Universidade do Minho
Escola de Engenharia

Ricardo Miguel Rego Mesquita

Where@UM – aplicação de posicionamento colaborativo para PC baseada em Wi-Fi *fingerprinting*

Where@UM – collaborative positioning application based on Wi-Fi fingerprinting for PC

Dissertação de mestrado

Mestrado em Engenharia de Telecomunicações e Informática

Trabalho efetuado sob a orientação de:

Professor Doutor Adriano Jorge Cardoso Moreira

Professor Doutor Filipe Miguel Lopes Meneses

Outubro de 2016

AGRADECIMENTOS

Um dos objetivos que tinha definido na vida era concluir a minha formação académica. Esta dissertação marca o fim de uma etapa e ao mesmo tempo institui o início de outra.

Começo por deixar uma palavra de agradecimento aos meus orientadores, o Professor Doutor Adriano Moreira e o Professor Doutor Filipe Meneses. Agradeço a partilha do saber, a disponibilidade, as valiosas contribuições e sugestões que foram fazendo ao longo de todo o processo. Acima de tudo agradeço por ter trabalhado sobre a vossa égide.

Agradeço aos meus amigos que estão lá para os bons e para os maus momentos, pelo companheirismo, pela força, pelo humor e pelas sugestões. Em particular, deixo uma palavra de agradecimento ao Ivo Silva, Ricardo Peixoto e ao João Delgado pela amizade, apoio e pelas revisões a este documento. Não posso deixar de referir vários colegas de curso e docentes que me marcaram e influenciaram de alguma forma, acredito veemente que tudo acontece por uma razão e assim, deixo-lhes também uma palavra de gratidão.

Queria também deixar o meu profundo agradecimento a todos aquele que de uma maneira ou de outra contribuíram para a concretização desta dissertação, oferecendo palavras de motivação e sugestões.

Agradeço à Catarina pela amizade, amor e companheirismo. Pelas revisões semânticas e pela força que me dás e deste. Deixo ainda três palavras: “Posso?”, “Obrigado” e “Desculpa”.

Por fim gostava de concluir deixando um especial agradecimento aos meus pais, Carlos e Cristina pela educação e valores que me transmitiram, pela paciência e dedicação que mostraram ao longo destes anos. A eles, principalmente, dedico este trabalho.

RESUMO

A utilização do posicionamento, no âmbito das aplicações fornecidas aos utilizadores, tem vindo a aumentar exponencialmente. Os trabalhos desenvolvidos na área do posicionamento *indoor* têm vindo a aumentar com o aumento da mobilidade dos utilizadores.

As possibilidades de uso destas tecnologias são imensas: aumentar a experiência do utilizador e a lealdade, aumentar as vendas através de *marketing* de proximidade, ajudar a movimentação de utilizadores em locais públicos, o uso de *geofencing* para encontrar pessoas, etc.

De forma a reaproveitar as infraestruturas já existentes nos edifícios, a tecnologia de Wi-Fi *fingerprinting* tem sido uma escolha frequente por parte das equipas de investigadores e programadores. O principal objetivo desta dissertação é desenvolver uma aplicação para computadores pessoais usando um sistema de posicionamento baseado em Wi-Fi *fingerprinting*, integrando-a no sistema Where@UM.

A solução apresentada detalhadamente na dissertação implementa funcionalidades já disponibilizadas na aplicação *Android*. Foram também desenvolvidas novas respostas a problemas já existentes e integrados novos módulos na arquitetura, como a integração com as redes sociais e o suporte multiplataforma, tendo especial cuidado em manter alguma homogeneidade no ambiente aplicacional Where@UM, através do uso de interfaces de utilizador similares.

Palavras-Chave: Wi-Fi *fingerprinting*, aplicação *desktop*, MVVM, posicionamento *indoor*, integração com o *Facebook*

ABSTRACT

The positioning usage in desktop and mobile applications have been increasing exponentially. New applications and research in the indoor positioning field area are increasing proportionally with the intensification of the user's mobility.

The use of positioning creates endless possibilities for example: increase user experience and loyalty, increase sales through proximity marketing, navigation to location inside public buildings, geofencing to find friends in public spaces, etc.

Buildings existent infrastructures can be used, if the positioning technology is Wi-Fi fingerprinting. This has been a frequent choice by teams and developers worldwide. The main goal of this thesis is to develop a desktop application using a positioning system based in Wi-Fi fingerprinting, the application will be integrated with the Where@UM system.

The detailed solution that will be presented implements functionalities already provided by the *Android* application. Of course that new solutions answering problems were, also, implemented with the creation of new modules in the architecture such as: social networks integration and multiplatform support. During the implementation it was one objective to create homogeneity in the Where@UM environment, using similar user interfaces.

KEYWORDS: Wi-Fi fingerprinting, desktop application, MVVM, indoor positioning, Facebook integration

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

Abreviaturas e Acrónimos

API – Application Programming Interface

APs – Access Points

GCM – Google Cloud Messaging

GPS – Global Positioning System

HTTP – Hypertext Transfer Protocol

IP – Internet Protocol

IPS – Indoor Positioning Systems

JSON – JavaScript Object Notation

MVC – Model-View-Controller

MVP – Model-View-Presenter

MVVM – Model-View-ViewModel

REST – Representational State Transfer

RF – Radio Frequência

RSSI – Received Signal Strength Indicator

RTLS – Real Time Location System

SOAP – Simple Object Access Protocol

WPF – Windows Presentation Foundation

XAML – eXtensible Application Markup Language

1. INTRODUÇÃO

O capítulo inicial elucida o leitor principalmente para o enquadramento e motivações que levaram à realização da dissertação. Os objetivos desejados e a abordagem utilizada para os alcançar é também parte integrante do capítulo, no final é apresentada a estrutura da dissertação.

1.1 Enquadramento e motivações

O uso de dispositivos com acesso à Internet, nomeadamente, *smartphones* e *wearables* tem vindo a aumentar e com ele a necessidade de mais desenvolvimentos tecnológicos na área do posicionamento com mais precisão e nas condições mais adversas.

O facto de ser possível detetar a posição de um indivíduo ou objeto permite criar aplicações dinâmicas, que alteram o seu padrão de utilização de acordo com o local onde o utilizador se encontra. Este tipo de aplicações pode ser usado para deteção, monitorização, navegação, controlo de entradas e saídas, segurança, publicidade orientada à localização nos mais variados locais, *i.e.,* armazéns, lojas, hospitais, *shoppings*, aeroportos, instituições públicas, empresas, etc.

A maioria dos *smartphones* dispõe do módulo *Global Positioning System* (GPS) que permite obter as coordenadas geográficas usando a rede de satélites global. Contudo, os sinais GPS têm dificuldades de propagação dentro de edifícios [1], criando a necessidade para o desenvolvimento de técnicas que permitam o posicionamento em ambientes interiores.

A proliferação de redes Wi-Fi instaladas nos mais diversos locais, como instituições públicas, hospitais, escolas, residências e mais recentemente em espaços urbanos permite que sejam exploradas técnicas de posicionamento que tirem partido desta imensa infraestrutura já instalada. Usando as assinaturas de rádio, resultantes dos sinais transmitidos pelos *Access Points* (APs) que, sendo distintas em cada local, permitem criar associações entre locais e assinaturas de rádio. Estes sistemas denominam-se *Real Time Location Systems* (RTLS) [2].

Atualmente, já existem vários sistemas que permitem obter o posicionamento *indoor* usando as mais diversas tecnologias [2][3]. Há sistemas que usam o Wi-Fi *fingerprinting* tal como o sistema RADAR, que usa radiofrequência para adquirir a localização, através do armazenamento e processamento de múltiplos *Received Signal Strength Indicator* (RSSI). Esta técnica usa as estruturas já existentes nos edifícios (APs) mas depende da construção de mapas rádio. Fazendo o *matching* dos níveis atuais com o mapa é possível saber o local em questão [4][5]. Existem também sistemas que são baseados em

beacons que apesar de, tipicamente, usarem sinais de APs 802.11 [6], também podem usar dispositivos *Bluetooth*, torres celulares GSM, transmissores de rádio FM e combinações entre estes [7][8][9].

No mercado já existem soluções que integram *Indoor Positioning Systems* (IPS) em aplicações móveis como: IndoorAtlas, Wifarer, Accurate, Guardly e GoIndoor [3][10]–[13]. Como foi acima referido, as vantagens de usar este tipo de sistemas são imensas e existem multinacionais como a Google, a Apple e a Microsoft interessadas no desenvolvimento destas tecnologias [14]–[16].

1.2 Objetivos

O sistema Where@UM que conta com uma aplicação para dispositivos móveis *Android* é um sistema de posicionamento que usa Wi-Fi *fingerprinting* para determinar a localização dos utilizadores. Permite construir mapas de rádio, de forma colaborativa e autónoma, para edifícios de grande escala *i.e.* campus da Universidade do Minho [17].

O principal objetivo desta dissertação é o desenvolvimento de uma aplicação *desktop* para o sistema Where@UM. Atualmente, este sistema consiste numa só aplicação móvel para *Android* apoiada por um servidor.

A nova aplicação a desenvolver tem o intuito de capturar mais utilizadores e de fornecer mais plataformas de uso, ficando assim acessível a uma população alvo maior. Para servir esta arquitetura mais fragmentada (uso de diferentes plataformas) têm de ser consideradas alterações ao servidor que foi anteriormente desenvolvido considerando apenas a aplicação cliente *Android*. Paralelamente, novas funcionalidades e alterações aos serviços atuais estão dentro dos objetivos da dissertação.

A integração com as redes sociais é também um dos requisitos chave da dissertação. O facto de as redes sociais facilitarem o registo de novos utilizadores e incentivarem à partilha de informação (no caso da Where@UM é expectável a partilha da localização atual) cria mais exposição da aplicação e, por conseguinte, a possibilidade de aumentar o número de registos e utilizadores. Como a aplicação depende intrinsecamente da construção de mapas rádio, além dos números de utilização, a qualidade da informação também será melhorada.

1.3 Abordagem

Sendo os objetivos propostos desafiadores, para os alcançar foi necessário definir um conjunto de tarefas. O que levou a que o desenvolvimento fosse progressivo e consistente, culminando com os resultados esperados. De todas as tarefas realizadas é possível distinguir as seguintes:

1. Caracterização do estado da arte, envolvendo o estudo das tecnologias e sistemas de posicionamento já existentes. A recolha bibliográfica é fundamental para aprender e compreender as principais técnicas de posicionamento e o conhecimento dos sistemas que já foram desenvolvidos permite desenhar uma solução mais rica e capaz. Como esta dissertação envolve a criação de um novo ramo num sistema já existente, a familiarização com a Where@UM também será fundamental;
2. Desenvolvimento de uma aplicação para computadores pessoais (para o sistema operativo *Windows*), que engloba a aprendizagem da linguagem de programação C#, a escolha de um modelo de programação e a modificação do servidor existente de forma a garantir o funcionamento do sistema para multi-plataforma;
3. Desenho de um mecanismo para integração da aplicação para computadores pessoais com as redes sociais, garantindo um registo rápido usando a conta do utilizador registada nas redes sociais (*Facebook*);
4. Implementação e testes da solução desenvolvida. Alterações aos sistemas já implementados de forma a garantir a integridade da solução a funcionar em multiplataforma;
5. Avaliação do desempenho da solução desenvolvida;
6. Redação da dissertação relatando concretamente o trabalho que foi desenvolvido explicitando os pontos mais importantes do desenvolvimento do trabalho.

1.4 Estrutura da dissertação

A dissertação encontra-se dividida em 7 capítulos. No primeiro capítulo, onde esta secção se encontra, é realizada a introdução à dissertação, esclarecendo o enquadramento e motivação para a sua realização, os objetivos e a abordagem utilizada para os concretizar.

No segundo capítulo é apresentado o estado da arte, onde é realizada uma descrição das técnicas de posicionamento, de sistemas de posicionamento de interiores e de aplicações disponíveis.

O terceiro capítulo é constituído por uma introdução ao sistema já em produção da Where@UM e uma descrição detalhada da arquitetura legada.

O capítulo quatro contempla o desenho do sistema, são analisados os requisitos do sistema, é apresentada a arquitetura do sistema, é exposto o modelo de dados e ainda explicado a abordagem ao uso de mapas na aplicação (*Bing Maps*) e a integração com as redes sociais.

A implementação do sistema é detalhada no capítulo cinco, que está dividido em 4 secções: a aplicação para computador pessoal, o servidor, as alterações de segurança e as alterações nas fingerprints. É apresentado, ainda, o padrão de desenvolvimento, a aplicação cliente, a implementação do servidor Where@UM e do servidor de chat.

O capítulo seis apresenta uma avaliação ao sistema com testes a funcionalidades.

A conclusão e os trabalhos futuros são apresentados no capítulo sete, expondo uma análise ao trabalho desenvolvido no âmbito da dissertação e sugerindo aspetos que podem ser melhorados no sistema e na aplicação.

2. ESTADO DA ARTE

Neste capítulo são apresentadas diferentes técnicas e sistemas de posicionamento usados para estimar a localização em ambientes *indoor.* São ainda abordadas diferentes soluções académicas e comerciais que usam algumas das técnicas de posicionamento mencionadas neste capítulo e que são referências nesta área.

O sistema de posicionamento mais conhecido a nível mundial é o GPS, que nos dias de hoje, já vem integrado na maior parte dos dispositivos móveis e em muitos dos novos veículos disponíveis comercialmente. Seria vantajoso se este sistema, que já é amplamente aceite, fosse também utilizado em ambientes *indoor*. No entanto, este é ineficiente dentro de edifícios: a transmissão do sinal entre o dispositivo e os satélites é claramente atenuada e não se pode negar que a localização neste tipo de espaços é bastante mais complexa. O facto de haver obstáculos (como paredes, tetos, janelas, móveis e equipamentos) que interferem com os sinais eletromagnéticos, dificulta o processo de posicionamento. A necessidade de existir uma "linha de vista" entre os satélites e os respetivos recetores é um problema que também afeta os sistemas de posicionamento global Galileo e Glonass.

Com a crescente necessidade de obter a localização de dispositivos e pessoas dentro de edifícios iniciou-se o estudo de técnicas e tecnologias que permitissem obter o posicionamento com maior precisão.

2.1 Técnicas de posicionamento

As técnicas de posicionamento podem ser separadas em dois grupos: as não baseadas em Radiofrequência (non- RF) ou as baseadas em Radiofrequência (RF) [18].

2.1.1 Triangulação – lateração e angulação

A triangulação é uma técnica de determinação da posição de um ponto formando triângulos com pontos já conhecidos. Esta técnica usa as propriedades geométricas dos triângulos para estimar a localização do alvo.

Existem várias formas de medir a distância ao ponto de referência:

- *RSS-Based* ou *Signal Attenuation-Based*, é calculado através da diferença de potência entre o sinal emitido e o sinal recebido. Devido ao *multipath* em ambientes *indoor*, este

tipo de técnicas serve-se de algoritmos que calculam as distâncias estimadas ao objeto [19];

- *Time of Arrival (TOA),* este indicador temporal indica o tempo que um sinal demora entre o emissor e o recetor por inferência, e sabendo a velocidade de propagação no meio, é possível estimar a distância. O maior desafio desta técnica é a necessidade de sincronização do emissor e do recetor [19]*;*
- *Time Difference of Arrival (TDOA),* determina a localização da fonte comparando a diferença no tempo de chegada do sinal a estações base separadas. Esta técnica como na TOA depende da sincronização das estações base [20]*;*
- *Roundtrip Time of Flight (RTOF)*, esta técnica mede o *time of flight* do sinal transmitido entre o emissor e o recetor mais o tempo do pacote de confirmação do recetor até ao emissor. Sabendo a velocidade de propagação do sinal no meio é possível determinar a distância [19];
- *Phase of Arrival (POA)* usa a fase da portadora do sinal recebido para determinar a distância, usando esta técnica é assumido que todos os emissores transmitem sinais sinusoidais à mesma frequência e com a mesma fase. Estes sinais quando recebidos vão chegar desfasados. Usando esta fase adquirida durante a propagação consegue-se estimar a distância. Num ambiente interior é aconselhado usar este método conjugado com outros, como o TOA ou o RSS para maior precisão [19].

2.1.2 Cell of Origin (CoO)

Usando esta técnica, a localização é determinada pela célula de origem (*CoO)*. Metodologia que identifica a célula a que o terminal móvel está ligado, assumindo como essa a sua localização. Infere-se, portanto, que o nível de precisão desta técnica varia de acordo com a dimensão das células existentes num determinado local. Quando é pretendida maior acuidade, tipicamente, usam-se técnicas como o GPS em paralelo. Como usa o *hardware* e as capacidades intrínsecas da rede, tem como vantagem imediata o facto de identificar rapidamente a localização e de não serem necessários quaisquer atualizações de *hardware* à rede [19].

2.1.3 Map Matching (MM)

Os algoritmos de *map matching* comparam os dados recolhidos do posicionamento em tempo real com um mapa digital, baseando-se nas teorias de reconhecimento de padrões. O *Map Matching* é usado tipicamente para trajetórias fixas [21], como por exemplo os sistemas de navegação GPS utilizam predominantemente esta técnica, normalmente nos veículos, apesar de agora os dispositivos móveis já fazerem parte do ambiente de utilização onde esta técnica é usada.

Como os sistemas de posicionamento global, como o GPS, têm alguns problemas de precisão é usual complementar este tipo de técnicas com outras, visando o aumento de desempenho na localização precisa de objetos (veículos, pessoas, etc..).

2.1.4 Dead Reckoning (DR)

O *Dead Reckoning* determina a localização atual tendo como base a posição interior. Estima a posição através da direção do movimento, da velocidade e da diferença temporal. Como é uma técnica cumulativa, que depende de informações anteriores, a posição atual vai ficando degradada e menos precisa com o passar do tempo. Esta é, sem dúvida, a maior desvantagem do uso desta técnica [22].

2.1.5 Fingerprinting ou Scene Analysis

É uma técnica baseada em RF e usa algoritmos que primeiro recolhem as *fingerprints* e posteriormente estimam a localização do objeto, fazendo a comparação entre os resultados recolhidos em tempo real com os valores recolhidos *a priori*.

As *fingerprints* podem ser realizadas usando várias métricas como o som, imagem, Wi-Fi, etc. Neste trabalho, a métrica mais interessante é o RSSI, indicando a potência do sinal recebido, proveniente dos sinais Wi-Fi.

Existem duas fases para usar esta técnica como método de localização, a *offline* (fase de calibração) e a *online* (ou *run-time*). A fase de calibração consiste na recolha de *fingerprints* em vários pontos de cada divisão dentro do edifício. Estes dados são armazenados numa base de dados e, de seguida, podem ser lidos e usados para a criação do mapa rádio; devem ser recolhidas várias *fingerprints* para aumentar a robustez dos mapas rádio tendo em mente que até a utilização da própria sala causa variações dos sinais de rádio.

Na fase *online* ou *real-time*, os algoritmos recebem os valores lidos pelos dispositivos móveis (ou estáticos) e comparam-nos com os existentes no mapa de rádio (os valores anteriormente armazenados), permitindo assim obter a estimativa da localização atual do dispositivo.

O uso do RSSI como *fingerprint* é *per se* a principal dificuldade no uso desta técnica, porque estes sinais são afetados por fenómenos que alteram a sua propagação como a reflexão, refração, difração e o *scattering.* A reflexão é causada quando a onda emitida incide numa superfície e volta para o mesmo meio. A refração é um fenómeno que consiste no desvio das ondas emitidas, do percurso original para outro percurso, quando mudam de meio de propagação. A difração é o fenómeno das ondas conseguirem contornar obstáculos ou conseguirem propagar-se por aberturas. O *scattering* é o fenómeno das ondas sofrerem alterações do percurso original e serem refletidas em múltiplas direções. Estes fenómenos causam o *multipath* no qual o sinal transmitido entre o emissor e o recetor vem por vários caminhos, isto faz com que seja causada interferência no sinal e uma mudança na fase do sinal recebido. Na Figura 2.1 é possível observar uma imagem que esclarece visualmente estes problemas que podem influenciar o alcance ou precisão dos sistemas de posicionamento.

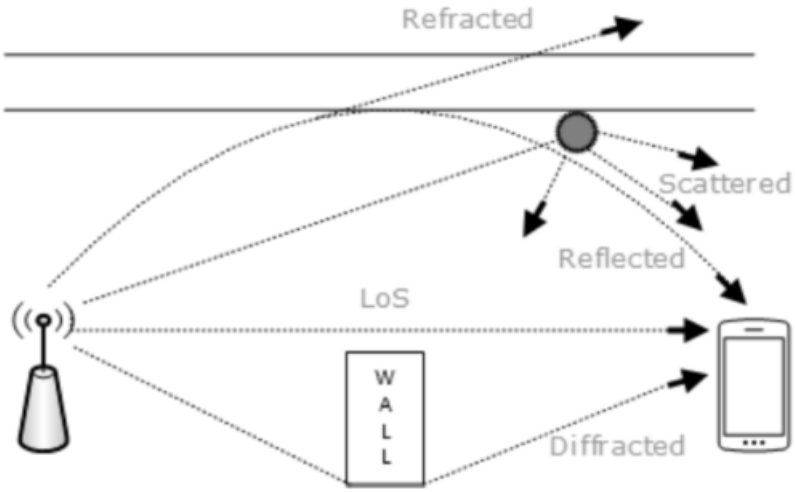

Figura 2.1 - Causas do multipath [19]

A facilidade com que se cria um *hot spot* ou o dinamismo que os ambientes interiores têm (mudanças de mobília, acréscimo de cortinas e tapeçarias, etc), criam variações nos sinais recebidos dos pontos de acesso. Assim, para que o sistema funcione nas melhores condições é importante manter o mapa de rádio atualizado. A construção e posterior atualização dos mapas rádio pode ser feito manualmente, automaticamente [6] ou através de um processo colaborativo [23].

Existem alguns algoritmos genéricos para implementar técnicas baseadas em *fingerprinting*: métodos probabilísticos, *k-nearest-neighbor* (kNN), redes neurais, *support vector machine* SVM e *smallest M-vertex polygon* (SMP) [2].

2.2 Sistemas de posicionamento

A técnica usada para a obtenção de informação dos APs é importante, mas não se pode descurar uma análise aos sistemas já existentes de forma a perceber qual a melhor maneira de receber essa informação. Simultaneamente, à análise das técnicas para obtenção de informação deve ser adicionada uma análise aos sistemas em geral, de forma a entender melhor como foram desenvolvidos e usar esse *know-how* para melhorar o sistema Where@UM.

2.2.1 Herecast

Quando foi desenvolvido, o sistema de posicionamento Herecast [24] tinha como objetivo colmatar algumas falhas nos sistemas de posicionamento *indoor*. Num ambiente em que eram, geralmente, utilizados sistemas especializados (com hardware específico) e com custos elevados era importante encontrar uma solução mais barata. Assim, os responsáveis por este projeto decidiram utilizar a rede *wireless* e os APs já existentes nos edifícios.

O sistema Herecast segue uma abordagem colaborativa, sendo os utilizadores a introduzir e atualizar as informações. Na Figura 2.2 é possível observar os formulários de introdução de APs. A sua arquitetura está dividida em três áreas: gestão dos dados, *software* no cliente e serviços.

Figura 2.2 - Formulários de introdução da aplicação Herecast [24]

Na componente que está instalada no dispositivo do cliente é recolhido o endereço MAC dos APs visíveis e o respetivo RSSI. A aplicação armazena toda a informação relativamente aos APs e determina qual deles apresenta o melhor RSSI, consequentemente envia esta informação para o servidor utilizando uma conexão *Hypertext Transfer Protocol* (HTTP).

Foram desenvolvidos vários serviços para o Herecast:

- **Area Maps** – tipicamente os serviços de mapas já existentes têm bom comportamento para navegação nas estradas, contudo cada vez são encontrados mais benefícios em mostrar informação mais detalhada da área em que nos encontramos. A *Area Maps* é um serviço que permite a criação de mapas detalhados de uma área especifica, são otimizados para um ecrã de um *Pocket PC* e podem ser navegados usando cliques.
- **Friend Finder** – este serviço apresenta numa página *web* a última localização conhecida dos utilizadores. Esta informação é publicada automaticamente conforme o utilizador se movimenta de local para local.
- **Heresay** – sendo um *message board* permite a introdução de mensagens num quadro público, estas mensagens só podem ser consultadas por utilizadores que estejam no mesmo edifício, no qual a mensagem foi enviada.
- **Bandwidth Advisor** – os utilizadores que tenham problemas de conectividade à *Internet* conseguem aceder a uma lista de APs na área e consultar quais são os que têm mais largura de banda disponível. Este serviço não só ajuda os utilizadores sugerindo áreas onde estes possam obter melhor conetividade, mas também ajuda a rede a balancear o tráfego. A implementação deste serviço implica recolher estatísticas dos APs usando o SNMP *(Simple Network Management Protocol)*.

2.2.2 Radar

O sistema Radar [5] desenvolvido pela *Microsoft Research* foi pioneiro no posicionamento em ambientes *indoor* usando o Wi-Fi *fingerprinting*, através de medições do RSSI.

Este sistema usa a técnica de Wi-Fi *fingerprinting* descrita na subsecção 2.1.5 e tem como mediana de precisão valores entre os 2 e 3 metros, possíveis de obter graças ao uso do algoritmo NNSS *(nearest neighbor(s) in signal space)*. Usando as medições do RSSI realizadas em cada localização é feita a comparação entre os valores armazenados na base de dados e os valores obtidos *online*, após a realização de cálculos usando a distância Euclidiana é possível estimar a localização do utilizador.

Na Figura 2.3 é possível observar o mapa do piso onde foi realizado o teste de campo do Radar, as estrelas cinzentas que se encontram na planta representam a localização dos APs e os círculos pretos indicam o local onde foram recolhidas as *fingerprints*.

A desvantagem deste sistema é a necessidade de recalibração ao longo do tempo face a mudanças no número e localização de APs ou estruturais nos edifícios.

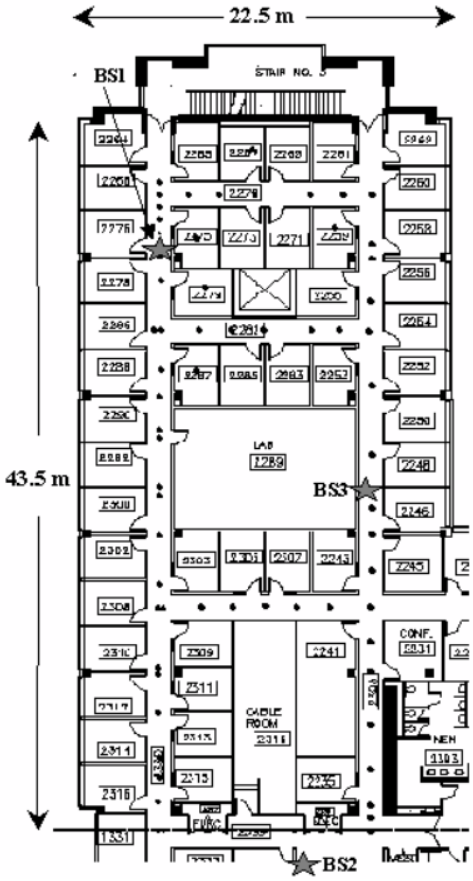

Figura 2.3 - Planta da experiência realizada pela equipa do Radar [5]

2.2.3 Redpin

É um sistema de posicionamento *indoor* que usa a técnica de Wi-Fi *fingerprinting,* apresentada na subsecção 2.1.5. A sua implementação teve influências de sistemas como o Radar (2.2.2) ou o Place Lab [25].

A maior inovação que o Redpin [23] apresenta é a abordagem de treino *online*, permitindo que o sistema seja mais dinâmico incorporando o treino da aplicação no uso comum dos utilizadores (através

de um processo colaborativo), o que retira a fase de treino *offline* que é morosa. O objetivo do Redpin é usar dispositivos móveis para recolher as *fingerprints*.

Através da análise de outros sistemas como o COMPASS [26] que consegue determinar as posições com um erro médio de 2.05 m, mesmo fazendo medições espaçadas de 1 m e em 8 direções diferentes, a equipa que desenvolveu este sistema concluiu que era pouco provável conseguirem atingir precisões abaixo de 2 m. Então, usando inquéritos sobre operacionalidade, conclui-se que é suficiente localizar os utilizadores por divisão dos edifícios (para a maioria das aplicações).

Como foi acima referido, o Redpin serve-se dos utilizadores para atualizar e criar mapas rádio. Na Figura 2.4 é possível observar a interface da aplicação cliente.

Figura 2.4 - Aplicação cliente do Redpin usando um Nokia N95 [23]

Além do uso dos APs, este sistema também usa a força de sinal GSM e do identificador *Bluetooth* proveniente de todos os dispositivos no alcance do dispositivo móvel.

Foram realizados testes pela equipa de desenvolvimento do sistema provando que é capaz de determinar a localização (na divisão) correta 9 em cada 10 vezes. No local onde foram realizados os testes a equipa provou que com apenas 10 pessoas o mapa fica completo num dia. É claro que o tempo necessário é influenciado pela mobilidade dos utilizadores do sistema e pela área do edifício.

2.2.4 COMPASS

O sistema COMPASS [26] sistema serve-se do uso de Wi-Fi *fingerprinting* e de bússolas digitais para determinar o posicionamento.

No dispositivo móvel, o utilizador envia *fingerprints* do RSSI e também usa a orientação do dispositivo para a fase de treino do sistema (técnica referida em 2.1.5). O COMPASS usa um algoritmo de estimação probabilística para determinar a posição do utilizador.

Os sinais emitidos pelos APs têm uma frequência de 2.4 Ghz ou de 5 Ghz. Em testes realizados pela equipa de desenvolvimento mostrou-se que o corpo humano introduz uma atenuação na ordem dos 15 dB. Na Figura 2.5 é possível observar graficamente a interferência causada pelo corpo humano.

Para contrariar este facto e defendendo que o corpo humano é responsável por uma grande parte destes problemas, a equipa de desenvolvimento do sistema decidiu explorar o uso de bússolas digitais na fase de treino.

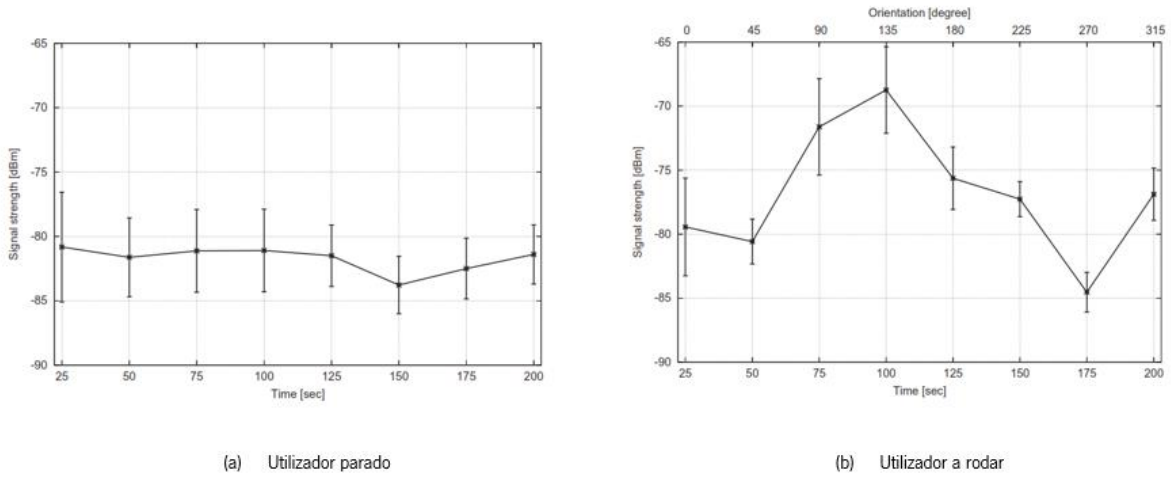

Figura 2.5 - Interferência causa pelo corpo humano [26]

O algoritmo de posicionamento serve-se de amostras do RSSI em direções selecionadas (8 direções) em cada ponto de referência (durante a fase de treino) e combina-os com os valores na fase *online*. Os pontos de referência foram encontrados aplicando uma grelha à área alvo com cada ponto espaçado 1 m do seguinte.

O ambiente de testes na Universidade de Mannheim com 312 m^2 traduziu-se em 146 080 medições (na fase de treino) e foram despendidas 10 horas.

A equipa provou que a diferença de uma fase de treino que faça medições em 8 direções por ponto de referência consegue melhorias na ordem dos 22% face a um cenário de uma medição.

Para o desenvolvimento deste sistema foi escolhido o algoritmo *Multiple Nearest Neighbors* [5] e relativamente ao RADAR, apresentado na subsecção 2.2.2, o COMPASS tem *performances* 25 % melhores em termos de precisão abaixo dos 2 m, simultaneamente também foi provado que tem um erro máximo menor, na ordem dos 11 m enquanto que o RADAR apresenta 15 m obtendo-se as médias finais de 1.65 m no caso do COMPASS e 2.26 m no caso do RADAR.

Na Figura 2.6 é possível observar uma comparação de *performance* entre o COMPASS e o RADAR.

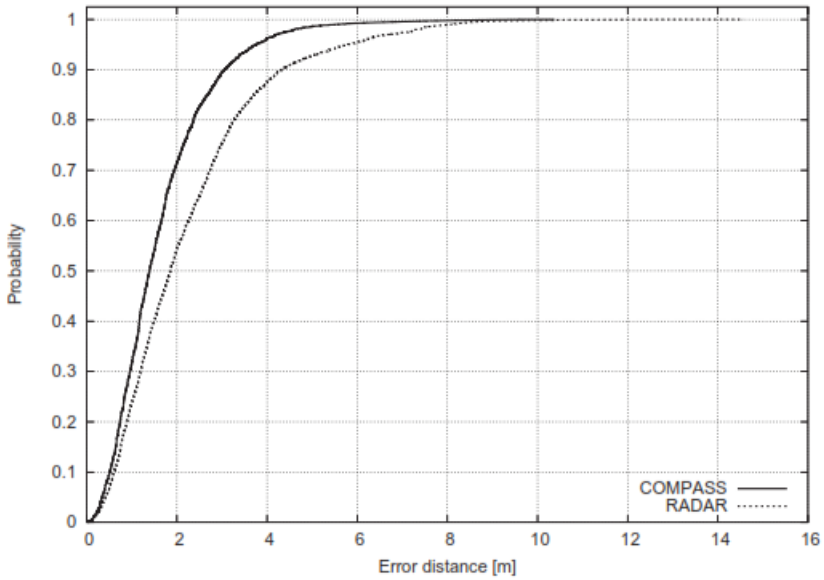

Figura 2.6 - Distribuição cumulativa da performance do COMPASS vs RADAR [26]

2.2.5 Molé

O Molé [27] foi apresentado na conferência internacional *Indoor Positioning and Indoor* Navigation (IPIN), 2011, em Guimarães.

É um sistema de posicionamento colaborativo que usa um sistema de nomeação hierárquico tendo uma abordagem restrita para os nomes de cada local (ver Figura 2.7) e, não sendo condicionado pelo uso de mapas, o Molé pretende conseguir ser um sistema mais flexível e escalável dentro dos sistemas de posicionamento *indoor*. Basicamente, o Molé organiza as *fingerprints* hierarquicamente com 5 níveis: país, região, cidade, área e local específico. Além da aparente simplicidade de hierarquização, outra vantagem é que os dados das *fingerprints* podem ser previamente carregados para os dispositivos, desde que respeitem a estrutura do sistema.

Figura 2.7 - Interface de um utilizador do Molé [27]

O Molé apresenta novas técnicas como um algoritmo de posicionamento estatístico para distinguir melhor os locais dos seus locais vizinhos, um detetor de movimento (usando o acelerómetro) e um sistema de *fingerprints* na *cloud*.

O seu algoritmo de posicionamento é o MAO *(Maximum Overlap)* que tem em consideração as variações temporais das *fingerprints* e usa uma função de similaridade. Foram identificadas duas grandes vantagens, sendo a primeira a eficiência da computação e a segunda a obtenção da função de distância (usada para estimar distâncias entre objetos, é útil para detetar *outliers* e para de seguida limpar a base de dados). Usando a técnica de sobreposição de duas distribuições Gaussianas é possível determinar a semelhança entre *fingerprints* (ver Figura 2.8).

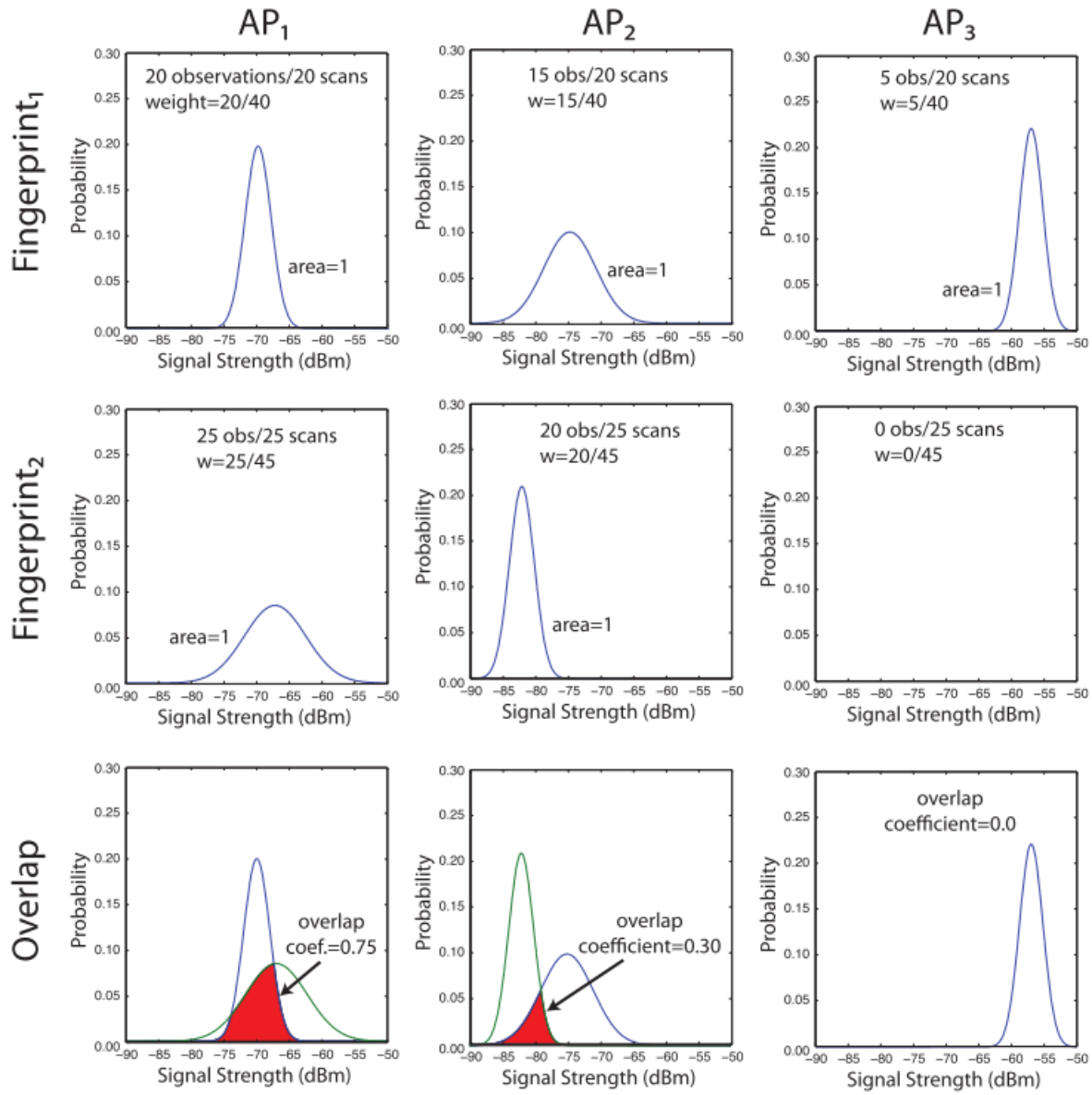

Figura 2.8 - Exemplo do funcionamento do algoritmo MAO [27]

Por observação dos gráficos conclui-se que dos 20 *scans* que foram realizados para a *Fingerprint 1* foram detetados 3 APs, o primeiro 20 vezes, o segundo 15 e o terceiro 5 respetivamente. Assim, neste local o peso resultante por AP (representado pela variável w) é 20/40, 15/40 e 5/40. Foi realizada uma observação igual para a *Fingerprint 2*, o peso para cada um dos APs é 25/45, 20/45 e 0/45.

A comparação entre *fingerprints* é realizada baseando-se na potência de sinal dos APs que são iguais e é penalizada quando algum dos APs não é encontrado numa das *fingerprints*. Na Figura 2.8 existe uma explicação visual de como é calculado o coeficiente de sobreposição.

2.2.6 Ekahau RTLS

Este sistema recentemente adquirido pela empresa Airista Flow [28] foi desenvolvido na Finlândia. Apresentando um erro médio inferior a 1 m, distingue-se por ser um dos sistemas mais precisos de posicionamento [29].

O Ekahau [30] combina redes Bayesianas, complexidade estocástica e aprendizagem *online*, serve-se de um servidor central para calcular o posicionamento [2] (a arquitetura é apresentada na Figura 2.9) permitindo uma escalabilidade enorme.

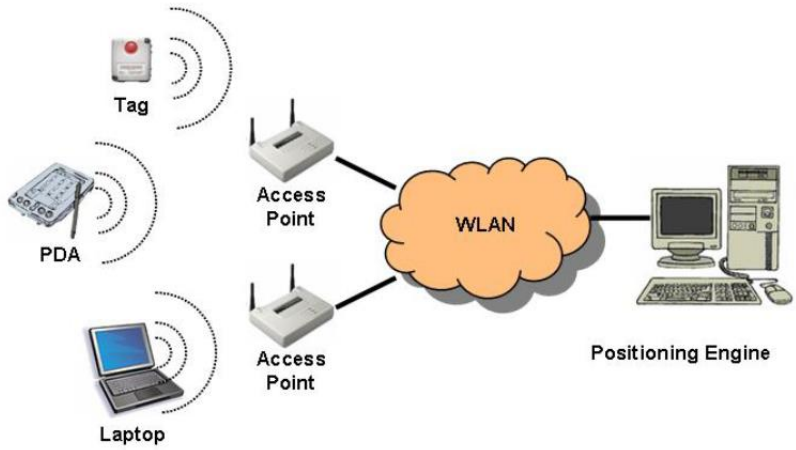

Figura 2.9 - Arquitetura do Ekahau RTLS [30]

O sistema que usa a técnica de triangulação é composto por 3 partes: o motor de posicionamento, o *software* de calibração e uma *tag* Wi-Fi. O motor de posicionamento permite localizar milhares de *tags* ou dispositivos. O *software* de calibração permite a calibração do sistema e a *tag* Wi-Fi, que é colocada em objetos que não tenham conexão, emite sinais de radiofrequência.

O motor de posicionamento, que é baseado em java, disponibiliza os serviços a aplicações cliente. Este sistema já é comercializado e tem excelentes resultados em ambientes hospitalares, na localização de *stocks* e na indústria automóvel [29].

2.2.7 Wifarer

Este sistema oferece diferentes soluções de posicionamento *indoor*. Tendo como alvo o mercado de *smartphones* oferece uma solução para iOS e *Android*.

Disponibiliza SDKs (*Software Development Kits*), para a comunidade de programadores, capazes de integrar as tecnologias principais já implementadas no sistema como o posicionamento, a navegação *bluedot*, LBS *(Location-based system)* e *movement analytics* [31].

Algumas das funcionalidades presentes neste sistema são [32]:

- Sistema de posicionamento *indoor;*
- Navegação da localização atual ao destino;
- Mapas atualizados;
- Diretórios interativos;
- Promoções relacionadas com sistemas de localização *indoor*.

Esta aplicação já tem sido usada nas mais diferentes áreas e ambientes *i.e.* aeroportos, hospitais, museus, lojas de retalho, estádios, universidades, etc [10]. Na Figura 2.10 é apresentado o menu principal da aplicação num dispositivo *Android*.

Figura 2.10 - Imagem da aplicação Wifarer para Android

2.2.8 Find My Friends

A aplicação móvel Find My Friends [33] permite localizar os amigos do utilizador em qualquer momento ou em qualquer lugar. O centro da aplicação gira à volta de um mapa, fornecido pelo serviço da *Google Maps*, no qual aparecem sob a forma de um ícone os amigos mais próximos da localização do utilizador.

A aplicação atualiza o mapa em tempo real permitindo localizar os amigos de forma rápida, normalmente usando o sistema GPS. No caso de um amigo não ser detentor de um *smartphone* ainda é possível localizá-lo porque a aplicação usa um sistema de triangulação das torres do fornecedor da rede celular. Assim, no mapa aparecem os amigos que usem *smartphones* ou telemóveis mais antigos.

Na Figura 2.11 é possível observar o mapa que é mostrado ao utilizador com a localização dos amigos.

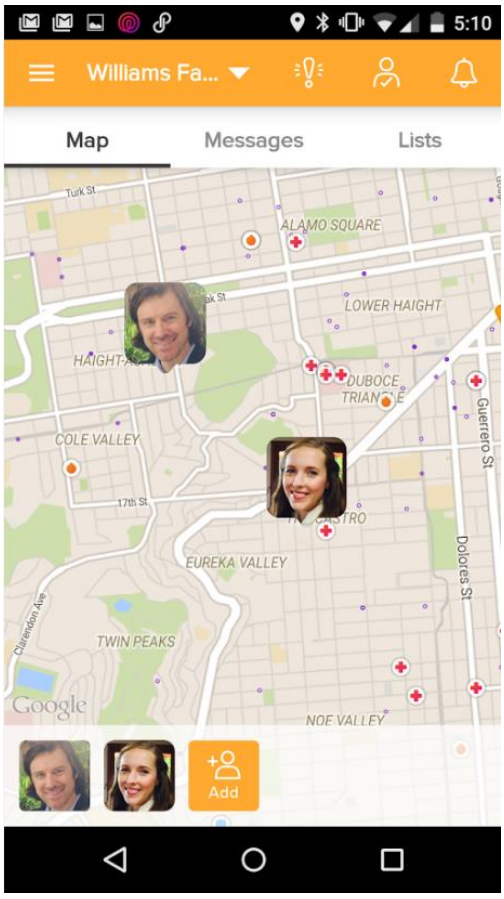

Figura 2.11 - Imagem da aplicação Find My Friends

2.2.9 Horus

Este sistema probabilístico foi baseado no Radar em 2.2.2. A sua implementação baseou-se em dois requisitos principais: excelente precisão e baixo esforço computacional. No local de testes, este sistema conseguiu garantir, em média, um erro inferior a 0.6 m.

Os modelos do Horus [34] servem-se das distribuições paramétricas e não-paramétricas, da potência de sinal recebida dos APs de forma a reduzir o efeito das variações temporais.

O Horus usa um módulo de *Clustering* para agregar as localizações que tenham os mesmos APs como fornecedores de rede. O *Discrete Space Estimator* apenas devolve o mapa de rádio que tem maior probabilidade tendo em consideração o vetor de potência de sinal recebido dos vários APs. O módulo *Correlation Modelling and Handling* utiliza um modelo autorregressivo para calcular a correlação entre duas amostras consecutivas e, por conseguinte, obter uma melhor estimativa da localização. Tem implementados dois módulos, nomeadamente: *Continuous Space Estimator* e *Small-Scale Compensator* que ajudam a corrigir a estimativa da posição do utilizador.

Em [34] demonstra-se que a correlação entre amostras, vindas do mesmo AP podem ter correlações até 90 %.

Comparativamente ao Radar, nos dois locais de teste, o sistema Horus mostrou ter uma *performance* 80 % superior. Na Figura 2.12 é possível observar a arquitetura deste sistema.

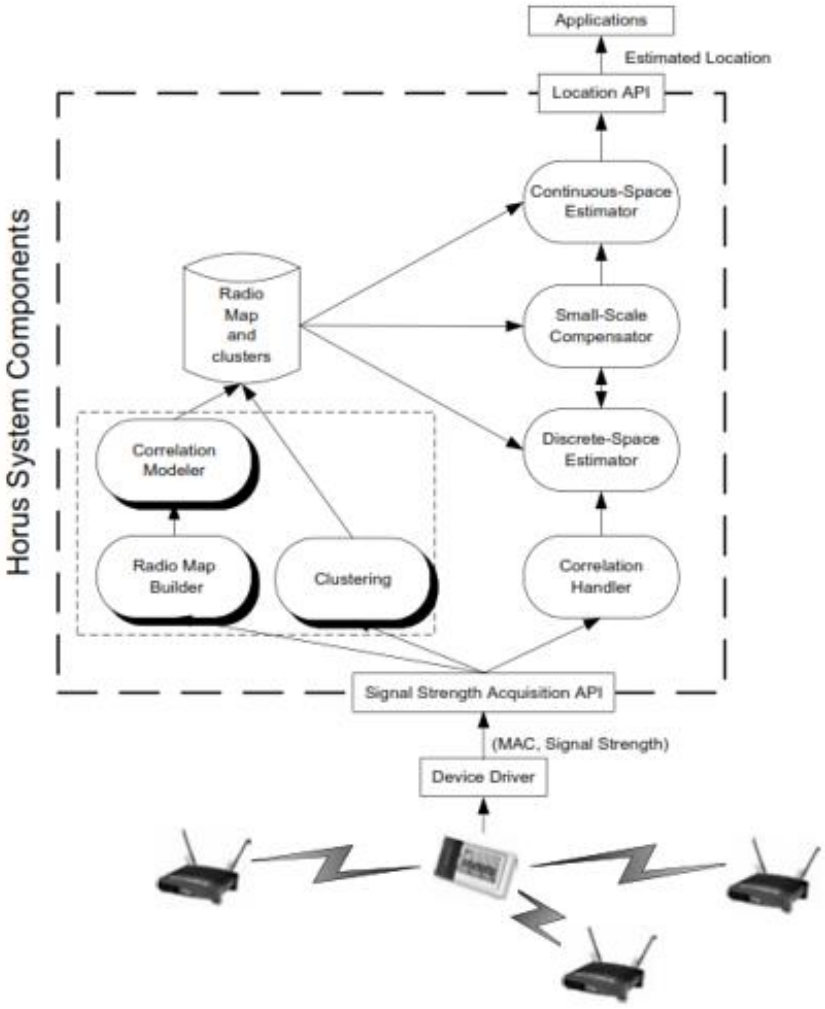

Figura 2.12 - Arquitetura do sistema Horus [34]

2.2.10 Pazl

A abordagem do sistema Pazl [35] centrou-se no uso de Wi-Fi *fingerprinting* juntamente com *Dead Reckoning* para a localização de utilizadores. Similarmente com outros sistemas, o Pazl serve-se do conceito de *mobile crowdsensing* para conseguir localizações mais precisas.

Foi desenvolvido um mecanismo de localização híbrido para dar resposta ao desafio que pode ser observado na Figura 2.13.

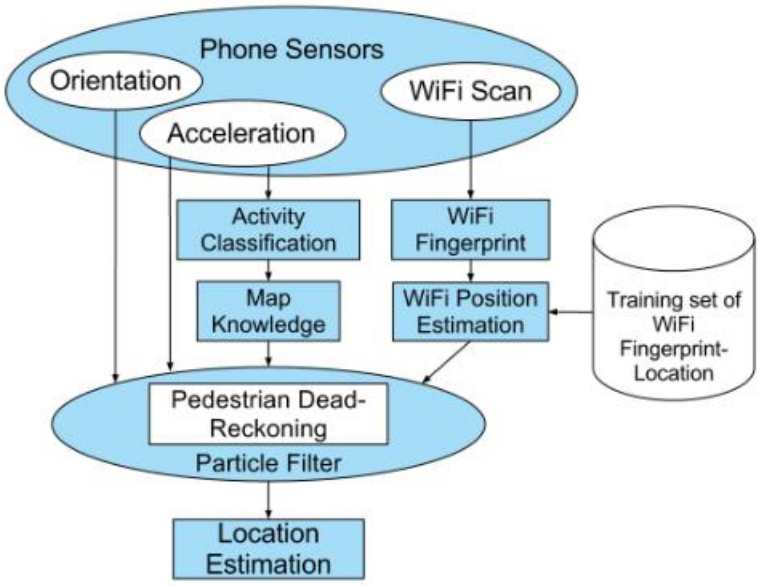

Figura 2.13 - Esquema do sistema de localização híbrido do Pazl [35]

Os sensores do dispositivo móvel recolhem amostras da aceleração, orientação e fazem *scans* aos APs. A aceleração é utilizada pelo classificador de atividade para detetar a movimentação do utilizador, juntamente com a orientação são *inputs* fundamentais para o funcionamento do *Pedestrian Dead Reckoning* que realiza a estimação da localização em movimento contínuo. O módulo de estimação da posição, usando Wi-Fi, compara os resultados obtidos nos *scans* em tempo real com os já armazenados para determinar a localização.

O Pazl tem duas partes: a aplicação móvel que faz recolha sensorial dos dados e um servidor que recebe os dados e os processa.

A aplicação apenas utiliza o acelerómetro, a bússola e a *interface* de rede quando o utilizador está em movimento, quando este está parado apenas o acelerómetro é mantido ligado para poupar bateria. Quando o utilizador recomeça a movimentar-se todos os sensores são ligados novamente.

A equipa de desenvolvimento do Pazl conclui que este tem resultados similares às melhores soluções de posicionamento *indoor,* mas fazendo-o de forma mais automática, graças à estimação do mapa de cobertura Wi-Fi usando interpolação espacial.

2.3 Considerações finais

O tempo despendido a analisar sistemas e técnicas, apesar de moroso, é necessário e bastante importante para a assimilação de conceitos chave e para haver uma base comparativa sobre a qual a dissertação estará assente.

Concluindo, a atenuação dos sinais RF é um dos problemas mais desafiantes no desenvolvimento de ferramentas de posicionamento *indoor*. E sendo o sistema Where@UM baseado em Wi-Fi *fingerprinting* este é um dos principais desafios ao aumento de precisão na localização. Na Tabela 1 é possível observar alguns valores de atenuação para efeitos de elucidação do leitor [36].

Tabela 1 - Tabela de atenuação causa por vários materiais nos sinais 5 Ghz e 2.4 Ghz [36]

Material do Edificio	5GHZ atenuação (dB)	2.4Ghz atenuação (dB)
Porta de madeira	6 a 7	3 a 4
Parede de tijolos/betão	10 a 30	6 a 18
Janela de vidro	6 a 8	2 a 3
Vidro duplo	20	13
Porta de emergência metálica	25 a 32	13 a 19

A análise de vários sistemas já utilizados a nível académico e empresarial mostrou-se benéfica, pois fornece *know-how* sobre vários tipos de aplicações e abordagens aos mesmos problemas.

Foram analisadas várias aplicações que fazem combinação de várias tecnologias para obter a localização dos seus utilizadores, como GPS, *Bluetooth*, GSM (ou outra qualquer tecnologia de rede fornecida pelo operador) ou *beacons.*

Como os computadores pessoais não vêm equipados com módulos GPS foi analisada a *Application Programming Interface* (API) *Windows Devices Geolocation* [37] que estima a posição baseando-se em *beacons* como APs Wi-Fi ou torres celulares ou no *Internet Protocol* (IP) do utilizador (caso o dispositivo tenha um módulo GPS também está preparado para o utilizar).

Assim esta análise realizada a diversas aplicações móveis para posicionamento *indoor* fez com que a perceção das necessidades dos utilizadores aumentasse, isto traduz-se numa maior preocupação no produto final, centrando a aplicação *desktop* naquilo que os utilizadores pretendem. Como o sistema Where@UM já integra uma aplicação *Android* é necessário que a nova aplicação mantenha a coerência funcional retirando resistência aos utilizadores que queiram usar a aplicação noutra plataforma e aumentando a confiança de todos os utilizadores.

3. ANÁLISE DO PROBLEMA

Neste capítulo é introduzido o problema tratado neste trabalho e é realizada uma análise do sistema Where@UM que já se encontra em produção. É ainda apresentada a arquitetura legada do sistema.

3.1 Introdução

O sistema Where@UM consiste num sistema de posicionamento de interiores que constrói mapas rádio de forma colaborativa. A arquitetura do sistema Where@UM já conta com um servidor, uma aplicação cliente para dispositivos *Android*, e armazenamento de dados para garantir integridade e o bom funcionamento da aplicação.

Este sistema não só permite obter a localização específica dentro da Universidade do Minho, como também faz uso do sistema de GPS para indicar a localização de alto-nível fora do campus. Em complementaridade, tem integração com a API *Foursquare* garantindo mais informações que auxiliam os utilizadores na descrição dos novos locais que registam no sistema.

Sendo um sistema colaborativo, todas as informações nele existentes dependem da boa vontade dos utilizadores em partilhar. Desta forma, têm sido pensadas formas de captar mais utilizadores e mantê-los como participantes regulares. Sabendo que a qualidade dos mapas rádio aumenta se as *fingerprints* forem mais recentes, havendo mais utilizadores há maior probabilidade de os locais serem atualizados mais vezes.

Surge assim a necessidade de facilitar o registo na aplicação através do uso do *Facebook.* Esta opção, que não fazia parte do sistema, permite fazer o registo de forma célere usando a conta do utilizador no *Facebook* como forma de registo e de *login* na aplicação. Com esta facilidade é expectável que a aplicação capte mais utilizadores para o seu uso regular.

A aplicação para computador pessoal vem assim suprir algumas das necessidades que têm vindo a ser encontradas ao longo do tempo no qual a aplicação se encontra disponível para o público. Sendo o registo da aplicação realizado através do *Facebook*, o uso de mapas para consultar a posição do utilizador e a consulta da localização *indoor* usando as plantas dos edifícios, pontos inovadores que integram a aplicação.

3.2 Arquitetura legada

Após uma breve introdução do problema é necessário realizar um estudo atento à arquitetura legada do sistema. Perceber de que forma é que se vai garantir a integração das novas plataformas no universo Where@UM e, ao mesmo tempo, manter o sistema simples e modular permitindo alterações no futuro, são necessidades decorrentes do desenvolvimento desta dissertação.

Na Figura 3.1 é possível observar a arquitetura legada do sistema. Na altura em que foi desenvolvida tinha o objetivo de suportar aplicações cliente de dispositivos móveis *Android*.

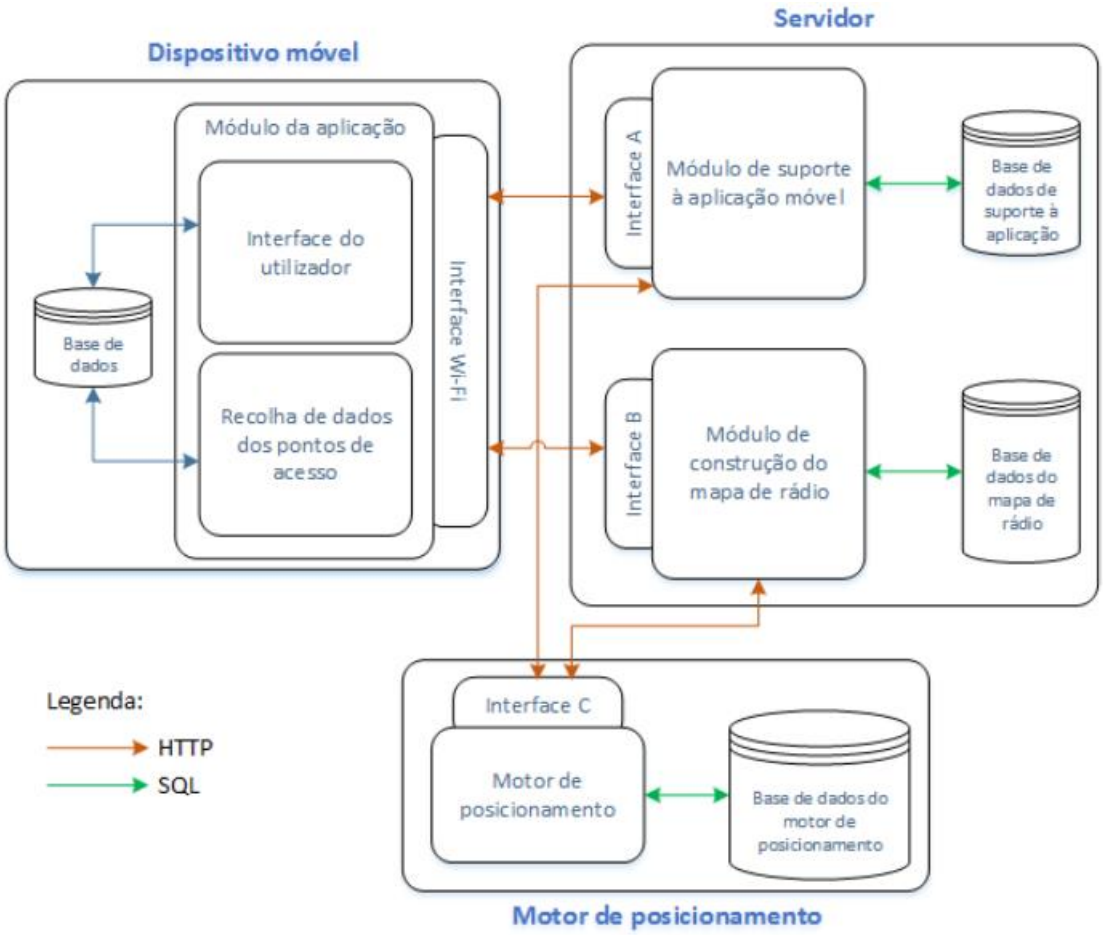

Figura 3.1 - Arquitetura legada do sistema Where@UM [17]

O módulo da aplicação móvel tem a capacidade de recolher *fingerprints* dos APs e de fornecer aos utilizadores as principais funcionalidades. As *fingerprints* são recolhidas automaticamente (por defeito de 5 em 5 min, mas este valor pode ser configurado pelo utilizador) e também manualmente.

O módulo de suporte à aplicação móvel é o responsável por fornecer os serviços para o funcionamento da aplicação como o registo na aplicação, *login*, adicionar e remover amigos, entre outros.

O motor de posicionamento serve-se do mapa de rádio, já construído, para estimar a posição dos dispositivos móveis. Por fim, o módulo de construção do mapa rádio tem a função de receber, processar e armazenar as *fingerprints*.

A principal funcionalidade do sistema é a localização de dispositivos móveis, que para garantir precisão na localização faz uso do algoritmo presente na Figura 3.2. A metodologia implementada permite que o dispositivo do qual são enviadas as *fingerprints* não seja forçosamente um dispositivo móvel, garantindo assim a total integração das novas aplicações cliente com o sistema de posicionamento já implementado no servidor.

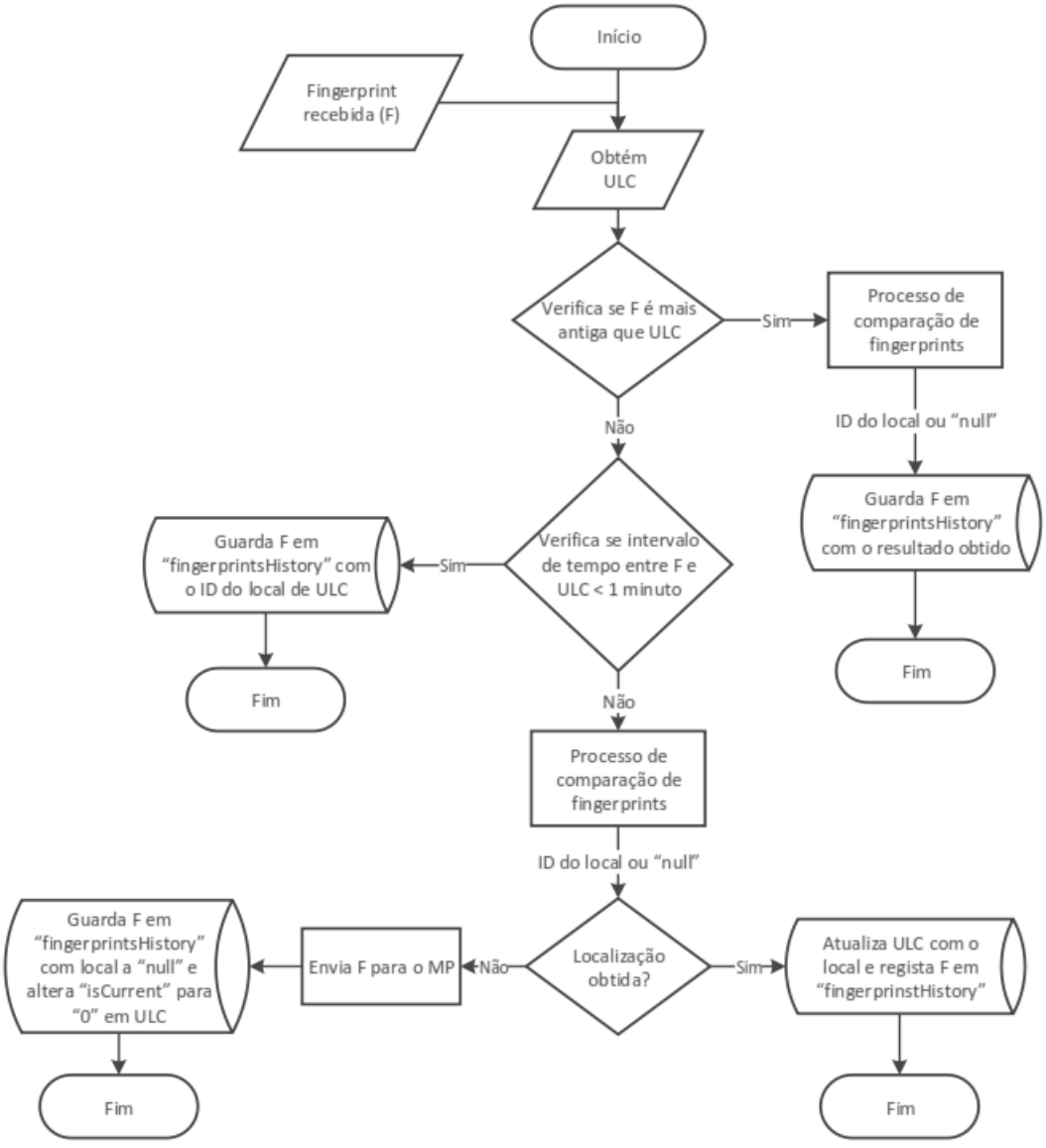

Figura 3.2 - Fluxograma do algoritmo de localização de utilizadores (ULC -user location, F fingerprint, MP motor de posicionamento) [17]

O sistema tem ainda implementado um módulo de integração com o *Foursquare*. Este foi pensado para colmatar a necessidade de obter dados de locais exteriores à Universidade do Minho. Assim, os utilizadores que utilizam a Where@UM têm locais disponíveis fora do campus, como restaurantes, museus, centros comerciais, hospitais, cafés, etc.

Foi utilizada a API do *Foursquare* [38] para obter informações sobre os locais armazenados na base de dados (do *Foursquare*). As coordenadas de GPS recolhidas pelo dispositivo móvel são fornecidas aos servidores que devolvem os locais próximos do utilizador. Deve ser realçado que a função implementada apenas devolve locais numa distância de 150 m em relação à posição atual.

4. DESENHO DO SISTEMA

O capítulo 4 visa elucidar o leitor dos requisitos impostos e das soluções que foram tomadas para resolver o problema. Será descrita em detalhe a arquitetura do sistema, os protocolos de comunicação, o modelo de dados e as integrações que é expectável introduzir no ambiente do sistema Where@UM.

4.1 Requisitos do sistema

O sistema Where@UM, que foi previamente desenhado, tem como alvo os dispositivos *Android* o que implica uma adaptação mútua (novas aplicações cliente e servidores) para manter o bom funcionamento. Considerando que a aplicação *Android* já está em produção, de forma a criar menos resistência à utilização de novas plataformas é expectável que os requisitos a nível de funcionalidades base sejam similares.

Foram consultadas as dissertações [17] e [39] dos programadores, que criaram as versões anteriores do servidor e da aplicação móvel *Android*, para perceber quais os requisitos que foram considerados anteriormente e, simultaneamente, foram ainda adicionados novos requisitos. Assim, para o desenvolvimento da aplicação para computadores pessoais foram apresentados os seguintes requisitos de implementação.

Requisitos Funcionais

- Registo na aplicação usando o *Facebook*;
- Login na aplicação usando o *Facebook;*
- Implementar um mecanismo de segurança que controle as sessões;
- Implementar um novo mecanismo de segurança que funcione nas diversas plataformas e que seja mais seguro que o DES;
- Desenvolvimento de uma solução de *chat* que funcione para computadores pessoais;
- Permitir a correção do nome da posição atual;
- Recolher *fingerprints* através da *interface* Wi-Fi;
- Enviar *fingerprints* periodicamente para o servidor;
- Visualizar a posição do dispositivo num mapa (usando um serviço de mapas como o *Google Maps ou o Bing Maps*), sob a forma de um *PushPin* ou similar;

- Visualizar a posição do dispositivo dentro de um edifício através do uso de uma planta (caso esta seja fornecida);
- Visualizar a posição dos amigos do utilizador no mapa como *overview*;
- Visualizar textualmente a posição dos amigos do utilizador no interior do edifício e se possível na planta do edifício.

Requisitos Não – Funcionais

- Integração da aplicação para computador pessoal no sistema Where@UM;
- Garantir a capacidade e disponibilidade dos serviços oferecidos;
- Garantir a confidencialidade, integridade, autenticidade e disponibilidade dos dados armazenados;
- Garantir a confidencialidade, integridade, autenticidade e disponibilidade dos dados das comunicações efetuadas na aplicação;
- Garantir bom desempenho do sistema;
- Garantir facilidade de utilização.

4.2 Arquitetura do sistema

O sistema Where@UM tem como base o posicionamento de dispositivos em ambientes *indoor* usando o Wi-Fi *fingerprinting*. Tendo como referência o desenho arquitetural explícito em [31] e [32], foram feitas alterações à arquitetura, em termos específicos, foi adicionado um novo servidor externo (Servidor de Plantas) e passou a ser utilizado um serviço de mapas (o *Google Maps* fornece mapas para os dispositivos *Android* e o *Bing Maps* fornece mapas para os computadores pessoais ou dispositivos com *Windows Phone*). Na Figura 4.1 é possível observar a arquitetura simplificada explicitando os módulos que interagem diretamente com os dispositivos.

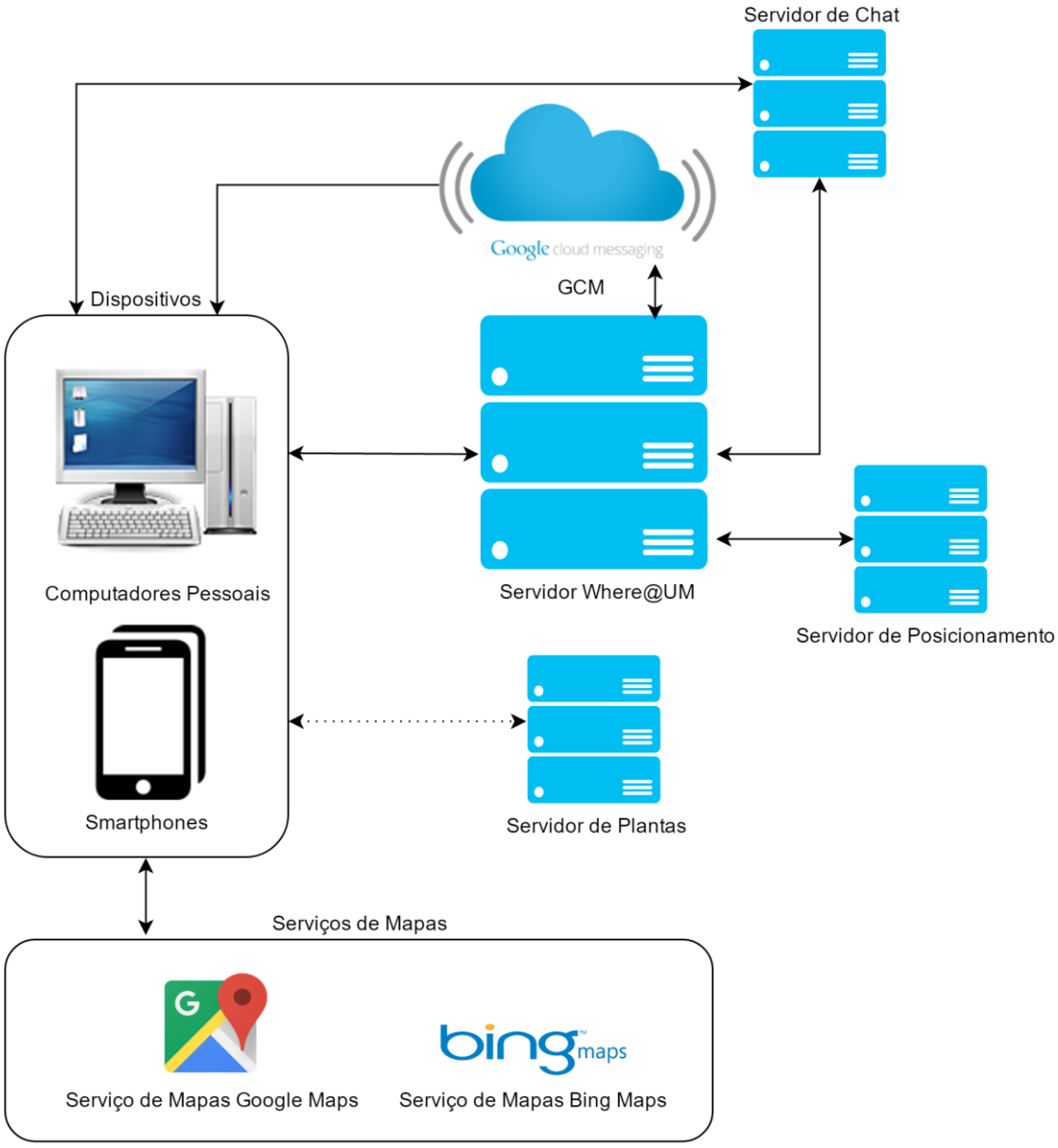

Figura 4.1 - Arquitetura alto nível tendo só em conta as interações diretas com os dispositivos

Utilizando uma metodologia *top down*, na Figura 4.2 é possível observar a arquitetura detalhada do sistema desenhado. Todos os módulos serão abordados detalhadamente nesta secção.

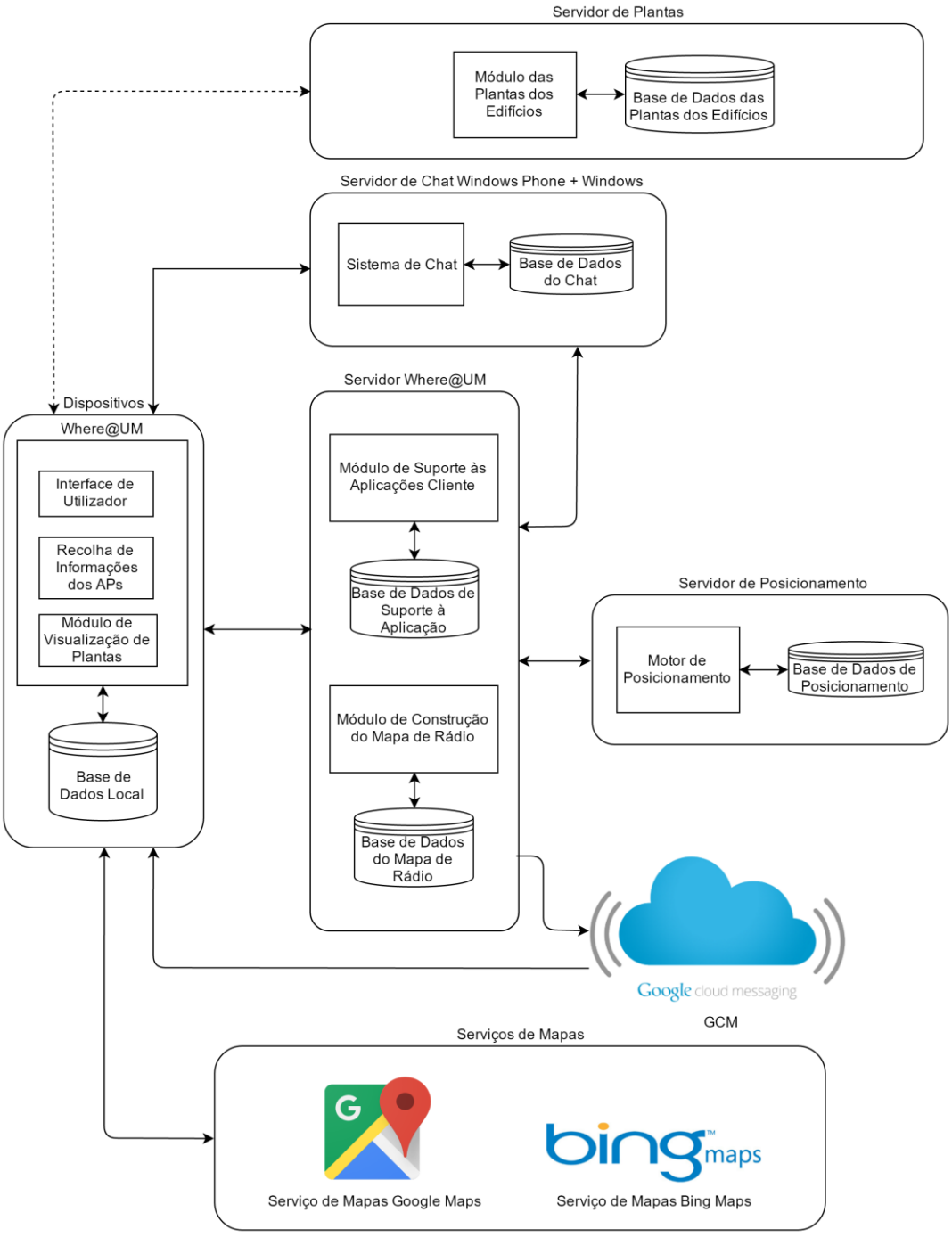

Figura 4.2 - Arquitetura detalhada do sistema Where@UM

4.2.1 Servidores

O sistema Where@UM, sendo já um serviço em produção, tem algumas limitações no que diz respeito a intervenções, nomeadamente, alterações no modelo de dados ou nos protocolos de comunicação entre as aplicações cliente e o servidor. Tais alterações podem perturbar o bom

funcionamento do sistema. Indubitavelmente, todas as intervenções que forem realizadas têm de garantir a disponibilidade da aplicação móvel e a integridade dos dados que o sistema já contém.

O âmbito desta dissertação implica alterações ao ambiente, nomeadamente a inserção de uma nova aplicação cliente, usando uma plataforma que não tinha sido contemplada.

No desenho do servidor central da Where@UM não foram implementadas alterações. Contudo, foram feitas modificações ao nível dos *web services* e das funções de suporte realizadas de forma a permitir a nova abordagem multiplataforma.

Na Figura 4.3 é possível observar o servidor Where@UM e o servidor de posicionamento, sendo explícito os módulos que os constituem.

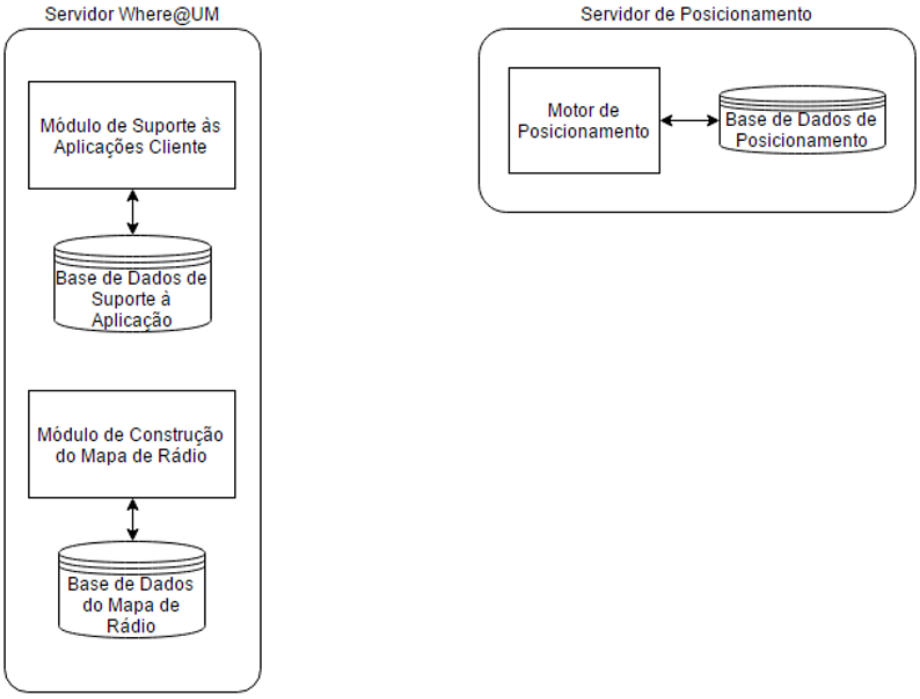

Figura 4.3 - Servidor Where@UM e servidor de posicionamento

O servidor Where@UM e o servidor de posicionamento são constituídos pelos módulos seguintes:

- Módulo de Suporte às Aplicações Cliente: módulo que tem como funcionalidade a disponibilização de serviços que garantam o funcionamento da aplicação, i.e., *login*, registo de utilizadores, definições, etc.;
- Módulo de Construção do Mapa de Rádio: o módulo tem a responsabilidade de receber e processar as *fingerprints* provenientes das diversas aplicações cliente, e de as utilizar na construção de um mapa de rádio colaborativo; este módulo também

contribui para o processo de estimação da localização de cada utilizador, conjuntamente com o Motor de Posicionamento;

- Motor de Posicionamento: usa o mapa de rádio previamente construído e responde a pedidos de localização tendo em conta as informações das *fingerprints* dos utilizadores.

Servidor de chat

De forma a cumprir os requisitos de comunicação foi indispensável desenhar uma solução que permitisse a comunicação de mensagens (*chat*) entre as diferentes aplicações clientes, tendo em consideração que se passou de uma plataforma *(Android)* para uma solução multiplataforma com aplicações clientes em *Android*, em *Windows para desktop* e também em *Windows Phone* (desenvolvimento a decorrer neste momento no âmbito de outra dissertação de mestrado). Anteriormente, o sistema de *chat* era apenas suportado pelo serviço *Google Cloud Messaging* (GCM) da *Google*. Uma vez que o serviço GCM não fornece solução para as plataformas Windows, então verificou-se a necessidade de desenhar uma resposta adequada à resolução deste problema através da implementação de um servidor próprio de mensagens.

Na Figura 4.4 é possível visualizar a arquitetura desenhada e encontra-se descrito todo o processo de troca de mensagens.

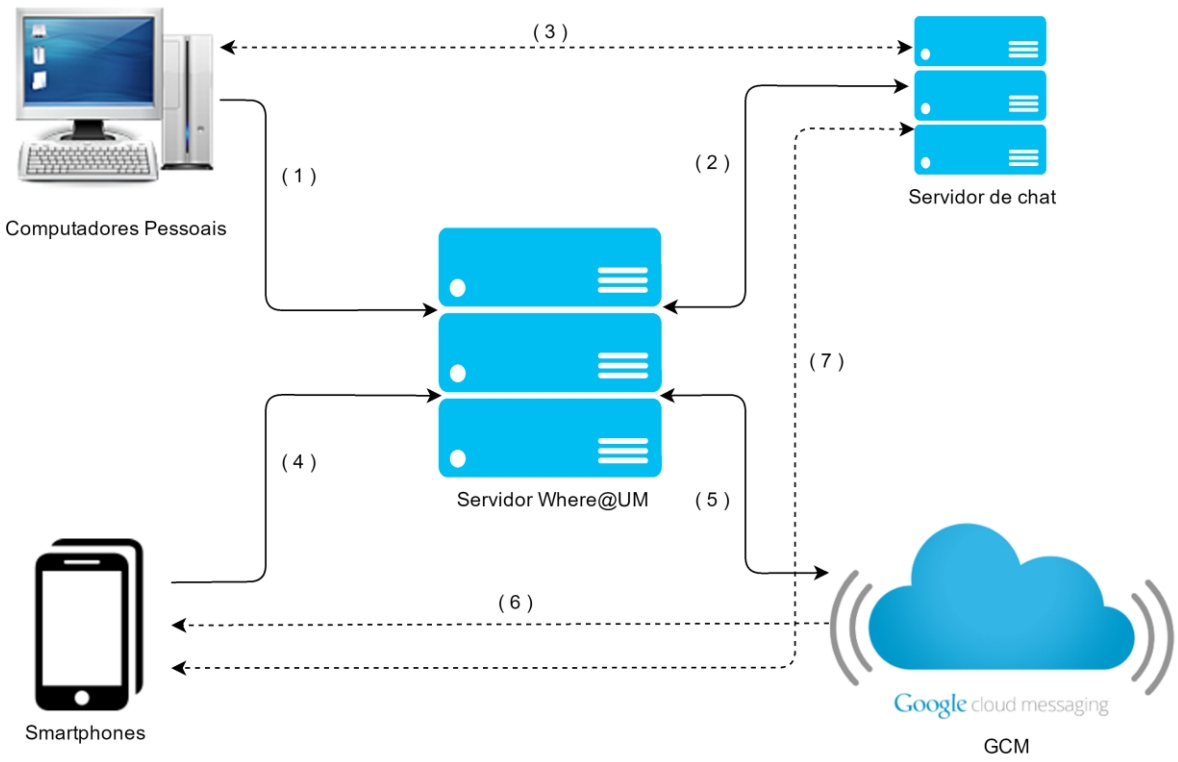

Figura 4.4 - Fluxo de informação do sistema de chat

No início, o cenário contemplado pelo sistema era apenas de dispositivo *Android* para dispositivo *Android* cujo fluxo de um envio de mensagem seria representado por (4)-(5)-(6). Após o desenho da nova arquitetura o sistema passou a contemplar todas as combinações entre as plataformas suportadas (*Android, Windows Desktop* e *Windows Phone*).

De forma a ficar mais claro para o leitor serão apresentados alguns exemplos de comunicações:

a) *Windows Desktop* para *Android*: a mensagem é enviada para o servidor Where@UM (1) que confirma a plataforma do destinatário e, posteriormente, é comunicado para o GCM (5). Como GCM usa um sistema de *Push*, envia automaticamente a mensagem para as aplicações cliente dos destinatários;

b) *Android*/Windows Phone para *Windows Desktop*/*Windows Phone*: a mensagem é enviada para o servidor Where@UM (4) que confirma a plataforma do destinatário e, envia para o servidor de *chat* (2).

c) *Windows Desktop* para *Windows Phone*/Windows Desktop: os remetentes enviam a mensagem para o servidor Where@UM (4), que confirma a plataforma destinatário, e neste caso, envia para o servidor de *chat* (2).

Quer no caso a) quer no b) as aplicações cliente têm de usar um mecanismo *Pull* (ver 5.3.1 para mais detalhes) para obterem as mensagens que lhes foram enviadas.

Servidor de Plantas

Este serviço foi desenvolvido externamente à Where@UM [40], tendo em consideração a necessidade de obter as plantas dos edifícios dentro do campus da Universidade do Minho. O servidor de plantas possui um *webservice* específico para o sistema Where@UM que permite obter o ficheiro da planta em formato XML/OSM, através de um pedido HTTP do tipo GET enunciando os parâmetros: Operador (Universidade do Minho), Área (*i.e.* Campus de Azurém), Edifício (*i.e.* Escola de Engenharia) e Piso (*i.e.* 1).

Assim, sempre que um amigo ou o próprio utilizador entra num edifício ou se desloca para outro piso a aplicação cliente Where@UM faz um pedido ao servidor de plantas confirmando se existe a planta no servidor e, caso exista, integra-a na interface de utilizador.

4.2.2 Aplicação para computadores pessoais

Com o intuito de manter o sistema uniforme, no desenho da aplicação cliente para computadores pessoais teve-se o cuidado de manter a abordagem modular semelhante à da aplicação *Android*. Na Figura 4.5 é possível observar a arquitetura da aplicação cliente desenvolvida no âmbito desta dissertação.

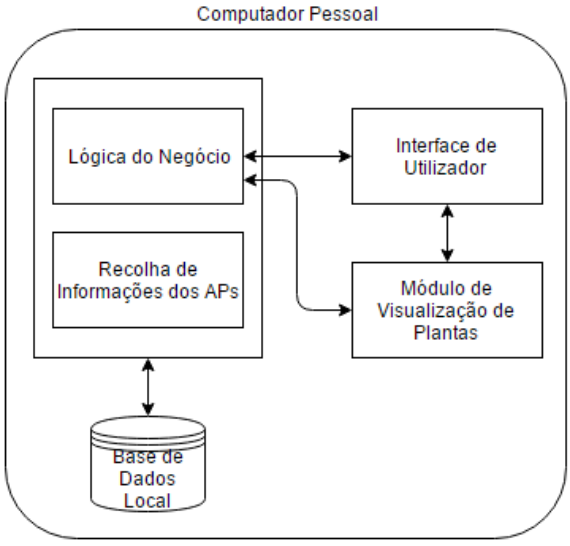

Figura 4.5 - Arquitetura da aplicação para computadores pessoais

Para uma melhor compreensão da necessidade da implementação dos vários módulos da arquitetura, a seguir descreve-se as suas funcionalidades:

- Interface de Utilizador: este módulo apenas contém as *views*, ou seja, a interação com utilizador. Não terá lógica garantindo, assim, total distanciamento da lógica do negócio.
- Lógica do Negócio: este módulo integra as funções que garantem a viabilidade da aplicação, funções base e *core* do sistema como o envio e receção de pedidos para o servidor;
- Recolha de Informações dos APs: este módulo garante a recolha, periódica ou quando pedido (pelo utilizador), de dados sobre os APs visíveis, e o seu envio para o servidor;
- Módulo de Visualização de Plantas: este módulo é responsável pelo processamento das plantas vindas do servidor externo. Faz o pedido de acordo com a localização do utilizador ou dos amigos e processa toda a lógica necessária até chegar aos objetos iterados que serão mostrados pela interface de utilizador;
- Base de Dados Local: um dos requisitos é que a recolha de informação dos APs seja armazenada mesmo em caso de falha de conexão ao servidor. Paralelamente as mensagens trocadas com os amigos devem da mesma maneira ser armazenadas localmente.

4.2.3 Protocolos de comunicação

Sendo a Where@UM uma aplicação distribuída é imprescindível definir um protocolo de comunicação entre o servidor e a aplicação cliente. Geralmente são usados *web services* para realizar comunicações, pois permitem uma boa interoperabilidade com os vários clientes, não tendo grandes limitações a nível de linguagens ou de plataformas.

Os *web services* são serviços que podem estar disponíveis na Internet ou numa *intranet*. As aplicações cliente não têm conhecimento sobre o conteúdo antes de realmente os usarem. Os dados são trocados através do protocolo HTTP e normalmente em linguagens estruturadas [41], que podem ser de dois tipos *Simple Object Access Protocol* (SOAP) ou *Representational State Transfer* (REST), e como são usados nas mais variadas aplicações já existem estudos comparativos de *performance* que podem ser consultados em [42], [43].

O SOAP é um protocolo criado para trocar dados estruturados de forma descentralizada e distribuída [44]. Tipicamente troca mensagens estruturadas em *eXtensible Markup Language* (XML) e o protocolo HTTP da camada aplicacional é utilizado para a transmissão das mensagens.

O REST é uma arquitetura usada para desenvolvimento, os dados e as funcionalidades são denominados por recursos que podem receber um conjunto de operações, definidas no protocolo HTTP:

- **GET** – o cliente pede ao servidor que envie dados;
- **POST** – o cliente envia novos dados ao servidor usando o corpo da mensagem;
- **PUT** – o cliente envia dados ao servidor para atualizar determinado recurso;
- **DELETE** – o cliente pede a remoção de algum recurso;

Este estilo arquitetural limita a arquitetura cliente/servidor ao uso de comunicação *stateless*, ou seja, trocam representações dos recursos usando um protocolo bem definido, tipicamente HTTP.

As aplicações RESTful devem ser simples, leves e rápidas, que se regem por um conjunto de princípios:

- Identificação dos recursos por URIs: fornecendo a capacidade de utilizar o endereçamento global para identificar os recursos e serviços;
- Interface uniforme: como referido anteriormente, existe um conjunto bem definido de operações que podem ser realizadas (GET, POST, PUT, DELETE);
- Mensagens auto descritivas: os recursos não estão acoplados às suas representações e podem ser acedidos em vários formatos como HTML, XML, PDF, etc. São disponibilizados os metadados[1] de cada recurso que servem para controlar a *caching*, detetar erros de transmissão, negociar os formatos de representação e permitem, ainda, fazer autenticação e gestão de acessos;
- Interações sem estado usando *hyperlinks*: todas as interações com um recurso devem ser *stateless*, ou seja, as mensagens devem ser *self-contained*. Existem várias técnicas para mudar o estado como reescrever o URI, *cookies* e formulários escondidos, havendo ainda a possibilidade de embeber o estado nas mensagens de resposta.

[1] Metadados são marcos ou pontos de referência necessários para classificar, organizar e pesquisar. Por exemplo *tags* XML são metadados [52].

4.3 Modelo de dados

Com a integração de novas funcionalidades na aplicação surgiu a necessidade de proceder a algumas alterações ao modelo de dados existente, quer ao nível local, quer ao nível do servidor que serve os diferentes tipos de clientes.

Na aplicação *desktop* existe a necessidade de implementar uma base de dados local para armazenar informações no caso de falha de conetividade com a rede Wi-Fi, e para preservar as mensagens trocadas com os amigos do utilizador. Quando há uma falha de conetividade a aplicação funciona da mesma maneira, mas armazena as *fingerprints* que são enviadas assim que o dispositivo recupera a conexão. As mensagens não são armazenadas no servidor para garantir privacidade das comunicações entre utilizadores.

Na Figura 4.6 é possível observar o modelo de dados implementado localmente em cada aplicação cliente. Tentou-se manter a coerência entre a nova aplicação e a já existente, mas foi necessário acrescentar e modificar tabelas.

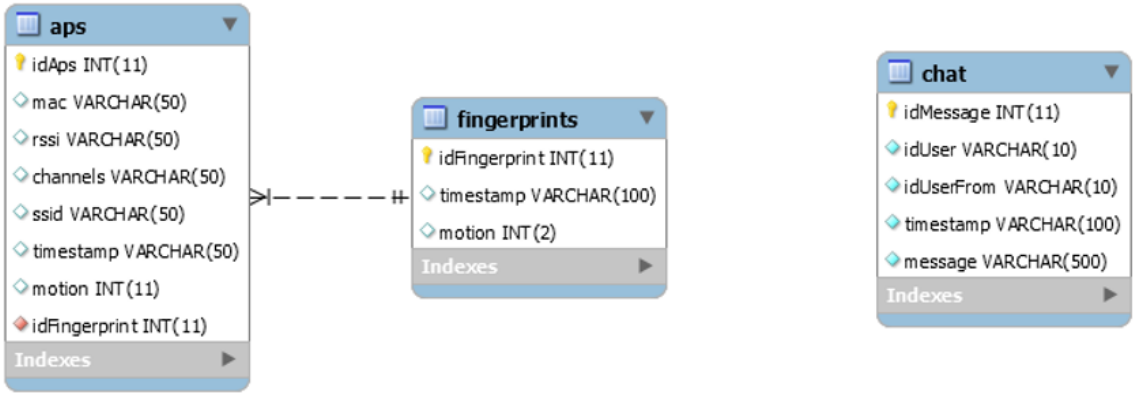

Figura 4.6 - Modelo de dados local, notar a separação das tabelas de recolha dos APs e da tabela de chat

A tabela **APs** sofreu alterações devido ao aumento de dados na recolha de informações dos APs, foram adicionados os campos "ssid" e "channels" onde serão armazenados o SSID e o canal do AP.

A tabela **Chat** foi adicionada e é necessária na aplicação local para armazenar as mensagens do utilizador nos dispositivos. Os campos que a constituem são:

- idMessage – identifica a mensagem;
- idUser – identifica o id do utilizador destinatário da mensagem;
- idUserFrom – identifica o id do utilizador remetente da mensagem;

- timestamp – explicita a data/hora que a mensagem foi enviada;
- message – é a mensagem em si, armazenada localmente.

Modelo de Dados do Servidor

A implementação do *token* de sessão implica que seja adicionada uma nova tabela (no modelo de dados do servidor) apelidada de **sessions**, que armazena os dados das sessões ativas. Os campos que a constituem são os seguintes:

- idSession – identifica a sessão;
- idUser – identifica o utilizador associado à sessão;
- mac – endereço MAC da *interface* de rede contida no dispositivo;
- platform – explicita a plataforma associada à sessão podendo tomar os valores: "windowsdesktop", "windowsphone" e "android";
- timestamp – data/hora em que foi criada a sessão;
- token – valor único para cada sessão, composto por uma *string* de 50 carateres;
- activeToken – como a sessão necessita de ser criada antes do utilizador se autenticar foi criado este campo que pode tomar os valores "Yes" ou "No", indicando se o *token* está ou não ativo. O *token* é ativado quando o utilizador faz o *login* com sucesso.

Para aproveitar todas as potencialidades dos dados recolhidos dos APs, foram adicionados os campos "SSID" e "canal", estes explicitam o nome da rede e o canal em que está configurada nas tabelas **aps** e **apsFingerprintsHistory**.

Tornou-se essencial de na tabela **users** ser adicionado o campo "password2" que contém o *hash* da *password* em MD5. Manteve-se o campo "password", pois é fundamental manter a interoperabilidade de versões da aplicação, sendo que no próximo *login* os utilizadores através de um processo automático (ao nível do servidor) passam a ter a sua *password* armazenada em MD5. Esta alteração foi requisito de implementação e garante que todos os dados de *passwords* passam nos canais de comunicação sob a forma de *hash*.

O modelo de dados já com as alterações introduzidas pode ser analisado na Figura 4.7.

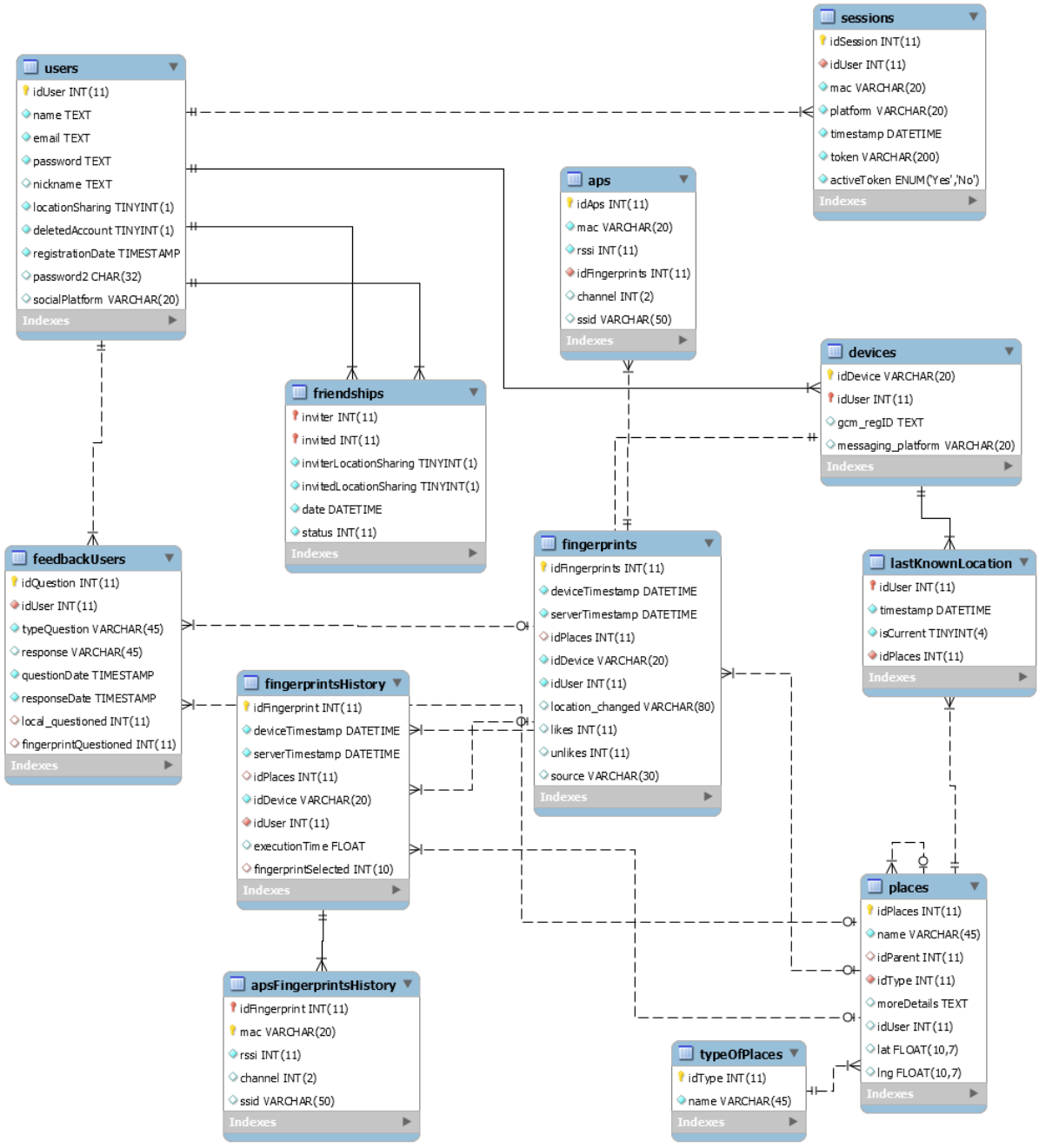

Figura 4.7 - Modelo de dados do servidor Where@UM

Sucintamente, serão apresentadas descrições das tabelas do modelo enunciando, quais os detalhes podem ser consultados em [39].

- **users** – armazena dados dos utilizadores, desde o nome até informações como a autorização de partilha de localização, a data de registo, entre outras, incluindo o registo das *passwords*;

- **friendships** – armazena informação das relações dos utilizadores, contendo o identificador do utilizador que realizou o pedido de amizade, do convidado e do estado da relação (pendente, terminada ou confirmada);
- **devices** – após o registo na aplicação é armazenado nesta tabela o endereço MAC da *interface* de rede do dispositivo (seja ele móvel ou não). Aqui é também guardado o identificador de registo efetuado pelos dispositivos *Android* no serviço GCM;
- **lastKnownLocation** – armazena a última posição conhecida dos utilizadores;
- **fingerprints** – armazena as *fingerprints* enviadas pelos utilizadores;
- **aps** – após ser adicionada uma nova *fingerprint* na tabela **fingerprints** é também adicionado na tabela **aps** os dados adquiridos dos APs pelos dispositivos (móveis ou computadores pessoais) como o endereço MAC, a força de sinal RSSI, o canal, o SSID e o identificador da *fingerprint*;
- **fingerprintsHistory** – armazena todas as *fingerprints* enviadas automaticamente pelos dispositivos;
- **apsFingerprintsHistory** – sempre que é armazenada uma nova *fingerprint* na tabela **fingerprintsHistory** também é armazenada nesta tabela;
- **places** – contém os locais inseridos manualmente pelos utilizadores;
- **typeOfPlaces** – esta tabela específica os tipos de locais existentes na tabela **places**, i.e. universidade, edifício, país, cidade, etc;
- **feedbackUsers** – armazena as opiniões dos utilizadores e informações relativas ao envio de questões por parte do servidor e respostas por parte do utilizador.

Modelo de Dados do Servidor de Chat

Foram criadas um conjunto de tabelas para armazenar algumas informações necessárias no servidor de Chat. Na Figura 4.8 é possível observar o modelo de dados de suporte.

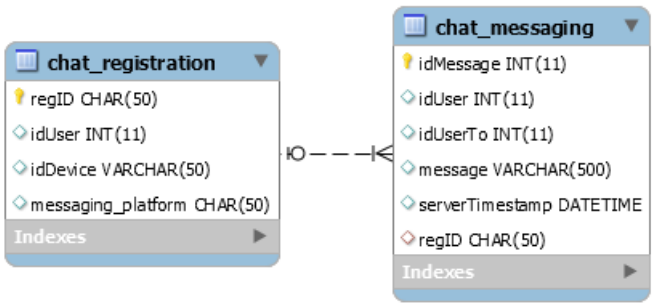

Figura 4.8 - Modelo de dados do servidor de chat

A tabela **chat_registration** armazena os registos de utilizadores que usam o serviço de chat e a tabela **chat_messaging** armazena as mensagens (temporariamente) que foram enviadas ou cujo destinatário seja utilizador *Windows* ou *Windows Phone*. Especificamente a tabela **chat_registration** é constituída por:

- regID – identifica o id de registo de utilizador
- idUser - identifica o utilizador associado ao registo;
- idDevice – endereço da interface de rede do utilizador;
- messaging_platform – plataforma usada pelo utilizador (*windowsdesktop* ou *windowsphone*).

Por sua vez, a tabela **chat_messaging** é constituída por:

- idMessage – identifica a mensagem;
- idUser – identifica o id do utilizador remetente da mensagem;
- idUsertTo – identifica o id do utilizador destinatário da mensage;
- message – mensagem armazenada temporariamente;
- serverTimestamp – explicita o a data/hora que a mensagem foi enviada;
- regID – associa a mensagem ao id de registo do utilizador.

4.4 Bing Maps

Para apresentar a *overview* do utilizador e dos seus amigos no mapa foi escolhido o *Bing Maps* para *Windows Presentation Foundation* (WPF). Foi necessário proceder ao registo na plataforma para obter a credencial de acesso. Posteriormente, foi analisada a API do *Bing Maps*, para perceber qual seria a implementação que melhor cumprisse os requisitos.

Como a aplicação para computadores pessoais utiliza o WPF foi escolhido usar o *Bing Maps WPF Control* [45].

4.5 Integração com redes sociais

Sendo a integração com as redes sociais um ponto fulcral desta aplicação, foi despendido algum tempo a desenhar esta solução. Foi decidido que numa primeira abordagem o sistema de registo e *login* usando esta plataforma seria o melhor para os requisitos e para os objetivos que eram desejáveis.

Para realizar este módulo é imperativo fazer o registo como *developer* no *Facebook* [46] e perceber como fazer um fluxo manual de registo, pois não é fornecido nenhum *Software Development Kit* (SDK) pela empresa.

Na Figura 4.9 é possível observar um diagrama de sequência que mostra o fluxo manual que teve de ser construído para que o registo, usando esta rede social, fosse bem-sucedido.

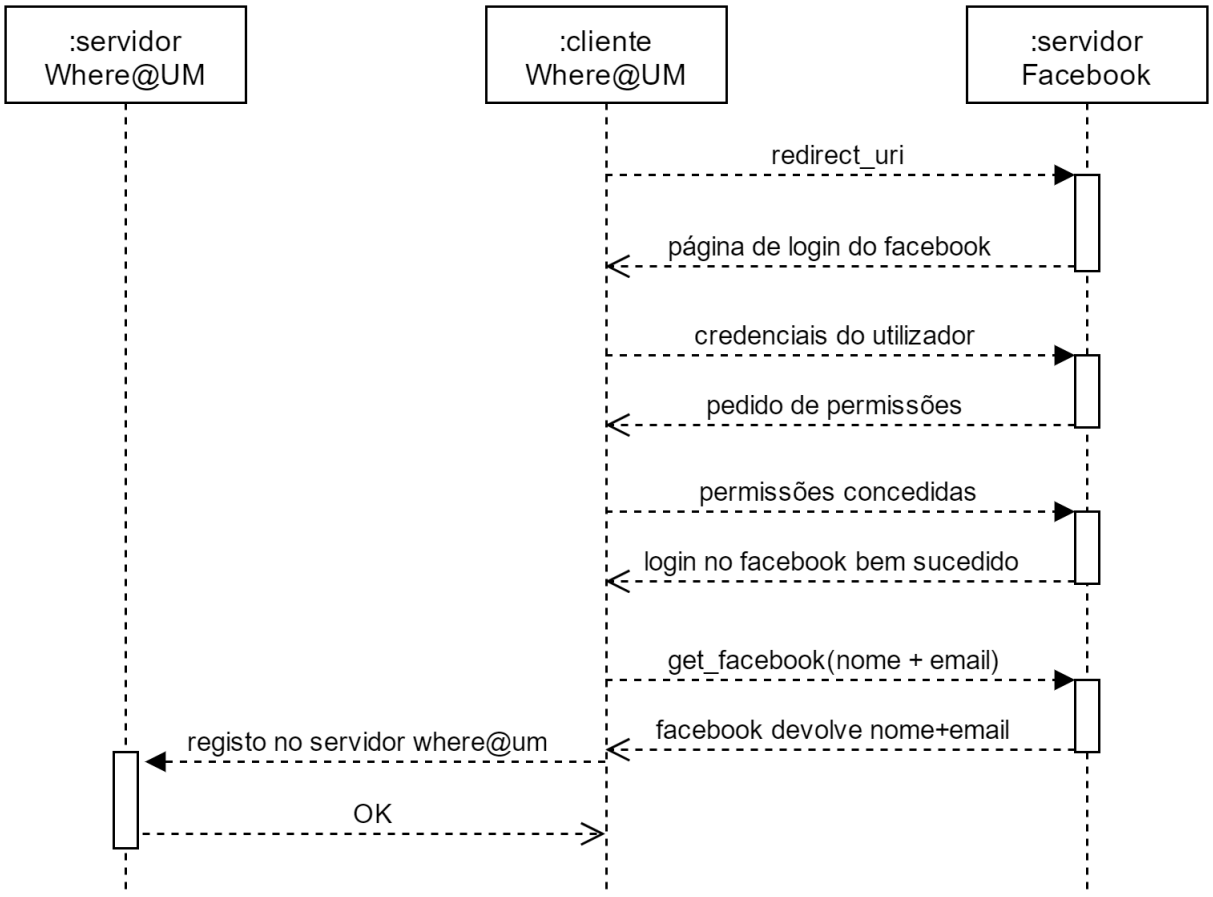

Figura 4.9 - Diagrama de sequência de registo de utilizadores usando o facebook

No início do fluxo a aplicação cliente pede ao *Facebook* que redireciona a página de *login* para a Where@UM. Após o preenchimento do formulário com as credenciais de acesso ao *Facebook* e posterior confirmação é pedido ao utilizador que dê as permissões necessárias para proceder ao *login*, ao receber a confirmação que o *login* foi bem-sucedido a aplicação para computador pessoal vai pedir ao *Facebook* que forneça o nome e o *email* do utilizador para posteriormente realizar o registo no servidor da Where@UM.

5. IMPLEMENTAÇÃO DO SISTEMA

Neste capítulo é abordado em detalhe todo o esforço de implementação. É apresentada a implementação da aplicação para computador pessoal e, posteriormente do servidor.

São ainda apresentadas as alterações efetuadas nos web services, as alterações em termos de segurança dos serviços e dos dados e as alterações que foram realizadas nas *fingerprints*.

5.1 Aplicação para computador pessoal

Esta secção apresenta o padrão de desenvolvimento utilizado, passando por uma descrição detalhada das funcionalidades da aplicação cliente, foi ainda referido a lógica de implementação do módulo de visualização de plantas e a forma como foram recolhidos os dados dos APs.

5.1.1 Padrão de desenvolvimento

Existem três padrões mais genéricos de alto nível que são mundialmente conhecidos: o *Model-View-Controller* (MVC), o *Model-View-Presenter* (MVP) e o *Model-View-ViewModel* (MVVM). Todos os modelos de desenvolvimento são baseados no conceito de *Separated Presentation* [47].

A arquitetura MVVM foi uma norma desenvolvida pela Microsoft com o intuito de ajudar os programadores a implementar as suas soluções. Normalmente é utilizado quando se desenvolve uma aplicação em WPF. As maiores motivações para a usar são:

- Separação de conceitos: separa a lógica da aplicação da interface de utilizador, tornando o código mais reutilizável e evitando problemas de manutenção a longo prazo;
- Padrão natural para o uso da *eXtensible Application Markup Language* (XAML): conexão da *View* ao *View Model* apoiada por boas metodologias de *data binding* e *dependency properties*;
- Fluidez de desenvolvimento das *Views*: permite aos programadores ter uma maior criatividade no desenvolvimento do produto;
- Aumento da facilidade de realizar testes: como a lógica é instanciada independentemente das *Views* torna a realização de testes muito mais fácil.

Na Figura 5.1 é possível observar os relacionamentos entre as três componentes do padrão de desenvolvimento.

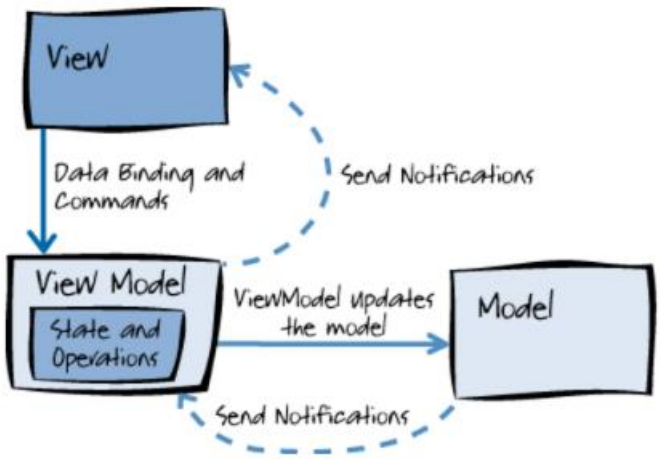

Figura 5.1 - Relação entre os componentes do padrão de desenvolvimento [48]

É possível observar na imagem que existem três componentes: a *View*, o *ViewModel* e o *Model*. Cada um dos módulos tem objetivos e funções diferentes:

- ***View*:** é a *interface* com o utilizador. É desenvolvida em XAML, nela são definidos todos os botões, caixas de texto, tabelas e outros controlos necessários para o bom funcionamento da aplicação;
- ***View-model:*** intermediário entre a *view* e o *model*, implementa a lógica e faz o *data binding* entre a *view e o model*;
- ***Model:*** contém todos os dados que podem ser consumidos e modificados pelo utilizador, define também os objetos necessários para aceder à base de dados, estes objetos são isolados dos outros e também podem ser testados isoladamente.

Benefícios:

- A modularização das componentes permite o trabalho independente e concorrente dos programadores;
- É possível testar as funcionalidades da *view model* sem usar a *view*, permitindo rapidez na implementação;
- A independência dos módulos permite desenvolver novas versões das *views* sem criar necessidade de alterar a lógica.

Para que os *View Models* recebam os pedidos dos utilizadores nas Views, nelas são implementados comandos em XAML que separam a semântica e o objeto que invoca o comando, da lógica que executa

o comando. Portanto, é possível testar a lógica separadamente das *Views*, sendo estas também independentes da lógica [49].

Em alguns casos é imprescindível que haja comunicação entre *View Models*, por exemplo, para pedir que sejam atualizados determinados dados. Para isso, mas também para garantir a independência entre as *View Models* são usadas mensagens. É essencial implementar uma classe extra apelidada de mediador que gere as mensagens todas (comportamento similar a um barramento). Na Figura 5.2 é possível observar um exemplo de um caso de uso [50].

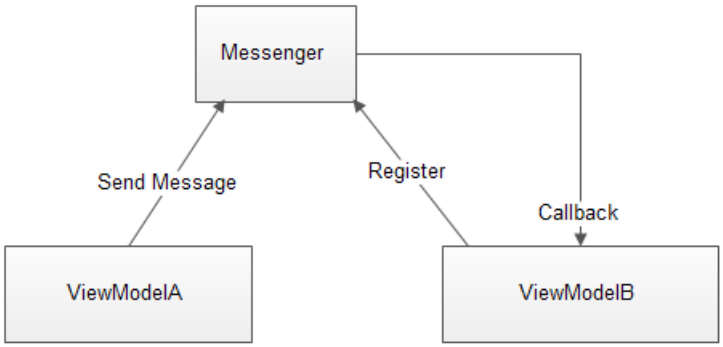

Figura 5.2 - Envio de mensagem do ViewModelA para o ViewModelB [50]

Assim os *View Models* têm acesso a duas principais funções que são: o *Send* e o *Register*. O primeiro deve ser utilizado pela classe que deseja enviar o aviso/mensagem e o segundo pela classe que irá estar à escuta. Usando a Figura 5.2 como exemplo, o **ViewModelA** deseja enviar uma mensagem ao **ViewModelB** que se regista no mediador **Messenger**. Este quando recebe a mensagem do **ViewModelA** vai imediatamente enviar a mensagem para o **ViewModelB** mantendo assim a capacidade de independência entre os *View Models*.

5.1.2 Aplicação cliente

Nesta subsecção serão apresentados os vários ecrãs e funcionalidades que a aplicação cliente oferece aos seus utilizadores. Durante a sua implementação foi sempre um objetivo latente ter especial atenção ao *design* da aplicação *Android* já disponível publicamente para os utilizadores. De forma a retirar o efeito de resistência que pode fazer com que os utilizadores não usem a nova aplicação foram usados *designs* similares e contendo a mesma informação que é mostrada na aplicação *Android*.

Para criar uma interação amiga do utilizador a aplicação foi desenvolvida tendo o cuidado de para qualquer ação não serem dados mais do que 3 cliques.

Login

O ecrã que é apresentado ao utilizador assim que ele inicia a aplicação pode ser observado na Figura 5.3.

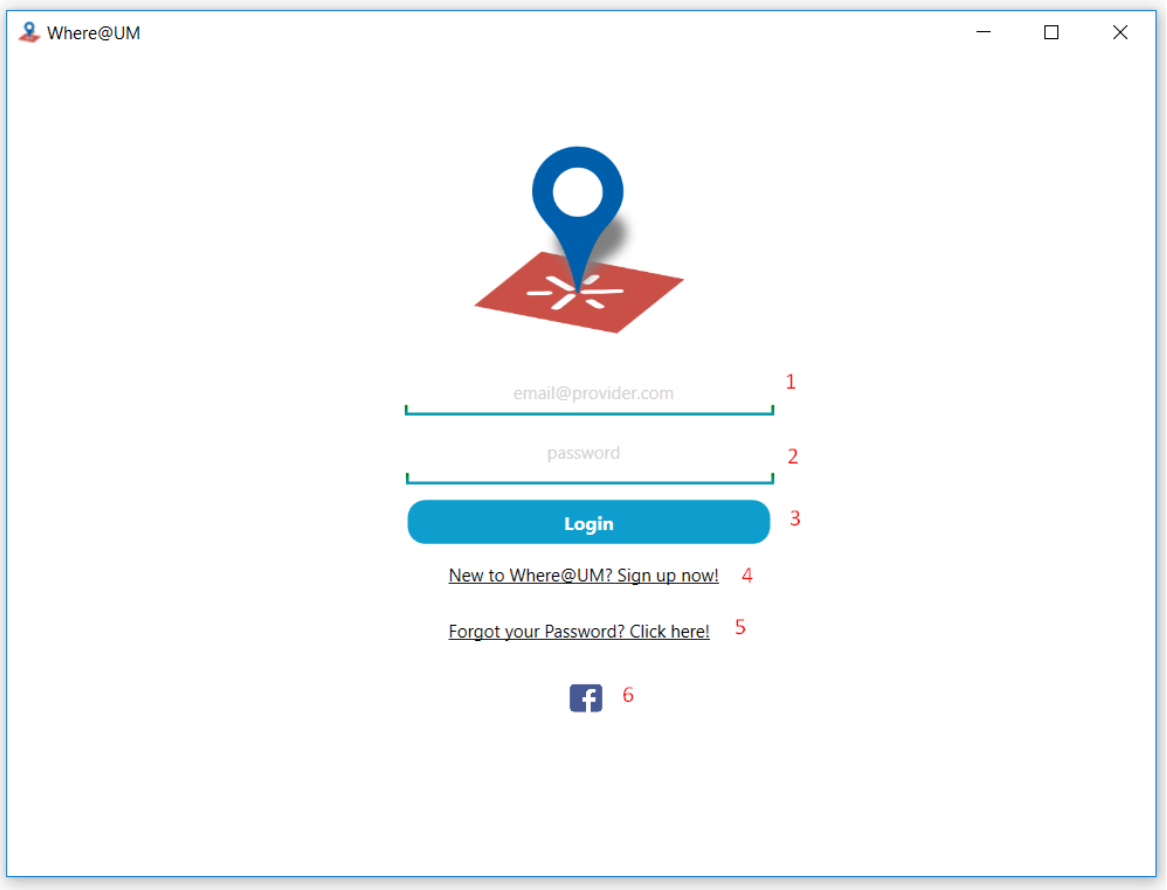

Figura 5.3 - Ecrã de login na aplicação

Permite ao utilizador fazer o *login* na aplicação usando as suas credenciais de acesso (*email* e a *password*) que têm de preencher (em 1 e 2) e posteriormente clicar no botão **Login** (3). Há ainda a possibilidade de efetuar o *login* na aplicação usando o *Facebook* e para isso é indispensável utilizar o botão com o logotipo do *Facebook* (6).

No caso do utilizador ainda não se ter registado na aplicação pode aceder ao ecrã que lhe permite fazer o registo manual clicando em (4) ou usar o *Facebook* (6). No caso de não se recordar da *password* pode, a qualquer momento, recuperá-la utilizando do botão que lhe dá acesso ao ecrã de recuperação (5).

Registo Manual

O ecrã de registo manual na aplicação é apresentado na Figura 5.4. O nome é descritivo da função do mesmo.

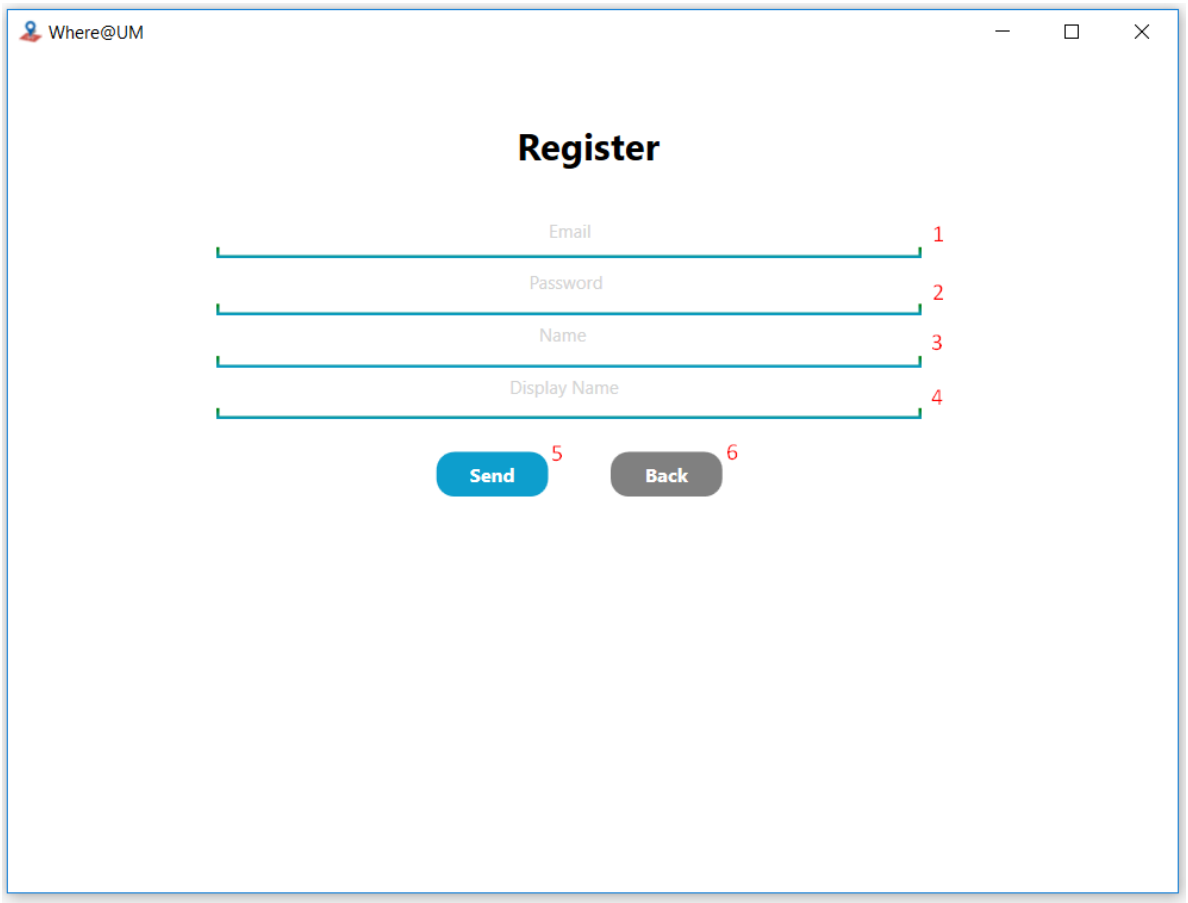

Figura 5.4 - Ecrã de registo manual na aplicação

Para efetuar o registo na aplicação é necessário preencher o *email* (1), a *password* (2), o *name* (3) e o *displayname* (4). Após o preenchimento do formulário, que tem como elementos obrigatórios o *email*, a *password* e o *name* é condição indispensável clicar no botão **Send** (5). Se o botão **Back** for premido a aplicação redireciona para o ecrã anterior que é o de Login.

Login/Registo com o Facebook

Um dos principais requisitos desta aplicação é a integração com as redes sociais. Na secção 4.5 é possível consultar informação detalhada sobre o processo de integração, em especial da sua implementação. Na Figura 5.5 pode ser observado o ecrã de Login no Facebook que após confirmação das credenciais na plataforma, irá proceder ao Login ou Registo no sistema Where@UM.

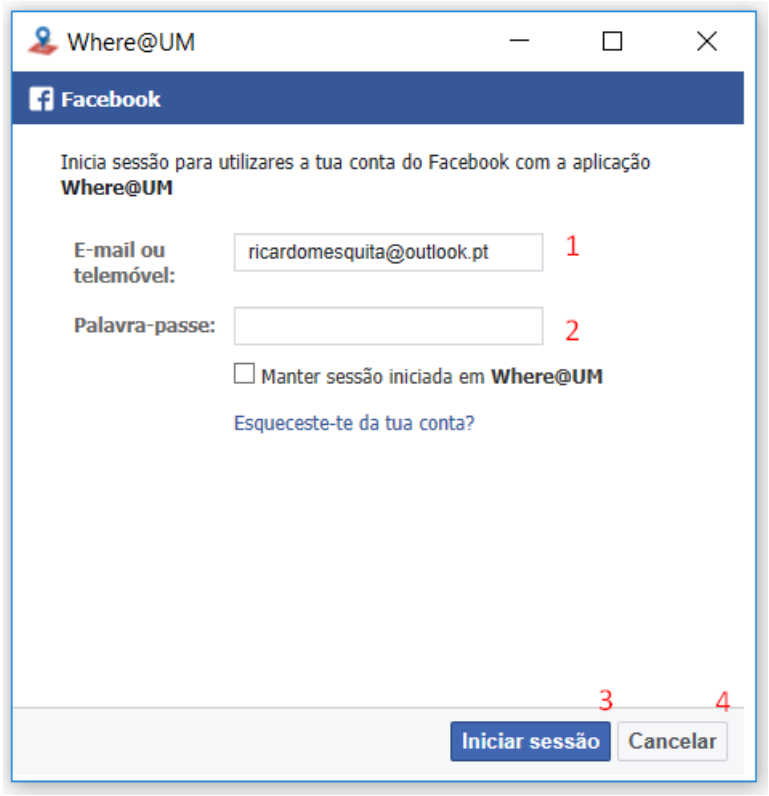

Figura 5.5 - Ecrã de login ou registo de sessão usando o Facebook

A introdução das credenciais para fazer o Login no *Facebook* é um processo conhecido pela maioria dos utilizadores sendo necessário o *email* ou telemóvel (1) e a *password* (2). Após concluído o formulário, o botão **Iniciar sessão** (3) deve ser premido. No caso de o utilizador querer cancelar o seu Login ou Registo pode utilizar o botão **Cancelar** (4) que irá voltar ao ecrã Login, acima referido.

Recuperação da *Password*

Na Figura 5.6 é possível observar o ecrã onde é realizado o pedido de recuperação de *password*.

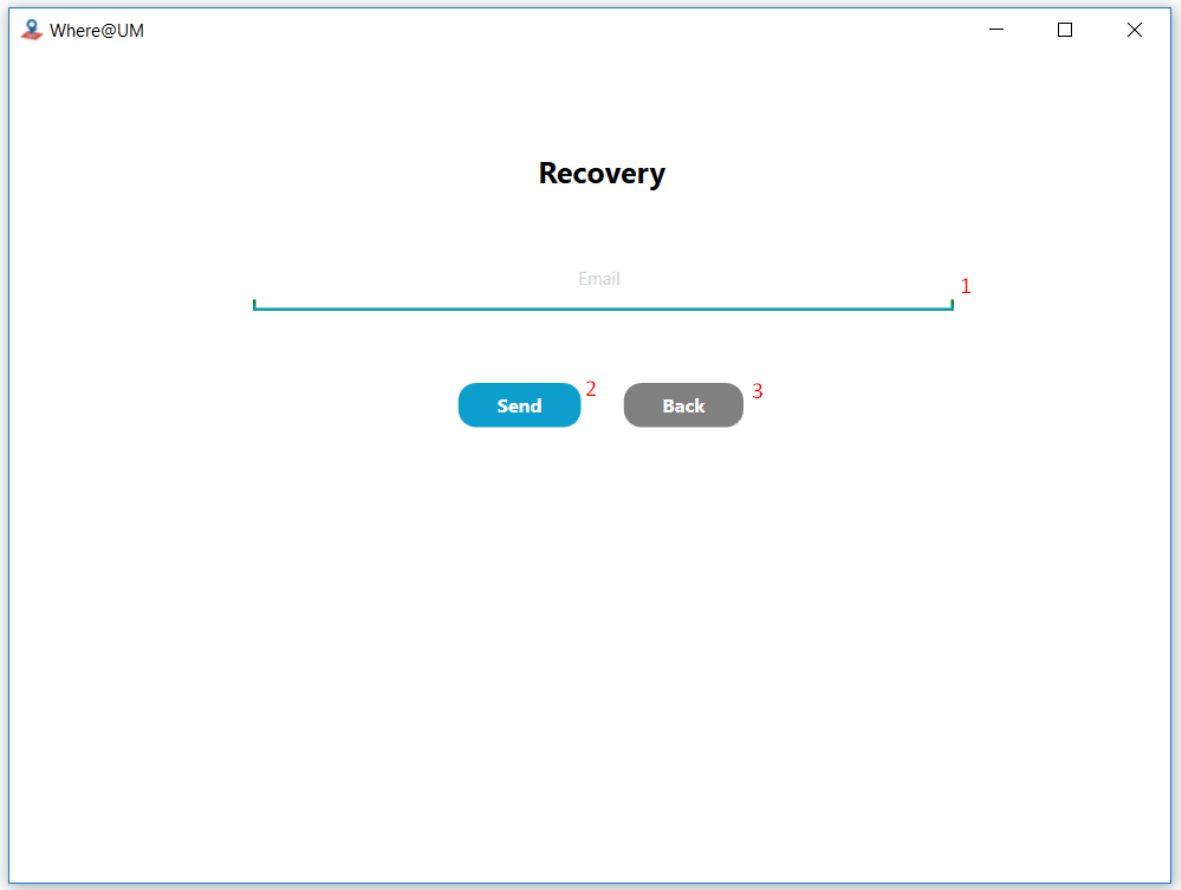

Figura 5.6 - Ecrã de recuperação de password

Para enviar o pedido de recuperação de *password* é necessário preencher o *email* (1) e, posteriormente premir o botão **Send** (2) que envia o pedido para o servidor. No caso de o utilizador querer voltar ao ecrã de Login tem de premir o botão **Back** (3).

Home

O ecrã principal da aplicação, denominado de Home, é apresentado na Figura 5.7. É expectável que este seja o ecrã mais utilizado pelos utilizadores e como tal deve que conter a maioria das funcionalidades, similarmente à aplicação *Android* já existente.

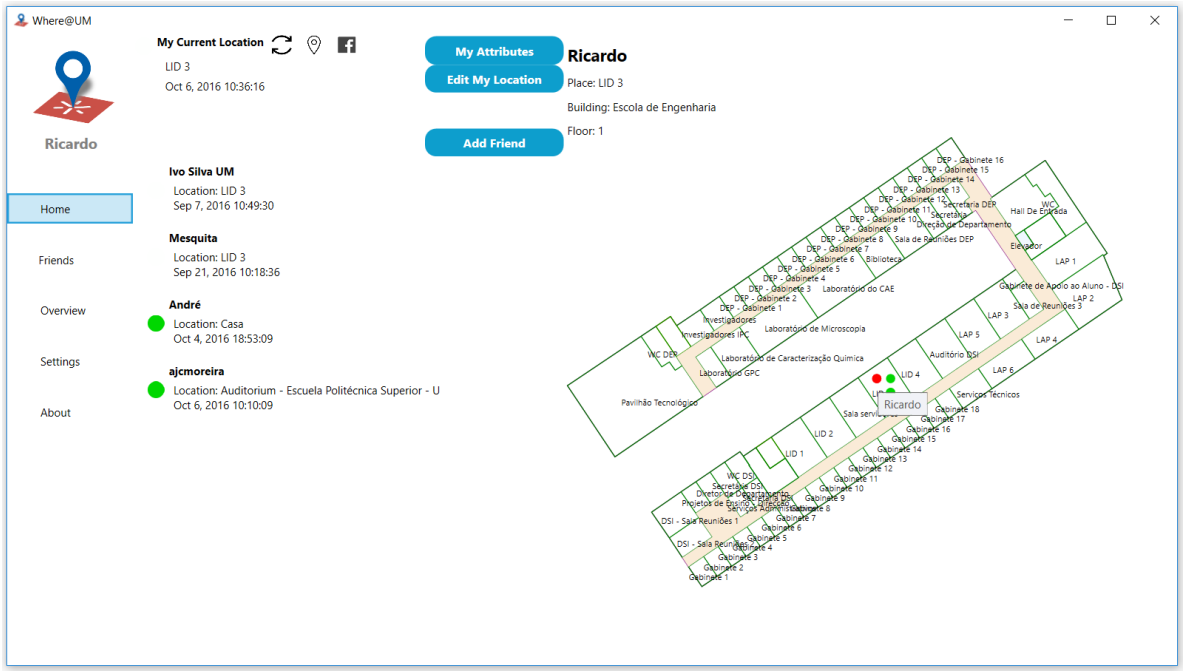

Figura 5.7 - Ecrã home

A Figura 5.8 mostra o ecrã principal da aplicação nos casos em que o utilizador se encontra num edifício do qual o servidor não contenha a respetiva planta. É possível verificar que o desenho da planta desaparece e a descrição detalhada do quarto/local dentro do edifício também.

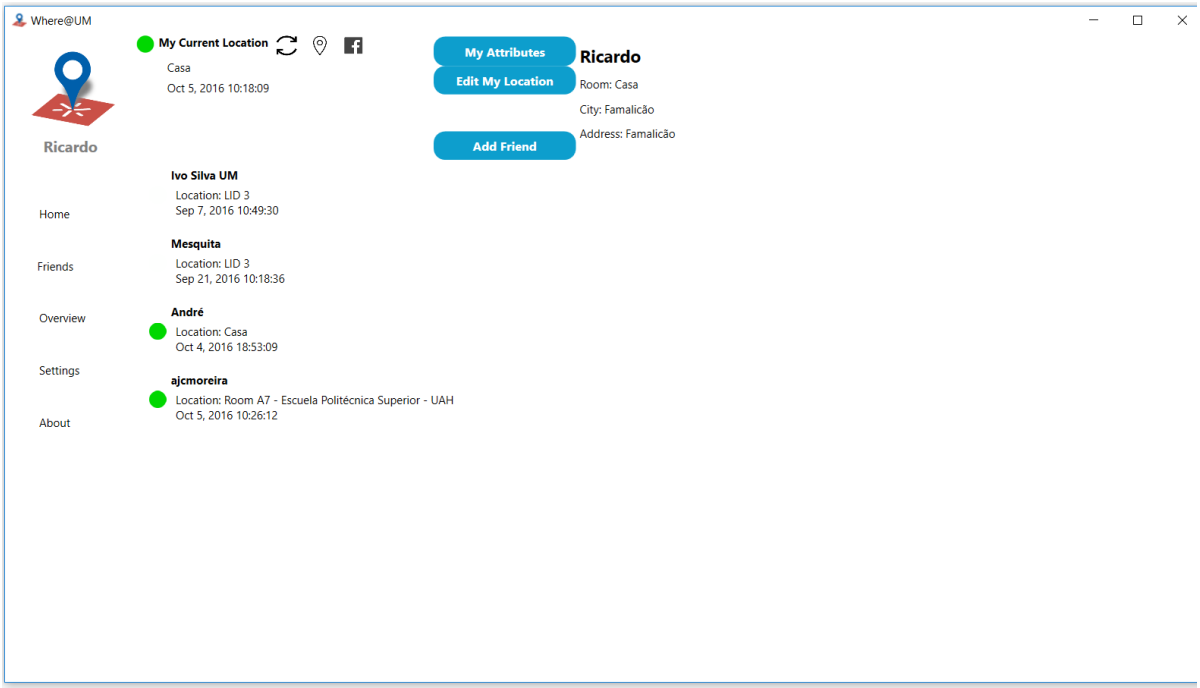

Figura 5.8 - Ecrã home (utilizador num local sem planta associada)

Todas as zonas da aplicação serão descritas em detalhe para uma maior compreensão da necessidade da sua existência e explicação das suas funcionalidades. A primeira zona que pode ser descrita é a zona pessoal denominada de **My Current Location** possível de ser observada na Figura 5.9. Esta zona permite ao utilizador muito rapidamente consultar informações pessoais, atualizar a sua informação de localização e voltar a observar os seus atributos.

Figura 5.9 - Pormenor da zona My Current Location (no ecrã Home)

O círculo verde (1) indica a antiguidade da *fingerprint* deste utilizador (neste caso o próprio, mas a mesma representação também é utilizada na lista de amigos). Na Tabela 2 é possível analisar a tabela de representação da antiguidade das *fingerprints*.

Tabela 2 - Representação da antiguidade das fingerprints

Cor	Antiguidade da Fingerprint
	0 e 15 min
	15 e 30 min
	30 e 60 min
	60 e 120 min
	120 e 240 min
	240 e 360 min
	360 e 1440 min
	Superior a 1440 min

Continuando a detalhar as funcionalidades da Figura 5.9, o botão de **Refresh** (2) permite atualizar a lista de amigos e os pedidos de amizade que o utilizador tenha recebido. O botão **Checkin** (3) além das mesmas funcionalidades presentes no (2) recolhe ainda uma *fingerprint* de forma a atualizar a posição atual do utilizador.

O botão de partilha no *Facebook* (4) permite que seja partilhada a informação de localização do utilizador na rede social.

Em baixo, nas *labels* (5) e (6) consegue-se observar a localização atual do utilizador e a data a que foi recolhida a última *fingerprint*, naquela localização. Caso esta esteja errada o utilizador pode sempre atualizar a sua localização premindo o botão **Edit My Location** (8). Quando o utilizador carrega num amigo para verificar os seus detalhes ou comunicar com ele e quer voltar aos seus detalhes basta fazer uso do botão **My Attributes** (7) para voltar a ver os seus detalhes, como é mostrado na Figura 5.10.

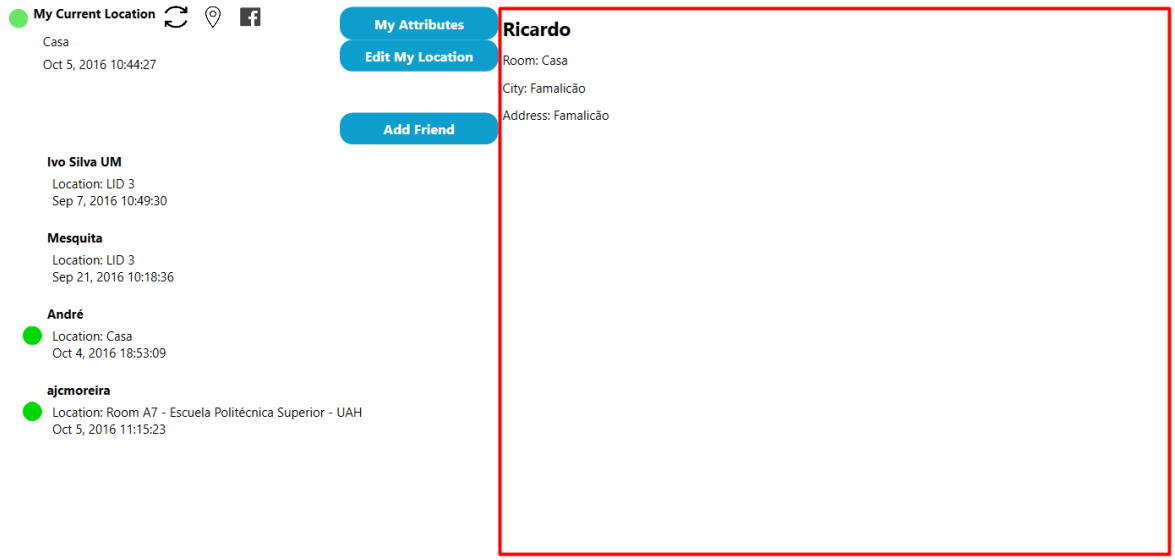

Figura 5.10 - Detalhes pessoais de um local sem Planta de Edifício (fora da universidade)

No caso da Figura 5.10, o utilizador encontra-se fora de um edifício do qual existe planta disponível logo esta não é apresentada. São, no entanto, exibidos os detalhes, em formato textual, da localização do utilizador: o *Room*, a *City* e o *Address*.

Para interações ou consultas de detalhes de um utilizador é necessário clicar nele. Na Figura 5.11 pode observar-se como fica um amigo selecionado.

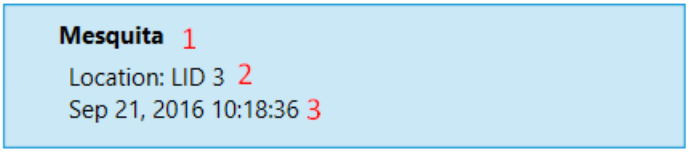

Figura 5.11 - Amigo seleccionado na lista de amigos

Na lista de amigos é possível visualmente consultar à primeira vista alguns detalhes: em (1) consegue-se ver o *nickname* e caso o amigo não tenha definido um, será mostrado o seu nome. Em (2) pode-se observar a localização com melhor resolução, ou seja, neste caso sabe-se a divisão que o utilizador estava, no caso de esta não ser disponibilizada pode ser fornecido o andar, etc. Por fim, é ainda mostrado a antiguidade da *fingerprint* do utilizador, basicamente é a última localização conhecida pelo sistema.

Na Figura 5.12 é possível observar o que mudou desde a Figura 5.7 (imagem do ecrã Home) após o clique no amigo desejado.

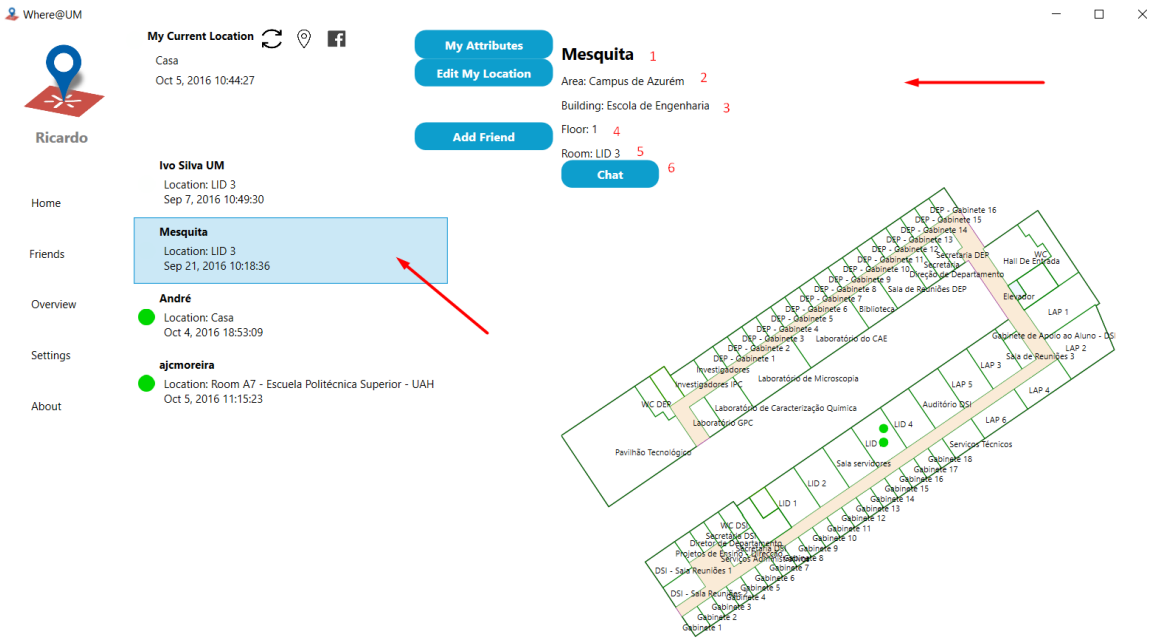

Figura 5.12 - Ecrã home após clique num amigo

São mostrados detalhes do amigo que foi clicado, no caso da Figura 5.12 o amigo estava num local que existia uma planta do edifício, então existe uma descrição pormenorizada da sua localização. Em (1) é mostrado o nome, segue-se a **Area** (2), o **Building** (3), o **Floor** (4) e o **Room** (5). Existe ainda

um botão **Chat** (6) que permite abrir o *chat* do qual é possível enviar mensagens para um destinatário, também são consultadas as mensagens previamente trocadas com o utilizador clicado.

Add Friends

Dentro do ecrã Home existe ainda a possibilidade de adicionar um novo amigo (através do email). Na Figura 5.13 é possível observar a opção de adicionar amigos.

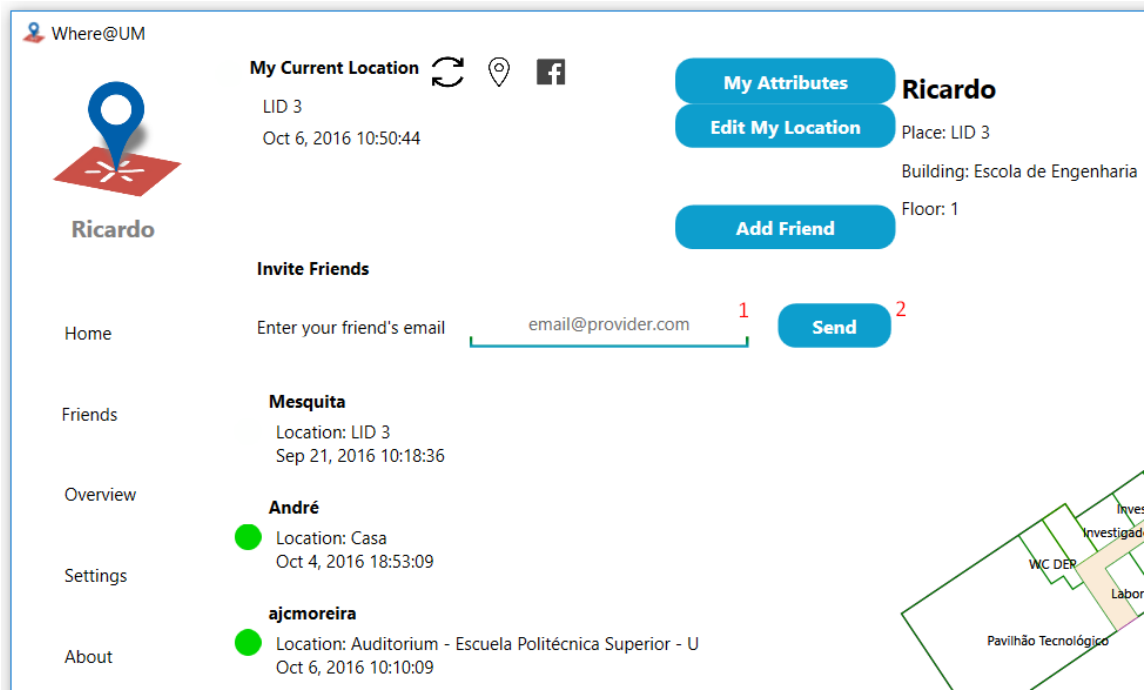

Figura 5.13 - Ecra home após clique no botão Add Friend

Para que a opção de adicionar amigos seja visível é necessário clicar no botão **Add Friend**, que quando clicado mostra o painel de Invite Friends que estava *hidden*.

Em (1) pode ser introduzido o *email* do amigo que é utilizador da Where@UM e, caso não seja recebe um convite para se registar na aplicação. O convite é enviado quando o utilizador clica no botão **Send** (2).

Edit Location

O ecrã Edit My Location só pode ser acedido a partir do Home. A sua principal funcionalidade é permitir ao utilizador que atualize a sua localização. Na Figura 5.14 pode ser visualizado o ecrã que permite a alteração manual da localização atual.

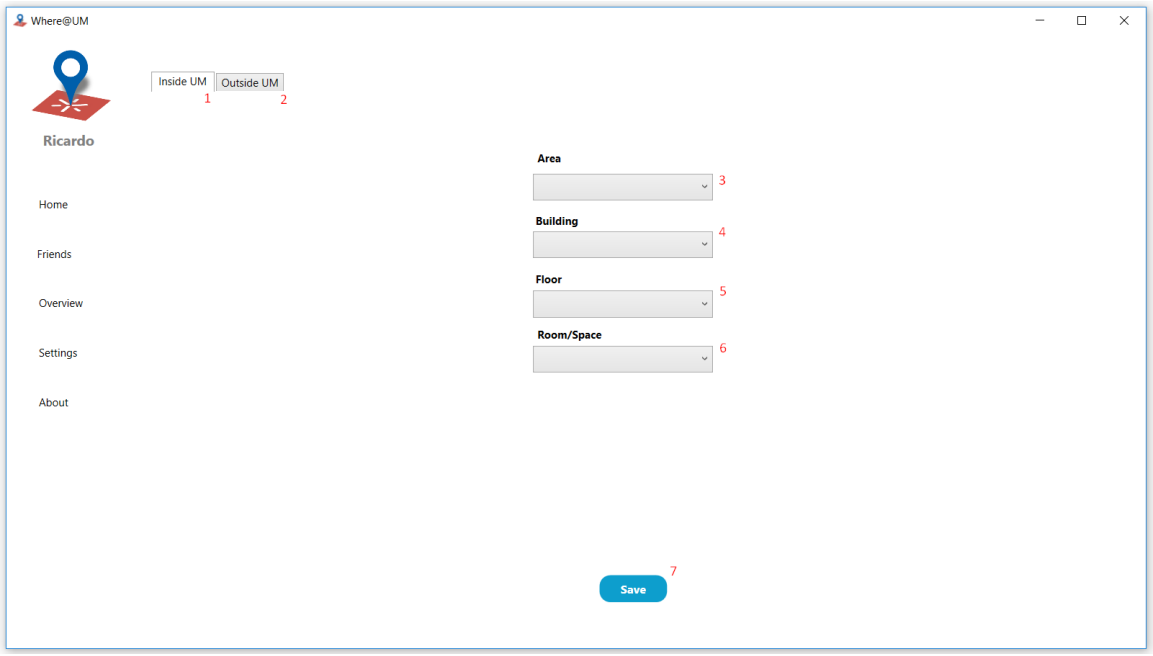

Figura 5.14 - Ecrã home após clicar no botão Edit My Location

Por norma, o ecrã irá ser iniciado mostrando a possibilidade de alterar a localização dentro da Universidade do Minho (1) mas caso o utilizador se encontre noutro local pode aceder ao separador **Outside UM** (2) e alterar a sua localização.

Para modificar a sua localização tem de usar as *combo boxes* (3, 4, 5 e 6) e seguir as sugestões facultadas. Na Figura 5.15 pode-se observar as sugestões para a "Area".

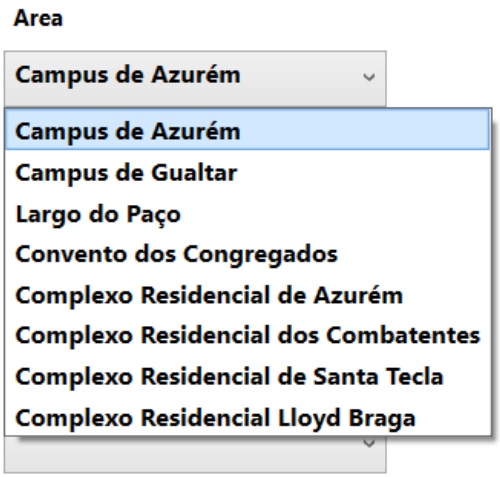

Figura 5.15 - Opções de área na vista do Edit My Location (Inside UM)

Usando uma hierarquia em árvore, para cada "Area" existem vários edifícios que podem pertencer à que foi selecionada, sendo estes também sugeridos ao utilizador. Quando um edifício é selecionado existe um conjunto de pisos que cada edifício tem, que também são sugeridos ao utilizador e, por último, por cada piso são também sugeridas divisões (rooms). Esta implementação permite manter a coerência dos dados, impedindo ambiguidades na denominação de locais dentro da Universidade do Minho.

Após escolhida a localização, deve ser premido o botão **Save** (7), imediatamente, surge um aviso de confirmação similar ao que pode ser observado na Figura 5.16. No painel pode ser observado o local introduzido manualmente e o utilizador pode fazer uma última confirmação da localização que introduziu.

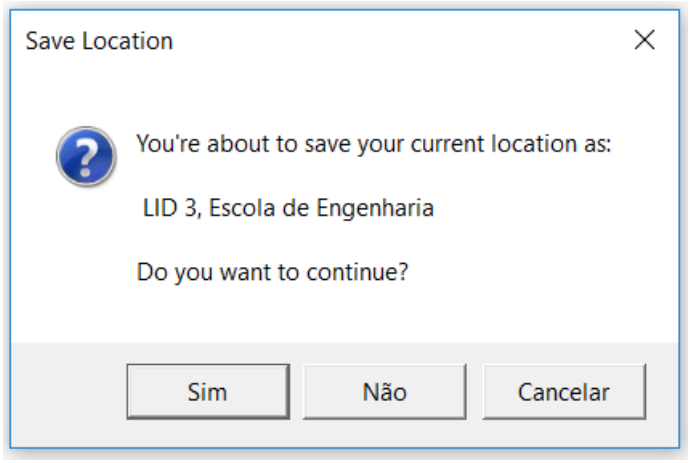

Figura 5.16 - Painel de confirmação de escolha antes de enviar a nova localização

Chat

Após escolher um dos amigos para consultar os seus detalhes pessoais é possível aceder ao chat com o amigo. Na Figura 5.17 é possível observar um exemplo de uma conversa entre o utilizador e um amigo.

Figura 5.17 - Ecrã home com um utilizador seleccionado e em conversação

A conversa do utilizador, neste caso, é realizada com o amigo "Mesquita". No botão **Map** (1) é possível voltar à vista da planta do edifício. Como enunciado, na subsecção 5.3.1, o servidor de *chat* está a funcionar usando a metodologia *Pull*, o botão *refresh* (2) serve para atualizar as mensagens trocadas.

Em (3) e (4) podemos observar dois exemplos de mensagens trocadas, a negrito é exposta a mensagem e em baixo é enunciado se a mensagem foi enviada ou recebida, além da data/hora da ação. Na *textbox* (5) é o local para escrever a mensagens que são enviadas quando o utilizador clica no botão **Send** (6).

Friends

No ecrã apresentado na Figura 5.18 é possível consultar os pedidos recebidos e enviados de amizade e os amigos que fazem parte da lista de amigos do utilizador. É também possível apagar ou recusar pedidos e amigos.

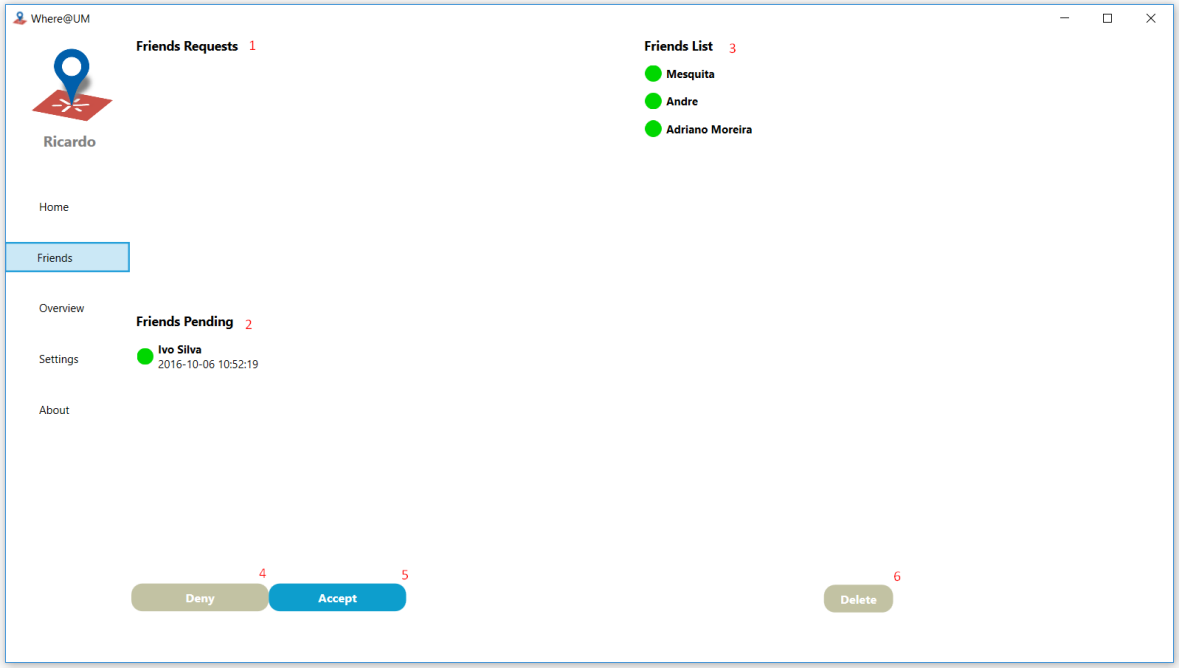

Figura 5.18 - Ecrã friends

Na lista **Friends Requests** (1) são listados os pedidos de amizade realizados pelo utilizador. Na lista **Friends Pending** (2) são apresentados os pedidos de amizade recebidos pelo utilizador. Nas duas listas podem ser aceites ou recusados pedidos, nos botões **Deny** (4) e **Accept** (5), respetivamente.

Na **Friends List** (3) são enumerados os utilizadores que fazem parte da lista de amigos do utilizador, estes podem ser apagados sendo para isso clicado o botão **Delete** (6)

Overview

O ecrã *Overview*, representado na Figura 5.19, permite ter uma vista geral do posicionamento do utilizador e dos seus amigos.

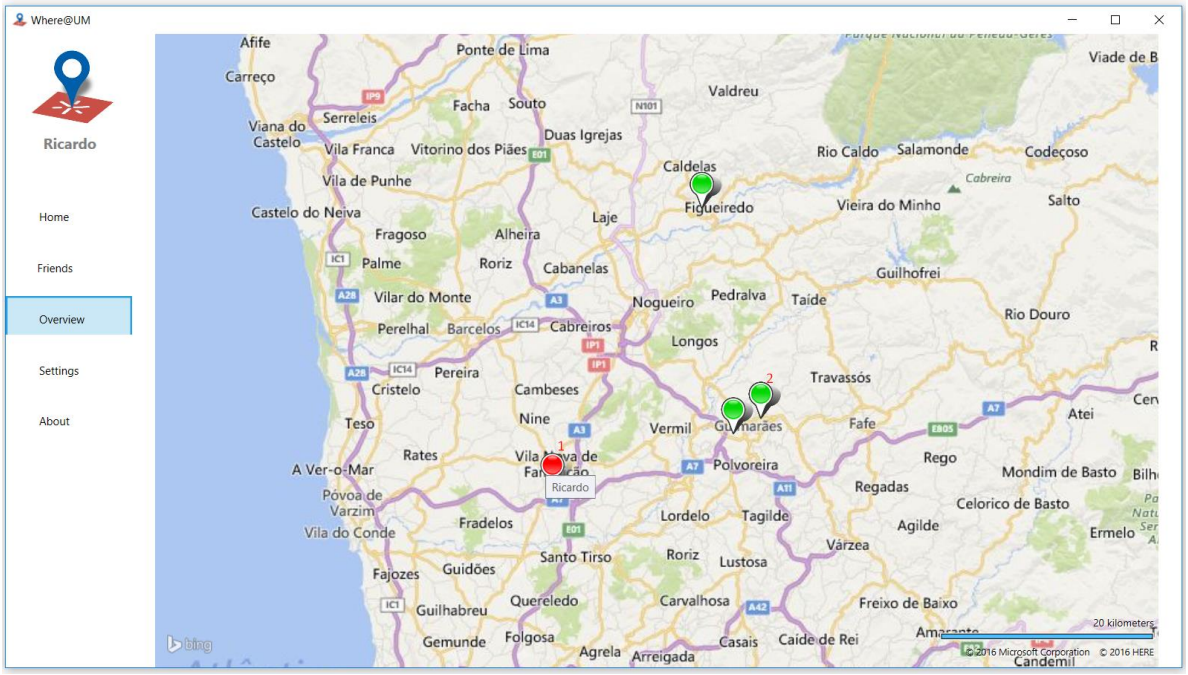

Figura 5.19 - Ecrã da overview

O utilizador é representado pelo *pin* vermelho (1) e o seu nome aparece quando o ponteiro do rato passa em cima dos *pins*. Da mesma forma os seus amigos são representados pelos *pins* a verde e os seus nomes aparecem quando o comportamento acima enunciado é realizado. A vista do mapa é perfeitamente controlável através de ações de arrastar o mapa ou de aumento/diminuição do *zoom* usando o *scroll*.

Settings

O ecrã de Settings, apresentado na Figura 5.20, permite ao utilizador consultar e alterar algumas das suas definições.

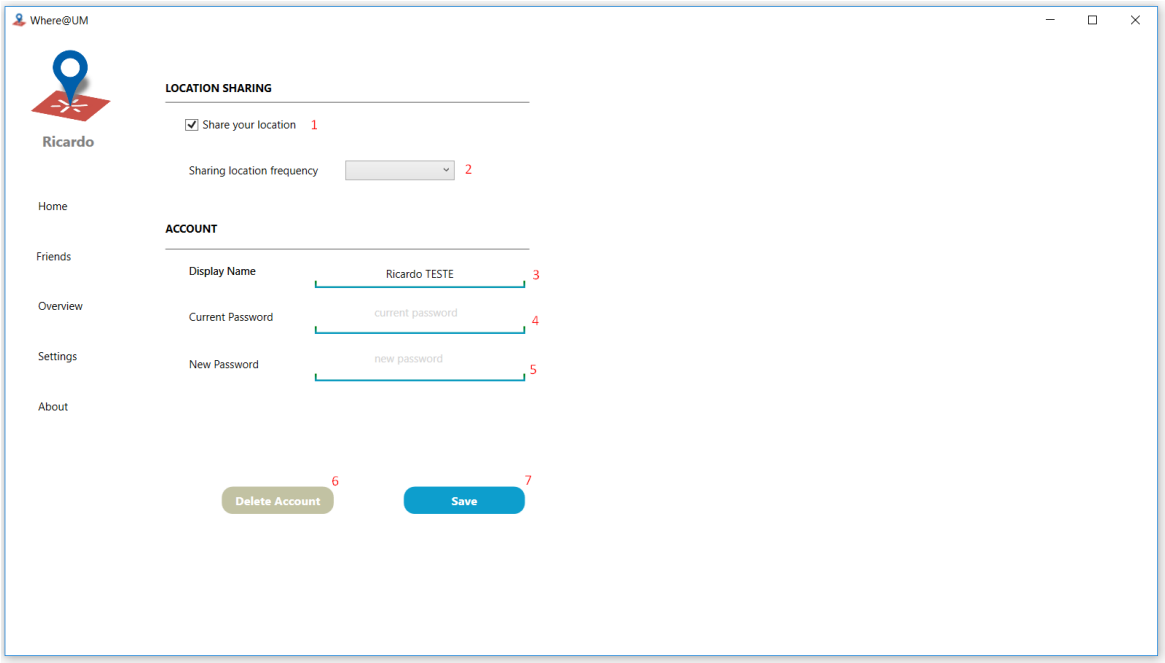

Figura 5.20 - Ecrã de settings

Umas das principais definições é a autorização de partilha de localização (1). O utilizador pode ainda optar pela frequência da partilha de localização (2), que por defeito é 5 min.

É ainda possível alterar o *display name* (3) e alterar a *password* (4 e 5). Todas as alterações são guardadas quando o botão **Save** (7) é premido. O botão **Delete Account** (6) tem a função de apagar a conta do utilizador.

About

O ecrã About contém informações sobre quem desenvolveu a aplicação e pode ser visualizado na Figura 5.21.

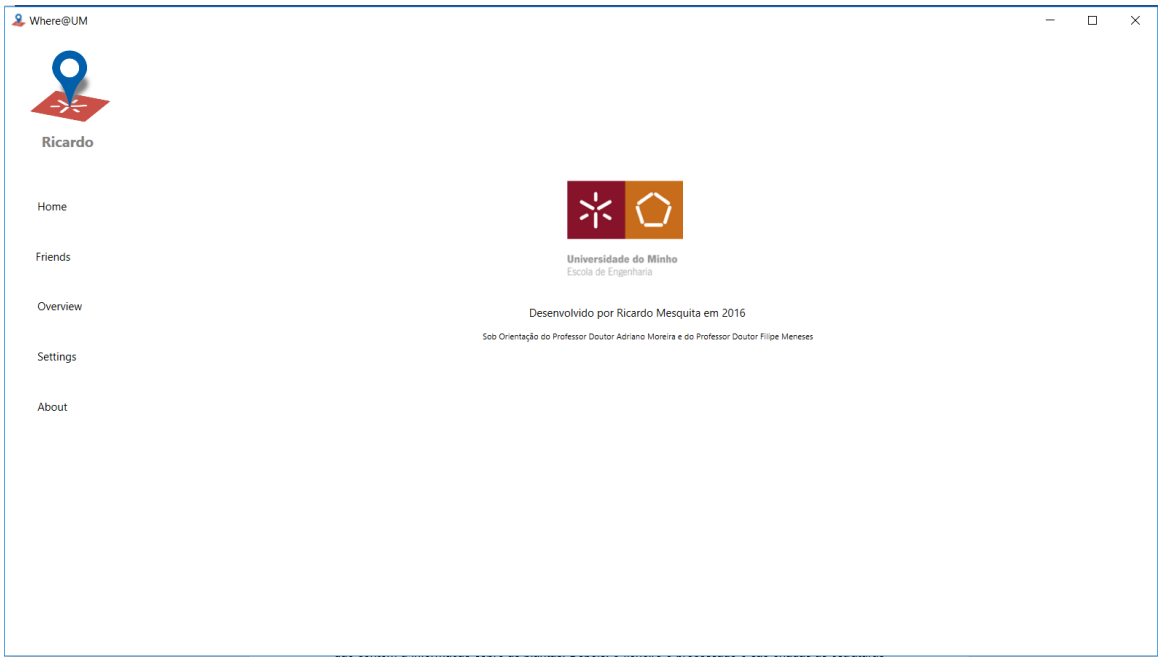

Figura 5.21 - Ecrã About

5.1.3 Implementação do módulo de visualização de plantas

Em resposta a um pedido efetuado ao servidor de plantas, a aplicação cliente recebe um ficheiro XML que contém a informação sobre a planta de um determinado edifício. O ficheiro é processado pela aplicação cliente e são criadas as estruturas de dados que descrevem cada divisão do piso (salas, áreas, corredores, escadas, etc..). Cada ponto é convertido do sistema de coordenadas dos mapas (latitude e longitude) para o sistema de coordenadas da aplicação (x, y).

No *rendering* da planta, por uma questão de coerência com a aplicação *Android*, foram utilizadas as seguintes cores:

- Roxo – estrutura externa do edifício;
- Verde – salas/quartos;
- Laranja – escadas;
- Azul – portas;
- Rosa – elevadores;
- Castanho – paredes;
- Sem cores – corredores.

Foram implementadas algumas funções auxiliares para permitirem que o desenho fosse realizado com sucesso e que a resolução da planta fosse adequada à aplicação para computadores pessoais.

- `Public int getScaleFactor()` – usando as dimensões do ecrã do dispositivo, é calculado um fator de escala com o objetivo de ajustar a planta ao tamanho requerido;
- `Public string middlePoint(List<Point>)` – recebendo a lista de pontos que constituem uma divisão esta função ajuda a descobrir o ponto médio, no qual irá ser escrito o texto que descreve a divisão (exemplo: "LID3", "Gabinete 1", etc.);
- `Private double DegreeToRadian(double angle)` – converte um ângulo de graus para radianos;
- `Private double RadianToDegree(double angle)` – converte um ângulo de radianos para graus;

5.1.4 Recolha de dados dos APs

As aplicações cliente têm um módulo responsável por fazer a recolha dos dados provenientes dos APs de forma automática e periódica ou quando requisitado pelo utilizador. É utilizada a *Network Shell* (*netsh*) para obter informações dos pontos de acesso que estão visíveis para o computador pessoal. O comando o utilizado é **wlan show networks mode=bssid** que devolve valores como mostrados na Figura 5.22.

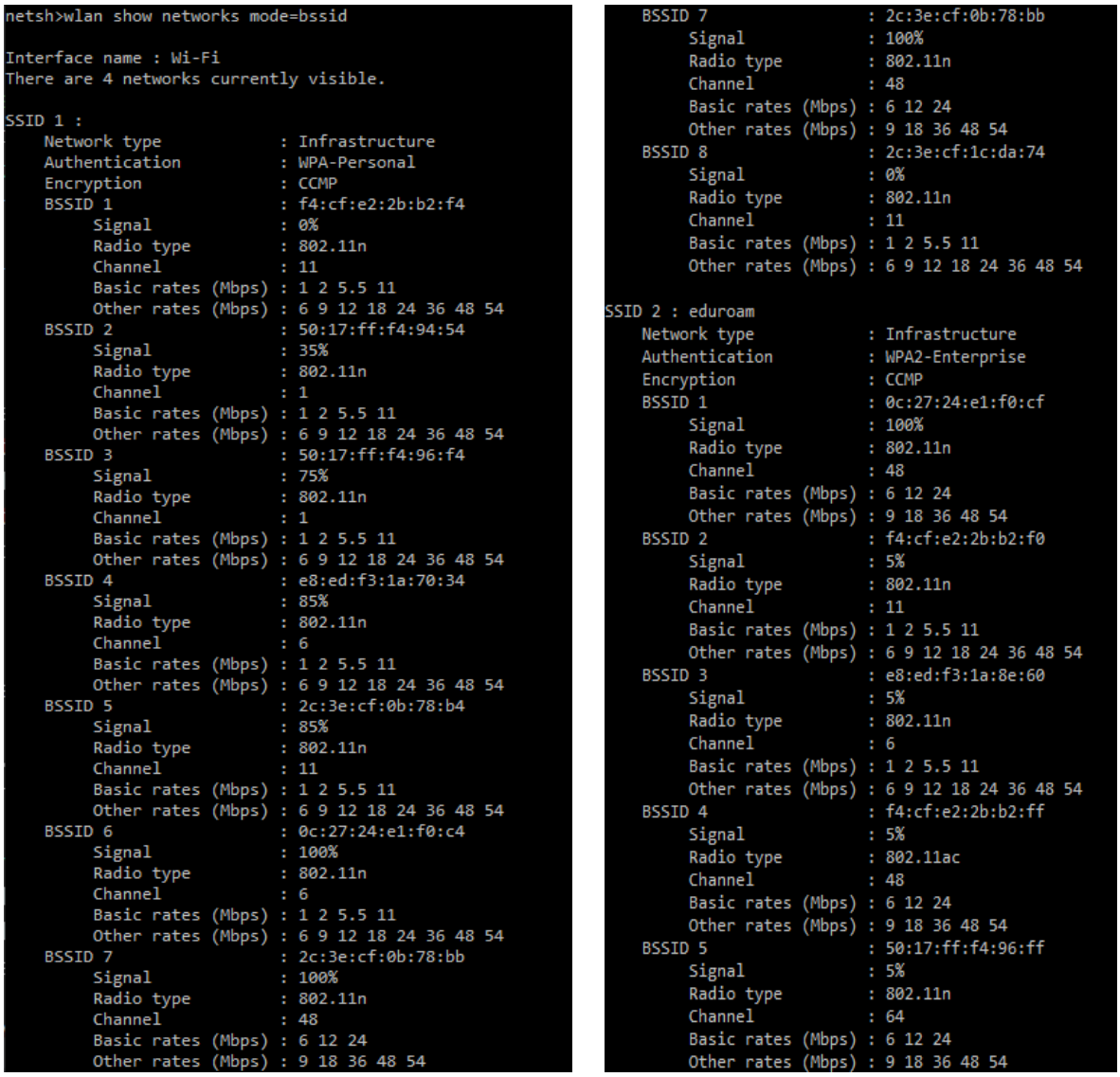

Figura 5.22 - Exemplo do comando realizado no netsh (não é o exemplo completo tendo em conta a dimensão total da resposta)

Como pode ser observado na imagem são devolvidos imensos dados que não são necessários para o sistema Where@UM. Assim, a resposta do comando é processada com vista a obter apenas os campos fundamentais como o SSID, o *Signal* e o *Channel*.

A potência de sinal é fornecida em percentagem (%) e como o sistema da Where@UM já se encontra a funcionar usando dBm foi utilizada uma tabela de conversão (Anexo I – Tabela de conversão percentagem (%) para dBm).

5.2 Servidor

Esta secção apresenta as modificações efetuadas no servidor Where@UM, a implementação do servidor de *chat* e a metodologia de comparação de *fingerprints*, todos os temas são abordados de forma detalhada.

5.3 Implementação do servidor Where@UM

Muitos dos serviços da Where@UM já estavam implementados no servidor. Todos sofreram alterações para integrar a capacidade de suportar multiplataformas. Foram, ainda, corrigidos *bugs* gerais no servidor e acrescentados campos quer na leitura dos objetos *JavaScript Object Notation* (JSON) quer na resposta aos clientes. No Anexo II – API Where@UM pode ser consultada a API completa dos serviços que a Where@UM disponibiliza.

Foram ainda criados *webservices* novos para responder aos requisitos de implementação de uma camada extra de segurança, nomeadamente na criação do conceito de sessões. Os novos *webservices* são denominados de "*token*", "*logout*" e "*refreshtoken*" e serão descritos de seguida:

Token

Uma autenticação que use um *token* não necessita de ficar persistente no lado do servidor (*stateless*). As credenciais são enviadas e o *token* é recebido pelas aplicações cliente, que vão adicionar o *token* em todos os pedidos subsequentes (também pode ser usado como *cookie*). A escolha do *token* como método de garantia de autenticidade do utilizador (nos pedidos subsequentes à autenticação) também considerou a necessidade de funcionamento em multiplataforma. É possível consultar mais detalhes da implementação na secção 5.4.

O *webservice* recebe o email, o endereço MAC da *interface* de rede do dispositivo e a plataforma que o utilizador está a utilizar. Tem como função principal gerar um *token* de 50 carateres e fornecê-lo à aplicação cliente que efetuou o pedido, fazendo simultaneamente algumas verificações fulcrais para garantir a integridade dos *tokens*, verificando, por exemplo, se este é único. Como os *tokens* são gerados automaticamente existe a probabilidade (que apesar de ínfima é real) de serem criados *tokens* iguais. Através deste processo, é garantido que cada *token* é único.

Na Figura 5.23 é possível observar um exemplo de *payload* do pedido:

```
{
    "email":"teste1@email.com",
    "mac": "75:BC:2C:9F:DC:7D",
    "platform": "windowsdesktop"
}
```

Figura 5.23 - Pedido em JSON do webservice token

RefreshToken

Cada *token* criado no Where@UM expira passadas 48 h. Assim, foi criado um mecanismo que garante que as sessões são renovadas automaticamente. Foi implementado um algoritmo (ver o fluxograma na Figura 5.24) que é executado sempre que são recolhidas *fingerprints* nos dispositivos.

É calculada a diferença entre a data atual e a data de criação do *token* (em horas). Caso esta seja entre 36 e 48h é realizada a renovação do *token*, caso seja superior às 48h é efetuado o *logout* porque se considera que a sessão expirou. Quando a aplicação está a correr normalmente e são recolhidas as *fingerprints* periódicas, como nenhuma das condições anteriormente especificadas se verifica, o *token* mantém-se o mesmo.

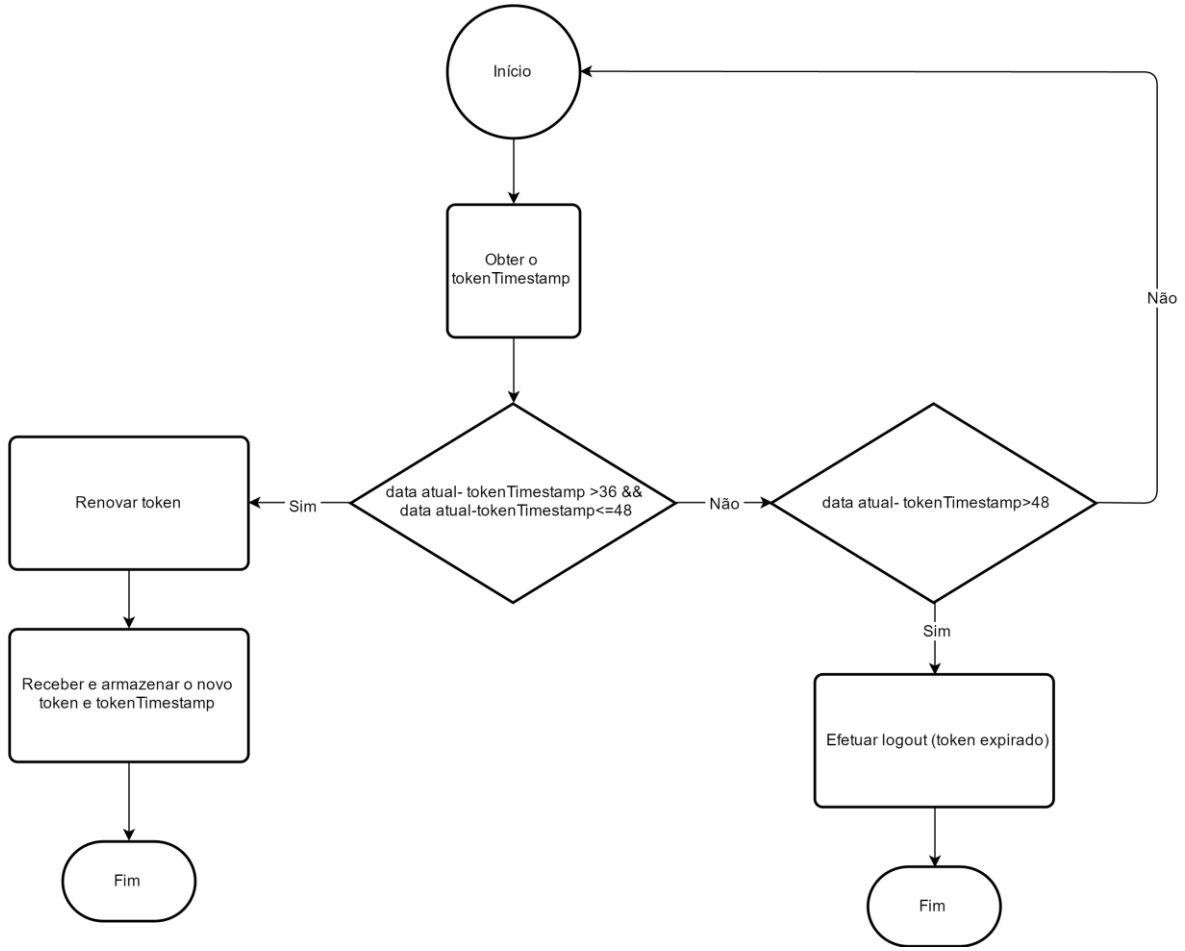

Figura 5.24 - Fluxograma do refresh token

Logout

Este *webservice* foi criado com o intuito de apagar a sessão do utilizador. É usado quando o utilizador fecha a aplicação manualmente. Nas verificações automáticas realizadas pelo algoritmo de renovação do *token*, quando este atinge o valor pré-definido, apaga automaticamente a sessão e o *token*.

5.3.1 Implementação do servidor de chat

A base de dados do servidor Where@UM contém informação dos utilizadores, associando o endereço MAC do dispositivo à plataforma usada, e caso seja *Android* é armazenado o id GCM (id de registo do dispositivo no serviço GCM). Quando um utilizador envia uma mensagem para outro utilizador, o servidor Where@UM vai averiguar qual é a plataforma ou plataformas que o destinatário utiliza, caso sejam *Android* o servidor obtém o seu id GCM e envia o pedido aos servidores da Google, por sua vez, estes enviam as mensagens sob a forma *push* para o destinatário (que tem um *listener* no dispositivo).

Na hipótese da plataforma do destinatário não ser *Android*, o servidor Where@UM envia a mensagem para o servidor de *chat* que irá guardar a mensagem tendo como campos o id do remetente, o id do destinatário, o *timestamp* e a mensagem. Quando o destinatário fizer *login* a aplicação cliente vai, automaticamente, pedir ao servidor para lhe enviar as mensagens que tenha para ele. Após confirmação deste facto, as mensagens são imediatamente apagadas da base de dados, o que garante a confidencialidade do sistema. A aplicação cliente em ambiente Windows continua a verificar de forma periódica a chegada de novas mensagens ao servidor de *chat*.

Por fim, caso o remetente seja de uma plataforma que não *Android* (*Windows Desktop* e/ou *Windows Phone*) o servidor Where@UM vai, da mesma forma, verificar a plataforma do destinatário. Na ocorrência de ser *Android*, obtém o id GCM do mesmo e envia a mensagem para os servidores GCM. Como os dispositivos *Android* têm implementados os módulos de *listener* fornecidos pelo serviço conseguem receber a mensagem sem esta necessitar de estar guardada no sistema Where@UM (servidor ou servidor de *chat*). Caso o destinatário seja utilizador de uma plataforma *Windows* o sistema comporta-se de maneira similar à já anteriormente referida (usando o servidor proprietário de *chat*).

Na Figura 5.25 é possível observar um diagrama de sequência com os casos acima enunciados.

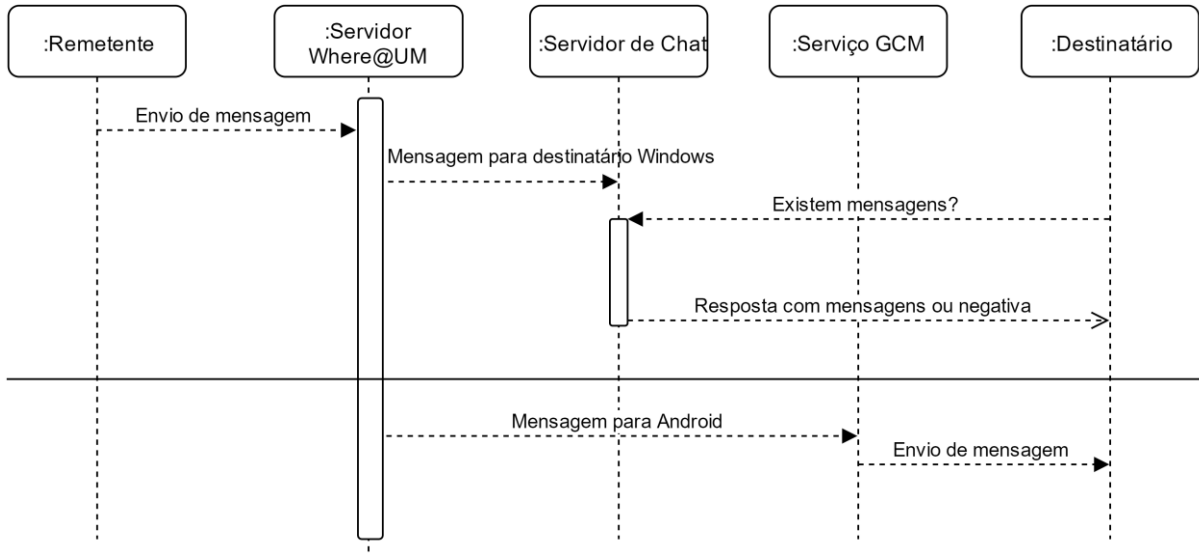

Figura 5.25 - Fluxo dos tipos de mensagens dos diferentes destinatários

Foi implementado um algoritmo para que o *pooling* não seja tão repetitivo e intensivo, mas, ao mesmo tempo, criar um efeito de *quasi realtime*. Foi implementado um *timer* inicial de 15 em 15 segundos, que é incrementado para o dobro sucessivamente, ou seja, corre a primeira vez e no caso de não existirem mensagens só passados 30 segundos é que volta a ser efetuada a pergunta ao servidor.

De forma a que este efeito não aconteça *ad eternum* sempre que é efetuada uma *fingerprint* automática (por norma de 5 em 5 min) o tempo de *pooling* volta a ser 15 s, recomeçando o processo algorítmico. Na Figura 5.26 é possível observar com clareza o fluxograma do algoritmo de pedidos de mensagens.

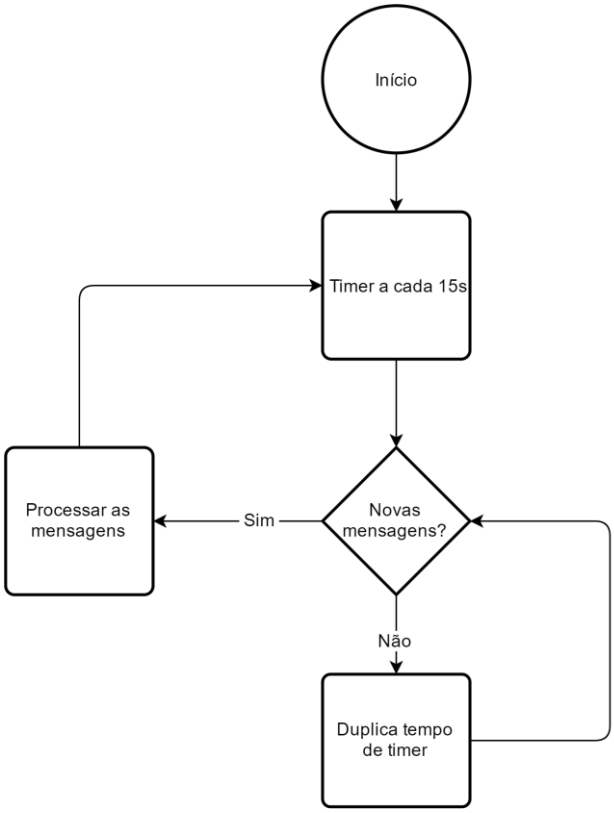

Figura 5.26 - Fluxograma do algoritmo incremental de pooling

5.3.2 Comparação de *fingerprints*

O processo de comparação de *fingerprints* não foi desenvolvido no âmbito desta dissertação. O estudo comparativo sobre as diferentes funções de cálculo de distância entre *fingerprints* já foi explorado em [17], concluindo que a função *Manhattan Distance* tem uma melhor *performance*.

Como referido em 2.1.5 existem vários fatores que causam alterações nos sinais de rádio e, por conseguinte, na recolha de *fingerprints*: como a movimentação, a existência de outros dispositivos e APs.

A distância entre *fingerprints* é uma parte fulcral da função de cálculo ver Equação 1[17].

$$D(FP^A, FP^B) = \frac{\sum_{i=1}^{n} \left| RSSI_i^A - RSSI_i^B \right|}{n}$$

Equação 1 - Distância entre fingerprints

Quando são adicionadas novas *fingerprints* ao sistema, a lista de APs pode não ser a mesma e é assumido um valor para o RSSI em falta (defRSSI). Foi realizada uma análise à similaridade entre *fingerprints* para encontrar o valor ótimo dos parâmetros defRSSI e do *threshold* (valor que foi definido para inferir se duas *fingerprints* são do mesmo local ou não) que foram definidos em -109 e 17.5, respetivamente, como demonstrado em [17].

Na Figura 5.27 é apresentado o fluxograma de comparação de *fingerprints*. Explicitamente o sistema obtém uma lista das últimas 5 *fingerprints*, associadas a cada local reconhecido pelo sistema, e calcula a distância da recebida com as da lista utilizando a função de cálculo da distância entre *fingerprints* (acima apresentada). Seguidamente, o sistema compara a *fingerprint* recebida com a distância mínima das *fingerprints* da lista (tendo em conta o *threshold*) e conclui se se trata do mesmo local ou não.

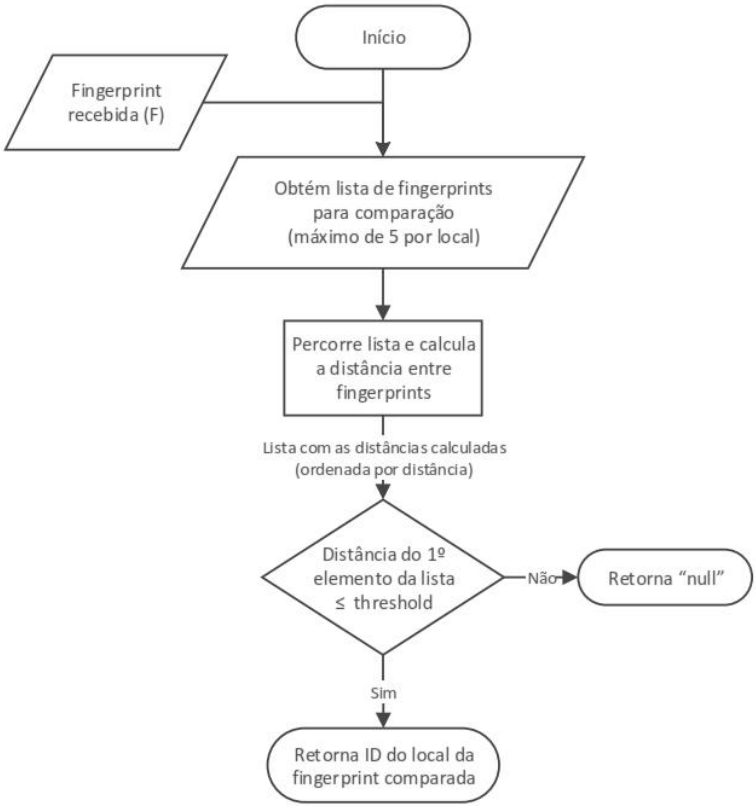

Figura 5.27 - Fluxograma de comparação de fingerprints [17]

5.4 Alterações de segurança

O sistema Where@UM não tinha implementado grandes sistemas de segurança, por opção da equipa de desenvolvimento, pois dado o seu foro académico e científico não são esperados problemas a esse nível. No entanto, foi decidido implementar o conceito do *token* de sessão aumentando, assim, o nível de segurança do sistema.

O *token* de sessão é criado quando a aplicação cliente faz um pedido do tipo *login* ou cria uma nova conta, é automaticamente criada uma sessão à qual é associada um *token* que é enviado à aplicação cliente com o estado inativo. Depois é enviado um pedido da aplicação cliente para o servidor com os dados da autenticação do utilizador, sendo estes validados pelo primeiro. Feita a validação, o *token* é ativado. A partir deste momento (similar a um *handshake*), todos os pedidos feitos ao servidor devem ter este *token* como *cookie* no *header* dos pedidos.

Na Figura 5.28 é apresentado o diagrama de sequência ilustrativo do processo após a introdução dos dados de *login*.

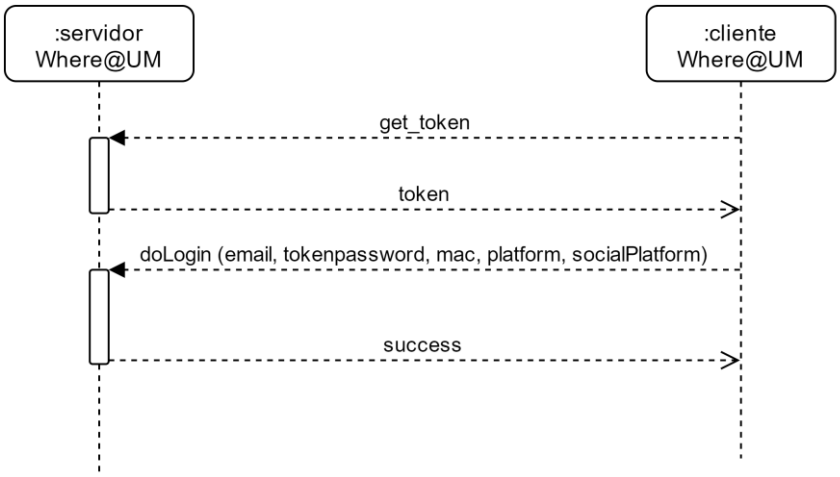

Figura 5.28 - Diagrama de sequência do mecanismo de obtenção de token e posterior login no sistema Where@UM

O processo demonstrado na Figura 5.28 é claro, começa com o preenchimento dos dados por parte do utilizador na aplicação cliente e segue-se um simples pedido ao servidor para obter o *token*, este devolve o *token* de sessão que está associado ao utilizador e ao dispositivo (tem que ser capaz de funcionar para dois dispositivos simultaneamente). A aplicação cliente recebe o *token* e faz o pedido de login no qual envia o *token* como *cookie* e as credenciais de acesso como o *email*, o endereço MAC, a plataforma ("*windowsphone*","*windowsdesktop*" ou "*android*") e ainda um campo denominado de **tokenpassword** que para ser calculado segue a fórmula:

$$tokenpassword = md5(md5(password) + token)$$

É calculado o MD5 da *password* do utilizador, a este valor é concatenado o *token* que está associado à sessão do utilizador. Posteriormente calcula-se o MD5 da *string* final e obtemos o **tokenpassword**.

Para que o *login* na aplicação seja concluído com sucesso, o servidor verifica as credenciais do utilizador. Caso estes tenham sido confirmados com sucesso, o servidor procede à ativação do *token* e os dados do utilizador são devolvidos à aplicação cliente. Se as credenciais não forem atestadas é devolvida uma mensagem de erro e é recusado o acesso à aplicação.

O principal objetivo do **tokenpassword** foi garantir que a password do utilizador que, no passado, era enviada "às claras" pela rede deixa-se de o ser.

Alteração das Passwords Antigas

Como detalhado em cima, a *password* enviada no *login* vai cifrada em MD5. A introdução deste conceito numa implementação de raiz não causaria problemas na implementação, mas como tem de ser implementada num sistema já existente teve de ser encontrada a melhor solução para que a nova versão consiga conviver com a antiga e ao mesmo tempo ser possível aos utilizadores mais antigos passarem a usufruir deste novo mecanismo de segurança. Na Figura 5.29 é possível observar um fluxograma da solução implementada ao nível da aplicação cliente.

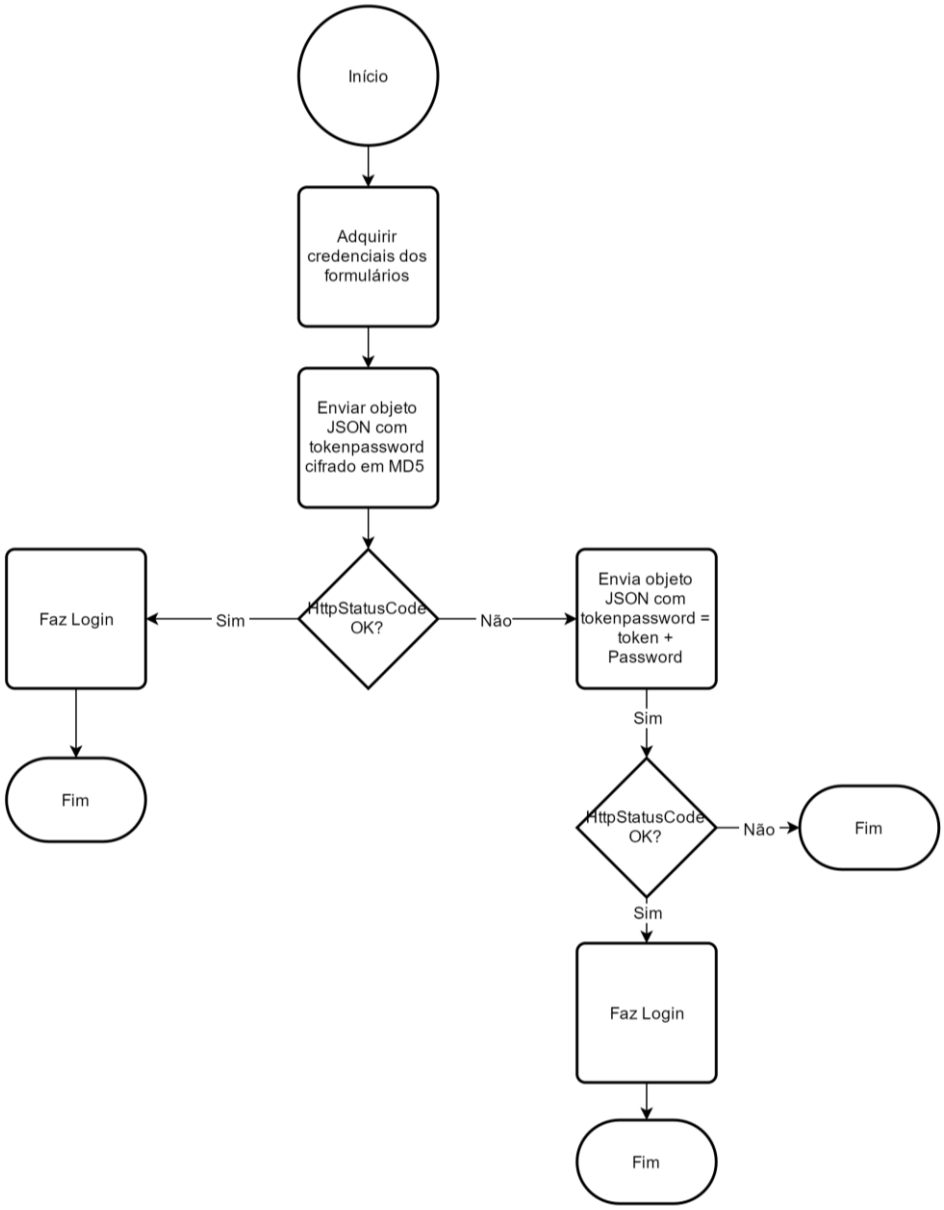

Figura 5.29 - Fluxograma do algoritmo de login (na aplicação cliente) tendo em consideração o mecanismo antigo e o novo

Deve ficar explícito que todo o processo ocorre após a aplicação cliente já ter feito a aquisição do *token*, processo descrito em detalhe no início desta secção.

No início, começa-se por adquirir os dados do formulário de *login* (*email* e *password*), após a aquisição dos formulários de *login* procede-se ao envio do objeto JSON devidamente preenchido e com o campo *tokenpassword* cifrado em MD5, respeitando o algoritmo e a função implementada, para o servidor. O servidor, por sua vez, efetua o seu algoritmo de confirmação (ver Figura 5.30) confirmando se se trata de uma conta com o novo mecanismo de segurança, devolve erro no caso da conta ainda usufruir do mecanismo antigo e o objeto não cumprir os requisitos necessários.

A aplicação cliente recebe a resposta e no caso desta ser afirmativa consegue realizar o *login* na aplicação. Quando a resposta do servidor é negativa então estamos numa situação em que a conta ainda utiliza o mecanismo de segurança antigo. Neste caso, é necessário construir um novo objeto JSON mas desta vez o campo *tokenpassword* apenas concatena o *token* com a *password*. Após a construção, o novo objeto é enviado ao servidor.

O servidor, efetua de novo o algoritmo e verifica o tipo de mecanismo de segurança que a conta utiliza e se o objeto cumpre os requisitos necessários. Caso cumpra e se trate de uma conta com o novo mecanismo implementado, é enviada uma mensagem de sucesso com os dados necessários e as aplicações cliente conseguem efetuar o *login* na aplicação.

Na Figura 5.30 é possível observar, em detalhe, o fluxograma do algoritmo de *login* no servidor.

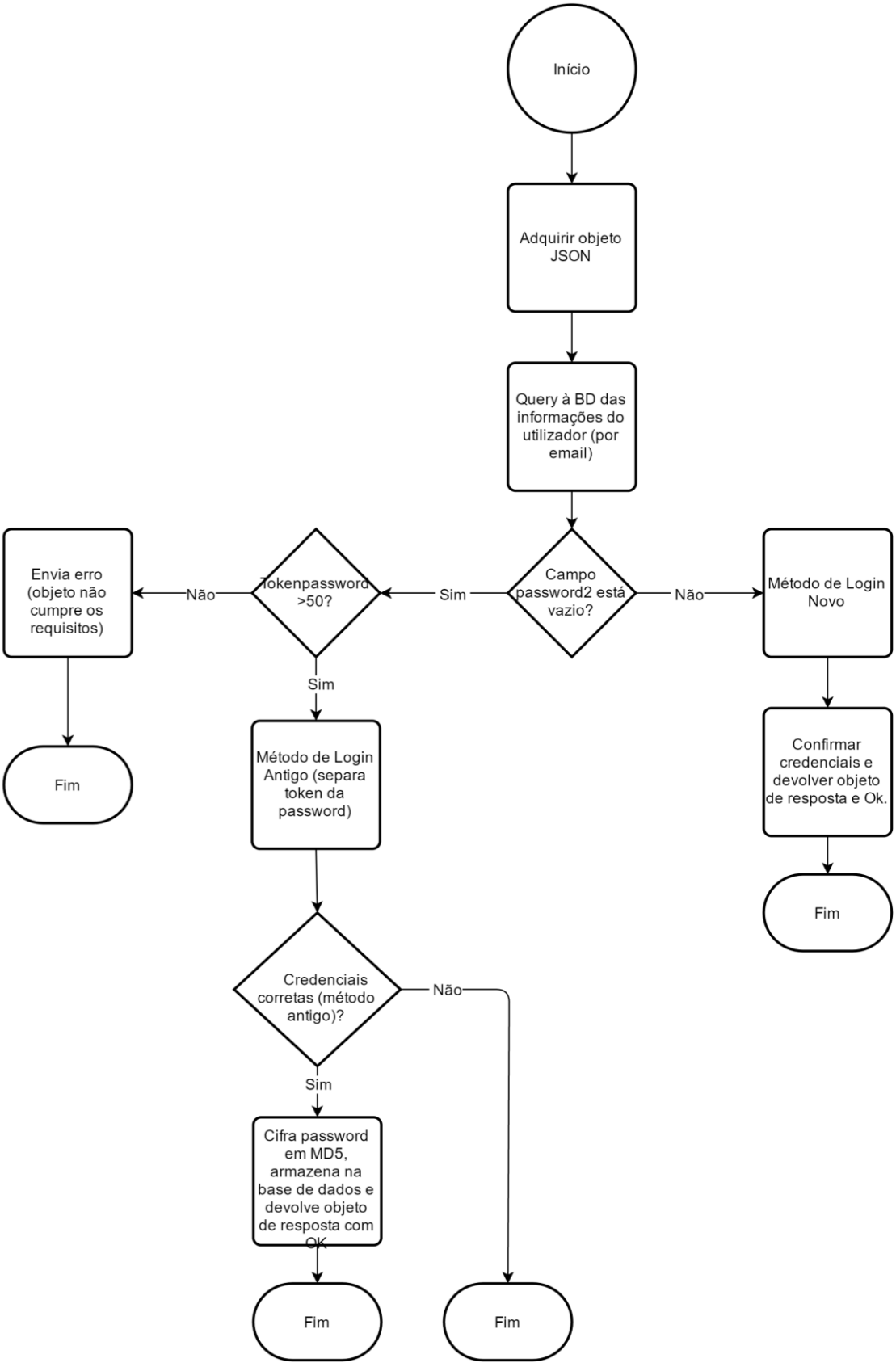

Figura 5.30 - Fluxograma do algoritmo de login (no servidor) tendo em consideração o mecanismo antigo e o novo

A conversão do velho mecanismo para novo é implementada em dois passos, primeiro é necessário separar o *token* da *password* que vêm concatenados da aplicação cliente. Após a separação é utilizado o mecanismo de cifra antigo e o valor resultante é comparado com o já armazenado na base de dados, em caso afirmativo o servidor tem a confirmação que a credencial é correta e é realizado o novo sistema de cifra usando MD5.

Por fim, é devolvido o objeto de resposta e o OK à aplicação cliente, concluindo assim o processo de *login*.

5.5 Alterações nas *fingerprints*

De forma a armazenar mais informações das *fingerprints* guardadas foi alterado o *webservice* que recebia as *fingerprints*, passando a ser guardado o *Service Set Identifier* (SSID) e o *channel*. Estes novos campos permitem que sejam realizadas análises mais intensivas à qualidade do sinal num dado edifício, valorizando os dados recolhidos pela aplicação.

Na Figura 5.31 é possível observar o novo objeto JSON que é enviado das aplicações cliente para o servidor. É de salientar que cada posição do *array fingerprint* representa 1 AP detetado.


```
{
 "idUser": "171",
 "idDevice": "64:BC:0C:7F:AC:7B",
 "timestamp": "2016-03-18 19:37:00",
 "motion": "",
 "place": "",
 "fingerprint": [{
  "mac": "2C:3E:CF:0B:78:BF",
  "rssi": "-37",
  "ssid": "eduroam",
  "channel": 1
 }, {
  "mac": "2C:3E:CF:0B:78:B4",
  "rssi": "-28",
  "ssid": "eduroam",
  "channel": 1
 }, {
  "mac": "2C:3E:CF:0B:78:BB",
  "rssi": "-94",
  "ssid": "",
  "channel": 1
 }]
}
```


Figura 5.31 - Exemplo de objeto JSON enviado para o servidor com a fingerprint

6. AVALIAÇÃO DO SISTEMA

No capítulo seis são apresentados os testes que foram realizados às funcionalidades do sistema após a sua implementação, com o intuito de verificar se a aplicação está em condições de ser publicada na *Windows Store*.

6.1 Testes a funcionalidades

O objetivo dos testes é a validação de requisitos e, acima de tudo, perceber o que ficou por implementar e que pode vir a ser introduzido num trabalho futuro. Nos testes que necessitavam de interações entre utilizadores foi utilizado aplicação cliente *Android*.

Para validar alguns dos requisitos foi imprescindível realizar intervenções no servidor. Todas as alterações realizadas foram testadas imediatamente, quer pelas aplicações cliente *Android*, quer pelas aplicações cliente *Windows*. Desta forma, garante-se que o servidor funciona e que não necessita de mais intervenções, permitindo um foco total no desenvolvimento das funcionalidades da aplicação cliente.

Na Tabela 3 é possível consultar os testes realizados.

Tabela 3 - Testes de funcionalidades

Página	Descrição da Funcionalidade	Estado
Página Inicial	Registar nova conta usando o *Facebook*	✓
	Registar nova conta usando o formulário da Where@UM	✓
	Login imediato após o registo (formulário Where@UM e Facebook)	✓
	Recuperar *password*	✓
Home	Quando um amigo tem um *nickname* mostrar *nickname,* quando tem nome mostrar o nome	✓
	Cor da imagem das fingerprints de acordo com a antiguidade - do próprio utilizador e dos amigos	✓
	Fazer *refresh*	✓
	Fazer *checkin*	✓
	Editar local manualmente fora da UM	✓
	Editar local manualmente dentro da UM	✓
	Enviar pedido de amizade	✓

	Convidar amigo (ainda não inscrito) a utilizar a Where@UM	✓
	Enviar mensagem para utilizador	✓
	Receber mensagem de utilizador	✓
	Mostrar planta do utilizador caso esteja disponível	✓
	Mostrar planta do amigo quando clicado se estiver disponível	✓
Friends	Apagar amigo	✓
	Aceitar amigo	✓
	Cancelar pedido de amizade enviado	✓
	Recusar pedido de amizade	✓
Overview	Mostrar localização do utilizador	✓
	Mostrar localização dos amigos do utilizador	✓
Settings	Alterar nickname	✓
	Alterar password	✓
	Alterar *sharing permission*	✓
Extra	Quando o utilizador fecha aplicação é chamado o *webservice logout*	✓

De seguida são descritos detalhadamente os testes apresentados na Tabela 3.

No primeiro parâmetro, página inicial, como forma de testar os registos, quer manual quer usando o *Facebook*, foram criadas contas de teste e verificadas a sua inserção na base de dados. Foi também realizado o *login* imediato após a criação das contas. A recuperação da *password* foi testada preenchendo o formulário com um *email* de uma conta válida e confirmado (no *email* usado) a receção da mensagem de *reset* da *password*.

Na página Home são validados muitos requisitos que só visualmente podem ser confirmados como, por exemplo, a cor da imagem das *fingerprints*, mostrar o *nickname* (quando o utilizador o tiver definido) e mostrar as plantas (se disponíveis). Os pedidos de *refresh* e *checkin* podem ser validados através de um clique. Todos os pedidos que envolvem interações com outros utilizadores foram validados usando outra aplicação cliente.

Posteriormente, na página Friends, à semelhança dos casos anteriores, usaram-se contas auxiliares para confirmar a implementação.

No quarto parâmetro, Overview, todas as funcionalidades são confirmadas visualmente, recorreu-se a contas auxiliares para confirmar o posicionamento dos amigos de forma a validar todos os requisitos.

Na página Settings foram testadas modificações às definições do utilizador, individualmente: alteração da *password*, alteração do *nickname* e alteração da *sharing permission*.

Por último, foi garantido que o utilizador quando faz *logout* a aplicação usa o *webservice logout* para terminar a sessão.

7. CONCLUSÃO

O foco principal desta dissertação foi o desenvolvimento de uma aplicação para computadores pessoais integrado no sistema Where@UM, tendo o referido desenvolvimento sido concluído com sucesso.

O desenho da arquitetura para a aplicação teve em conta a solução já existente para *Android* tentando, dentro do possível, uniformizar a abordagem na implementação da aplicação, quer a nível de funcionalidades quer a nível de visualização das mesmas (interface gráfico).

Este trabalho vem dar resposta a um inquérito realizado em 2014 [17] no qual: 19 % dos inquiridos desejava um meio de integração com o *Facebook*; 19 % pretendia um mapa com a localização dos utilizadores e 13 % pretendia a possibilidade de enviar mensagens entre utilizadores. Pode assim afirmar-se que, no âmbito desta dissertação, foram respondidas mais de 50 % das preferências dos utilizadores. É assim expectável que o uso da aplicação no futuro se intensifique, pois além das novas funcionalidades foram adicionadas novas plataformas (computador pessoal).

No decorrer do trabalho foram encontradas algumas dificuldades, nomeadamente, a escolha da utilização de padrões de desenvolvimento. Apesar de ser um método de implementação muito útil, cujos benefícios foram apontados durante este trabalho, tendo em conta o curto espaço de tempo para adquirir conhecimentos a nível de linguagens e de metodologias, o uso do padrão torna-se penoso e moroso, requerendo recursos temporais que podiam ter sido aplicados noutras áreas de desenvolvimento.

O facto do sistema Where@UM ser maduro e estar já implementado cria alguns desafios, nomeadamente: modificação de tabelas; alterações nos *webservices*; utilização de novos sistemas de segurança e introdução de multiplataformas. No âmbito desta dissertação, foram também resolvidos alguns *bugs* e reestruturados alguns *webservices* para responder a requisitos e dar mais fiabilidade e segurança aos serviços.

Foi desenvolvido um módulo de visualização de plantas de edifícios onde os utilizadores podem consultar a sua localização (caso o mapa exista) e a dos seus amigos. Foi ainda desenvolvido outro módulo, que usa a API do *Bing Maps*, para fornecer uma *overview* no mapa mundial da posição do utilizador e dos seus amigos.

De forma a manter a possibilidade do envio de mensagens entre utilizadores foi implementado um servidor de *chat*, que é modular, para dar suporte à introdução de novas plataformas no ambiente

aplicacional. O serviço anteriormente implementado (GCM) não dava resposta ao problema apresentado e, como tal, teve de ser implementada uma nova solução.

Concluído esta dissertação, pode ser afirmado que os principais requisitos de implementação e abordagem deste trabalho foram cumpridos. Foram ainda identificados novos desafios que serão descritos, deixando sugestões de melhoria contínua da aplicação e do sistema.

7.1 Trabalhos futuros

O trabalho desenvolvido pode ser melhorado e complementado. Os *deadlines* e novas ideias são os maiores inimigos da implementação, criando a necessidade de atribuir esforço nas principais funcionalidades antes de alargar o espectro alvo.

A integração com as redes sociais é ainda um tema que pode ser explorado e alargado, nomeadamente, através dos amigos do *Facebook* saber quais os que estão registados ou não na Where@UM e enviar o convite (quer para registo, quer para pedido de amizade).

De forma a melhorar a segurança na aplicação, deveria ser implementado o protocolo SSL nas comunicações entre as aplicações cliente e o servidor, garantindo assim confidencialidade e integridade nos dados trocados.

O servidor que agora usa uma metodologia *Pull* devia ser implementado usando uma metodologia *Push*, deixando o trabalho de gestão de mensagens para o servidor e tirando a necessidade de pedidos de mensagens por parte das aplicações cliente.

Para criar viralidade e interação entre os utilizadores pode ser sugerido ao utilizador que adicione amigos que estejam na mesma localização ou em localizações próximas, pode ainda ser enviada uma notificação quando um amigo entra na divisão na qual o utilizador se encontra (*geofencing*).

Dentro do *campus*, devem ainda ser introduzidas plantas de todos os edifícios da universidade. Podem ainda ser introduzidas novas funcionalidades, i.e., integrar o horário de salas disponíveis com a aplicação, quando se clicava numa sala aparecia a ocupação semanal da mesma. Pode ainda ser implementado um serviço *lead me to the spot* que após especificação do destino ajude visitantes e novos alunos a movimentarem-se dentro do *campus*.

O enriquecimento da aplicação a nível visual e de funcionalidades é algo que deve ser sempre tido em conta e a utilização dos utilizadores como críticos é umas das melhores maneiras, implementar um método de *feedback* diretamente da aplicação é algo que a curto prazo iria trazer *feedback* conciso de problemas e sugestões para a melhoria contínua da Where@UM, enquanto aplicação e sistema.

BIBLIOGRAFIA

[1] Y. Luo, O. Hoeber, and Y. Chen, "Enhancing Wi-Fi fingerprinting for indoor positioning using human-centric collaborative feedback," *Human-centric Comput. Inf. Sci.*, vol. 3, no. 1, p. 2, 2013.

[2] H. Liu, S. Member, H. Darabi, P. Banerjee, and J. Liu, "Survey of Wireless Indoor Positioning Techniques and Systems," *IEEE Trans. Syst. MAN, Cybern. C Appl. Rev.*, vol. 37, no. 6, pp. 1067–1080, 2007.

[3] "IndoorAtlas." [Online]. Available: https://www.indooratlas.com/. [Accessed: 20-May-2016].

[4] "Microsoft Research - Radar." [Online]. Available: https://www.microsoft.com/en-us/research/project/radar/. [Accessed: 20-May-2016].

[5] P. Bahl and V. N. Padmanabhan, "RADAR: An In-building RF-based User Location and Tracking System," *Proc. IEEE INFOCOM 2000. 19th Annu. Conf. Comput. Commun.*, vol. 2, pp. 775–784, 2000.

[6] A. Lamarca, J. Hightower, I. Smith, and S. Consolvo, "Self-Mapping in 802 . 11 Location Systems," *Proc. Int. Conf. Ubiquitous Comput. (UbiComp '05)*, pp. 87–104, 2005.

[7] K. Laasonen, M. Raento, and H. Toivonen, "Adaptive On-Device Location Recognition," *2nd Int. Conf. LNCS 3001, Springer Verlang*, pp. 287–304, 2004.

[8] J. Krumm and K. Hinckley, "The NearMe Wireless Proximity Server," pp. 283–300, 2004.

[9] J. Krumm, G. Cermak, and E. Horvitz, "RightSPOT : A Novel Sense of Location for a Smart Personal Object," in *UbiComp*, 2003, no. October.

[10] "Wifarer." [Online]. Available: http://www.wifarer.com/. [Accessed: 20-May-2016].

[11] "Accurate." [Online]. Available: https://www.accuware.com/. [Accessed: 20-May-2016].

[12] "Guardly." [Online]. Available: http://www.guardly.com/. [Accessed: 20-May-2016].

[13] "GoIndoor." [Online]. Available: https://www.goindoor.co/. [Accessed: 20-May-2016].

[14] "Google Indoor." [Online]. Available: http://www.google.com/maps/about/partners/indoormaps/. [Accessed: 21-May-2016].

[15] "Apple Indoor." [Online]. Available: https://developer.apple.com/maps/. [Accessed: 21-May-2016].

[16] "Microsoft Indoor Location Competition." [Online]. Available: https://www.microsoft.com/en-us/research/event/microsoft-indoor-localization-competition-ipsn-2016/. [Accessed: 23-May-2016].

[17] D. A. da S. Matos, "where @ UM - Aplicação móvel de posicionamento," 2014.

[18] H. A. Karimi, *Indoor Wayfinding and Navigation*. 2015.

[19] I. Bisht, "Indoor Positioning Using Modulated Echo Radio Localization Instrument," 2016.

[20] S. Kim and J. W. Chong, "An Efficient TDOA-Based Localization Algorithm without Synchronization between Base Stations," *Int. J. Distrib. Sens. Networks*, vol. 2015, 2015.

[21] J. Sun, J. Liu, S. Fan, and X. Lu, "China Satellite Navigation Conference (CSNC) 2015 proceedings: Volume I," *Lect. Notes Electr. Eng.*, vol. 340, 2015.

[22] O. P. Dewhirst, H. K. Evans, K. Roskilly, R. J. Harvey, T. Y. Hubel, and A. M. Wilson, "Improving the accuracy of estimates of animal path and travel distance using GPS drift-corrected dead reckoning," *Ecology and Evolution*, 2016.

[23] P. Bolliger, "Redpin - Adaptive, Zero-Configuration Indoor Localization through User Collaboration," in *Melt'08*, 2008, pp. 55–60.

[24] M. Paciga, "Herecast: An Open Infrastructure for Localitation-Based Services using Wi-Fi," vol. 9, 2004.

[25] A. LaMarca, Y. Chawathe, S. Consolvo, J. Hightower, I. Smith, J. Scott, T. Sohn, J. Howard, J.

Hughes, F. Potter, J. Tabert, P. Powledge, G. Borriello, and B. Schilit, “Place Lab: Device Positioning Using Radio Beacons in the Wild,” *Pervasive Comput.*, vol. 3468, pp. 116–133, 2005.
[26] T. King, S. Kopf, T. Haenselmann, C. Lubberger, and W. Effelsberg, “COMPASS: A probabilistic indoor positioning system based on 802.11 and digital compasses,” *Online*, no. September, pp. 34–40, 2006.
[27] P. Taylor, J. Ledlie, J. Park, D. Curtis, L. Camara, A. Costa, and R. Vieira, “Journal of Location Based Services Molé : a scalable , user-generated WiFi positioning engine,” no. April 2013, pp. 37–41, 2012.
[28] A. Flow, “Airista Flow.” [Online]. Available: http://www.airistaflow.com/applications/. [Accessed: 30-Aug-2016].
[29] J. F. M. Carvalho, “Localização de Dispositivos Móveis em Redes Wi-Fi,” 2007.
[30] Y. Gu, A. Lo, and I. Niemegeers, “A survey of indoor positioning systems for wireless personal networks,” *IEEE Commun. Surv. Tutorials*, vol. 11, no. 1, pp. 13–32, 2009.
[31] “Wifarer.” [Online]. Available: http://www.wifarer.com/technology.html. [Accessed: 05-Sep-2016].
[32] “Wifarer Google Play.” [Online]. Available: https://play.google.com/store/apps/details?id=com.wifarer.android. [Accessed: 05-Sep-2016].
[33] “Find my friends.” [Online]. Available: https://play.google.com/store/apps/details?id=com.fsp.android.friendlocator&hl=en. [Accessed: 26-Oct-2016].
[34] M. A. Youssef and A. Agrawala, “The Horus WLAN location determination system,” *Proc. 3rd Int. Conf. Mob. Syst. Appl. Serv. - MobiSys ’05*, pp. 205–218, 2005.
[35] V. Radu, L. Kriara, and M. K. Marina, “Pazl: A mobile crowdsensing based indoor WiFi monitoring system,” *2013 9th Int. Conf. Netw. Serv. Manag. CNSM 2013 its three collocated Work. - ICQT 2013, SVM 2013 SETM 2013*, pp. 75–83, 2013.
[36] A. Gain, “RF Basics - Part 1,” 2007.
[37] “Windows Device Geolocation.” [Online]. Available: https://msdn.microsoft.com/library/windows/apps/windows.devices.geolocation.aspx. [Accessed: 19-Sep-2016].
[38] “API Foursquare.” [Online]. Available: https://developer.foursquare.com/docs/. [Accessed: 19-Sep-2016].
[39] V. P. B. Barroso, “Posicionamento colaborativo em redes Wi-Fi - Where @ UM2,” 2015.
[40] I. Silva, “Sistemas de posicionamento WiFi,” Universidade do Minho, 2016.
[41] P. P. Snehal Mumbaikar, “Web Services Based On SOAP and REST Principles,” *Int. J. Sci. Res. Publ.*, vol. 3, no. 5.
[42] K. Wagh and R. Thool, “A Comparative Study of SOAP Vs REST Web Services Provisioning Techniques for Mobile Host,” *J. Inf. Eng. Appl.*, vol. 2, no. 5, pp. 12–16, 2012.
[43] G. Mulligan and D. Gračanin, “A comparison of soap and rest implementations of a service based interaction independence middleware framework,” *Proc. - Winter Simul. Conf.*, pp. 1423–1432, 2009.
[44] “W3C Soap Recommendation.” [Online]. Available: https://www.w3.org/TR/soap12/. [Accessed: 05-Sep-2016].
[45] “Bing Maps WPF Control.” [Online]. Available: https://msdn.microsoft.com/en-us/library/hh750210.aspx. [Accessed: 17-Sep-2016].
[46] “Facebook Dev Page.” [Online]. Available: https://developers.facebook.com/. [Accessed: 20-Jul-2016].
[47] M. Fowler, “Separated Presentation.” [Online]. Available: http://martinfowler.com/eaaDev/SeparatedPresentation.html. [Accessed: 06-Oct-2016].

[48] “Microsoft MVVM.” [Online]. Available: https://msdn.microsoft.com/en-us/library/hh848246.aspx.
[49] “Commands.” [Online]. Available: http://stackoverflow.com/questions/22285866/why-relaycommand. [Accessed: 03-Oct-2016].
[50] “MVVM Messenger.” [Online]. Available: http://dotnetpattern.com/mvvm-light-messenger. [Accessed: 03-Oct-2016].
[51] J. Bardwell, “Converting Signal Strength Percentage to dBm Values,” *WildPackets, Inc*, no. November, pp. 1–12, 2002.
[52] “O que são metadados?” [Online]. Available: http://www.metadados.pt/oquesaometadados. [Accessed: 05-Oct-2016].

ANEXO I – TABELA DE CONVERSÃO PERCENTAGEM (%) PARA DBM

Os valores de potência de sinal adquiridos usando o *netsh* vêm sob a forma de percentagem (%) como tal foi usada a tabela seguinte como regra para *match* de valores entre percentagem e dBm.

Tabela 4 - Tabela de conversão da Cisco [51]

0	= -113	34	= -78	68	= -41
1	= -112	35	= -77	69	= -40
2	= -111	36	= -75	70	= -39
3	= -110	37	= -74	71	= -38
4	= -109	38	= -73	72	= -37
5	= -108	39	= -72	73	= -35
6	= -107	40	= -70	74	= -34
7	= -106	41	= -69	75	= -33
8	= -105	42	= -68	76	= -32
9	= -104	43	= -67	77	= -30
10	= -103	44	= -65	78	= -29
11	= -102	45	= -64	79	= -28
12	= -101	46	= -63	80	= -27
13	= -99	47	= -62	81	= -25
14	= -98	48	= -60	82	= -24
15	= -97	49	= -59	83	= -23
16	= -96	50	= -58	84	= -22
17	= -95	51	= -56	85	= -20
18	= -94	52	= -55	86	= -19
19	= -93	53	= -53	87	= -18
20	= -92	54	= -52	88	= -17
21	= -91	55	= -50	89	= -16
22	= -90	56	= -50	90	= -15
23	= -89	57	= -49	91	= -14
24	= -88	58	= -48	92	= -13
25	= -87	59	= -48	93	= -12
26	= -86	60	= -47	94	= -10
27	= -85	61	= -46	95	= -10
28	= -84	62	= -45	96	= -10
29	= -83	63	= -44	97	= -10
30	= -82	64	= -44	98	= -10
31	= -81	65	= -43	99	= -10
32	= -80	66	= -42	100	= -10
33	= -79	67	= -42		

ANEXO II – API WHERE@UM

Especificação dos *webservices* que servem como protocolo de comunicação entre as aplicações cliente e o servidor. Existem 2 principais componentes onde se enquadram os *webservices*: suporte à aplicação, construção de mapas rádio.

Suporte à aplicação

1. Pedido de Token de sessão

Método HTTP	**POST**
URI	{server}/users/self/token
Token como Cookie	Não
Parâmetros	• String email – endereço eletrónico do novo utilizador • String mac – endereço mac • String platform – plataforma proveniente (android, windowsdesktop, windowsphone)
***Payload* pedido**	{ "email":"teste1@email.com", "mac": "64:BC:0C:7F:AC:7B", "platform": "windowsdesktop" }
Resposta (formato)	Sucesso – devolve objeto com os dados do perfil de utilizador. • String token 400 Bad Request – Em caso de insucesso

Descrição: Pedido efetuado antes do utilizador iniciar sessão. Como resposta é devolvido o token que o utilizador usa para se poder autenticar na plataforma.

2. Autenticação de um utilizador

Método HTTP	**POST**
URI	{server}/users/self/login
Token como Cookie	Sim
Parâmetros	• String email – endereço eletrónico do novo utilizador • String tokenpassword – Hash MD5 da password com o token. Este campo é o MD5(token+MD5(password)) no novo método de autenticação, caso o utilizador ainda não tenha a password como MD5, o valor token password é uma string com o valor "token+password"

	• String mac – endereço mac • String platform – plataforma proveniente (android, windowsdesktop, windowsphone)

***Payload* pedido**	{
	"email":"teste1@email.com",
	"tokenpassword":"3db3545816dbfdb548863a15a41865ff",
	"mac": "64:BC:0C:7F:AC:7B",
	"platform": "windowsdesktop"
	}

Resposta (formato)	Sucesso – devolve objeto com os dados do perfil de utilizador.
	• Int idUser
	• String name
	• String nickname
	• String email
	• String locationSharing
	• Int deletedAccount
	• String registrationDate
	• String token
	• String tokenTimestamp
	{
	"idUser": "171",
	"name": "Ricardo Mesquita",
	"email": "ricardomesquita@outlook.pt",
	"nickname": "Ricardo",
	"locationSharing": "1",
	"deletedAccount": "0",
	"registrationDate": "2016-03-13 10:38:09",
	"tokenTimestamp": "2016-05-11 17:00:12"
	}
	400 Bad Request – Em caso de insucesso

Situação extraordinária:

Como na base de dados ainda existem passwords guardadas noutro tipo de cifra que não MD5, devolve uma mensagem específica.

```
{
     "status": "Failed",
     "msg": "Please use the old authentication method or wrong email-password"
}
```

Neste caso o campo tokenpassword deve conter o token+password sem qualquer tipo de hashing.

Descrição:. Após a obtenção do *token* de sessão, é necessário proceder à autenticação do utilizador. Caso as credenciais estejam corretas é devolvido um objeto JSON com os dados do utilizador.

3. Logout de um utilizador

Método HTTP	**POST**
URI	{server}/users/self/logout
Token como Cookie	Sim
Parâmetros	
***Payload* pedido**	
Resposta (formato)	200 Sucesso – devolve OK 400 Bad Request – Em caso de insucesso

Descrição: Quando o utilizador faz *logout* é necessário realizar este pedido para apagar a sessão ativa no sistema.

4. Refresh do token de sessão

Método HTTP	**POST**
URI	{server}/users/{idUser}/refreshtoken
Token como Cookie	Sim
Parâmetros	
***Payload* pedido**	

Resposta (formato)	{
	"token": "uKioww1Kxf5TsKQAWR5yNyPl5HSxDgYacfXs4ioWeomjjqTq03",
	"tokenTimestamp": "2016-05-07 13:39:44"
	}

Descrição: É necessário renovar o *token* de sessão antes deste expirar. É então realizado este pedido que recebe como resposta o novo *token*.

5. Registo de um utilizador

Método HTTP	**POST**
URI	{server}/users
Token como Cookie	Não
Parâmetros	• String name – nome do novo utilizador • String nickname (opcional) – alcunha do novo utilizador • String email – endereço eletrónico do novo utilizador • String password – valor md5 da password • String idDevice – idDevice do dispositivo

	• String platform – plataforma proveniente (android, windowsdesktop, windowsphone)

***Payload* pedido**	{
	"name":" Ricardo ",
	"nickname":"",
	"email":" ricardomesquita@outlook.pt",
	"password":"123345",
	"idDevice": "64:BC:0C:7F:AC:7B",
	"platform": "windowsdesktop"
	}

Resposta (formato)	{
	"idUser": "171",
	"name": "Ricardo",
	"email": "ricardomesquita@outlook.pt",
	"nickname":"Ricardo",
	"locationSharing": "1",
	"deletedAccount": "0",
	"registrationDate": "2016-05-09 16:38:49",
	"token": "1dzkBxBzwbmAVzH3Bi71AX4Zdf2haeK5ZZzkK4UDvwMGOOVYKD",
	"tokenTimestamp": "2016-05-09 16:38:49"
	}

Descrição: Pedido de registo de novo utilizador no sistema. Como resultado é devolvido o objeto JSON com os dados do utilizador

6. Obter a planta de um piso

({server}=http://urano.dsi.uminho.pt/whereatumdev/cal)

Método HTTP	GET
URI	{server}/floormaps/maps?operator=Universidade%20do%20Minho&area=Campus%20de%20Azurém&building=Escola%20de%20Engenharia&floor=1
Parâmetros	• Operador – Nome do operador • Área – Área geográfica • Edifício – nome do edifício • Piso – nome do piso (por exemplo, 0, 1, 2, etc)
Resposta (formato)	Conteúdo do ficheiro da planta (XML/OSM) Caso a planta não exista, é devolvido o código 404 (not found).

Descrição: Este pedido foi desenvolvido especificamente para servir a aplicação Where@UM, mas pode ser utilizado por qualquer serviço. Através dos nomes dos parâmetros é possível obter a planta de cada piso, caso exista. Como os nomes permitem identificar cada um dos elementos (operador, área, edifício e piso), é possível obter a planta do piso utilizando estas informações.

7. Envio de mensagens

Método HTTP	**POST**
URI	{server}/users/{idUser}/messages
Token como Cookie	Sim
Parâmetros	• Regid – identificador de registo do serviço GCM do destinatário; • Message – mensagem a enviar.

***Payload* pedido**	{
	"regid":"APA91bHPpzbLGrRwJMjGVuq0q02jx5bf6hBHKVcjX347LjMcWFaMkQsK6PHIV0X3o-wP5CxRFvPa6tc2CO0BCshzDWcgceBW0O_EVBEZMlXQYEbTgXp-1iA",
	"message":"olá"
	}

Resposta (formato)	{
	"multicast_id": 6417630731239064644,
	"success": 1,
	"failure": 0,
	"canonical_ids": 0,
	"results": [{
	"message_id": "0:1464080873442675%aa711c20f9fd7ecd"
	}]
	}

Descrição: Pedido responsável por receber a mensagem juntamente com o identificador de registo no serviço GCM do destinatário e o identificador do utilizador que enviou a mensagem. Envia estes dados para o servidor do GCM (invocando a função "*send_push_notifications*") que se certificará de enviar a mensagem ao utilizador destinatário.

8. Envio do GCM do utilizador

Método HTTP	**POST**
URI	{server}/users/{idUser}/gcm
Token como Cookie	Sim
Parâmetros	• regID – identificador de registo do serviço GCM do utilizador; • idDevice – endereço mac do dispositivo.

***Payload* pedido**	{
	"regID":"APA91bEDiRO-63QzK0zZVsVd3IJVpRa__P1eVYdu5b9OzHCFWcnFWNrQpS8n2f-lnDDOr19vgH8pBquUr7FcPunP3FkSsyw0QFovpXS0utJs7UC54IxOLps",
	"idDevice": "00:04:4B:2C:BE:3A"
	}

Descrição: Envia o regID (fornecido pelo GCM) e o idDevice da aplicação cliente para o servidor da where@um.

9. Obter GCM do utilizador

Método HTTP	**Get**
URI	{server}/users/{idUserDestination}/Regid
Token como Cookie	Sim
Parâmetros	
***Payload* pedido**	
Resposta (formato)	{ "gcm_regID":"APA91bHX1vrwQNBUphE6kgGkvzOSTJjyK1JYsSxRGX0X…(continua)" }

Descrição: Sempre que um utilizador deseja enviar uma mensagem a outro utilizador (amigo) a aplicação móvel solicita a esta função o identificador de registo no serviço GCM referente ao amigo a que pretende enviar uma mensagem.

10. Enviar pergunta1

Método HTTP	**Get**
URI	{server}/send/{idUser}/question1
Token como Cookie	Sim
Parâmetros	
***Payload* pedido**	
Resposta (formato)	

Descrição: Pedido responsável por enviar uma notificação aos utilizadores confirmando a localização da aplicação.

11. Enviar pergunta2

Método HTTP	**Get**

URI	{server}/send/{idUser}/question2
Token como Cookie	Sim
Parâmetros	
***Payload* pedido**	
Resposta (formato)	

Descrição: Pedido responsável por enviar uma notificação aos utilizadores alertando que estão numa localização desconhecida, e perguntando se desejam inserir a sua localização atual.

12. Enviar resposta1

Método HTTP	**POST**
URI	{server}/send/{idUser}/responseQ1
Token como Cookie	Sim
Parâmetros	• Response – valor da resposta; • Date – data da resposta.

***Payload* pedido**	{
	"response":"YES",
	"date_e":"24/05/2016 16:05:55"
	}
	Nota: o valor da response pode ser "YES" ou "NO"

Resposta (formato)	Devolve erro caso exista algum problema

Descrição: Sempre que um utilizador responder a uma questão, enviada pela função "sendquestion1", esta função é invocada e tem como função guardar na base de dados, na tabela "feedbackUsers", as respostas emitidas pelos utilizadores e atualizar os campos "likes" e "unlikes" da tabela "fingeprints".

13. Enviar resposta2

Método HTTP	**POST**
URI	{server}/send/{idUser}/responseQ2
Token como Cookie	Sim
Parâmetros	• Response – valor da resposta; • Date – data da resposta.
***Payload* pedido**	{ "response":"YES",

	"date_e":"24/05/2016 16:09:10" } Nota: o valor da response pode ser "YES" ou "NO"
Resposta (formato)	Devolve erro caso exista algum problema

Descrição: Quando um utilizador responde a uma questão enviada pela função "sendquestion2", esta é invocada e tem como função guardar na base de dados as respostas emitidas pelos utilizadores.

14. Obter informação relativa aos amigos do utilizador

Método HTTP	**GET**
URI	{server}/users/{idUser}/friends
Token como Cookie	Sim
Parâmetros	
***Payload* pedido**	

Resposta (formato)	Sucesso – devolve objeto com os dados os amigos
	400 Bad Request – Em caso de insucesso
	{
	"pending": null,
	"friends": [
	{
	"idUser": "4",
	"date": "2016-02-17 12:45:10",
	"name": "Adriano Moreira",
	"email": "ajcm2appls@gmail.com",
	"nickname": "ajcmoreira"
	}
	],
	"requests": [
	{
	"idUser": "173",
	"date": "2016-02-24 23:28:32",
	"name": "Ricardo Outlook",
	"email": "ricardomesquita@outlook.pt",
	"nickname": null
	},
	{
	"idUser": "180",
	"date": "2016-02-24 23:26:59",
	"name": "joaoteste@teste.com",
	"email": "joaoteste@teste.com",
	"nickname": null

```
            }
        ]
    }
```


Descrição: Pedido das informações relativamente ao estudo das relações de amizade do utilizador.

15. Obter localização do utilizador e seus amigos

Método HTTP	**GET**
URI	{server}/users/{idUser}/checkins
Token como Cookie	Sim
Parâmetros	
***Payload* pedido**	
Resposta (formato)	Sucesso – devolve objeto com os dados do perfil de utilizador.


```
{
        "self": {
                "deviceID": "64:BC:0C:7F:AC:7B",
                "timeStamp": "Mar 17, 2016 12:15:06",
                "xPos": 0,
                "yPos": 0,
                "building": "null",
                "floor": "null",
                "room": "null",
                "confidence": 0,
                "raio": -1,
                "lat": "0.00000",
                "lon": "0.00000"
        },
        "friends": [{
                "idUser": "4",
                "name": "Adriano Moreira",
                "nickname": "ajcmoreira",
                "idDevice": "cc:fa:00:b4:90:63",
                "location": {
                        "roomID": "76",
                        "room": "LID 3",
                        "floorID": "49",
                        "floor": "1",
                        "buildingID": "22",
                        "building": "Escola de Engenharia",
                        "areaID": "10",
                        "area": "Campus de Azurém",
                        "timeStamp": "Mar 16, 2016 11:09:42",
                        "lat": "0.00000",
                        "lon": "0.00000"
```

```
                    }
                }]
            }
```


Nota: O *timestamp* do self é atualizado sempre que que este pedido é feito. Ao invés de receber com o último que foi enviado.

Descrição: Pedido realizado para obter a localização do utilizador e dos seus amigos.

16. Alterar permissão de partilha da localização

Método HTTP	**PUT**
URI	{server}/users/{idUser}/permission
Token como Cookie	Sim
Parâmetros	• Int permission – 0 indica que não é permitida a partilha da localização. 1 indica que a partilha da localização é permitida
***Payload* pedido**	{ "permission":"0" }
Resposta (formato)	Sucesso – devolve OK 400 Bad Request – Em caso de insucesso

Descrição: Alteração da autorização de partilha da localização.

17. Remover amigo

Método HTTP	**POST**
URI	{server}/users/{idUser}/unfriend
Token como Cookie	Sim
Parâmetros	• Int idUser – identificador do utilizador que se pretende remover
***Payload* pedido**	{ "idUser":"123" }
Resposta (formato)	Sucesso – devolve OK 400 Bad Request – Em caso de insucesso

Descrição: Para a remoção de um amigo é enviado o id do amigo que se pretende remover, usando um objeto JSON.

18. Enviar pedido de amizade a um utilizador

Método HTTP	**POST**
URI	{server}/users/{idUser}/request
Token como Cookie	Sim
Parâmetros	• String email – Email do utilizador que pretende convidar
***Payload* pedido**	{ "email":"ivo_silva11@hotmail.com" }
Resposta (formato)	Sucesso: OK Insucesso: 400 Bad Request

Descrição: Envio de pedido de amizade a um utilizador, o parâmetro enviado é apenas o email do utilizador.

19. Cancelar pedido de amizade efetuado

Método HTTP	**POST**
URI	{server}/users/{idUser}/cancel
Token como Cookie	Sim
Parâmetros	• Int idUser – identificador do utilizador associado ao pedido efetuado
***Payload* pedido**	{ "idUser":"171" }
Resposta (formato)	Sucesso: OK Insucesso: 400 Bad Request

Descrição: Pedido que permite cancelar um pedido de amizade já realizado através do id do utilizador.

20. Aceitar pedido de amizade

Método HTTP	**POST**
URI	{server}/users/{idUser}/approve
Token como Cookie	Sim
Parâmetros	• Int idUser – identificador do utilizador que se pretende aceitar
***Payload* pedido**	{ "idUser":"171"

	}
Resposta (formato)	Sucesso: OK Insucesso: 400 Bad Request

Descrição: Pedido enviado quando o utilizador aceita o pedido de amizade. Para aceitar é enviado o id do utilizador que fez o convite.

21. Recusar pedido de amizade

Método HTTP	**POST**
URI	{server}/users/{idUser}/deny
Token como Cookie	Sim
Parâmetros	• Int idUser – identificador do utilizador que se pretende recusar

***Payload* pedido**	{
	"idUser":"171"
	}

Resposta (formato)	Sucesso: OK Insucesso: 400 Bad Request

Descrição: Pedido realizado para recusar um pedido de amizade.

22. Reenvio de palavra-chave, gerada automaticamente pelo servidor

Método HTTP	**POST**
URI	{server}/users/new/password
Token como Cookie	Não
Parâmetros	String email – Email do utilizador

***Payload* pedido**	{
	"email":"ricardomesquita@outlook.pt"
	}

Resposta (formato)	Sucesso: OK Insucesso: 400 Bad Request

Descrição: Quando o utilizador perde a *password* e pretende receber uma nova. O serviço faz *reset* da *password* e gera uma automaticamente que é enviada ao utilizador.

23. Mudar Alcunha

Método HTTP	**PUT**
URI	{server}/users/{idUser}/nickname
Token como Cookie	Sim
Parâmetros	• String nickname – nickname que se deseja (tem que ser diferente do atual senão o servidor dá erro).

***Payload* pedido**	{
	"nickname":"teste"
	}

Resposta (formato)	Sucesso: OK Insucesso: 400 Bad Request

Descrição: Pedido realizado quando o utilizador pretende alterar a sua alcunha.

24. Mudar Password

Método HTTP	**PUT**
URI	{server}/users/{idUser}/password
Token como Cookie	Sim
Parâmetros	• String currentTokenPassword – md5(token+md5(password)), onde a password, é o valor atual do utilizador • String newPassword –md5(password) nova password que o utilizador deseja.

***Payload* pedido**	{
	"currentTokenPassword":"gaOfl",
	"newPassword":"teste"
	}

Resposta (formato)	Sucesso: OK Insucesso: 400 Bad Request

Caso seja devolvida uma mensagem de aviso semelhante à do pedido de Login, é necessário fazer o pedido novamente com os seguintes parâmetros:

currentTokenPassword = ”token+password”
newPassword = ”newPassword” (conteúdo às claras)

Descrição: O pedido permite alterar a *password* do utilizador, é necessário enviar a *password* atual e a nova.

POSICIONAMENTO

({server}=http://urano.dsi.uminho.pt/whereatumdev/pe2)

25. Obter lista de locais UM

Método HTTP	**GET**
URI	{server}/places/um
Token como Cookie	Sim
Parâmetros	
***Payload* pedido**	
Resposta (formato)	JSON com os dados dos locais da UM

Descrição: Pedido para obter lista de locais dentro da universidade.

26. Obter lista de locais fora da UM

Método HTTP	**GET**
URI	{server}/places/{coordenadas GPS}/search Exemplo: {server}/places/41.5924797,-8.7857590/search
Token como Cookie	Sim
Parâmetros	
***Payload* pedido**	
Resposta (formato)	{

```
        "places": [{
                "id": "51d1f929498e1352a72316b6",
                "name": "HOOL Restaurante",
                "location": {
                        "address": "Largo da Oliveira",
                        "lat": 41.442932653653,
                        "lng": -8.2928088075987,
                        "distance": 4,
                        "cc": "PT",
                        "city": "Guimar\u00e3es",
                        "state": "Braga",
                        "country": "Portugal",
                        "formattedAddress": ["Largo da Oliveira",
"Guimar\u00e3es", "Portugal"]
                }
        }, {
```

```
                "id": "486",
                "name": "Largo da Oliveira",
                "checkinsCount": "1",
                "location": {
                        "country": "Portugal",
                        "cc": "PT",
                        "city": "Guimar\u00e3es",
                        "address": "Largo da Oliveira",
                        "distance": 9,
                        "lat": "41.4430008",
                        "lng": "-8.2929325"
                }
        }, {
                "id": "84",
                "name": "Manifestis Probatum",
                "checkinsCount": "2",
                "location": {
                        "country": "Portugal",
                        "cc": "PT",
                        "city": "Guimar\u00e3es",
                        "address": "R. Egas Moniz, 57-63",
                        "distance": 97,
                        "lat": "41.4421005",
                        "lng": "-8.2928705"
                }
        }]
}
```


Descrição: Quando se pretende alterar a localização dentro da UM é necessário obter a lista de locais suportados pelo sistema.

27. Enviar fingerprint

Método HTTP	**POST**
URI	{server}/fingerprints
Token como Cookie	Sim
Parâmetros	• String idDevice – Endereço MAC da interface Wi-Fi do dispositivo do utilizador; • Array fingerprint: String MAC – endereço MAC de cada AP; Int RSSI – nível de sinal do AP em dBm; Int channel – canal associado à frequência de funcionamento do AP; String SSID – nome da rede do AP;

	• Int motion – valor que indica se o utilizador está em movimento (opcional); • String place – informação relativa ao local (opcional); String timestamp – data e hora da recolha da *fingerprint* (`yyyy-mm-dd HH:mm:ss`).

***Payload* pedido**	{
	"idUser": "171",
	"idDevice": "64:BC:0C:7F:AC:7B",
	"timestamp": "2016-03-18 19:37:00",
	"motion": "",
	"place": "",
	"fingerprint": [{
	"mac": "2C:3E:CF:0B:78:BF",
	"rssi": "-37",
	"ssid": "eduroam",
	"channel": 1
	}, {
	"mac": "2C:3E:CF:0B:78:B4",
	"rssi": "-28",
	"ssid": "eduroam",
	"channel": 1
	}, {
	"mac": "2C:3E:CF:0B:78:BB",
	"rssi": "-94",
	"ssid": "",
	"channel": 1
	}]
	}

Resposta (formato)	OK – Em caso de sucesso 400 Bad Request – Em caso de insucesso

Descrição: Envio de *fingerprints* para o servidor.

28. Introduzir local que pertence à Universidade do Minho

Método HTTP	**POST**
URI	{server}/places/um
Token como Cookie	Sim
Parâmetros	• Int idUser – identificador do utiizador • String idDevice – Endereço MAC da interface Wi-Fi do dispositivo do utilizador; • Array fingerprint: String MAC – endereço MAC de cada AP;

Int RSSI – nível de sinal do AP em dBm;

Int channel – canal associado à frequência de funcionamento do AP;

String SSID – nome da rede do AP;

- Array location:
  String campus – nome do campus;

  Int campusID – identificador do campus;

  String building – nome do edifício;

  Int buildingID – identificador do edifício;

  String floor – nome do piso;

  Int floorID – identificador do piso;

  String room – nome do quarto/espaço;

  Int roomID – identificador do quarto/espaço;

  Long lng – latitude;

  Long lat – longitude;

  Int location_changed – tem o valor 0 ou 1 dependendo da alteração da localização.

***Payload* pedido**


```
{
"idUser": "171",
"idDevice": "64:BC:0C:7F:AC:7B",
"timestamp": "2016-03-21 09:43:00",
"fingerprint": [{
        "mac": "d4:d7:48:0c:8d:40",
        "rssi": "-70",
        "ssid": "eduroam",
        "channel": 11
}, {
        "mac": "00:3a:98:ef:fa:50",
        "rssi": "-62",
        "ssid": "eduroam",
        "channel": 1
}, {
        "mac": "b4:14:89:d1:b3:8b",
        "rssi": "-95",
        "ssid": "",
        "channel": 36
}],
"location": {
        "campus": "Campus de Azurém",
```

```
                    "campusID": "1",
                    "building": "Bloco A",
                    "buildingID": "1",
                    "floor": "1",
                    "floorID": "1",
                    "room": "teste",
                    "roomID": "570",
                    "lat": "",
                    "lng": "",
                    "location_changed": "1"
            }
}
```


NOTA:

roomID:"new" > quando se pretende inserir um novo local não definido até ao momento.

Resposta (formato)	OK – Em caso de sucesso 400 Bad Request – Em caso de insucesso

Descrição: Pedido enviado quando o utilizador adiciona um novo local associado à UM. Além da informação do local é enviado uma *fingerprint*.

29. Introduzir local fora da Universidade do Minho

Método HTTP	**POST**
URI	{server}/places
Token como Cookie	Sim
Parâmetros	• Int idUser – identificador do utiizador • String idDevice – Endereço MAC da interface Wi-Fi do dispositivo do utilizador; • Array fingerprint: String MAC – endereço MAC de cada AP; Int RSSI – nível de sinal do AP em dBm; Int channel – canal associado à frequência de funcionamento do AP; String SSID – nome da rede do AP; • Array location: String campus – nome do campus; Int campusID – identificador do campus;

	String building – nome do edifício; Int buildingID – identificador do edifício; Long lng – latitude; Long lat – longitude; Int location_changed – tem o valor 0 ou 1 dependendo da alteração da localização.

***Payload* pedido**	{
	"idUser": "171",
	"idDevice": "64:BC:0C:7F:AC:7B",
	"timestamp": "2016-03-21 09:52:00",
	"fingerprint": [{
	"mac": "d4:d7:48:0c:8d:40",
	"rssi": "-58",
	"ssid": "eduroam",
	"channel": 11
	}, {
	"mac": "00:3a:98:ef:fa:50",
	"rssi": "-60",
	"ssid": "eduroam",
	"channel": 1
	}, {
	"mac": "b4:14:89:d1:b3:8b",
	"rssi": "-94",
	"ssid": "",
	"channel": 36
	}],
	"location": {
	"country": "Portugal",
	"city": "Guimarães",
	"address": "Largo da Oliveira",
	"place": "",
	"lat":" 41.442784",
	"lng":", -8.292726",
	"location_changed":"1"
	}
	}

Resposta (formato)	OK – Em caso de sucesso 400 Bad Request – Em caso de insucesso

Descrição: Pedido enviado quando o utilizador adiciona um novo local fora da UM. Além da informação do local é enviado uma *fingerprint*.