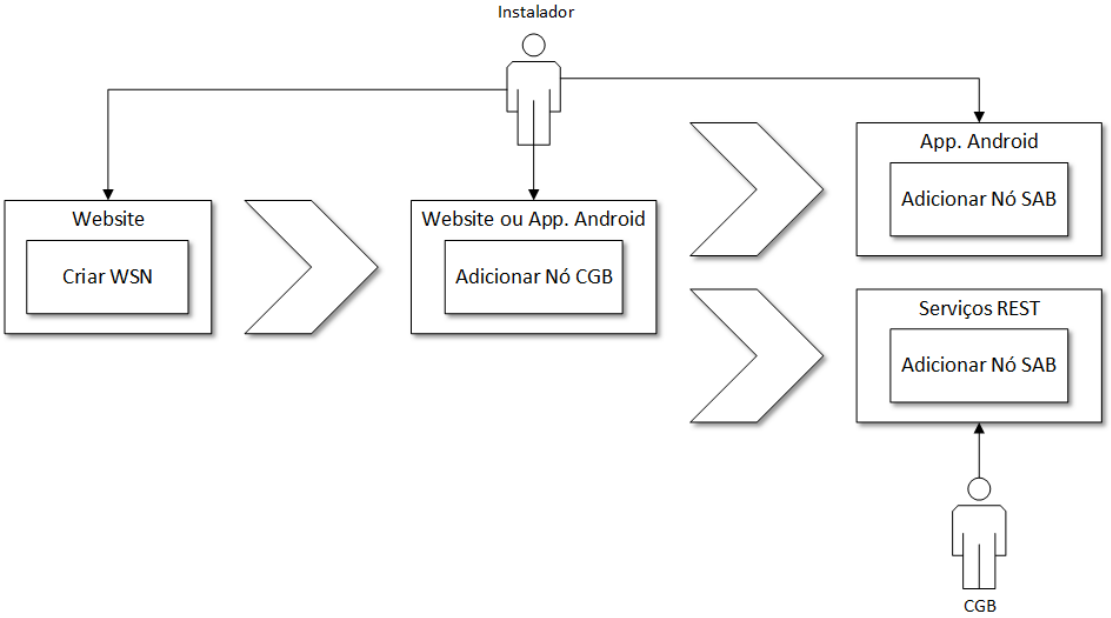


Figura 5.29: Representação da sequência do processo de instalação.

Assim, em i), o utilizador *instalador* deve criar a rede de sensores sem fios recorrendo à utilização da funcionalidade de criação de *WSNs* da aplicação *Web*, podendo ser acedida no *menu* lateral da funcionalidade de operações *CRUD* presente na figura 5.3. Esta funcionalidade é composta por um formulário, figura 5.30, definido pelas propriedades da rede a adicionar. No passo ii), o instalador, através do *Website* pode adicionar à *WSN* o nó *CGB* selecionado através da funcionalidade presente na página de visualização do nó, figura 5.31. Nesta ação, o

estado do *CGB* é definido como instalado, mas não ativo. Mas, após se proceder à ligação deste nó pela primeira vez, é definido o seu estado como ativo através da utilização do serviço *REST* que permite alterar o estado do nó no sistema. O serviço de alteração do estado do nó é definido pelos seguintes pontos:

- *URI*: */wsnnode/idNode*
- Método: PUT
- Parâmetros: *""id_installation_state":"1", "id_node_state":"1""*

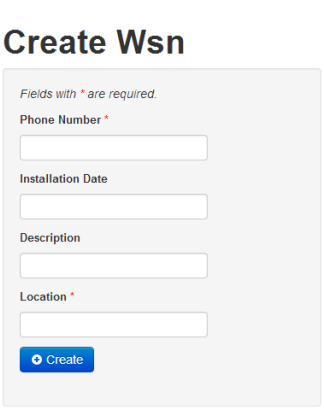


Figura 5.30: Página da aplicação *Web* com formulário de criação de *WSN*.

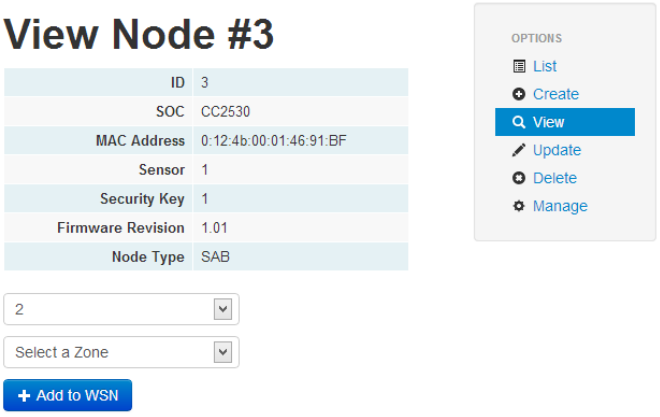


Figura 5.31: Página de visualização do nó com opção de associação a uma *WSN*.

O último procedimento de instalação resulta na adição de nós *SAB* à *WSN*. Este método é semelhante ao do nó coordenador, sendo necessário que o instalador associe o nó à rede através do *Website* ou através da aplicação *Android*. Com esta aplicação, o instalador pode aceder à funcionalidade de consulta das informações e de associação ao nó, figura 5.19. O acesso a este ecrã é realizado através da leitura do código *QR* ou através da pesquisa do nó pelo seu *ID*, figura 5.27.

Método de identificação do Nó

A cada nó da rede de sensores sem fios foram anexadas as *TAGs NFC e QR Code*, na parte superior e inferior respetivamente, figuras 5.32 e 5.33. Para a sua leitura é necessário utilizar a funcionalidade presente na figura 5.11 e proceder às ações pretendidas sobre na atividade apresentada na figura 5.12 após a obtenção do conteúdo do nó.

Figura 5.32: *Hardware* do nó com a *TAG QR Cde* anexada na parte inferior da placa.

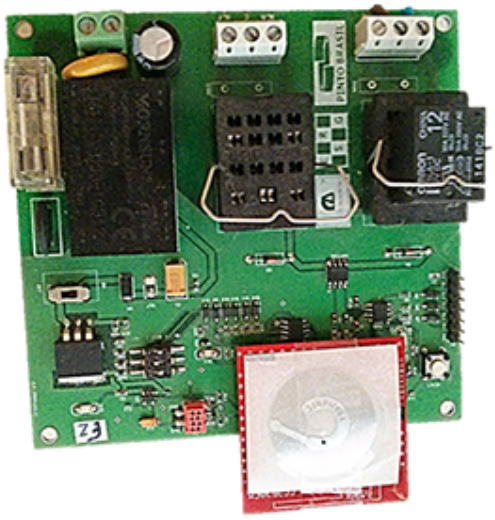


Figura 5.33: *Hardware* do nó com a *TAG NFC* anexada na parte superior da placa.

Capítulo 6

Conclusões e trabalho futuro

Neste capítulo são apresentadas as conclusões retiradas após o desenvolvimento do projeto da dissertação. Para além disso, são definidos possíveis melhoramentos a efetuar no sistema de forma a expandir a sua arquitetura e aumentar as funcionalidades oferecidas e a sua fiabilidade.

6.1 Conclusão

O sistema implementado no âmbito da presente dissertação propõe um método de instalação de *WSNs* e auxílio do utilizador *instalador* neste processo ou mesmo durante a manutenção do sistema tirando partido da utilização das tecnologias *QR Code* e *NFC* na identificação dos componentes da rede. Através deste método, os nós da *WSN* podem ser instalados com recurso à ferramente de leitura de códigos *QR* ou *TAGs NFC*, ou por outro lado, o processo de instalação pode ser realizado com recurso aos serviços *REST* implementados. Desta forma, o nó coordenador da rede, ao detetar um novo nó na rede, executa o serviço *Web* que corresponde à associação do nó à *WSN*. Neste sentido, é possível agilizar e automatizar o processo de instalação da *WSN* associada ao projeto *Eco-Smart Heat Pump* e monitorizar o sistema num único ponto através da aplicação *Web* ou com recurso à utilização da aplicação de instalação para a plataforma *Android*.

A arquitetura orientada a serviços implementada permitiu definir um barramento de serviços REST que realizam a interface entre as aplicações cliente e nó CGB com a base de dados. Assim, com a arquitetura *SOA* foi possível definir um relacionamento entre as diferentes plataformas e aplicações desenvolvidas com

linguagens diferentes através da ligação à *internet*.

O serviço de *Cloud Computing* da *Amazon* permitiu obter uma instância de uma máquina com o sistema operativo *Linux* para instalação do servidor *Apache* e *MySQL* para alojamento do *Website*, serviços e base de dados. Desta forma não foi necessária a utilização de um servidor físico, evitando custos de aquisição, manutenção e processo de configuração desta máquina.

Em relação aos requisitos do projeto, estes foram cumpridos pelo facto do sistema facilitar e dinamizar o processo de instalação através da utilização dos vários módulos implementados, fornecendo ao utilizador um conjunto de aplicações que lhe permite realizar a instalação da rede de sensores sem fios do projeto *Eco-Smart Heat Pump* de forma viável e flexível.

6.2 Trabalho Futuro

Apesar dos objetivos propostos terem sido cumpridos, é possível melhorar o sistema através da adição de diversas funcionalidades e melhoria da segurança de alguns componentes.

A primeira funcionalidade sugerida visa aumentar a versatilidade do sistema através implementação de uma componente que permita a adição de novos tipos de sensores e consequente registo de amostras dos novos sensores adicionados.

O aumento da segurança dos serviços *REST* recorrendo à encriptação dos dados utilizando função criptográficas tais como o *SHA1*, de forma a obter uma melhoria significativa na execução do sistema.

A terceira sugestão passa pela utilização de outras *frameworks Web* de modo a obter um *Website* mais dinâmico e com maior facilidade de acesso às funcionalidades em comparação com a implementação usada recorrendo à *framework Yii*.

Por último, o desenvolvimento da aplicação de instalação para os *smartphones* com sistema operativo *iOS* permitiria abranger um maior número de dispositivos que podem ser usados para realizar o processo de instalação, uma vez que os dispositivos com este *SO* em conjunto com as plataformas com o *SO Android* dominam a maior parte do mercado de dispositivos móveis, em termos de *tablets* e *smartphones*.

Bibliografia

[1] J. Kang, H. Lim, J. Kim, D.-H. Yeo, P. M. Jeong, T. Choi, D. Kim, W. Yang, and S. Kim, "Demo: Micro energy efficiency system based on QR code mote," in *SenSys 11 Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, NY, USA, 2011, pp. 379–380.

[2] "Millenial Media Q2 2013 Mobile Mix Report." [Online]. Available: http://www.millennialmedia.com/mobile-intelligence/mobile-mix/

[3] "Android Source Website." [Online]. Available: http://source.android.com/

[4] Denso, "QR Code Essentials," pp. 1–12, 2011.

[5] D. Mati, H. Kegalj, and D. Butorac, "Data access architecture in object oriented applications using design patterns," in *Electrotechnical Conference, 2004. MELECON 2004. Proceedings of the 12th IEEE Mediterranean*, 2004, pp. 595 – 598.

[6] K. Gill, S.-H. Yang, F. Yao, and X. Lu, "A ZigBee-Based Home Automation System," *IEEE Transactions on Consumer Electronics*, vol. 55, no. 2, pp. 422–430, May 2009.

[7] "HoneyWell RedLink Wireless System Website." [Online]. Available: http://www.forwardthinking.honeywell.com/products/wireless/zoning/zoning_feature.html

[8] "Homeseer Website." [Online]. Available: http://www.homeseer.com/

[9] M. Galeev, "Catching the z-wave," *Embedded Systems Design*, 2006.

[10] "Control4 Website." [Online]. Available: http://www.control4.com/

[11] A. I. Wasserman, "Software engineering issues for mobile application development," *Proceedings of the FSE/SDP workshop on Future of software engineering research - FoSER '10*, p. 397, 2010.

[12] J. Ohrt and V. Turau, "Cross-Platform Development Tools for Smartphone Applications," *Computing Practices*, pp. 72–79, 2012.

[13] "Android Developers Website." [Online]. Available: http://developer.android.com

[14] "Eclipse IDE Website." [Online]. Available: http://www.eclipse.org/

[15] "iOS Developers Website." [Online]. Available: https://developer.apple.com/devcenter/ios/

[16] "Google Android Fundamentals." [Online]. Available: http://developer.android.com/guide/components/fundamentals.html

[17] T. Erl, *SOA Principles of Service Design*. Prentice Hall, 2008.

[18] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, University of California, 2000.

[19] W3C Working Group, "Web Services Architecture W3C Working Group Note 11 February 2004," p. 98, 2004.

[20] C. Pautasso and F. Leymann, "RESTful Web Services vs . " Big " Web Services : Making the Right Architectural Decision Categories and Subject Descriptors," in *WWW '08 Proceedings of the 17th international conference on World Wide Web*, 2008, pp. 805–814.

[21] R. H. Erich Gamma, Ralph Johnson, John Vlissides, *Design Patterns Elements of Reusable Object-Oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc, 1995.

[22] "NFC Forum." [Online]. Available: http://www.nfc-forum.org/

[23] M. M. P. Paraye and M. S. V. Kulkarni, "Near field communication," in *Proceedings of the National Conference NCNTE-2012*, 2012, pp. 24–25.

[24] R. Want, "Near Field Communication," *Pervasive Computing*, vol. 10, no. 3, pp. 4–7, 2011.

[25] R. E. Johnson and B. Foote, "Designing Reusable Classes," *Journal of object-oriented programming*, vol. 1, no. 2, pp. 22–35, 1988.

[26] M. Mattsson, "Object-Oriented Frameworks A survey of methodological issues," Ph.D. dissertation, Lund University, 1996.

[27] S. Burbeck, "Applications programming in smalltalk-80 (tm): How to use model-view-controller (mvc)," *Smalltalk-80 v2. 5. ParcPlace*, 1992.

[28] M. Fowler, D. Rice, and M. Foemmel, *Patterns of Enterprise Application Architecture*. Addison-Wesley Longman Publishing Co., Inc, 2002.

[29] A. Leff and J. Rayfield, "Web-application development using the model view controller design pattern," in *Enterprise Distributed Object Computing Conference, 2001. EDOC '01. Proceedings. Fifth IEEE International*, Seattle, WA, 2001, pp. 118–127.

[30] W. Cui, L. Huang, L. Liang, and J. Li, "The Research of PHP Development Framework Based on MVC Pattern," *2009 Fourth International Conference on Computer Sciences and Convergence Information Technology*, pp. 947–949, 2009.

[31] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, Apr. 2010.

[32] Armbrust, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, and A. Rabkin, "A View of Cloud Computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[33] M. B. Mollah, K. R. Islam, and S. S. Islam, "Next generation of computing through cloud computing technology," *2012 25th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pp. 1–6, Apr. 2012.

[34] "Protocolo RF SimpliciTI." [Online]. Available: http://www.ti.com/corp/docs/landing/simpliciTI/index.htm?DCMP=hpa_rf_general&HQS=NotApplicable+OT+simpliciti

[35] Q. Xue and X. W. Zhuo, *The Definitive Guide to Yii 1.1*. Yii, 2012.

[36] "HTTP Status Codes." [Online]. Available: http://www.w3schools.com/tags/ref_httpmessages.asp

[37] "Amazon Web Services Website." [Online]. Available: http://aws.amazon.com/

[38] "MySQL Website." [Online]. Available: http://www.mysql.com/

[39] "Yii Framework Website." [Online]. Available: http://www.yiiframework.com/

[40] “Twitter Bootstrap Website.” [Online]. Available: http://getbootstrap.com/2.3.2/

[41] “Biblioteca ZXing (Zebra Crossing).” [Online]. Available: https://code.google.com/p/zxing/

[42] L. Finzgar and M. Trebar, “Use of NFC and QR code identification in an electronic ticket system for public transport,” in *Software, Telecommunications and Computer Networks (SoftCOM), 2011 19th International Conference on*, Split, 2011, pp. 1–6.

[43] M. Media, “Q1 2013 Mobile Mix Report,” *Q1 2013 Mobile Mix Report*, pp. 1–7, 2013.

[44] T. J. Soon, “QR Code,” *Synthesis Journal*, pp. 59–78, 2010.

[45] P. Sutheebanjard and W. Premchaiswadi, “QR-code generator,” *2010 Eighth International Conference on ICT and Knowledge Engineering*, pp. 89–92, Nov. 2010.

[46] “World Wide Web Consortium.” [Online]. Available: http://www.w3.org/

[47] “Embedded Automation.” [Online]. Available: http://www.embeddedautomation.com/

[48] “IDC MOS Mercado.” [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=prUS24108913

Anexos

Anexo A

Aplicações e GUIs de empresas

Figura A.1: Aplicação Web *Honeywell*: Controlo de temperatura de uma zona.

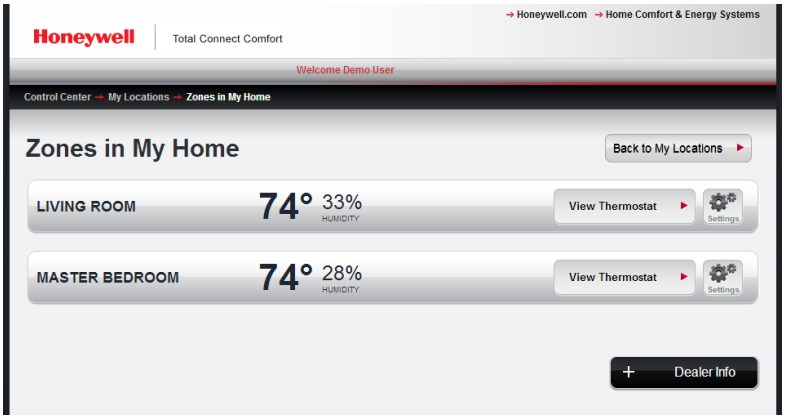


Figura A.2: Aplicação Web *Honeywell*: Monitorização das zonas do domicílio.

Anexo B

Classes do *Website*

Listagem B.1: Declaração de um botão sob a forma de *widget*

```
1  <?php
2
3  class UserIdentity extends CUserIdentity
4  {
5      public function authenticate(){
6          $record = User::model()->findByAttributes(array('username'=>$this->
               username));
7          if($record === null)
8              $this->errorCode=self::ERROR_USERNAME_INVALID;
9          else if($record->password !== md5($this->password))
10             $this->errorCode=self::ERROR_PASSWORD_INVALID;
11         else
12         {
13             $this->setState('fullname', $record['name']);
14             $this->setState('username', $record['username']);
15             $this->setState('UserID', $record['id_user']);
16             $this->setState('isAdmin', $record['id_authentication_level']);
17             $this->setState('logUser', $record['id_log_user']);
18             $this->errorCode = self::ERROR_NONE;
19         }
20         return !$this->errorCode;
21     }
22 }
23
24 ?>
```

Listagem B.2: Classe modelo que representa um nó da *WSN*

```
<?php
class Node extends CActiveRecord
{
 public function rules()
 {
  return array(
   array('soc, mac_address, sensor, sec_key, fir_rev, id_node_type', 'required'),
   array('sensor, id_node_type', 'numerical', 'integerOnly'=>true),
   array('soc, mac_address, sec_key, fir_rev', 'length', 'max'=>45),
                        array('mac_address','unique','message'=>'{attribute}:{value} already exists!'),
   array('id_node, soc, mac_address, sensor, nodeType, sec_key, fir_rev, id_node_type', 'safe', 'on'=>'search'),
  );
 }

 public function relations()
 {
  return array(
   'idNodeType' => array(self::BELONGS_TO, 'NodeType', 'id_node_type'),
   'zones' => array(self::MANY_MANY, 'Zone', 'node_on_zone(id_node, id_zone)'),
   'qrCodes' => array(self::HAS_MANY, 'QrCode', 'id_node'),
   'tags' => array(self::HAS_MANY, 'Tag', 'id_node'),
   'wsns' => array(self::MANY_MANY, 'Wsn', 'wsn_node(id_node, id_wsn)'),
   'idLogUser' => array(self::BELONGS_TO, 'LogUser', 'id_log_user'),
  );
 }

 public function attributeLabels()
 {
  return array(
   'id_node' => 'ID',
   'soc' => 'SOC',
   'mac_address' => Yii::t('home', 'MAC Address'),
   'sensor' => 'Sensor',
   'sec_key' => Yii::t('home', 'Security Key'),
   'fir_rev' => Yii::t('home', 'Firmware Revision'),
   'id_node_type' => Yii::t('home', 'Type'),
   'nodeType' => Yii::t('home', 'Type'),
  );
 }
}
?>
```

Listagem B.3: Ação correspondente aos serviços relacionados com o modelo *Node*.

```php
<?php
public function actionNode() {
 //verificacao se o utilizar esta autenticado e se tem nivel
 //de autenticacao suficiente para aceder ao servico
 if (User::model()->hasWSAuth() AND !Yii::app()->user->isGuest)
 {
  $get_method = strtolower($_SERVER['REQUEST_METHOD']);
  //criar instancia da classe DAO relativa ao No
  $daoNode = new WebservicesDAONode();
  switch ($get_method) {
   case 'get': //listagem de todos os elementos ou por ID
    if(isset($_GET['id']))
     $daoNode->findByID($_GET['id']);
    else
     $daoNode->findAll();
    break;
   case 'post': //metodo para criacao de uma instancia
    $json = file_get_contents('php://input'); //ler parametros
    $obj = CJSON::decode($json);

    //funcao definida para criacao do objeto
    $daoNode->create($obj);
    break;
   case 'put': //metodo relativo a edicao de uma instancia
    $json = file_get_contents('php://input'); //recepcao dos parametros em
        JSON
    if(isset($_GET['id']) AND !is_null($json))
     $daoNode->update($json, $_GET['id']);
    else
     http_response_code(400); //bad request parametros invalidos
    break;
   case 'delete': //metodo de remocao
    if(isset($_GET['id'])) //verificar se o id foi passado como parametro
     $daoNode->delete($_GET['id']);
    else
     http_response_code(400);
    break;
   default:
    http_response_code(405); //invalid method
    break;
  }
 }
 else {
  http_response_code(401);
 }
}
?>
```

Listagem B.4: Regras das *URIs* dos serviços *Web*

```
<?php
//User services
array('wsrest/login', 'pattern'=>'wsrest/login', 'verb'=>'POST'),
array('wsrest/logout', 'pattern'=>'wsrest/logout', 'verb'=>'GET'),
array('wsrest/currentuser', 'pattern'=>'wsrest/user', 'verb'=>'GET'),
array('wsrest/registergcm', 'pattern'=>'wsrest/registergcm', 'verb'=>'POST'),
//wsn services
array('wsrest/wsn', 'pattern'=>'wsrest/wsn', 'verb'=>'GET|POST|PUT'),
array('wsrest/wsn', 'pattern'=>'wsrest/wsn/<id:\d+>', 'verb'=>'GET|DELETE'),
array('wsrest/wsnnode', 'pattern'=>'wsrest/wsnnode', 'verb'=>'POST|PUT|DELETE'
    ),
array('wsrest/incompletewsns', 'pattern'=>'wsrest/incompletewsns', 'verb'=>'
    GET'),
array('wsrest/wsnfreezones', 'pattern'=>'wsrest/wsnfreezones/<id:\d+>', 'verb'
    =>'GET'),
array('wsrest/getwsnbynodeid', 'pattern'=>'wsrest/getwsnbynodeid/<id:\d+>', '
    verb'=>'GET'),
array('wsrest/getwsnnodes', 'pattern'=>'wsrest/getwsnnodes/<id:\d+>', 'verb'=>
    'GET'),
//node services
array('wsrest/node', 'pattern'=>'wsrest/node', 'verb'=>'GET|POST|PUT'),
array('wsrest/node', 'pattern'=>'wsrest/node/<id:\d+>', 'verb'=>'GET|DELETE'),
//user services
array('wsrest/user', 'pattern'=>'wsrest/user', 'verb'=>'GET|POST|PUT'),
array('wsrest/user', 'pattern'=>'wsrest/user/<id:\d+>', 'verb'=>'GET|DELETE'),
//tag services
array('wsrest/tag', 'pattern'=>'wsrest/tag', 'verb'=>'GET|POST|PUT'),
array('wsrest/tag', 'pattern'=>'wsrest/tag/<id:\d+>', 'verb'=>'GET|DELETE'),
//sample services
array('wsrest/sample', 'pattern'=>'wsrest/sample', 'verb'=>'GET|POST|PUT'),
array('wsrest/sample', 'pattern'=>'wsrest/sample/<id:\d+>', 'verb'=>'GET|
    DELETE'),
//log services
array('wsrest/log', 'pattern'=>'wsrest/log/<model:\w+>', 'verb'=>'GET'),
array('wsrest/log', 'pattern'=>'wsrest/log/<model:\w+>/<id:\d+>', 'verb'=>'GET
    '),
?>
```

Anexo C

Estrutura de ficheiros de uma aplicação *Yii*

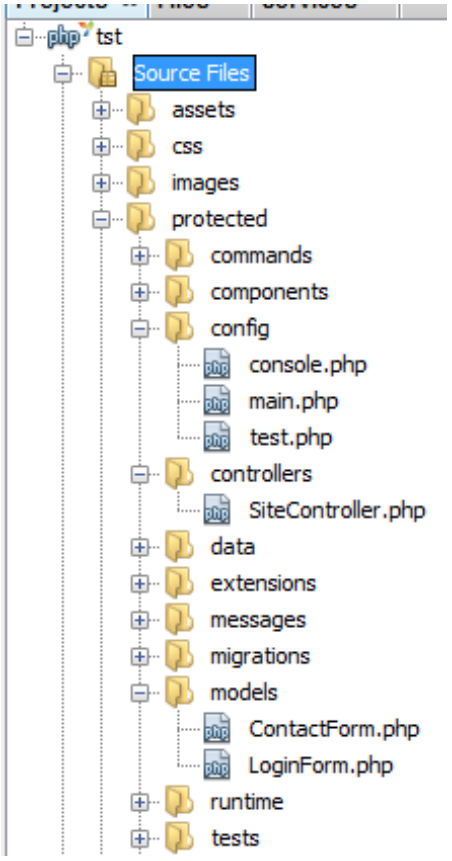


Figura C.1: Estrutura de uma aplicação *Web* utilizando a *framework Yii*(1).

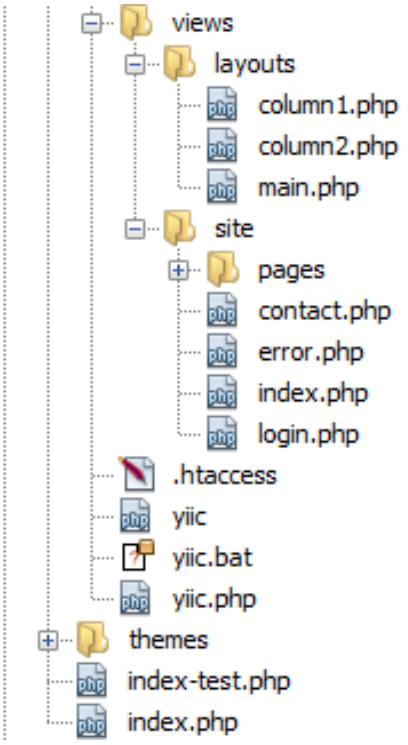


Figura C.2: Estrutura de uma aplicação *Web* utilizando a *framework Yii*(2).