Universidade do Minho
Escola de Engenharia

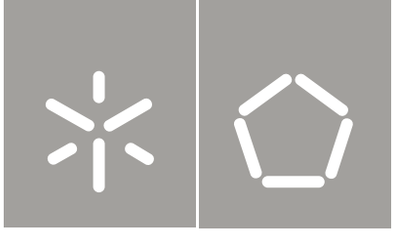
André Luís Roda Araújo

Sistemas de interação mínima baseados em plataformas móveis e Near Field Communication

Sistemas de interação mínima baseados em plataformas móveis e Near Field Communication
André Luís Roda Araújo

UMinho | 2013

dezembro de 2013



Universidade do Minho
Escola de Engenharia

André Luís Roda Araújo

Sistemas de interação mínima baseados em plataformas móveis e Near Field Communication

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 Paulo Cardoso

dezembro de 2013

Universidade do Minho, _____/_____/_____

_____

Agradecimentos

Esta dissertação, é o culminar de anos de conhecimento adquirido ao longo de um curso, ao longo de uma vida. Sem dúvida alguma que os meus maiores agradecimentos vão para os meus pais, Salvador Araújo e Maria de Fátima Araújo, porque sem o suporte deles em tudo nada seria possível. Neste contexto quero também agradecer aos meus irmãos, Sueli, Odete e Márcio, por em conjunto com os meus pais, me proporcionarem todas as condições para ser educado da melhor das maneiras.

Ao Professor Doutor Paulo Cardoso, meu orientador, que desde muito cedo contribuiu no sentido de melhorar cada vez mais a ideia desta dissertação e que sempre criticou, construtivamente, todas as ideias de modo a nunca deixar que a confiança na realização da mesma fosse abalada.

Ao ESRG e a todos os que formam o grupo de desenvolvimento, mas em especial ao César Meneses, ao Gabriel Pinto e ao António Campos, que sempre se mostraram disponíveis para me ajudar nas várias fases da dissertação.

Ao Rui Rodrigues que desde o inicio se mostrou disponível para ajudar e emprestar diverso material com o qual me foi possível trabalhar depois da operação.

Ao longo destes anos fiz vários amigos e todos eles sabem que mereciam ser mencionados pelo apoio nos bons e maus momentos, mas agradeço de forma especial aos meus amigos Carlos Esteves, Filipe Alves e David Cunha, que não só na Universidade mas fora dela, estão sempre presentes em todos os momentos.

A todos um muito obrigado.

Resumo

As evoluções tecnológicas na saúde tem acompanhado a grande revolução tecnológica que acontece todos os anos.

O Ambient Assisted Living (AAL) é uma área em grande crescimento na atualidade pois cada vez mais a idade média de vida da população aumenta e com isso mais cuidados específicos são necessários. A criação de condições para que a população idosa possa viver mais tempo em suas casas, independente e dignamente é o objetivo do AAL.

Esta dissertação baseia-se em dois tópicos:

1. aliar a massificação dos *smartphones* com o Sistema Operativo (SO) Android;
2. criar um ecossistema enquadrado em AAL, priorizando a interação mínima.

Para isto, foi criado um conjunto de aplicações que interligadas formam um ecossistema tecnológico de suporte à população idosa. Este ecossistema conta com suporte ao nível web, definido por um conjunto de serviços para a manipulação de informação entre a base de dados e as aplicações e de um *website* para gestão da informação. Ao nível do ecossistema Android, o sistema implementa dois métodos de garantir segurança do cliente: um detetor de quedas e um botão de alarme. É também permitida a receção de notificações para a toma de medicação e a leitura de *tags* Near Field Comunication (NFC) contendo informações sobre medicação e lista telefónica.

A aplicação foi implementada com sucesso e o seu uso em casos reais pode provar-se como uma mais valia em termos de AAL, melhorando de facto a vida de pessoas idosas.

Palavras-chave: AAl, NFC, Android, Arquitetura de *plugins*, Ecossistema

Abstract

Technological developments in health have accompanied the great technological revolution that takes place every year.

The AAL is an area of great exploration, with the increasing of the average age of living comes more specific health measures. The creation of possibilites for the elderly people to live longer, independent and dignified on their own is the goal of AAL.

This paper is built on two topics:

1. combining the massification of smartphones, specifically those who come with the Android Operating System;

2. the creation of an ecosystem framed in AAL giving priority to minimal interaction of the system.

For this it was created a set of interconnected application, forming a technological ecosystem, to support the elderly population. This ecosystem is web supported by a set of services to manipulate information between the database and the application, and a website used to manage information contained in the database. On the Android application level, the system implements two methods to ensure customer safety: a fall detector and an alarm button. it also allows the reception of notifications for taking medication and the reading of NFC tags containing information about medication and phone numbers for fast dial.

The application has been successfully implemented and its use in real cases proves to pay off in terms of AAL improving the lives of elderly people .

Keywords: AAl, NFC, Android, Plugin Architecture, Ecosystem

Acrónimos e siglas

AAL Ambient Assisted Living

AIDL Android Interface Definition Language

API Application Programming Interface

bps bits per second

CRUD Create Read Update Delete

CSS Cascading Style Sheets

DEI Departamento de Eletrónica Industrial

GCM Google Cloud Messaging

GPS Global Positioning System

GSM Global System for Mobile Communications

GUI Graphical User Interface

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

Hz Hertz

IDE Integrated Development Environment

IEEE Institute of Electrical and Electronics Engineers

IP Internet Protocol

IPC Inter-process Communication

IMC Índice de Massa Corporal

iOS iPhone Operative System

JAR Java Application Archive

JSON Javascript Object Notation

LAN Local Area Network

NDEF NFC Data Exchange Format

NFC Near Field Comunication

ORM Object Relational Mapping

OSGi Open Services Gateway initiative

PC Personal Computer

PHP PHP: Hypertext Preprocessor

POM Project Object Model

REST Representation State Transfer

RFCOMM Radio Frequency Communication

RFID Radio Frequency Identification

SD Secure Digital

SDK Software Development Kit

SMS Short Message Service

SMTP Simple Mail Transfer Protocol

SNS Serviço Nacional de Saúde

SO Sistema Operativo

SOAP Simple Object Access Protocol

SQL Structured Query Language

UE União Europeia

USD United States Dollar

UM Universidade do Minho

UML Unified Modeling Language

URI Uniform Resource Identifier

WAR Web Application Archive

WEP Wired Equivalent Privacy

WPA Wired Protected Access

WSDL Web Services Description Language

XAML Extensible Application Markup Language

XML Extensible Markup Language

Conteúdo

Página

Índice de tabelas

Índice de figuras

Capítulo 1

Introdução

1.1 Enquadramento

A saúde ocupa um lugar de importância absoluta na nossa sociedade e os avanços na área da medicina fazem com que a esperança média de vida da população seja cada vez maior. O envelhecimento da população mundial evidencia novos problemas e com ele novas soluções [3].

Em termos estatísticos a taxa de natalidade é agora inferior, somado a isto, estudos indicam que a população sénior vai duplicar até ao ano 2050. Esta camada populacional, é por natureza bastante carente de cuidados específicos e de ajuda para algumas tarefas tidas como básicas.

É neste contexto que surge o conceito de AAL. Os sistemas de AAL são soluções tecnológicas que visam a criação de ambientes que possam proporcionar ajuda e supervisão nas tarefas diárias dos utilizadores, garantindo uma monitorização sem a necessidade de existência de uma pessoa a tempo inteiro.

Estes sistemas garantem uma maior qualidade de vida em termos de saúde, segurança e bem-estar do utilizador, permitindo que vivam mais tempo autonomamente em suas casas . E este é o tema base desta dissertação.

Esta dissertação está focada em aspetos que estão na vanguarda da tecnologia como é o caso da programação para a plataforma *Android*, que atualmente é líder de mercado no que diz respeito a sistemas operativos móveis [4], e o uso da tecnologia NFC, que é uma forma de comunicação rápida e sem necessidade de interação física. O NFC dado a sua recente inclusão em *smartphones*, não possui muitas aplicações associadas pois ainda não é uma forma de comunicação maturada. Pode-se, no entanto, prever que o seu uso irá ser alargado e poderá ocupar uma posição muito promissora no mercado atual [5] pois as suas aplicações podem facilitar muitas das tarefas diárias.

Tendo em conta o atual contexto tecnológico e social, o que se pretende com esta dissertação é aplicar uma tecnologia recente, NFC, num contexto tecnológico que tem ganho cada vez mais atenção na atualidade, o AAL.

1.2 Objetivos

O objetivo da dissertação é a criação de um ambiente tecnológico centrado na utilização de aplicações móveis e com a menor interação possível, dado o utilizador alvo ser uma pessoa de idade avançada que normalmente não estão habituados a lidar com ambientes tecnologicamente avançados e tampouco têm facilidade em aprender a utiliza-las. O ecossistema aplicacional deve ser flexível e integrado, isto é, permitir a utilização de aplicações específicas para cada utilizador e que estas possam interagir entre si. O ecossistema deve funcionar no SO Android.

Para que possa ser criado um ecossistema, todas as aplicações tem de ser capazes de coexistir e utilizarem-se umas ás outras como se fossem apenas extensões das suas funcionalidades. Para isto ser possível, é necessária a criação de uma ambiente que estabeleça as formas de comunicação entre as aplicações, *middleware*.

Este ecossistema tem de ser capaz de reconhecer e diferenciar várias *tags* de NFC e para cada uma delas, efetuar um leque de ações previamente programadas tais como: realizar chamadas telefónicas para números definidos, identificar medicamentos e informar

o utilizador sobre a próxima toma e quantidades da mesma. Tem de ser capaz monitorizar atividade usando para esse fim os sensores existentes no equipamento móvel. Em adição a isto, o ecossistema tem de ser capaz de estabelecer uma comunicação com outros dispositivos externos a ele próprio. Para que isto possa ser feito, o sistema tem de comunicar com um servidor que contenha os *device drivers* apropriados a cada aparelho externo ao sistema.

O conjunto de aplicações finais deve primar pela facilidade de modo a ser considerada viável num contexto de utilização real.

1.3 Organização do documento

Esta dissertação está dividida em seis capítulos. O primeiro deles é a Introdução, capítulo atual, onde foram abordados os objetivos desta dissertação e qual o problema que ela vem colmatar.

No segundo capítulo, o Estado da arte, são analisados outros sistemas semelhantes cuja aprendizagem do funcionamento beneficia esta dissertação fortalecendo o caminho que se deve seguir ao ser analisada a concorrência. São também analisadas várias tecnologias que estão em jogo de modo a se poder perceber quais delas trarão um maior benefício quando utilizadas.

O terceiro capítulo diz respeito à Analise do problema e tem por objetivo perceber como vão ser aplicadas as tecnologias estudadas e qual vai ser o modelo a implementar do sistema, de modo a preencher os casos de uso e cumprir os objetivos propostos.

O quarto capítulo entra em detalhe sobre a implementação e a funcionalidades da camada web do sistema, sempre tendo em vista a sua integração com os diversos componentes dessa camada e com a integração com a camada Android. Neste capítulo é também analisada a base de dados, dando a conhecer a estrutura implementada podendo-se perceber melhor todo o sistema e toda a informação que nele vai circular.

No quinto capítulo é abordada a camada Android de modo a que no fim a informação possa ser um complemento dos capítulos anteriores, pois no ecossistema todos os micro

sistemas funcionam como um só. Neste capítulo dá-se ênfase à arquitetura de *plugins* implementada.

O sexto e último capítulo é a conclusão de toda a dissertação onde são tiradas as conclusões às perguntas e lacunas que esta dissertação vinha a responder e colmatar. É apresentado um resumo dos resultados obtidos e sugeridos alguns melhoramentos futuros da dissertação.

Capítulo 2

Estado da arte

O mercado atual está cada vez mais ciente das oportunidades de negócio geradas pelas soluções de *healthcare* destinadas à camada mais idosa da população. Existem já diversas empresas especializadas na criação de *hardware* e *software* para monitorização não intrusiva de pessoas, proporcionando uma maior qualidade de vida e independência aos utilizadores em suas casas.

Neste capitulo é descrito com algum pormenor algumas soluções já existentes no mercado, bem como abordar as tecnologias utilizadas na realização desta dissertação. Entre estas estão o sistema operativo *Android*, o NFC, utilizado para a interação com objetos físicos, a tecnologia subjacente à criação de um serviço *web* e a forma de comunicação com o dispositivo móvel.

2.1 Soluções de *healthcare*

Os serviços apresentados não contemplam todos os projetos existentes. Ao longo desta secção serão referenciados alguns serviços que servem como uma base sólida para o desenvolvimento de novos projetos como é o caso desta dissertação.

2.1.1 True-Kare

"Um telefone para seniores, controlado através da internet" [6] é assim que a empresa descreve o seu serviço, constituído por um telemóvel utilizado pela pessoa sénior e por um *website* gerido pela pessoa que acompanha o sénior.

O *website* apresentado na Figura 2.1, é a base de todo o sistema, nele é possível configurar e monitorizar o telemóvel. Segundo os seus criadores as funcionalidade do *website* permitem [6, 7]:

- Criação de contactos, atalhos no telefone, definir os números favoritos e configuração de menus do telemóvel;
- Fazer a gestão de toda a medicação, desde alertas para a toma de medicamentos até avisos de falta de *stock*;
- Definir alertas para datas de aniversários, visitas médicas e atividades diárias;
- Receber *e-mails* e Short Message Service (SMS) com a localização, medicamentos e relatórios de saúde;
- Monitorizar a pressão arterial, glicose, peso, IMC e temperatura corporal;
- Configurar a tecla SOS para alertas de SMS, *e-mails* e telefonemas com alta-voz e atendimento automático;
- Estabelecer um perímetro de segurança no Google Maps com alertas de SMS e chamadas no caso de saída do perímetro.

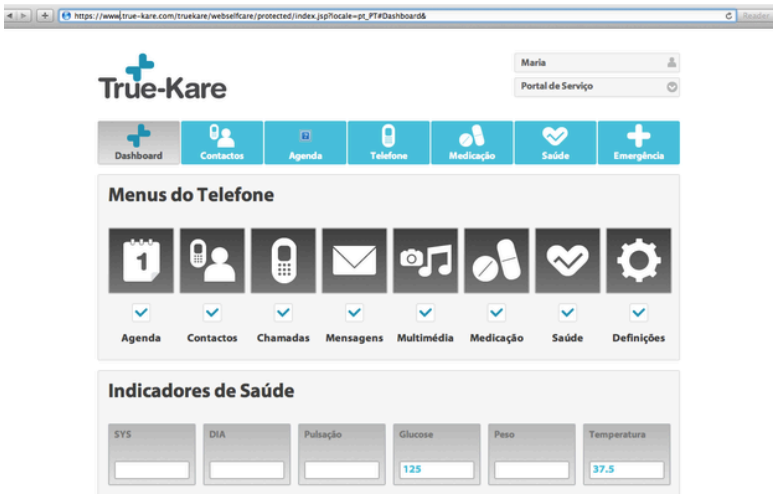


Figura 2.1: Aspeto do *website* do serviço True-Kare

O telemóvel, Figura 2.2, foi desenvolvido pela própria empresa, tem por objetivo ser o interface do sistema para o sénior que está a ser acompanhado. As características do telemóvel são:

- Visor a cores com teclas grandes e separadas;
- Rádio, Global Positioning System (GPS), *Bluetooth*, Câmera, Vídeo, Lanterna;
- Tecla de SOS no telefone configurável a partir do *website*;
- Som ALTO compatível com aparelhos auditivos;
- Avisos de agenda e medicação.

Figura 2.2: Telemóvel True-Kare

Esta é uma solução nacional, que se encontra disponível em 29 países da União Europeia (UE). Apesar de não estar explicito no site [7] da empresa pode-se concluir quais as tecnologias envolvidas neste sistema. A capacidade de ligação à Internet é indispensável para configurar os menus do telefone e para trocar informações com o servidor. O GPS é utilizado para obter a localização do dispositivo podendo controlar-se assim as áreas sobre as quais o sénior pode andar. Supõe-se que o *hardware* esteja conectado ao telemóvel via *Bluetooth*, permitindo a monitorização de vários aspetos como o Índice de Massa Corporal (IMC), temperatura do corpo, peso e pressão arterial. O uso de alertas SMS deixa a entender que o servidor está dotado de um terminal Global System for Mobile Communications (GSM) para o envio das mensagens escritas.

2.1.2 onAll

O onAll é um produto da empresa *Bluecaring*, esta é uma solução igualmente desenvolvida em Portugal. Consiste num sistema fácil de usar para a população sénior. É composto por um conjunto de vários sensores que permitem manter informação em tempo real sobre a condição do utilizador, permitindo assim ao *caregiver* atuar de forma mais rápida e eficaz.

O sénior utiliza uma pulseira ou um cinto, de modo a que consiga movimentar-se livremente enquanto que os dados sobre o seu estado são enviados continuamente. No caso de algum evento de risco, como é o caso de batimentos cardíacos anormais, perigo de queda,

uma localização estranha ou acionado o alarme através do botão de pânico, o *caregiver* mais próximo é alertado com toda a informação sobre o idoso em perigo, permitindo o auxílio.

Todos os alarmes são personalizáveis, para que, por exemplo, os níveis de tensão que despoletam o alarme sejam ajustados às necessidades de cada pessoa. Os sensores permitem, além de um controlo em tempo real, manter um registo no servidor de cada utilizador [8].

Este sistema não se destina apenas aos pacientes, é fomentada uma maior comunicação entre os *caregivers* e os pacientes, criando uma rede na qual o paciente é beneficiado dada a rapidez de resposta a emergências. As famílias também saem beneficiadas porque sabem de antemão que o seu familiar está a ser constantemente monitorizado por uma equipa competente e que os dados sobre a sua condição estão a ser armazenados criando-se um histórico médico relevante.

O sistema foi desenhado para a utilização em casas particulares, em lares ou hospitais. A Figura 2.3 organiza numa pirâmide, as interligações e a gestão deste sistema. No topo encontra-se o sistema de monitorização e gestão no qual os dados dos clientes e dos *caregivers* são interpretados. No caso de uma emergência o sistema alerta o *caregiver* mais próximo da ocorrência para permitir um auxílio mais rápido, representado pela faixa do meio da pirâmide. A base da pirâmide representa o alerta gerado através sensores utilizados pelos idosos. O alerta é feito de modo eficiente porque todas as informações do paciente foram disponibilizadas ao *caregiver* no nível anterior da pirâmide. No final do processo o sistema facilita a elaboração de relatórios e recolha de dados para ser possível a criação de um historial de cada utilizador.

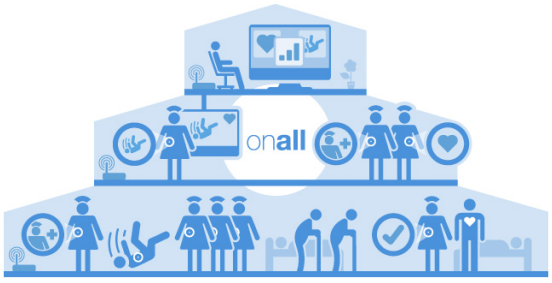


Figura 2.3: Modelo geral do onAll [1]

2.1.3 Prescrição móvel

No que diz respeito a aplicações de NFC em AAL, alguns autores propuseram a criação de um sistema de gestão de receitas médicas digitais [3].

Neste modelo os utilizadores do serviço poderiam pedir novas receitas médicas a partir de suas casas sendo o pedido reencaminhado para o médico de família que, através da aplicação desenvolvida, geraria uma nova receita médica conforme o pedido do seu cliente. Posteriormente os medicamentos poderiam ser levantados numa farmácia normalmente e incluiriam uma *tag* NFC contendo a informação da receita médica, quando deve ser tomado o medicamento e a quantidade que deve ser tomada.

Este sistema incluiria também *software* para o telemóvel que permitiria ao utilizador consultar as informações do medicamento, isto é, se é necessário tomar o medicamento ou não mas para isto o telemóvel teria que estar munido da tecnologia NFC.

Esta solução daria uma maior independência aos seus utilizadores mas ao mesmo tempo iria ser difícil a sua implementação uma vez que depende de vários fatores externos como a adesão ao serviço por parte de farmácias e do Serviço Nacional de Saúde.

2.1.4 Resumo das soluções de *healthcare*

Tabela 2.1: Tabela comparativa de soluções de *healthcare*

Serviço	**Modularidade**	**NFC**	**Serviço localização**
True-Kare	Sim	Não	Sim
OnAll	Sim	Não	Sim
Prescrição móvel	Não	Sim	Não
Serviço a desenvolver	Sim	Sim	Não

2.2 Sistemas operativos móveis

O aparecimento dos primeiros *smartphones* e a massificação/banalização da Internet transformaram um dispositivo cuja função era interligar pessoas por meio escrito ou de voz, num

dispositivo equiparável a um computador, com a vantagem de ser ultra portátil. Abriram-se assim portas a um mercado de novas soluções tecnológicas.

Mas para que isto tudo fosse possível houve a necessidade de uma evolução quer ao nível de *hardware*, quer de *software*, e é neste contexto que apareceram e se desenvolvem os SO móveis.

O primeiro SO móvel foi o Palm OS, apareceu em 1996, permitia a criação de listas de afazeres, gestão da lista de contactos. As versões seguintes acompanharam a evolução tecnológica permitindo o acesso ao e-mail e suporte de cores de 8-bits.

O Windows mobile apareceu e trouxe consigo novidades na ligação à internet e aplicação para leitura de conteúdo multimédia, Windows Media Player, e conexão com o MSN Messenger em 2002.

A integração de *bluetooth* e wi-fi trouxeram novas versões do Palm OS e do Windows mobile em 2003.

No ano de 2005 surge uma nova empresa de *smartphones* a Blackberry com o seu SO próprio e no ano de 2007 é lançado o iPhone.

2008 é o ano onde todos os SO atuais aparecem, é lançado o Symbian OS1 e a primeira versão do Android 1.0. Os SO já existentes ganharam também novas versões do iPhone Operative System (iOS), Windows mobile e Blackberry OS [9, 10].

Nesta dissertação apenas vão ser abordados o iOS [11], o Windows phone e o Android por serem aqueles mais direcionados a uma quota de mercado maior e também por serem os mais utilizados atualmente [12].

2.2.1 iOS

No ano de 2007 a Apple revolucionou o mercado das telecomunicações anunciando uma proposta ousada de um telemóvel inteligente totalmente controlado pelo toque, e com ele surgiu também o *iPhone Operative System*. Contrariamente ao que sucede com outras empresas, a Apple não permite que o seu sistema operativo seja instalado em qualquer outro dispositivo que não seja criado por si. Portanto quando se fala do iOS e do iPhone,

Figura 2.4, os dois estão interligados de tal modo que se trata apenas de um sistema.

Atualmente o iOS encontra-se na versão 7. Ao longo dos anos o SO foi sofrendo várias modificações tendo em vista sempre a inovação, exemplo disso é a assistente pessoal de voz Siri, que permite controlar o *smartphone*, agendar eventos, fazer chamadas e pesquisar conteúdos através de voz. Esta funcionalidade não se encontra disponível na língua portuguesa [13].

Mas apesar dos grandes desenvolvimentos no nível de *software*, o *hardware* escolhido pela marca é sempre algo obsoleto, trazendo poucas melhorias em relação às versões anteriores e quando comparado com outros *smartphones* da mesma gama, o iPhone tende a ficar aquém das expetativas. Na versão mais recente do iPhone foram deixados de fora o NFC e o 4G [14, 15].

Dada a concorrência e a disparidade de preços de equipamentos existentes no mercado, o iPhone tem vindo a perder quota [16], e consequentemente o seu SO tem vindo a ser menos utilizado.

Figura 2.4: Aspecto do iPhone 5

Oficialmente o desenvolvimento de aplicações para iOS é feito exclusivamente no sistema operativo da Machintosh e é recomendado o uso do Xcode pela comunidade de *developers* [17], a linguagem de programação utilizada é o objective-C [18].

Para além do Xcode é necessária a instalação do Software Development Kit (SDK) do iPhone que mune o sistema de todas as bibliotecas necessárias para a programação. Existem vários tutoriais que permitem ter um primeiro contacto com todo este ambiente de programação numa linguagem, sistema operativo e Integrated Development Environment (IDE) diferentes [19].

2.2.2 Windows phone

Windows phone é o SO móvel desenvolvido pela Microsoft numa tentativa de acompanhar e conquistar uma posição no mercado atual de *smartphones* e tal como o Android, este SO não se destina apenas a uma marca de *smartphone*, é assim um SO de uso genérico. O antecessor é o Windows mobile que é um SO voltado para o mundo empresarial, ao passo que o Windows phone está, como os seus concorrentes, virado para mercado de massas.

O ambiente gráfico adotado é "simple. clean. modern" [1], constituído por mosaicos, figura 2.5 de conteúdo dinâmico tal como acontece com o Windows 8 para se atingir o objetivo de universalidade e compatibilidade máxima entre *smartphones*, *tablets*, computadores e consolas de jogos[2].

Figura 2.5: Aspeto Metro do Windows Phone 8

Para criar aplicações para este SO basta transferir e instalar o seu SDK que inclui o Microsoft Visual Studio Express 2012 para Windows Phone, os *templates* de programação necessários para a criação de novas aplicações e o emulador para o teste. As linguagens de programação utilizadas são o C e o C++, mas para o interface do utilizador é utilizado um ficheiro Extensible Application Markup Language (XAML), uma linguagem declarativa tal como o Extensible Markup Language (XML), e pode ser usada para inicializar objetos e definir as suas propriedades. É uma linguagem de programação que não depende de

[1] Microsoft sobre o interface do utilizador Metro do novo SO móvel
[2] Mais concretamente a nova consola da Microsoft, a Xbox one

nenhuma outra, isto é, pode ser usada com outras linguagens de programação [20].

O desenvolvimentos de aplicações para este dispositivo pode ser feito por qualquer pessoa, a sua publicação na *Store* tem custos, $99 United States Dollar (USD) ao ano para utilizadores regulares ou grátis para estudantes abrangidos pelo projeto Microsoft DreamSpark.

2.2.3 Android

O SO Android é baseado em Linux e conta com uma comunidade de *developers* bastante ativa o que permite que qualquer programador consiga criar as suas próprias aplicações e personalização de *smartphones* tornando-os cada vez mais num objeto único e pessoal porque cada pessoa pode optar por um leque extremamente variado de aplicações.

A versatilidade, a facilidade de manuseio e o preço competitivo tornaram rapidamente este SO no mais utilizado globalmente, ao nível dos dispositivos móveis [4], ultrapassando o seu concorrente mais direto, o iOS.

Esta rápida ascensão no mercado e a facilidade na programação fizeram do Android o sistema operativo escolhido para a implementação desta dissertação. A versão sobre a qual se vai desenvolver é o Android 4.0, não por ser o mais recente mas sim por ser aquele na qual todas as funcionalidades do sistema estão disponíveis e também porque o Android permite retro compatibilidade das suas aplicações.

Desenvolvimento de aplicações

A programação é feita em Java e a Google aconselha o uso do Eclipse para o desenvolvimento das aplicações. O Eclipse está disponível para *download* juntamente com o conjunto de ferramentas necessárias para começar desde logo a programar: o Android Development Tools, que contém uma versão do Eclipse, o SDK do Android, o emulador de Android para testar os programas criados e todas as ferramentas para obtenção de *plugins* e outras versões de Android e drivers.

Quando se programa para Android existem alguns ficheiros obrigatórios e importância

elevada, que contêm informações sobre a própria aplicação e também sobre permissões que o sistema necessita de adquirir.

R.java

É um ficheiro gerado automaticamente pelo compilador que contém informações sobre vários recursos usados no programa; cores utilizadas, *strings*, *arrays*, informações sobre ids de objetos de design gráfico, etc., Figura2.6.

É usado para aceder ao conteúdo dos recursos de uma forma simples e condensada numa mesma classe.

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found.  It
 * should not be modified by hand.
 */

package com.example.nfc_escrever;

public final class R {
    public static final class attr {
    }
    public static final class color {
        public static final int Grey=0x7f050000;
        public static final int White=0x7f050001;
    }
    public static final class drawable {
        public static final int ic_launcher=0x7f020000;
    }
    public static final class id {
        public static final int button1=0x7f080002;
        public static final int button2=0x7f080003;
        public static final int editText1=0x7f080001;
        public static final int menu_settings=0x7f080004;
        public static final int textView1=0x7f080000;
    }
    public static final class layout {
        public static final int activity_main=0x7f030000;
    }
    public static final class menu {
        public static final int activity_main=0x7f070000;
    }
    public static final class string {
        public static final int app_name=0x7f040000;
        public static final int botao=0x7f040003;
        public static final int botao2=0x7f040009;
        public static final int erro_escrita=0x7f040008;
        public static final int hello_world=0x7f040001;
```

Figura 2.6: Aspeto do ficheiro R

Strings.xml

É o ficheiro utilizado para guardar *strings*, definir cores, dimensões. Além de ser um ficheiro onde deve ser colocada esta informação, a sua utilidade vai mais além disso. No caso de existirem vários idiomas de utilização, basta ter um ficheiro Strings.xml para cada idioma

no qual é atribuído para uma mesma variável a frase na língua definida. Torna o código simples e sem longos campos de texto misturados com código.

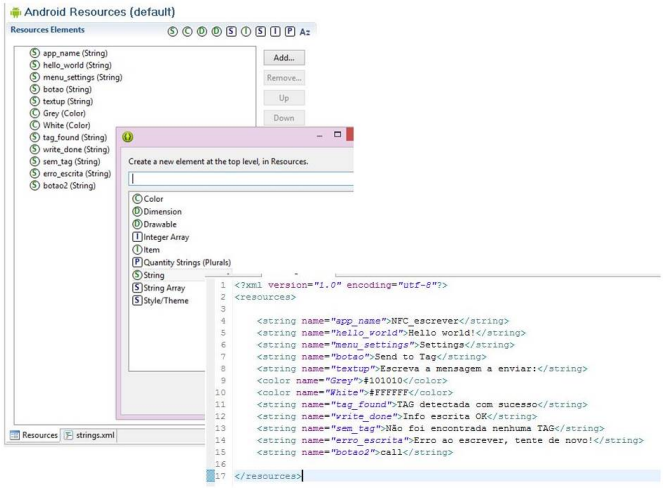


Figura 2.7: Aspecto do ficheiro strings.xml

AndroidManifest.xml

Todas as aplicações têm de ter este ficheiro, Figura 2.8, ele contém informações sobre o nome do projeto Java, descrição dos componentes da aplicação, permissões que necessárias para aceder a conteúdo protegido das Application Programming Interface (API)s do SO, é também indicado o nível mínimo de versão de Android necessária para correr a aplicação.

O SO necessita verificar este ficheiro antes de correr a aplicação, para que assim na altura da execução não existam erros no acesso a APIs protegidas, por exemplo [21].

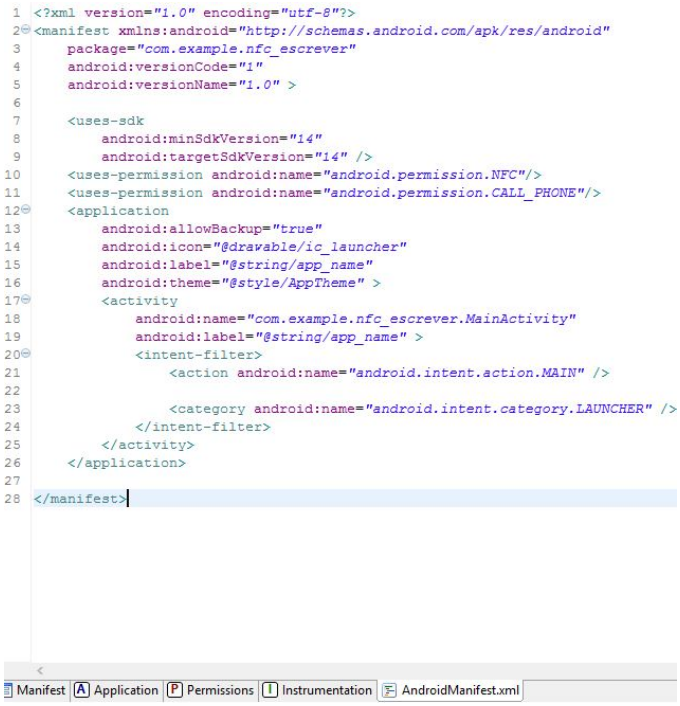


Figura 2.8: Aspeto do ficheiro AndroidManifest.xml

2.2.4 Tabela Resumo dos SO

Tabela 2.2: Tabela resumo de SO móveis

SO	**Número de utilizadores**	**NFC**	**Linguagem de programação**
Android	Elevado	Sim	Java
iOS	Elevado	Não	Objective-C
Windows phone	Baixo	Sim	C++ ; C#

2.3 *Middleware* Android

O objetivo da criação ou utilização de um *middleware* é a criação de um barramento que visa facilitar a expansão e a comunicação entre vários módulos de um projeto de *software*.

2.3.1 OSGi

O OSGi é uma *framework* para Java que permite a implementação de aplicações, ou componentes de aplicação, que podem ser inicializadas, paradas, atualizadas e até desinstaladas sem que o sistema precise ser reiniciado.

As especificações do OSGi começaram em 1998 destinando-se à aplicação no mercado da domótica tendo como objetivo resolver problemas referentes ao desenho de aplicações a partir de componentes independentes [22].

Nos dias de hoje muitas da aplicações não são criadas de raiz mas sim juntando vários projetos *open-source* cujo objetivo não foi inicialmente o de funcionar em conjunto. Por analogia, este foi um dos problemas que o OSGi veio resolver no mercado da domótica. Por esta razão muitos dos projetos têm vindo a utilizar o OSGi para conseguir-se uma estrutura mais fácil de implementar, deixando o programador mais focado em resolver o problema em si [22].

A arquitetura do OSGi é formada por camadas, Figura 2.9.

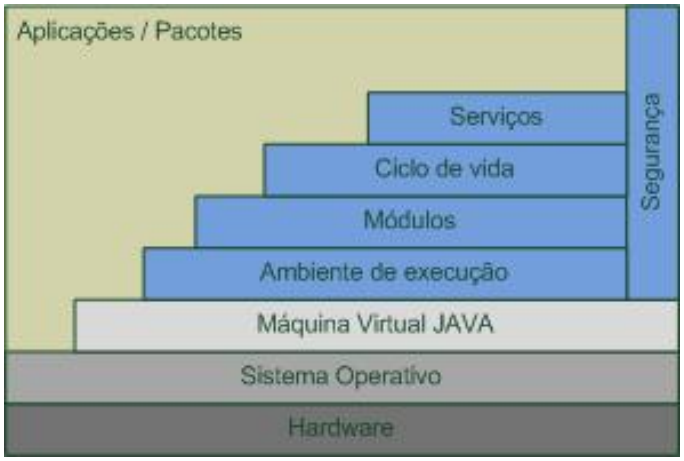


Figura 2.9: Arquitetura do OSGi [2]

As aplicações ou pacotes, são componentes do OSGi desenvolvidos pelo programador é nela que está definida a API a ser divulgada.

A camada de serviços é usada para interligar as aplicações/pacotes de uma forma di-

nâmica.

A camada do ciclo de vida contém a API para iniciar, parar, atualizar, instalar e desinstalar as aplicações.

Módulos é a camada onde se define a forma como as aplicações/pacotes podem importar ou exportar código.

No ambiente de execução estão definidos os métodos e classes disponíveis numa plataforma específica.

A camada de segurança lida com os aspetos relacionados com a segurança limitando acessos a funcionalidades das aplicações/pacotes.

Existem várias implementações do OSGi, as mais referenciadas são: o Equinox [23], o Knopflerfish OSGi [24] e o Apache Felix [25].

2.3.2 Arquitetura de *plugins*

Uma arquitetura de *plugins* é uma das formas de desenvolvimento de aplicações que permite que a aplicação principal possa ser munida de novas funcionalidades, através de *plugins*, sem que seja necessária a atualização total na aplicação principal.

O SO Android tem algumas limitações ao nível de permissões, sempre que se quer aceder a alguma funcionalidade do sistema a permissão tem de ser declarada previamente no ficheiro AndroidManifest.xml [26].

Com os *plugins* é permitido estender as permissões de acesso ao sistema, declarando no *plugin* a permissão necessária e posteriormente instalando o *plugin* no programa principal.

No caso desta dissertação o objetivo passa não só por garantir novas permissões ao ecossistema mas também por incrementar as funcionalidades do ecossistema. Com esta solução de implementação, o sistema torna-se modular podendo ser acrescentadas ou retiradas funcionalidades, de acordo com as necessidades do utilizador. A criação de novas funcionalidades fica também mais facilitada pois todo o processo é parcialmente independente da aplicação principal.

Para se implementar uma aplicação sob a forma de arquitetura de *plugins* é necessário

criar uma aplicação base que estenda uma *Activity* [27], é sobre esta aplicação que se vão inserir os *plugins* em que cada um é uma aplicação que estende um *Service* [28].

Deve ter-se em conta que a aplicação principal e cada um dos serviços devem ter o mesmo nome de pacote, para que o sistema possa identificar a que aplicação pertence o *plugin*. A Figura 2.10 ilustra como se interliga e como se deve conceber uma arquitetura de *plugins* tendo em conta os aspetos referidos.

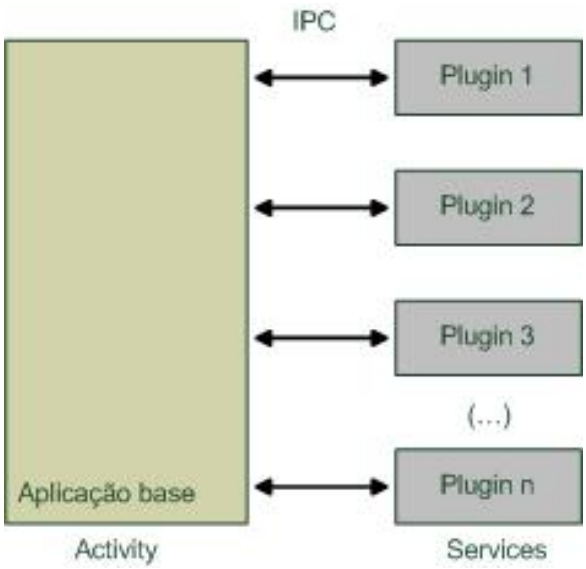


Figura 2.10: Esquema básico de uma arquitetura de *plugins*

Neste contexto uma das questões que se coloca é como efetuar a comunicação entre *plugins* e a aplicação principal e mesmo a comunicação entre os vários *plugins* que possam ser adicionados. Alguns desses métodos de Inter-process Communication (IPC) estão descritos de seguida.

AIDL

O Android Interface Definition Language (AIDL) é utilizado para definir uma interface que será utilizada para que tanto o cliente como o serviço consigam saber como comunicar entre eles usando IPC.

Os processos do SO Android normalmente não podem aceder à memória de outros

processos, para o fazer os objetos têm de ser tornados em tipos básicos do sistema para que o SO os possa compreender e transmitir de um processo para o outro, essa é a função do AIDL.

Para se ligarem dois processos, utilizando o método de AIDL, têm de ser seguidos 3 passos: criação do ficheiro .aidl; implementação da interface; exposição da interface aos clientes.

Este ficheiro contém a interface de vários métodos. São definidos quais os parâmetros de entrada e aquilo que retorna a função. O AIDL suporta os tipos primitivos da linguagem JAVA[3], *String*, *CharSequence*, *List* e *Map*. Para cada tipo extra que se queira utilizar tem de ser importado mesmo que estejam no mesmo pacote que o ficheiro .aidl [29]. Em baixo encontra-se um exemplo de um ficheiro .aidl.

```
1 package aexp.plugin;
2 interface IBinaryOp {
3         int op( in int i1, in int i2 );
4 }
```

A implementação do código da interface tem de seguir algumas regras importantes: as chamadas ao serviço na *thread* principal não são 100% garantidas. Apenas são retransmitidos para quem chamou a função os dados do parâmetro de retorno.

Neste passo é implementado aquilo que a função do serviço executa. O excerto de código apresentado serve como exemplo de um destes ficheiros.

```
1 public class PluginService2 extends Service {
2         static final String LOG_TAG = "PluginService2";
3         static final String CATEGORY_MUL_IF =
              "aexp.intent.category.MUL_PLUGIN";
4
5         public void onStart(Intent intent, int startId) {
6                 super.onStart( intent, startId );
7         }
```

[3] *int, long, boolean, char, etc.*

```
8
9           public void onDestroy() {
10                  super.onDestroy();
11          }
12
13          public IBinder onBind(Intent intent) {
14                  return mulBinder;
15          }
16
17          private final IBinaryOp.Stub mulBinder =
18          new IBinaryOp.Stub() {
19                  public int op( int i1, int i2 ) {
20                          return i1*i2;
21                  }
22          };
23
24  }
```

Este ficheiro contém a implementação da função cujo interface foi definido no ficheiro.aidl, na variável *mulBinder*.

O terceiro passo é referente à divulgação do interface aos clientes que vão utilizar este serviço. Quando o cliente quer aceder ao serviço, utiliza a função *bindService()*; do lado do serviço a função *onServiceConnected()* recebe o parâmetro *mulBinder* que é retornada pelo método *onBind()* do serviço, ver código apresentado na implementação do interface.

Como o cliente e o serviço são aplicações diferentes, o ficheiro .aidl do serviço tem de ser incluído no código do cliente para que assim exista um modo comum de estabelecer a comunicação [30]. Em baixo encontra-se o exemplo de um cliente.

```
1  IRemoteService mIRemoteService;
2  private ServiceConnection mConnection = new ServiceConnection() {
3
4           public void onServiceConnected(ComponentName className, IBinder
               service) {
```

```
5                           mIRemoteService =
                                IRemoteService.Stub.asInterface(service);
6           }
7
8           public void onServiceDisconnected(ComponentName className) {
9                   Log.e(TAG, "Service has unexpectedly disconnected");
10                  mIRemoteService = null;
11          }
12  };
```

Broadcast Intents

Uma das suas funcionalidades é iniciar atividades de uma aplicação, mas para além disso os *Broadcast Intents* pode ser usados para transmitir mensagens para outros componentes do sistema. Esta segunda funcionalidade envolve a implementação de *Broadcast Intents* e de *Broadcast Receivers*.

Os *Broadcast Intents* são utilizados para enviar os *intents* para qualquer outro componente do sistema, cada *intent* tem um identificador único que é inicializado na *string "Action"* e os dados a transmitir são passados como parâmetros *extra*. O excerto de código apresentado serve como exemplo de implementação de um *Broadcast Intent*.

```
1  Intent intent = new Intent();
2  intent.setAction("NOME_EXEMPLO");
3  intent.putExtra("Valor", 100);
4  sendBroadcast(intent);
```

Os *Broadcast Receivers* são utilizados para receber *broadcasts* que outras aplicações enviem. Para serem criados a classe deve estender a classe *BroadcastReceiver* na qual existe uma função *onReceive* que deteta a receção de *broadcasts*. Em baixo tem-se o exemplo de código de um *Broadcast Receiver*.

```
1  public class MyReceiver extends BroadcastReceiver {
2
```

```
3           @Override
4           public void onReceive(Context context, Intent intent) {
5           //Código a executar quando se recebe broadcast
6           }
7   }
```

Através do parâmetro de entrada *intent* podem ser extraídos os valores *extra* que foram adicionados no *Broadcast Intent*.

Para que a aplicação não fique à escuta de todos os *broadcasts*, o ficheiro AndroidManifest.xml tem de ser editado e configurado para filtrar os *broadcasts* que ativam a função [31]. Em baixo tem-se um excerto de código de um ficheiro AndroidManifest.xml.

```
1  <application
2    (...)
3           <receiver android:name="MyReceiver" >
4                       <intent-filter>
5                               <action android:name="NOME_EXEMPLO" >
6                               </action>
7                       </intent-filter>
8           </receiver>
9    (...)
10 </application>
```

2.4 Protocolos de comunicação sem fios

Os protocolos analisados nesta secção dizem respeito a análises de tecnologias de comunicação sem fios, para a comunicação entre dispositivos móveis ou entre dispositivos externos e os dispositivos móveis.

Foram escolhidas três tecnologias que são atualmente incorporadas no *smartphones* à venda no mercado: o NFC, o *Bluetooth* e o Wi-Fi.

2.4.1 *Near Field Comunication*

O NFC é um protocolo de comunicação sem fios de curta distância, desenvolvido numa cooperação entre a *Sony* e a *Phillips* em 2002.

Para funcionar, um dispositivo NFC gera uma onda rádio com uma frequência na ordem dos 13,56 MHz, quando dois dispositivos são colocados a uma certa distância, geralmente 10cm, são transferidos energia e dados através de um campo magnético indutivo e esta é a principal diferença desta tecnologia para outras como o *Bluetooth* e o *Wi-Fi*.

Existem dois tipos de dispositivos NFC, os ativos e os passivos. O primeiro grupo distingue-se por conter uma fonte de energia interna ao passo que os dispositivos passivos caraterizam-se por serem alimentados pelo campo magnético gerado aquando a aproximação a um dispositivo ativo. Nesta aproximação, e como já foi referido, é trocada energia entre os dois dispositivos permitindo assim ao dispositivo passivo trocar dados.

A transferência de dados entre dispositivos NFC é feita a uma velocidade máxima de 420 Kbps, que torna esta tecnologia pouco viável para transferências de grandes quantidades de dados. Como a distância de operação é baixa, a segurança da tecnologia é bastante elevada pelo que a interceção de dados é praticamente impossível [32].

As funcionalidades do NFC fazem com que ganhe um papel importante em vários setores da sociedade. Atualmente existem vários projetos de I&D para a aplicação da tecnologia em pagamentos de serviços [33] que visam facilitar e tornar mais seguras as compras [34]. No caso desta dissertação o âmbito do uso do NFC está diretamente ligado a soluções de prestações de serviços de saúde para população idosa. Nesta área, as características técnicas do NFC são aproveitadas ao máximo desde logo por não ser intrusivo, isto é, a sua utilização não envolve nenhum *know how*, basta aproximar um objeto de outro para se dar automaticamente interação, pelo que o tempo de familiarização é baixo.

NFC no Android

Para se utilizarem as funcionalidades do NFC num *smartphone* Android, deve-se começar pela edição do ficheiro AndroidManifest.xml de modo a conceder permissão ao uso do NFC pelo programa a desenvolver. A informação pode ser armazenada na tag em vários tipos de formato, no entanto a maior parte das APIs do Android utilizam o *standard* NFC Data Exchange Format (NDEF).

As bibliotecas a importar são todas do domínio "android.nfc.", permitem ao programa ler as mensagens NDEF contidas nas tags ou noutros dispositivos com NFC.

Nele estão incluídas as classes apresentadas pela Tabela 2.3 [35].

Tabela 2.3: Classes contidas no ficheiro android.nfc

NdefMessage	Representa uma mensagem Ndef
NdefRecord	Representa um NdefRecord
NfcAdapter	Representa o adaptador NFC local
NfcEvent	Coleta informação associada com qualquer evento que envolva NFC
NfcManager	Usado para obter uma instancia de um NfcAdapter
Tag	Representa uma tag descoberta

2.4.2 *Bluetooth*

O *Bluetooth* é um protocolo de comunicação sem fios que viabiliza as trocas de dados a uma distância curta. Foi desenvolvido pela *Ericsson* por Jaap Haartsten e por Sven Mattisson. Hoje em dia o *Bluetooth* é uma tecnologia totalmente maturada ao ponto que a grande maioria dos dispositivos a incorporam, desde Personal Computer (PC)s a telemóveis passando ainda por uma gama variada de *gadgets* e acessórios.

Ao longo dos anos o protocolo tem vindo a ser alterado de modo a acompanhar a evolução da tecnologia, existindo atualmente 3 classes de *Bluetooth*, classes que definem a potencia máxima utilizada e a distância máxima de comunicação, ver Tabela 2.4 [36]. A versão mais recente deste protocolo, versão 3.0, permite taxas de transmissão de 24 Mbit/s.

Tabela 2.4: Classes de dispositivos de *Bluetooth*

Classe	**Potência Máxima**	**Alcance**
Classe 1	100 mW	100 m
Classe 2	2,5 mW	10 m
Classe 3	1 mW	1 m

***Bluetooth* no Android**

O Android possui um conjunto de funções, API, que permitem ao programador controlar o dispositivo presente no *smartphone*.

A API permite a pesquisa de outros dispositivos ativos na área de alcance, enviar pedidos para emparelhar dispositivos, estabelecer canais para Radio Frequency Communication (RFCOMM), conectar-se a outros dispositivos, transferir data entre dispositivos e gerir múltiplas conexões.

Para usar o *Bluetooth*, tal como acontece com o NFC, é necessário editar o ficheiro AndroidManifest.xml para dar permissões de utilização ao programa. Sendo que existem dois tipos de permissão, BLUETOOTH e BLUETOOTH_ADMIN. Para a aplicação funcionar corretamente pelo menos um destes dois tem de ser declarado, o primeiro diz respeito a permissões relacionadas com qualquer comunicação *Bluetooth* e o segundo diz respeito a permissões de manipulação de definições do *Bluetooth* ou para dar inicio a procura de novos dispositivos.

As bibliotecas a importar estão contidas no pacote "android.bluetooth." e contém uma variedade de classes que permitem o acesso ao *hardware*, Tabela 2.5 [37].

Tabela 2.5: Ficheiro android.bluetooth

BluetoothAdapter	Representa o adaptador Bluetooth local
BluetoothDevice	Representa o dispositivo Bluetooth remoto
BluetoothSocket	Representa a interface para uma Bluetooth socket
BluetoothServerSocket	Representa uma socket do servidor aberta, para escutar pedidos que cheguem
BluetoothClass	Descreve as características de um dispositivo Bluetooth
BluetoothProfile	Interface representativa de um perfil Bluetooth
BluetoothHeadset	Permite o suporte para auriculares de telemóvel Bluetooth
BluetoothA2dp	Define a qualidade do audio que pode ser transmitida por Bluetooth
BluetoothHealth	Representa um Health Device Profile que controla o serviço Bluetooth
BluetoothHealthCallback	É uma classe abstrata utilizada para implementar os BluetoothHealth callbacks
BluetoothHealthAppConfiguration	Representa a configuração registada para comunicar com o dispositivo remoto
BluetoothProfile.ServiceListener	Interface para notificar clientes quando se (des)conectaram do serviço

2.4.3 Wi-Fi

O Wi-Fi é uma tecnologia de comunicação sem fios, que permite que um dispositivo eletrónico troque dados e se conecte à Internet usando par isso uma tecnologia rádio denominada de Institute of Electrical and Electronics Engineers (IEEE) 802.11 que fornece segurança, confiança e velocidades de comunicação elevadas. [38].

Existem variações da tecnologia IEEE 802.11 que alteram a largura de banda de operação e a frequência, a Tabela 2.6 sumariza essas mesmas diferenças.

Tabela 2.6: Tecnologias Wi-Fi

Tecnologia	Frequência	Largura de banda
802.11a	5 GHz	54 Mbps
802.11b	2.4 GHz	11 Mbps
802.11n	2.4 e 5 GHz	450 Mbps
802.11g	2.5 GHz	54 Mbps

O Wi-Fi é um protocolo universal pelo que produtos de um fabricante conseguem comunicar com os de outro qualquer fabricante.

Dada a sua capacidade de cobrir uma vasta área, um dos aspetos chave desta tecnologia é a segurança, existindo três formas de encriptação de chaves da rede:

1. Wired Equivalent Privacy (WEP) pode ter vários níveis de segurança, com chaves de tamanho variável, 64, 128 ou 256 bits;

2. Wired Protected Access (WPA) contém chaves que são trocadas periodicamente sendo que a sequencia é definida na configuração da rede;

3. WPA2 é uma variação do WPA, é o mais seguro dos 3, mas tem como desvantagem o alto processamento necessário.

Wi-fi no Android

Com a API disponibilizada, "android.net.wifi", as aplicações podem comunicar com um nível mais baixo de abstração para que lhes seja concedido acesso à rede Wi-fi. Quase todas as informações do dispositivo estão disponíveis, incluindo velocidade da rede, endereço Internet Protocol (IP) e informação de outras redes que estejam disponíveis. É permitido ao programador, procurar, adicionar e guardar uma rede bem como iniciar ou terminar ligações Wi-fi.

Tal como nos exemplos anteriores o ficheiro AndroidManifest.xml tem de ser editado de modo a obterem-se permissões de uso de Wi-Fi. No caso da utilização de algumas APIs são

necessárias outras permissões do sistema ACCESS_WIFI_STATE, CHANGE_WIFI_STATE, CHANGE_WIFI_MULTICAST_STATE [39].

Tabela 2.7: Ficheiro android.net.wifi

ScanResult	Informação sobre um ponto de acesso detetado
WifiConfiguration	Representação de uma rede Wi-fi configurada
WifiConfiguration.AuthAlgorithm	Algoritmos de autentificação IEEE 802.11
WifiConfiguration.GroupCipher	Grupo de cifras conhecidas
WifiConfiguration.KeyMgmt	Esquemas de gestão de chaves conhecidos
WifiConfiguration.PairwiseCipher	Cifras para WPA
WifiConfiguration.Protocol	Protocolos de segurança conhecidos
WifiConfiguration.Status	Estado de uma configuração de rede
WifiInfo	Descreve o estado de conexões Wi-fi ativas
WifiManager	API para gerir todos os aspectos de uma conexão Wi-fi
WifiManager.MulticastLock	Permite à aplicação receber pacotes do tipo Wifi Multicast
WifiManager.WifiLock	Permite à aplicação manter o Wi-fi ligado
WpsInfo	Classe representativa de um setup de protecção Wi-fi

2.5 Serviços Web

Os serviços web estão a impulsionar a Web 2.0 [40] com o conceito de colocar na web *software* que convencionalmente estaria no PC. O seu objetivo é a comunicação de aplicações através de rede, quer seja ela Local Area Network (LAN) ou Internet. Associado aos serviços web está o modelo de cliente-servidor. Este modelo permite que vários serviços estejam alojados num servidor sendo o seu acesso feito através de pedidos por parte dos clientes [41]. A Figura 2.11 demonstra algumas das propriedades dos serviços web.

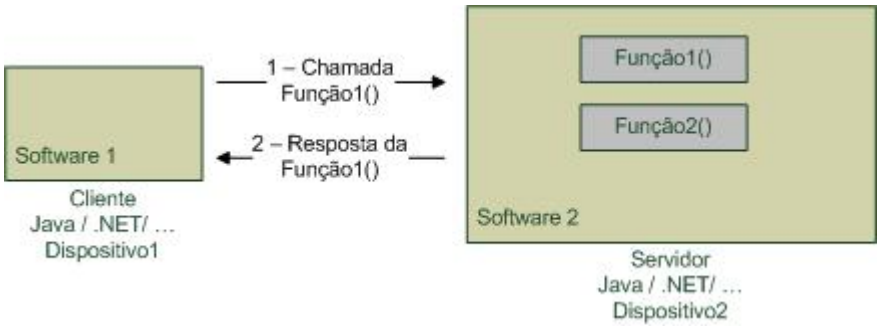


Figura 2.11: Propriedades dos serviços web

Os serviços web são portanto funções alojadas num servidor, que responde a pedidos de clientes, permitindo o acesso a aplicações de *software* a uma determinada rede. Uma das muitas vantagens de usar serviços web é a interoperabilidade de linguagens de programação, isto é, o cliente pode ser feito em Java e o servidor estar codificado em .NET, por exemplo. Na figura 2.11 é de destaque que o cliente e o servidor não precisam de estar no mesmo dispositivo, precisam apenas de um meio de comunicação LAN ou Internet [42].

Para que esta comunicação, independente de linguagens ou dispositivos, seja possível é necessário que o protocolo de comunicação seja feito de uma forma bem definida, sendo os dois mais utilizados o Simple Object Access Protocol (SOAP) e o Representation State Transfer (REST).

2.5.1 SOAP

O SOAP é um protocolo de comunicação de dados, ou pequenas quantidades de informação, através da Internet. No SOAP as mensagens circulam no formato XML usando Hypertext Transfer Protocol (HTTP) ou Simple Mail Transfer Protocol (SMTP).

As mensagens encontram-se num SOAP *envelope* que contém cabeçalho e o corpo da mensagem, o excerto de código apresentado de seguida ilustra o exemplo de um *envelope*.

```
1 <SOAPenv:Envelope
2         xmlns:SOAPenv="http://schemas.xmlSOAP.org/SOAP/envelope/"
3         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
4           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
5           <SOAPenv:Body>
6                   <req:getNumberOfArticles
7                           xmlns:req="http://daily-moon.com/CMS/">
8                           <req:category>classifieds</req:category>
9                   </req:getNumberOfArticles>
10          </SOAPenv:Body>
11  </SOAPenv:Envelope>
```

São estas mensagens que circulam numa comunicação cliente-servidor. O sistema que a recebe interpreta a mensagem, processa-a, e envia de volta um novo SOAP *envelope* com a resposta ao pedido que recebeu.

O cabeçalho, fornece informação sobre a mensagem em si. Pode conter informações sobre o endereço do destinatário da mensagem e para que endereço deve seguir a resposta. O cabeçalho contém estas várias informações porque segundo o protocolo SOAP, nem em todos os casos a mensagem vai diretamente do cliente para o servidor. Muitas vezes a mensagem pode passar e ser processada por vários sistemas antes de chegar ao destinatário final, e em cada um destes intermediários a mensagem pode ser modificada consoante o que esteja descrito no cabeçalho do SOAP *envelope*.

O corpo da mensagem, contém a mensagem ou instruções a serem entregues no destinatário da mensagem [43].

2.5.2 REST

O REST consegue tirar partido de todas as funcionalidades da web aliando isto com a sua simplicidade fazem do REST uma das opções mais utilizadas na implementação de serviços web, como se tem notado pela mudança da Yahoo, Google, e Facebook, que abandonaram o SOAP e o Web Services Description Language (WSDL) para adotar o REST [44].

Para se compreender o REST têm de ser entendidos 3 conceitos que constituem a base da tecnologia.

Resource, ou recurso, define tudo aquilo que seja importante ao ponto de ser referenci-

ado, pode ir desde um objeto físico a um conceito abstrato, mas normalmente é algo que possa ser armazenado num computador e representado por um conjunto de bits.

Representation, ou representação, contém informação útil sobre o estado de um recurso. Cada recurso pode ter múltiplas representações.

State, ou estado, é composto por 2 tipos, o *resourse state*, estado do recurso, e pelo *application state*, estado da aplicação. O primeiro diz respeito a informação sobre o recurso e permanece no servidor, o segundo refere-se ao caminho que o cliente tomou através da aplicação e apenas existe no lado do cliente.

A Figura 2.12 ilustra as ligações entre o cliente e o servidor numa arquitetura de serviços web do tipo RESTful [45].

Figura 2.12: Arquitetura de serviços web RESTful

No REST o servidor é abstraído como sendo um conjunto de recursos, cada recurso é identificado com um Uniform Resource Identifier (URI) sabendo-se sempre ao que se está a aceder, pelo que a estrutura deve ser simples e previsível.

http://(myServiceAddress)/RestGesp/service/alarm

Este exemplo mostra que se vai aceder ao serviço de alarmes da aplicação, sendo o URI previsível com o qual qualquer pessoa pode facilmente perceber qual o recurso a aceder. Os recursos podem transitar em vários formatos, sendo os mais comuns XML e Javascript Object Notation (JSON).

Tal como está representado na Figura 2.12 o REST faz uso dos métodos HTTP. Quando se quer criar um recurso no servidor, deve-se usar o método POST, para obter um recurso GET, para alterar o estado de um recurso ou para atualiza-lo deve-se usar PUT ou

UPDATE, para remover um recurso do servidor deve ser usado o método DELETE [44].

2.5.3 Tabela resumo dos serviços Web

Após serem analisadas duas das mais utilizadas arquiteturas de criação de serviços web, podem ser resumidas as suas principais características na Tabela 2.8.

Tabela 2.8: Tabela resumo de serviços web

Arquitetura	Simplicidade	Overhead	Performance	Modos comunicação
REST	X	Nenhum	X	
SOAP		Cabeçalho		X

2.6 Resumo do estado da arte

A análise de tecnologias existentes no mercado permite tomar decisões sobre as tecnologias a utilizar no decorrer desta dissertação. O SO Android deve ser o escolhido por ser o mais utilizado. A variedade de equipamentos móveis que utilizam o SO é um fator que faz com que haja uma diversificação de preços permitindo que a aquisição de *smartphones* esteja acessível a toda a população. No que diz respeito a tecnologias, os *smartphones* com o SO Android estão na vanguarda da tecnologia, havendo já um leque considerável de equipamentos com todas as tecnologias estudadas neste capitulo, NFC, *Bluetooth* e Wi-Fi.

No caso dos serviços web a solução mais utilizada e de mais facilidade de implementação e utilização é o REST.

Capítulo 3

Análise do problema

Como já foi referido nesta dissertação, pretende-se utilizar a tecnologia NFC como mecanismo de interação entre o utilizador e a plataforma computacional, bem como na utilização de dispositivos externos que comuniquem com esta. Isto permite a criação de um ecossistema de escala variável para a execução de aplicações e utilização de dispositivos.

O ambiente alvo de utilização é a casa de um idoso, onde este interage com a tecnologia quase sem dar por isso. Por um lado, pretende-se transformar a plataforma/equipamento móvel num sistema ubíquo de apoio aos idosos nas tarefas diárias que possam trazer alguma dificuldade de utilização.

3.1 Requisitos do sistema

O objetivo principal desta dissertação é a criação de um ecossistema de escala variável, isto é, consoante a necessidade, o seu tamanho pode aumentar ou diminuir, assim como funcionalidades podem ser acrescentadas ou retiradas sem que o funcionamento do sistema, no geral, seja afetado negativamente.

Concorrentemente o sistema tem de se enquadrar no âmbito de AAL e fazer uso da tecnologia NFC. Deste modo foram criadas algumas aplicações para o SO Android que cumprem os requisitos enunciados anteriormente.

Criou-se uma lista de funcionalidades consideradas importantes e pertinentes, cuja utilização visa o aumento da qualidade de vida de pessoas idosas:

- Efetuar chamadas de uma forma automática, recorrendo à tecnologia NFC incorporada no *smartphone*;
- Munir os medicamentos de tags NFC, de modo a que de uma forma facilitada se possam saber informações sobre o medicamento;
- Munir dispositivos eletrónicos médicos de tags NFC, de modo a que seja possível a obtenção dos *drivers* correspondentes;
- Criar um conceito de botão de pânico que permita ser acedido de forma rápida e que providencie informação relevante para quem vigia o sistema;
- Criar um mecanismo de alerta para a toma de medicamentos;
- Criar um sistema que permita detetar uma queda e providenciar informação relevante para quem vigia o sistema;
- Informar sobre atividade ou não atividade da pessoa idosa a ser vigiada;
- Desenvolver uma plataforma web que permita gerir todo o sistema, na qual seja possível editar e resolver alertas pendentes e visualizar informações sobre clientes do sistema;

3.2 Casos de uso

Uma análise dos requisitos do sistema permite destacar três atores que interagem com o sistema: o utilizador do *smartphone*, o utilizador que gere todo sistema, que pode ser um médico ou um cuidador designado para o utilizador do *smartphone*, e ainda o familiar.

Cada um dos atores pode desempenhar funções diferentes no sistema.

3.2.1 Utilizador

Ao utilizador é permitido realizar algumas ações utilizando a tecnologia NFC disponível no dispositivo. Com essa característica o utilizador pode fazer chamadas telefónicas encostando o seu *smartphone* a uma fotografia munida de uma *tag* NFC; a informação sobre os medicamentos pode ser consultada de forma idêntica à forma de fazer uma chamada telefónica. A utilização de equipamentos externos é outra forma de utilização do NFC, fazendo o download da aplicação correspondente ao hardware a usar.

O utilizador pode também ler as notificações que são enviadas através de um dos serviços web.

No caso de uma queda, apesar de esta não ser uma interação direta com o *smartphone*, é feita uma chamada para um dos familiares designados e será dado o alerta no *website*. A Figura 3.1 é referente aos casos de uso do utilizador do sistema.

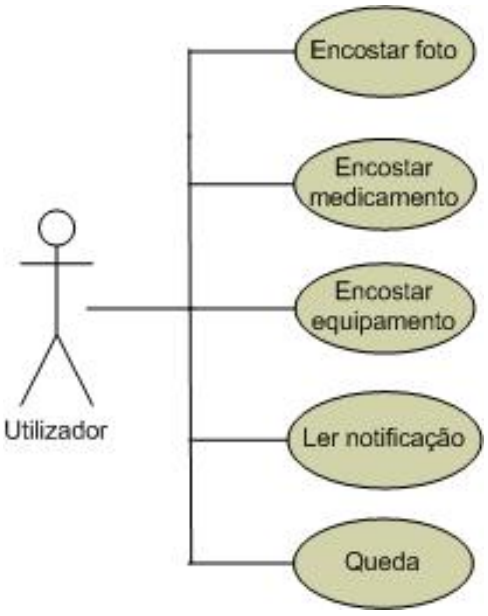


Figura 3.1: Caso de uso do utilizador

3.2.2 Gestor

Ao gestor é permitido efetuar as suas ações através do *website*. Ele pode editar e visualizar as informações dos seus utilizadores, editar as horas de aviso e os medicamentos de cada um dos seus utilizadores. Para além disto é-lhe possibilitada a visualização de alertas relativos aos seus utilizadores.

A Figura 3.2 é referente aos casos de uso do gestor.

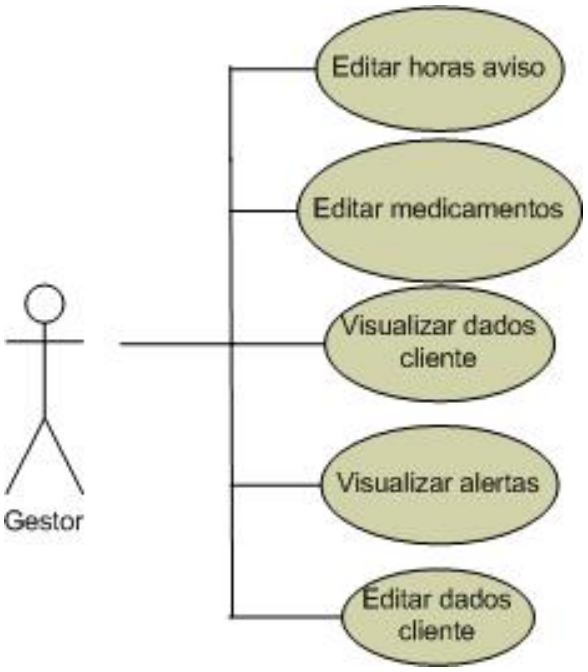


Figura 3.2: Caso de uso do gestor

3.2.3 Familiar

O familiar é alguém que pertença à família do utilizador e que tenha um papel ativo nos cuidados prestados ao utilizador.

Ao familiar é permitida a visualização dos dados e alertas referentes ao seu familiar e também a edição dos seus dados pessoais.

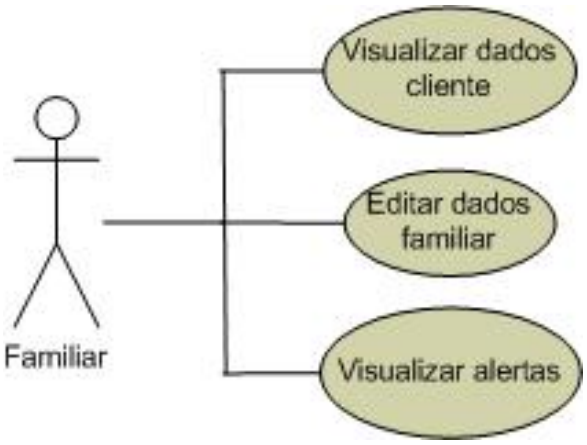


Figura 3.3: Caso de uso do familiar

3.3 Ferramentas utilizadas

3.3.1 Maven

O Maven é um conhecido gestor de dependências integrável com o Eclipse. O seu propósito é facilitar o processo de compilação, gestão e integração de projetos, tornando a gestão de um projeto complexo numa atividade simples pois a gestão das bibliotecas Java é feita automaticamente pelo Maven [46].

A base desta ferramenta é o Project Object Model (POM), um ficheiro XML que contém uma descrição detalhada do projeto incluindo a versão do projeto, as várias dependências, módulos externos e até a ordem pela qual os ficheiros devem ser compilados.

As bibliotecas Java e os *plugins* são obtidos dinamicamente a partir da lista de repositórios do próprio Maven. Como nem em todas as fases do projeto são necessários todos os ficheiros importados, alguns deles servem, por exemplo, para fins de teste da aplicação, o Maven permite otimizar esta situação, podendo ajustar-se o *scope* das dependências para que estas sejam apenas utilizadas quando necessário. O *scope* pode ser ajustado para 4 situações:

- *compile*: permite que a dependência esteja disponível em todas as fases;

- *provided*: quando a dependência é utilizada apenas no momento da compilação;
- *runtime*: para quando a dependência é apenas necessária na execução da aplicação;
- *test*: quando a dependência é utilizada para compilar testes.

3.3.2 Hibernate

Durante o processo de desenvolvimento de um projeto de *software* que envolva uma base de dados relacional e uma linguagem orientada a objetos, o programador vê-se obrigado a despender muito tempo a estabelecer relações entre as várias classes e as várias tabelas da base de dados. Este tempo tem custos no produto final.

O Hibernate é uma Object Relational Mapping (ORM) *tool*, ou seja, é uma ferramenta que permite fazer o mapeamento de objetos Java numa base de dados e vice versa, Figura 3.4.

Figura 3.4: Funcionamento do Hibernate

A vantagem do Hibernate é criar uma abstração da base de dados permitindo ao programador focar-se no seu código orientado a objetos e deixar de lado as *queries* Structured Query Language (SQL). As *queries* são geradas pelo Hibernate e são otimizadas para produzir resultados o mais eficientemente possível. O código final da aplicação torna-se mais simples e legível pois deixa de conter linguagem SQL embebida [42].

3.3.3 Quartz

O Quartz é um *job scheduler*, ou seja, permite agendar tarefas para horas ou dias específicos. Disponibilizando uma biblioteca fácil de integrar e utilizar em Java. Pode ser incluído em qualquer aplicação Java, e para além disso a sua utilização é gratuita [47].

De acordo com o seu website [48] a utilização é de tal modo simples que pode ser resumida nos passos apresentados na Figura 3.5.

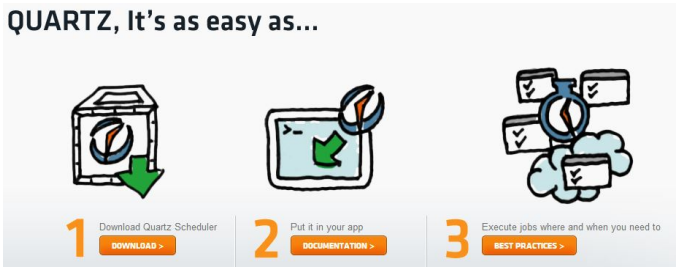


Figura 3.5: Passos para utilização do Quartz segundo o seu website

Uma das características do Quartz é ser um sistema com tolerância a falhas, permitindo recuperar a execução de tarefas entre reinicializações do sistema, isto é, o contador do tempo não se perde após falha do sistema.

3.4 Diagrama do sistema

Atendendo aos requisitos do sistema e após o estudo de várias ferramentas inseridas no âmbito desta dissertação, é possível criar o diagrama da Figura 3.6 que ilustra os componentes do sistema bem como as suas interligações.

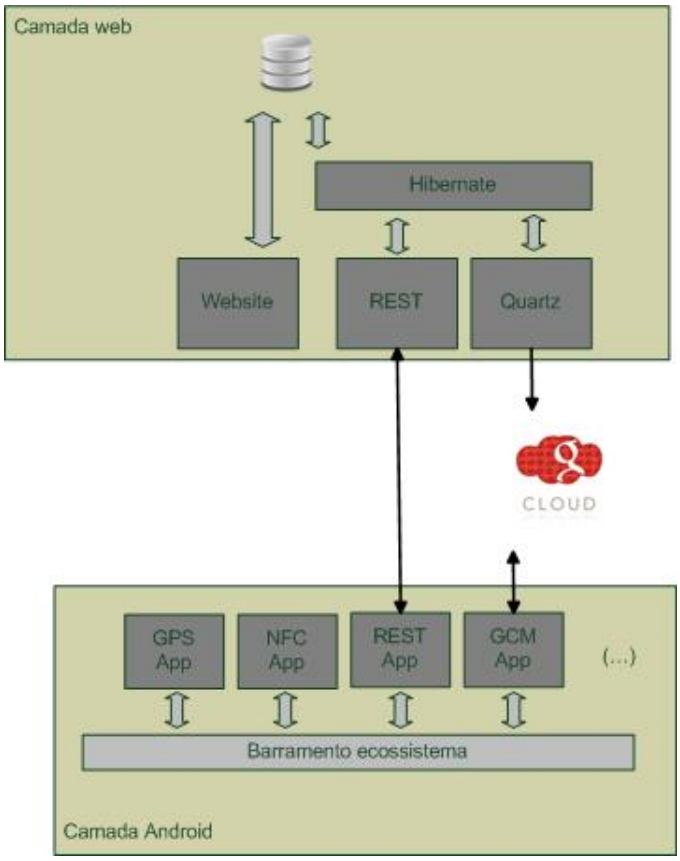


Figura 3.6: Diagrama de blocos do sistema

Desde logo se evidenciam dois blocos distintos do sistema, a camada web e a camada Android. A camada web está alojada num servidor web e é composta pelos serviços RESTful, onde são processados os pedidos enviados pelo lado da camada Android; o *website*, no qual os utilizadores com credenciais de autenticação podem visualizar e editar informação sobre clientes; e o Quartz, *job scheduler*, que permite agendar o envio de alertas para o *smartphone* do cliente. Os acessos à base de dados quer pelo serviços RESTful quer pelo Quartz, são geridos pelo Hibernate. Por outro lado o website acede diretamente a base de dados alojada no servidor.

Na camada Android só estão representadas as aplicações desenvolvidas no âmbito desta dissertação, pois a camada Android é composta pelo SO e todas as outras aplicações, *drivers* e máquina virtual que dizem respeito ao SO Android.

No que diz respeito ao ecossistema, é possível analisar a sua variabilidade, a aplicação base é o barramento do ecossistema no qual se encontram definidos os meios de comunicação entre os vários módulos instaláveis do sistema. Os módulos instaláveis, *plugins*, são anexados ao barramento e passam a funcionar como uma única aplicação. Alguns dos exemplos dos módulos encontram-se ilustrados na Figura 3.6.

3.5 Diagrama de sequência

3.5.1 Website

Login

O *login* no *website* pode ser feito por um familiar registado ou por um gestor de conta. Cada um destes dois tipos tem uma página, depois da autenticação, com funcionalidades diferentes mas com o mesmo processo de *login*.

A situação de *login* apenas pode acontecer se o familiar ou gestor não estiverem já com o sessão iniciada. Para o fazer, cada utilizador insere as suas credenciais de *login* no *website*, que depois vão ser autenticadas para que seja mostrada a página correspondente, ver Figura 3.7.

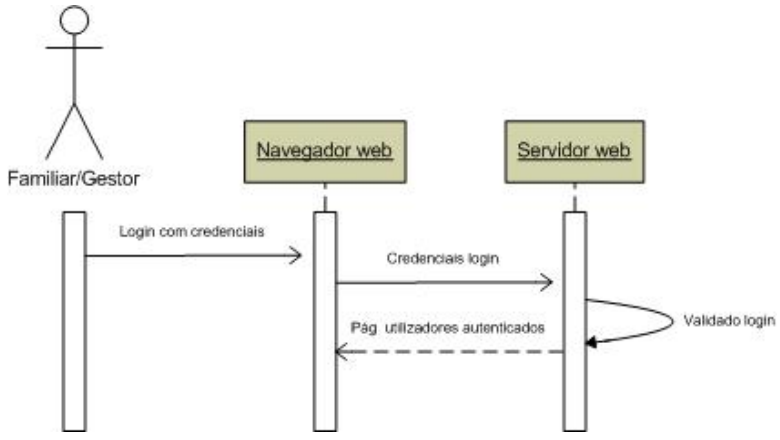


Figura 3.7: Diagrama de sequência da ação de login no website

Logout

A situação de *logout* apenas pode ocorrer depois de um familiar ou gestor terem efetuado *login*. Trata-se da situação inversa do *login*, em que uma página de utilizadores não autenticados é carregada no navegador *web*, Figura 3.8.

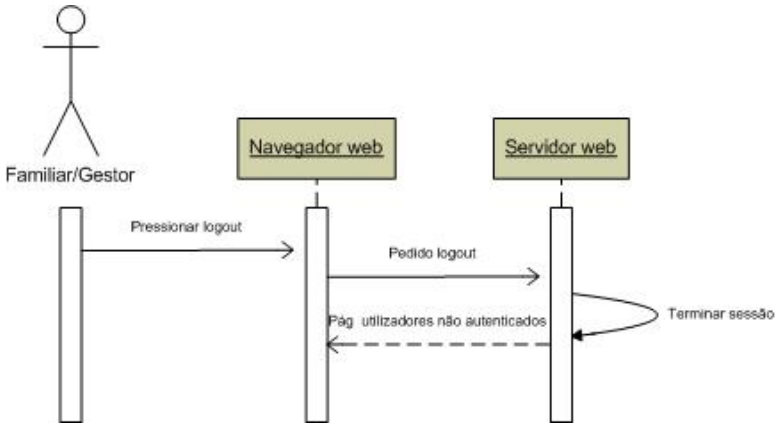


Figura 3.8: Diagrama de sequência da ação de logout no website

Diagrama geral de pedidos

Em todos os pedidos de dados por parte do *website* o procedimento é sempre o mesmo, são feitas *queries* a base de dados e o resultado é mostrado a quem esteja a aceder ao *website*. O diagrama de sequência é ser sempre idêntico para todos os pedidos, variando apenas a informação acedida, pelo que pode ser generalizado pelo diagrama da Figura 3.9.

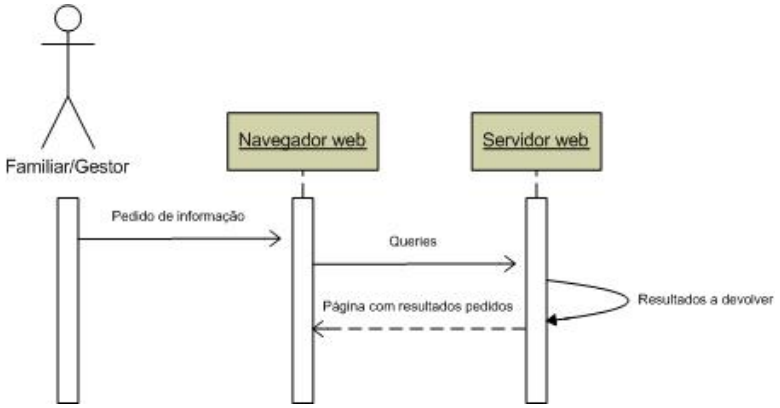


Figura 3.9: Diagrama de sequência genérico para o *website*

3.5.2 Aplicação Android

Registo de novo cliente

O registo de um novo cliente só é possível através da aplicação Android. O *caregiver* encarrega-se de fazer o registo preenchendo os dados pessoais do idoso. Estes dados são enviados para o serviço web, que os processa e retorna um resultado de confirmação de registo com sucesso ou erro, Figura 3.10.

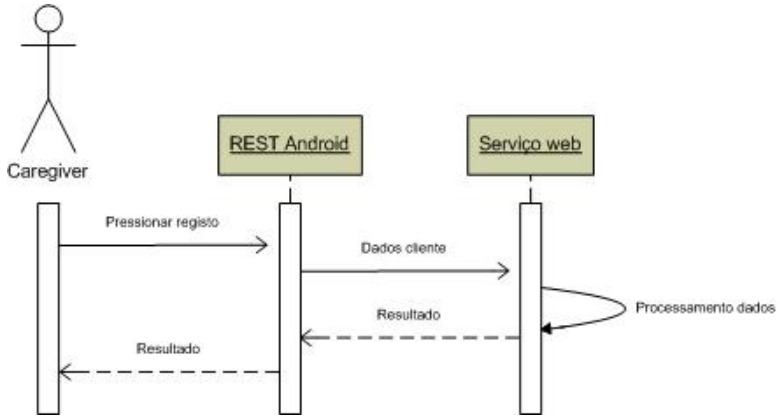


Figura 3.10: Diagrama de sequência da ação de registo na aplicação Android

Login

A situação de *login* ocorre sempre antes da aplicação ser iniciada. As credenciais de *login* geradas no momento do registo são enviadas para o serviço web que as verifica e envia de volta um resultado de confirmação de *login* com sucesso, Figura 3.11.

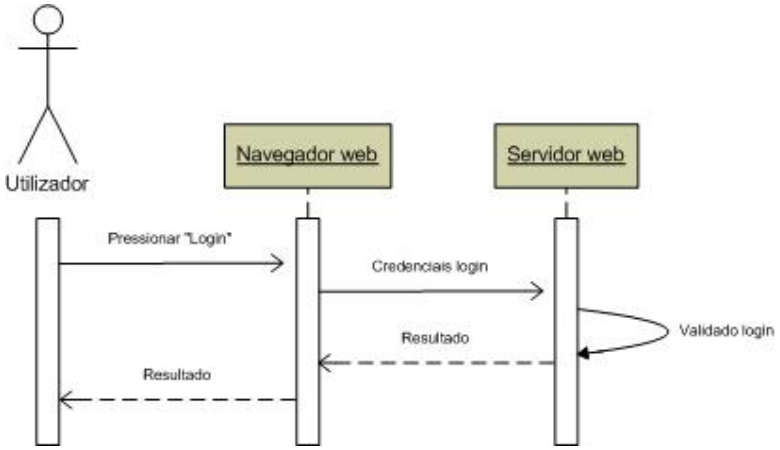


Figura 3.11: Diagrama de sequência da ação de *login* na aplicação Android

Scan de uma *tag* NFC

As *tags* são lidas pelo próprio dispositivo e não é feita nenhuma comunicação com os serviços web, pois toda a informação necessária pode ser inserida na própria *tag*. O *smartphone* está munido de um serviço que lhe permite detetar a presença de uma *tag* e a partir daí interpretar a informação nela contida, Figura 3.12.

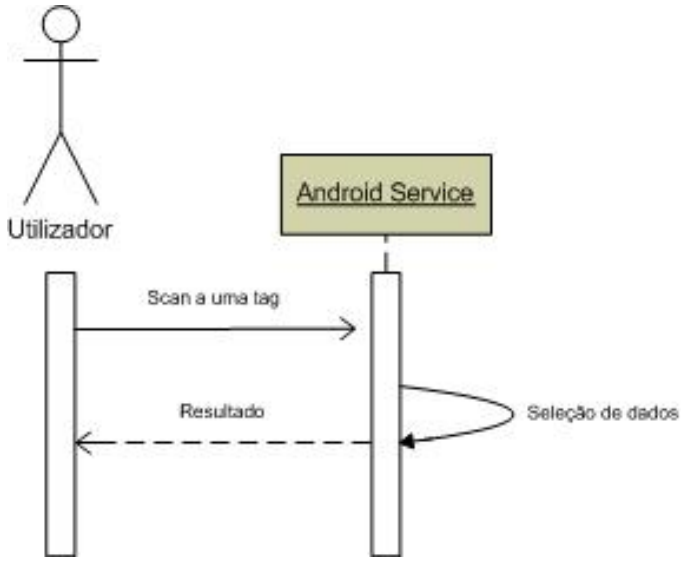


Figura 3.12: Diagrama de sequência da ação de *scan* de uma *tag* NFC

3.5.3 Serviço web

Envio de alerta de medicamento

O envio do alerta de toma de medicação é uma ação que ao contrário de todas as outras parte do lado do servidor. Quando algum cliente precisa de ser notificado para a toma de um medicamento, o servidor envia para o serviço de notificações da Google a identificação do dispositivo do cliente em questão bem como a mensagem a enviar. A notificação é depois encaminhada para o dispositivo onde o utilizador a pode ler, Figura 3.13.

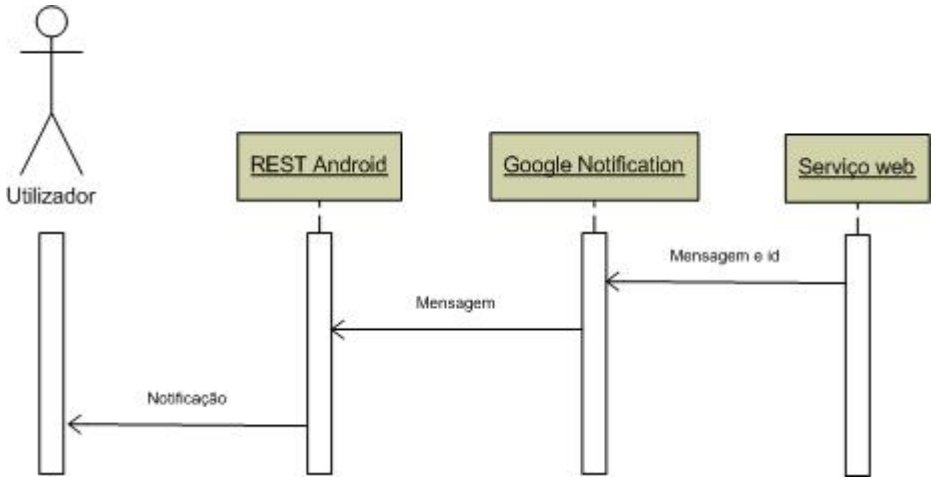


Figura 3.13: Diagrama de sequência da ação de envio de um alerta de medicamento

Capítulo 4

Camada web

4.1 Base de dados

A base de dados permite organizar e armazenar grandes quantidades de dados. Uma base de dados corretamente construída, permite uma pesquisa mais rápida evidenciada sobretudo por valores não repetidos dos resultados [49].

No ecossistema a base de dados permite manter um registo completo de informação detalhada centrada em clientes. Com a base de dados torna-se possível uma maior reutilização de código porque a base de dados é acedida quer pelos serviços web, através do Hibernate, quer pelo próprio website, não sendo assim preciso existir outros processos de armazenamento de informação.

Tabelas

Clientes

A tabela de clientes é a central de toda a base de dados. É em torno desta que toda a informação é gerada e todas as outras tabelas incluem informação complementar a esta tabela.

Esta tabela contém a informação pessoal do cliente: nome, morada, número de telefone,

idade e password de acesso. Cada cliente tem associado um só gestor. A sua chave primária é o "id_cliente", um inteiro que é gerado automaticamente aquando a criação de uma nova entrada desta tabela, Figura 4.1.

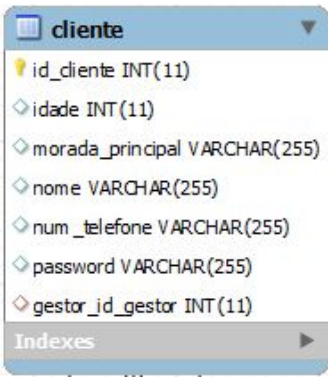


Figura 4.1: Tabela de clientes

Medicamentos

A tabela de medicamentos contém o nome e uma breve descrição daquilo que é o medicamento em questão. Existe também uma tabela de ligação denominada "cli_med" que liga cada cliente a vários medicamentos, contendo parâmetros como a hora de toma e data do fim da toma. Tem duas chaves estrangeiras que a ligam às tabelas de cliente e medicamentos, Figura 4.2.

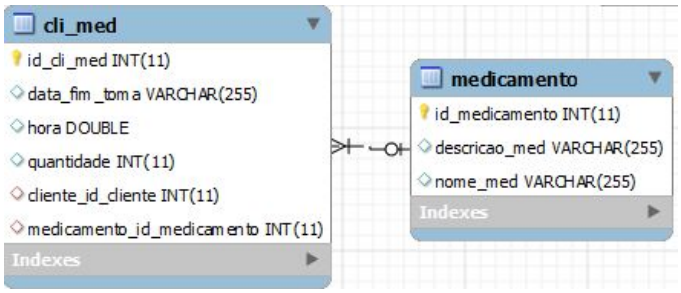


Figura 4.2: Tabela dos medicamentos

Gestores

Nesta base de dados as informações dos gestores de conta não são muito relevantes, o seu papel é apenas o de monitorizar e editar as definições dos seus clientes, pelo que na tabela apenas existe o seu nome, a password e a chave primária "id_gestor", Figura 4.3.

Cada cliente pode apenas ter um gestor associado, mas um gestor pode ter vários clientes a si associados.

Figura 4.3: Tabela dos gestores

Familiares

Estas tabelas são necessárias para poder existir um registo de contactos associados ao utilizador. Os familiares vão ter permissões para fazer login no site e assim acompanhar a situação do utilizador daí existir um campo para a password, Figura 4.4. Cada cliente pode ter vários familiares associados.

A tabela de grau de parentesco é utilizada para evitar a repetição de dados na criação de um novo familiar.

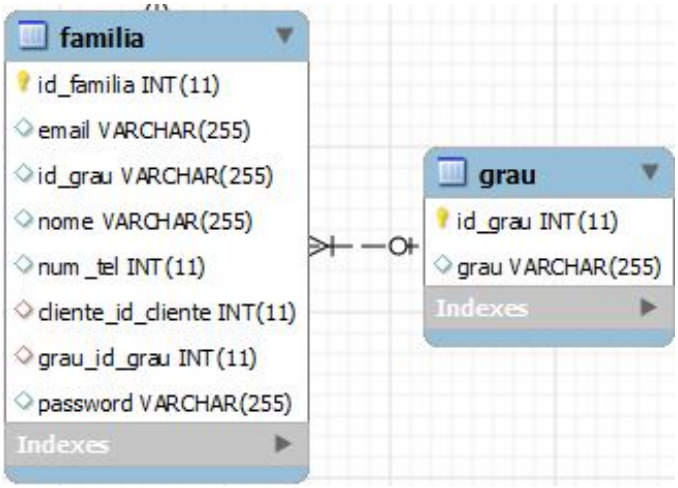


Figura 4.4: Tabela dos familiares dos clientes

Alertas

Os alertas são mensagens que contém localização, data e hora de onde foram geradas. Pode-se manter um registo e verificar alguma situação de perigo do lado do utilizador monitorizando-se esta tabela. Da mesma forma que os familiares estão ligados a uma tabela de grau de parentesco para evitar dados duplicados, a tabela de alertas está ligada a uma tabela de tipos de alerta que contém todos os alertas que são possíveis de acontecer, Figura 4.5.

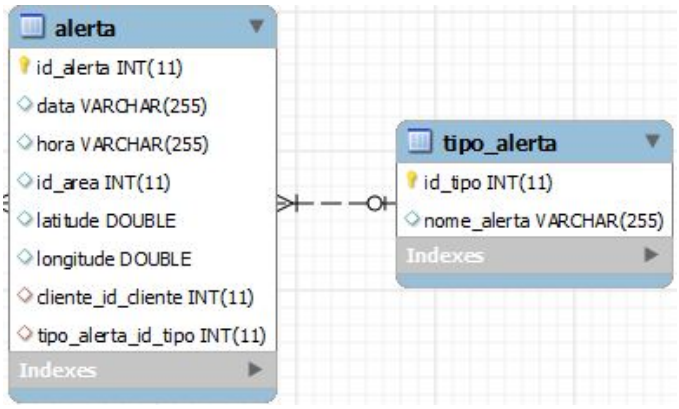


Figura 4.5: Tabela dos alertas

Áreas do cliente

As áreas do cliente são áreas definidas como seguras, nas quais o utilizador do sistema pode andar. Estas áreas em conjunto com os alertas permitem gerar avisos de alerta mais detalhados, contendo informações sobre em que área se gerou o alerta.

Por definição geral sabe-se que um conjunto de pontos forma uma área, e a cada área está definida uma morada. Cada cliente pode ainda ter várias áreas seguras, Figura 4.6.

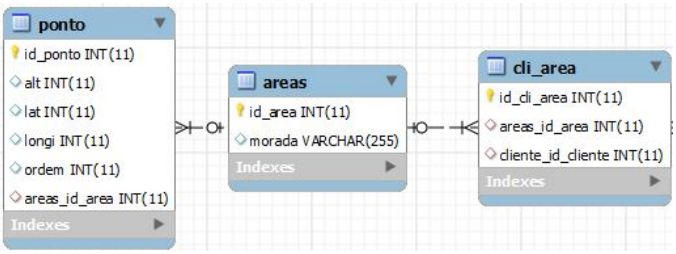


Figura 4.6: Tabela das áreas dos clientes

Drivers

A tabela de drivers contém a ligação, "url", para o driver descrito possibilitando a sua transferência para o equipamento que o requisitou, Figura 4.7.

Esta tabela não contém ligação com nenhuma outra tabela da base de dados. Apesar de estarem envolvidos vários *smartphones*, o SO Android fornece um nível de abstração suficiente para que os *device drivers* de dispositivos externos consigam ser utilizados em qualquer *smartphone* daí a não existência de qualquer relação com outra tabela qualquer desta base de dados.

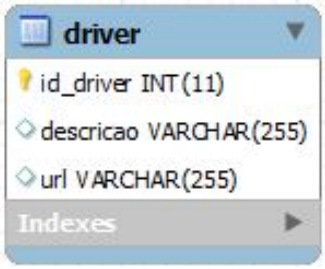


Figura 4.7: Tabela dos drivers

Dispositivos

Os dispositivos permitem ampliar a informação de cada utilizador, caraterizando o equipamento que estão a usar. A cada equipamento de cliente está associado um "token" que é usado para a receção de GCM. Este parâmetro é único por dispositivo.

A fim de evitar dados duplicados existe uma tabela auxiliar que contem as marcas e os modelos dos telemóveis, Figura 4.8.

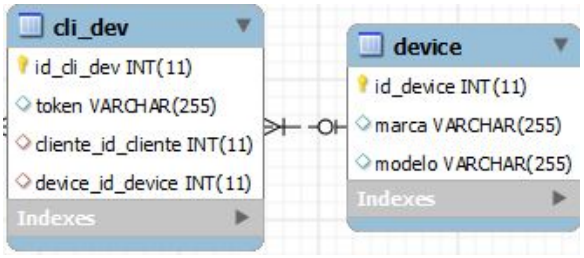


Figura 4.8: Tabela dos dispositivos

4.1.1 Esquema da base de dados

Após se analisarem as tabelas constituintes da base de dados, para se ficar a perceber inteiramente o seu funcionamento, a Figura 4.9 ilustra as relações entre todas as tabelas.

Tal como já foi referido, a tabela "cliente" tem um papel central. A base de dados foi construída seguindo os passos para a normalização de uma base dados [50] e seguindo as 12 regras de Codd [51].

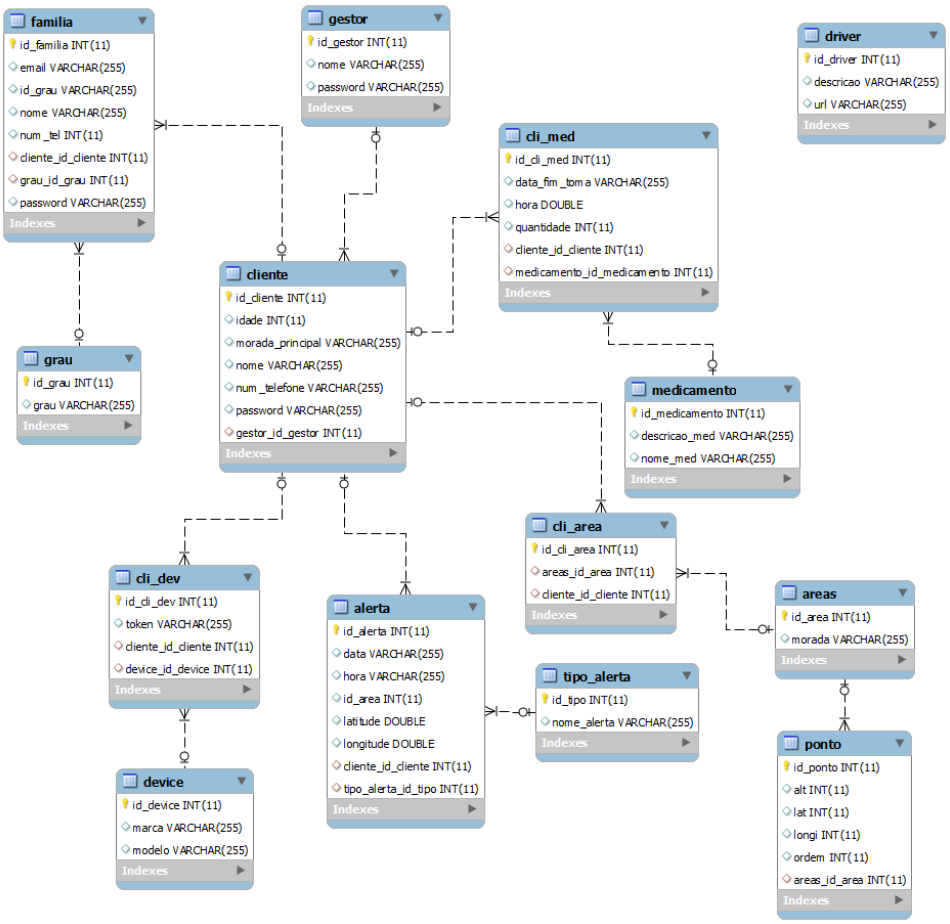


Figura 4.9: Base de dados relacional completa

Algumas das tabelas apresentadas, ”cli_areas”, ”areas”e ”ponto”, não foram utilizadas nesta dissertação, mas permaneceram na base de dados dada a sua utilidade numa futura evolução do sistema. As tabelas que se utilizaram são mais que suficientes para a prova de conceito.

A camada web é o resultado da composição dos serviços web com a base de dados e o website. A camada web encontra-se alojada num servidor online para que assim possa ser acedida por todos os dispositivos, da camada Android, em qualquer lugar que seja possível estabelecer uma ligação à Internet.

4.2 Serviços web

A interação com componentes externos é feita através dos vários serviços disponibilizados, permitem obter ou inserir nova informação na base de dados.

Como já foi referido os serviços foram criados usando uma arquitetura do tipo REST. A Tabela 4.1 serve de resumo do interface dos serviços implementados neste sistema.

Tabela 4.1: Interface dos serviços web

URI	**Método HTTP**	**Parâmetros de entrada**	**Retorno**
/service/dr/all	GET	-	Lista todos os médicos
/service/getFam//{id}	GET	-	Familiar do cliente
/service/newcliente	POST	Dados do cliente	Código HTML
/service/login	POST	Credenciais de login	Código HTML
/service/verify	POST	Credenciais de login e *token* GCM	Código HTML
/service/alarm	POST	Credenciais de login e dados do alerta	Código HTML

Os parâmetros de entrada são sempre objetos JSON contendo a informação especificada pelo parâmetro de entrada de cada um dos serviços.

Aquando a criação de um novo cliente, através da aplicação de registo, são geradas as credenciais de login. Estes dados guardados no ficheiro de texto presente no *smartphone*, como está apresentado no excerto de código abaixo.

```
{
        "password":"aqui_fica_a_password",
        "num_tel":"aqui_fica_o_número_telefone"
}
```

Os outros dados a enviar como parâmetros de entrada para os serviços web, para além das credenciais, são anexados ao mesmo ficheiro, no caso dos alertas o JSON a enviar tem o aspeto apresentado no excerto de código abaixo.

```
{
        "password":"aqui_fica_a_password",
        "num_tel":"aqui_fica_o_número_telefone"
        "dia":"dia_atual"
        "mes":"mes_atual"
        "ano":"ano_atual"
        "horas":"hora_alerta"
        "latitude":"ponto_latitude"
        "longitude":"ponto_longitude"
        "tipoAlerta":"id_tipo_alerta"
}
```

Para o caso da passagem de parâmetros dos dados de um novo cliente, o objeto JSON tem o seguinte aspeto:

```
{
        "password":"aqui_fica_a_password",
        "num_tel":"aqui_fica_o_número_telefone"
        "nome":"nome_cliente"
        "idade":"idade_cliente"
        "gestor":"id_gestor"
        "morada":"morada_cliente"
        "device":"marca_smartphone"
        "model":"modelo_smartphone"
}
```

/service/dr/all

O serviço contido no URI `/service/dr/all`, usa o método HTTP GET, não tem nenhum parâmetro de entrada e retorna um objeto JSON contendo informações sobre todos os médicos registados no sistema, o resultado de retorno é também acompanhado de um código Hypertext Markup Language (HTML) identificador do sucesso, código HTML 202, ou insucesso, código HTML 404, da operação.

A Figura 4.10 ilustra o fluxograma do serviço de listagem de todos os gestores de conta.

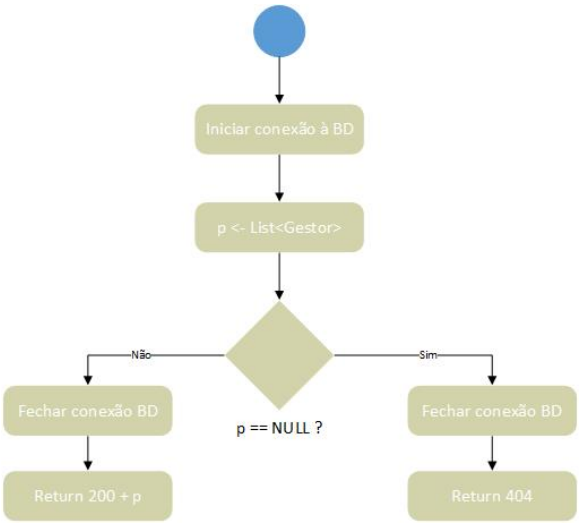


Figura 4.10: Fluxograma do serviço para obter todos os gestores

O serviço é chamado e é aberta uma sessão na base de dados para permitir que sejam feitas *queries*. O segundo ponto é passar para uma variável 'p', do tipo *List<Gestor>* o resultado da *query* de pesquisa de todos os gestores existentes. A variável 'p' nunca pode ser nula porque existem sempre gestores associados a este sistema daí que se a *query* retornar um resultado nulo deve ser enviado como resultado do serviço o código HTML 404.

/service/getFam//{id}

Através do URI `/service/getFam//{id}` é possível aceder-se ao serviço que usa o método HTTP GET para obter o número de contacto do familiar do utilizador passado como parâmetro do endereço, {id}.

Este serviço é utilizado após uma situação de queda, permitindo que o utilizador possa efetuar uma chamada para o seu familiar de modo a comunicar a sua situação. A Figura 4.11 ilustra o fluxograma do serviço de obtenção do número de telefone do familiar do cliente definido pela variável{id}.

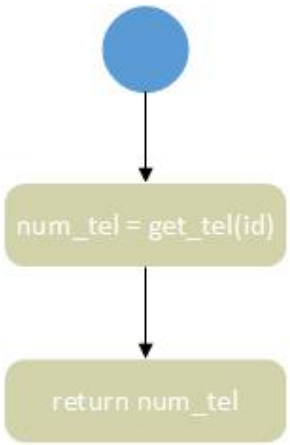


Figura 4.11: Fluxograma do serviço para obter o número de telefone do familiar

O serviço retorna um objeto JSON contendo o número de telefone do familiar a ser usado para a marcação rápida de número.

/service/newcliente

O URI `/service/newcliente` é utilizado para a criação de um novo cliente, como são enviados dados para o serviço, o método HTTP usado é o POST. O que é envido para este serviço são os dados do cliente inseridos na aplicação Android aquando o registo de um novo cliente. Caso o cliente já exista ou exista algum erro durante a criação de um novo cliente na base de dados é retornado o código HTML 404, no caso de o cliente ser único e não ocorrer nenhum erro é retornado o código HTML 200.

A Figura 4.12 diz respeito ao fluxograma do serviço de criação de um novo cliente.

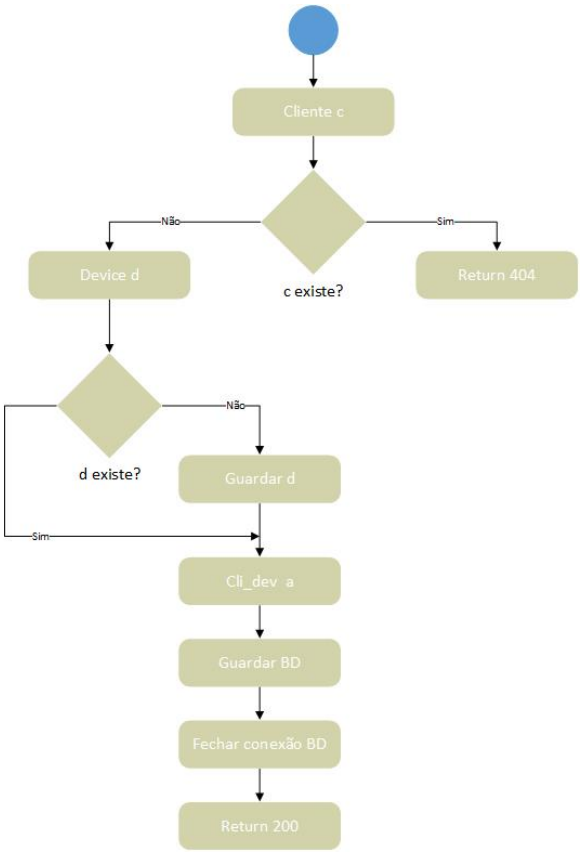


Figura 4.12: Fluxograma do serviço para a criação de um novo cliente

Para a criação de um novo cliente, um dos primeiros passos é verificar se já existe algum cliente igual, pelo número de telemóvel, no caso de existir é retornado o código de insucesso. Caso não exista nenhum cliente com o mesmo número de telefone, é verificado se já existe algum dispositivo igual na base de dados, caso não exista é adicionado um equipamento novo, este passo permite manter uma base de dados auto-atualizável no que diz respeito a dispositivos.

Seguidamente é associado o cliente e o dispositivo através da criação de uma nova entrada na tabela 'Cli_dev'.

As alterações feitas na base de dados são gravadas e a conexão pode ser fechada. No caso de todos os passos não gerarem erros, é retornado no fim o código HTML 200.

/service/login

Através do URI /service/login são verificadas as credenciais de login do cliente enviadas através do método HTTP POST. A confirmação positiva é retornada pelo código HTML 200 ao passo que qualquer discrepância nas credenciais é retornado um código HTML 404. Tal como está demonstrado pelo fluxograma da Figura 4.13.

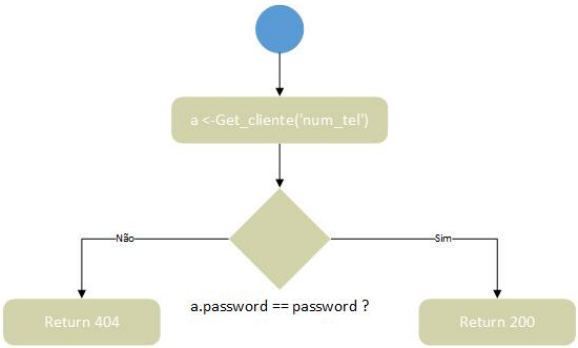


Figura 4.13: Fluxograma do serviço para efetuar o login no sistema

/service/verify

O URI /service/verify é usado para atualizar os dados referentes às GCMs. É utilizado o método HTML POST para enviar para o serviço as credenciais do cliente e o token identificador do dispositivo, adquirido através dos serviços de GCM da Google. Tal como nos outros serviços são retornados os códigos HTML 200 e 404 no caso de a informação estar correta e ser bem inserida e para o caso da existência de algum erro, respetivamente, tal como se pode verificar pelo fluxograma da Figura 4.14.

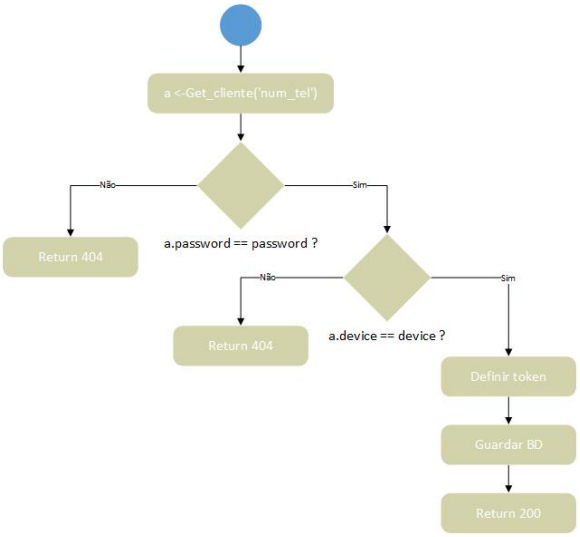


Figura 4.14: Fluxograma do serviço para efetuar a atualização do token GCM

/service/alarm

Com o URI `/service/alarm` é possível inserir no sistema um novo alerta. É utilizado o método HTML POST para enviar para o serviço as credenciais do cliente e o tipo de alerta que deve ser registado. Este serviço é utilizado no caso de uma queda, por exemplo.

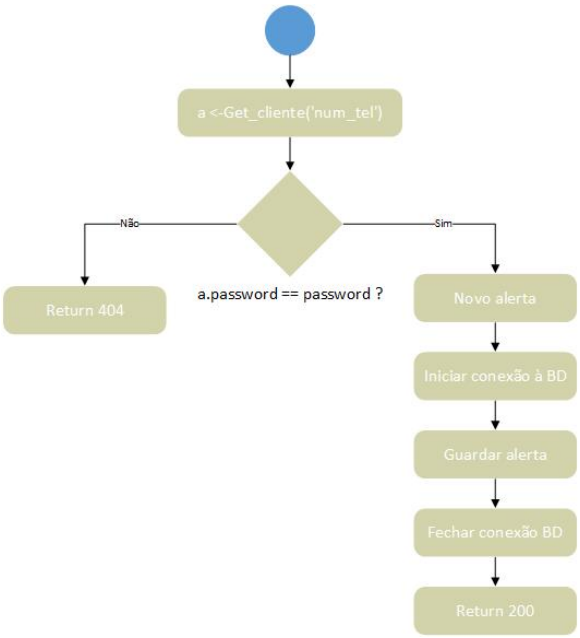


Figura 4.15: Fluxograma do serviço para a criação de um alerta

Tal como em todos os outros casos, as credenciais do cliente são verificadas e é criado um novo alerta com os dados que foram recebidos como parâmetro de entrada. O processo repete-se para aceder, guardar e fechar a conexão com a base de dados e o retorno é dado como código de sucesso ou insucesso.

4.3 *Website*

O *website* foi desenhado para permitir que utilizadores registados no sistema possam interagir com a base de dados, de modo a visualizar ou editar campos conforme os níveis de privilégio.

Existem três níveis de privilégio:

- o nível de **Guest**, que é atribuído a pessoas que não estejam autenticadas no *website*, o seu acesso é o mais limitado, estando apenas acessíveis algumas das páginas existentes, sendo que nenhuma delas contém informação da base de dados;
- o nível de **Admin**, que é atribuído a utilizadores do sistema autenticados que pertençam a tabela de gestores;
- o nível de **User**, que é atribuído a utilizadores do sistema autenticados que pertençam a tabela de familiares.

Aos utilizadores autenticados é-lhes permitido realizar ações Create Read Update Delete (CRUD) sobre a base de dados, existindo mesmo assim regras a serem seguidas:

- Qualquer utilizador autenticado pode alterar apenas informação que lhe pertença, isto é um utilizador autenticado não pode alterar informações de outro utilizador com credenciais de *login*;
- Qualquer utilizador autenticado apenas pode visualizar informação que lhe pertença ou que esteja diretamente ligado a si, isto é, não pode ser visualizada informação que pertença a outros utilizadores com credenciais de *login*;

- A ação de apagar só está disponível a utilizadores autenticados com o privilégio de *Admin*, esta ação só pode ser realizada para informação que pertença diretamente ao *Admin*, ou seja, nenhum utilizador administrador pode eliminar informação que pertença a outro utilizador administrador;

No caso desta dissertação, o *website* foi desenvolvido em PHP: Hypertext Preprocessor (PHP), com recurso à *framework* Yii [52] e em HTML.

O PHP encarregou-se da criação das ligações com o servidor e de toda a dinâmica envolvida no *website*, já o HTML foi utilizado para criar a modelação gráfica, utilizou-se a linguagem de estilo Cascading Style Sheets (CSS) de modo a estruturar a informação sobre a formatação das páginas web num único ficheiro, facilitando a reutilização e modificação de código.

Index

A página de index, Figura 4.16 é o rosto do *website*, é exibida a utilizadores não autenticados.

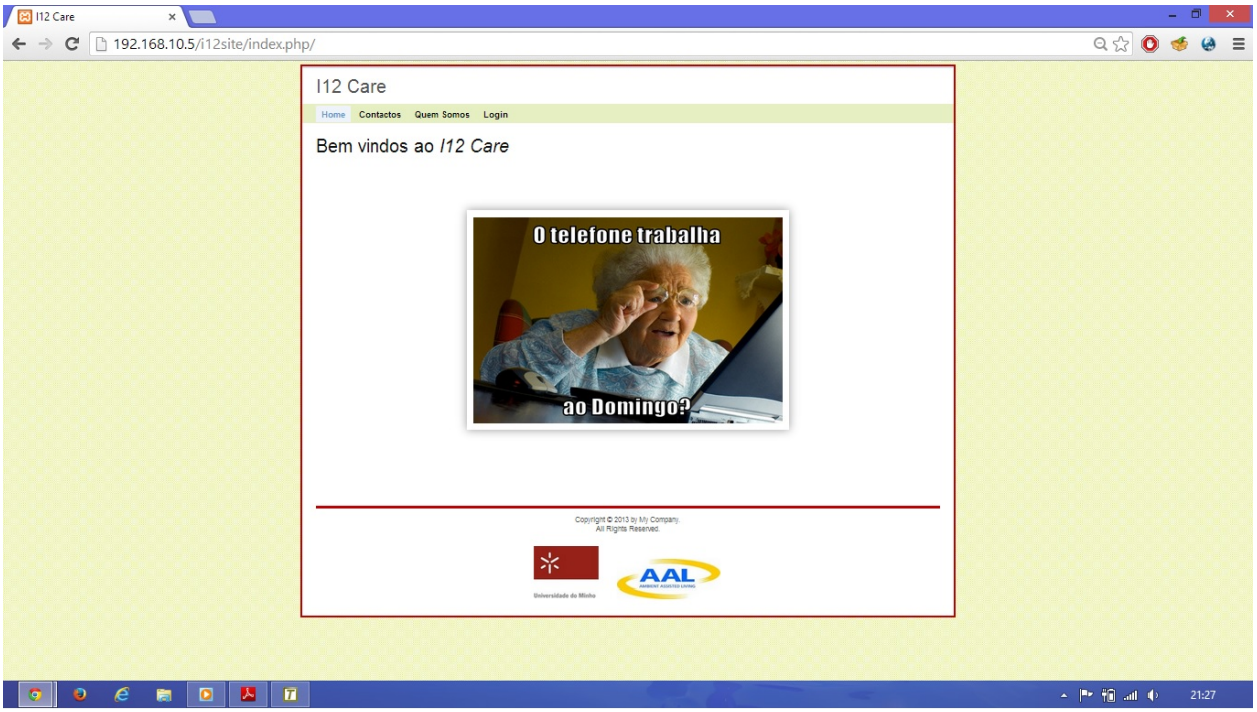


Figura 4.16: Página inicial do *website*

A sua função é divulgar informações sobre a empresa, e sobre as características principais do serviço.

A partir desta página o utilizador não autenticado pode navegar até à pagina de contactos que contém um formulário para que seja possível enviar duvidas ou sugestões diretamente para a empresa; à página ”Quem somos”que explica com um maior detalhe os serviços prestados pela empresa, a sua credibilidade e as vantagens de utilização do serviço; também é possível ir para a página de ”Login”.

Login

A página de *login*, Figura 4.18, contém o design para que seja possível validar as credencias dos utilizadores de modo a que se tornem em utilizadores autenticados.

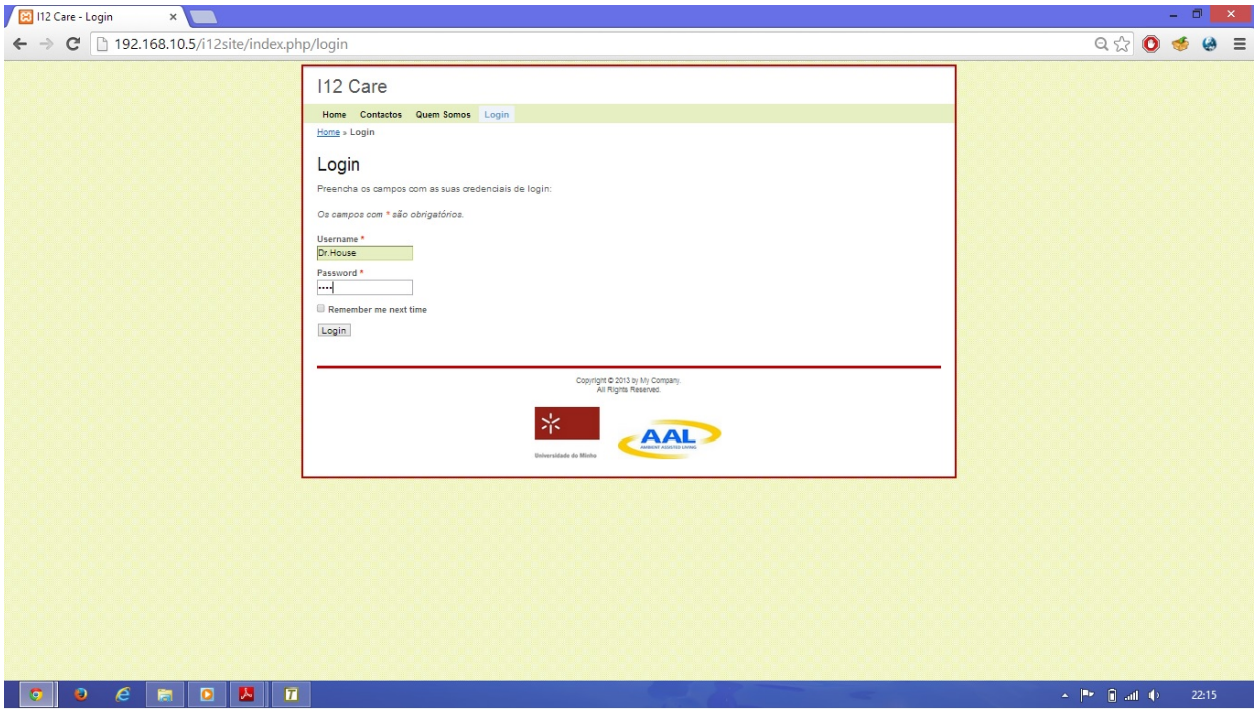


Figura 4.17: Página de *login* do *website*

Os dados inseridos são comparados com os dados existentes na base de dados, em caso de erro é exibida uma mensagem indicando um disparidade de informação de *login*. Para o caso de os dados estarem em sintonia com aqueles que existem na base de dados, a página é encaminhada para um index de utilizadores autenticados. Este index contém informações diferentes consoante o utilizador autenticado é administrador ou não.

Index admin

Depois de autenticado o administrador o site é redirecionado para a página de gestão de clientes onde é apresentada uma lista com todos os clientes associados aquele gestor. São também colocadas as opções de ver com mais detalhe o cliente, editar a sua informação ou apagar o cliente.

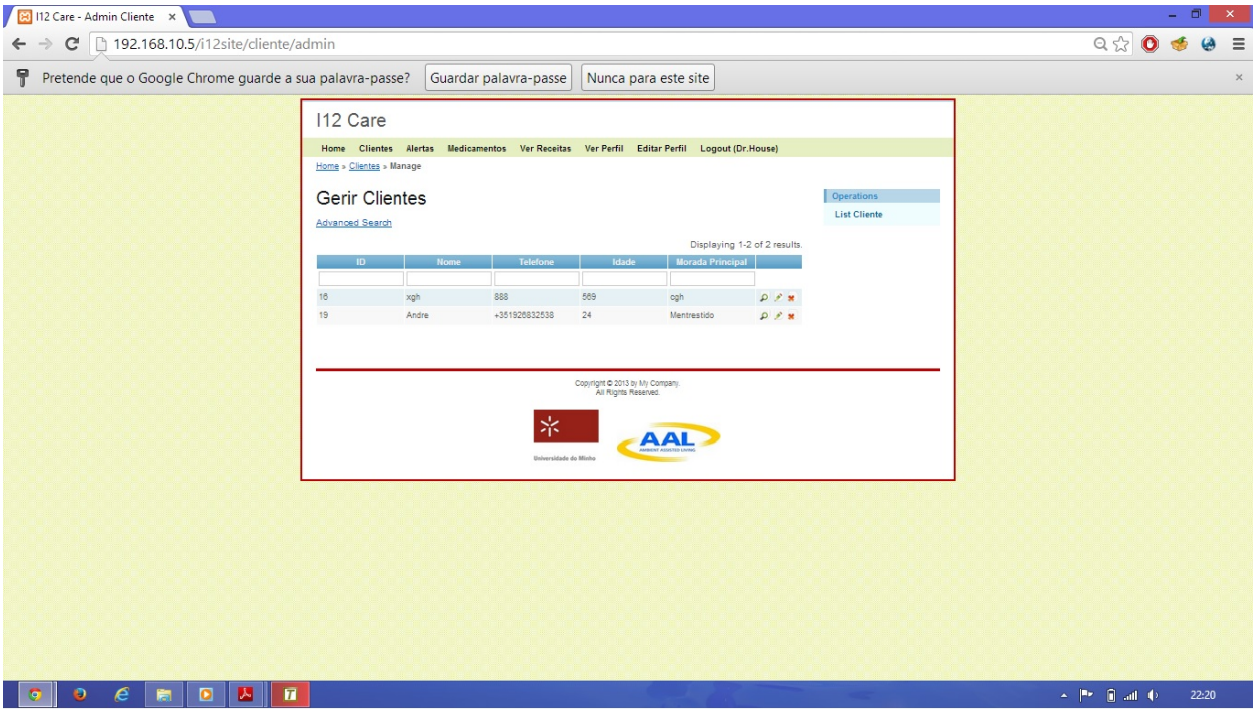


Figura 4.18: Página de início do administrador do *website*

A barra principal também mudou o seu aspeto sendo adicionadas ligações para ver medicamentos, alertas, ver receitas médicas, ver o perfil e editar o perfil.

Ver perfil de cliente como administrador

Ao ver o perfil do cliente da lista, o utilizador é encaminhado para a página da Figura 4.19

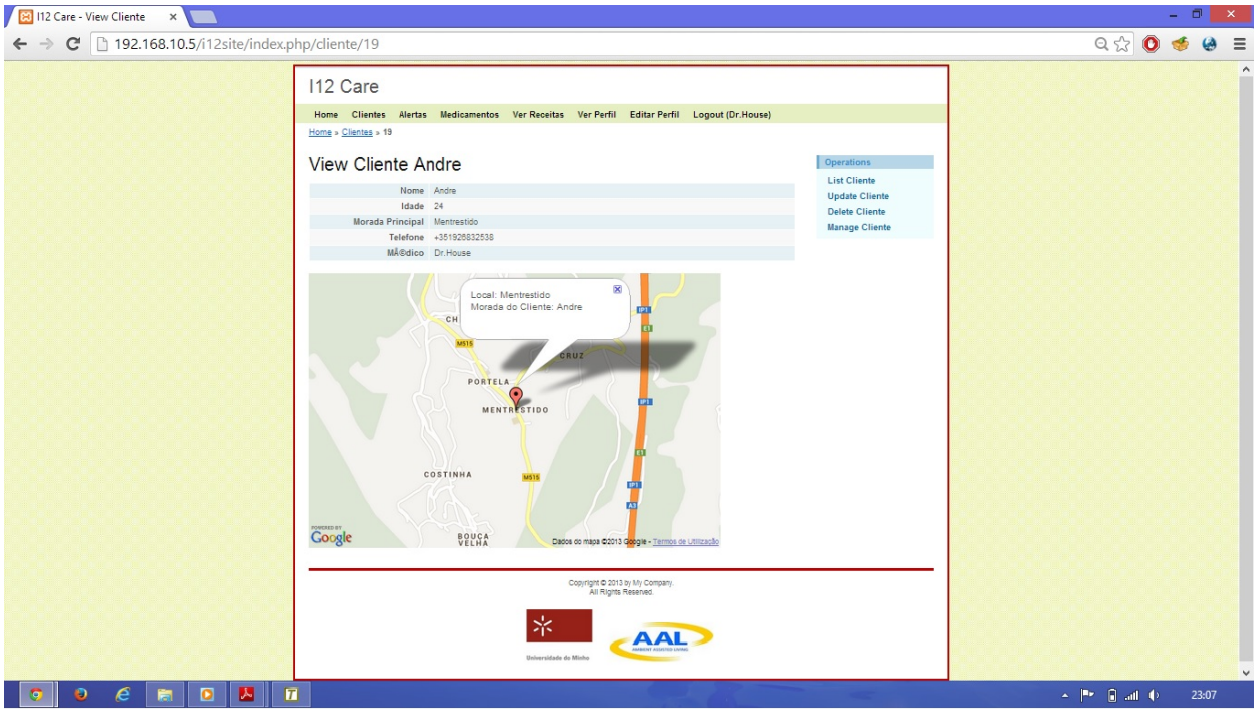


Figura 4.19: Página de perfil de cliente vista pelo administrador do *website*

Através do menu da direita também é possível editar, apagar ou ver a lista toda dos clientes. Este menu lateral acompanha a maior parte das páginas e permite um acesso mais fácil às ações CRUD do objeto em questão.

Gerir receitas

A gestão das prescrições médicas permite manter atualizada a tabela "Cli_med", usada para o envio de GCM para os *smartphones* dos utilizadores.

Apenas os administradores podem editar esta tabela, mas os familiares também podem ter acesso a esta informação mas num modo de visualização apenas.

A Figura 4.20 apresenta a página web apresentada para a gestão de receitas por parte do administrador.

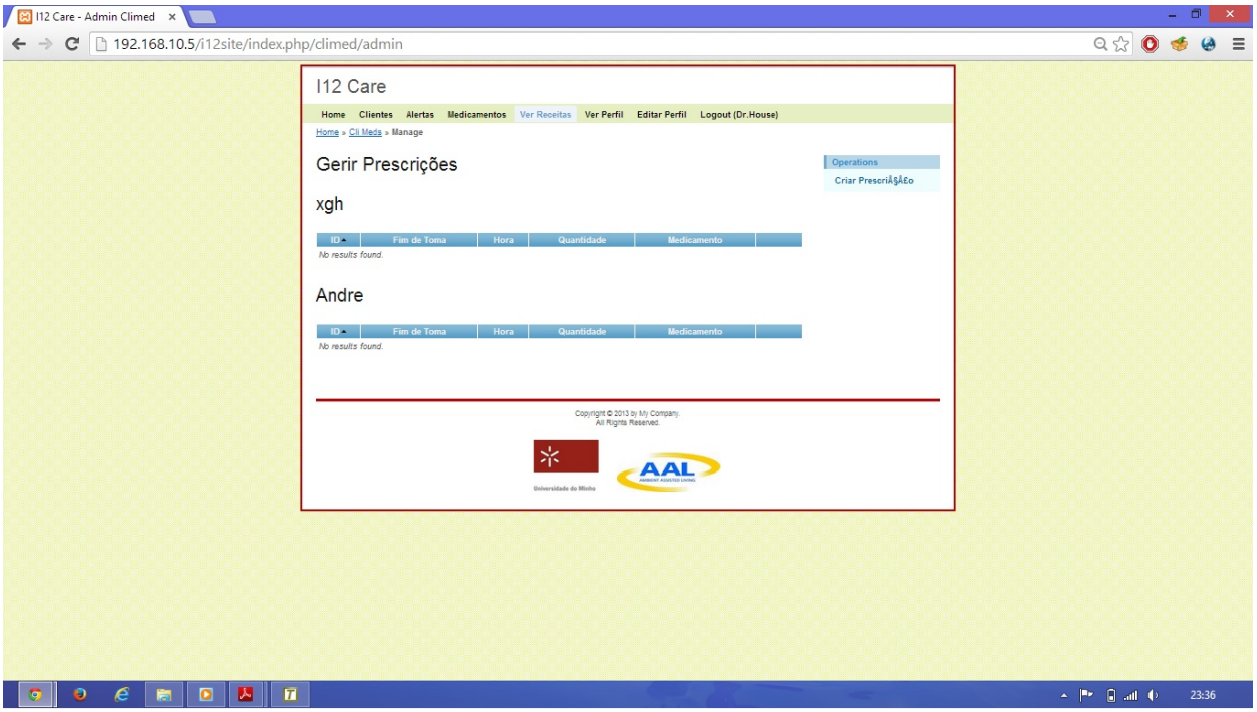


Figura 4.20: Página de gestão de receitas vista pelo administrador do *website*

Como cada gestor pode ter vários clientes associados, as prescrições foram separadas por utilizador, em tabelas diferentes, facilitando a pesquisa de informação. No caso do familiar a informação é apresentada referente ao seu cliente associado.

Os gestores podem criar prescrições novas para cada um dos seus clientes com os medicamentos existentes na base de dados.

Medicamentos

A gestão de medicamentos é permitida a administradores para que estes possam indicar novos medicamentos a serem tomados pelo cliente, permitindo que esta tabela seja atualizada e que contenha apenas dados necessários, isto é, que não contenha dados que nunca vão ser utilizados.

Na Figura 4.21 é apresentado o design desta página.

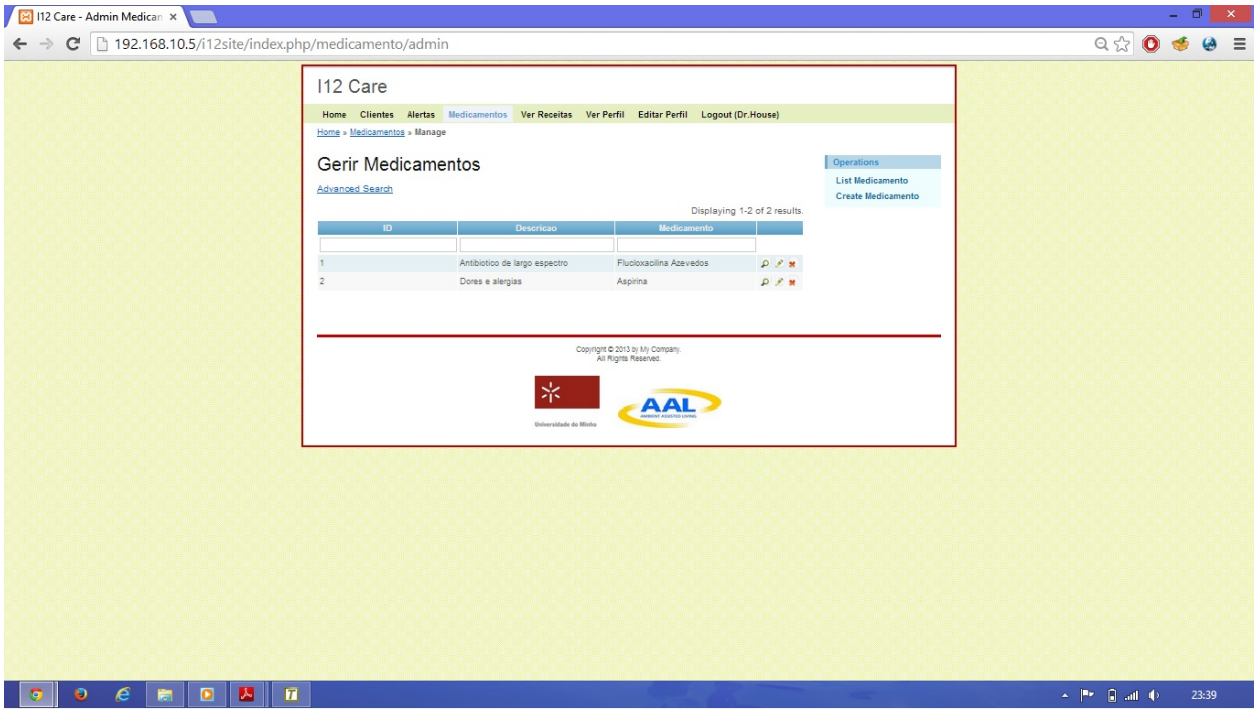


Figura 4.21: Página de medicamentos vista pelo administrador do website

Esta página é exclusiva para administradores pelo que os familiares não podem ver a lista de medicamentos existentes no sistema, por consequência também não podem nem editar nem eliminar entradas de medicamentos.

Alertas

A página de alertas tem como objetivo a recolha dos dados da base de dados da tabela "Alerta", para cada utilizador autenticado a página apresenta na tabela resultados diferentes de clientes conforme quem esteja a visualizar. Para o caso de o utilizador ser administrador, é apresentada a hipótese de ver e eliminar um alerta, no caso de ser um familiar, apenas é possível ver o alerta.

A Figura 4.22 ilustra o design da página vista por um utilizador administrador do sistema.

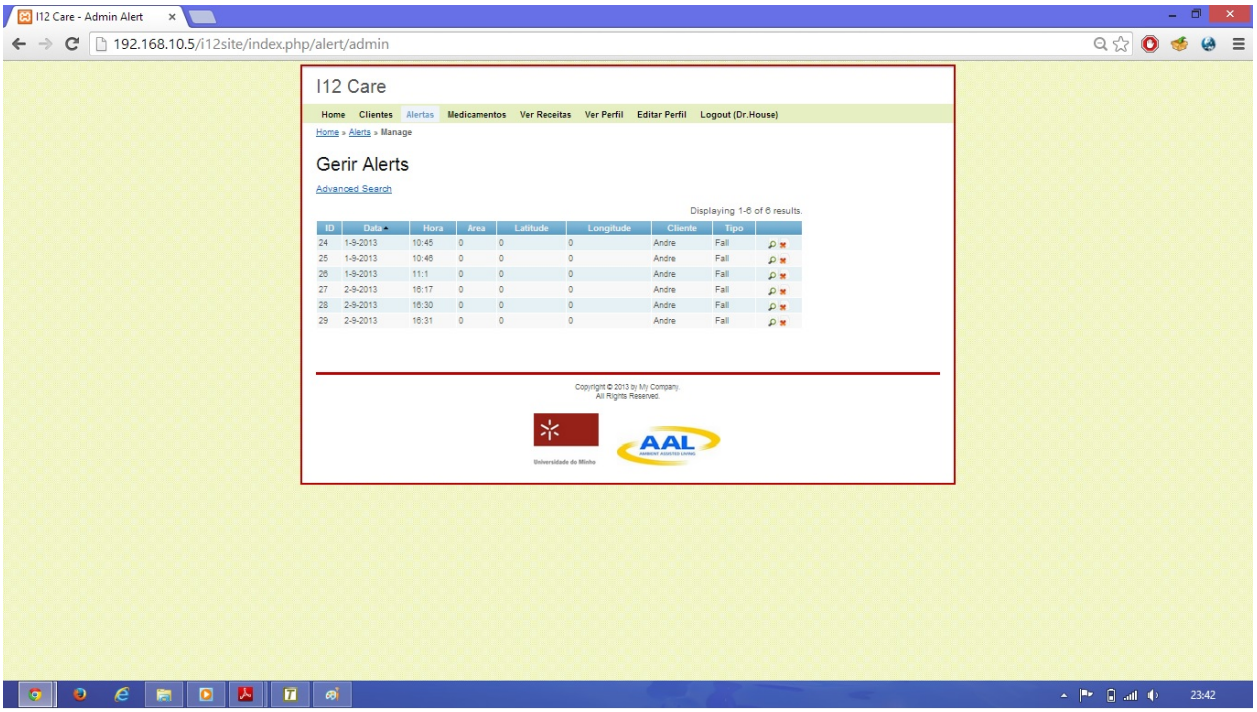


Figura 4.22: Página de alertas vista pelo administrador do website

A opção de ver um alerta, apresenta um menu com informações extra bem como uma mapa onde é indicada a localização de onde ocorreu o alerta, através da latitude e da longitude, ver Figura 4.23.

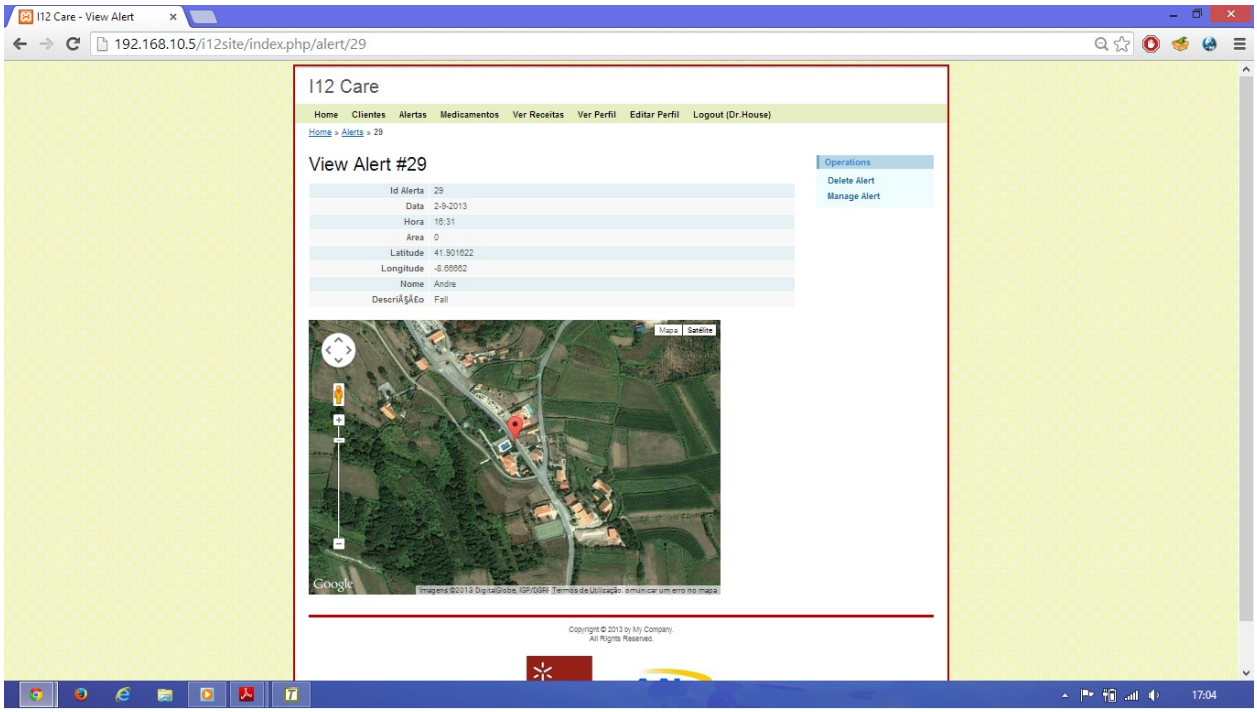


Figura 4.23: Página de exemplo de um alerta vista pelo administrador do website

O *design* do *website* para o utilizador não administrador não difere em nada para o design do administrador, apenas algumas opções não estão disponíveis e alguns menus não são acessíveis pois o conteúdo não representa interesse para o utilizador, pelo que não vão ser apresentadas figuras representativas do *website* quando autenticado como utilizador não administrador.

Capítulo 5

Camada Android

A camada Android é representada pelo ecossistema desenvolvido nesta dissertação: o barramento de comunicação e o conjunto de *plugins*. Estes dois componentes ficam a fazer parte do sistema operativo.

Esta camada vai comunicar com a camada web, permitindo a obtenção de dados e a criação de um espectro de funcionalidades mais alargadas. Como os dados não estão armazenados localmente o dispositivo não precisa de ocupar a sua memória interna.

5.1 Barramento de comunicação

O barramento é uma aplicação que está encarregue de fornecer um meio de comunicações entre os *plugins* instaláveis do sistema.

A gestão de *plugins* é assegurada neste nível da aplicação, onde o barramento ”conhece”todos os *plugins* do sistema, conseguindo assim encaminhar a informação, entre eles, de forma correta.

É nesta aplicação que se encontra também todo o ambiente gráfico da aplicação, onde são inseridos os dados de um novo registo e onde é permitido o *login* aos clientes.

A operação de *logout* está também contemplada e permite a paragem de execução de todos os *plugins* instalados.

Menu principal

O menu principal é o primeiro contacto que o utilizador tem com a aplicação. A sua função é encaminhar o utilizador para as outras funcionalidades da aplicação, ir para o menu de *login* ou para o menu de registo de um novo utilizador

Nesta altura da execução da aplicação nenhum serviço está ativo.

Além disso, é neste menu que é preenchida a lista de *plugins* instalados no dispositivo. Isto é possível pois todos os *plugins* possuem um campo comum, que permite ao barramento saber que são extensões suas.

```
1  <application android:label="@string/app_name" >
2     <service
3        android:name=".PluginServiceGCM"
4        android:exported="true" >
5           <intent-filter>
6              action android:name="aexp.intent.action.PICK_PLUGIN" />
7
8              <category android:name="aexp.intent.category.GCM_PLUGIN" />
9           </intent-filter>
10    </service>
```

Todos os *plugins* incorporam no seu ficheiro AndroidManifest.xml um filtro de ação "aexp.intent.action.PICK_PLUGIN", tal como está exemplificado no excerto de código acima.

A Figura 5.1 ilustra o ambiente gráfico deste menu.

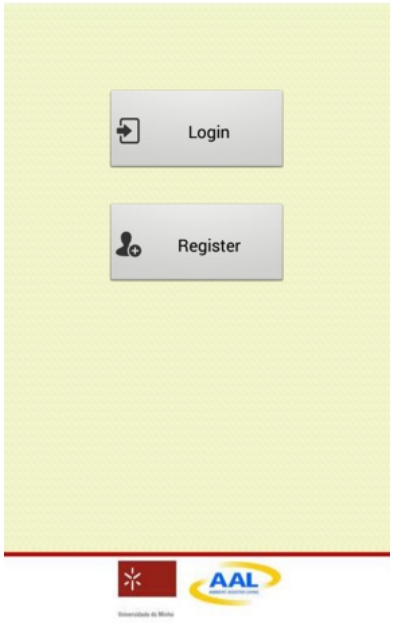


Figura 5.1: Aspeto da atividade do menu principal da aplicação

Novo cliente

A introdução de um novo cliente no sistema é feita através da aplicação Android. O fluxo de ações está ilustrado na Figura 5.2. O fluxograma corresponde à ação gerada quando se pressiona o botão de ”Efectuar Registo”, ver Figura 5.3, nesse momento os dados são verificados para que nenhum dado esteja em falta.

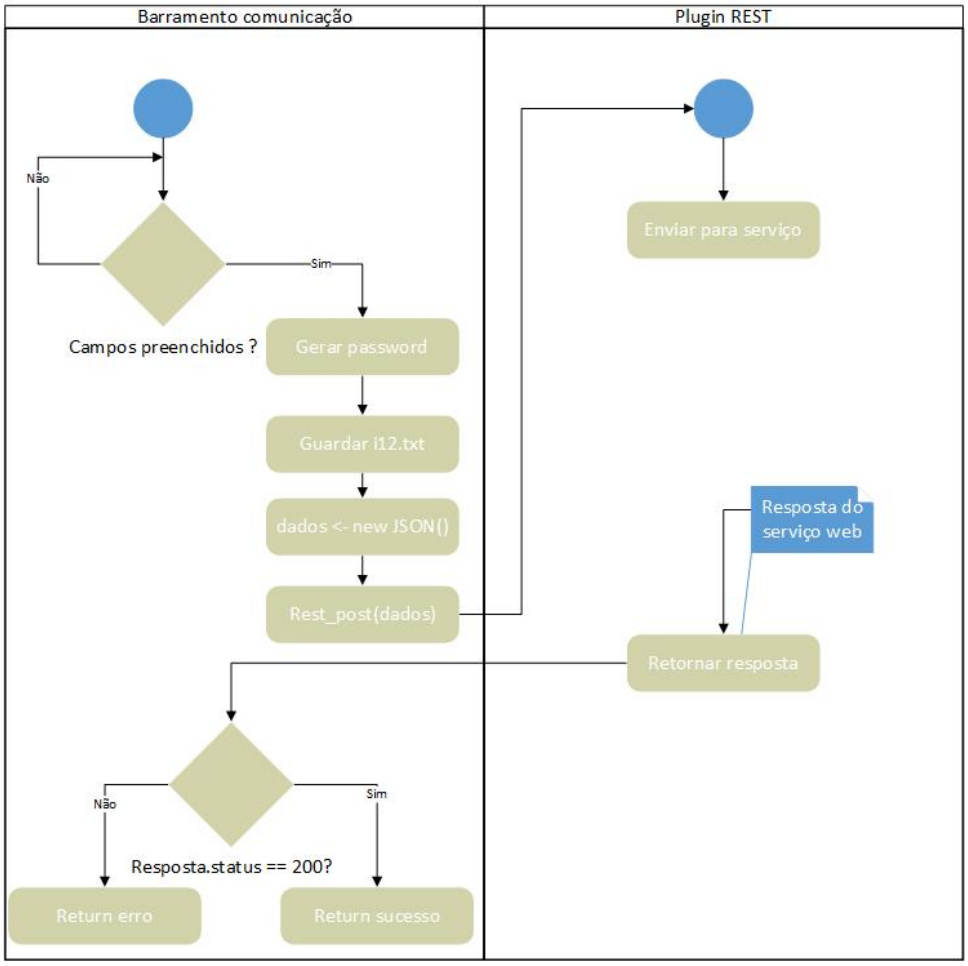


Figura 5.2: Fluxograma referente à criação de um novo cliente na camada Android

Após a verificação, o sistema gera uma *password* aleatória e guarda este campo e o número de telefone num ficheiro de texto com o nome "i12.txt"na raiz do cartão Secure Digital (SD). Este ficheiro é utilizado sempre que for necessário validar as credenciais de utilizador.

Todos os dados inseridos são depois enviados para o serviço web através do *plugin* REST, que se encontra numa *thread* à parte do barramento.

A resposta do serviço é enviada para o *plugin* REST, que de seguida a encaminha para o barramento de comunicação que procede à validação de dados recebidos e continua a execução do registo de um novo cliente.

A Figura 5.3 apresenta o aspeto final do ambiente gráfico da atividade de criação de um novo cliente.

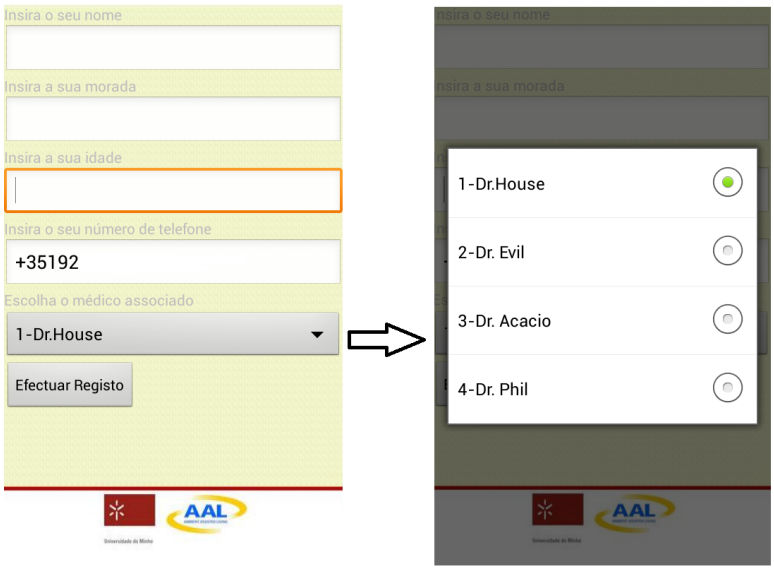


Figura 5.3: Aspeto da atividade da criação de um novo cliente

O campo do número de telefone é preenchido automaticamente pelo sistema, no entanto é possível ser editado manualmente para a prevenção de erros em alguns dispositivos. A lista de gestores é carregada antes do ambiente gráfico ser mostrado, esta lista incorpora todos os gestores disponíveis no sistema para que o cliente possa escolher um gestor da sua preferência.

Login

Quando o botão de *login* é pressionado, a informação contida no ficheiro "i12.txt"é copiada para um objeto JSON, que posteriormente é enviado para o serviço web a fim de serem verificados os dados.

Após a resposta do serviço, o programa decide se emite uma mensagem de erro ao utilizador, ou se confirma o *login* e salta para o menu de *logout*, como está ilustrado na Figura 5.4.

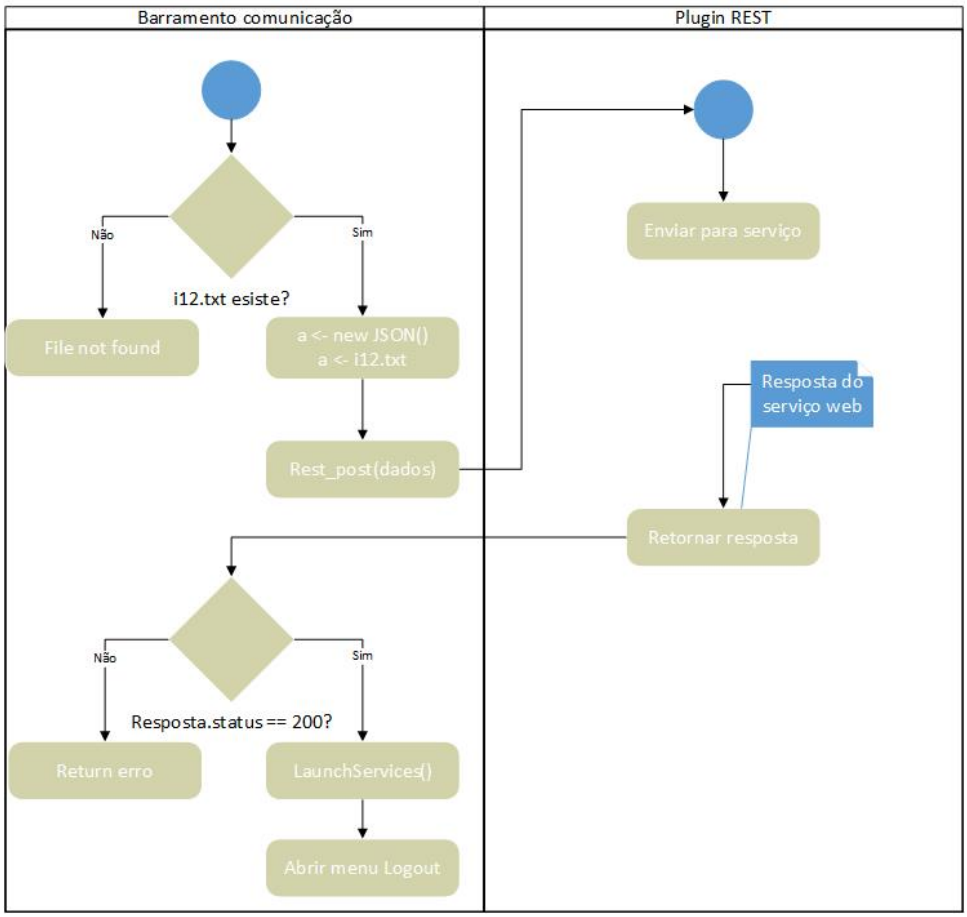


Figura 5.4: Fluxograma referente ao login de um cliente

Antes de haver a mudança de atividade, a função LaunchServices(), é invocada para que todos os *plugins* instalados possam iniciar a sua atividade.

Nem todos os *plugins* são iniciados com esta função, já que o *plugin* REST é utilizado quer no registo de um novo cliente quer no login. Outros *plugins* como é o caso do GPS, GCM e o detetor de quedas são inicializados nesta função consoante a sua instalação no ecossistema.

A inicialização dos serviços associados a cada *plugin* é explicada pelo fluxograma da Figura 5.5.

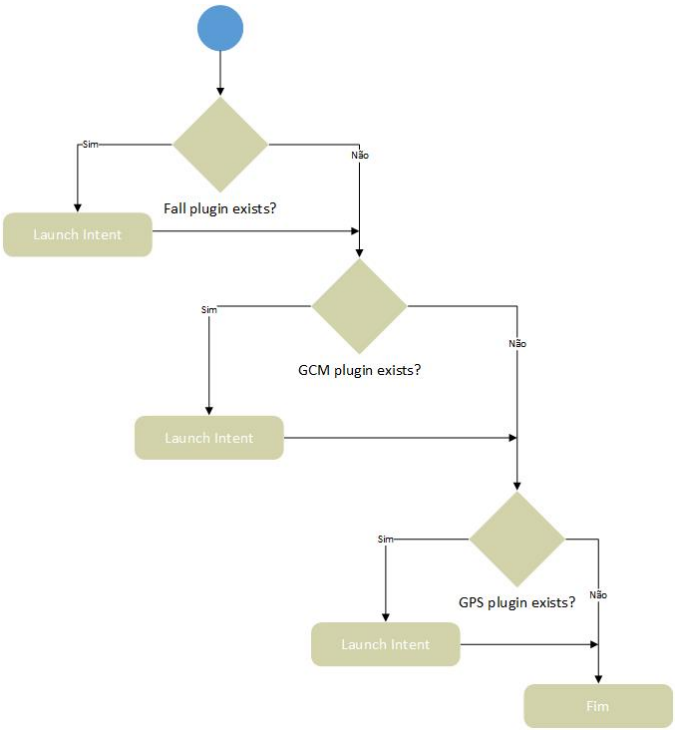


Figura 5.5: Fluxograma referente à função de iniciação dos serviços

A lista de *plugins* é preenchida quando a aplicação é iniciada, sempre que é adicionado ou removido um *plugin*, a lista é também atualizada, permitindo ao sistema saber sempre que *plugins* pode utilizar.

Logout

A atividade de *logout* é a que tem maior tempo de utilização entre todas as outras. O programa entra nesta atividade após ser feito o *login* e permanece nela até o SO encerrar ou até ser feito o *logout*.

Quando pressionado o botão "Logout", Figura 5.8, a aplicação termina a execução dos serviços ativos e volta para o menu principal da aplicação, Figura 5.6.

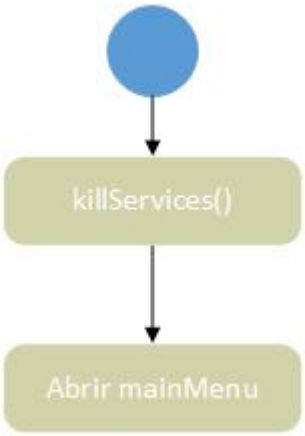


Figura 5.6: Fluxograma referente ao logout da aplicação

O método de paragem dos serviços é similar ao de iniciação. A lista de serviços instalados é percorrida fazendo-se a paragem de cada um dos serviços individualmente, Figura 5.7.

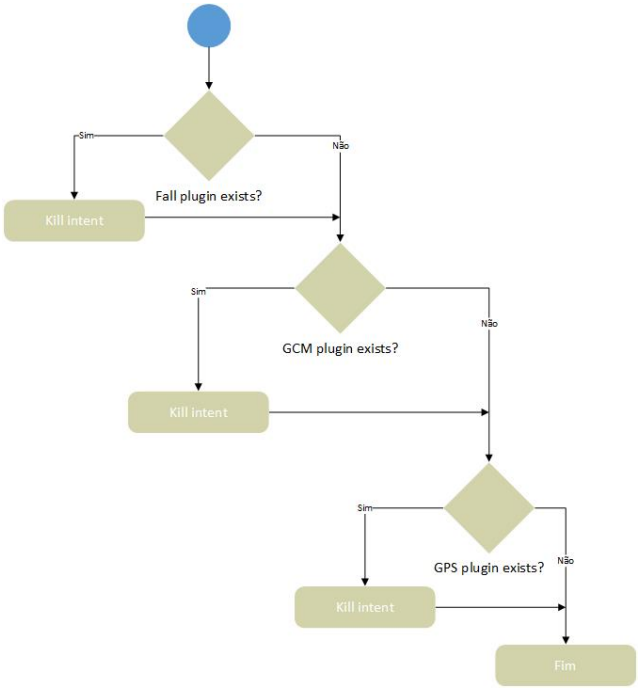


Figura 5.7: Fluxograma referente à função de paragem dos serviços

É importante que todos os *plugins* que possuam serviços a executar, sejam parados a fim de evitar a ocorrência de erros causados pela não existência de métodos para tratar as

respostas dos serviços, que estavam presentes no contexto da atividade de *logout*.

Este método encerra o ciclo de vida do ecossistema, que começa na criação de um cliente novo, passa pela validação e iniciação dos serviços no *login*, seguindo-se a fase em que esta a executar e culmina como já foi referido na ação de *logout*.

Após todas as aplicações serem terminadas e o programa retornar para a atividade principal, efetuando-se, assim, um novo ciclo com a operação de *login*.

O aspeto final do ambiente gráfico da atividade de *logout* é ilustrado pela Figura 5.8.

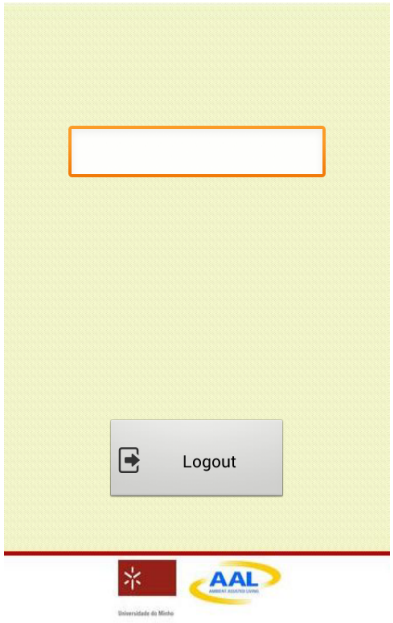


Figura 5.8: Aspeto da atividade de logout

5.2 *Plugins*

O desenvolvimento de *plugins* para esta aplicação, tem de seguir algumas regras de modo a existir um código uniforme entre todos os *plugins*. Esta uniformização permite que os acessos barramento->*plugin*->barramento sejam mais facilmente percebidos e implementados.

Para se perceber a forma de implementação, vai ser usado o *plugin* REST como exemplo.

Em primeiro lugar foram criados dois *packages* um de nome ”aexp.plugin”e outro

"aexp.plugin3". No primeiro *package* estão os ficheiros .aidl, já referenciados no Capítulo 2.3.2.

```
package aexp.plugin;

interface IBinaryREST {
  boolean do_post(in String addr, in String data);
  String do_get(in String url);
}
```

O nome da interface é "IBinaryREST", possui duas funções: uma para o método HTTP POST e outra para o método HTTP GET.

No que diz respeito ao segundo *package* é onde se encontra a implementação do serviço referente ao *plugin*.

```
public class PluginService3 extends Service {

        public IBinder onBind(Intent intent) {

                        return RestBinder;
                }

         private final IBinaryREST.Stub RestBinder = new IBinaryREST.Stub() {

                        public String do_get(String url){
                                // Código para a função HTTP GET
                        }

                        public boolean do_post(String addr, String data){
                                // Código para a função HTTP POST
                        }
}
```

A chamada ao *plugin* é feita através do barramento de comunicações. Para se iniciar

a execução é primeiro invocada uma atividade, neste caso é a atividade ”InvokeRest”. A atividade recebe parâmetros extra, tais como o método HTTP a utilizar, a categoria do *plugin*, os dados a enviar para o serviço e o endereço para enviar. O excerto de código abaixo completa a explicação de como invocar a atividade.

```
Intent intent = new Intent();
intent.putExtra(PluginApp.BUNDLE_EXTRAS_CATEGORY,
        "aexp.intent.category.REST_PLUGIN");
intent.putExtra(Constants.METHOD_KEY, Constants.POST_KEY);
intent.putExtra(Constants.URL_KEY, Constants.address_login);
intent.putExtra(Constants.PARAMS_KEY, a.toString());
intent.setClassName("aexp.pluginapp",
        "aexp.pluginapp.InvokeRest");
startActivityForResult(intent, 2);
```

O ”InvokeRest”é uma atividade que não contém *layout*. No seu método ”onCreate”os parâmetros passados como extra são atribuídos a varáveis globais da atividade e seguidamente é feito o *bind* da atividade com o *plugin* através do código que pode ser visualizado abaixo.

```
opServiceConnection = new OpServiceConnection();
Intent i = new Intent(PluginApp.ACTION_PICK_PLUGIN);
i.addCategory(category);
bindService(i, opServiceConnection, Context.BIND_AUTO_CREATE);
```

Após isto, quando a atividade ganha foco e entra no método ”onWindowFocusChanged”é feito o pedido de GET ou POST para o *plugin* consoante o método que foi passado como extra na altura da invocação desta atividade. O excerto de código apresentado abaixo mostra o método ”onWindowFocusChanged”da atividade ”InvokeRest”.

```
if (hasFocus) {
        try {
        if (method.equalsIgnoreCase("GET")) {
        resultGET = opService.do_get(url);
```

```
5           Intent i = new Intent();
6           i.putExtra("string", resultGET);
7           setResult(RESULT_OK, i);
8           } else {
9                   postResult = opService.do_post(url, parameters);
10                  Intent i = new Intent();
11                  i.putExtra("RESULT", postResult);
12                  setResult(RESULT_OK, i);
13                  }
14
15  this.finish();
```

A função "setResult()"está diretamente ligada à função "startActivityForResult()"usada para invocar a atividade "InvokeRest". Com "setResult()"podem ser devolvidos resultados para a atividade que invocou a segunda atividade, ou seja o contexto da aplicação deixa de estar na atividade "InvokeRest"e volta a estar na aplicação principal.

As atividades, no SO Android, possuem outro método que é relevante para o contexto da aplicação, o método "onActivityResult", que é chamado sempre que a atividade recebe o resultado que outra atividade lhe devolva, como é o caso deste exemplo que se está a dissertar.

Dentro deste método, as atividades tem de ser filtradas e os resultados processados afim de se saber se a execução terminou ou se a informação recebida tem de ser encaminhada para outro *plugin*. A execussão de um ciclo de programa eventualmente vai acabar neste método, pois o ciclo vai chegar a um ponto em que não vai existir mais nenhum *plugin* ao qual seja requerida informação.

A Figura 5.9 ilustra como se processa um destes ciclos de execução.

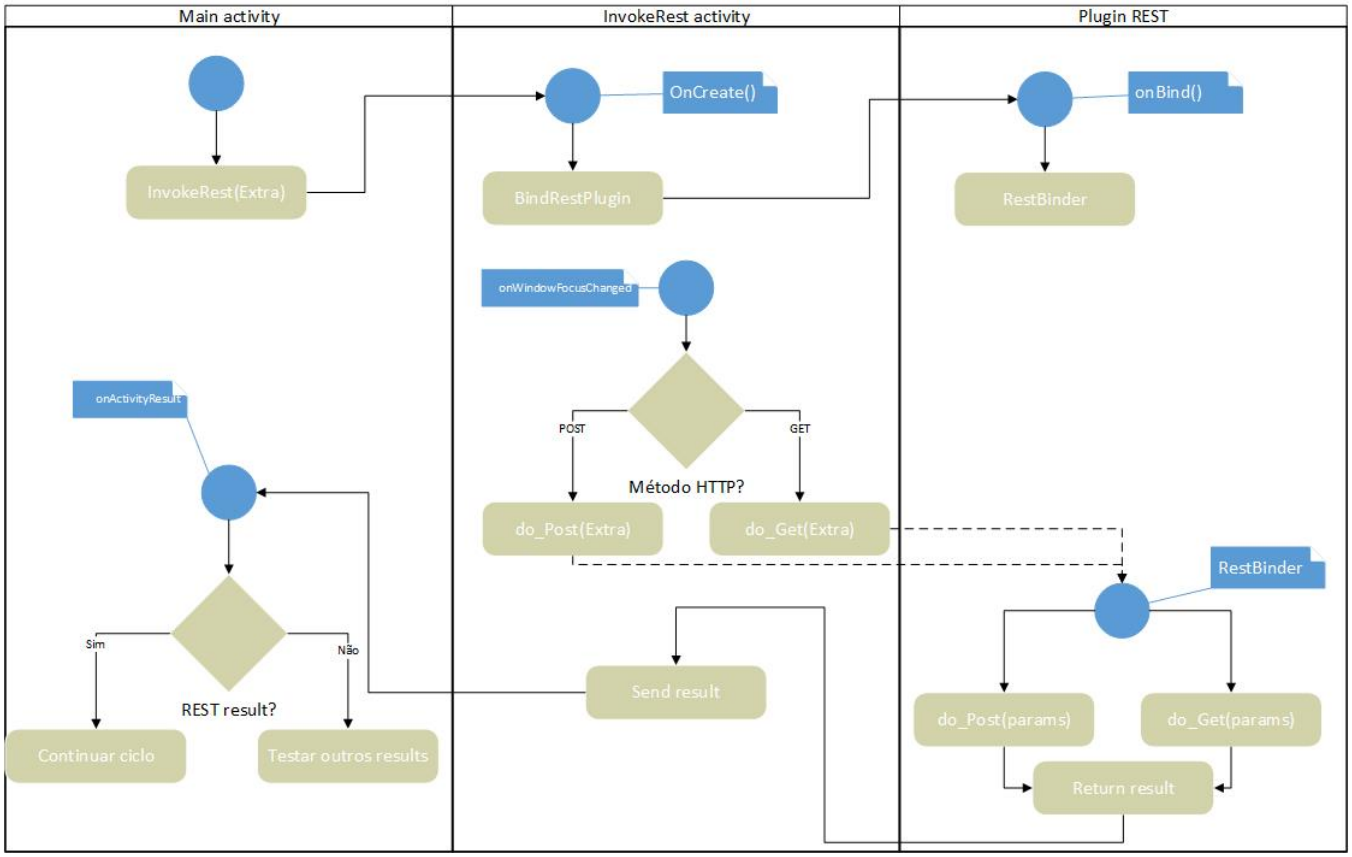


Figura 5.9: Fluxograma de um ciclo completo de atividades entre a aplicação principal e *plugin*

Plugin GCM

Um dos *plugins* a utilizar é o que permite a receção de GCMs no sistema.

A sua execução começa por adquirir uma identificação nos serviços da Google de *Cloud Messaging*. Esta identificação é única por dispositivo e pode mudar na altura de outro pedido de registo pelo que este identificador é enviado para um serviço web , o qual é armazenado na base de dados.

A resposta que o serviço web envia para o telefone, indica que se pode prosseguir com sucesso.

No caso de uma resposta positiva, o programa regista um *broadcast receiver* dedicado a receber GCMs.

Quando uma destas mensagens é enviada pelo serviço web para o identificador de um *smartphone* específico o resultado é o da Figura 5.10.

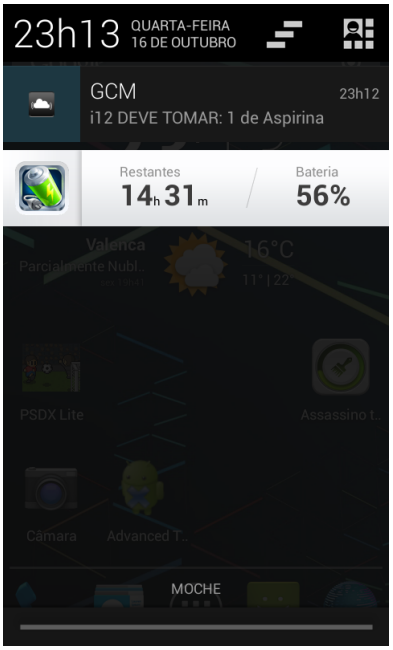


Figura 5.10: *Screenshot* relativo à receção de uma GCM

A mensagem é mostrada na gaveta das notificações do Android, e indica qual o medicamento a tomar e qual a quantidade.

Plugin **quedas**

Este *plugin* permite adicionar ao ecossistema a funcionalidade de detetar quedas. Aquando a sua primeira execução é criado um serviço que está constantemente à escuta de variações no acelerómetro. Quando as variações são o equivalente para ser considerado uma queda, o serviço criado inicialmente envia um *broadcast intent* para a aplicação do barramento de comunicação de modo a alertar uma queda.

No cenário de uma queda é enviado, para o servidor, informação referente à queda como a localização, a hora e a data para que possa ser apresentado no *website* dados gráficos com informação detalhada. Posteriormente é pedida, pela aplicação, alguma informação referente ao contacto telefónico do familiar do cliente cujo *smartphone* detetou uma situação

de queda. A Figura 5.11 mostra como a aplicação muda o seu aspeto visual quando existe uma situação de queda.

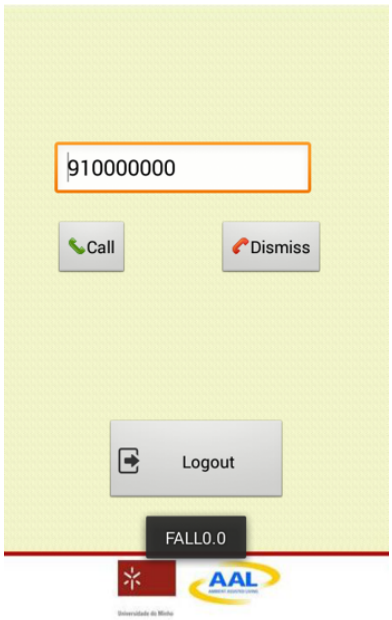


Figura 5.11: *Screenshot* relativo à deteção de uma queda

A partir destas novas opções de são apresentadas, o utilizador pode efetuar uma chamada para o número do seu familiar que foi disponibilizado automaticamente.

5.3 Botão de alerta

O botão de alerta é uma ferramenta usada para que o utilizador do sistema possa dizer que estão numa situação de perigo não detetada pelo ecossistema, ou seja que não se trata de uma queda.

Para esta dissertação, o botão de alerta foi implementado como um botão em *software* e não está inserido na lista de *plugins*. No entanto a sua utilidade faz com que seja parte do ecossistema. O facto de não estar integrado como um *plugin* prende-se pela necessidade de um ambiente gráfico e de interação direta para acionar o botão de emergência, o que não se enquadra em termos genéricos com o propósito da dissertação.

Idealmente este botão deveria ser realizado através de *hardware* externo e assim ser usado um *plugin* que registasse os eventos de pressionar esse botão externo.

A Figura 5.12 mostra o ambiente gráfico da aplicação de botão de alarme.

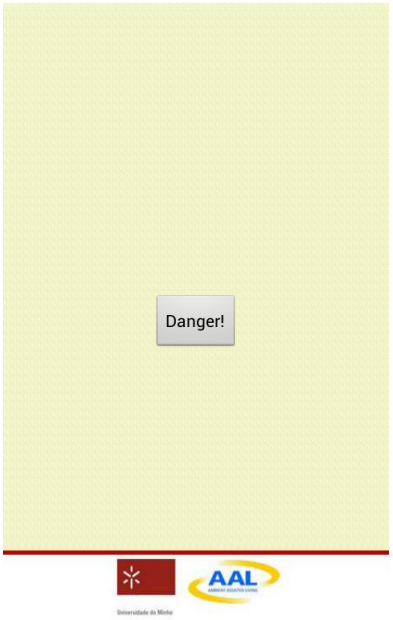


Figura 5.12: *Screenshot* da aplicação do botão de alarme

Quando o botão é pressionado, é enviado para o serviço web de receção de alarmes, a informação sobre o alarme: tipo de alarme, coordenadas GPS, data, hora e as credenciais de *login*.

O serviço web regista o pedido enviado na base de dados e o resultado pode ser visualizado através do website como mostra a Figura 5.13.

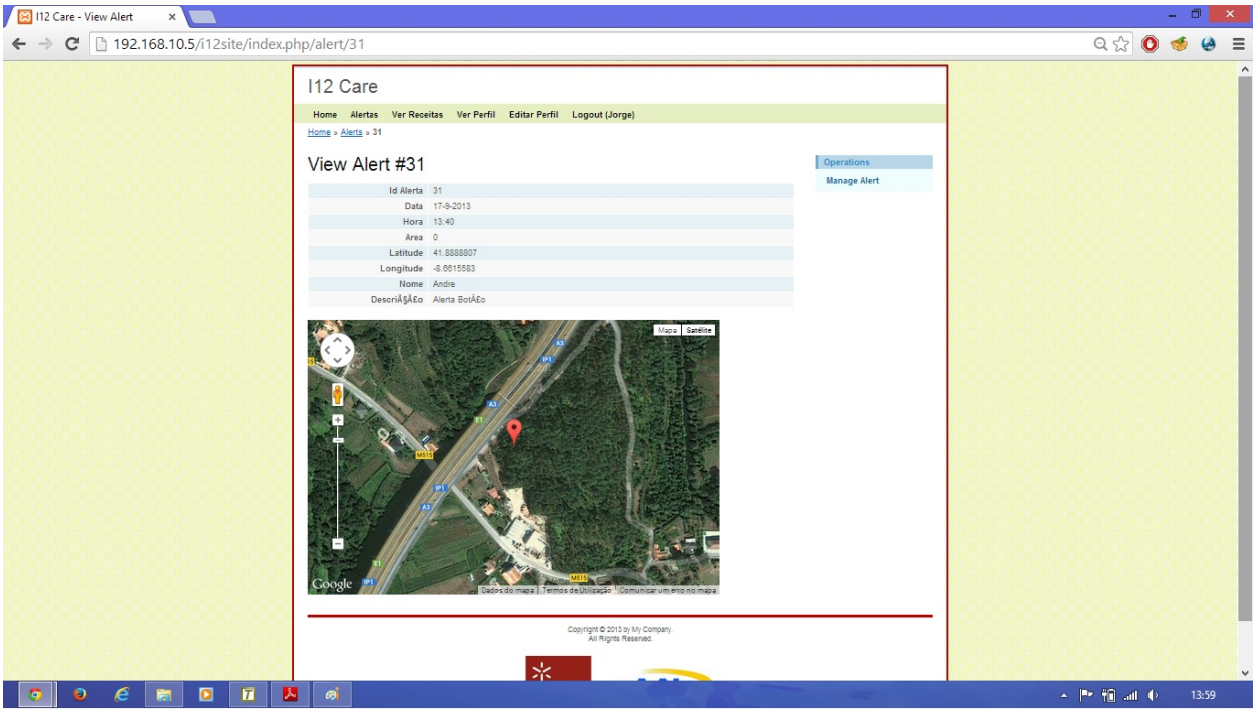


Figura 5.13: *Screenshot* do site de de um pedido de alerta através do botão

5.4 Ferramentas NFC

Um dos aspetos inovadores e chave desta dissertação é o uso da tecnologia NFC, que começa a ganhar uma dimensão considerável de uso em aplicações, como já foi anteriormente analisado.

Existem já várias aplicações que permitem a escrita em *tags* NFC de informação nos mais variados tipos de formatos, mas para o caso desta dissertação apenas seriam explorados dois tipos, o de lançar o marcador de números com um número de telefone pré-programado na *tag* ou então uma mensagem de texto. Para isso optou-se pela criação de um *software* próprio, cujas funcionalidades implementadas servissem de propósito de suporte a toda a dissertação, tendo código original e não reutilizado de outras aplicações, de outros *developers*. A Figura 5.14 apresenta o resultado final do menu principal da aplicação de escrita nas *tags* NFC.

Figura 5.14: Aspeto do menu principal da aplicação de escrita em *tags* NFC

A partir deste menu surgem duas hipóteses de navegação: escrever um número de telefone ou escrever um bloco de texto.

Para a escrita de texto, foram usadas as funções providenciadas pelos vários exemplos disponibilizados pela equipa de *Developers* da Google. A atividade de escrita de texto tem o aspeto da Figura 5.15.

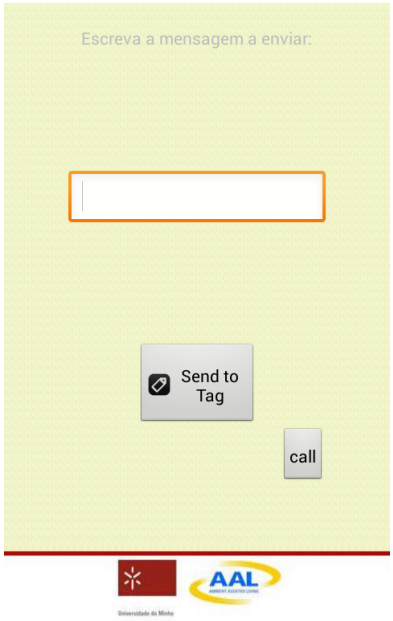


Figura 5.15: Aspeto da atividade de escrita de texto em *tags* NFC

Quando pressionado o botão de enviar para a *tag*, o texto que se encontra na caixa de texto é convertido para uma mensagem NDEF.

Em seguida a *tag* é analisada para saber se se trata de uma *tag Read-only* e se o seu tamanho permite gravar a mensagem desejada. Se estes dois casos forem verdade, a *tag* pode ser escrita com o texto inserido.

Se estes casos falharem, é verificado se a *tag* é protegida contra a escrita e nesse caso nada há a fazer. Se não o for, procede-se à formatação da *tag* e de seguida à sua escrita com o texto inserido.

Para a escrita de um número de telefone, não em formato de texto mas sim sob a forma de um link interativo, o processo é idêntico ao de escrever texto, com a diferença que há mensagem têm de ser inseridos alguns parâmetros indicativos da ação que se quer usar como link.

Neste caso, tem de ser adicionado um prefixo ”tel:”que está mapeado na posição 0x05. O código apresentado abaixo mostra como é criada a mensagem a escrever na *tag*, para que seja feita uma ligação ao marcador do telefone com o número pretendido.

```
String uniqueId = "NÚMERO_DE_TELEFONE";
byte[] uriField = uniqueId.getBytes(Charset.forName("US-ASCII"));
byte[] payload = new byte[uriField.length + 1];  //add 1 for the URI Prefix
payload[0] = 0x05;                               //prefixes tel to the URI
System.arraycopy(uriField, 0, payload, 1, uriField.length);  //appends URI
    to payload
NdefRecord rtdUriRecord = new NdefRecord(
     NdefRecord.TNF_WELL_KNOWN, NdefRecord.RTD_URI, new byte[0], payload);
```

O ambiente gráfico desta atividade, é similar ao da atividade de escrever texto e pode ser visto na Figura 5.16.

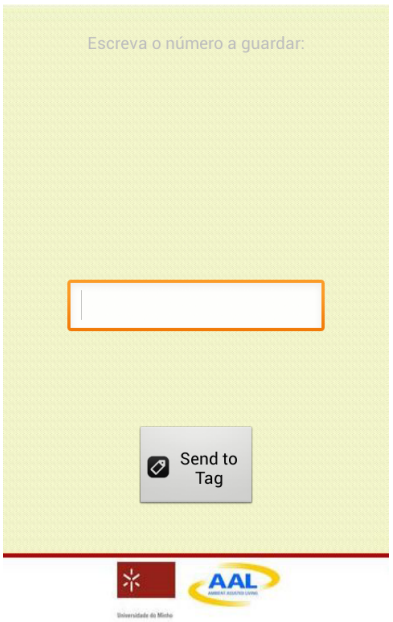


Figura 5.16: Aspeto da atividade de escrita de um número de telefone em *tags* NFC

Para se comprovar os resultados da escrita, liga-se o NFC do dispositivo e passa-se a *tag* programada junto do leitor, que se encontra geralmente na parte de trás do telefone, e imediatamente o resultado é mostrado pelo sistema como se pode ver pela Figura 5.17.

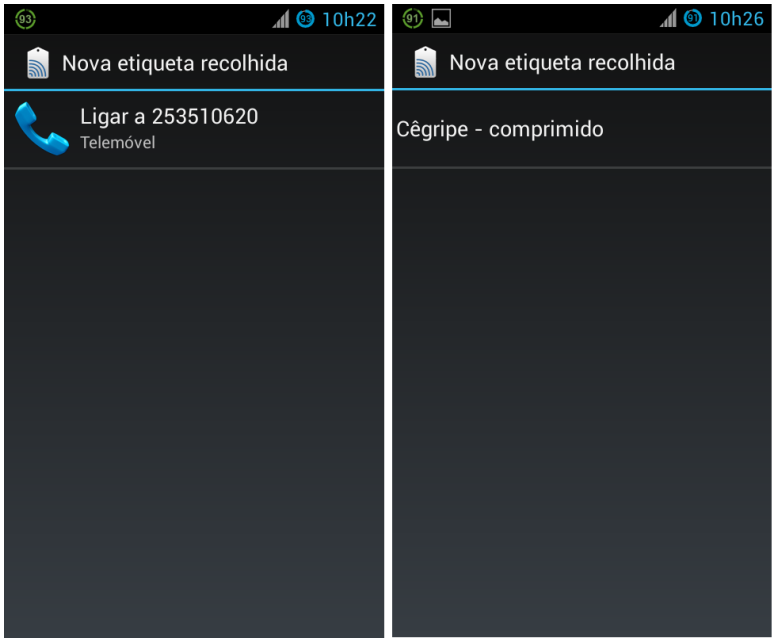


Figura 5.17: Aspeto da leitura de *tags* NFC

O ciclo de atividade de escrita e leitura é independente do barramento de comunicação, no entanto enquadra-se perfeitamente no ecossistema de auxilio ao cliente, ajudando a aumentar as suas funcionalidades.

Em resumo, o uso no NFC começa pela escrita de uma ou mais *tags* através da aplicação de ferramentas NFC criada no âmbito desta dissertação. O segundo passo é utilizar as *tags* programadas. Esta ação é feita com recurso a ferramentas já embutidas no próprio SO.

Capítulo 6

Conclusão

O objetivo da dissertação prendia-se em criar um ecossistema enquadrado no âmbito de AAL cujas principais características seriam a modularidade, possível através de uma estrutura de *plugins*, e o acesso maioritariamente feito sem a utilização de interação direta com o dispositivo. Esta interação mínima devia ser conseguida fazendo uso da tecnologia NFC.

Para além desta funcionalidade de implementação no dispositivo móvel, o sistema deveria incluir suporte ao sistema, através de aplicações *web* e de um *website* que permitiria uma melhor gestão e monitorização das tabelas existentes na base de dados.

6.1 Análise de resultados

Após uma análise cuidadosa e tendo em conta todos os resultados obtidos durante a realização desta dissertação, pode-se concluir que o método de aplicação não é o mais adequado pois aquando a criação de um *plugin* completamente novo, a aplicação base, barramento de comunicação, tem de ser alterado para que tome conhecimento do interface .aidl deste novo *plugin*. Aparte disso, no que diz respeito a novos *plugin* o sistema funciona na perfeição podendo-se jogar com várias combinações existentes.

O uso do NFC traz um novo tipo de dimensão ao AAL, abrindo portas a novas aplicações que facilitam comprovadamente o uso de novas tecnologias nas camadas idosas da popula-

ção, cuja dificuldade em acompanhar as tendências tecnológicas é mais que percetível. O interface sendo unificado, fazer *scan* a uma *tag* NFC, torna-se fácil de decorar e de realizar por pessoas com mais necessidades.

O facto da aplicação poder ser acompanhada de perto por um especialista na medicina através do site, acrescenta valor ao ecossistema, podendo monitorizar-se alertas de segurança e editar avisos que são enviados ao cliente do sistema.

Para o familiar do cliente este portal é em tudo vantajoso pois permite seguir mais de perto o seu familiar em qualquer lugar que se encontre.

6.2 Trabalho futuro

O passo mais imediato de trabalho futuro é a implementação de áreas de segurança as quais o cliente pode estar, a base de dados já se encontra preparada para este tipo de dados pelo que este seria a primeira ideia a implementar para um trabalho futuro.

Um possível trabalho futuro seria trabalhar numa maior segurança nas transferências de dados entre o serviço e o *plugin* REST já que de momento o único mecanismo de segurança existente é o envio constante de credenciais de login do utilizador sempre que se acede ao serviço.

Outro modo interessante seria implementar um *hardware* externo ao dispositivo para funcionar como botão de pânico.

Bibliografia

[1] OnAll, "Overview," 2012. [Online]. Available: http://oncaring.com/products.php

[2] O. Alliance, "The OSGi Architecture," 2013. [Online]. Available: http://www.osgi.org/Technology/WhatIsOSGi

[3] M. Vergara, P. Díaz-Hellín, J. Fontecha, R. Hervás, C. Sánchez-Barba, C. Fuentes, and J. Bravo, "Mobile Prescription: An NFC-Based Proposal for AAL," *2010 Second International Workshop on Near Field Communication*, pp. 27–32, 2010. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5476469

[4] A. T. Pereira, "Jornal de Negócios," 2012. [Online]. Available: http://www.jornaldenegocios.pt/empresas/detalhe/2012_11_27_android_tem_mais_quota_de_mercado_mas_sistema_da_apple_gera_mais_traacutefego.html

[5] R. Iglesias, J. Parra, C. Cruces, and N. G. de Segura, "Experiencing NFC-based touch for home healthcare," *Proceedings of the 2nd International Conference on PErvsive Technologies Related to Assistive Environments - PETRA '09*, pp. 1–4, 2009. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1579114.1579141

[6] L. Santos, "Simple True-Kare," 2012. [Online]. Available: http://www.simple-true-kare.com/

[7] True-Kare, "True-Kare," 2012. [Online]. Available: https://www.true-kare.com/

[8] OnAll, "Oncaring," 2012. [Online]. Available: http://www.oncaring.com/products.php?product=onall&section=how-onall-works

[9] Geek, "Operating System," 2010. [Online]. Available: http://www.geek.com/smartphone-buyers-guide/operating-system/

[10] B. McCarty, "The History of the Smartphone," 2011. [Online]. Available: http://thenextweb.com/mobile/2011/12/06/the-history-of-the-smartphone/

[11] C. Bonnington, "The evolution of iOS: from fake leather to iOS 7's stark redesign," 2013. [Online]. Available: http://www.wired.co.uk/news/archive/2013-06/12/ios-evolution

[12] I. C. USA, "Apple Cedes Market Share in Smartphone Operating System Market as Android Surges and Windows Phone Gains, According to IDC," 2013. [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=prUS24257413

[13] Apple, "Siri FAQ," 2013. [Online]. Available: http://www.apple.com/ios/siri/siri-faq/

[14] T. Worstall, "The iPhone 5S, Proof That Apple Has Given Up On Innovation," 2013. [Online]. Available: http://www.forbes.com/sites/timworstall/2013/09/17/the-iphone-5s-proof-that-apple-has-given-up-on-innovation/

[15] C. Mims, "Apple is no longer an innovative company, says the man who helped Steve Jobs design the Mac," 2013. [Online]. Available: http://qz.com/123388/hartmut-esslinger-says-apple-no-longer-innovative-helped-steve-jobs-design-the-mac/

[16] N. Arora, "Apple Bleeds Market Share With iPhone." [Online]. Available: http://www.forbes.com/sites/greatspeculations/2013/04/29/apple-bleeds-market-share-with-iphone/

[17] A. Inc, "Developers apple," 2013. [Online]. Available: https://developer.apple.com

[18] A. Inc., "Programming with Objective-C," 2012. [Online]. Available: http://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html

[19] J. I. Johnson, "Getting Started with iPhone and iOS Development," 2012. [Online]. Available: http://www.codeproject.com/Articles/88929/Getting-Started-with-iPhone-and-iOS-Development

[20] Microsoft, "XAML Overview," 2013. [Online]. Available: http://msdn.microsoft.com/en-us/library/cc189036(v=vs.95).aspx

[21] D. Android, "The AndroidManifest.xml File," 2012. [Online]. Available: http://developer.android.com/guide/topics/manifest/manifest-intro.html

[22] O. Alliance, "Benefits of Using OSGi," 2013. [Online]. Available: http://www.osgi.org/Technology/WhyOSGi

[23] T. E. Foundation, "equinox OSGi," 2013. [Online]. Available: http://www.eclipse.org/equinox/

[24] T. K. Project, "Knopflerfish - Open Source OSGi Service Platform," 2013. [Online]. Available: http://www.knopflerfish.org/

[25] T. A. S. Foundation, "Apache felix," 2013. [Online]. Available: http://felix.apache.org/

[26] D. Android, "Permissions," 2011. [Online]. Available: http://developer.android.com/guide/topics/security/permissions.html

[27] ——, "Activity," 2013. [Online]. Available: http://developer.android.com/reference/android/app/Activity.html

[28] ——, "Services," 2013. [Online]. Available: http://developer.android.com/guide/components/services.html

[29] ——, “Android Interface Definition Language,” 2012. [Online]. Available: http://developer.android.com/guide/components/aidl.html

[30] S. Developers, “Services with AIDL in Android,” 2012. [Online]. Available: http://developer.samsung.com/android/technical-docs/Services-with-AIDL-in-Android

[31] Techtopia, “Android Broadcast Intents and Broadcast Receivers,” 2013. [Online]. Available: http://www.techotopia.com/index.php/Android_Broadcast_Intents_and_Broadcast_Receivers

[32] S. O. Jr, “Is near-field communication close to success?” *Computer*, pp. 18–20, 2006. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1607943

[33] J. a. Lopes, “MobiPag permite-nos fazer pequenos pagamentos através do NFC,” 2013. [Online]. Available: http://www.revolucaodigital.net/2013/04/22/mobipag-fazer-pequenos-pagamentos-nfc/

[34] J. Ondrus and Y. Pigneur, “An Assessment of NFC for Future Mobile Payment Systems,” no. Icmb, 2007.

[35] D. Android, “android.nfc,” 2012. [Online]. Available: http://developer.android.com/reference/android/nfc/package-summary.html

[36] F. Reynolds, “Whither Bluetooth?” *Pervasive Computing, IEEE*, 2008. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4563901

[37] D. Android, “Bluetooth,” 2012. [Online]. Available: http://developer.android.com/guide/topics/connectivity/bluetooth.html

[38] W.-F. Alliance, “Discover and Learn,” 2013. [Online]. Available: http://www.wi-fi.org/discover-and-learn

[39] D. Android, “android.net.wifi,” 2013. [Online]. Available: http://developer.android.com/reference/android/net/wifi/package-summary.html

[40] S. Murugesan, "Understanding Web 2.0," *It Professional*, no. August, 2007. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4287373

[41] R. Dhand, "Web Services: A Trend Shift from Conventional Distributed Computing Model," *2009 Second International Conference on Computer and Electrical Engineering*, pp. 313–317, 2009. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5380466

[42] T. Downey, *Web Development with JAVA using Hibernate, JSPs and Servlets*. Springer, 2007.

[43] N. Chase, "Understanding web services specifications , Part 1 :," pp. 1–48, 2006.

[44] A. Rodriguez, "Restful web services: The basics," *Online article in IBM DeveloperWorks Technical Library*, pp. 1–11, 2008. [Online]. Available: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:RESTful+Web+services+:+The+basics#0

[45] X. Feng and Y. Fan, "REST: An alternative to RPC for Web services architecture," *2009 First International Conference on Future Information Networks*, pp. 7–10, Oct. 2009. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5339611

[46] T. A. S. Foundation, "What is Maven?" 2013. [Online]. Available: http://maven.apache.org/what-is-maven.html

[47] M. Lipton and S. Jang, "Job scheduling with Quartz," 2006. [Online]. Available: http://www.ibm.com/developerworks/library/j-quartz/

[48] Terracotta, "What is Quartz?" 2012. [Online]. Available: www.quartz-scheduler.org

[49] P. Morris, S.A. and Coronel, C. and Rob, *Database Principles: Fundamentals of Design, Implementation and Management*. Course Technology Cengage Learning, 2011. [Online]. Available: http://books.google.pt/books?id=vZj5RQAACAAJ

[50] W. Kent, “A Simple Guide to Five Normal Forms in Relational Database Theory,” 1982. [Online]. Available: http://www.bkent.net/Doc/simple5.htm

[51] E. F. Codd, *The RELATIONAL MODEL for DATABASE MANAGEMENT : VERSION 2.* ADDISON-WESLEY PUBLISHING COMPANY, 1990.

[52] Y. S. LLC, “The Fast, Secure and Professional PHP Framework,” 2013. [Online]. Available: http://www.yiiframework.com/