Universidade do Minho
Escola de Engenharia

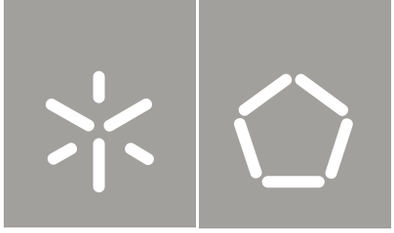
César Filipe Martins Meneses

Sistema de Apoio a Rondas de Segurança

outubro de 2013

César Filipe Martins Meneses Sistema de Apoio a Rondas de Segurança

UMinho | 2013



Universidade do Minho
Escola de Engenharia

César Filipe Martins Meneses

Sistema de Apoio a Rondas de Segurança

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 José Mendes

outubro de 2013

DECLARAÇÃO

Nome: César Filipe Martins Meneses

Correio eletrónico: meneses.cesar89@gmail.com

Tel./Tlm.: 913290150

Número do Bilhete de Identidade: 13559990

Título da dissertação:

Sistema de apoio a rondas de segurança

Ano de conclusão: 2013

Orientador(es):

José Mendes

Designação do Mestrado:

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

Escola de Engenharia

Departamento de Eletrónica Industrial

1. É AUTORIZADA A REPRODUÇÃO INTEGRAL DESTA DISSERTAÇÃO APENAS PARA EFEITOS DE INVESTIGAÇÃO, MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE.

Guimarães, 18/12/2013

Assinatura: _____

Agradecimentos

Em primeiro lugar gostaria de agradecer ao Professor Doutor José Mendes, por ter aceitado orientar-me e por todos os recursos fornecidos.

Agradecer aos meus colegas de laboratório e em especial ao Doutor Engº Nuno Cardoso pela ajuda e apoio que sempre demonstraram.

Um especial agradecimento ao meu amigo Engº Ricardo Pinto, que sempre me orientou, e foi essencial para que no inicio deste projeto eu começasse com o pé direito.

Gostaria de deixar aqui também uma palavra de apreço ao meu amigo e colega Jaime Costa que me ajudou ao longo das dificuldades que fui encontrando no decorrer do curso.

Por último, mas não menos importante, quero deixar o meu profundo agradecimento aos meus pais, que sem eles esta caminhada não se tornaria possível, por toda a confiança e apoio transmitido nos momentos bons e maus durante toda a minha vida académica.

Resumo

Nos dias de hoje cada vez mais são utilizados os *smartphones* devido à sua massificação durante os últimos anos. Estes tendem cada vez mais a ser utilizados para tarefas, que até então, eram executadas por outros dispositivos.

Esta dissertação é centrada no desenvolvimento de um sistema de apoio a rondas de segurança, com a utilização de um *smartphone Android*. É pretendido tirar proveito das tecnologias já integradas nos *smartphones* e monitorização em tempo-real do sistema.

Cada segurança deverá estar munido de um *smartphone* com ligação à internet e com NFC (*Near Field Communication*). NFC é uma tecnologia de transmissão de dados sem fios com distância menor a quinze centímetros que começa a surgir nos *smartphones* mais recentes. O segurança ao fazer a sua ronda deverá passar por diversos *checkpoints* sendo a tecnologia NFC utilizada para a leitura de informação dos mesmos. No *smartphone* o segurança terá disponível também a sua ronda atual e deteção de queda/pânico.

Os *smartphones* estarão ligados a um servidor que é gerido por um supervisor através de um *WebSite*. Neste, é possível visualizar todos os acontecimentos dos seguranças, controlar passagem por *checkpoints*, monitorizar alertas e designar novos *checkpoints* ou seguranças.

Pretende-se com o desenvolvimento desta dissertação, criar um sistema com a utilização de *smartphones* e fazer um protótipo que se poderá tornar num produto final.

Palavras-chave: NFC; GPS; rondas de segurança; *Android*; *checkpoints*; *smartphones*.

Abstract

The use of Smartphones has become significantly more popular. These are increasingly being used for tasks that until now were done by other devices. The aim of this project is the development of a support system for security rounds, using an android smartphone and the reduction of carried devices by the guard and a real time monitoring. Each guard must have a smartphone with network connection and NFC (Near Field Communication). NFC is a technology for wireless data transmission with low range that begins to be incorporated with the newer smartphones. While doing his round, the guard must pass through several checkpoints in which the NFC technology will be used to read the checkpoint details. Smartphone has other functionalities, such as, information about actual round and fall/panic detection.

A supervisor manages a server where all smartphones will be connected. He can view all events regarding guards: to control checkpoints presences, monitoring alerts and to assign new checkpoints or guards. In conclusion the objective of this thesis is to create a system that uses smartphones and make a prototype that could become a final product.

Keywords: NFC; GPS; Security Rounds; guard; checkpoint; real time; smartphones.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

ADT *Android Development Tools*

AWS *Amazon Web Services*

CCTV *Closed Circuit Television*

CRUD *Create, Read, Update, Delete*

DAO *Data Access Object*

DER *Diagrama Entidade e Relações*

DHTML *Dynamic Hyper Text Markup Language*

GCM *Google Cloud Messaging*

GPRS *General Packet Radio Service*

GPS *Global Positioning System*

HTTP *Hyper Text Transfer Protocol*

IDE *Integrated Development Environment*

JSON *JavaScript Object Notation*

MVC *Model View Controller*

NFC *Near Field Communication*

NIST *National Institute of Standards andTechnology*

NSA *National Security Agency*

REST *Representational State Transfer*

RFID *Radio-Frequency Identification*

RPC *Remote Procedure Call*

SAAS *Software as a Service*

SHA *Secure Hash Algorithm*

SSH *Secure Shell*

SSL *Secure Sockets Layer*

TLS *Transport Layer Security*

URI *Uniform Resource Identifier*

XML *Extensible Markup Language*

Capítulo 1

Introdução

Esta dissertação apresenta o trabalho de desenvolvimento de um sistema de apoio a rondas de segurança, o Better Patrol. Este foi realizado no ESRG (*Embedded Systems Research Group*) do Departamento de Eletrónica Industrial da Universidade do Minho no âmbito da dissertação do 5º ano do Mestrado Integrado em Engenharia em Eletrónica Industrial e Computadores. Consistiu no desenvolvimento de um *WebSite*, uma aplicação *Android* e um servidor. O segurança utiliza um *smartphone Android* para efetuar as suas rondas de segurança, onde realiza a leitura da informação dos *checkpoints* com a tecnologia NFC, permite a monitorização em tempo-real dos acontecimentos e envio de um sinal de alarme por parte do *smartphone* em caso de pânico. Utiliza também a tecnologia GPS (*Global Positioning System*) para obter a localização atual do segurança. A aplicação para o servidor tem acesso à base de dados para permitir a salvaguarda de todas as informações, no *WebSite* é feita a gestão e monitorização de todos os dados do sistema pelo supervisor.

As vantagens desta solução são: a diminuição dos dispositivos que o segurança tem que transportar, facilitar a verificação de *checkpoints*, permitir uma monitorização em tempo-real e a deteção de pânico caso o segurança esteja em perigo.

1.1 Motivação

Atualmente existem soluções para rondas em que o segurança tem um dispositivo dedicado para validar a sua presença/monitorização num determinado local.

As soluções móveis mais comuns para rondas de segurança, são um dispositivo dedicado que valida a presença dos seguranças pelos diversos *checkpoints*. Alguns destes dispositivos guardam a informação e ao fim do turno 'descarregam' a informação para um computador para verificar horários de presença[12].

Outros dispositivos já têm equipamento adicional onde é possível monitorizar os seguranças em tempo real. Estes possuem também um botão de emergência caso o segurança necessite de assistência [11].

Com a evolução das tecnologias móveis, as melhorias no processo de fabrico de componentes e com isso a massificação da utilização dos *smartphones* [1][2], abre novas oportunidades para soluções a problemas já existentes. Os *smartphones* têm a vantagem de terem várias tecnologias incorporada como, GPS, NFC, Wifi, o que permite uma redução do *time-to-market* do sistema.

NFC é uma tecnologia baseada em rádio frequência que permite transferência de dados num curto alcance entre dispositivos NFC e *tags Mifare*. Esta tecnologia está a ser cada vez mais utilizada no uso de pagamentos e serviços de transportes [16][17].

O sistema operativo mais utilizado pelos *smartphones* é o *Android*[1], o sistema operativo da *Google* para dispositivos móveis.

> "Android is an open source software stack that includes the operating system, middleware, and key applications along with a set of API libraries for writing mobile applications that can shape the look, feel, and function of mobile handsets."[3]

Nos próximos anos, grande parte do mercado, focar-se-á nas aplicações e serviços móveis [4], onde este sistema se encaixa. É também possível tirar partido da utilização de *smartphones*, a eliminação de dispositivos dedicados e limitados, a facilidade de utilização e a adição de novas funcionalidades.

Os *smartphones Android* também estão equipados com uma série de sensores, sendo os acelerómetros bastante relevantes para a deteção de quedas. Têm sido feitos estudos recentes para observar a resposta a várias ações humanas, como andar, sentar e cair. Através destes estudos é possível criar algoritmos para a monitorização destes acontecimentos para permitir criar alertas quando uma queda é detetada [18].

A crescente presença de redes móveis, como por exemplo, plano de dados de operadores móveis ou redes *Wireless* e com a massificação de *Web Services*, as aplicações móveis obtêm facilmente dados da Internet.

Estas tecnologias podem assentar num modelo de negócio emergente denomi-

nado por SaaS (*Software as a Service*). Consiste num modelo de desenvolvimento de software, onde são fornecidos serviços através da internet e são mantidos pelos próprios criadores do software. Assim, os utilizadores não necessitam de uma licença de software tendo como forma de pagamento uma mensalidade pelo serviço. O utilizador não tem que se preocupar com manutenção nem melhoramentos, pois estas tarefas estão ao cargo dos fornecedores do serviço [9][10].

Capítulo 2

Estado da Arte

Neste capítulo são descritas algumas das soluções já existentes no mercado, no que diz respeito aos sistemas para rondas de segurança.

2.1 Soluções Existentes no Mercado

Artsensor – Soluções de Segurança

A Artsensor é uma empresa no ramo da segurança eletrónica que existe há doze anos. Esta empresa fornece um produto de suporte a rondas de segurança, o CLOSEPATROL, figura 2.1.

"O Sistema de Gestão e Controlo de Rondas CLOSEPATROL, permite monitorizar as atividades de guardas, técnicos ou equipas de limpeza, durante as suas patrulhas de vigilância ou prestação de serviços no exterior."

O funcionamento desta solução é mostrado na figura 2.1. Os *checkpoints* são *tags* circulares com uma identidade única que podem ser associados aos *checkpoints* desejados. É possível também identificar os seguranças com um porta-chaves que tem associado o segurança. Os seguranças fazem as rondas onde a informação é guardada na memoria do dispositivo e no final é descarregada para o computador. No computador é possível fazer a gestão dos dados e obter relatórios segundo os variados requisitos. Estes dados podem ser impressos, enviados por e-mail, guardados em PDF ou *Excel*, entre outros.

Figura 2.1: Sistema ClosePatrol

Infocontrol

A Infocontrol foi fundada em 1984 e dedica-se à comercialização de componentes e sistemas nos ramos da automação industrial e dos sistemas para edifícios. Desde 1998 a Infocontrol representa os mais prestigiados fabricantes europeus de Sistemas de Gestão e Controlo de Rondas.

O sistema Datix, figura 2.2, é um sistema de controlo de rondas em que o segurança percorre a ronda com o equipamento, este equipamento recolhe os dados e mais tarde descarrega estes dados para o computador onde é possível fazer a gestão.

Figura 2.2: Datix

A empresa fornece também o sistema DatixWi-Track, figura 2.3. É um sistema de gestão de rondas em tempo real, onde é possível receber os dados na central e além disso tem outras funcionalidades como envio de alarme, chamada rápida, localização de GPS exterior e deteção de queda. As transmissões de dados são feitas através de um canal GPRS (*General Packet Radio Service*) de baixo custo.

Figura 2.3: DatixWi-Track

TimeKeepingSystems

Empresa fundada em 1986, é uma fabricante de produtos coletores de dados. Desenvolveram um produto de suporte a rondas de segurança, o PIPE, "O coletor de dados de ronda de segurança mais resistente do mundo". Foi projetado para a utilização sob condições e ambientes desfavoráveis e é resistente a choques, temperatura, água e humidade. A empresa disponibiliza também uma solução de software, o Guard1 Plus. Este software está disponibilizado para o sistema operativo *Windows* que permite a comunicação com o hardware bem como a impressão de relatórios.

Figura 2.4: PIPE

Real-Link

O sistema Real-Link, figura 2.5, foi desenvolvido pela empresa Real Time e utiliza telemóveis com suporte a tecnologia NFC para efetuar as rondas de segurança. O segurança percorre os seus *checkpoints* e faz a verificação utilizando o telemóvel para ler as tags NFC de cada *checkpoint*. Toda a informação é enviada através de GPRS, ficando disponível no *Website* do sistema.

Figura 2.5: Real Link

Esta dissertação segue o mesmo conceito deste sistema, no entanto o objetivo central é utilizar um *smartphone* Android com a vantagem da possibilidade de utilizar Wifi para a transferência de dados. Este sistema foi desenvolvido para funcionar no Symbian OS e utiliza GPRS para a transferência dos dados.

Capítulo 3

Análise do Sistema

Neste capítulo é feita a analise do sistema, seguindo a estrutura: na secção 3.1 são definidos os requisitos funcionais e não funcionais do sistema, na secção 3.2 é demonstrado o diagrama de casos de uso do sistema, na 3.3 é demonstrado o diagrama de blocos e na secção 3.4 são descritas as tecnologias e ferramentas utilizadas no projeto onde é feita uma abordagem sobre os componentes de software e hardware.

3.1 Requisitos do Sistema

3.1.1 Requisitos funcionais

A aplicação *Android* deve permitir o login, tanto ao segurança como ao supervisor. O login online pode ser efetuado com ou sem cartão *Mifare* pessoal. Existe a funcionalidade de login offline que só deve funcionar com a utilização de cartão *Mifare*.

Na aplicação *Android* o segurança deve ser capaz de iniciar uma aplicação CCTV como também iniciar uma ronda, fazer passagens por *checkpoints* através da leitura de cartões *Mifare* e visualizar a informação dos mesmos.

A aplicação *Android* deve permitir a monitorização da posição através do GPS e a monitorização do acelerómetro, de modo a detetar situações de pânico.

O supervisor pode associar cartões *Mifare* aos *checkpoints* e aos seguranças.

Só o supervisor pode utilizar o *Website* para gerir o conteúdo: criar, editar, eliminar *checkpoints* e seguranças. Visualização de *logs* e monitorização em tempo real.

O servidor deve estar munido de um conjunto de *web services* de modo a permitir as funcionalidades tanto do *Website* como da aplicação *Android*.

3.1.2 Requisitos não funcionais

A aplicação *Android* deve ser desenvolvida com recurso ao IDE *Eclipse* utilizando o SDK *Android*.

A aplicação deve ser desenvolvida utilizando a linguagem de programação Java.

O *Website* deve ser implementado com a *framework* de *javascript*, o *extjs*.

O servidor deve ser desenvolvido em Java com recurso ao IDE *Netbeans*, utilizando o formato de texto JSON para a transferência de informação e deve ter acesso a uma base de dados relacional *MySQL*.

O sistema é considerado tempo-real quando as notificação no *Website* aparecem com um tempo inferior a dez segundos, pois assim permite ao supervisor visualizar as informações e tomar as devidas precauções se necessário.

3.2 Sistema Better Patrol

Após definidos os requisitos pode-se estruturar o comportamento que é de esperar do sistema. Estas especificações do sistema podem ser representados através do diagrama de Casos de Uso, figura 3.1.

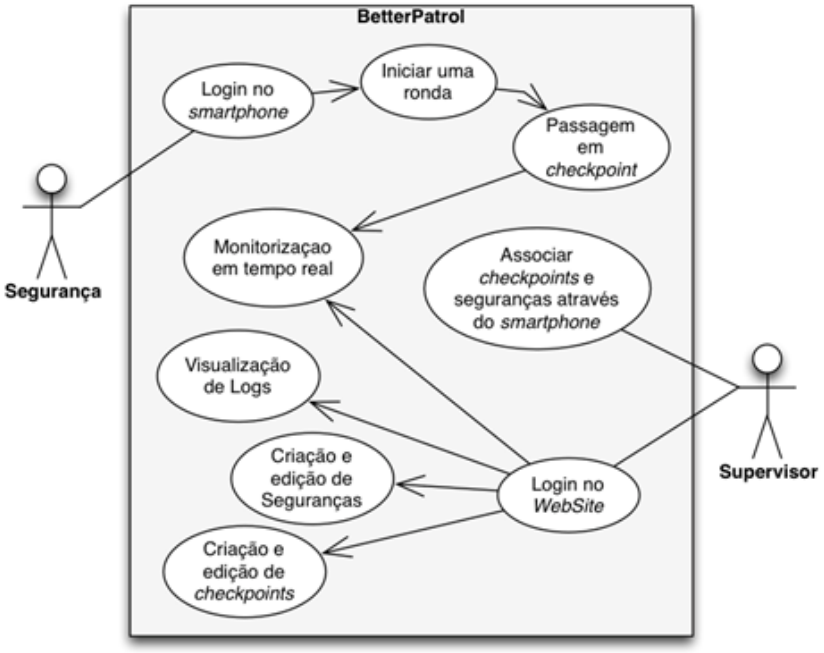


Figura 3.1: Casos de Uso

No diagrama estão presentes 2 atores, o segurança e o supervisor. O segurança deve operar através do *smartphone*, onde tem que efetuar o login pessoal, de seguida iniciar a sua ronda e fazer a passagem pelos *checkpoints*. Quando o segurança faz a leitura do *checkpoint* através da tecnologia NFC, é enviada a informação para o servidor e fica então disponível no *WebSite* onde o supervisor a pode visualizar. Além disso, no *WebSite* o supervisor pode gerir o sistema, com a possibilidade de criação e edição de seguranças,*checkpoints* e visualização de *Logs*. O supervisor pode também utilizar o *smartphone* para associar novos *checkpoints* e seguranças a *tags Mifare*.

3.3 Diagrama de Blocos

No Diagrama de Blocos da figura 3.2, estão representados os blocos constituintes do sistema, facilitando a interpretação do mesmo. A aplicação *Android* é executada num *smartphone*, este contem várias tecnologias, GPS, NFC, acelerómetro e ligação de dados. Com o *smartphone* é possível ler as informações contidas nas *tags Mifare*. A informação do *smartphone* é enviada para o servidor onde está alojado também o *WebSite* para permitir a visualização e gestão dos dados.

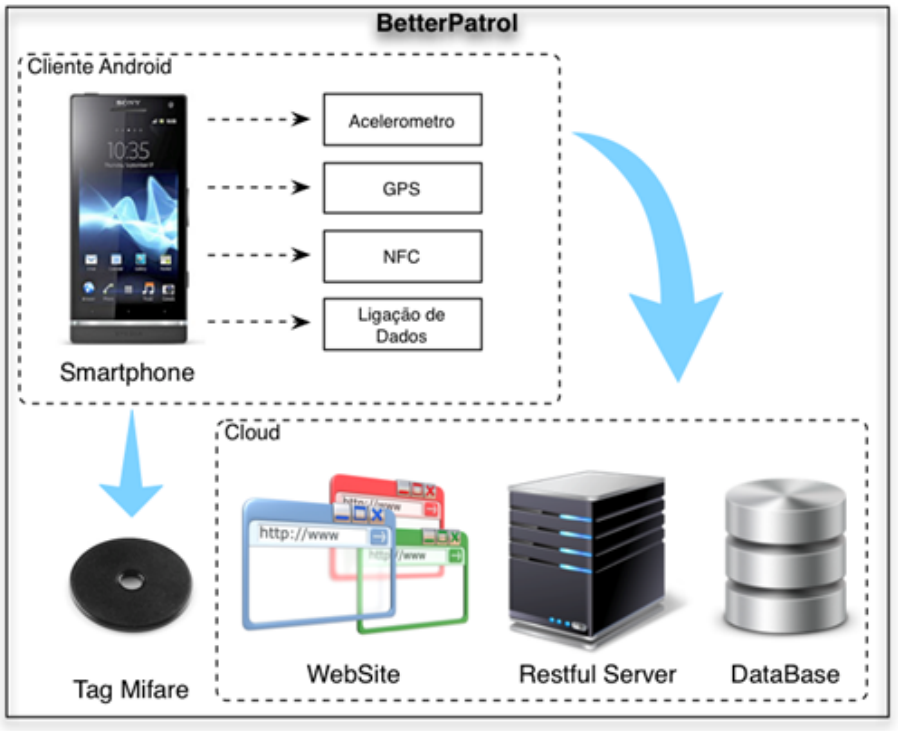


Figura 3.2: Diagrama de Blocos

3.4 Componentes de *Software*

3.4.1 Sistema Operativo *Android*

O sistema operativo *Android* tem tem como base o *kernel* do *Linux*, sendo desenvolvido e gerido pela *Google*. Inicialmente desenvolvido pela *Android Inc*, esta empresa foi adquirida pela *Google* em 2005. O *SDK Android* disponibilizado pela Google permite a criação de aplicações com a linguagem de programação Java, bem como a linguagem de programação C. Como o código fonte deste sistema operativo está disponível, existe uma comunidade de programadores que desenvolvem distribuições diferentes do *Android* como também bibliotecas para executar inúmeras tarefas. A arquitetura do sistema operativo *Android* está dividida em várias camadas: *'Linux Kernel'*, *'Libraries'*, *'Android Runtime'*, *'Application Framework'* e *'Applications'* (figura 3.3).

Figura 3.3: Arquitetura do Sistema Operativo *Android*

A camada básica do *Android* é o *Linux Kernel*. O sistema operativo *Android* foi desenvolvido, tendo como base o *kernel* 2.6 do *Linux* com algumas alterações feitas pela *Google*. Esta camada é responsável pela interação do hardware, contendo todos os drivers essenciais. Usa também todas as funcionalidades do *Linux* como gestão de memoria, gestão de processos, segurança, rede, etc. Assim sendo

funciona como uma camada de abstração entre o *hardware* e o *software*.

A camada *'Libraries'* contêm as bibliotecas nativas do sistema operativo *Android*. Estas bibliotecas, escritas em C ou C++, permitem a interação com determinado hardware. Entre estas bibliotecas, destacam-se algumas importantes como *OpenGL*, *WebKit*, *SQLite*, *SSL Libraries*.

'Android Runtime' consiste numa maquina virtual *Dalvik*, que é uma maquina virtual java projetada e otimizada para o sistema operativo *Android*. Permite que sejam criadas varias instancias proporcionando segurança, isolamento, gestão de memoria e suporte a multitarefa. Proporciona também uma serie de bibliotecas que permite aos programadores executar aplicações desenvolvidas em Java.

Na camada *'Application Framework'* existem programas que gerem as funções básicas do *smartphone*. Os programadores podem utilizar varias destas funções para desenvolver as suas aplicações. Algumas destas funções são *Activity Manager*, *Content Providers*, *Location Manager*, *Resource Manager*.

A camada superior da arquitetura do sistema operativo *Android* é *'Applications'* onde as aplicações são instaladas.

3.4.2 Ciclo de vida de aplicação *Android*

Cada atividade de uma aplicação *Android* tem um ciclo de vida associado. Esse ciclo está demonstrado na figura 3.4. Este ciclo de vida é separado nos seguintes métodos: *onCreate(), onStart(), onResume(), onPause(), onStop(), onDestroy(), onRestart()*. De seguida são descritas as utilidades de cada método:

- *onCreate()*: é chamada quando a atividade é criada. Aqui é onde se criam os componentes a preencher no layout. Este método também oferece um pacote de dados (*Bundle*) contendo o estado da atividade anterior.

- *onStart()*: é o método chamado quando a atividade fica visível para o utilizador.

- *onResume()*: chamada quando a atividade começar a interagir com o utilizador e esta ficará no topo da *stack*.

- *onPause()*: este método é invocado quando uma atividade é movida para o *'background'*, mas ainda não foi destruída. O método *onResume()* pode retomar esta atividade.

- *onStop()*: quando a atividade não esta visível ao utilizador este método é chamado. Dependendo do uso da atividade, esta pode ser reiniciada ou destruída.

- *onRestart()*: serve para reiniciar uma atividade, prosseguindo para o método *onStart()*.

- *onDestroy()*: o ultimo método antes de a atividade ser destruída. Pode acontecer porque o método *finish()* foi chamado, ou o sistema destrói esta instancia para recuperar espaço, quando tem poucos recursos.

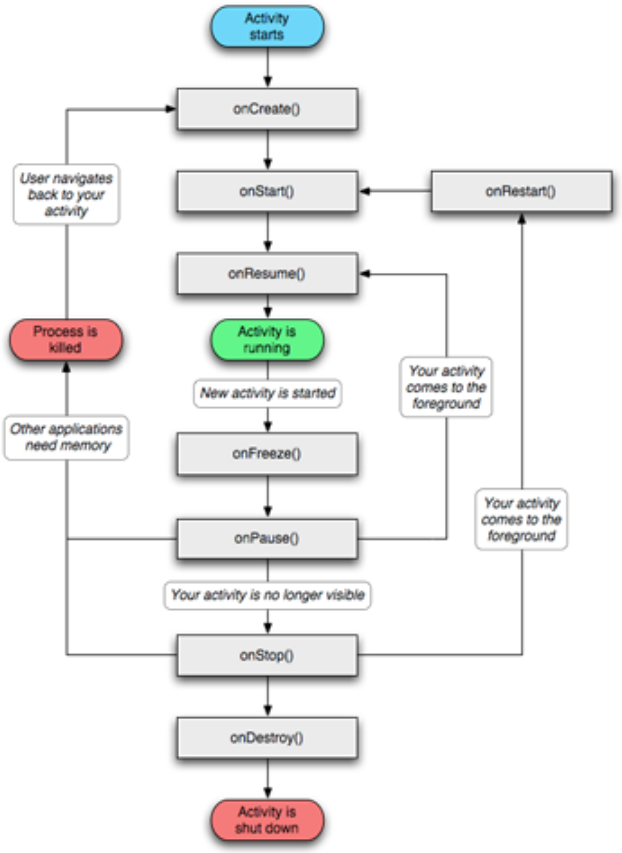


Figura 3.4: *Android* ciclo de vida de Atividades [21]

Para além das atividades, existem outros componentes utilizados nas aplicações *Android* sendo eles os *Intents*, *Services*, *Broadcast Receivers*, entre outros.

Os dispositivos *Android* são constituídos por diversos componentes de hardware que permitem a utilização de várias tecnologias, como o NFC, o GPS, acelerómetros, entre outros.

3.4.3 NFC

A tecnologia NFC é uma forma de comunicação sem fios de curta distancia (<15 centímetros) entre dispositivos. Este protocolo de comunicação foi o resultado de uma união das empresas, *Sony*, *Philips* e *Nokia* em 2004. É baseada na

tecnologia RFID (*Radio Frequency Identification*) com o *standard* ISO/IEC 18000-3 e comunica a 13,56MHz.

Existem dois tipos de dispositivos NFC, os dispositivos ativos e os dispositivos passivos. Os ativos podem ler e enviar informação, ler de uma *tag* NFC ou comunicar com outro dispositivo ativo para troca de informação. Estes dispositivos podem também alterar a informação contida nas *tags* NFC. Os dispositivos passivos apenas contém informação que os dispositivos ativos possam ler.

Esta tecnologia substitui os cartões tradicionais, permitindo efetuar pagamentos, transferência de dados para partilha de contactos, fotos e vídeos. Serve também como identidade eletrónica e cartões com chave de acesso. Os *smartphones* mais recentes trazem hardware incorporado que permite a utilização da tecnologia NFC.

3.4.4 GPS

O GPS, ou Sistema de Posicionamento Global é um sistema de navegação por satélite que fornece a localização e informação horária de um determinado dispositivo. Funciona idealmente com a ligação a quatro satélites GPS. Este sistema foi desenvolvido em 1973, com origens dos seus antecessores, estes de origens militares, criados em 1940 que foram utilizados na segunda Guerra Mundial. Atualmente este sistema é muito utilizado em vários tipos de serviços, desde a aviação, navegação marítima, transportes bem como utilização civil.

A precisão da localização é entre 10 a 20 metros, dependendo do dispositivo receptor. Os dispositivos com receptor GPS além da informação da posição, têm outras informações como a velocidade, direção e altitude. Estes dispositivos trazem suporte e software de gestão de mapas, o que permite a criação de rotas e localização de endereços.

3.4.5 Acelerómetro

Um acelerómetro é um sensor que mede acelerações físicas. Internamente é constituído por uma massa e mede a aceleração em g's. Com o acelerómetro é possível medir vibrações, rotações, colisões, velocidade, entre outros. Os acelerómetros tem muitas aplicações, tanto na biologia, indústria, medicina, navegação, transportes ou engenharia. Por exemplo, os discos rígidos têm um acelerómetro que mede movimentos bruscos e pára a rotação do disco de modo a evitar danos

internos.

Os acelerómetros mais sofisticados têm 3 eixos e são do tipo MEMS, *Micro-Electro-MechanicalSensors*. Normalmente, estes acelerómetros capacitivos ao sofrerem uma força externa, provoca uma deslocação na massa interna e em seguida uma alteração na capacitância. Estas alterações são detetadas pelo circuito integrado que calculam a aceleração. Os acelerómetros têm algumas especificações como número de eixos, largura de banda, sensibilidade, massa, intervalo de saída, entre outros.

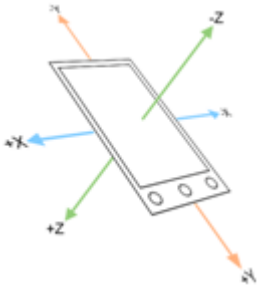


Figura 3.5: Acelerómetro

3.4.6 REST *Web Services*

Um *Web Service* é um conjunto de objetos de software instalados num servidor Web, que permite flexibilidade bem como a transferência de dados bidireccional e estruturados entre o servidor e aplicações [5][6].

Nesta área dos *Web Services*, destacam-se os REST (*Representational State Transfer*) *Web Services*, figura 3.6. Estes têm vindo a substituir as arquiteturas RPC (*Remote Procedure Call*) devido à sua simplicidade e escalabilidade, pois fazem total uso das capacidades da Web[7][8]. Utilizam métodos HTTP (*Hypertext Transfer Protocol*)explicitamente, sem estado. Utilizam a estrutura de URI's (*Uniform Resource Identifier*) onde são identificados os recursos disponíveis e podem ser utilizados em conjunto com métodos CRUD (*Create, Read, Update, Delete*) e utilizam transferência de dados XML (*Extensible Markup Language*) ou JSON (*Java Script Object Notation*). Nesta dissertação, os *Web Services* foram implementados utilizando a linguagem de programação Java e o servidor *Glassfish*.

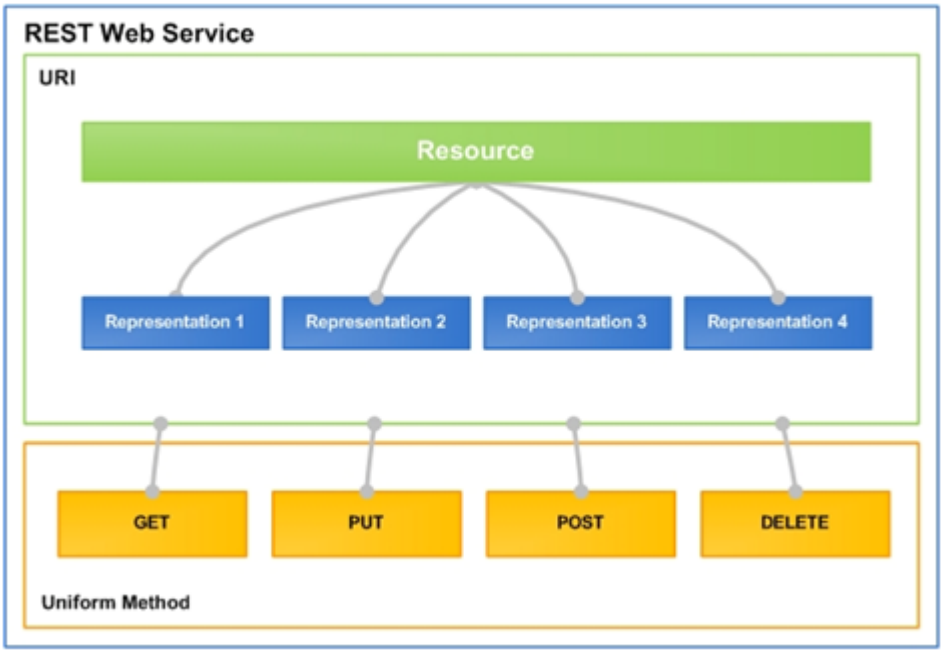


Figura 3.6: REST

3.4.7 Java e *Glassfish*

Java é uma linguagem de programação orientada a objetos, baseada em classes, que foi desenvolvida com o propósito de ser desenvolvida uma vez e ser executável em qualquer plataforma. A linguagem foi inicialmente desenvolvida por James Gosling na *SunMicrosystems* e disponível em 1995.

O código das aplicações é compilado para *bytecode*, o qual é interpretado pela maquina virtual Java, independentemente da plataforma. O Java é uma das linguagens de programação mais utilizadas para o desenvolvimento de aplicações Web cliente-servidor.

Para executar as aplicações Web cliente-servidor em Java, a *SunMicrosystems* desenvolveu o servidor *Glassfish*. Este é um projeto *open-source* que permite aos programadores desenvolver aplicações empresariais que são portáteis e escaláveis. O *Glassfish* tem uma grande comunidade que suporta o seu desenvolvimento e os principais objetivos é a robustez, qualidade de produção e além do mais é grátis para desenvolvimento, implementação e distribuição. Assim é possível utilizar um processo estruturado para a criação de aplicações, acelerando o seu desenvolvimento.

3.4.8 JSON

Java Script Object Notation é um formato de texto leve e independente de qualquer linguagem de programação. É de fácil compreensão para os humanos, é também mais rápido a ser analisado e gerado[20]. Os tipos básicos do JSON são os números, *String*, *Boolean*, *Arrays*, *Objects* e *null*. Na figura 3.7 é possível verificar um exemplo de um JSON, este tem exemplos de objetos simples bem como um *Array* de objetos.

```
{
    "firstName": "John",
    "lastName": "Smith",
    "age": 25,
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": 10021
    },
    "phoneNumbers": [
        {
            "type": "home",
            "number": "212 555-1234"
        },
        {
            "type": "fax",
            "number": "646 555-4567"
        }
    ]
}
```

Figura 3.7: Exemplo JSON

3.4.9 SHA-2

SHA-2, ou *Secure Hash Algorithm*, é um conjunto de funções de criptografia *hash* publicadas em 2001 pela NIST (*National Institute of Standards and Technology*) e NSA (*National Security Agency*), sendo o sucessor do SHA-1, mas com um nível maior de segurança. Uma função *hash* é um algoritmo que permite transformar um texto em valores hexadecimais com um tamanho fixo. É utilizada em vários protocolos, como por exemplo, TLS (*Transport Layer Security*) e SSL (*Secure Sockets Layer*). Este sistema pode ser utilizado para proteger *passwords* e verificar a integridade de cópias originais. Assim, os dados não são transferidos na sua forma original o que permite maior segurança a ataques computacionais.

3.4.10 *MySQL*

O MySQL é o sistema de gestão de bases de dados relacionais *open-source* mais utilizado mundialmente. Inicialmente foi desenvolvido pela empresa sueca MYSQL AB, tendo sido adquirido pela *Oracle Corporation* em 2006. O sistema é constituído por um conjunto de utilitários e serviços que permitem a vários utilizadores o acesso às base de dados que estão ativas no servidor. A base de dados é relacional, tendo como aspetos principais escalabilidade, confiável, fácil utilização e a rapidez de funcionamento.[26]

Figura 3.8: MySQL

3.4.11 ExtJS

Ext JS é uma *framework* de *Javascript* para o desenvolvimento de aplicações Web, que utiliza Ajax e DHTML (*Dynamic Hyper Text Markup Language*). Atualmente é desenvolvida pela empresa *SenchaInc*, esta ferramenta permite acelerar a criação de *WebSites*, através dos componentes oferecidos pela *framework*, tendo suporte a arquitetura MVC (*Model View Controller*).

Figura 3.9: Ext JS

3.4.12 MVC - *Model View Controller*

O MVC ou *Model View Controller* é um *pattern* arquitetural que tem como objetivo separar o código em três principais componentes: *model*, *view* e *controller*.

O *Model* é constituído pelos dados, lógica e funções da aplicação. A *View* é constituída pela interface gráfica, onde é possível representar os dados da aplicação. O *Controller* é responsável por processar o interface com o utilizador redirecionando as ações deste para funções do *Model* ou da *View*. Este *pattern* tem como objetivo principal a flexibilidade do código, permitindo uma maior facilidade na migração e reutilização de código. Assim permite ao programador alterar uma componente sem afetar as restantes.

3.4.13 DAO - *Data Access Object*

O DAO ou *Data Access Object* é uma camada de abstração entre a base de dados e os componentes de logica da aplicação. Como as comuns aplicações utilizam uma base de dados para salvaguardar os dados do sistema é necessário existir uma logica de acesso à base de dados. Esta camada contém os métodos necessários para aceder à informação contida na base de dados. Se houver uma alteração de base de dados no sistema é apenas necessário efetuar alterações nesta camada para tornar as comunicações entre a base de dados e a logica da aplicação possíveis, evitando uma alteração na logica da aplicação.

3.5 Ferramentas de Desenvolvimento

3.5.1 Eclipse

O *Eclipse* é um IDE (*Integrated Development Environment*), grátis e *open-source*, que foi originalmente desenvolvido para desenvolver aplicações na linguagem de programação Java. É possível instalar *plugins* para estender as capacidades deste IDE, como por exemplo, *toolkits* para programar noutras linguagens de programação. Assim, é possível instalar o *plugin* ADT (*Android Development Tools*) para ser possível o desenvolvimento de aplicações para *Android*. O ADT expande as capacidades do Eclipse e permite uma criação rápida de novos projetos *Android*, adicionar pacotes baseados na *framework Android* e a possibilidade de fazer debug das aplicações.

Figura 3.10: Eclipse

3.5.2 *MySQL Workbench*

O *MySQL Workbench* é uma aplicação visual para a criação e manutenção de base de dados. Permite a criação de tabelas e ligações de uma base de dados. Facilita o processo de criação, ao gerar o código correspondente ao projeto desenvolvido através do interface gráfico. É também possível a conexão ás base de dados para a alteração de dados, bem como a criação de *queries* SQL.

3.5.3 *NetBeans*

O *NetBeans* é um IDE, grátis e *open-source*, para programação na linguagem Java, mas com a possibilidade de programação noutras linguagens. Foi inicialmente

desenvolvido por um estudante da Universidade Charles de Praga. Mais tarde foi adquirido pela *SunMicrosystems* que tornou o software *open-source* e a comunidade começou a crescer cada vez mais. Suporta vários tipos de desenvolvimento Java, como por exemplo, Java SE, Java EE. Facilita a criação de aplicações Web em Java e tem suporte ao servidor *Glassfish*.

Figura 3.11: NetBeans

3.6 Componentes de *Hardware*

3.6.1 *Mifare*

Mifare é uma tecnologia sem fios proprietária da NXP *Semiconductors*. Utiliza o protocolo de comunicação ISO14443A, funciona a 13,56MHz e é alimentada através da própria comunicação sem fios. Podem ser utilizados para guardar informação de identidade, valores monetários ou como bilhetes. Existem vários tipos de cartões *Mifare*, como por exemplo *Classic*, *Ultralight*, *DESFire*, entre outros. A estrutura da memoria é representada na figura 3.12. Os cartões *Mifare Classic* são os mais utilizados, onde existem 3 versões: 320 bytes, 1024 bytes e 4096 bytes. Em 1994 surgiram os cartões *Mifare Classic* 1k que guardam a informação dividida em 4 blocos de 16 bytes por 16 sectores, onde cada sector é protegido com duas chaves diferentes, A e B, que são guardadas no bloco 3 de cada sector. A autenticação com estas chaves permite operações como leitura e escrita[22]. Apesar das comunicações entre o cartão e o leitor serem encriptadas, e a informação não estar acessível sem o acesso às chaves, esta tecnologia é vulnerável [23].

		Byte Number within a Block														
Sector	**Block**	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	**Description**
15	3	Key A						Access Bits				Key B						Sector Trailer 15
	2																	Data
	1																	Data
	0																	Data
14	3	Key A						Access Bits				Key B						Sector Trailer 14
	2																	Data
	1																	Data
	0																	Data
:	:																	
:	:																	
:	:																	
1	3	Key A						Access Bits				Key B						Sector Trailer 1
	2																	Data
	1																	Data
	0																	Data
0	3	Key A						Access Bits				Key B						Sector Trailer 0
	2																	Data
	1																	Data
	0																	Manufacturer Block

Figura 3.12: Estrutura de Memoria Mifare

3.6.2 *Smartphone*

O *smartphone* escolhido foi o Sony Xperia S, lançado em Fevereiro de 2012, pois contem hardware que permite a utilização da tecnologia NFC. Além disso, este *smartphone* é capaz de desempenhar as funções normais como acesso a redes *Wireless*, GPS, acelerómetro, entre outros [24].

Figura 3.13: Sony Xperia S

Capítulo 4

Modelação do Sistema

Neste capítulo é feita a descrição dos componentes do sistema *Better Patrol*:

- Base de dados;
- Aplicação *Android*;
- *WebSite*;
- Servidor.

4.1 Base de Dados

Este sistema utiliza uma base de dados relacional, onde é possível guardar a informação. Para isso foi utilizado o *MySQL* por ser uma base de dados *open-source* e sem licença. A base de dados foi criada utilizando o utilitário *MySQLWorkbench*, que permite criar as tabelas e as suas relações de uma forma mais intuitiva. Na figura 4.1, é apresentado o DER (Diagrama Entidade e Relações) que demonstra o modelo de dados da base de dados do sistema.

O sistema foi desenvolvido para suportar diversas entidades separadas. Assim, a base de dados pode ser analisada começando pela tabela 'Empresa'. Esta tabela guarda a informação das empresas existentes no sistema, como o nome, o estado de funcionamento e o endereço de servidor. Este endereço de servidor foi inserido para permitir separar as ligações de empresas para servidores dedicados.

Existem duas tabelas para guardar as configurações de cada empresa. Uma tabela que guarda o tipo de configuração e outra para guardar o valor correspon-

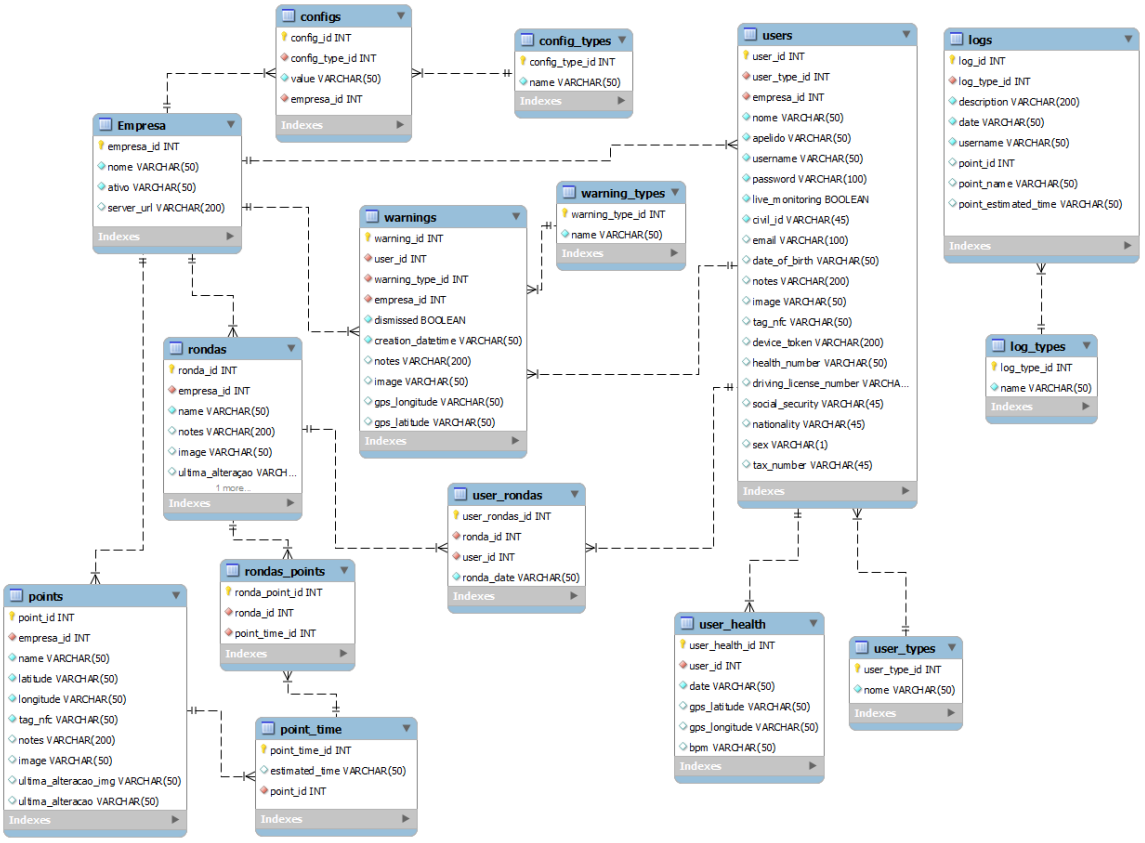


Figura 4.1: Diagrama ER

dente.

Na tabela 'users' são guardados todos os seguranças do sistema correspondente a uma determinada empresa. Nesta tabela é necessário guardar informação importante relativa ao segurança, no entanto os dados mais importantes para o sistema estão relacionados com os seguintes campos:

- *username* – Nome de utilizador que permite efetuar o Login.

- *password* – Palavra passe para efetuar o Login.

- *live_monitoring* – Este campo é utilizado para definir se o dispositivo deste segurança deverá enviar informações em tempo real sobre a sua posição.

- *device_token* – Esta chave serve para identificar qual o dispositivo que está associado a determinado segurança no momento.

A tabela 'users' está relacionada com mais 4 tabelas além da tabela 'Empresa'. São as tabelas 'user_types', 'user_health', 'warnings' e 'user_rondas'.

A tabela 'user_types' é utilizada para identificar o tipo de segurança, neste caso ele poderá ser segurança ou supervisor.

Na tabela 'user_health' é guardada a informação relativa dos dados provenientes da monitorização em tempo real.

'Warnings', esta tabela está relacionada com os seguranças e com a empresa à qual eles pertencem. Aqui são salvaguardados os dados relativos aos avisos originados pela aplicação *Android*, como por exemplo, uma situação de pânico.

A tabela 'user_rondas' que está relacionada com os seguranças, em conjunto com as tabelas 'rondas', 'rondas_points' e 'point_time', fazem parte de uma funcionalidade não implementada, mas que ficou decidido deixar o suporte à mesma. Esta funcionalidade consiste na possibilidade da existência de rondas com uma ordem de *checkpoints* específica, com tempos e datas definidas, tanto nos *checkpoints* como nas próprias rondas. Estas rondas poderiam estar relacionadas com seguranças e ser agendadas em calendário.

Na tabela 'points' é guardada a informação correspondente aos *checkpoints* do sistema de cada empresa. Aqui é possível guardar informações como o 'nome', 'latitude', 'longitude', 'notas','tag_nfc', entre outros. É a partir da chave 'tag_nfc' que é possível associar os *checkpoints* a determinadas localizações.

Nas tabelas 'log_types' e 'logs', são guardadas todas as informações relativas às ações executadas no sistema.

Depois de modelar o sistema, utilizando o *MySQLWorkbench*, o código *MySQL* correspondente foi gerado de forma automática, para ser executado. Juntamente com este código foi também inserido o código de inserção de informação, com dados de teste.

4.2 Aplicação *Android*

A aplicação *Android* é o componente do sistema que tem mais interação com os seguranças. O fluxo de execução da aplicação para o *smartphone Android* é demonstrado na figura 4.2.

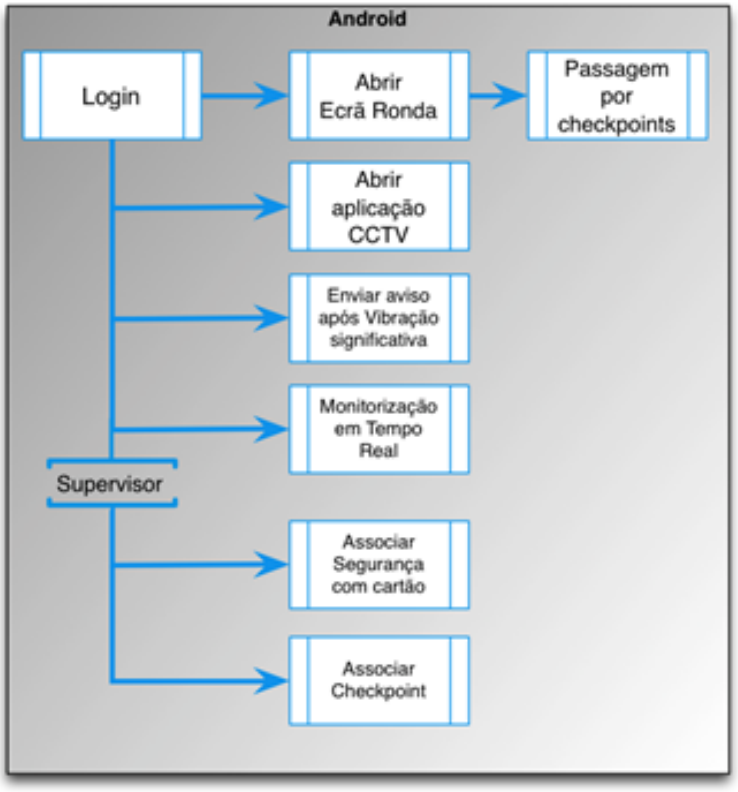


Figura 4.2: Fluxo aplicação Android

O *login* pode ser feito em modo *offline* ou em modo *online*. Para o modo *online* o dispositivo deve ter uma ligação à internet ativa e pode ser feito com a inserção de texto ou com a passagem do cartão *Mifare* pessoal. No modo *offline* o segurança deve fazer o login com o seu cartão pessoal. O fluxograma do login pode ser verificado na figura 4.3.

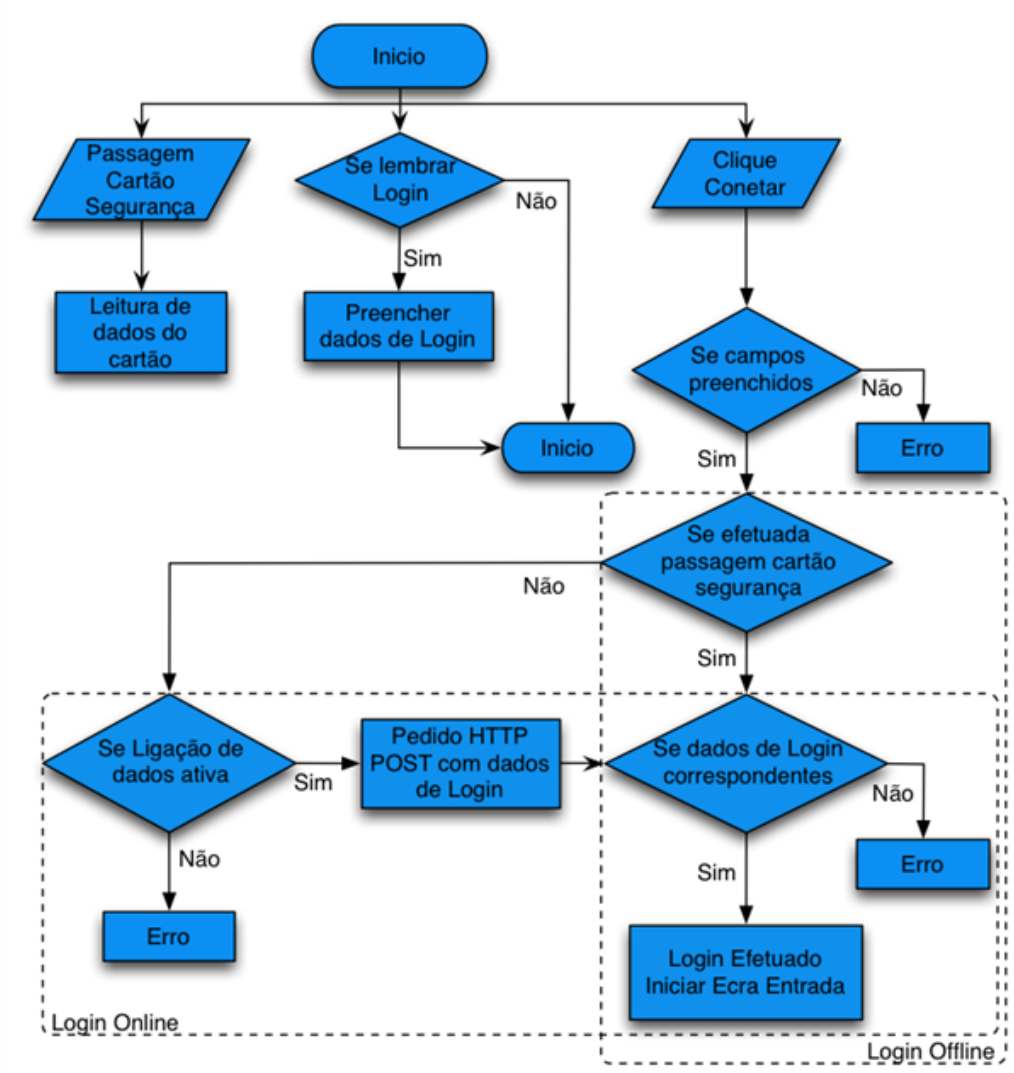


Figura 4.3: Fluxograma *Login*

No *Login online* é enviado um pedido HTTP POST para o servidor, onde vão inseridos o *username* e a *password* encriptada com o algoritmo SHA-256. Caso o *login* seja válido a resposta por parte do servidor consiste no id de segurança, o id da empresa ao qual pertence o segurança, o tipo de utilizador, segurança ou supervisor, o estado da monitorização em tempo real e o endereço ao qual o dispositivo deverá fazer os pedidos. Este endereço foi adicionado caso mais tarde seja pretendido ter empresas separadas em máquinas diferentes/dedicadas.

No *Login offline* o segurança passa o seu cartão *Mifare* pessoal no *smartphone* e insere os dados de *username* e *password*. É então feita uma verificação dos dados guardados no cartão e os dados inseridos pelo segurança. Caso estes sejam validados, o segurança começa a aplicação no modo *offline*.

Após feito o login, é criado um serviço para correr em *'background'*, figura 4.4. O serviço é responsável por executar várias tarefas. Caso o *login* seja feito em modo *offline*, este procura efetuar o *login* automático, quando o *smartphone* obtiver ligação de dados ativa.

Se o *Login* for verificado, o serviço procede ao pedido de uma nova chave GCM (*Google Cloud Messaging*). Após recebida esta chave, é enviada uma notificação ao servidor com a chave para permitir futuras mensagens *push*.

O serviço inicia a atualização dos dados contidos no *smartphone*, isto é, a informação de *checkpoints*. Para isso é utilizado como comparação a última data de alteração do *checkpoint* e assim é possível verificar se é necessário descarregar uma atualização. É também verificado se a imagem foi alterada pela data de ultima alteração da imagem, evitando o *download* de dados repetidos.

O serviço ativa igualmente a monitorização do acelerómetro para enviar um aviso para o servidor, caso seja detetada uma situação de panico.

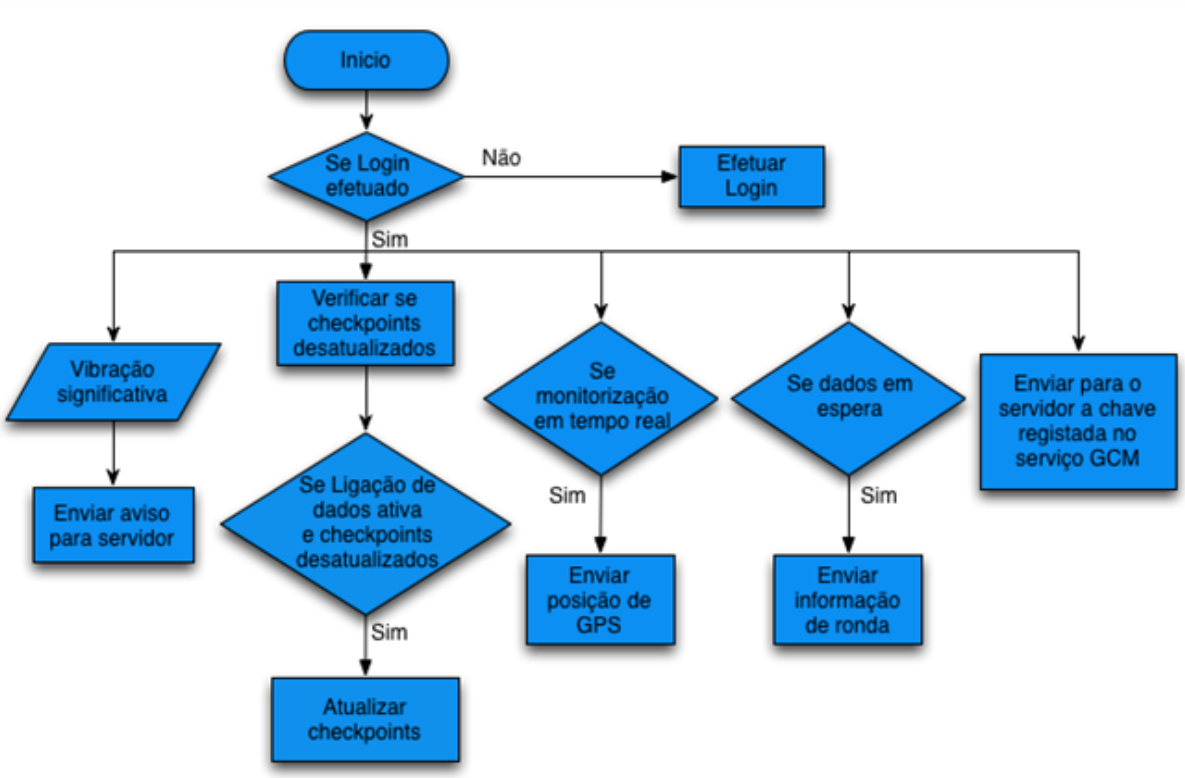


Figura 4.4: Fluxograma Serviço

Caso este utilizador tenha ativa a sua monitorização em tempo real, este serviço está encarregue de enviar a informação da sua posição atual através do sinal de GPS. Esta monitorização é definida pelo supervisor no *WebSite*. Quando o supervisor efetuar a alteração desta definição é enviado uma mensagem *push* do

servidor para o *smartphone* do segurança através do GCM, o seu funcionamento pode ser demonstrado na figura 4.5. As mensagens deste serviço têm um tamanho máximo de 4Kb, que são suficientes para enviar uma simples mensagem ao *smartphone* do segurança para sinalizar que houve uma atualização. Em seguida é feito um pedido HTTP para receber os novos dados.

Quando o segurança está a efetuar uma ronda é possível que este perca a sua ligação de dados, no entanto, todos os *checkpoints* registados serão guardados para mais tarde serem enviados, quando for retomada a ligação de dados. Este serviço ao detetar ligação de dados, verifica se tem conteúdo a enviar para o servidor. Deste modo a informação da ronda não é perdida caso haja uma falha na ligação. Este serviço permanece ativo durante todos os ecrãs, evitando assim a repetição de código em todos eles.

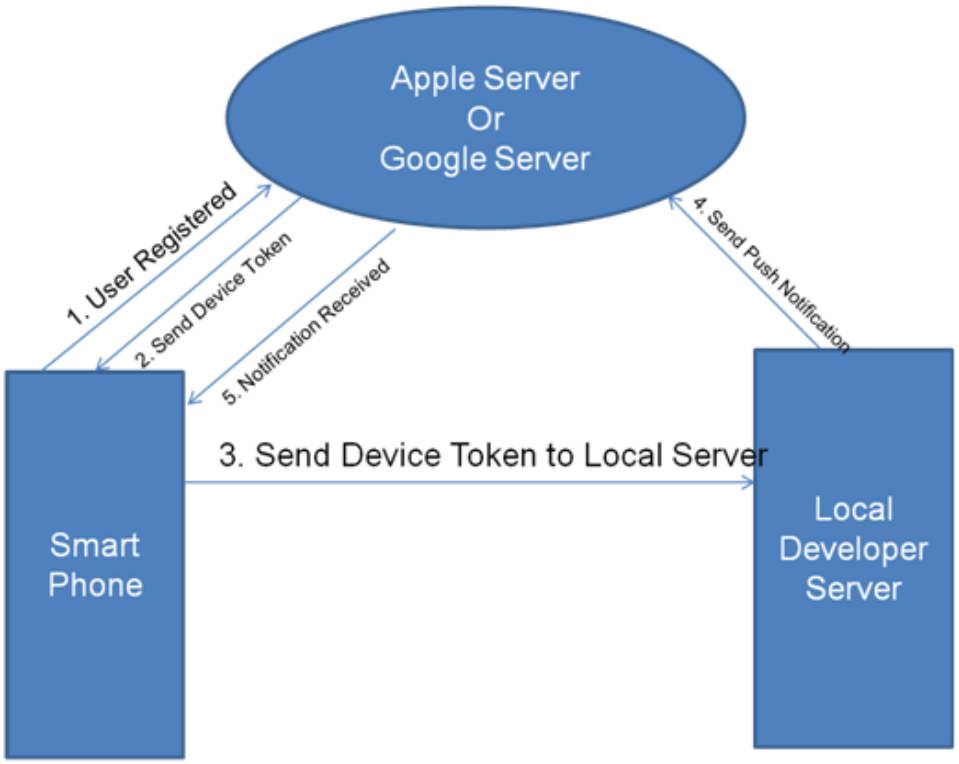


Figura 4.5: GCM *Push Notifications*

Depois do login o segurança pode iniciar uma nova ronda e efetuar a passagem por *checkpoints* através da leitura de cartões *Mifare*. Pode visualizar os pontos por onde passou, bem como, detalhes e imagem do *checkpoint*. É feito o download destes dados logo após a passagem por um *checkpoint* caso exista uma ligação de dados ativa. Estes dados, são então guardados no *smartphone* de modo a que na próxima passagem não seja necessário efetuar o *download* novamente. No modo *offline* é guardada a informação do cartão *Mifare* de modo a que quando a

ligação de dados for reestabelecida, se possa proceder ao pedido dos dados correspondentes.

O segurança tem a possibilidade de definir um botão de atalho para uma aplicação para câmaras de CCTV.

Como já foi referido anteriormente o valor do acelerómetro é monitorizado para detetar algum evento de perigo, como uma situação de queda ou panico. Com base na equação estudada no artigo [18],

$$|A_t| = \sqrt{|A_x|^2 + |A_y|^2 + |A_z|^2} \tag{4.1}$$

e após algumas amostras foi definido o valor de 25 que ao ser ultrapassado envia o aviso para o servidor.

Ainda na aplicação *Android*, o supervisor poderá efetuar o login, onde terá todas as funcionalidades da versão para segurança, bem como a possibilidade de programar cartões pessoais *Mifare* para seguranças e *tags* para *checkpoints*.

4.3 *WebSite*

As ações relativas ao *WebSite* são demonstradas na figura 4.6.

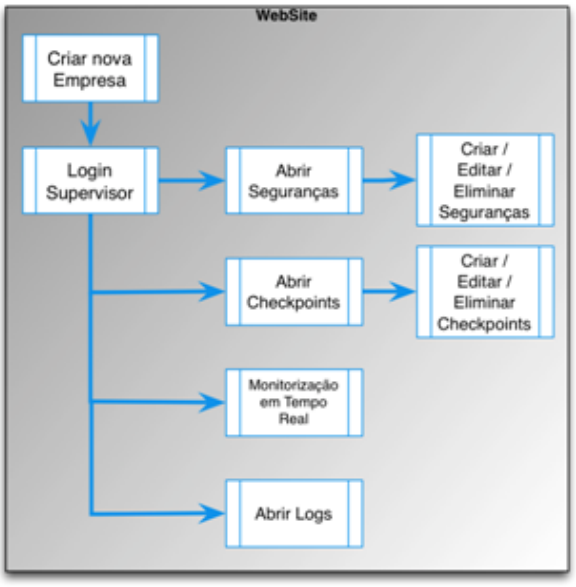


Figura 4.6: Fluxo WebSite

No *Website* pode ser criada uma nova entidade, uma empresa. Ao criar uma nova empresa deve ser criado um perfil de supervisor para permitir o login.

Como referido anteriormente o login no *Website* só pode ser efetuado pelo supervisor onde este pode executar as ações de listar, criar, editar seguranças e *checkpoints*.

Na criação de segurança ou *checkpoint*, o supervisor deve respeitar todos os campos obrigatórios a serem preenchidos, pode também efetuar o *upload* de uma imagem para o servidor associado a um segurança ou a um *checkpoint*.

O supervisor tem a possibilidade de monitorizar os eventos em tempo real e a visualização de *Logs*.

Quando o supervisor edita um segurança e caso altere o estado da monitorização em tempo real é enviada uma mensagem *push* do servidor para o dispositivo do segurança, de forma a que o dispositivo comece a enviar a informação do GPS atual do segurança.

4.4 Servidor

O servidor *Glassfish* é responsável por responder aos pedidos efetuados pela aplicação *Android* e pelo *WebSite*. Estes são pedidos HTTP POST, onde a informação é transferida utilizando o formato de texto JSON. O servidor ao receber os pedidos, na camada *Services*, direciona-os para a camada *Business*.

A camada *Business* tem todas as funções necessárias para responder às solicitações e é responsável pela 'inteligência' do servidor.

Na camada DAO, existem todos os métodos que permitem a ligação da camada *Business* à base de dados.

Por sua vez, a camada *Model* disponibiliza as classes correspondentes à estrutura da base de dados *MySQL*, tornando mais fácil a manipulação da informação. O fluxograma do servidor pode ser verificado na figura 4.7.

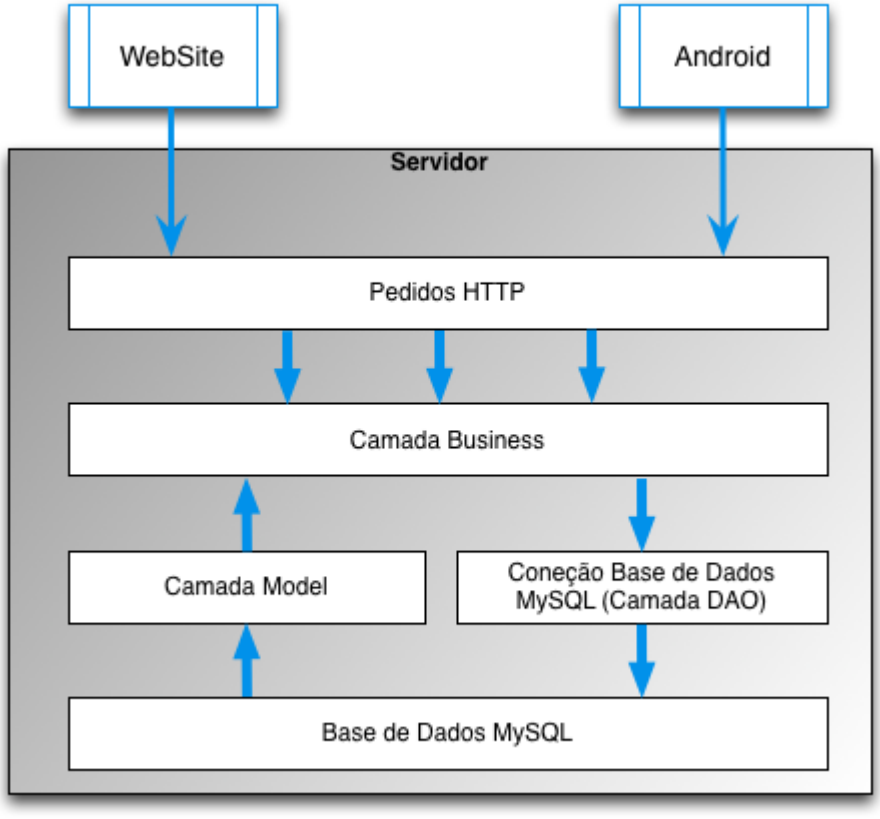


Figura 4.7: Fluxo Servidor

A camada *Services* é utilizada para definir os objetos URI para a sua utilização como *WebServices*. Quando um pedido é feito a determinado URI, o JSON recebido é descodificado em variáveis. Se este JSON não corresponder com o es-

perado é enviado um erro. Após a descodificação do JSON estas variáveis são reencaminhadas para o método especifico na camada *Business*, relacionado tanto com a aplicação *Android* ou com o *WebSite*.

Na figura 4.1 estão demonstrados os serviços para a aplicação *Android*.

Método	HTTP	URI	*Content-type*
Login na aplicação *Android*	POST	/MobileApp_LoginUser	JSON
Verificar se *checkpoint* atualizado	POST	/MobileApp_CheckChkp	JSON
Obter *checkpoint*	POST	/MobileApp_getChkp	JSON
Listagem de *checkpoints*	POST	/MobileApp_get_all_CHKP	JSON
Listagem de seguranças	POST	/MobileApp_get_all_users	JSON
Passagem por *checkpoint*	POST	/MobileApp_authCHKP	JSON
Associar *checkpoint* a *tag Mifare*	POST	/MobileApp_associateCHKP	JSON
Monitorização em tempo real	POST	/MobileApp_UserStatus	JSON
Sinal de Alarme	POST	/MobileApp_UserAlarmOn	JSON

Tabela 4.1: *Web services* para aplicação *Android*

Na figura 4.2, estão demonstrados os serviços que dão suporte as funcionalidade presentes no *Website*.

Método	HTTP	URI	*Content-type*
Criação de nova empresa	POST	/WebApp_addacc	MultipartFormData
Login no *Website*	POST	/WebApp_weblogin	JSON
Listagem de seguranças	POST	/WebApp_getusers	JSON
Adicionar novo segurança	POST	/WebApp_addUser	MultipartFormData
Editar segurança	POST	/WebApp_editUser	MultipartFormData
Eliminar segurança	POST	/WebApp_deleteUser	JSON
Listagem de *checkpoint*	POST	/WebApp_getchkps	JSON
Adicionar *checkpoint*	POST	/WebApp_addChkp	MultipartFormData
Editar *checkpoint*	POST	/WebApp_editChkp	MultipartFormData
Eliminar *checkpoint*	POST	/WebApp_deleteChkp	JSON
Listagem de *logs*	POST	/WebApp_getlogs	JSON
Listagem de *logs* com filtros	POST	/WebApp_getlogsfilter	JSON
Lista de últimos eventos	POST	/WebApp_getlive	JSON
Verificar se há novos eventos	POST	/WebApp_checklive	JSON

Tabela 4.2: *Web services* para o *Website*

Capítulo 5

Implementação do Sistema

No capítulo da implementação é demonstrado o desenvolvimento dos componentes constituintes do sistema, que foram especificados no capítulo anterior. No capítulo anterior foram definidas as funcionalidades e requisitos que o sistema deve ter, sendo a solução proposta demonstrada neste capítulo.

Este capítulo aborda a implementação da aplicação *Android*, do *Website* e do servidor.

5.1 Aplicação *Android*

A aplicação *Better Patrol* no *Android*, pode ser iniciada através do ícone de aplicação, através da passagem de um cartão de segurança ou quando o *smartphone* é inicializado.

A aplicação *Android* pode ser dividida em várias secções:

- Ecrã de Login;
- Ecrã de Entrada;
- Serviço;
- Ecrã Ronda;
- Ecrã Entrada de Administração;
- Ecrã Associar Segurança;
- Ecrã Associar *Checkpoint*.

5.1.1 Ecrã de Login

No ecrã de *Login* (figura 5.1), o segurança poderá efetuar o *login* de duas maneiras, o *login online* e o *login offline*. O *login online* pode ser efetuado com recurso a cartão *Mifare* pessoal ou à inserção dos seus dados de autenticação, o nome de utilizador (*username*) e a palavra passe (*password*). Poderá escolher também a opção de lembrar estes dados. O *login offline* só é possível com o cartão *Mifare* pessoal.

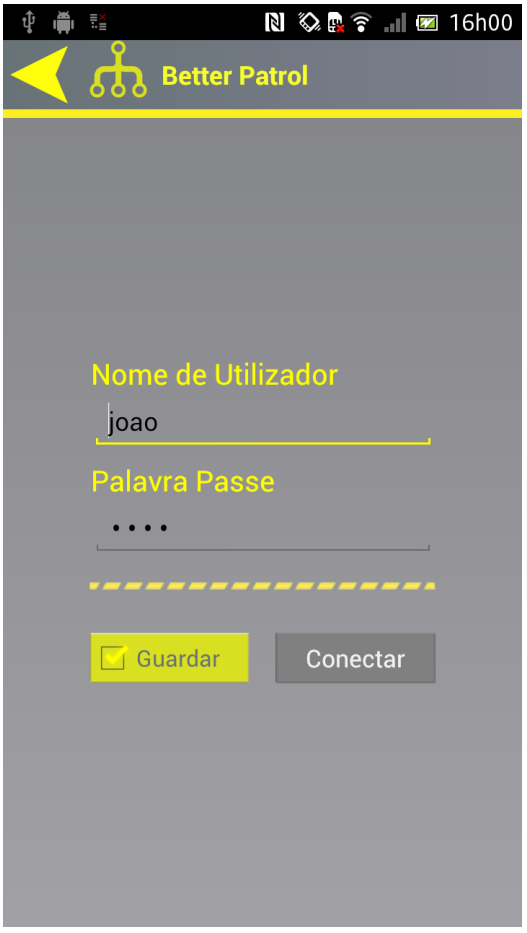


Figura 5.1: Ecrã *Login*

Na figura 5.2, pode-se visualizar a autenticação feita no cartão Mifare, onde será feita também a leitura dos dados do cartão.

```
//funçao para ler o cartao de segurança.
//dados: tipo de cartao, id empresa, username, id segurança, password em sha256
public void read_card(MifareClassic mfc)
{
    int sector_for_type=4;
    int sector_for_user_data=5;
    int sector_for_user_pass=6;
    byte authBlockData2[] = {(byte) 0xAA, (byte) 0xBB, (byte) 0xCC, (byte) 0xCC,(byte) 0xBB, (byte) 0xAA, (by

    try
    {
        mfc.connect();

        boolean auth = mfc.authenticateSectorWithKeyA(sector_for_type,authBlockData2);
```

Figura 5.2: Função leitura de cartão

A figura 5.3, mostra a função utilizada para efetuar o Login através de um HTTP POST ao serviço definido na modelação.

```
public static int Login (String username, String url)
{
    HttpClient httpclient = new DefaultHttpClient();
    HttpResponse response;
    JSONObject json = new JSONObject();
    try{

        GlobalVars.url = url;
        HttpPost post = new HttpPost(GlobalVars.url+ GlobalVars.TXT_LOGIN_URL);
        json.put("login", username);
        json.put("password", GlobalVars.user_pass);
        json.put("version", GlobalVars.APP_VERSION);
        json.put("devicetoken", GlobalVars.device_token);
        post.setHeader("Content-type", "application/json");
        StringEntity se = new StringEntity(json.toString());

        se.setContentEncoding(new BasicHeader(HTTP.CONTENT_TYPE, "application/json"));
        post.setEntity(se);
        response = httpclient.execute(post);
```

Figura 5.3: Função Login

Após efetuado o *Login* a aplicação passa para o ecrã de entrada.

5.1.2 Ecrã de Entrada

No Ecrã de Entrada, figura 5.4, o segurança pode pressionar no botão para o ecrã de ronda ou pressionar no botão para iniciar a aplicação de CCTV (*Closed Circuit Television*) que pode ser alterada nas definições.

Caso o utilizador seja um supervisor, é então visível a opção para aceder ao ecrã entrada de administração. Neste ecrã é iniciado também o serviço que é executado em *background*.

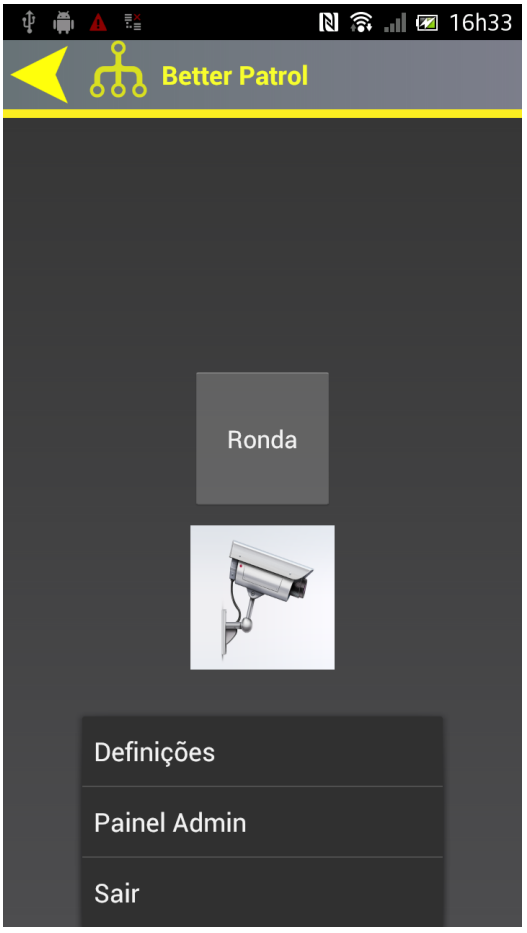


Figura 5.4: Ecrã Entrada

Na figura seguinte é possível visualizar o protótipo das funções utilizadas pelo serviço que é executado em *background*. Esta funções servem para várias funções, como enviar a chave GCM do dispositivo, atualizar os *checkpoints* do *smartphone*, enviar passagem por *checkpoint*, enviar alarme de pânico, enviar monitorização em tempo real bem como obter configurações do servidor.

```
private class update_device_token extends AsyncTask<String, Void, Void> { ... }
private class check_get_chkps extends AsyncTask<Integer, Void, Void> { ... }
private class authCHKP extends AsyncTask<String, Void, Void> { ... }
private class send_alarme extends AsyncTask<String, Void, Void> { ... }
private class send_livemonitoring extends AsyncTask<String, Void, Void> { ... }
private class GetConfigs_from_server extends AsyncTask<String, Void, Void> { ... }
```

Figura 5.5: Funções do serviço

5.1.3 Ecrã Ronda

O ecrã Ronda é o ecrã principal da aplicação, onde o segurança pode iniciar uma ronda e efetuar a passagem pelos *checkpoints*. Ao iniciar a ronda, a aplicação fica à espera de uma passagem por *checkpoint*, figura 5.6.

Figura 5.6: Ecrã Ronda

Quando é feita uma passagem por um *checkpoint*, o *smartphone* lê o conteúdo do cartão e verifica se este é um *checkpoint*. Com a chave única do cartão é possível verificar se a informação relativa ao *checkpoint* está disponível no *smartphone*.

Caso não esteja, é feito o *download* deste *checkpoint* para permitir a visualização dos dados correspondentes. É enviado ao servidor a informação de passagem do *checkpoint* mas se a ligação de dados falhar, esta passagem é guardada num vector auxiliar que irá ser enviada para o servidor pelo serviço que corre em *background*, logo que a ligação de dados retorne.

No ecrã é possível ver uma lista das passagens pelos *checkpoints*. É possível pressionar num *checkpoint* e uma secção com as notas e acesso a fotografia do

checkpoint irá deslizar até abrir totalmente, figura 5.7. Também é possível abrir um novo ecrã de *checkpoint* pressionando na seta correspondente. Este ecrã contem a fotografia e os detalhes do *checkpoint* e decidiu-se não demonstrar este ecrã por ser uma repetição de informação. No entanto este ecrã foi criado na aplicação de modo a permitir uma fácil inclusão de futuras funcionalidades, visto que a separação do *focus* da linha da lista com o botão nessa linha foi uma tarefa mais difícil devido a necessidade de ser criada uma lista customizada através do *Base Adapter*.

Ao pressionar no símbolo da câmara é iniciado uma janela tipo *pop-up* que mostra a imagem correspondente ao *checkpoint*, figura 5.8.

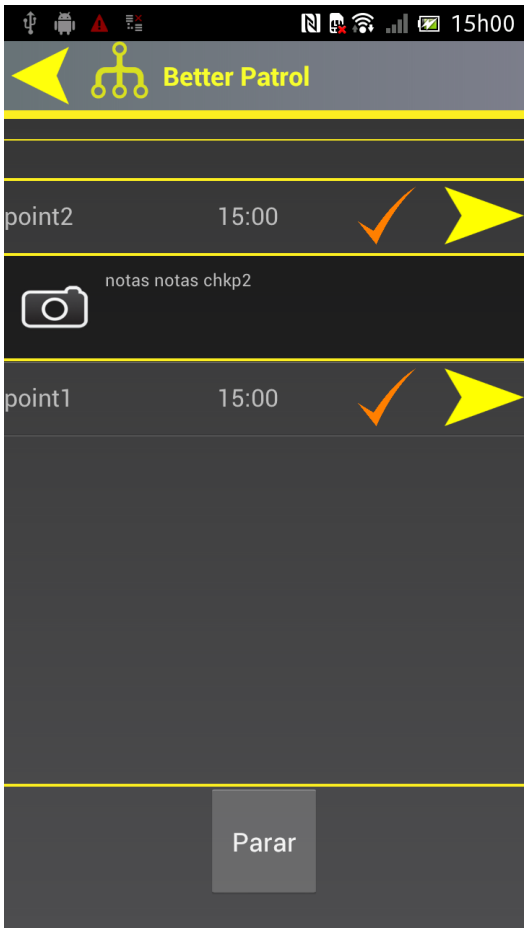


Figura 5.7: Ecrã Ronda, Detalhes *Checkpoint*

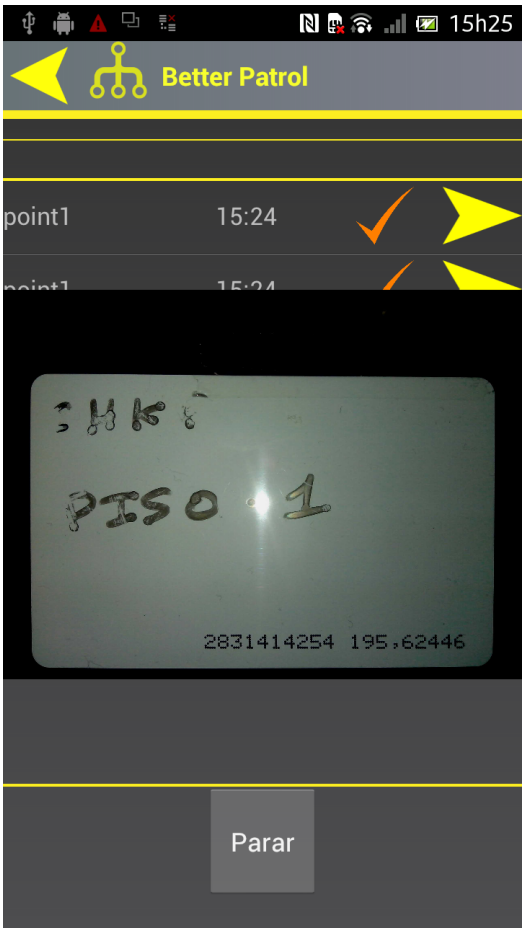


Figura 5.8: Ecrã Ronda, Imagem *Checkpoint*

5.1.4 Ecrã Entrada Administração

No Ecrã Entrada de Administração é possível escolher o próximo ecrã a abrir, associar segurança ou associar *checkpoint*, figura 5.9.

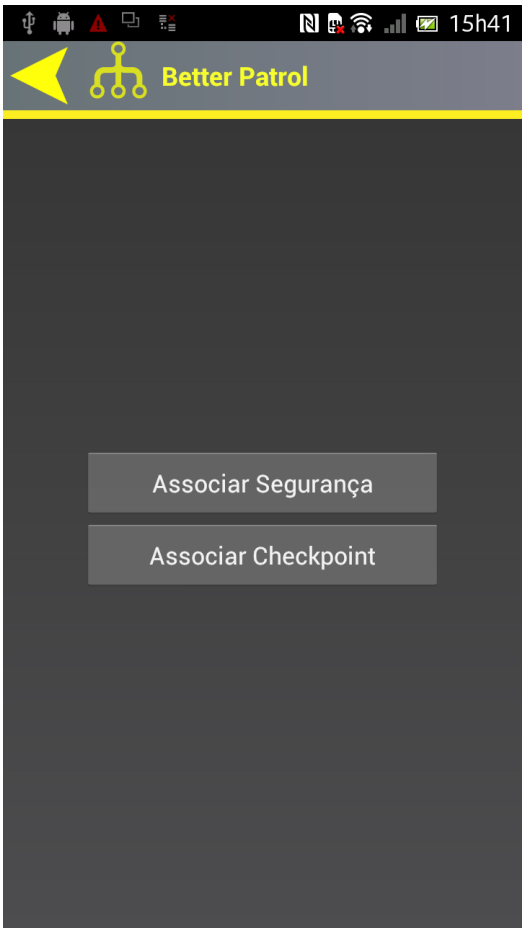


Figura 5.9: Ecrã Entrada Administração

5.1.5 Ecrã Associar Segurança

Neste ecrã é possível associar um segurança a um cartão *Mifare*. Para isso, é feito o *download* de uma lista de seguranças da empresa em questão. O supervisor pode escolher o segurança a programar, figura 5.10. A programação do cartão de segurança guarda o id de segurança e o id de empresa, o *username* e a *password*. No sector 4 e bloco 0 do cartão é guardado o tipo de cartão, neste caso um cartão de segurança e no bloco 1, o id de empresa. No sector 5, bloco 0 é guardado o id de segurança, no bloco 1 é guardado o *username* e no bloco 2 é guardado uma parte da *password* do segurança. No sector 6, blocos 0, 1 e 2 é guardado as restantes partes

da *password* do segurança. Em todos estes sectores, 4, 5 e 6, no bloco 3 é guardada a chave do cartão, que é apenas conhecida pela aplicação, o que dificulta a leitura dos dados guardados. A *password* é encriptada pelo algoritmo SHA-256 antes de ser guardada. A *hash* gerada tem um tamanho fixo de 64 carateres, portanto dividida em blocos de 16, são necessários 4 blocos para guardar a *password*. As partes desta *password* não são guardadas de uma forma sequencial, misturando a *hash* da *password*. A *password* é guardada desta maneira para dificultar a obtenção da mesma, pois apesar dos cartões *Mifare* serem protegidos por uma chave que encripta os dados contidos nos sectores, é possível decifrar o algoritmo de proteção, como vai ser demonstrado posteriormente. Mas com um ataque mais sério ao sistema é possível obter a *password* guardada. Se o atacante efetuar o reverse ao código da aplicação *Android*, encontrar o algoritmo que mistura as partes da *password* e conseguir obter a *hash* guardada no cartão, poderá reconstruir a *hash* original. Apesar de todos os cuidados na segurança com a encriptação das chaves, estas podem ser descobertas. No entanto é requirido muito trabalho pelo atacante para ter sucesso.

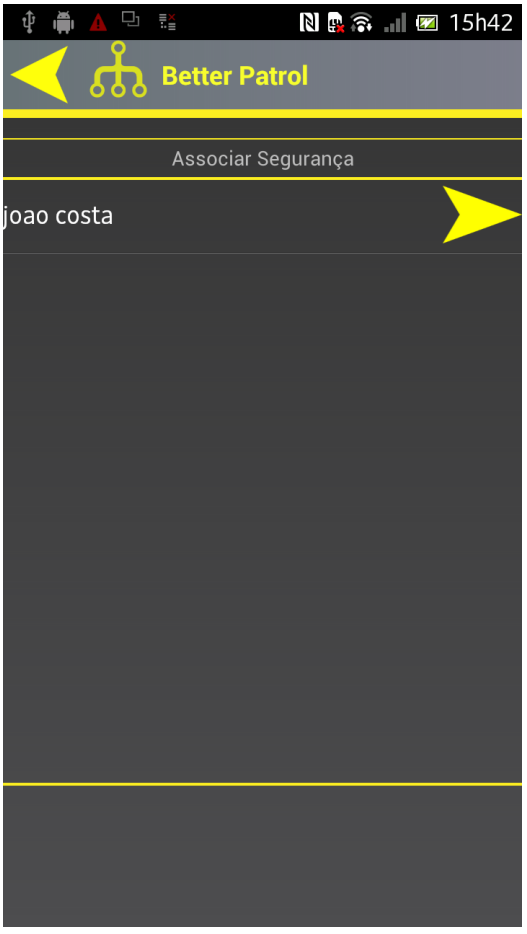


Figura 5.10: Ecrã Associar Segurança

5.1.6 Ecrã Associar *Checkpoint*

Para associar um *checkpoint* é feito o download da lista de todos os *checkpoints* da empresa, onde o supervisor pode escolher qual deseja programar, figura 5.11. Ao programar o cartão *Mifare* é gravado no sector 4, bloco 0 o tipo de cartão. É enviado para o servidor a chave correspondente ao cartão, e o *checkpoint* fica associado. Caso o supervisor associe um *checkpoint* a um cartão já associado, este *checkpoint* é associado ao novo cartão e o *checkpoint* antigo perde qualquer associação ao cartão.

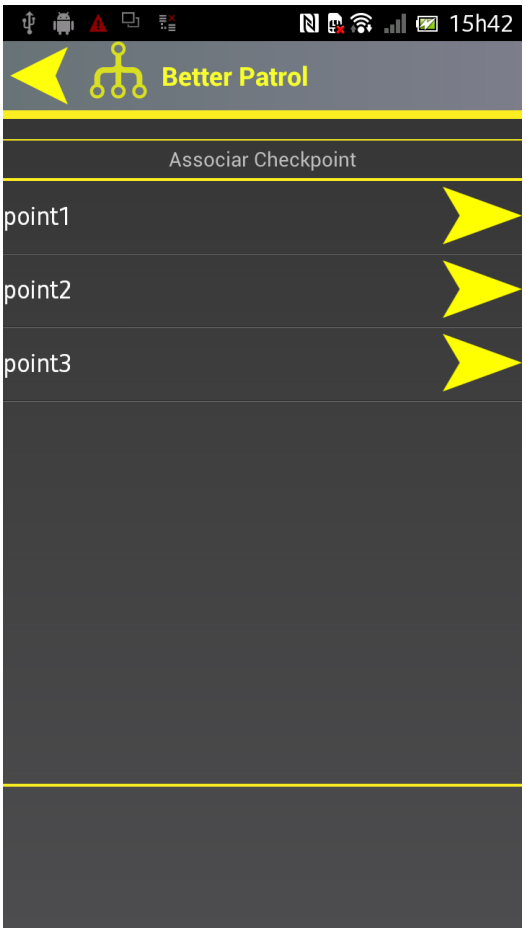


Figura 5.11: Ecrã Associar *Checkpoint*

Na figura 5.12, é possível visualizar a função que obtém a lista de *checkpoints* da empresa.

```
public static String get_all_chkp()
{
    HttpClient httpclient = GlobalVars.httpclient;
    HttpResponse response;
    JSONObject json = new JSONObject();
    try{
        HttpPost post = new HttpPost(GlobalVars.url+ GlobalVars.GET_ALL_CHKP);
        json.put("user_id", GlobalVars.user_id);

        post.setHeader("Content-type", "application/json");
        StringEntity se = new StringEntity(json.toString());
        se.setContentEncoding(new BasicHeader(HTTP.CONTENT_TYPE, "application/json"));
        post.setEntity(se);

        response = httpclient.execute(post);
```

Figura 5.12: Função para obter lista de *Checkpoints*

5.1.7 Resumo do fluxo da aplicação *Android*

A figura 5.13, demonstra o fluxo da aplicação *Android*.

É possível verificar os ecrãs da aplicação, que começa no ecrã de Login e pode interagir com o cartão de segurança. Em seguida segue para o ecrã de Entrada, onde caso seja um segurança pode iniciar uma nova ronda através do ecrã de Ronda, ou caso seja um supervisor pode aceder a parte de programação tanto de cartões de segurança como a programação de *checkpoints*.

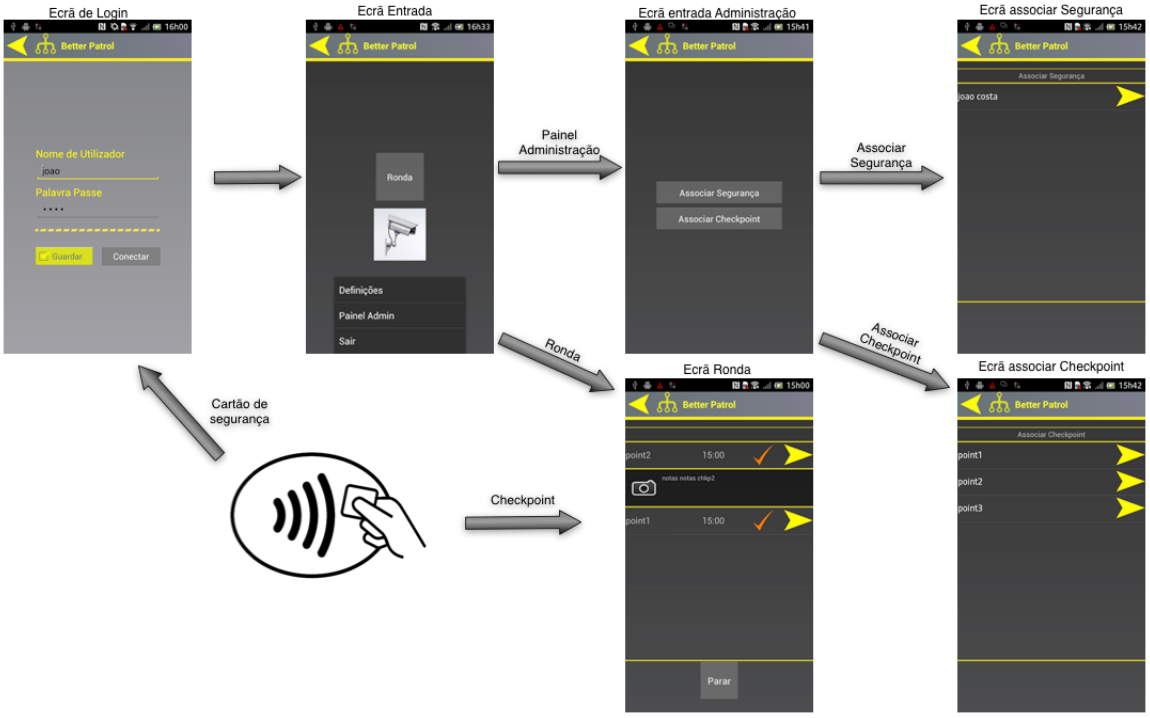


Figura 5.13: Fluxo Aplicação *Android*

5.2 *WebSite*

Para desenvolver o *WebSite* foi utilizada a *framework* ExtJS. O *WebSite* foi desenvolvido para permitir a gestão do sistema por parte do supervisor.

O *WebSite* é constituído por várias secções:

- *Login*;
- Seguranças;
- *Checkpoint*;
- Monitorização em tempo real;
- *Logs*.

5.2.1 Login

Este é o ecrã inicial do site, onde é possível efetuar o *Login* de supervisor, figura 5.14.

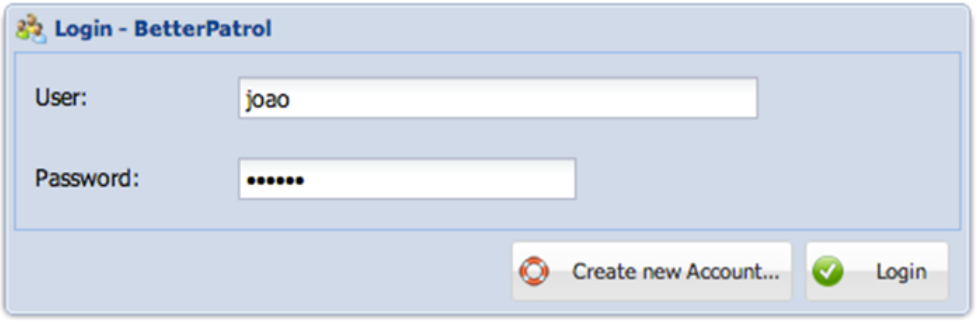


Figura 5.14: Janela de Login *WebSite*

Ao efetuar *login* o supervisor insere o seu *username* e *password*. Um pedido Ajax é efetuado ao endereço '.../services/WebApp_weblogin', com um JSON constituído com o *username* e a *password* encriptada com sha-256.

Nesta secção é possível também abrir a janela para a criação de uma nova empresa.

Na janela de criação de uma nova empresa é necessário inserir o nome da nova empresa, mas também a inserção de dados do supervisor, para ser possível efetuar o *login* para gerir a empresa, figura 5.15.

É feita uma verificação dos dados inseridos, de forma a não existirem campos em branco, o *username* e a *password* devem estar entre 6 a 16 carateres, o campo de *password* deve combinar com o campo de confirmação de *password* e

para o email ser válido, deverá ter no mínimo um caractere '.' e um '@'. Estes dados são enviados para o servidor através de um *formsubmit* para o endereço '.../services/WebApp_addacc'.

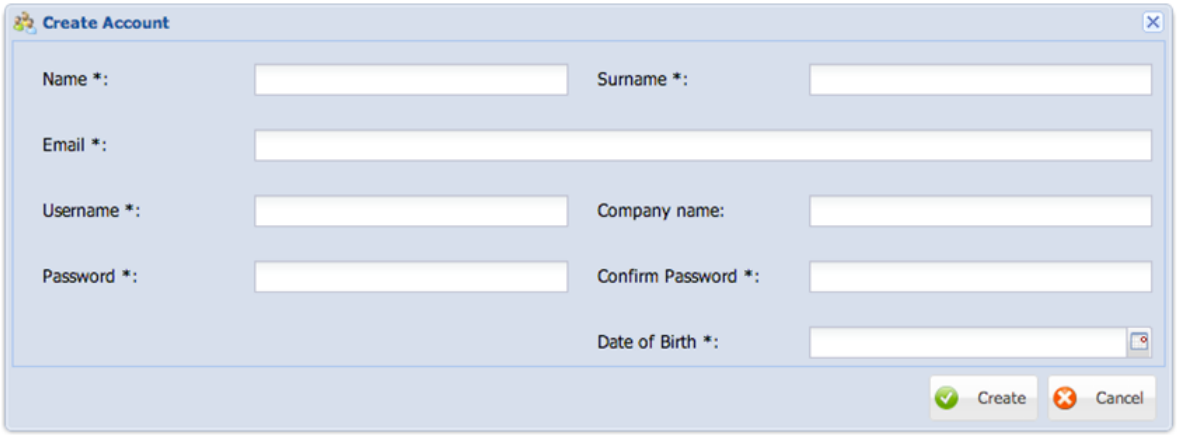


Figura 5.15: Janela Criação de Conta

5.2.2 Seguranças

Nesta secção é possível visualizar todos os seguranças pertencentes à empresa, figura 5.16.

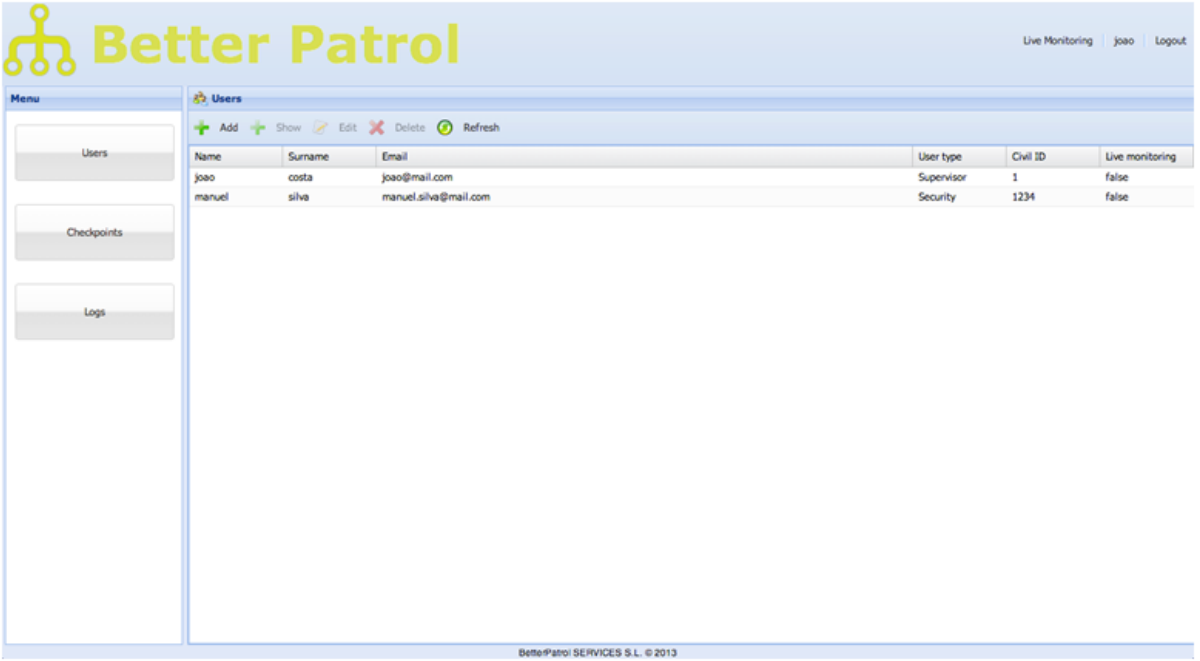


Figura 5.16: Lista de Seguranças

É feito um pedido Ajax ao endereço '.../services/WebApp_getusers', que

retorna um JSON com os dados de todos os campos de todos os seguranças pertencentes à empresa em questão.

Nesta secção é possível adicionar, editar, eliminar e visualizar mais detalhes de seguranças. Na janela de adicionar segurança, figura 5.17, é possível inserir os dados correspondentes aos campos existentes na base de dados. O supervisor pode ainda adicionar uma imagem correspondente ao segurança para fazer *upload* para o servidor. É feita uma verificação se todos os campos obrigatórios estão preenchidos e se o *email*, *username* e numero civil já existem na base de dados. Caso não existam é adicionado o novo segurança.

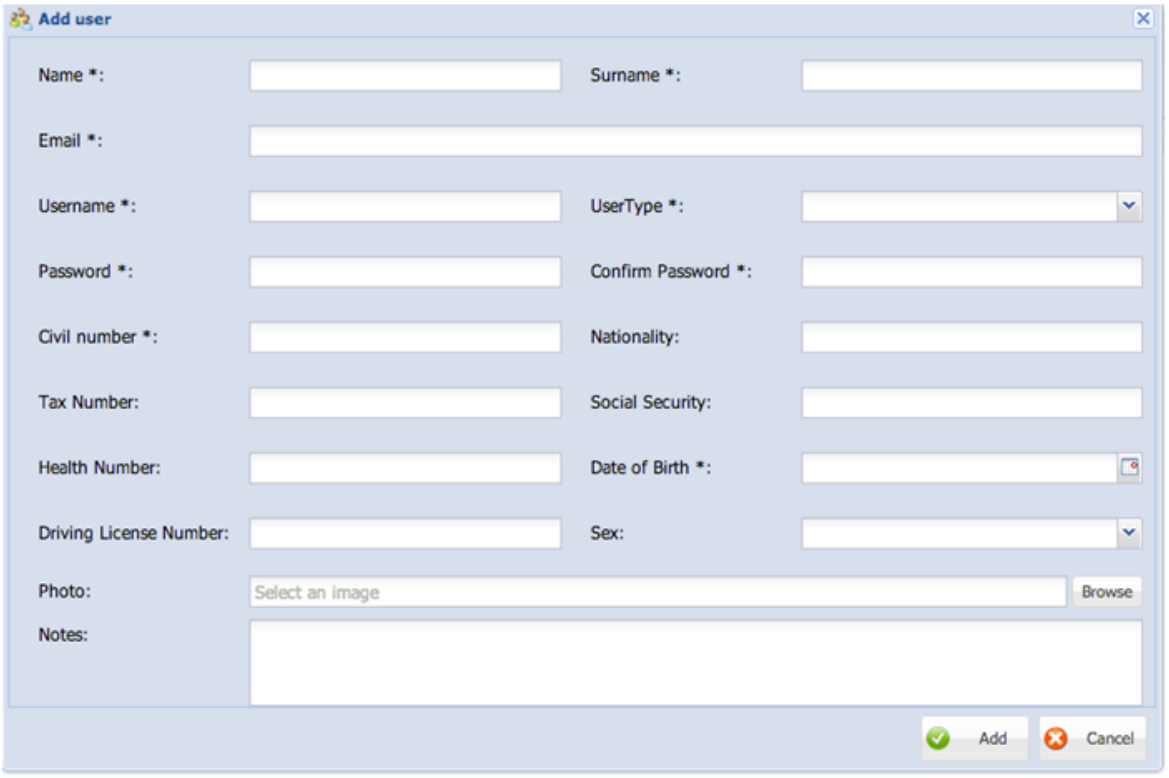


Figura 5.17: Adicionar Segurança

A janela de edição de segurança contem todos os campos existentes na janela de adicionar e também a possibilidade de mudar o estado da monitorização em tempo real do segurança, figura 5.18. Caso a opção do estado de monitorização seja alterado, é enviada uma mensagem *push* através do GCM, para o dispositivo do segurança que sinaliza esta atualização.

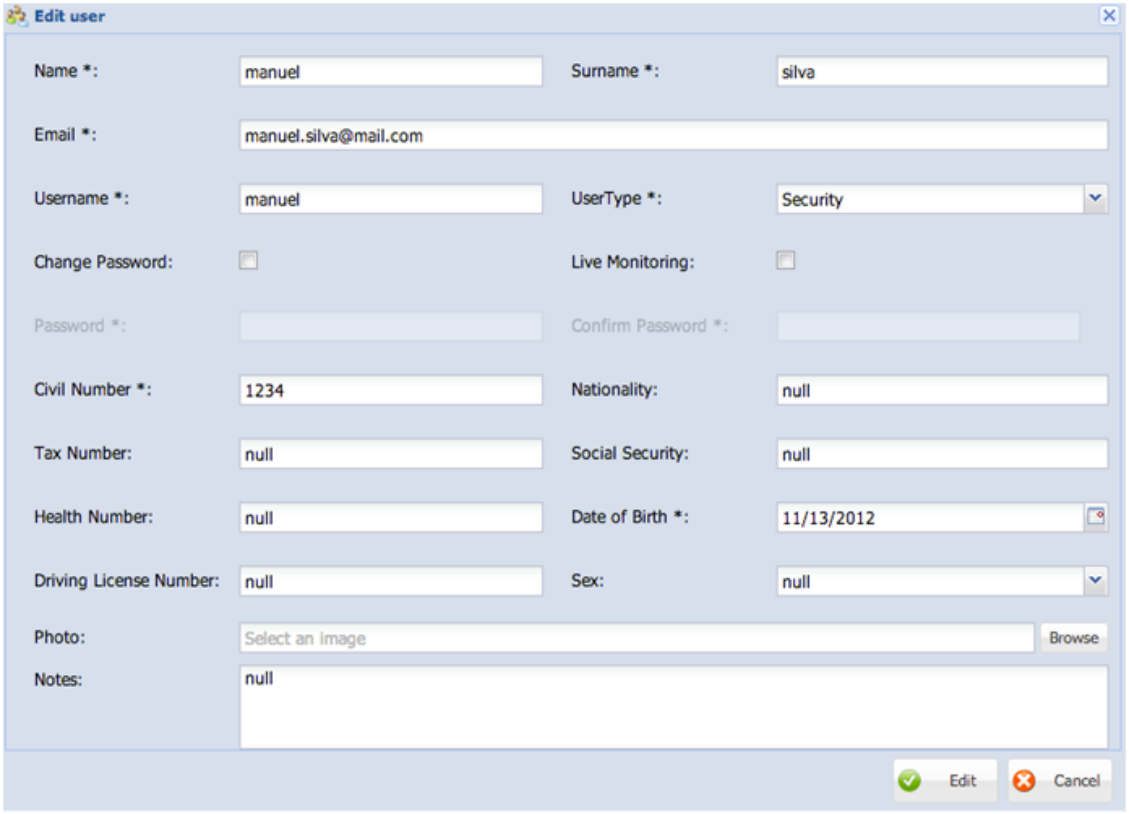


Figura 5.18: Editar Segurança

Na janela para visualizar detalhes do segurança é possível ver todos os dados relacionados com o segurança e a imagem do mesmo, figura 5.19.

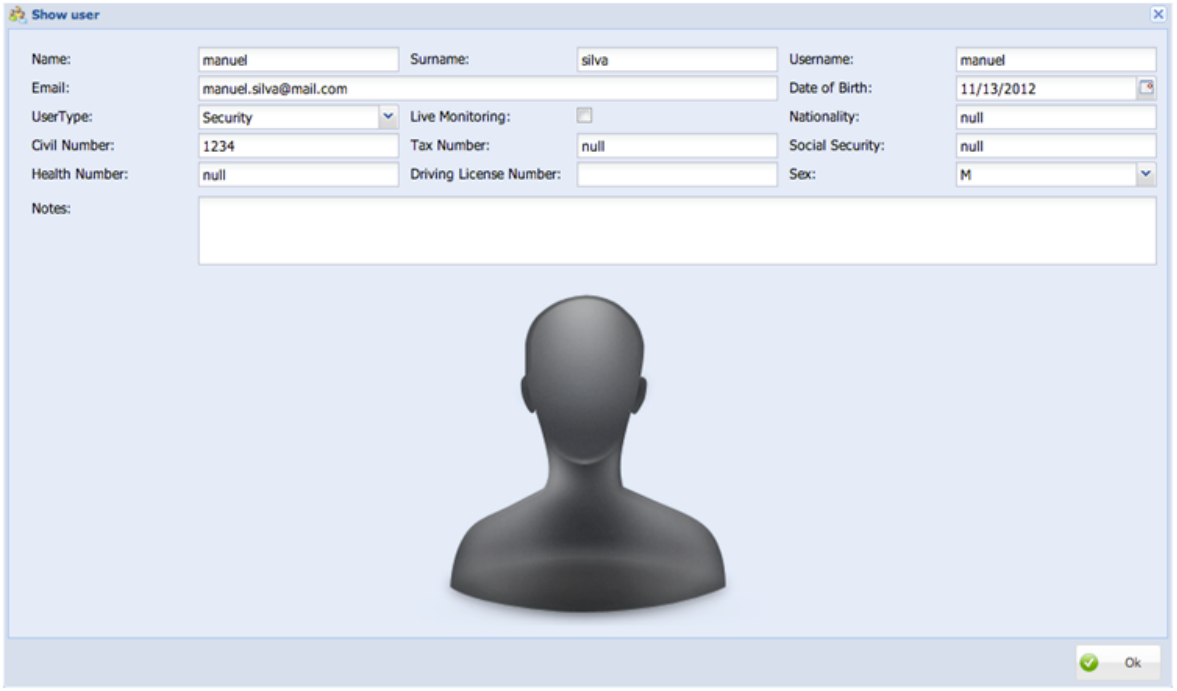


Figura 5.19: Visualizar Segurança

5.2.3 *Checkpoints*

Na secção dos *checkpoints* é possível obter uma lista completa para observar todos os que pertencem à empresa através do endereço '.../services/WebApp_getchkps', figura 5.20.

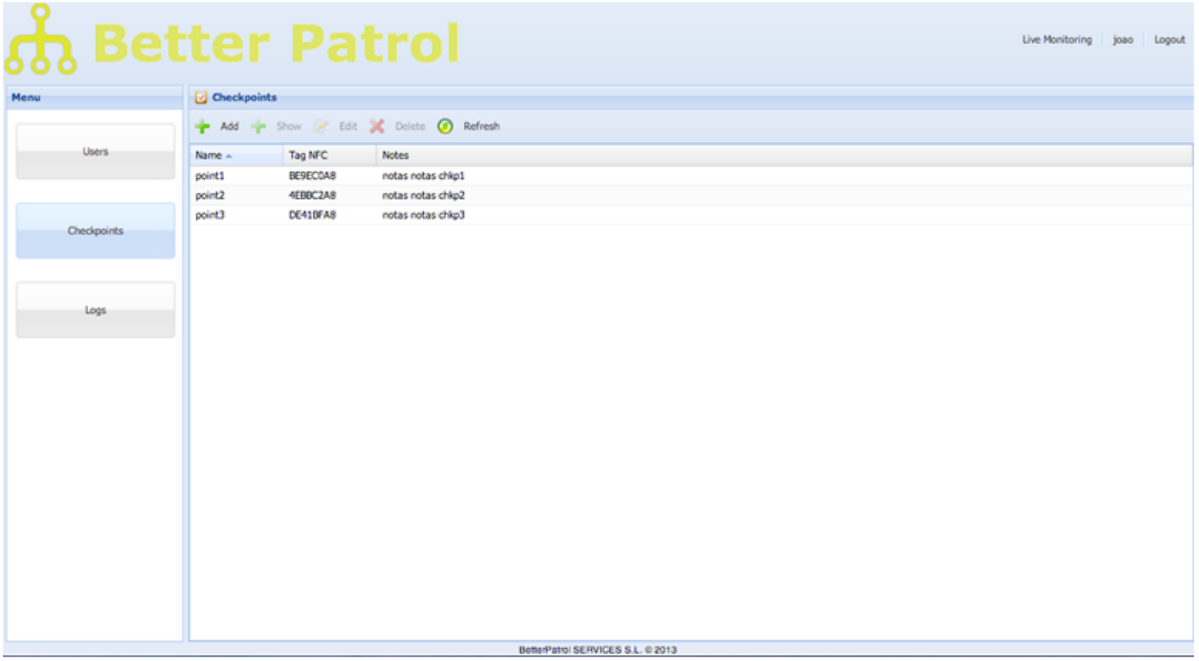


Figura 5.20: Lista de *Checkpoints*

Nesta secção, o supervisor pode adicionar *checkpoints*, editar, eliminar e mostrar detalhes.

Na criação pode inserir o nome, as coordenadas GPS, uma imagem e as notas do *checkpoint*, figura 5.21.

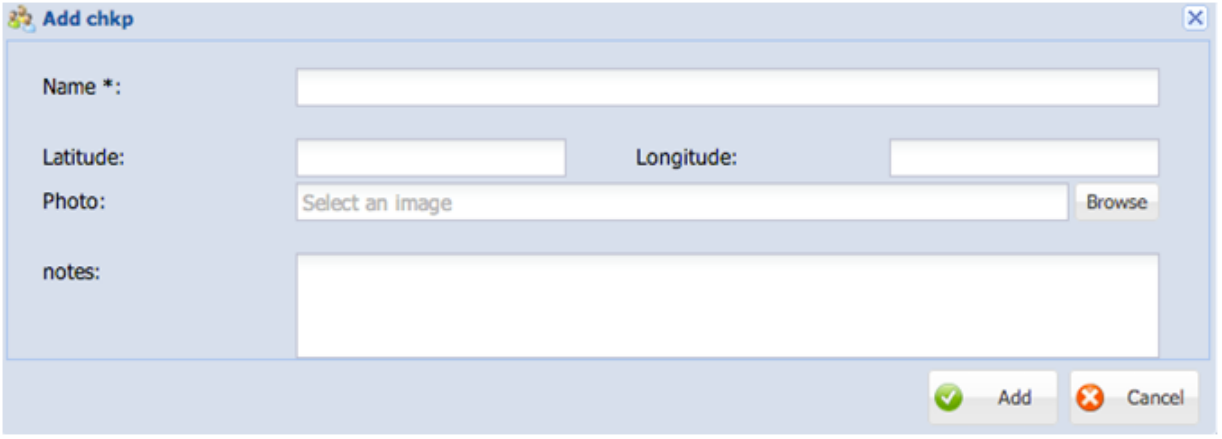


Figura 5.21: Inserir *Checkpoint*

Na janela de editar *checkpoint* o supervisor pode efetuar alterações ao mesmo,figura 5.22.

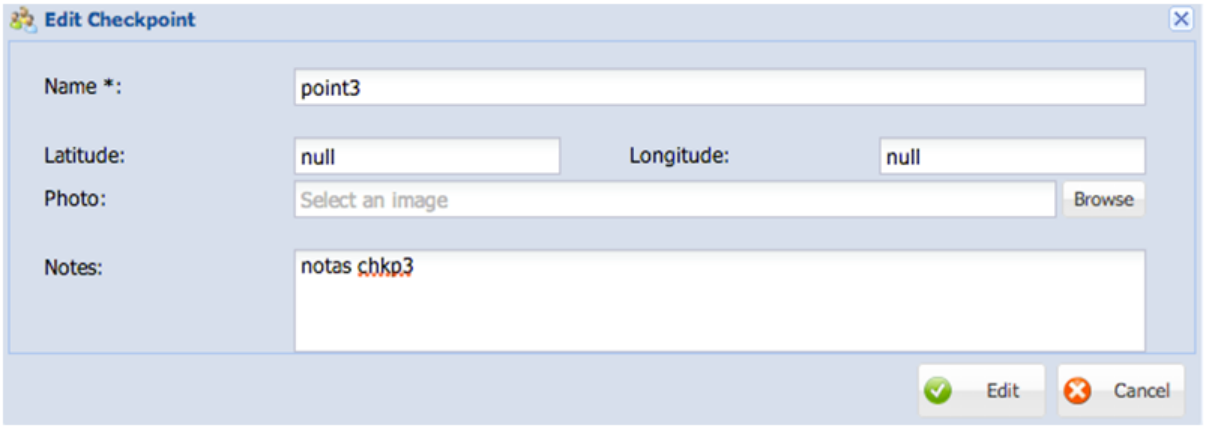


Figura 5.22: Editar *Checkpoint*

É também possível visualizar a informação e imagem do *checkpoint*, figura 5.23.

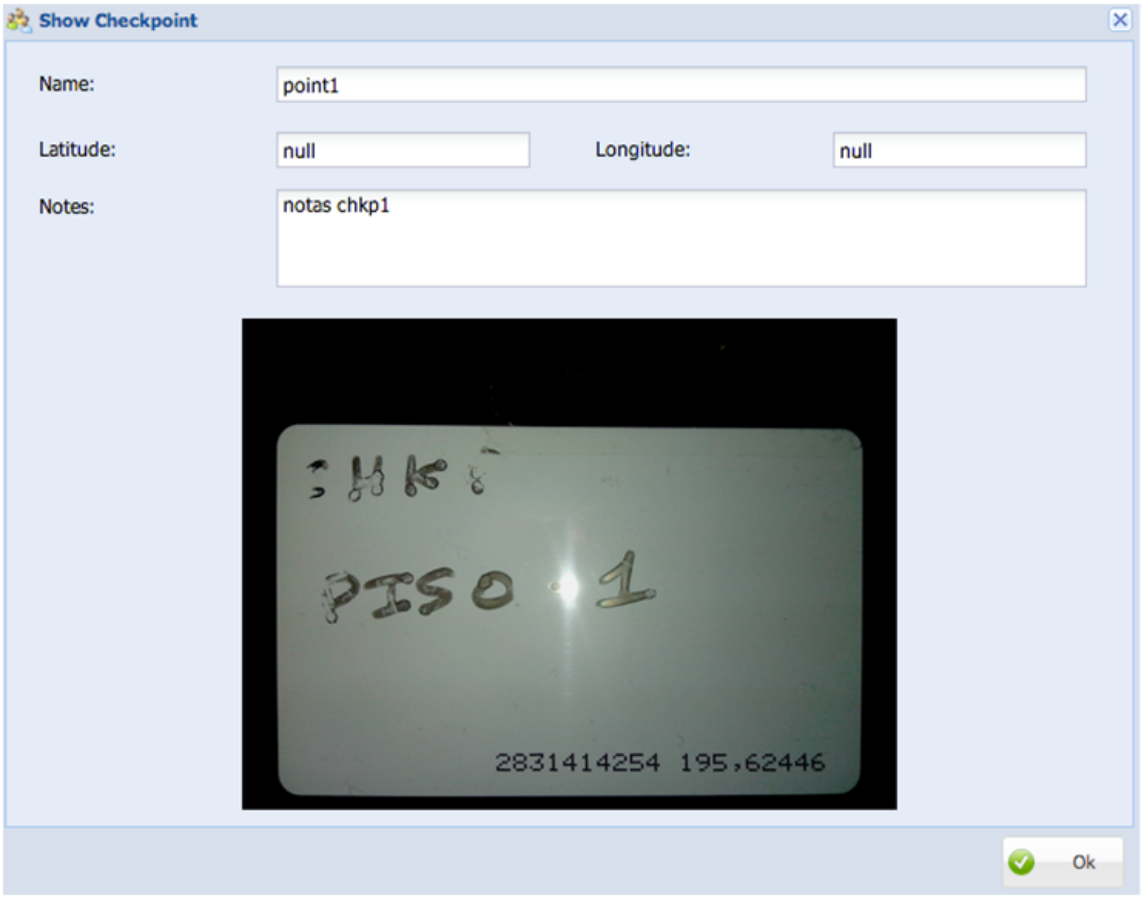


Figura 5.23: Visualizar *Checkpoint*

5.2.4 Monitorização tempo real

Na monitorização em tempo real é possível visualizar os eventos que existem no sistema, através do endereço '.../services/WebApp_getlive'. Estes eventos acontecem em três situações: quando há uma passagem por um *checkpoint*, quando há um sinal de alarme e quando a monitorização de seguranças está ativa. A lista vai sendo preenchida conforme chegam novos eventos, figura 5.24.

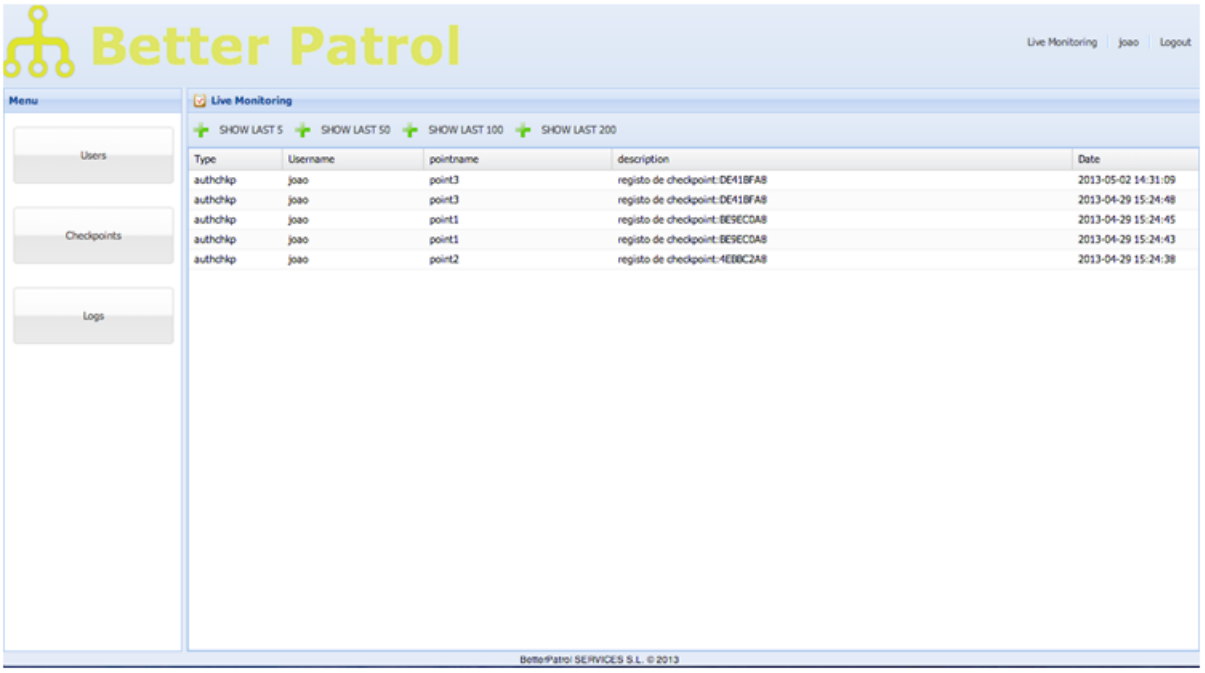


Figura 5.24: Monitorização em Tempo Real

Inicialmente é feito um pedido para receber os últimos 5 registos, o supervisor pode mostrar os últimos cinquenta, cem ou duzentos registos. Os novos eventos recebidos pelo sistema vão sendo preenchidos no topo da lista. Esta monitorização é feita com um intervalo de tempo de cinco segundos. Este atraso foi inserido no *Website*, pois sobrecarregaria muito o servidor ao pedir uma atualização todos os segundos. No entanto não existe nenhum *delay* entre o servidor e o *smartphone* do segurança, onde a informação é transmitida no instante em que ocorre.

5.2.5 *Logs*

Na secção de *Logs* é possível verificar os registos do sistema, onde é possível pesquisar com filtros por seguranças e por tipo de *log*. Caso a posição de GPS do segurança seja conhecida é criado um link direto para o *Google Maps* para visualizar a posição do mesmo, figura 5.25.

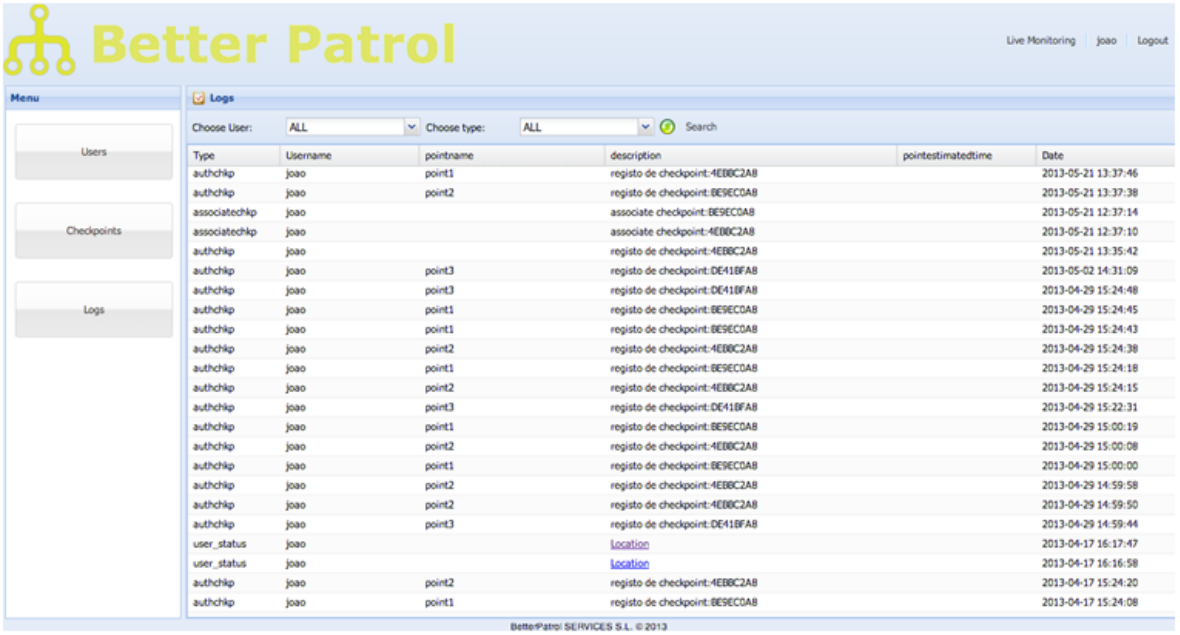


Figura 5.25: Lista de *Logs*

5.3 Servidor

Como já foi referido anteriormente o servidor utilizado foi o *Glassfish* e a aplicação foi desenvolvida em Java utilizando o IDE *NetBeans*.

A aplicação foi estruturada em quatro sectores: camada *Model*, camada DAO (*Data Access Object*), camada *Business* e camada *Services* ou também conhecida como camada Apresentação, figura 5.26.

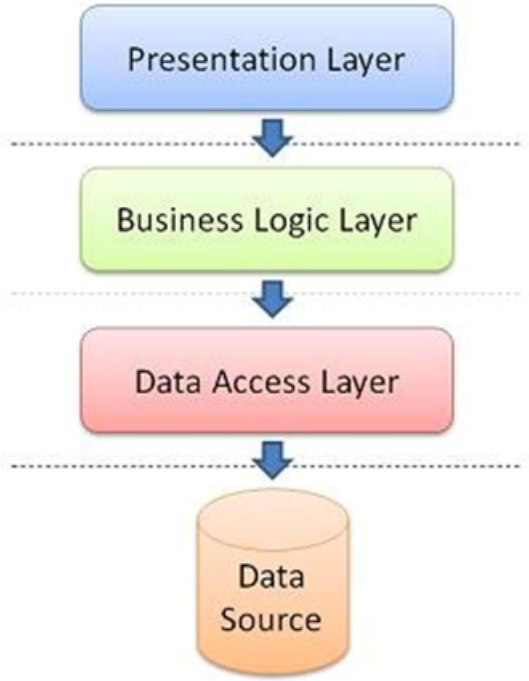


Figura 5.26: *Presentation, Business, DAO Layer*

O *NetBeans* facilita a criação das camadas *Model* e DAO, pois gera o código através de uma base de dados já existente.

Na camada *Model* foram geradas as classes correspondentes às tabelas existentes na base de dados, com todos os SET's e GET's necessários.

A camada DAO é constituída com os métodos que permitem manipular a base de dados, criar, editar, remover e encontrar registos. Como este código é gerado automaticamente acelera o desenvolvimento do servidor. Na camada *Business* foram desenvolvidos os métodos necessários para o sistema. Os métodos desenvolvidos foram separados de forma a responder em especifico tanto à aplicação *Android* como ao *WebSite*, tornando assim as respostas o mais eficiente possível. Nesta camada são também utilizadas as sessões quando o *login* é efetu-

ado. Em todos os métodos estas sessões são verificadas, para autenticar os pedidos do utilizador e caso não se verifique é necessário efetuar o *login* novamente. Todas as respostas são geradas no formato de text JSON, para facilitar a transferência de dados.

Na figura 5.27, é possível visualizar alguns dos *web services* relativos aos pedidos do *Android* presentes no servidor.

```
// serviço de login para o android
@POST
@Path("MobileApp_LoginUser")
public String MA_LoginUser(String data){ ... }

//serviço para fazer download do checkpoint
@POST
@Path("MobileApp_getChkp/")
@Produces(value="application/json")
public String MA_getCHKP(String data) { ... }

//serviço para atualizar token GCM do dispositivo
@POST
@Path("MobileApp_set_device_token/")
@Produces(value="application/json")
public String MA_set_device_token(String data){ ... }

//serviço para gravar passagem por checkpoint
@POST
@Path("MobileApp_authCHKP/")
@Produces(value="application/json")
public String MA_authCHKP(String data) { ... }

//serviço para associar checkpoint a um cartao
@POST
@Path("MobileApp_associateCHKP/")
@Produces(value="application/json")
public String MA_associateCHKP(String data) { ... }

//serviço que envia informaçao sobre a monitorizaçao em tempo real
@POST
@Path("MobileApp_UserStatus/")
public String MA_userStatus(String data) { ... }

//serviço que recebe o alarme do dispositivo
@POST
@Path("MobileApp_UserAlarmOn/")
public String MA_AlarmOn(String data){ ... }
```

Figura 5.27: *Web services* para *Android*

5.4 Amazon Web Services

> "A Amazon Web Services (AWS) oferece um conjunto completo de serviços de aplicativos e infraestrutura que permitem que você execute praticamente tudo na nuvem: desde aplicativos empresariais e projetos de dados grandes a jogos sociais e aplicativos móveis."[25]

Foi escolhida a *Amazon* para alojar o servidor pois permite o uso de Linux onde é possível instalar e utilizar o software pretendido, além disso, o alojamento é gratuito (primeiro ano de funcionamento). O produto que fornece o nível gratuito é o *Amazon* EC2, *Amazon Elastic Compute Cloud*, tendo no entanto vários níveis, com capacidade de computação redimensionáveis. É possível escolher entre vários sistemas operativos aquando da criação da conta gratuita. Neste caso foi escolhida a distribuição *Amazon Linux AMI* com 8Gb de disco disponível. Depois de instalada a distribuição, é necessário abrir a porta vinte e dois, com o intuito de se poder ter acesso remoto via SSH (*Secure Shell*). Deste modo, foi necessário instalar o servidor *MySQL* e o servidor *Glassfish*, sendo assim possível alojar o servidor na *Cloud*.

Figura 5.28: Amazon

5.5 Vulnerabilidade *Mifare*

Como foi referido anteriormente a tecnologia *Mifare* tem algumas falhas no protocolo, onde é possível obter as chaves que codificam os blocos de memória [23].

Esta falha foi testada neste sistema, com uma distribuição de Linux, instalaram-se os pacotes necessários, *nfclib* e *nfcutils* e com o hardware ACR122U NFC Reader da ACS, figura 5.29.

Figura 5.29: ACR122U

Após aproximadamente cinco minutos o programa *mfoc*, figura 5.30, conseguiu com sucesso obter as chaves, figura 5.31, e criar um ficheiro hexadecimal com todos os dados existentes no cartão.

```
cesar@linux:~/nfc$ sudo mfoc -O testuser.mfd
sudo: unable to resolve host linux
ISO/IEC 14443A (106 kbps) target:
    ATQA (SENS_RES): 00  04
* UID size: single
* bit frame anticollision supported
       UID (NFCID1): 4e  bb  c2  a8
      SAK (SEL_RES): 08
* Not compliant with ISO/IEC 14443-4
* Not compliant with ISO/IEC 18092

Fingerprinting based on MIFARE type Identification Procedure:
* MIFARE Classic 1K
* MIFARE Plus (4 Byte UID or 4 Byte RID) 2K, Security level 1
* SmartMX with MIFARE 1K emulation
```

Figura 5.30: mfoc

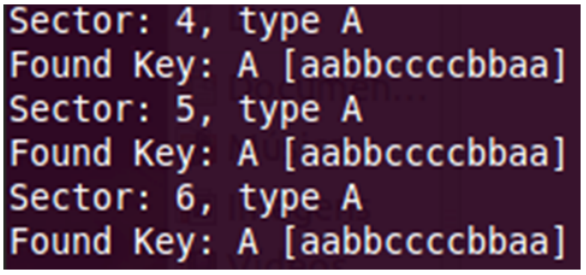


Figura 5.31: Chaves Mifare

Com outro programa de análise de ficheiros hexadecimais, *hexer*, foi possível obter a informação legível que estava presente no cartão, figura 5.32.

```
00000140 3100 0000 0000 0000 0000 0000 0000 0000 6A6F 616F  1...............joao
00000154 0000 0000 0000 0000 0000 0000 3945 3133 4639 3738  ............9E13F978
00000168 4437 4338 3436 4634 AABB CCCC BBAA FF0F 0000 AABB  D7C846F4............
0000017C CCCC BBAA 3033 4143 3637 3432 3136 4633 4531 3543  ....03AC674216F3E15C
00000190 3736 3145 4531 4135 4532 3535 4630 3637 3935 3336  761EE1A5E255F0679536
000001A4 3233 4338 4233 3838 4234 3435 AABB CCCC BBAA FF0F  23C8B388B445........
```

Figura 5.32: Conteúdo Cartão

A figura 5.32 demonstra a informação a partir do 'user_id', que, como foi dito anteriormente, é gravado no sector cinco. Efetuando os cálculos é possível verificar que se encontra na posição correta, pois quatro blocos multiplicando por dezasseis bytes multiplicando por cinco sectores obtém-se o valor trezentos e vinte que em hexadecimal equivale ao valor 0x140. A informação fica assim disponível para o atacante.

Capítulo 6

Testes, conclusões e trabalho futuro

6.1 Testes

Foram feitos testes à aplicação com o *smartphone* utilizado, *Sony Xperia S*, para verificar a duração da bateria.

É importante ter um conhecimento deste valores, pois, se esta utilização consumir muita bateria, esta aplicação deixa de ser viável.

Foram feitos quatro testes distintos, em todos estes testes foram feitas passagens por *checkpoints* de 5 em 5 minutos, a saber:

1º- *Wireless* ligado, 3G desligado, GPS desligado, '*live monitoring*' desligado;

2º- *Wireless* desligado, 3G ligado, GPS ligado, '*live monitoring*' desligado;

3º- *Wireless* desligado, 3G ligado, GPS ligado, '*live monitoring*' ligado, sem sinal GPS;

4º- *Wireless* desligado, 3G ligado, GPS ligado, '*live monitoring*' ligado, com sinal GPS.

A opção '*live monitoring*' da aplicação permite, como foi dito anteriormente, enviar para o servidor os valores de posição GPS do segurança. Através da seguinte tabela 6.1 é possível verificar os resultados aproximados dos testes efetuados:

	1º teste	2º teste	3º teste	4º teste
Wifi	✓	✗	✗	✗
GPS	✗	✓	✓	✓
3G	✗	✓	✓	✓
Sinal GPS	-	-	✗	✓
Live monitoring	✗	✗	✓	✓
Checkpoint de 5 em 5 minutos	✓	✓	✓	✓
Resultados	46h	20h	8h	7h

Tabela 6.1: Resultados dos testes efetuados

Através destes resultados pode verificar-se que a utilização do *wireless* em detrito do 3G, aumenta a autonomia do *smartphone*.

Foram feitos mais testes concentrados no 3G e no GPS, pois estas rondas têm como objetivo o ambiente exterior, sendo portanto o 3G a conexão mais fiável. Verifica-se também que a utilização do *'live monitoring'* ativa o GPS no *smartphone* o que reduz para menos de metade a autonomia do mesmo.

O 4º teste representa uma utilização do sistema com todas as funcionalidades ativas e obtém uma autonomia de 7 horas que se pode considerar uma boa autonomia visto o hardware a ser utilizado.

No caso de ser necessário uma autonomia maior, o segurança poderá levar consigo uma bateria auxiliar, ou terá que utilizar outro *smartphone* com melhores resultados.

6.2 Conclusão

No âmbito desta dissertação, foi desenvolvido um sistema de suporte a rondas de segurança, o *Better Patrol*, que é constituído por um aplicação *Android*, um servidor e um *Website*. Com este sistema é possível efetuar rondas de segurança em tempo real e gerir estes eventos no *Website*. Os seguranças podem efetuar as rondas através de passagens por *checkpoints* com *tags Mifare* e estas informações serão enviadas para o servidor e guardadas na base de dados. O supervisor pode aceder ao *Website* criar/editar/eliminar seguranças e *checkpoints*, verificar os *logs* e monitorizar em tempo real.

Ao desenvolver o trabalho descrito nesta dissertação foram obtidas competências na área dos *Web Services*, a aprendizagem da linguagem de programação Java, iniciação no desenvolvimento de páginas *web* e aprendizagem no ambiente de programação *Android*. É possível concluir que os objetivos iniciais foram alcançados com sucesso.

Esta dissertação é um protótipo faltando, porém, adicionar ainda algumas funcionalidades.

6.3 Trabalho futuro

Como trabalho futuro pretende-se incluir suporte à criação de ocorrências por parte dos seguranças de forma que o supervisor verifique estas no *Website*. Estas ocorrências poderiam ter anexados ficheiros de vídeo, áudio ou imagem e separadas por categorias.

Outra funcionalidade é a criação de rondas agendadas, como já foi referido anteriormente, a base de dados já está preparada para esta funcionalidade. O supervisor poderia criar um ronda com uma ordem e horários específicos, atribuindo estas a determinados seguranças.

Existe a possibilidade de adicionar o suporte ao *Google Maps*, assim o segurança poderia ver a localização dos *checkpoints* e a rota a percorrer no caso de rondas agendadas, no *smartphone*.

Permitir o uso de *tags* que utilizam protocolos de comunicação diferentes do protocolo *Mifare*.

Melhorar os *web services* de modo a que estes estejam em conformidade com o protocolo REST.

Melhorar a listagem de informação no *Website*.

Por fim, adicionar uma funcionalidade na monitorização em tempo real através do recurso à tecnologia ANT+, que permite a conexão a um dispositivo que deteta os batimentos cardíacos do segurança.

Bibliografia

[1] Gartner: http://www.gartner.com/it/page.jsp?id=2237315

[2] Gartner: http://www.gartner.com/it/page.jsp?id=2227215

[3] Meier, Reto., Professional Android Application Development. Wiley 2008, ISBN: 978-0-470-34471-2.

[4] Gartner: http://www.gartner.com/it/page.jsp?id=2194115

[5] Jian Li, Ruofeng Tong, MinTang, Jinxiang Dong, "Webservice-Based Distributed Feature Library", Computer Supported Cooperative Work in Design, 2004. Proceedings. The 8th International Conference on, On page(s): 753-758 Vol.1

[6] Chuangwei Zhang, Xu Yin, "Design and implementation of single-service multifunction Webservice", Computer Science and Service System (CSSS), 2011 International Conference on, On page(s): 3912-3915

[7] XinyangFeng, JianjingShen, Ying Fan, "REST?An Alternative to RPC for Web Services Architecture", Future Information Networks, 2009. ICFIN 2009. First International Conference on, On page(s): 7-10

[8] Chakrabarti, S.K., Kumar, P., "Test-the-REST: An Approach to Testing RESTfulWeb-Services", Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, 2009. COMPUTATIONWORLD '09. Computation World, On page(s): 302-308

[9] JiehuiJu, Ya Wang, Jianqing Fu, Jiyi Wu, Zhijie Lin, "Research on Key Technology in SaaS", Intelligent Computing and Cognitive Informatics (ICICCI), 2010 International Conference on, On page(s): 384-387

[10] Yang Wang,Bin Zhang, Ying Liu, Deshuai Wang, “The Modeling Tool of SaaS Software”, Advanced Computer Control (ICACC), 2010 2nd International Conference on, page(s): 298-302

[11] Tagtronics: http://www.tagtronics.co.uk/tagtronics-patrol-live.aspx

[12] Bcsint: http://www.bcsint.com/proximity_wand.php

[13] NFCpatrol: http://www.nfcpatrol.com/

[14] serosolutions: http://www.serosolutions.co.uk/default.aspx

[15] Freitas, Filipe Manuel Lopes., “SegUA: um sistema movel para apoio a rotina de vigilancia”, Universidade de Aveiro, 2009

[16] Luka Finzgar, Mira Trebar, “Use of NFC and QR code Identification in an Electronic Ticket System for Public Transport”, Software, Telecommunications and Computer Networks (SoftCOM), 2011 19th International Conference on, On page(s): 1-6

[17] Gartner: http://www.gartner.com/it/page.jsp?id=2028315

[18] Shih-Hau Fang, Yi-Chung Liang, Kuan-Ming Chiu, “Developing a mobile phone-based fall detection system on Android platform”, Computing, Communications and Applications Conference (ComComAp), 2012, On page(s): 143-146

[19] Android: http://developer.android.com/google/gcm/index.html

[20] JSON: http://www.json.org/

[21] Anddev: http://www.anddev.org/novice-tutorials-f8/lifecycle-of-an-activity-t81.html

[22] Netcard: http://www.netcard.com.au/products/what-is-mifare

[23] Gerhard de KoningGans, Jaap-HenkHoepman,Flavio D. Garcia, “A Practi-

cal Attack on the MIFARE Classic", Institute for Computing and Information Sciences, Radboud University Nijmegen

[24] Gsmarena: http://www.gsmarena.com/sony_xperia_s-4369.php

[24] Amazon AWS: http://aws.amazon.com/pt/

[26] MySQL: http://www.mysql.com/products/enterprise/techspec.html

Todos os *WebSites* operacionais na data: 01/10/2013