Universidade de Aveiro Departamento de Eletrónica, Telecomunicações
2018 e Informática

Tiago Alexandre Lucas de Bastos

Vitals Recorder: sistema móvel para apoiar a realização de estudos de psicofisiologia

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Ilídio Castro Oliveira, e coorientação do Doutor José Maria Fernandes, Professores Auxiliares do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

Dedico este trabalho à minha Tia Natércia.

o júri

presidente

Professor Doutor José Luís Guimarães Oliveira
Professor Associado com Agregação do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

Professor Doutor Ricardo Alexandre Peixoto Queirós
Professor Adjunto do Instituto Politécnico do Porto

Professor Doutor Ilídio Castro Oliveira
Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

agradecimentos

Gostaria de agradecer em primeiro lugar ao Professor Ilídio Oliveira, orientador deste projeto, por toda a disponibilidade e apoio prestado, pelos conselhos e orientação fornecida.

Agradecer também a todos os voluntários que realizaram os testes da aplicação, e aos colegas de laboratório que sempre se mostraram disponíveis a ajudar em alguns momentos de dúvida na parte de desenvolvimento do projeto.

Por último agradecer à minha família, em especial aos meus pais, por me proporcionarem as condições necessárias para concluir esta etapa académica.

palavras-chave

Biossinais; psicofisiologia; agregação de dados de sensores; aplicações móveis; Android.

resumo

A realização de estudos experimentais com recolha de dados fisiológicos dos participantes, obriga frequentemente os investigadores a usar equipamentos "fechados", com poucas possibilidades de sair do laboratório ou adaptar os protocolos de recolha.

O objetivo deste trabalho consiste no desenvolvimento de um sistema composto maioritariamente por aplicações móveis, para apoiar a recolha e agregação de dados fisiológicos, usando dispositivos acessíveis. Para além de permitir a utilização de um leque extensível de sensores de recolha, a solução deverá permitir aos investigadores monitorizar as experiências em curso, especialmente as recolhas realizadas em grupo, com vários participantes.

O sistema desenvolvido, Vitals Recorder, inclui dois módulos principais: uma aplicação Android para a recolha dos dados fisiológicos, que corre num *smartphone* associado a um participante (VR-Unit); uma aplicação de monitorização, que corre num *tablet* associado ao investigador (VR-Remote). Neste módulo, o investigador pode gerir o grupo de participantes, marcar eventos de interesse e inspecionar, em tempo real, os dados dos vários participantes. Os dados das experiências são consolidados num *backend*, que permite a exportação e pré-visualização na web.

As aplicações desenvolvidas foram utilizadas em estudos-piloto de psicofisiologia, com sessões de grupos, recolhendo ECG, EDA, Áudio e eventos marcados pelo utilizador.

A solução desenvolvida permite uma fácil extensão para incluir novos sensores, e facilita diferentes tipos de protocolos nos estudos de psicologia (individuais, grupos). Para além disso, pode ser usada em novos domínios, como a aquisição de dados fisiológicos de bombeiros no terreno ou desportistas praticantes de *fitness*.

keywords Bio-signals; psychophysiology; sensor data aggregation; mobile devices; Android.

abstract

The acquisition of physiological data collection in experimental research studies is often dependent on proprietary, closed equipment, with little chance for out of the lab observation or adaption the acquisition protocols.

The objective of this work is to develop a system composed mainly by mobile components to support the collection and aggregation of physiological data using accessible devices.

The solution should make it easy to extend the array of supported sensors and monitor ongoing group experiments with several participants.

The developed system, Vitals Recorder, includes two main modules: an Android application for the collection of physiological data, running on a smartphone associated with an individual (VR-Unit); a monitoring application that runs on a tablet associated with the coordinating researcher (VR-Remote). In this module, the researcher can manage the group of participants, mark events of interest and inspect, in real time, the data of the various participants. Data from different sessions is consolidated in a backend, which allows previewing on the web and export to convenient formats.

The developed applications were used in pilot studies of psychophysiology, with group sessions, collecting ECG, EDA, Audio and marked events from the researchers.

The solution can be extended extend to incorporate new sensors and facilitates different types of protocols in psychology studies (individual, groups). In addition, it can be used in new domains, such as the acquisition of physiological data from firefighters on the field or fitness scenarios.

Índice

Lista de Figuras

Lista de Tabelas

1 Introdução

Desde a invenção do primeiro telemóvel em meados do século passado que esta tecnologia não tem parado de evoluir. Foi devido a esta evolução que hoje em dia as pessoas conseguem transportar consigo um ou mais dispositivos deste género, que utilizam sobretudo para realizar chamadas, enviar mensagens, ou estarem ligadas com o "mundo" da web através das mais diversas aplicações. Esta explosão de tecnologia nos dispositivos móveis contribuiu para a adesão em massa aos *smartphones*, criando uma dependência geral da população em geral a estes dispositivos, que nem sempre é vista com bons olhos [1].

A mobilidade oferecida pelos dispositivos, para além de permitir a cada um de nós levar um minicomputador no bolso, permitiu também o desenvolvimento de sensores de tamanho reduzido, mas capazes de captar as mais diversas informações do mundo real. A informação recolhida, pode assim ser armazenada, estudada e utilizada de modo bastante proveitoso para as mais diversas áreas de investigação, tornando os telemóveis capazes de recolher diversos dados através da utilização do leque de sensores com que estão equipados ou com que se podem ligar [2].

Uma das áreas que pode beneficiar de maneira positiva da criação de aplicações móveis especializadas é a psicofisiologia. Nesta área, existe uma lacuna de *setups* móveis e ecológicos, que permitam aos investigadores realizar livremente as suas experiências sem depender do laboratório, ou de material estacionário presente no mesmo. Este tipo de recolhas podia beneficiar caso existisse um conjunto de aplicações gratuito e extensível que oferecesse um ponto de controlo único sobre os vários dispositivos móveis que tratam da recolha, permitindo orquestrar e monitorizar toda a recolha a partir de um único ponto.

1.1 Motivação

A maior motivação que levou ao aparecimento deste trabalho prende-se com a necessidade de criar uma forma de facilitar o trabalho do investigador. Ele tem a tarefa de gerir todo o grupo de participantes, bem como garantir o controlo das variáveis intervenientes, durante a recolha.

A melhor maneira de seguir o estado do grupo de forma a avaliar o decorrer da experiência é de facto monitorizá-lo de forma contínua. É também importante ter maneira que esses dados sejam armazenados para

que no período pós-experiência seja possível estudar e detetar padrões sobre o comportamento da pessoa durante a experiência de modo a poder chegar a conclusões.

De forma a cumprir estes requisitos, e de forma a que também não existissem custos avultados na preparação e execução de uma experiência de psicologia foi pensada uma solução baseada em aplicações para dispositivos móveis. Essas aplicações, deverão permitir a agregação de dados de sensores internos e também externos, na qual o responsável pela experiência poderá controlar, monitorizar e configurar as recolhas para os seus estudos.

Este trabalho vai de encontro ao âmbito do trabalho do grupo de investigação bitMob da unidade de investigação IEETA da Universidade de Aveiro [3], que tem colaborado na recolha de dados fisiológicos em experiências de psicologia realizadas no Departamento de Educação e Psicologia da Universidade de Aveiro. O grupo dispõe de ferramentas que carecem de um processo de reengenharia, de modo a aplicar tecnologias mais recentes no âmbito da computação móvel. Também existe a necessidade de tornar o sistema mais fiável e resistente a falhas em cenários de recolha, numa perspetiva de manutenção evolutiva e corretiva.

1.2 Objetivos

O objetivo principal do trabalho em questão é o de criar um conjunto de aplicações que permitam substituir em pleno as aplicações móveis existentes no projeto *Vr2Market*[4], que eram utilizadas até agora para a recolha de dados fisiológicos em cenários de experiências de psicofisiologia, ou em recolhas de dados em teatros de operação de bombeiros. Os objetivos deste trabalho foram os seguintes:

- Desenvolvimento de uma aplicação de controlo capaz de reformular os processos de monitorização e controlo da experiência de grupo, de modo a permitir ao responsável pelas recolhas um acompanhamento detalhado de todo o ambiente experimental, fornecendo visualização individual de cada dispositivo em tempo real;
- Permitir a integração de novos sensores na plataforma, de forma fácil;
- Reengenharia das implementações atuais das aplicações existentes de modo a adequar as tecnologias utilizadas aos padrões mais modernos de desenvolvimento em computação móvel.

O conjunto de aplicações do sistema de recolha, deverá ser suficientemente genérico e extensível de modo a poder ser aplicável a outros cenários, como missões de salvamento, ou outro tipo de cenários de controlo de grupo que não sejam de psicofisiologia.

1.3 Estrutura da Dissertação

O capítulo 2 foca-se na introdução ao contexto do problema e também às aplicações móveis e tecnologias associadas ao desenvolvimento de aplicações. Serão revistos conceitos sobre bio sinais, bem como será discutido um estudo sobre padrões de arquitetura em aplicações móveis.

No capítulo 3 será abordado o sistema de recolha baseado em aplicações móveis pré-existentes, e a sua avaliação.

No capítulo 4 serão explicados os requisitos do sistema, de modo a serem desenvolvidas aplicações móveis que se enquadrem nas necessidades dos atores do problema.

No capítulo 5 será explicada a arquitetura geral do sistema, comunicação entre componentes, e será abordado o modelo de dados do problema.

No capítulo 6 será explicada em detalhe a implementação de todo o sistema, bem como metodologias de testes que foram aplicadas.

No capítulo 7 serão apresentados os resultados, onde serão analisados os resultados dos testes de usabilidade realizados no período de teste da versão piloto do sistema. Também será realizada uma avaliação geral ao trabalho desenvolvido numa comparação com as soluções previamente existentes.

Na conclusão, capítulo 8, serão discutidos os resultados, e analisado o trabalho realizado e o trabalho futuro que será possível fazer de modo a evoluir o sistema.

2 Estado da Arte

2.1 Sistemas móveis ao serviço da psicofisiologia experimental

Os dispositivos móveis têm uma capacidade elevada de produzir e angariar dados, que podem ser armazenados e exportados de modo a ser analisados. Esta capacidade, permite associar uma expressão a estes dispositivos "*Small Devices, Big Data"* [5]. A evolução destes dispositivos possibilitou a criação de novas oportunidades aos investigadores, permitindo reunir maior quantidade de dados [5]. A capacidade dos dispositivos móveis de captarem o "contexto" envolvente do utilizador através da utilização dos seus sensores permite aumentar um leque de oportunidades na perceção do impacto do contexto que envolve o utilizador no seu comportamento [6].

A psicofisiologia é o ramo da psicologia que estuda as relações entre os fenómenos fisiológicos e os psicológicos, logo a obtenção de dados fisiológicos nestes cenários é essencial. Antes do *boom* da tecnologia, o método de recolhas de dados de participantes por parte dos investigadores era baseado em questionários. Outro dos métodos de recolha de comportamento como a colocação dos participantes em cenários hipotéticos ou em cenários controlados dentro de laboratórios. No entanto estes métodos acabavam por não ser naturais e também fora do contexto das práticas normais do dia a dia dos participantes [2], resultando em dados que podiam não ser inteiramente fiáveis, devido à pouca quantidade de dados angariados sobre os comportamentos naturais das pessoas.

Com a evolução dos smartphones, e métodos de captação associados a estes dispositivos, neste caso vários sensores que foram aparecendo ao longo dos anos com capacidades de recolher as mais diversas variáveis (ambientais, localização e movimento ou monitorização do corpo humano), é possível aos investigadores captarem dados mais naturais e objetivos, que não sejam tão intrusivos para os participantes visto que não é necessário criar cenários fictícios para realizar a monitorização do comportamento [2]. Para além dos pontos enumerados anteriormente, o aparecimento dos *smartphones* e das aplicações móveis dedicadas a psicologia, permitiram a algumas destas experiências "sair" do laboratório, ou da sala experimental, levando ao aparecimento de experiências diferenciadas das existentes, onde é possível estar a recolher dados sobre os participantes na sua rotina diária, ou até mesmo em experiências "ao ar livre". É possível concluir então, que a tecnologia, mas em especial os dispositivos móveis, alteraram as metodologias

de recolha utilizadas neste tipo de experiências, mas não só em cenários de psicologia ou psicofisiologia, todos as ciências que estudem comportamentos podem beneficiar deste tipo de tecnologias [7].

2.1.1 Arquiteturas tipo

A arquitetura destes sistemas móveis pode ser dividida em 3 componentes, o *smartphone* e um conjunto de sensores externos que se emparelham com ele, e um conjunto de componentes de *backend* que poderão suportar armazenamento, análise, processamento e visualização de dados. O *smartphone* por si só possui um conjunto de sensores internos que podem ser utilizados, o seu controlo e captação de dados pode ser feito através da utilização das *API* fornecidas pela plataforma do sistema operativo móvel[8].

Pode-se então associar este tipo de arquiteturas à típica arquitetura cliente-servidor, onde os clientes serão os *smartphones*, e existe um servidor que terá sobre si a responsabilidade sobre as funcionalidades *core* dos *setups*. Uma arquitetura típica utilizada num estudo de comportamento, aplicado a cirurgiões e profissionais médicos [9], encontra-se esquematizada na Figura 2-1. Neste estudo, cada cirurgião ia estar acompanhado por um *smartphone*, capaz de recolher dados de batimento cardíaco, frequência, movimento e temperatura corporal com sensores colocados no seu corpo. A aplicação utilizada enviava os dados recolhidos para o servidor, que tinha a funcionalidade de processar, analisar e armazenar os dados.

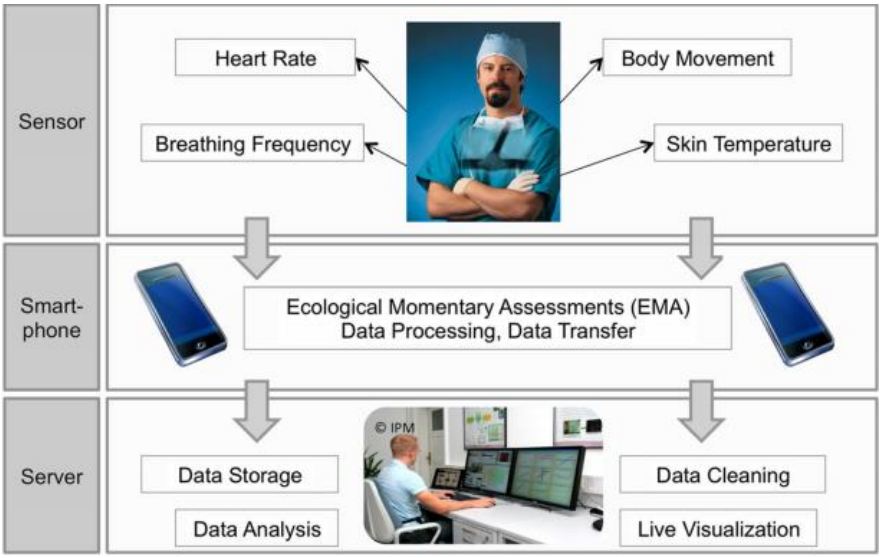


Figura 2-1 - Arquitetura geral de um setup associado a um estudo psicofisiológico. [9]

Como equipamento auxiliar, mas ainda importante no desenrolar das recolhas, podem ser utilizados questionários em papel, tipicamente utilizados em recolhas de laboratório. Atualmente, várias aplicações permitem que a resposta a questionários seja efetuada na própria aplicação, contendo mecanismos de *upload* de respostas, que *o backend* terá de processar e armazenar. Nestes casos, a arquitetura irá alterar-se ligeiramente, de modo a permitir a capacidade de resposta e armazenamento a questionários.

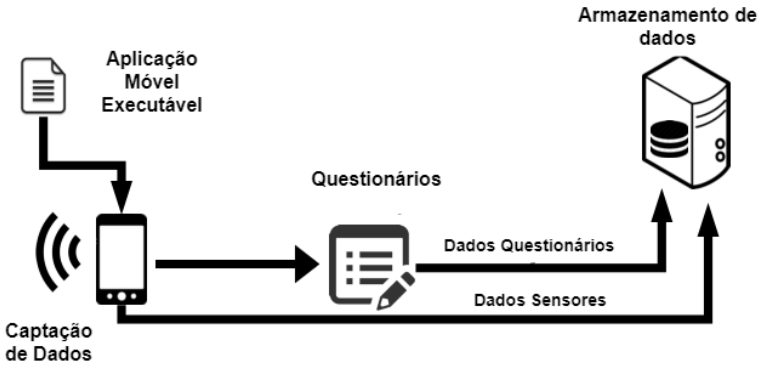


Figura 2-2 - Arquitetura típica com suporte de questionários. [10]

Quando são utilizados sensores externos, de forma a comunicar com os mesmos, tipicamente são utilizados 3 tipos de protocolos de comunicação: *Bluetooth*, *Zigbee* ou *WiFi* [11][7][12][13]. O estabelecimento e a comunicação em si, podem ser facilitados se forem fornecidos aos programadores *API* ou *SDK* para o desenvolvimento de código de comunicação e recolha de dados com os mais diversos sensores. Dado o protocolo de comunicação utilizado, a distância do *smartphone* pode variar, sendo que alguns sensores possam estar mais limitados nesse sentido devido à versão do protocolo que suportem.

O *smartphone* estará tipicamente associado a um único participante num *setup* de equipamento comum de um estudo que utilize sistemas móveis como fonte de recolha de dados, permitindo assim a agregação de vários tipos de variáveis através da utilização dos sensores internos e de alguns sensores [10].

Figura 2-3 - Exemplo de arquitetura móvel tipo

Associado a este tipo de setups existem sempre decisões a serem tomadas, cada decisão poderá variar consoante o estudo que se quer fazer visto que terão implicações no resultado final de cada experiência. As escolhas mais importantes são as seguintes[2]:

- Duração do estudo/recolha;
- *Sampling rate* de recolha dos sensores de captação de dados;
- Que tipo de dispositivo móvel usar (plataforma, tamanho, marca, etc..);

- Que aplicação utilizar (aplicações pagas, gratuitas, que suportem determinados sensores, desenvolver uma app específica para a recolha);
- Sensores a utilizar e que dados aproveitar;
- Espaço necessário para armazenamento (servidor na cloud, servidor local, base de dados ou armazenamento em ficheiro);
- Que tipo de armazenamento utilizar (Bases de dados relacionais, não relacionais);
- Processos de análise e gestão de dados (algoritmos e software a utilizar).

2.1.2 Aplicações existentes

Atualmente existem diversas aplicações e plataformas dedicadas que se podem encaixar neste tipo de *setups* experimentais, essas aplicações dividem-se em aplicações do tipo *Desktop* e *Mobile*. No entanto a maioria delas apresenta várias lacunas a nível de suporte de sensores, visto que muitas delas estão preparadas para apenas um sensor, ou apenas utilizar sensores internos do telemóvel. Vários exemplos dessas aplicações serão analisados seguidamente.

MyExperience

Aplicação desenvolvida para *Pocket PC* que permite recolha e armazenamento de dados de contexto, permite também o acréscimo de sensores para além dos suportados pelo dispositivo através da sua arquitetura orientada a *plugins*[15]. Para além de sensores esta aplicação também é capaz de captar dados de imagens, vídeos ou questionários. Esta aplicação foi utilizada em vários estudos de comportamento, sendo destacado no *website* do produto vários deles como por exemplo, um estudo que investiga o batimento cardíaco e a sua variação em sessões de gestão de stress/raiva, ou, numa investigação da relação entre os hábitos de turismo de cada um e as suas preferências por esses locais.

AndWellness

Plataforma que permite a recolha de dados e respostas de questionários a utilizadores, sobretudo utilizada em experiências de comportamento. É composta por uma aplicação *Android* que permite a recolha de dados de sensores internos ao telemóvel e também suporta alguns externos como sensores de monitorização de batimento cardíaco, pressão sanguínea, entre outros [16]. Esta plataforma também é composta por uma página *web* que permite a visualização dos dados recolhidos pelos sensores e questionários, contendo ainda ferramentas de análise básica que podem ajudar a descobrir correlações entre os dados.

EmotionSense

Aplicação *Android* utilizada em estudos sociais de psicologia. É possível medir a atividade física de cada pessoa, mas também a proximidade a outras pessoas através de tecnologia presente em *smartphones* recentes. Utiliza sensores internos do telemóvel e o *Bluetooth* do telemóvel de modo a aperceber-se da proximidade a outros dispositivos[17]. Internamente, é composta por diferentes componentes, existindo um conjunto de sensores que recolhe dados independentemente, orquestrados por um componente denominado de *EmotionSense Manager*. Os dados recolhidos por cada sensor são processados e analisados internamente, onde através da utilização de estratégias de análise de dados, permite inferir alguns padrões sobre os utilizadores da aplicação [17].

OhMage

Plataforma que permite a criação de estudos, recolha e análise de dados de *Participatory Sensing*. São recolhidos dados de localização temporal e espacial, dados de movimentação, controlo de sono, movimento e áudio dos participantes através das aplicações móveis. Os estudos, isto é, questionários e tipos de dados a recolher, são configuráveis através de uma plataforma *web* que também permite visualização de resultados dos mesmos para pós-análise e consulta[18]. No *website* do produto, é possível ver uma extensa lista de estudos onde a plataforma foi utilizada como ferramenta de investigação, no ensino ou em estudos médicos, que abrange um leque de estudos de depressão, estudos de exercício físico ou comportamentais. [1]

Purple Robot

Aplicação *Android* desenvolvida para a criação de experiências dependentes do contexto [19]. Permite recolher dados de sensores internos do telemóvel como sensores de movimento, localização, estado meteorológico atual, também permite a ligação a outros dispositivos *Bluetooth*. Existe ainda uma cifragem dos dados recolhidos no momento de envio para o servidor de armazenamento. Esta aplicação tem sido utilizada de modo a detetar depressão nos seus utilizadores, para isso, para além da recolha de dados do contexto, esta aplicação liga-se à rede social *Facebook*, de modo a recolher algumas informações que possam ser úteis na estimativa do estado da depressão [20].

OpenDataKit

Ferramenta de agregação de dados e análise de resultados. É composta por uma aplicação *Android*, *ODK Collect*, que permite a recolha de dados e resposta a questionários, os dados recolhidos podem ser visualizados e exportados na ferramenta *ODK Aggregate*, e os dados dos questionários analisados e exportados na ferramenta *ODK Briefcase*. Permite a criação de questionários através da ferramenta *ODK*

[1] http://ohmage.org/research.html

XLSForm, que corre num *browser* [21]. A análise de resultados na aplicação pode ser efetuada numa página *web* fornecida ao utilizador. Esta ferramenta apesar de não ser diretamente utilizada em estudos de comportamento ou outro tipo de estudos de psicofisiologia, contém todas as capacidades para permitir a um investigador organizar um *setup* de recolha e análise de dados [10].

Na Tabela 2-1, será possível ver um quadro-resumo de alguns dos parâmetros e observações efetuadas sobre os sistemas abordados anteriormente.

Aplicação	**Multiplataforma**	**Sensores Internos**	**Sinais Vitais**	**Outros Sensores**	**Obs.**	**Open Source**
MyExperience	Não (Pocket PC)	GPS	Sim	Permite adicionar	Descontinuado (2009 última referência)	Não
AndWellness	Não (Android)	GPS, acelerómetro, câmara	Sim	Não permite adicionar	Descontinuado (2011 última referência)	Não
EmotionSense	Não (Android)	GPS, Sensor de proximidade, Acelerómetro, Microfone	Não	Não permite adicionar.	-------	Sim
OhMage	Sim (desktop incluído)	Acelerómetro, GPS, Wifi.	Não	Não	-------	Não
PurpleRobot	Não (Android)	GPS, Acelerómetro	Não	Sim	-------	Sim
OpenDataKit	Não(Android)	GPS	Não	Não permite adicionar	-------	Sim

Tabela 2-1 - Tabela de sumário das aplicações existentes.

2.1.3 Biossinais comuns e outras variáveis

Os biosinais ou sinais fisiológicos são sinais produzidos pelos seres vivos. Esses sinais podem ser medidos e monitorizados de modo a analisar detalhadamente quem os produz. Os mais comuns são o EEG (Eletroencefalograma), ECG (Eletrocardiograma), EMG (Eletromiograma), EOG (*Electrooculography*) e o EDA (*Electrodermal Activity)*. De entre estes biosinais, um dos mais usados em estudos é o ECG, devido à sua proximidade a doenças cardiovasculares, dado que é responsável por captar a atividade elétrica do coração durante um período de tempo[22].

O ECG é o sinal que apresenta mais instrumentos capazes de o ler com precisão adequada. Tradicionalmente, este sinal é detetado através de equipamento profissional médico em que são ligados elétrodos ao paciente, no entanto este equipamento é pouco portável, o que significa que as atividades físicas do paciente são limitadas durante a recolha destes sinais. Com o aparecimento de sensores, e de *smartphones* com capacidade para se ligar a estes, começaram a surgir mecanismos de monitorizar sinal de *ECG* de uma maneira mais prática, portável e comportável economicamente, como por exemplo a incorporação deste tipo

de sensores em *smartbands*, *armbands*, ou algum vestuário[23]. Um exemplo de traçado e de equipamento está presente na Figura 2-4.

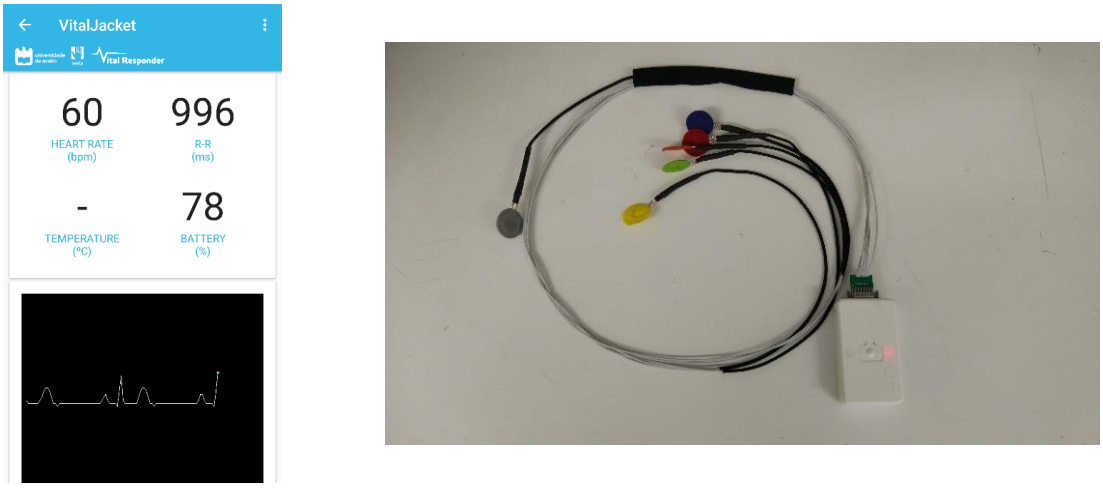


Figura 2-4 - Traçado ECG recolhido numa aplicação móvel e equipamento VitalJacket.

O sinal EEG, capta a atividade elétrica do cérebro. Dado esta especificidade, inicialmente a recolha deste tipo de sinal era estacionária e efetuada num laboratório com equipamento adequado para tal[13]. Apenas recentemente começaram a surgir equipamentos com capacidade de recolher este tipo de sinais de forma *wireless*, devido ao aumento da capacidade de processamento dos *smartphones* e também ao aparecimento de sensores de baixo custo que consigam captar sinais deste tipo. Com este avanço tecnológico, é possível captar EEG com a colocação de um conjunto de elétrodos sobre o crânio do paciente e mostrar num *smartphone* a imagem de atividade cerebral[12]. Na Figura 2-5 é possível ver um *setup* exemplificativo da recolha de EEG, com um capacete composto por múltiplos elétrodos.

Figura 2-5 - Setup de recolha de EEG.[2]

O sinal EMG capta a atividade elétrica muscular de cada pessoa, os sistemas de recolha deste tipo de sinal assemelham-se ao de ECG, apesar de internamente serem muito diferentes, por exemplo o sinal de ECG

[2] http://www.eegelectrodecap.com

tem uma largura de banda utilizável de cerca de 100Hz enquanto o de EMG possui componentes que a estendem até aos 500Hz. Portanto, a recolha deste sinal também faz uso de elétrodos colocados sobre a pele do paciente [24]. Na Figura 2-6 encontra-se um *setup* de recolha de EMG, com 3 elétrodos colocados sobre o braço da pessoa.

Figura 2-6 - Setup de recolha de EMG.[3]

O sinal EDA, corresponde a um fenómeno elétrico captado pelo suor produzido pelo corpo humano, pode ser causado pela excitação mental ou tensão na pessoa [25]. Mais propriamente na psicologia, o EDA é utilizado de modo a medir o grau de impacto de situações de stress, tensão ou excitação [26]. Existem dois métodos para medir o valor de EDA através de eletrificação ou potenciodinâmica, no método da eletrificação, a alteração da impedância da pele é medida através da eletrificação da pele, com o método da potenciodinâmica é medida a diferença de potencial da pele diretamente [25]. Na Figura 2-7 é possível ver um *setup* de recolha de EDA, com dois elétrodos colocado sobre os dedos do participante.

Figura 2-7- Setup de recolha EDA, equipamento da BioPac.[4]

No decorrer das experiências de psicofisiologia, nem são só os dados fisiológicos que são relevantes. Outros dados como áudio, vídeo, ou de localização poderão ser importantes de ser estudados e agregados no decorrer das recolhas.

[3] https://www.seeedstudio.com
[4] https://www.biopac.com

Áudio

A captação de áudio pode-se tornar relevante em cenários de psicologia, a sua gravação pode ser útil em cenários de angariação de respostas através da voz ao invés dos típicos questionários, este tipo de procedimentos pode ser muito útil com analfabetos, ou com crianças [7]. Os microfones utilizados, poderão ser de um *smartphone*, ou com equipamento de som dedicado como microfones.

Vídeo

A gravação de vídeo é muito útil no cenário de recolhas, onde com uma só câmara, ou conjunto de câmaras, é possível captar visualmente o comportamento dos participantes[7]. Tecnologias mais avançadas como câmaras com *eye-tracking*, ou aplicações que detetem em tempo real dados sobre a expressão facial, ou até mesmo a fotografia térmica da pessoa, fornecendo fontes de dados diferenciadas aos investigadores[7][27].

GPS

O sinal de *GPS* define uma posição com coordenadas geográficas num determinado momento no tempo. Estes dados poderão ser relevantes em estudos fora de laboratório, no estudo de rotinas diárias de uma pessoa, onde o seu trajeto diário poderá fornecer medidas importantes no comportamento e interações sociais, permitindo saber como uma pessoa é afetada por trânsito, tempo local, ou outro tipo de eventos[7].

Acelerómetro e outros sensores internos

O acelerómetro, sensores de luz e proximidade, barómetros, giroscópios, são normalmente parte integrante do leque de sensores internos de um *smartphone*. Individualmente, ou em conjunto, permitem recolher dados que podem servir de auxílio em estudos, mais propriamente recolhendo dados dos comportamentos das pessoas no seu dia-a-dia. O acelerómetro poderá indicar se uma pessoa está parada ou em movimento, o barómetro indicar se uma pessoa está a descer ou a subir. Todos estes sensores, em conjunto com mais alguns que possam ser incorporados em telemóveis futuros, como sensores ambientais, poderão fornecer dados importantes em estudos comportamentais de pessoas no dia-a-dia[7].

2.1.4 Revisão de alguns sensores selecionados

A fisiologia é o ramo da biologia que estuda o funcionamento físico, orgânico, mecânico e bioquímico dos seres vivos. Os sistemas móveis estão diretamente relacionados com este conceito, através da sua capacidade para comunicarem com sensores que permitem o estudo destas variáveis, permitindo a análise da fisiologia do corpo humano de cada participante em experiências. De modo a agregar as variáveis associadas

a fisiologia [2] e outras que façam sentido em cenários particulares, existem vários sensores, muitos dos quais internos ao *smartphone* tais como as abordadas na Tabela 2-2.

Sensor	Variáveis agregadas
GPS	Localização (latitude, longitude), altitude, marca temporal, fornecedor de sinal
Sensores de humidade	Percentagem de humidade [5]
Sensores de pressão	Nível de pressão atmosférica (bar) [1]
Sensores de temperatura	Graus centígrados/fahrenheit [1]
Sensores de luminosidade	Luminância (lx) [1]
Sensores de proximidade	Distância a objetos [6]
Microfone	Áudio
Câmara	Fotografia e Vídeo
Giroscópio	Orientação do dispositivo [14]
Acelerómetro	Aceleração (m/s^2) [14]
Magnetómetro	Força do campo magnético da terra (T) [14]

Tabela 2-2 - Quadro de sensores internos e variável recolhida.

Também é possível conectar ao dispositivo móvel vários sensores externos, através de tecnologias como o *Bluetooth*, *WiFi*, *Zigbee*, entre outras tecnologias. Com esse aglomerado de sensores, é possível utilizar os sistemas móveis como entidade principal no objetivo de compreender não só o contexto onde se encontram, bem como recolher dados sobre uma ou várias pessoas [8]. Nesta sub-secção irão ser abordados alguns dos sensores disponíveis para este tipo de recolha de dados, bem como os biosinais usualmente captados e também algumas aplicações existentes.

Revemos, de seguida, alguns sensores com interesse para um sistema como o que se pretende desenvolver.

VitalJacket

O *VitalJacket* é um sensor que faz a monitorização de sinais vitais, num sistema que junta uma camisola e uma caixa onde está assente toda a eletrónica do dispositivo. O sensor foi utilizado em cenários clínicos, em hospitais, em situações de resgate e incêndios [28]. No grupo bitMob, foi utilizado recorrentemente no âmbito do projeto *VR2Market* [7].

Este sensor permite uma monitorização de sinais vitais (ECG, Batimento cardíaco, temperatura corporal) através dos elétrodos presentes na camisola que fazem a recolha desses mesmos sinais, visível na Figura 2-8. O sensor permite o armazenamento offline de todos os dados recolhidos durante a utilização, permitindo recolhas com a duração de 5 dias [29]. Para comunicar com o dispositivo, a *BioDevices*, empresa

[5] https://developer.android.com/guide/topics/sensors/sensors_environment
[6] https://developer.android.com/guide/topics/sensors/sensors_position
[7] http://vitalresponder.web.ua.pt/

que distribui o equipamento, fornece um SDK que permite controlar, monitorizar e adquirir dados do dispositivo através de *Bluetooth.*

Figura 2-8 - Componentes do sistema VitalJacket

BITalino

O BITalino [8] corresponde a uma placa semelhante a um microcontrolador tal como um *Arduino*, mas com a particularidade de ser dedicada a biosinais. Com esta placa é possível adquirir ECG, EMG, EEG e EDA, a placa contém ainda como sensores ACC (acelerómetro) e um sensor de luz. A taxa de recolha pode ser ajustada entre 4 valores: 1, 10, 100 ou 1000Hz. [30]

Existem 3 configurações possíveis desta placa:

- *Board*: BITalino sem modificações, onde é possível utilizar os sensores internos da placa;
- *Plugged:* São adicionados módulos ao BITalino, de modo a ser possível conectar diversos tipos de sensores;
- *Freestyle:* Todos os blocos individuais estão fora da placa, permitindo construir versões personalizadas da placa.

A comunicação com placa é feita através de *Bluetooth*, sendo que a empresa que fabrica o dispositivo, fornece uma API disponível em várias linguagens de modo a realizar a comunicação e controlo dos sensores. Tendo em conta a comunicação estão disponíveis duas versões diferenciadas da placa, *BITalino BLE (Bluetooth Low Energy)* e *BITalino* BT (*Bluetooth*)[31], na Figura 2-9 é possível ver a versão *Board* da placa.

Figura 2-9 - Placa BITalino, versão Board

[8] http://bitalino.com

BioPac Advanced Eye Tracking System

A Biopac é uma empresa americana que distribui equipamento e software que permite a recolha de variáveis fisiológicas, fornecendo vários produtos utilizados no domínio da investigação, ensino e profissional.

O *Advanced Eye Tracking System* é um conjunto de *hardware* e *software* que permite montar um *setup* de *Eye-tracking* profissional, sendo composto por câmaras de elevada qualidade, um *software* próprio entre outros periféricos, visíveis na Figura 2-10. Com este equipamento de elevada precisão é possível ter uma visualização em tempo real do trajeto percorrido pelos olhos, bem como um histórico de pontos, onde também é possível ver o tamanho das pupilas, duração temporal na fixação da visão, entre outros [32].

Figura 2-10 - Equipamento constituinte do sistema de eye Tracking.

O *Eye-Tracking*, na psicologia, pode por exemplo ser utilizado em estudos de medição de atenção, que são tipicamente aplicados a pessoas com autismo, de modo a medir o nível de atenção social e motivação social [27].

Biopac MP160 Data Acquisition Systems

Este equipamento, utilizado no Departamento de Educação e Psicologia da Universidade de Aveiro, permite a aquisição de diferentes tipos de sinais fisiológicos. É um sistema flexível e modular de aquisição e também análise de dados podendo por exemplo ser estendido com pacotes de software e hardware. O *Starter Kit*[9] deste sistema é visível na Figura 2-11. Para além deste equipamento, as recolhas de dados são feitas com um conjunto de elétrodos ou equipamento alternativo, como recetores *wireless*, camisolas, câmaras de vídeo, microfones, que ligados ao equipamento MP160, permitem a sincronização dos dados recebidos na plataforma *AcqKnowled* [10].

Exemplos dos sinais fisiológicos que são possíveis de recolher com este sistema são: Pressão arterial, EDA, EMG, EEG, ECG [33].

[9] https://www.biopac.com/product-category/research/systems/mp150-starter-systems/
[10] https://www.biopac.com/product/acqknowledge-software/

Figura 2-11 - Sistema Biopac MP160.

2.2 Desenvolvimento de Aplicações Móveis

Os dispositivos móveis produzem um impacto significativo na vida das pessoas, tornando-se cada vez mais uma necessidade do que uma possibilidade. Com eles foi permitido a quem os possui interagir com todas as partes do mundo, quer seja para comunicar com a família ou com amigos, mas não só. Com um dispositivo móvel e uma ligação à internet abre-se um leque alargado de possibilidades oferecidas ao utilizador, como a consulta do seu e-mail, o acesso a redes sociais, o acesso a sistemas de localização, entre outros. Para além do acesso a todos estes serviços, o aparecimento dos dispositivos móveis permitiu o seu proveito ao nível da saúde podendo ajudar no tratamento de várias doenças ou até na descoberta das mesmas através do aparecimento de aplicações orientadas à área da saúde ou na área de fitness[34].

A evolução nos dispositivos móveis permitiu o aparecimento de sistemas operativos móveis, tornando os telemóveis quase tão capazes como os computadores possibilitando a instalação e o desenvolvimento de aplicações por programadores, mas com o benefício de terem um tamanho muito mais reduzido do que um computador pessoal. As grandes companhias aventuraram-se pelo mundo dos sistemas operativos móveis, sendo que em 2007 com o lançamento do primeiro *iPhone* foi juntamente lançada a primeira versão do *iOS*, este sistema operativo permitiu a criação de aplicações através da introdução do *SDK* fornecido pela *Apple*, externamente às aplicações básicas fornecidas pela companhia como o bloco de notas, o browser, o mapa, entre outras[35].

O *Android* provém de uma companhia formada em 2003 denominada de *Android Inc*, sendo que em 2005 foi adquirida pela *Google*[36]. A primeira versão do *Android* foi lançada em 2008 num T-Mobile G1, dispositivo com ecrã tátil e teclado QWERTY, contendo já algumas aplicações da *Google* e permitindo também o seu desenvolvimento. A *Microsoft* também tem o seu próprio sistema operativo móvel, atualmente denominado de Windows Phone 10, adquirindo o nome do seu correspondente em versão *desktop* ao longo dos anos, mas que até 2010 era denominado de Windows Mobile. Este sistema operativo irá ser descontinuado, esta decisão foi tomada pela *Microsoft* em outubro de 2017, e foi encorajada pela falta de vendas de telemóveis com o sistema operativo, bem como a dificuldade da companhia em atrair interesse dos programadores em desenvolver aplicações para dispositivos deste género.

Hoje em dia, para além de *smartphones*, os dispositivos móveis podem também ser *tablets*, pequenos microcontroladores que correm sistemas operativos Android ou até mesmo TV *Boxes* como a *Apple* TV ou qualquer box Android existente no mercado. Muitos destes dispositivos, utilizados no dia-a-dia, conseguem ter capacidades computacionais superiores a alguns computadores chegando-se a discutir a possibilidade de tablets com melhores especificações poderem começar a substituir computadores no mercado, ou até mesmo substituírem o uso de televisões[37].

O *Android* é com larga margem o sistema operativo móvel mais utilizado, seguido de *iOS*. As restantes alternativas praticamente não têm expressão de mercado quando comparadas com estas duas. Os dados do gráfico apresentado correspondem a uma utilização à escala global. Na Figura 2-12 é possível observar um gráfico de utilização das plataformas móveis existentes.

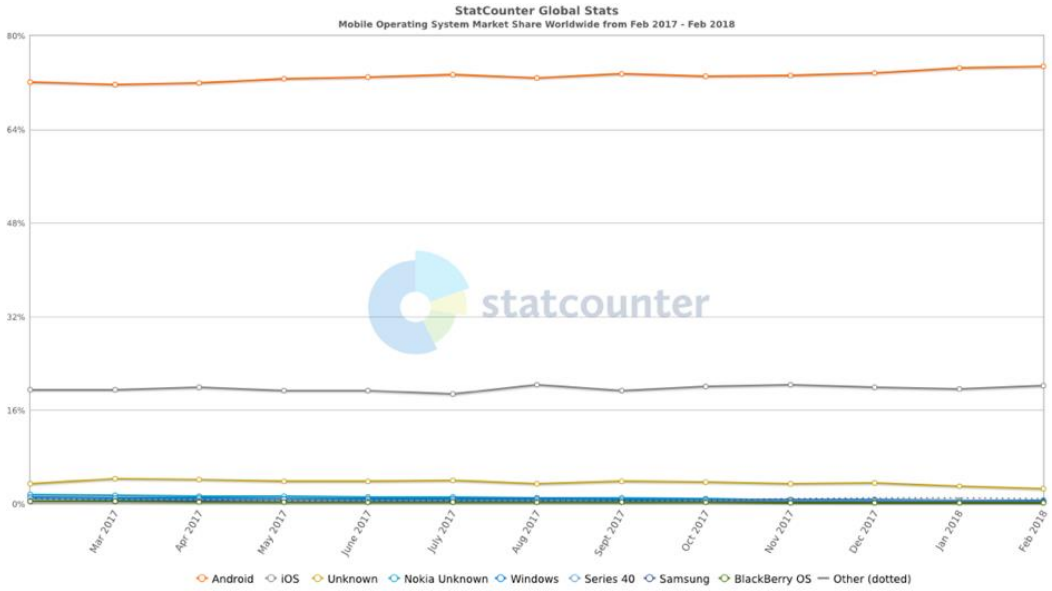


Figura 2-12 - Gráfico de utilização mundial de sistemas operativos móveis.[11]

2.2.1 Plataforma Android

A plataforma *Android* é o sistema operativo dominante no mercado dos dispositivos móveis [38], sendo utilizado não só em telemóveis mas também em tablets, TV *Boxes*, *SmartTV's*, *Smartwatches*, automóveis, micro controladores, entre outros dispositivos. O sistema operativo é *Open Source* e para além disso é baseado no sistema operativo *Linux*. A plataforma Android não é restrita a qualquer marca ou dispositivo, podendo ser livremente instalada por qualquer fabricante de telemóveis nos seus dispositivos, o que poderá ser um ponto positivo, bem como um ponto negativo visto que não existe um dispositivo standard para o qual o software esteja 100% otimizado para o hardware, ao contrário dos *iPhones.*

[11] http://gs.statcounter.com/os-market-share/mobile/worldwide

Para a criação de aplicações que corram neste sistema operativo, os programadores têm duas opções de escolha como linguagem para realizar um projeto, *Java* e *Kotlin* [39], as duas linguagens são interoperáveis, visto que o código *Kotlin* é compilado para código máquina *Java*, não tendo assim qualquer impacto de performance quando se desenvolve código em *Java* ou *Kotlin*. De modo a criar estes ficheiros e a fazer a instalação da aplicação em dispositivos físicos ou virtuais é fornecido um ambiente integrado de desenvolvimento denominado de *Android* Studio[12].

A criação de interfaces visuais no *Android* é independente do tamanho do dispositivo físico, sendo possível criar *layouts* dinâmicos, ou específicos para dispositivos de maior ou menor dimensão. De modo às interfaces visuais do *Android* serem de alguma forma unificadas, é fornecido aos programadores e aos designers um conjunto de normas padrão para o desenho e implementação de interfaces, que vai desde as cores dos elementos visuais ao tamanho e formato dos ícones[40][41].

2.2.2 Plataforma iOS

Este sistema operativo desenvolvido pela *Apple* apenas pode ser encontrado legalmente em dispositivos da *Apple*, tais como *iPhones*, *iPads* e *iPods*. O sistema operativo é baseado no MacOS, partilhando assim uma série de similaridades com o mesmo. Sendo um sistema muito mais restrito, o design da interface visual do *iOS* é muito mais "limpo" do que o Android, não havendo possibilidade de fazer grandes alterações ao *look and feel* dos grafismos, também é caracterizado pelo conceito de manipulação direta utilizando gestos multi-toque.

As restrições dos equipamentos da *Apple* e dos seus sistemas operativos sempre foram conhecidas, e o desenvolvimento de aplicações para *iOS* não escapa a essa regra, o desenvolvimento apenas é possível com um computador que corra MacOS, onde é necessário a instalação do ambiente de programação *XCode*. O desenvolvimento dessas aplicações, tal como em *Android*, pode ser feito com o recurso a duas linguagens de programação, *Objective-C* ou *Swift*, onde a primeira representa uma abordagem mais primitiva, mais de baixo nível e a segunda opção uma linguagem mais moderna e que evolui aproveitando outros pontos positivos e modernos de outras linguagens[42].

Uma das grandes vantagens em termos de desenvolvimento de haver apenas um tipo de dispositivos, é a facilidade com que se consegue construir as interfaces visuais das aplicações, que ainda se torna mais facilitado com o *auto layout*, que é a ferramenta principal no incentivo ao desenvolvimento para múltiplos tamanhos de ecrã, permitindo criar *layouts* dinâmicos e que se adaptem automaticamente à resolução e tamanho do ecrã[43].

[12] https://developer.android.com/studio/

2.2.3 Outras plataformas e Soluções Híbridas

Quando se fala do desenvolvimento de aplicações móveis, para além da abordagem nativa, onde se desenvolve algo apenas exclusivo para *Android*, *iO*S ou *Windows Phone*, existem mais duas abordagens que se podem considerar, abordagens híbridas, ou abordagens web. Neste tipo de abordagens, a aplicação desenvolvida funciona em mais do que uma plataforma, com a mesma base de código.

Abordagens Nativas

Como abordagem nativa já foi descrito o desenvolvimento das duas principais plataformas, *Android* e *iOS*. Em projetos de grandes companhias de desenvolvimento de aplicações, a abordagem nativa ainda é a mais utilizada, apesar de requerer mais recursos monetários, visto que é necessária uma equipa para cada vertente. Apresenta uma série de vantagens, sendo a principal a melhor performance comparativamente a outras, principalmente em aplicações que necessitem de mais *hardware* do dispositivo móvel, no entanto começam a surgir opiniões de que aplicações que apenas dependam de pedidos à web tornam-se mais eficazes se for adotado um desenvolvimento híbrido ou web [44][45].

Abordagens Web (*Websites* responsivos)

A abordagem web consiste no desenvolvimento com tecnologias web, mais propriamente os típicos *HTML, CSS, JS,* juntando também várias *frameworks* web muito utilizadas hoje em dia. Em relação à abordagem nativa, é suposto ser mais fácil todo o design da aplicação visto que a abordagem para o fazer é como se o programador estivesse a desenhar um *website,* mas apenas tem de o tornar responsivo de modo a adaptar-se aos vários dispositivos e tamanhos de ecrã. É uma abordagem menos eficiente que a nativa, mas que em aplicações mais leves ou que não necessitem de aceder ao *hardware* do telemóvel ou que necessitem de grande poder computacional pode ser a mais indicada [45].

Abordagens Híbridas

As abordagens híbridas, permitem disponibilizar uma aplicação para as várias plataformas distintas, consistindo numa aplicação web que é encapsulada num *container* nativo de cada plataforma[44]. Pode-se dizer, que é uma abordagem barata em comparação com a abordagem nativa, visto que no mundo profissional, uma empresa não teria custos de desenvolvimento e manutenção de 2 bases de código, mas apenas de uma. Muitas das abordagens deste tipo consistem na utilização de *frameworks* web, como o *React Native* ou o *Ionic*, havendo também alternativas que utilizam linguagens "não *web"* como por exemplo o *Xamarin* onde a linguagem utilizada é o *C#*.

No desenvolvimento deste projeto foi pensado numa fase inicial desenvolver um protótipo de uma das aplicações em *React Native*. No entanto, devido às limitações deste tipo de tecnologias em termos de performance, e por existência de problemas de compatibilidade e ligação a sensores através de tecnologias

deste género, seria difícil tornar a aplicação eficiente e estável, sendo que foi optado pelo caminho do desenvolvimento nativo.

2.2.4 Padrões de arquiteturas em aplicações móveis

Os padrões de arquitetura em aplicações móveis definem o guião interno da aplicação. Existem vários tipos de padrões, uns mais utilizados do que outros, cada um com os seus próprios pontos fortes e pontos fracos. Contudo, antes de se iniciar um projeto para o desenvolvimento de uma aplicação móvel é suposto haver uma fase conceção da arquitetura interna, onde o padrão de arquitetura deve ser escolhido de modo a facilitar o processo de desenvolvimento em vários pontos:

- Um padrão de arquitetura deve ser pensado de modo a que a fase de testes unitários, testes de stress, ou testes à interface sejam o mais facilmente possível de ser executados, mas também, que permita testar a aplicação módulo a módulo;
- O padrão de arquitetura deverá facilitar o desenvolvimento da aplicação em equipa, onde cada programador consiga desenvolver o seu próprio módulo, sem ter dependências diretas do trabalho dos outros programadores.
- Deve tornar o código o mais simples e conciso possível, não criando ficheiros demasiado grandes, ou ficheiros de código que englobem um número de funcionalidades elevado dando origem a porções de código monolíticas e difíceis de trabalhar.

Apresentam-se de seguida os padrões mais relevantes, para a estrutura lógica de uma aplicação móvel.

MVC – Model View Controller

A arquitetura MVC, corresponde ao padrão mais utilizado no desenvolvimento de aplicações móveis, é muito utilizado tanto no desenvolvimento em *Android* bem como em *iOS*. Esta arquitetura separa a aplicação em 3 componentes distintos, o *Model*, componente que contém dados da aplicação, o *Controller*, componente responsável por lidar com a maior parte da lógica da aplicação, que é notificado pelo comportamento do utilizador, sendo que também é responsável por alterar o modelo, e a *View*, que corresponde a toda a parte visual da aplicação. As trocas de informação entre os componentes podem ser vistas esquematicamente na Figura 2-13.

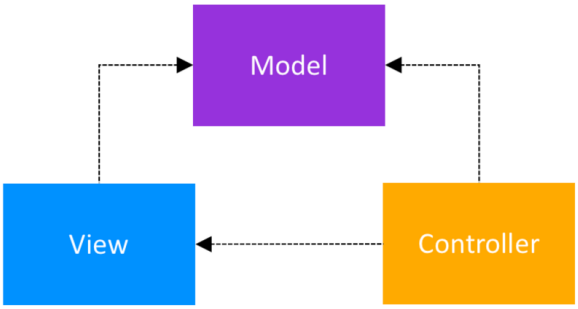


Figura 2-13 – Representação esquemática do padrão MVC. [13]

Os pontos fortes deste tipo de padrão teoricamente são a separação dos 3 conceitos principais numa aplicação, dados, lógica e interface visual, tornando o código mais testável e extensível. No entanto, o que pode ser visto como uma vantagem torna-se rapidamente uma desvantagem, visto que a maioria dos programadores torna cada um destes conceitos dependentes um do outro, sendo muitas vezes visível a camada *Model* estar inserido no *Controller*, ou então parte da função da camada da *View* estar implementada também no *Controller*. Este ciclo de dependências torna as aplicações muito difíceis de testar, visto que não há uma separação clara de conceitos entre a parte visual e a parte da lógica da aplicação. Para além da dificuldade em testar, o código dos *Controllers* acaba por se tornar exageradamente grande dando origem a ficheiros com centenas de linhas de código [46].

Neste padrão, em termos de ficheiros de código na plataforma *Android*, a componente da *View* acaba por estar distribuída entre os ficheiros de *layout (xml)* e as *Activities* ou *Fragments*, o componente *Controller* corresponde também às *Activities* e *Fragments*, e o modelo, acaba muitas vezes por estar presente também nesses mesmos ficheiros, onde o acesso aos dados é feito diretamente no código do *Controller*.

Esta dificuldade na parte de testes, e a criação de ficheiros de código demasiados extensos, levou ao aparecimento recente de um padrão renovado, o *Passive MVC*. Este, resulta numa implementação do modelo *MVC* mais extensível, mais fácil de manter e também que acaba por ter mais performance em termos de execução das aplicações, a maior característica deste modelo passivo é a de que as *Views* desconhecem por completo o modelo, tornando-a passivas, sendo muito parecido ao modelo que vai ser apresentado de seguida [47].

MVP – Model View Presenter

Esta arquitetura é tipicamente utilizada no desenvolvimento de *Android*, sendo considerada pela comunidade como uma alternativa viável ao *MVC*. Nesta arquitetura existem três grandes componentes, o *Model*, responsável por gerir os dados, o *Presenter*, que funciona como entidade mediadora de comunicação entre o *Model* e a *View*. É função do *Presenter* fazer as *queries* ao modelo, atualizar a *View* e reagir a

[13] https://medium.com/upday-devs/android-architecture-patterns-part-1-model-view-controller

interações do utilizador que impliquem uma alteração no modelo, a *View*, é responsável por apresentar os dados e as animações de acordo com a decisão do Presenter[48], tal como é possível ver na Figura 2-14.

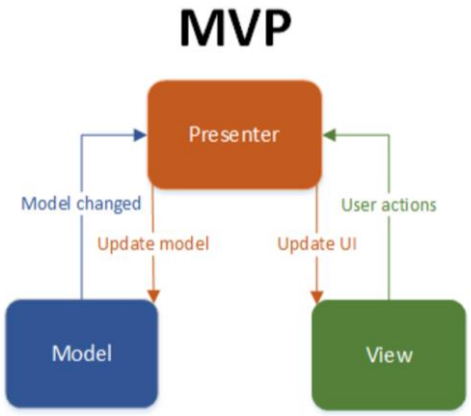


Figura 2-14 - Exemplo gráfico da arquitetura MVP. [14]

As vantagens deste tipo de arquiteturas, é que o programador é forçado a separar a *View* de qualquer lógica e também do modelo, tendo o *Presenter* toda essa responsabilidade, sendo possível implementar testes para as *Views*, sendo apelidada muitas vezes de *Passive View*, dado que não conhece nada da lógica da aplicação. Neste modelo, apesar do *Presenter* comunicar com a *View*, ele apenas deve conhecer o contrato da *View* – uma *interface* – tornando assim a arquitetura muito desacoplada respeitando um dos princípios da metodologia SOLID [49], tornando o *Presenter* muito mais testável. A grande desvantagem desta arquitetura, é a de que o *Presenter* acaba por ter uma parte da lógica da *View* visto que é ele que lhe dá ordens diretamente através da interface que partilham, não havendo uma separação total destes dois conceitos [49].

Em termos de ficheiros de código, existe normalmente uma interface em Java associada às operações da *View* e também às operações *Presenter*, essas interfaces terão de ser implementadas, tendo também um papel de comunicação entre as entidades para além do papel de contrato. Nesta arquitetura, os ficheiros de *layout*, as *Activities* e *Fragments* correspondem à camada da *View*, sendo que o *Presenter* é uma classe em separado. O modelo tipicamente está numa entidade separada, num ficheiro denominado de repositório, que terá todas as operações de acesso aos dados.

MVVM – Model View ViewModel

A última arquitetura a descrever é o *MVVM*, muito utilizado em *iOS* e Android como alternativa ao *MVC*. Recentemente, o *Android* lançou suporte nativo para este padrão com a introdução de novos componentes dentro da própria plataforma [50]. É formada por três camadas, a *View*, que informa o *ViewModel* das ações do utilizador, o *ViewModel* que expõe conjuntos de dados para a *View*, e o *Model*, que funciona como interface de acesso aos dados. À primeira vista parece uma arquitetura muito semelhante ao *MVP*, mas funcionam de maneira diferente, porque no *MVP*, o *Presenter* diz diretamente à *View* que dados

[14] https://android.jlelse.eu/android-mvp-for-beginners

quer que ela mostre e contém também alguma lógica da *View*, no *MVVM* o *ViewModel* envia para a *View* vários *streams* de dados tendo a *View* possibilidade de subscrição aos que lhe interessam. Desta forma, o *ViewModel* não tem qualquer referência à *View*, sendo responsabilidade da *View* subscrever/observar as *streams* de dados enviadas pelo *ViewModel*[51], tal como é possível ver na Figura 2-15.

Figura 2-15 - Exemplo gráfico da arquitetura MVVM.[15]

Todo este desacoplamento do padrão, torna cada componente muito fácil de testar, mais fácil ainda do que os componentes do modelo *MVP*. Concluindo, o *MVVM* combina as vantagens da separação de conceitos com as vantagens da subscrição de dados por parte da *View* ao *ViewModel*, tornando este modelo muito eficiente em programação orientada a eventos [48].

Em termos de código real, a *View* corresponderá ao conjunto do ficheiro de *layout* e à *Activity* ou *Fragment*, o *ViewModel* será uma classe em separado, sobre o qual a *View* terá *Observers*, sendo que tipicamente existirá um repositório ou outra entidade sobre o qual o *ViewModel* aceda aos dados, ou realize alterações.

2.2.5 Componentes de Arquitetura

O *MVVM* tem ganho protagonismo no desenvolvimento *Android* devido ao aparecimento dos *Architecture Components* [52]. Com a introdução destes componentes de arquitetura, o *Android* fornece aos programadores um guião de como construir as aplicações de modo a respeitarem as boas práticas na gestão do ciclo de vida dos componentes e assim evitar *memory leaks* ou outros problemas. Também encoraja a tornar a arquitetura interna limpa e organizada, de modo a tornar o código mais estruturado e modular [50]. O aparecimento destes novos componentes, para além de tornar os componentes básicos (fragmentos, atividades) *LifecycleAware* [53], introduziu uma classe nova denominada de *ViewModel* que permite implementar um *ViewModel* de forma adequada a respeitar os padrões definidos pela arquitetura *MVVM*. Uma arquitetura tipo com a utilização destes componentes de arquitetura é vísivel na Figura 2-16 onde podemos ver um *ViewModel* associado à *Activity/Fragment*, sendo que esse *ViewModel* faz pedidos e alterações de dados a um repositório, que é a entidade que tem acesso direto aos dados, internos, e aos dados provenientes de um serviço *Web*.

[15] https://medium.com/upday-devs/android-architecture-patterns-part-3-model-view-viewmodel

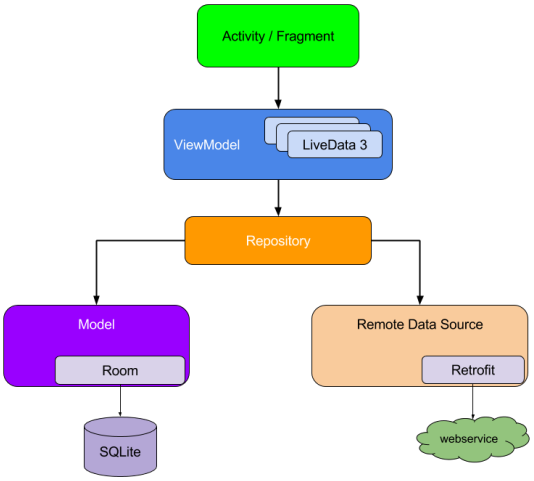


Figura 2-16 - Arquitetura tipo de uma aplicação com os Architecture Components.[16]

Esta inclusão de novos componentes e alteração dos existentes de modo a serem *LifecycleAware*, juntamente com outro tipo de bibliotecas como *ReactiveX* ou *EventBus* e *Databinding* permitem tornar o fluxo do padrão *MVVM* numa arquitetura orientada a eventos ao contrário das outras alternativas. Existe assim uma noção e implementação clara do padrão de *Observer* onde existem entre os componentes pontos que funcionam como *Observables (*receção das *streams* emitidas) e *Producers* (enviam *streams* de dados). Este último ponto ganhou ainda mais ênfase com a introdução das variáveis *LiveData*, que não são nada mais do que variáveis *Observable* do tipo *LifecycleAware*. Na Figura 2-17 é possível ver o fluxo de "observações" e "produções" de dados entre as diversas entidades da aplicação.

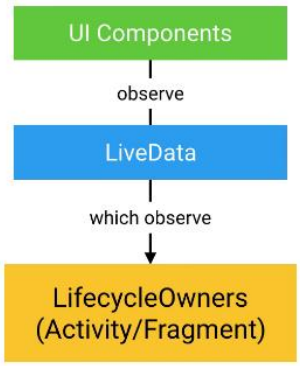


Figura 2-17 - Imagem demonstrativa do fluxo das interações entre Publishers e Observables.[17]

Para além da introdução do *ViewModel* nos componentes de arquitetura, houve a inclusão do *Room*, uma API de alto nível sobre *SQLite*. Este componente, veio introduzir no componente *Model* a possibilidade de criar queries que funcionam como variáveis do tipo *LiveData* e que permitem em tempo real atualizar a *View* sem necessidade de recorrer a operações assíncronas de modo a fazer *fetch* dos dados diretamente da base de dados. Este componente, fornece a possibilidade de definir as entidades sobre a forma de *POJO's* –

[16] https://developer.android.com/topic/libraries/architecture/guide
[17] https://github.com/KucherenkoIhor/Android-Architecture-Components

Plain Old Java Objects – onde o acesso aos dados, de modo a realizar operações *CRUD*, é realizado através de *Data Access Objects*. Um diagrama de interação entre o *Room* e uma aplicação é vísivel na figura seguinte.

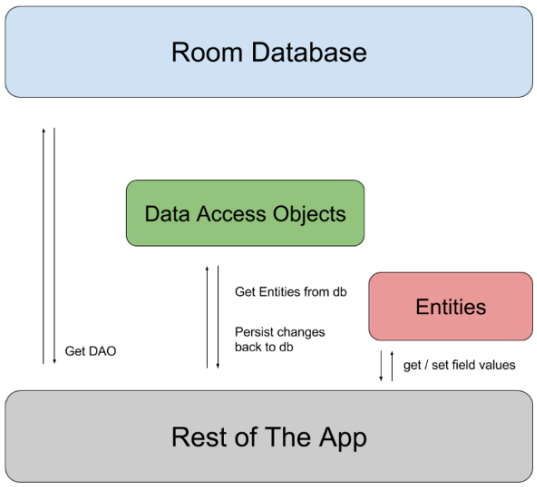


Figura 2-18 - Diagrama de interação do componente Room com as aplicações. [18]

[18] https://medium.com/mindorks/sqlite-made-easy-room-persistence-library

3 Soluções de recolhas de sinais pré-existentes

O trabalho desta dissertação parte dos resultados pré-existentes e casos de uso do projeto *Vital Responder*, *Intelligent management of critical events of stress, fatigue and environmental hazard* [4], que provém de uma colaboração entre o IEETA [3], INESCTEC [54], Instituto de Telecomunicações (IT) [55] e CESAM (Centro de Estudos do Ambiente e do Mar) [56]. Este projeto tem o objetivo principal de monitorizar e suportar pessoas e equipas em situações e profissões de risco, monitorizando os sinais vitais, o contexto envolvente, e aspetos ambientais em seu redor.

3.1 Projeto Vital Responder

O Projeto Vital Responder, engloba várias vertentes de investigação que exploram uma junção entre tecnologias de *wearables* inovadores, com redes de sensores, procurando construir tecnologia inteligente e serviços de localização precisos de modo a fornecer ferramentas de monitorização em cenários de emergência. O foco principal deste projeto é o cenário dos bombeiros, em que através de sensores que são transportados no seu equipamento e roupa (Figura 3-1), são recolhidos dados fisiológicos e também do contexto envolvente num cenário de missão de incêndio. Os dados podem ser visualizados através de interfaces Web, permitindo ao comandante ou supervisor de equipa fazer uma monitorização do estado de cada bombeiro no terreno.

Figura 3-1 - Grupo de sensores presente no equipamento de um bombeiro.

O projeto conheceu 3 fases principais:

1. VR1 (2009-2012): que tinha como objetivos principais a criação de um sistema seguro, fiável e efetivo de primeira resposta e de suporte à decisão em cenários críticos de emergência.
2. VR2 (2013-2015): que tinha como objetivo a gestão inteligente de eventos críticos de stresse, fadiga e intoxicação por inalação de fumo.
3. VR2Market (2015-2018): que tem como objetivo criar um produto de monitorização móvel para primeira resposta para situações perigosas, consolidando a linha de investigação desenvolvida desde 2009

Neste período foi desenvolvida uma arquitetura com vários módulos, que permite auxiliar e estudar os bombeiros em situações de risco, permitindo também, aos seus superiores terem uma maneira de ter algum *feedback* em tempo real, bem como ferramentas de análise posterior.

3.2 Solução existente

Na Figura 3-2 podemos ver a arquitetura global do projeto, em que no lado esquerdo da imagem, no componente *VR Unit* temos a aplicação agregadora, que é colocada em cada bombeiro, e que ao ligar-se a sensores procede à agregação dos dados, e que também engloba maneiras de publicar os dados em tempo real para um *broker RabbitMQ*. Esta aplicação, implementada em *Android*, permite a agregação de dados de sinais vitais, de localização e ambientais.

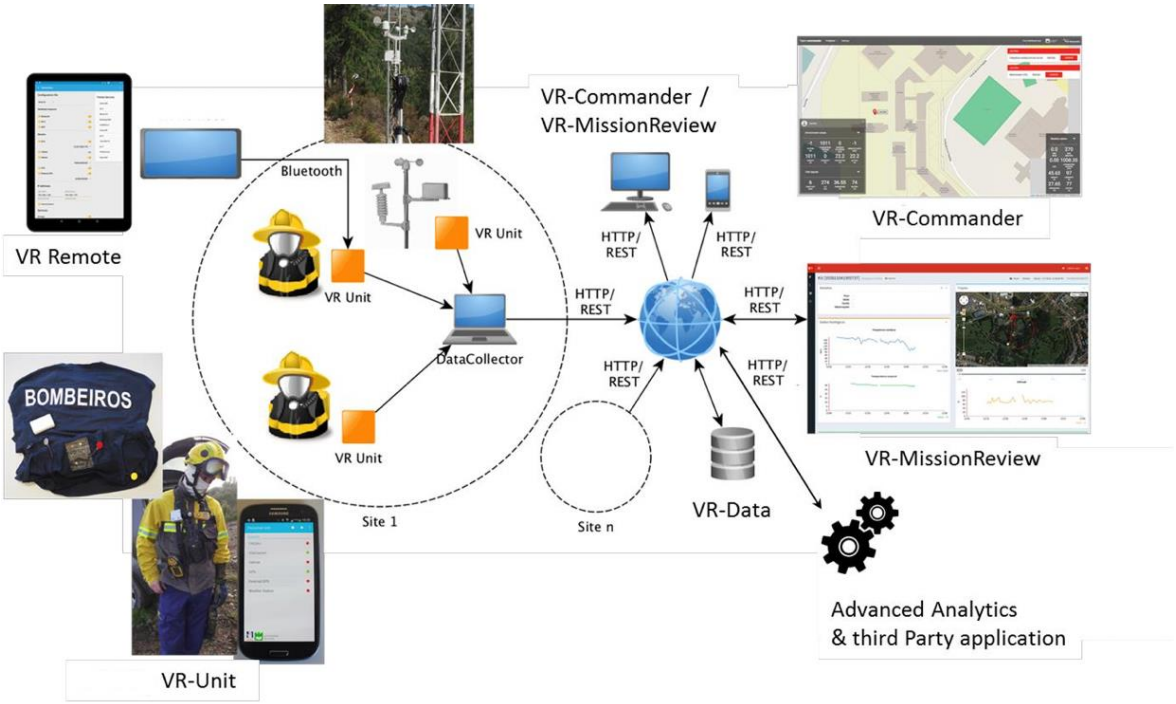


Figura 3-2 - Arquitetura do projeto VR2Market.

No canto superior esquerdo, temos outro dos componentes, o *VR Remote*, aplicação *Android*, que tem como função permitir a configuração em grupo de vários *VR Unit*'s, fornecendo uma maneira simples de monitorizar em tempo real a ligação dos sensores do *VR Unit* através da visualização de um código de cores que traduz o estado de ligação.

Outra das ferramentas de controlo, mas que permite monitorização dos dados e do bombeiro em tempo real é o *VR Commander*. Este componente, corresponde a uma *dashboard* web que é atualizada em tempo real, pelos dados que são recebidos pelas aplicações agregadoras no *DataCollector*. Com o *VR Commander*, o comandante, ou a pessoa responsável pela missão de incêndio/salvamento em questão, tem uma maneira de monitorizar em tempo real os operacionais em campo, onde poderá ver os dados recebidos pelos sensores e daí tomar decisões consoante alguns desses valores.

Por fim, temos o componente de pós-análise das missões realizadas, o *VR-Mission Review*, que permite o carregamento de análise dos ficheiros resultantes de missões passadas. Com o auxílio de uma escala temporal é possível percorrer cada momento da missão realizada, verificando a posição de cada bombeiro no terreno, mas também visualizar os seus dados fisiológicos e ambientais recebidos naquele instante.

3.3 Sensores

O sistema gere dados colhidos por sensores através de comunicações *Bluetooth*. Podemos identificar vários tipos de variáveis e sensores que são possíveis de ser recolhidos:

- Dados ambientais:
    - Sensor FREMU (Figura 3-3 b))
    - Sensor HELMET (Figura 3-3 a))
    - *Weather Station* (Figura 3-3 c))
- Dados de localização:
    - GPS Interno de cada telemóvel
    - GPS Externo G2RAYS (Figura 3-3 d))
- Dados Fisiológicos:
    - Sensor *Vital Jacket*. (Figura 3-3 e))

Figura 3-3 - Exemplos de sensores: Helmet, Fremu, Weather Station, G2Rays, Vital Jacket.

Os sensores de localização, GPS interno e G2Rays, permitem recolher dados de localização (latitude e longitude), bem como o tempo atual da tranche de dados recolhida (*timestamp)* e altitude. Este tipo de dados é utilizado para relacionar e analisar as adversidades da localização com as reações do bombeiro a cada momento.

Os dados fisiológicos permitem recolher o valor de batimento cardíaco de cada bombeiro, bem como a temperatura corporal e variações de acelerómetro. Através destas variáveis é possível correlacionar o batimento cardíaco de cada bombeiro com o seu movimento a cada situação durante a sua missão de salvamento.

Os sensores ambientais identificados, permitem recolher valores de temperatura, humidade, monóxido de carbono, pressão atmosférica, velocidade e direção do vento, e também se está a ocorrer precipitação ou não. Num cenário de incêndio estas variáveis são importantes de serem recolhidas devido à informação que fornecem sobre o contexto envolvente de cada operacional, permitindo o comandante estar ciente das

variáveis ambientais de cada momento durante a operação, permitindo tomar decisões que seriam impossíveis de tomar sem ter acesso a dados deste género em tempo real.

3.4 Limitações e oportunidades

Como parte integrante do trabalho desta dissertação, foi feita uma análise das aplicações *VR Remote* e *VR Unit*, na qual se conclui que em termos tecnológicos as aplicações estavam desatualizadas e já não seguiam os padrões de "boa programação" na área da computação móvel utilizados atualmente. É visível no código fonte a desatualização do padrão de arquitetura interno utilizado, onde os ficheiros de código são demasiados extensos e confusos, tornando difícil a manutenção e acréscimo de novas funcionalidades. A utilização do padrão de arquitetura MVC levou a extensos ficheiros de código fonte, tornando também as aplicações pouco modulares.

De modo aos componentes da aplicação (atividades, fragmentos, serviços) comunicarem entre si, eram utilizados *Intents* e *BroadcastReceivers*, o que apesar de ser um mecanismo fornecido pelo *Android* torna o código algo verboso e mais difícil de localizar as interações entre módulos. Esta abordagem é limitativa para a escalabilidade do código quando comparado com outro tipo de tecnologias como *EventBus* ou *ReactiveX* que podem ser utilizadas de modo a desacoplar os módulos.

A nível de armazenamento interno, está a ser utilizado SQLite. É caracterizado como difícil de trabalhar, visto que é necessário definir manualmente todas as tabelas de dados existentes no *schema* da base de dados. Para além disso, um dos pontos menos positivos da aplicação é o de que os dados dos sensores possíveis de associar à aplicação estarem inseridos na base de dados manualmente, não permitindo uma adição ou remoção de sensores na aplicação de forma dinâmica.

Os *layouts* nas aplicações existentes correspondiam muitas vezes a *layouts* dentro de *layouts* - *nested layouts* - diminuindo a performance no desenho dos mesmos por parte do dispositivo. Atualmente existe o *Constraint Layout*, que permite resolver este tipo de problemas, tornando os ficheiros de *layout* mais simples e com uma performance de renderização maior quando comparado com *nested layouts*.

Outro dos problemas encontrados foi o de que é muito difícil acrescentar suporte a novos sensores sem realizar um grande trabalho de escrita de código. Foram também detetados vários erros ao nível de algum código nativo de plataforma, onde foi detetada a existência de blocos de código que produziam *memory leaks*, que não estavam resolvidos.

Para além da desatualização tecnológica, havia algumas falhas ocasionais da aplicação no terreno, provocando o fecho da aplicação. Na sua utilização em recolhas de psicofisiologia, houve erros reportados pelos investigadores que utilizavam as aplicações. Verifica-se que o maior problema deste género será mesmo o da comunicação entre as aplicações, *VR-Remote* e *VR-Unit*, que é baseada em *Bluetooth* e *Sockets de Wifi*, tornando a comunicação pouco fiável, levando por vezes à não entrega de mensagens de comunicação, o que leva por vezes a cenários não desejáveis no decorrer de recolhas de psicofisiologia.

4 Casos de utilização do *Vitals Recorder*

4.1 Processos associados às experiências de psicofisiologia

O foco deste projeto é o de criar um sistema com base em aplicações móveis que permita realizar recolhas das mais diversas variáveis fisiológicas, de localização e ambientais. O sistema deve ser genérico, dando a possibilidade aos seus utilizadores de se adaptar a vários cenários, sendo os principais as missões de salvamento ou de bombeiros, recolhas de psicologia, ou cenários de fitness.

Devido à colaboração existente entre o grupo de investigação bitMob do IEETA e o Departamento de Educação e Psicologia da Universidade de Aveiro, foi definido como foco principal do sistema o cenário das recolhas de psicologia, onde são efetuadas principalmente recolhas de diversos biosinais com as aplicações existentes do projeto *VR2Market*.

Após uma fase de levantamento de requisitos, na qual foram realizadas algumas conversas com investigadores de psicologia de modo a perceber os processos no âmbito das recolhas dos estudos de psicologia, mas também por observação direta de uma dessas recolhas, foi possível criar uma perceção sobre todos os processos que envolvem as fases pré, pós e durante a recolha. Os participantes das experiências são tipicamente voluntários que poderão estar sujeitos a um processo de seleção, em que através de um questionário ou de uma pequena entrevista é feita uma triagem aos mesmos. As experiências podem variar um pouco os métodos de recolha de dados relevantes para o estudo, mas pode-se dividir os seus procedimentos em duas variantes. Numa delas a recolha de dados é suportada por questionários ou por mera observação direta do participante, e noutro tipo, podemos ter o auxílio de instrumentos, nomeadamente câmaras, telemóveis, computadores, ou outro tipo de tecnologia, que servirá de suporte à experiência, permitindo recolher vários tipos de dados que poderão no final da experiência ser avaliados e estudados.

4.2 Cenários a suportar

O sistema tem dois atores principais: o responsável pela experiência; e a(s) pessoa(s) que está sujeita à aquisição, que não terá de interagir diretamente com o sistema. A interação com o *Vitals Recorder* tem lugar através de três subsistemas:

A aplicação VR (*VitalsRecorder)* Unit que tem como objetivo agregar os dados recolhidos pelos sensores internos, ou pelos sensores externos registados na aplicação. Esta funcionará autonomamente, ou seja, a recolha é efetuada sem o auxílio da aplicação *Remote*, ou também poderá funcionar de maneira controlada, onde é configurado e controlado pela aplicação *Remote*. Na figura Figura 4-1 é possível ver o diagrama de casos de uso deste módulo, e na tabela seguinte uma descrição de alto nível sobre cada caso de uso.

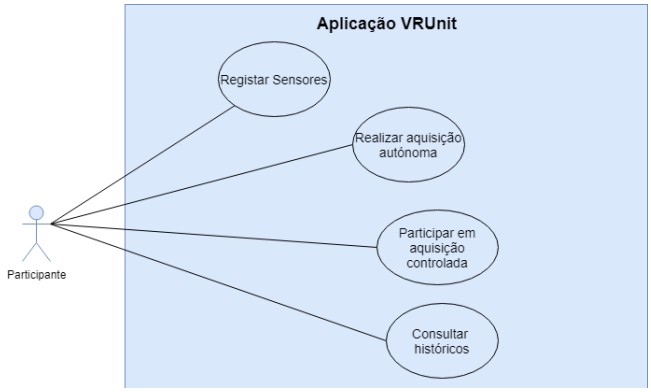


Figura 4-1 - Caso de uso do módulo pessoal.

Caso de Uso	Descrição
Registar Sensores	O utilizador da aplicação cliente de modo a realizar aquisições autónomas ou de modo a participar em aquisições controladas deverá registar sensores na aplicação.
Realizar aquisição autónoma	O utilizador de modo a realizar aquisições autónomas deverá selecionar os sensores que deseja utilizar na aquisição, sendo que após essa escolha é presenciado com uma *dashboard* de controlo de monitorização, onde poderá controlar a aquisição.
Participar em aquisição controlada	O utilizador ao querer realizar uma recolha controlada, terá de optar entre 2 caminhos, ser configurado por *Bluetooth*, ou realizar um *scan* a um código *QR* disponibilizado pela a aplicação controladora.
Consultar históricos	O utilizador pode verificar o histórico de projetos/estudos, grupos e informação associada, nos quais já esteve presente. Poderá também visualizar o histórico de alarmes e eventos, em opções separadas no menu da aplicação.

Tabela 4-1 - Descrição de casos de uso do módulo VR-Unit.

A aplicação controladora VR (*Vitals Recorder*) *Remote*, tem como objetivo controlar e monitorizar uma aquisição de diversas variáveis, sendo que é adaptável a vários cenários. Com esta aplicação será possível definir grupos de aquisição, definir alarmes sobre algumas variáveis, verificar em tempo real valores recolhidos pelas aplicações VR-Unit, e ainda consultar alguns elementos de histórico relacionados com os grupos de aquisição formados. Na Figura 4-2 é possível ver o diagrama de casos de uso deste módulo, seguido do quadro da descrição dos mesmos.

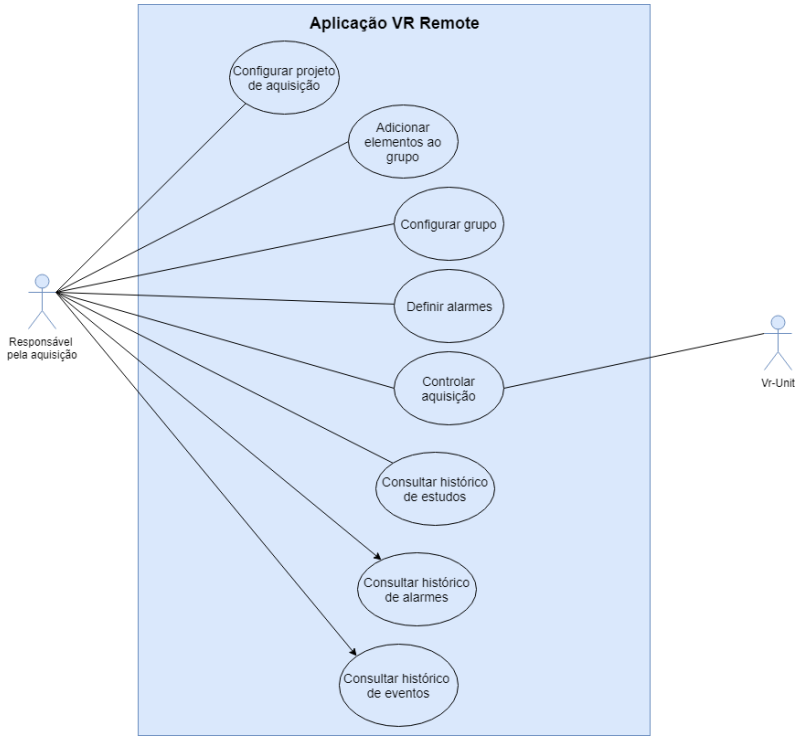


Figura 4-2 Casos de uso para o responsável pela aquisição (Tablet).

Caso de Uso	**Descrição**
Configurar estudo	O responsável configura o projeto/estudo sobre o qual quer realizar aquisições.
Adicionar elementos ao grupo	O responsável adiciona elementos ao grupo de aquisição após a configuração do estudo, podendo fazê-lo por *Bluetooth* ou código *QR*. Opcionalmente poderá definir nomes amigáveis para os elementos do grupo.
Configurar grupo de aquisição	O responsável define o nome identificador do grupo de aquisição, eventos a marcar durante a aquisição e variáveis a recolher.
Definir Alarmes	O responsável pode definir alarmes previamente a controlar a aquisição, definindo valores máximos/mínimos sobre os quais deverá ser alertado caso esses limites sejam ultrapassados.
Controlar Aquisição	O responsável controla a aquisição, onde poderá iniciar, parar, visualizar os dados recolhidos pelas aplicações de recolha associadas a cada participante e ainda marcar eventos. Poderá também ordenar o *upload* dos dados recolhidos pelos dispositivos de recolha.
Consultar histórico de estudos	O responsável consulta o histórico de projetos/estudos, grupos dentro desse estudo, onde poderá ver as configurações atribuídas a esse grupo bem como todos os participantes, podendo ainda visualizar um *log* detalhado das aquisições desse grupo.
Consultar histórico de alarmes	O responsável pode visualizar todos os alarmes que já ocorreram, onde será apresentada uma lista com o nome de todos os participantes que já tiveram alarmes. Ao carregar numa dessas células, será possível ter uma visão sobre todos os alarmes que esse participante gerou.
Consultar histórico de eventos	O responsável pode consultar os eventos que foram marcados durante o decorrer das aquisições.

Tabela 4-2 - Descrição de casos de uso do módulo VR-Remote.

O módulo VR Exporter *web app* permite a exportação dos dados, após um processo de registo e *login* por parte dos investigadores responsáveis pelas experiências. Na Figura 4-3 temos representado o diagrama de casos de uso, seguido do quadro de descrição.

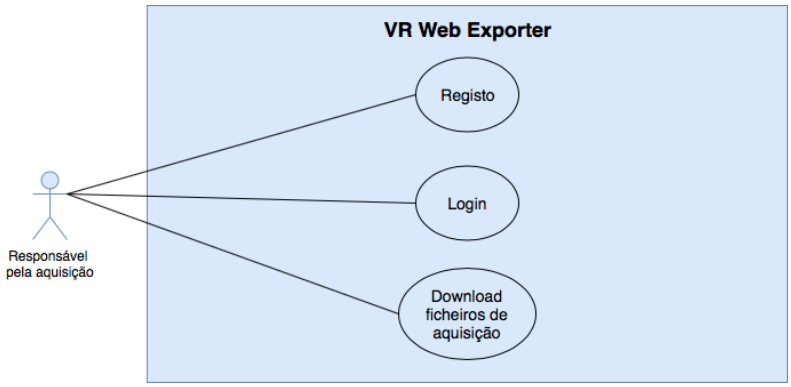


Figura 4-3 - Casos de uso para o responsável pela aquisição (Web).

Caso de Uso	Descrição
Registo	O responsável pode-se registar na *web app* de modo a ter acesso à exportação dos dados dos seus estudos.
Login	O responsável introduz as suas credenciais de modo a ter acesso à lista dos seus estudos.
Exportar ficheiros de aquisição	O responsável pela aquisição após realizar uma ou várias aquisições pode efetuar a exportação dos ficheiros de dados recolhidos através da plataforma *web*.

Tabela 4-3 - Tabela de casos de uso do módulo VR-Web Exporter.

4.3 Requisitos não funcionais

Devido à sensibilidade dos dados que estão a ser recolhidos, e de modo a evitar que os dados das aquisições fiquem comprometidos de alguma maneira, foram definidos vários requisitos não funcionais, de modo a garantir o funcionamento esperado do sistema, e de modo a zelar pela preservação dos dados. As tabelas Tabela 4-4 e Tabela 4-5 especificam os requisitos não funcionais e também requisitos de compatibilidade.

Requisito não funcional
As recolhas controladas devem de continuar caso haja uma desconexão ou problema com algum dispositivo.
As recolhas controladas devem continuar caso haja a desconexão ou problema com o dispositivo controlador.
Após uma desconexão ou problema deve ser possível restabelecer a conexão, durante uma aquisição controlada.
Os dados de recolha devem permanecer íntegros em caso de um erro que leve a aplicação a fechar.
Em caso de desconexão ou problema na aplicação a conexão deve ser restabelecida automaticamente

Tabela 4-4 - Especificação dos requisitos não funcionais do sistema.

Requisito de Compatibilidade e Portabilidade
Ambas as aplicações disponíveis em dispositivos com dimensões abaixo de 7" (telemóveis).
Aplicação controladora disponível em tablets (> 7”)
Aplicação controladora capaz de correr em modo *landscape*.
Aplicações suportarem língua inglesa.
Aplicações suportarem língua portuguesa.

Tabela 4-5 - Especificação dos requisitos de compatibilidade e portabilidade.

5 Arquitetura do Sistema

5.1 Arquitetura Proposta

O sistema de recolha que foi proposto está dividido em 3 grandes componentes, uma aplicação que é controladora da recolha, denominada de *Vitals Recorder Remote*, uma aplicação que serve de cliente a esse controlador, *Vitals Recorder Unit*, que tratará de se ligar aos sensores, ou utilizar os próprios sensores internos do telemóvel e efetuar a recolha, e ainda uma *Web App* de exportação de dados.

Na imagem seguinte, é possível ver um esquema da arquitetura geral do sistema, onde é possível ver os fluxos de comunicação e trocas de informação entre as 3 partes constituintes do sistema. Mais à esquerda temos a aplicação *VR Remote*, que comunica com o *VR MessageCollector*, com a API de integração e com o armazenamento na *Cloud*. Do lado direito da imagem, é possível ver um conjunto de aplicações *VR Unit*, e também outros sistemas externos integrados que comunicam também com o *Cloud Storage* e com a aplicação de controlo através do *MessageCollector*. Será possível uma visualização dos dados recolhidos por cada aplicação agregadora num sistema de visualização externo a este projeto, inserido no projeto VR2Market.

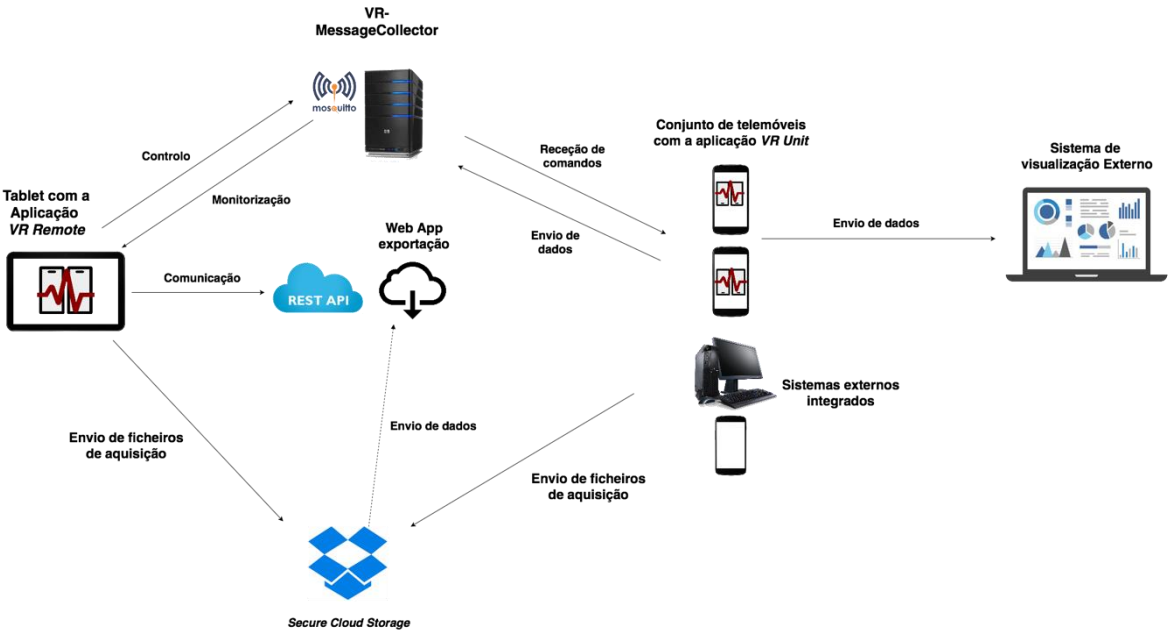


Figura 5-1 - Arquitetura geral do sistema

5.1.1 Aplicação móvel – *Vitals Recorder Remote*

Esta aplicação utilizará vários componentes de hardware do telemóvel, como o microfone de modo a ser possível a gravação de som, o sistema de ficheiros de modo a ser possível gravar ficheiros em persistência e também para implementação de uma base de dados interna para salvaguarda de alguma informação de histórico, também do *Bluetooth* que é uma das formas de emparelhamento com outros dispositivos. Internamente, será composta por atividades e fragmentos que fazem parte da componente *View* juntamente com os ficheiros de *layout*, sendo que cada um destes componentes contém um *ViewModel* que trata também do acesso à base de dados local, ou sistema de ficheiros do dispositivo para escrita e leitura de dados necessários. Comunica externamente com a *RestfulAPI* de integração, mas também com o *MessageCollector* e também com a *Dropbox* de modo a enviar os dados que tem no seu sistema de ficheiros.

A aplicação será ainda composta por múltiplos serviços que tratarão de rotinas mais pesadas computacionalmente ou comunicações externas como por exemplo comunicar com o *MessageCollector* ou fazer *upload* de dados, que têm de ser obrigatoriamente retiradas da *Main Thread* – fio de execução principal das aplicações em Android. A arquitetura encontra-se esquematicamente representada na Figura 5-2.

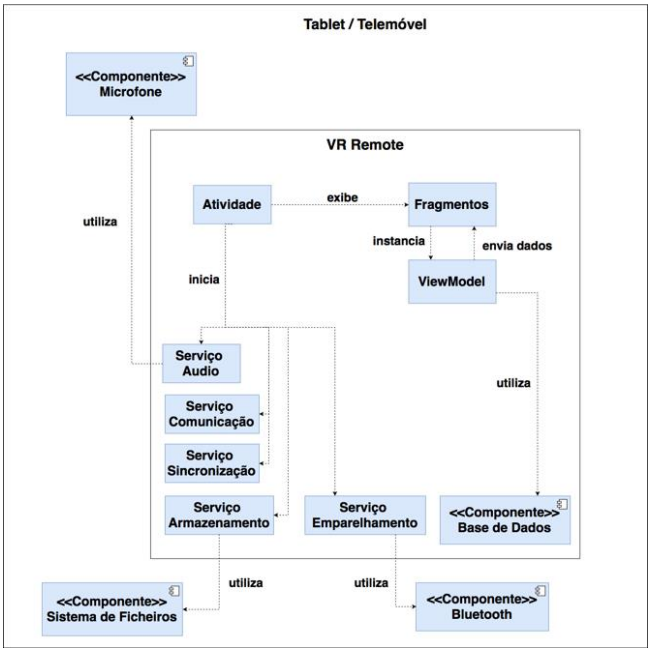


Figura 5-2 - Arquitetura aplicação VR-Remote.

5.1.2 Aplicação Móvel – *Vitals Recorder Unit*

A aplicação será compatível com dispositivos *Android*, e terá como padrão de arquitetura também o *MVVM*. A arquitetura encontra-se esquematicamente representada na Figura 5-3, onde é possível ver os vários serviços principais que irão compor a aplicação, bem como os componentes externos de *hardware* que utilizará, como a câmara e o *Bluetooth*, para emparelhar com a aplicação *Remote*, o sistema de ficheiros, de modo a armazenar os dados das recolhas. Esta aplicação terá associada a si uma biblioteca, que deverá ser

desenvolvida de modo a tratar de toda a ligação, comunicação e receção de dados dos sensores, denominada de *AcquisitionSensorManager*. Desta forma, será possível que o módulo da aplicação *VR-Unit* esteja desacoplado deste módulo que trata especificamente da ligação e comunicação com os sensores.

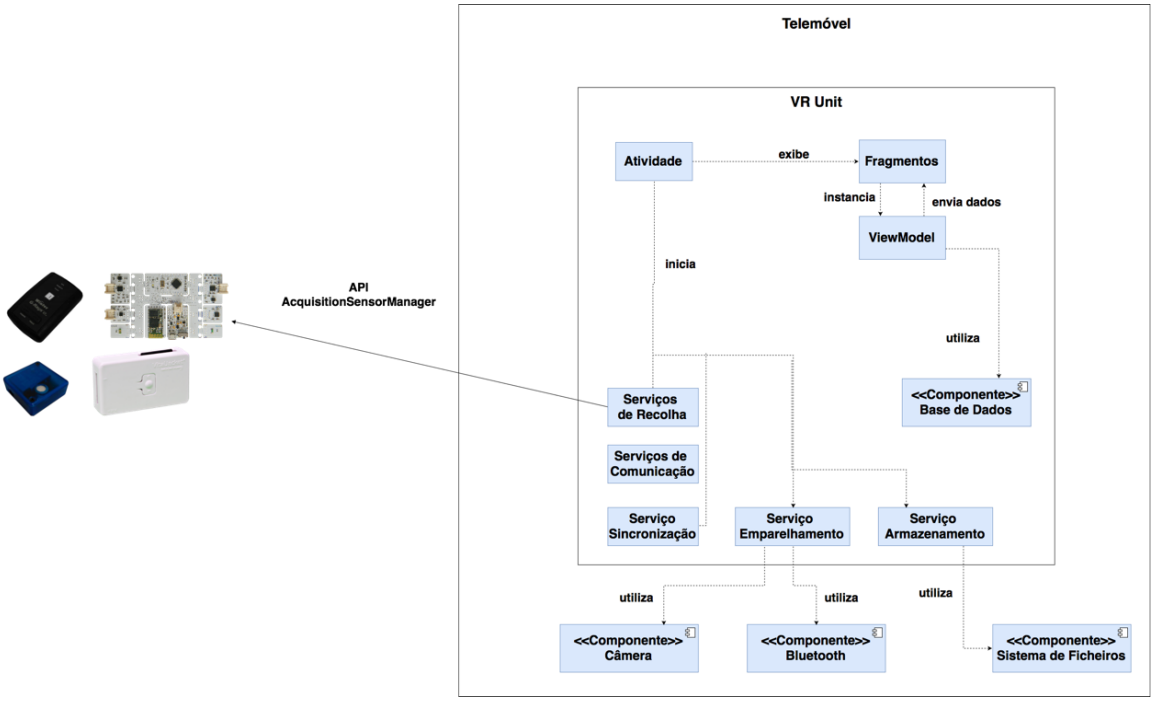


Figura 5-3- Arquitetura da aplicação VR-Unit.

5.2 Interação entre componentes

As aplicações *VR-Remote* e *VR-Unit*, comunicarão através do envio de mensagens pelo *MessageCollector*. A comunicação num broker de mensagens está assente em tópicos. todos esses clientes escutam as mensagens enviadas por qualquer um dos restantes clientes nesse dado tópico desde que o subscrevam, implementando o modelo de *Publish/Subscribe*. Foi escolhido este tipo de modelo de comunicação, devido à facilidade deste tipo de tecnologias em implementar por si só mecanismos de *acknowledge* e retransmissão de mensagens, poupando trabalho ao programador em implementar esse tipo de mecanismos, mas também tornando as comunicações mais fiáveis.

Um exemplo de interação entre a aplicação *VR-Remote* e a aplicação *VR-Unit* está esquematizada na Figura 5-4, onde a aplicação *VR-Remote* deverá fazer um pedido de dados, mediado pelo *MessageCollector*, após uma aplicação *VR-Unit* ter iniciado a receção de dados enviados pelos sensores, este pedido deverá de ser renovado temporalmente. O pedido a qualquer momento poderá ser parado por parte da aplicação *VR-Remote*, quando não for necessário o envio de mais dados.

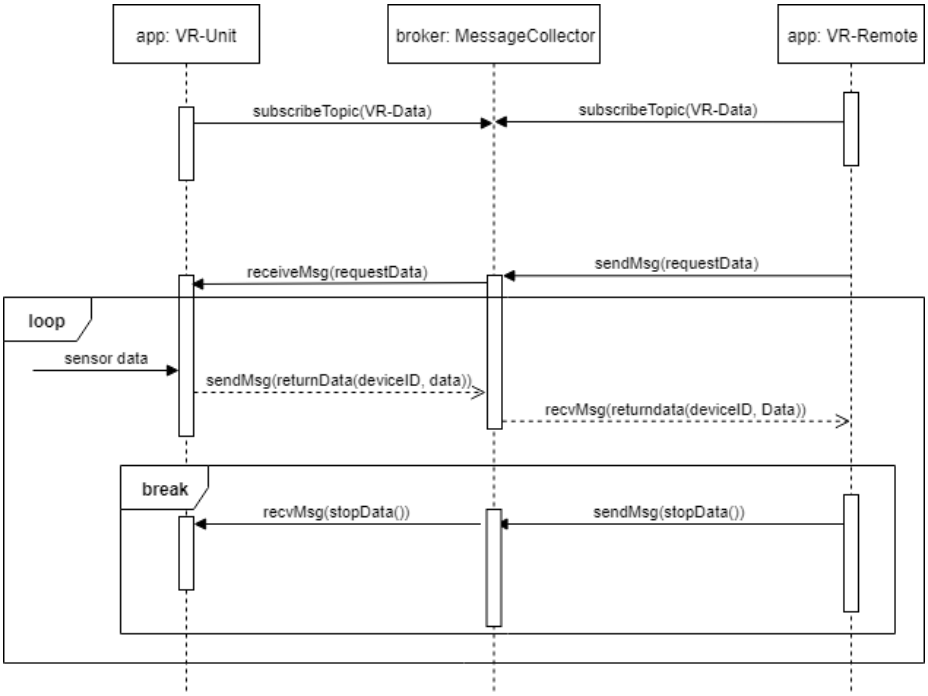


Figura 5-4 - Diagrama de sequência exemplificativa da troca de mensagens.

De modo a comunicar com o armazenamento Cloud seguro será utilizada a API disponibilizada aos programadores, onde é possível ter acesso a uma conta de *Dropbox* de modo a armazenar dados.

Na comunicação interna das aplicações, será utilizada uma tecnologia diferente da normalmente utilizada. Tipicamente, nas aplicações *Android* a comunicação interna entre componentes é feita através do envio de *Intents* e da sua receção através de *BroadcastReceivers*, no entanto a sua utilização torna as aplicações pouco desacopladas, sendo que o código para implementar este tipo de comunicação é também verboso e extenso. De modo a criar uma aplicação em que os componentes fossem desacoplados, tornasse o código mais "limpo" e modular, evitasse a criação de *memory leaks*, mas que aumentasse ainda mais a facilidade de comunicação entre componentes *Android (*atividades, fragmentos, serviços, *adapters*, etc.…) será utilizado o *EventBus* [57], que de uma forma geral funciona como um *broker* de mensagens, mas que neste caso é interno à aplicação.

No EventBus são definidas mensagens, que podem ser publicadas, e diversos elementos podem subscrever a essas mensagens, podendo executar blocos de código na receção do evento, mas sendo também possível transmitir dados com a criação destes eventos. Uma das grandes vantagens do *EventBus*, é que permite por omissão escolher em que *thread* de execução queremos executar a receção da mensagem, na *Main Thread* quando queremos executar uma operação visual, ou numa *thread* em background quando pretendemos correr código assincronamente. Também permite definir a prioridade na receção de mensagens, bem como definir mensagens *sticky*, ou seja, que se mantêm à espera de ser recebidas e também se mantêm na *queue* de mensagens até serem retiradas, o que em termos de implementação permite muitas vezes resolver vários problemas que teriam de ser resolvidos com a criação de mecanismos de *cache*.

5.3 Integração com sistemas externos

A utilização do *MessageCollector* como canal de comunicação, fornece um mecanismo de integração das soluções desenvolvidas em outros sistemas, que não os pertencentes ao *pipeline* do projeto *VR2Market*, esquematicamente visível na Figura 5-5.

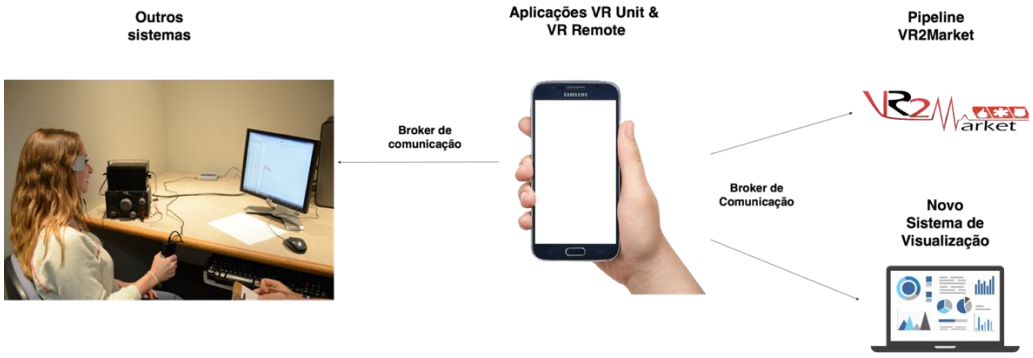


Figura 5-5 - Esquema da integração com serviços externos.

Qualquer solução externa que queira aproveitar as funcionalidades de controlo e agregação de sinais do sistema desenvolvido, poderá ter acesso ao *MessageCollector*, com as devidas credenciais, de modo a poder escutar tópicos específicos e daí aproveitar as mensagens trocadas para aplicações específicas.

Uma integração que este sistema deverá ter à partida, é a comunicação com o pipeline do projeto VR2Market, onde os dados recolhidos pela aplicação agregadora para os anteriores sistemas de visualização, mas também para o novo sistema de visualização em tempo real que está a ser desenvolvido noutro trabalho de dissertação de Mestrado, englobado no projeto.

5.4 Modelo de informação

Nesta subsecção será introduzido o domínio da psicologia e das recolhas da área, onde será feita uma introdução aos conceitos dos principais papeis no contexto da psicologia. Também será efetuado uma abordagem geral sobre o tipo de dados associados aos estudos da área.

Visão geral do domínio

O domínio associado aos estudos de psicologia pode ser descrito esquematicamente com recurso a um diagrama, visível na Figura 5-6. Este modelo de dados, deverá ser o seguido na adaptação do contexto do problema a código.

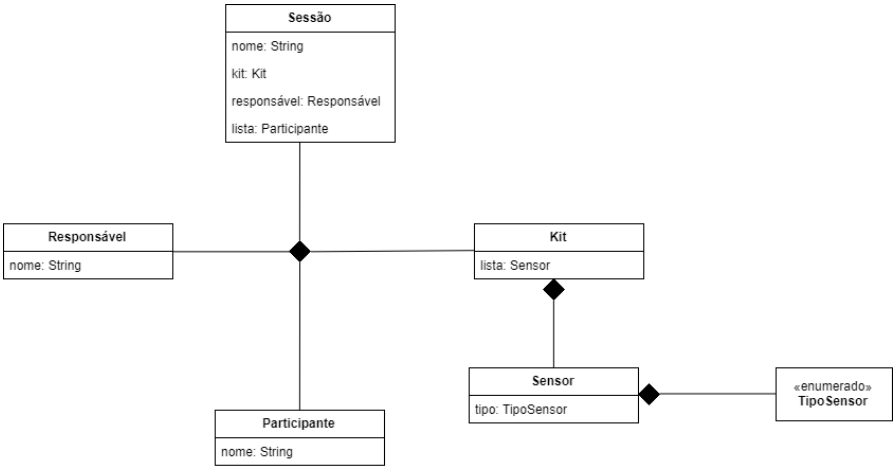


Figura 5-6 – Diagrama do modelo de dados do problema.

O Responsável, ou investigador, é a entidade principal de gestão e controlo de uma recolha, podendo ter ao seu encargo várias sessões, que serão compostas por um nome, por uma série de participantes, e terão um *kit* de recolha associado.

Cada participante deverá ser identificado por um nome amigável, que não o identifique pessoalmente de modo a não comprometer a privacidade dos dados, mas sim no estudo, como um identificador numérico sequencial, por exemplo: "Participante 204".

Equipamento utilizado

O equipamento nestes estudos com o sistema desenvolvido deverá de ser composto por:

- *Smartphones Android* com versão de sistema operativo de nível 18 ou superior.
- Opcionalmente: Um tablet, de modo a servir de suporte ao investigador.
- Sensores capazes de realizar recolhas de sinais fisiológicos, tais como *VitalJacket ou BITalino*.
- Elétrodos funcionais e fiáveis capazes de recolher dados de precisão elevada.

Dados dos estudos

Nas recolhas dos estudos de psicofisiologia realizados na Universidade de Aveiro, são tipicamente recolhidos dados fisiológicos de vários tipos, tais como: batimento cardíaco, sinal *ECG*, *EMG* e *EDA*. Esses dados, recolhidos pelos sensores, devem de ser agregados e organizados em ficheiros de maneira a que seja possível fazer programaticamente análise aos mesmos, e que também permitam uma conversão fácil para algum tipo de sistema de visualização, sob a forma de gráficos com escala temporal por exemplo.

Na linha de aplicações existente no projeto *VR2Market*, os dados das recolhas efetuadas eram guardados em formato *CSV* com a particularidade de cada "célula" de dados conter um campo, correspondente a um número inteiro que correspondia a um mapeamento do conteúdo da célula. Os dados podiam ser posteriormente visualizados no *Matlab*, ou em alguns dos *softwares* existentes do projeto, como o

VR-Mission Review que permitia importar os dados recolhidos, embora fosse mais indicado para as recolhas de bombeiros.

Posto isto, os dados deverão ser organizados em ficheiros, num formato que programaticamente seja de fácil leitura, como *Json*, *XML* ou *CSV*. Nestes ficheiros, cada linha deverá corresponder a uma amostra de dados recolhidos, que deverá ter uma marca temporal associada – *timestamp* – que identifique o momento em que aquela amostra foi recolhida, e que para além disso deverá ter os metadados associados presentes.

6 Implementação

Neste capítulo serão abordados os métodos do processo de implementação do projeto, serão explicados em detalhe as várias fases de desenvolvimento das aplicações, bem como a implementação de toda a parte de *backend* associada.

6.1 Elementos comuns de desenvolvimento

Ambas as aplicações desenvolvidas, *VR-Unit* e *VR-Remote*, foram desenvolvidas em *Android*, partilhando o padrão de arquitetura utilizado, *MVVM*, levando a que grande parte da arquitetura interna e fluxos de dados entre componentes sejam semelhantes nas aplicações. Na seguinte subsecção será realizada uma pequena introdução a componentes que foram usados de maneira idêntica em ambas as aplicações.

Ambas as aplicações foram escritas com a linguagem de programação *Kotlin* [19] e os exemplos de código demonstrados neste documento encontram-se escritos nessa linguagem.

6.1.1 Aplicação dos componentes de arquitetura

Nas duas aplicações *Android* desenvolvidas foram utilizados os componentes de arquitetura introduzidos em 2017 na plataforma *Android*. Estes permitem uma melhor gestão de memória dos componentes nativos do *Android*, reduzindo a probabilidade de acontecimentos de *memory leaks* e outras más práticas de programação. Esta utilização dos componentes de arquitetura permitiu tornar a aplicação mais adequada ao padrão *Model-View-ViewModel*, dado que estes componentes são um incentivo da própria plataforma à utilização desse padrão.

Foram utilizados os seguintes componentes de arquitetura:

- *DataBinding*: Inserção direta dos dados dos *ViewModel* para as *Views* da aplicação;
- *Lifecycles*: Componentes de *UI* LifecycleAware, como *Activities* e *Fragments;*

[19] https://kotlinlang.org/

- *LiveData*: Variáveis do tipo *Observable*, mas que são *LifecycleAware*;
- *Room*: Camada de alto nível de acesso à base de dados SQLite;
- *ViewModel*: Entidade que gere toda a lógica de dados associada a *Views*, resistindo a rotações, permitindo também uma melhor gestão de operações assíncronas relacionadas com os dados.

DataBinding

Para tornar o fluxo de dados no padrão *MVVM* o mais correto possível, o *Android* fornece a ferramenta de *Data Binding*, que permite tornar os *layouts* mais declarativos [58]. É possível passar referências para o *ViewModel* diretamente para o ficheiro *.xml*, ou declarar outros tipos de dados, sendo que os dados são diretamente injetados para a *View* sem o fazer explicitamente. Este automatismo é possível através da utilização de variáveis do tipo *Observable* ou *LiveData* na declaração do *ViewModel*. Para além disto, o módulo de *DataBinding* permite também incluir alguma lógica diretamente no ficheiro *xml* como por exemplo comparações ou outro tipo de mecanismos de decisão.

Nas aplicações desenvolvidas, este componente foi utilizado na construção de *layouts* onde através da utilização de variáveis de *databinding* era efetuada a ligação ao código do fragmento ou da atividade (Figura 6-1 a)), também foi utilizado como componente que realiza alguma lógica nos próprios ficheiros de *layout* de modo a mostrar ou esconder alguns elementos visuais (Figura 6-1 b)). Ainda na figura, temos uma *TextView*, que aproveitando a variável de *Databinding* associada ao *ViewModel*, extrai diretamente o campo *name* da variável *observableDevice*, de modo colocar esse texto na *TextView*.

a)

```
<data>

    <variable
        name="callback"
        type="com.bitmob.vrremote.features.devicedashboard.DeviceDashboardFragment" />

    <variable
        name="viewModel"
        type="com.bitmob.vrremote.features.devicedashboard.DeviceDashboardViewModel" />
</data>

<android.support.constraint.ConstraintLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/team_name_tv"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginEnd="@dimen/normal_margin"
        android:layout_marginStart="@dimen/normal_margin"
        android:text="@{viewModel.observableDevice.value.name}"
        android:textSize="@dimen/regular_text_size"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toEndOf="@+id/textView7"
        app:layout_constraintTop_toTopOf="@+id/textView7" />
```

b)

```
override fun bindViews(container: ViewGroup?) {
    fragmentBinding = DataBindingUtil.inflate(layoutInflater, R.layout.fragment_device_dashboard,
            container,  attachToParent: false)
    fragmentBinding.fragment = this
    fragmentBinding.viewModel = viewModel
    startMissionFab = fragmentBinding.startMissionBtn
    stopMissionImgBtn = fragmentBinding.stopMissionBtn
    aloneMissionBtn = fragmentBinding.aloneMissionBtn
    endConnectionBtn = fragmentBinding.leaveAssociationBtn
    chrono = fragmentBinding.chronometer2
    cameraPreview = fragmentBinding.recognitionSv
    scanQRFab = fragmentBinding.fabCamera
```

Figura 6-1 - Exemplo de DataBinding com o recurso ao ViewModel a), ligação no fragmento b).

LifeCycles

No início do desenvolvimento das aplicações, foi necessário utilizar componentes explicitamente *LifecycleAware*, nomeadamente, *LifecycleActivities* e *LifecycleFragments*, no entanto, atualmente o componente *Activity* e *Fragment* já são *LifecycleAware.*

Com esta alteração, o impacto no desenvolvimento da utilização de *LifeCycles* está "escondido" aos olhos do programador. No entanto, a importância do *LifecycleAware* nestes componentes é elevada, permitindo que todos os componentes de arquitetura introduzidos (*LiveData, ViewModel, DataBinding)* respeitem o ciclo de vida de cada um desses componentes. Portanto, os componentes *Activity* e *Fragment* utilizados no desenvolvimento das duas aplicações, são *LifecycleAware*.

LiveData

As variáveis do tipo *LiveData* foram bastante utilizadas no retorno de dados da base de dados para o *ViewModel*, que através do *DataBinding* injeta esses valores na *View*. As *queries* do tipo *LiveData* permitem assim, assincronamente, aceder à base de dados em operações que tipicamente necessitariam do recurso a *AsyncTasks* ou outros mecanismos de sincronismo de modo a realizar a *query*.

Na Figura 6-2 a) pode-se verificar uma realização de uma *query* do tipo *LiveData*, que irá expor uma *streams* de dados contínua que estará a ser recebida pelo trecho de código visível na figura b). Caso exista alguma alteração nos dados resultantes da *query* realizada, automaticamente a variável *mObservableGroups* irá ser atualizada, dando origem a alterações na respetiva *View*. Na Figura 6-2 c), do lado do *Fragment*, temos o *Observer* que tratará de colocar os dados recebidos da *query* na lista presente na interface visual. Este tipo de componente foi bastante utilizado no preenchimento de *RecyclerViews* de histórico com elementos da base de dados.

a)

```
@Query( value: "SELECT * FROM groups WHERE projName LIKE :projName")
fun loadGroupsByProjectName(projName: String) : LiveData<List<GroupAndAllComponents>>
```

b)

```
lateinit var mObservableGroups: LiveData<List<GroupAndAllComponents>>

fun setProjectName(projName: String) {
    mObservableGroups = db.groupAndAllAccessoriesDao().loadGroupsByProjectName(projName)
}
```

c)

```
viewModel.mObservableGroups.observe( owner: this,
        Observer<List<GroupAndAllComponents>> { groups ->
            historyRecyclerAdapter.addAll(groups as ArrayList<GroupAndAllComponents>)
            clearFab.setVisibleList(groups.size)
        })
```

Figura 6-2 - Exemplo de uma query no formato LiveData.

Room

O *Room* foi utilizado como "camada" de acesso à base de dados local da aplicação, de modo a poder realizar *queries*, inserções e apagar de maneira muito parecida a outro qualquer *DBMS* normal, não sendo necessária a implementação de cursores como seria necessário com a utilização de *SQLite* puro. Nas figuras seguintes, é possível ver a definição de uma entidade da base de dados através de um *POJO*, caso não fosse utilizado *Room*, a definição desta entidade seria feita de uma maneira programaticamente arcaica. Todas as entidades utilizadas neste projeto, foram definidas da mesma forma, e como é possível ver na imagem, toda a configuração do mecanismo de chaves, índices e *constraints* de chaves é realizado nestes ficheiros. Associado às entidades, é possível ter *Dao – Data Access Object* – que são as entidades que permitem o acesso, manipulação e remoção de dados da base de dados.

```
@Entity(tableName = SENSORS_TABLE_NAME,
        primaryKeys = ["mac", "groupName", "projName"],
        foreignKeys = [(ForeignKey(entity = GroupEntity::class,
                parentColumns = arrayOf("name", "projName"),
                childColumns = arrayOf("groupName", "projName"),
                onDelete = ForeignKey.CASCADE))],
        indices = [(Index( ...value: "mac", "groupName"))])
class SensorEntity(val mac: String,
                   val name: String,
                   var projName: String,
                   @ColumnInfo(name = "groupName")
                   var groupName: String,
                   var sensorType: SensorType) : Serializable
```

```
@Dao
interface SensorDao {
    @Query( value: "SELECT * FROM $SENSORS_TABLE_NAME")
    fun loadAllSensors() : LiveData<List<SensorEntity>>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    fun insertAll(devices: List<SensorEntity>)

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    fun insertSensor(sensorEntity: SensorEntity)

    @Query( value: "DELETE FROM ${DatabaseConstants.SENSORS_TABLE_NAME}")
    fun deleteAllSensors()
}
```

Figura 6-3 - Exemplo da definição de uma entidade e dos seus métodos de acesso aos dados, utilizando Room.

Em algumas entidades, uma delas visível na Figura 6-3, por vezes as variáveis podem ser de tipos diferentes dos tipos primitivos do *Java/Kotlin*, como é visível na variável *sensorType*, que é do tipo *SensorType*. Nestes casos, o *Room* fornece um mecanismo de conversores, que servirão para converter cada tipo de dados numa *String*.

ViewModel

O *ViewModel*, nas aplicações desenvolvidas, funciona como componente ponte entre a camada de dados das aplicações (base de dados e outros *providers*) e a camada visual. É o componente que também trata de toda a lógica da aplicação, deixando apenas para a *View* o trabalho de realizar animações visuais, ou outras operações relacionadas com a interface da aplicação.

Cada fragmento, ou atividade presente nestas aplicações tem um ficheiro de *ViewModel* associado, de modo a tratar de toda a lógica desse componente. A inicialização e associação do fragmento ou da atividade ao *ViewModel* é realizada no método *onCreateView (*Figura 6-4 *a))*. De modo a internamente tornar mais fácil a utilização do *ViewModel*, e também por questões de facilidade na análise de código caso esta aplicação seja continuada no futuro, foi criada uma interface que fragmentos que subscrevam a variáveis *Observable* do *ViewModel* devem implementar. Um diagrama de classes da utilização dessa interface é visível na Figura 6-5.

a)

```
override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {
    activity?.title = "Device Dashboard"
    (activity as FragmentToActivity).setNavState(R.id.dashboard_menu)
    setHasOptionsMenu(true)
    (activity as MainActivity).showBackButton( enable: false)

    viewModel = ViewModelProviders.of( fragment: this).get(DeviceDashboardViewModel::class.java)
```

b)

```
override fun subscribeUI() {
    viewModel.observableMissionDevices.observe( owner: this,
            Observer<ArrayList<Sensor>> { sensors ->
        sensors?.let { dashboardSensorsRecyclerAdapter.addAll(it) }
    })
}
```

Figura 6-4 - Implementação da inicialização do ViewModel, e método de subscrição a um Observable.

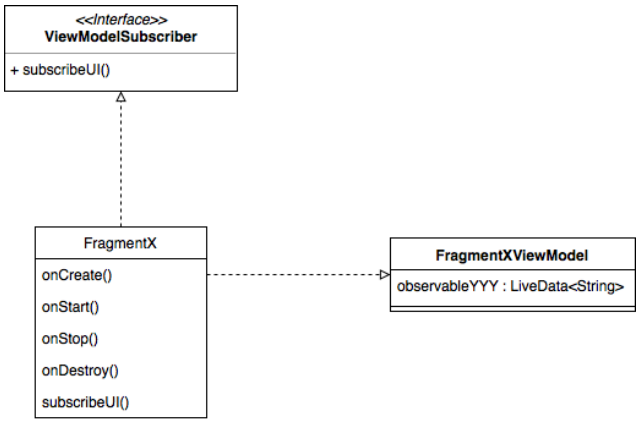


Figura 6-5 - Diagrama de classes demonstrativo da interface ViewModelSubscriber.

6.2 Recolhas em modo autónomo

Nesta subsecção serão revistos os processos de implementação mais importantes da aplicação VR-Unit. Esta aplicação permite realizar recolhas em modo autónomo, nas quais o utilizador controla a recolha, com os sensores que decidir utilizar, que podem ser dinamicamente adicionados ou removidos da aplicação. Para além disso, a aplicação também pode ser controlada pela aplicação VR-Remote, como se verá na secção 6.3.

6.2.1 Desenho do VR-Unit

O padrão *MVVM*, permite construir a aplicação orientada a eventos. Existe assim um fluxo de dados transversal a toda a aplicação, que permitirá propagar os dados recebidos pelos sensores aos ecrãs de visualização, base de dados, sistema de ficheiros e também para os módulos de comunicação externa, através do *Eventbus*. Esta abstração criada com a utilização do *Eventbus* como canal de comunicação interno entre

componentes, permite ver a aplicação quase como um *broker* de mensagens, onde temos componentes que subscrevem a mensagens produzidas por outros componentes, tornando a aplicação mais desacoplada e modular.

A aplicação é composta por uma atividade principal que trata do *inflate* de vários fragmentos consoante a navegação efetuada na aplicação. O componente que trata da navegação da aplicação é um *NavigationDrawer*, que consoante a escolha do utilizador irá abrir o respetivo fragmento (Figura 6-6). Um desenho de componentes interno deste género, suportado por uma atividade que com o auxílio de um *NavDrawer*, é cada vez mais encorajado pelo *Android*, que com o recente lançamento do componente de *Navigation*[20] nos *architecture components* marca a posição na preferência por uma organização deste género, ao invés de múltiplas atividades. O *NavDrawer* está dividido em 3 secções, seguindo a ordem vertical da imagem: recolha e configuração (secção a)), consulta de histórico (secção b)) e configurações (secção c)).

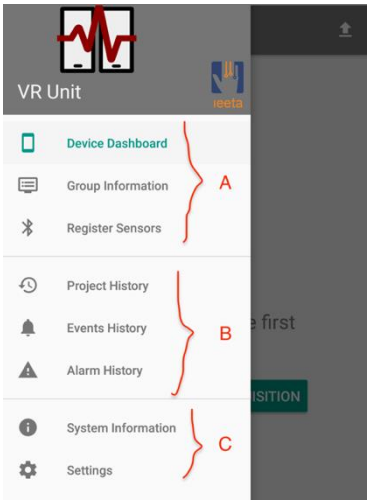


Figura 6-6 - Menu de navegação da aplicação.

Internamente, a aplicação é também composta por múltiplos Serviços que trabalham em *background*, de modo a realizar operações mais pesadas, ou que simplesmente necessitem de correr à parte do fio normal de execução da aplicação, tais como:

- Ligação e comunicação com sensores;
- Operações de rede (ligação a canais de comunicação externos, *upload* de dados para a *cloud*);
- Operações de escrita no sistema de ficheiros local;
- Mecanismos de reconexão e verificação de serviços internos.

Visualmente, foram seguidos os padrões de *Material* design para *Android*, que corresponde a um padrão que contém regras para o desenho e interação dos componentes dentro das aplicações *Android* [59]. São característicos deste padrão o uso de *FloatingActionButtons*, *App Bar*, *NavigationDrawer* e *CardViews*, utilizados no desenvolvimento desta aplicação, que poderão ser visualizados nas imagens presentes desta

[20] https://developer.android.com/topic/libraries/architecture/navigation/

subsecção. O desenho das interfaces foi realizado com recurso a *ConstraintLayout*, que permite um aumento de performance na renderização dos *layouts*, bem como são evitados *nested layouts* de *Relative Layouts* dentro de *Relative layouts*, entre outros.

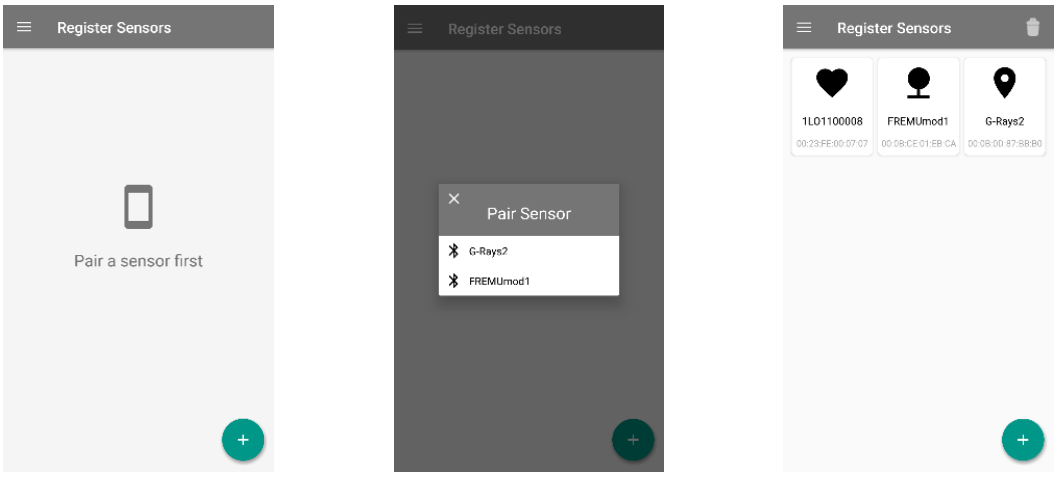
6.2.2 Interações Suportadas

A interação principal de um utilizador com esta aplicação é a recolha autónoma de dados. Esses dados são recolhidos por sensores que podem ser "registados" na aplicação, caso haja suporte para os mesmos. Noutro cenário, a aplicação funciona como ponto de recolha num grupo organizado, que é controlado a 100% por uma aplicação externa. Nessa situação é possível visualizar o histórico de várias variáveis ligadas à aquisição de grupo.

Recolha autónoma e configuração

O utilizador deve registar sensores na aplicação de modo a poder utilizá-los, podendo-o fazer no ecrã da Figura 6-7 a). De modo a ser possível o registo, o telemóvel precisa previamente de estar emparelhado aos mesmos por *Bluetooth*, caso esteja, o utilizador terá acesso a uma lista de sensores aos quais está associado (Figura 6-7 b)). Esta lista apenas apresenta dispositivos de *Bluetooth* que possam ser sensores, o *deviceClass* do *Bluetooth* é filtrado de modo a impedir que sejam disponibilizados ao utilizador outros tipos de dispositivos que não sejam sensores. É possível realizar o registo de sensores de *Bluetooth*, mas também de *Bluetooth Low Energy*.

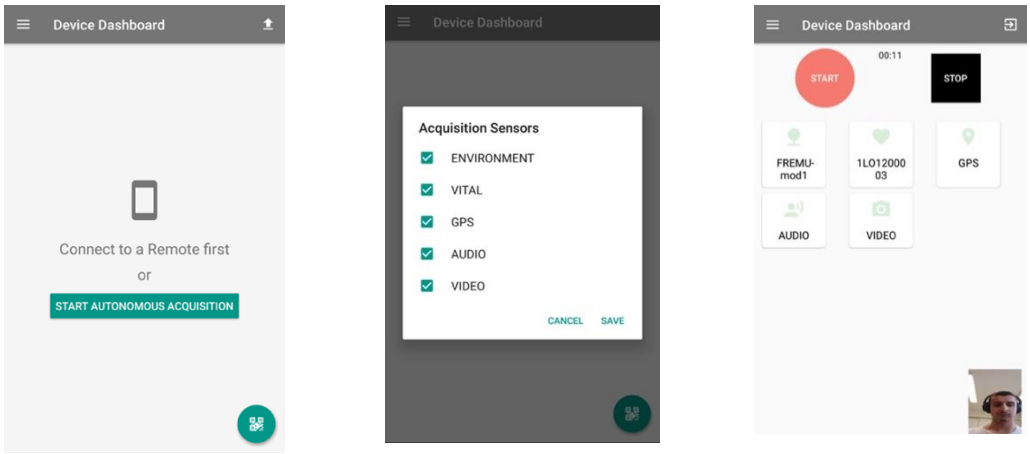
Após selecionar um, apenas tem de indicar o tipo de sensor ao qual aquele sensor corresponde, por exemplo se é um sensor de sinais vitais, ou GPS, ou de ambiente, entre outros. Os sensores selecionados irão ficar registados em memória como sensores possíveis de realizar aquisições (Figura 6-7 c)).



a) b) c)

Figura 6-7- Registo de sensores na aplicação.

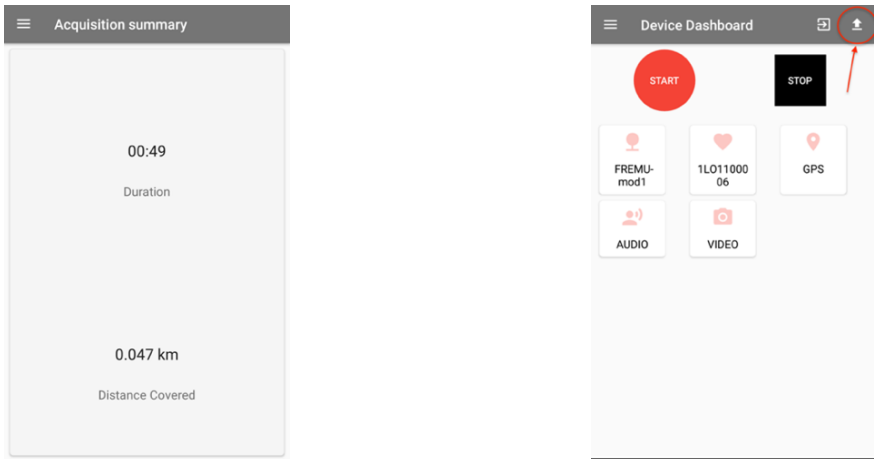
De modo a realizar-se uma aquisição autónoma, no ecrã principal da aplicação (Figura 6-8 a)), o utilizador deverá escolher essa opção, sendo-lhe disponibilizado a lista de sensores registados internamente na aplicação (Figura 6-8 b)), da lista disponibilizada o utilizador poderá escolher livremente que sensores quer que entrem na sua aquisição. Após a seleção, terá acesso a um *RecyclerView* em grelha, com todos os sensores que escolheu, bem como dois botões que permitirão iniciar ou parar a aquisição. As cores dos sensores na grelha, variam consoante o estado de ligação ao sensor, transitando entre verde (ligado e a recolher), vermelho (desligado), amarelo (tentar ligar), na Figura 6-8 c) todos os sensores se encontram ligados e a recolher dados.



a) b) c)

Figura 6-8 - Configuração e realização de aquisição autónoma.

No final de cada aquisição, o utilizador terá à sua disposição um ecrã de sumário da recolha realizada, onde poderá ver a duração da aquisição que realizou bem como a distância percorrida durante esse período, caso tenha selecionado o sensor de GPS, como é visível na Figura 6-9 a). Também poderá, após ter realizado pelo menos uma aquisição, enviar os dados para o armazenamento *cloud*, sendo apenas necessário pressionar o botão de *upload*. Todos os dados recolhidos serão comprimidos por um Serviço dedicado e posteriormente a isso enviados para um diretório próprio, no armazenamento *cloud*, Figura 6-9 *b)*.



a) b)

Figura 6-9 - Sumário de aquisição e Upload de dados recolhidos.

Visualização de dados

Cada tipo de sensor tem associado a si um fragmento para visualização. Este visualizador é atualizado em tempo real com os dados enviados pelos sensores. De modo a fazer essa atualização dos dados demonstrados, a biblioteca de gestão dos sensores, por cada tranche de dados que é recebida envia uma mensagem pelo *Eventbus* que irá ser recebida pelo *ViewModel* de cada visualizador (Figura 6-10 a)). As variáveis *Observable* presentes nesse *ViewModel* automaticamente com o recurso ao módulo de *Data Binding* atualizam o ecrã do dispositivo, mostrando os valores ao utilizador, visíveis na Figura 6-10 b).

a)

```
@Subscribe(sticky = true)
fun onValueReceived(bitalinoValuesEvent: BitalinoValuesEvent) {
    gson.fromJson(bitalinoValuesEvent.values, BitalinoData::class.java).let {
        onUiThread {
            viewModel.observableBitalinoData.set(it)
        }
    }
}
```

b)

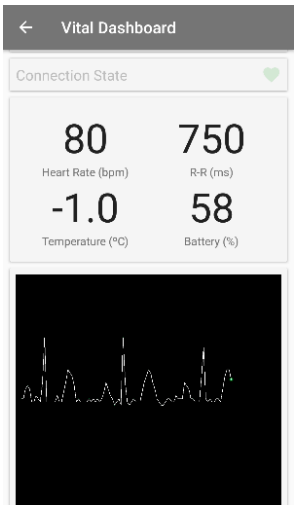


Figura 6-10 - Receção de dados em tempo real e injeção de dados no fragmento através de DataBinding.

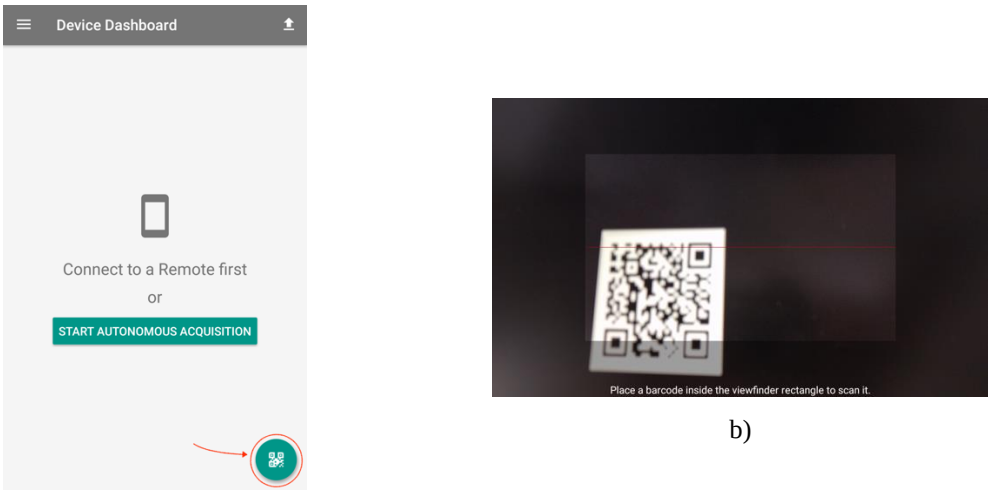
Caso o sensor tenha batimento cardíaco, como o da figura anterior, o utilizador terá acesso a um gráfico que permitirá ver a variação dos valores de *ECG* desse mesmo sensor, no caso de ser um *GPS* será possível ver um mapa com a localização atual. Noutros tipos de sensor apenas será possível visualizar os valores de cada variável recolhida.

Processos de associação a recolhas em grupo

Para estabelecer a associação entre os vários *VR-Unit* dos participantes e o *VR-Remote* de controlo há 2 opções: utilizar o *Bluetooth* ou através da leitura de um código QR. O serviço de emparelhamento *Bluetooth* que corre na aplicação após esta ser iniciada irá receber um pedido de estabelecimento de um *socket Bluetooth*, enviado pelo *VR-Remote*. Nesse *socket* irão ser trocadas mensagens, o *VR*-Remote envia o IP do *MessageCollector* e o *token* de projeto/estudo, e a aplicação VR-Unit o seu identificador de hardware, após

isto, é estabelecida a ligação. A realização desta operação é transparente ao utilizador, visto que a comunicação é processada por um Serviço, que desde que a aplicação é ligada está à escuta de eventuais comunicações de associação.

Caso seja escolhido emparelhamento por código QR (Figura 6-11 a)), na leitura, a aplicação receberá o IP do *MessageCollector* e o *token* do projeto/estudo, estabelece ligação ao mesmo e envia uma mensagem com o seu identificador. O leitor de códigos QR utilizado foi o fornecido pela biblioteca de leitura de códigos da *Google*, onde através da utilização de uma API muito simples é possível ler e interpretar códigos *QR* e códigos de barras, a utilização do leitor está representada na Figura 6-11 b).



a) b)

Figura 6-11 - Opção de leitura de código QR e realização de uma leitura.

Consulta do histórico

Nesta aplicação, é possível consultar o histórico de alarmes gerados pelos valores recolhidos pelos sensores, eventos marcados pela aplicação VR-Remote e também os estudos e grupos de recolha onde a aplicação já esteve presente, sendo possível visualizar as configurações aplicadas pelos mesmos, ou informações sobre os restantes membros do grupo de recolha.

Essas listas de histórico, como a presente na Figura 6-12, são preenchidas por conteúdo da base de dados e foram utilizadas *RecyclerView* sob a forma de lista, que permitem ao utilizador fazer *scroll* vertical pelos elementos da lista. Todas as células correspondentes às *RecyclerView* foram desenhadas utilizando *ConstraintLayout* e a injeção dos dados para as mesmas é feita através de *Data Binding*, não sendo, portanto, necessário na parte dos *Adapters* ou *Fragments* fazer a injeção explícita dos dados para os componentes visuais. No *ViewHolder* de cada *RecyclerAdapter*, apenas é necessário associar o item de *DataBinding* da célula ao tipo de dados correspondente (Figura 6-13 a)), delegando o trabalho de injeção de dados para o mesmo. A variável do tipo de dados da célula, encontra-se declarada na Figura 6-13 b).

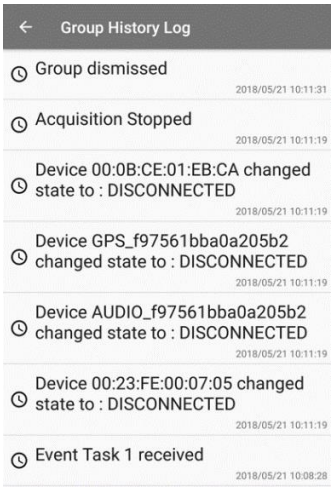


Figura 6-12 - Histórico de interações durante uma recolha.

a)

```
class ViewHolder(val view: View, private val itemDeviceBinding: RecyclerItemNotificationBinding) :
        RecyclerView.ViewHolder(view) {
    fun bind(notification: EventHistoryEntity) {
        itemDeviceBinding.notification = notification
    }
}
```

b)

```
<layout xmlns:tools="http://schemas.android.com/tools"
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    tools:keep="@layout/recycler_item_history_event_uplevel">

    <data>
        <variable
            name="event"
            type="com.bitmob.vrunit.shared.db.entity.EventHistoryEntity"/>
```

Figura 6-13 - Aplicação de DataBinding em células de uma RecyclerView.

6.2.3 Serviços de recolha de dados

A recolha de dados é efetuada por serviços, no entanto, toda a lógica de receção de dados está presente na biblioteca criada para a gestão e manipulação de sensores que será abordada numa secção mais à frente. A ligação a esses serviços e respetiva configuração, é realizada através de *Intents* onde são configurados os parâmetros necessários, um exemplo dessa configuração é visível na figura seguinte.

```
Intent(this, VitalService::class.java).apply {
    putExtra(IntentValues.SENSOR_ADDR_INT, it.mac)
    putExtra(IntentValues.DEV_ID_INT, viewModel.getDeviceId())
    putExtra(IntentValues.ALARMS_INT, viewModel.getEcgAlarms())
    startService(this)
}
```

Figura 6-14 - Estabelecimento de ligação a um sensor através da biblioteca desenvolvida.

No entanto, a ligação a esses sensores pode ser perdida durante a aquisição, devido a falhas de conectividade por parte do sensor, dado isso, existe nesta aplicação um Serviço que verifica a conexão a cada sensor de 10 em 10 segundos através de um *Runnable*, caso a ligação que perca, este serviço tenta efetuar uma nova ligação ao sensor. Caso a tentativa de reconexão ao sensor falhe a primeira vez, será realizada uma nova tentativa 10 segundos depois, que será seguida de outra 20 segundos depois, em intervalos de tempo que crescem para o dobro até fazer um máximo de 10 minutos, de modo a não estar insistentemente a realizar uma tentativa de conexão.

Existe apenas um tipo de dados recolhido, que por razões de integração do *SDK*, não foi possível incluí-lo na biblioteca na forma de um serviço, devido a ser exigido a este SDK ser integrado num componente com interface visual, ou seja, atividade ou fragmento. Este tipo de recolha de dados corresponde ao reconhecimento de emoções através da visualização e análise da expressão facial, a análise é realizada pelo SDK da *Affectiva*[21], que permite realizar a recolha de emoções em tempo real, neste trabalho em concreto estão a ser analisados os níveis de alegria, tristeza, raiva, medo e desgosto.

6.3 Recolhas de grupos de participantes

Nesta subsecção serão abordados os métodos aplicados no desenvolvimento da aplicação *VR-Remote*. Esta aplicação funciona como ponto de controlo e monitorização de sessões com vários participantes a utilizar o módulo individual, o *VR-Unit*. A capacidade de monitorização desta aplicação, permite uma visualização em tempo real dos dados recolhidos por cada nó agregador, bem como uma visualização de histórico de alguns acontecimentos marcantes nas recolhas como eventos, alarmes e anteriores configurações.

6.3.1 Desenho do VR Remote

A aplicação *VR-Remote* apresenta um desenho interno orientado a eventos suportado pelos mecanismos já apresentados como comuns às aplicações de *DataBinding*, *ViewModel*, *EventBus* e *LiveData/Observables*. Nesta aplicação o fluxo de dados interno depende muito das mensagens recebidas pelo *MessageCollector*, dado que o objetivo principal desta aplicação é o controlo e a monitorização.

A dependência da aplicação no *MessageCollector* não é só em termos de receção, onde os dados recebidos são propagados internamente, as mensagens enviadas por esse canal de comunicação permitem realizar toda a orquestração das sessões de recolha. A importância desse componente é de tal ordem, que nesta aplicação é importante existirem mecanismos de resistência a falhas, segurança e divisão correta de estudos que permitam controlar as aquisições de modo fiável, o que desde já permite decifrar que a aplicação terá uma componente forte de serviços em *background* que suportem este tipo de mecanismos.

[21] https://www.affectiva.com/product/emotion-sdk/

A aplicação é composta por uma única atividade, os ecrãs da aplicação correspondem a fragmentos que são mostrados ou escondidos pelo *FragmentManager*, suportados pelo componente de navegação principal, o *NavigationDrawer*. Este componente está dividido em 3 secções: definição dos parâmetros de recolha (Figura 6-15 a)), consulta de histórico (Figura 6-15 b)) e configurações (Figura 6-15 c)). Na primeira secção, a mais importante no contexto da aplicação, o investigador transitará entre as fases de configuração até chegar à fase de recolha e controlo, por ordem semântica, onde primeiro terá de definir o estudo, de seguida compor o grupo e por fim controlar. Em termos gráficos, foi optado por seguir o padrão *Material.*

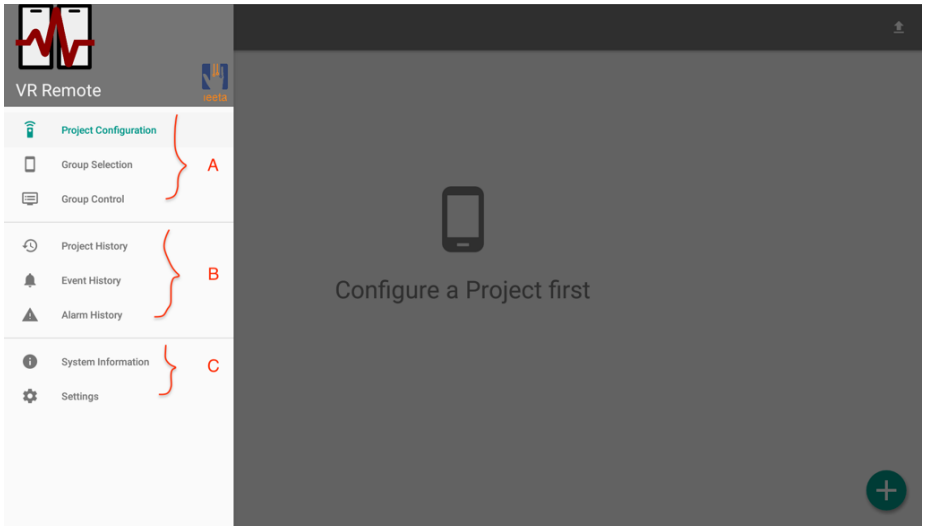


Figura 6-15 - Navigation Drawer da aplicação VR-Remote.

6.3.2 Implementação da Coordenação entre nós

A utilização de um broker de mensagens, *MessageCollector*, para mediar a comunicação interna entre as aplicações, mas também como comunicação com eventuais sistemas externos integrados, permite criar uma coordenação entre os nós vital na configuração e controlo de uma recolha de psicologia, ou de outro tipo de recolhas de grupos.

O *broker* utilizado é o Mosquitto MQTT, que está a correr sobre um *container Docker*, configurado de modo a ter autenticação com *username* e *password*, prevenindo eventuais acessos de intrusos. Este container é iniciado no arranque do sistema operativo da máquina onde está *deployed*.

O *MessageCollector* encontra-se dividido em tópicos, visíveis na Tabela 6-1, apesar de ser utilizado um número relativamente alargado de tópicos, permite um controlo e uma diferenciação muito maior em todas as mensagens, tornando mais fácil a receção e tomada de decisão nas diversas aplicações. Os tópicos estão classificados com diferentes qualidades de serviço, tendo em conta a importância de cada tipo de mensagem, que pode ser dividida em 3 níveis:

- 0: modo de entrega rápido, mas sem garantia de entrega.

- 1: modo de entrega com garantia de destino de nível básico, onde a mensagem é apenas confirmada uma vez.
- 2: mensagem é entregue com uma confirmação dupla, garantia de destino de nível elevado.

Nome do tópico	**Propósito**	**Qualidade de Serviço**
VR_TOPIC/ACK	Sinalizar a receção de Eventos/outras mensagens por parte do cliente.	2
VR_TOPIC/EVENT	Envio de eventos.	2
VR_TOPIC/CONF	Envio de configurações.	2
VR_TOPIC/ACQUISITION	Envio de comandos correspondentes ao controlo da recolha.	2
VR_TOPIC/PING	Envio de comandos de Ping.	2
VR_TOPIC/DATA_SEND	Envio de mensagens que sinalizem o pedido de dados a um dado cliente.	1
VR_TOPIC/ALARMS	Envio de alarmes despoletados nos clientes e configuração.	1
VR_TOPIC/UPLOAD	Envio da ordem de upload de dados.	1
VR_TOPIC/CANCEL	Cancelamento da configuração de grupo.	1
VR_TOPIC/QR INIT	Envio de informações do cliente para o controlador na fase de configuração.	1
VR_TOPIC/DISMISS	Tópico utilizado para terminar o grupo.	2
VR_TOPIC/LEAVE GROUP	Tópico utilizado pelo cliente de modo a avisar o controlador que abandonou o grupo na fase de recolha.	2

Tabela 6-1- Tópicos do MessageCollector *utilizados pela aplicação controladora*.

Quando os dispositivos com a aplicação *VR-Unit*, realizam o emparelhamento com o *VR-Remote* de modo a alistar-se num grupo de recolha, é entregue um *token* que identifica o estudo referente à sessão de recolha que vão realizar. Este, funciona como elemento distintivo entre estudos, permitindo a utilização do *MessageCollector* para estudos simultâneos diferenciados. O *token* corresponde a um *digest* entre o nome do estudo, o tipo do estudo, a *timestamp* atual e o ID do telemóvel que contém o *VR-Remote*. Para além de diferenciar estudos, tem um propósito de segurança visto que, mesmo se algum intruso conseguir adivinhar a password de autenticação do *MessageCollector*, sem um *token* deste género não conseguirá subscrever aos tópicos, não conseguindo aceder às mensagens.

A ligação ao *MessageCollector* é feita em *background*, num serviço, através da utilização da *API* fornecida pela *Eclipse* que contém métodos assíncronos. O envio de mensagens é suportado internamente pelo *EventBus*, onde em qualquer componente da aplicação, uma mensagem interna poderá ser enviada por este mecanismo, sendo que o *Serviço* que trata do envio de mensagens no *MessageCollector*, ao recebê-la faz o desacoplamento da mensagem, transforma-a em *bytes*, coloca-a no tópico específico e procede ao envio.

O VR-Remote, como entidade principal da recolha, não pode perder a ligação ao *MessageCollector* durante esse período, colocando em causa o controlo e monitorização da sessão. No entanto, eventuais falhas de rede poderão acontecer, de modo a prevenir essas situações, com as *callbacks* fornecidas pela biblioteca utilizada na manipulação do *MessageCollector*, que fornecem instantaneamente *feedback* em caso de perda de ligação, foi criado um serviço de verificação e reconexão ao *MessageCollector* em caso de falha. Esse mecanismo contém um *Runnable* que irá tentar restabelecer a conexão (Figura 6-16), de modo a mesmo que a ligação seja perdida durante um período, eventualmente a aplicação se consiga voltar a ligar, o que não reduz o impacto na recolha. Para além deste serviço de verificação e reconexão, por configuração, a biblioteca tem um mecanismo de *keepalive*, em que de 8 em 8 segundos, o broker e o cliente presente nesta aplicação trocam uma mensagem de *keepalive*. Caso essa mensagem não seja recebida por parte da aplicação, a ligação é perdida e o *runnable* de reconexão inicia.

```
private val brokerRunnable = object : Runnable {
    private var attempt = 0
    override fun run() {
        val connectionState = sp.getBoolean(BROKER_CONNECTED, false)
        if (!connectionState) {
            stopService(Intent(applicationContext, BrokerConnectionService::class.java))
            startService(Intent(applicationContext, BrokerConnectionService::class.java))
            if (attempt < attemptsIntervals.size)
                brokerHandler.postDelayed(this,
                        attemptsIntervals[attempt++].toLong())
            else
                brokerHandler.postDelayed(this,
                        attemptsIntervals[attemptsIntervals.size-1].toLong())
        } else {
            attempt = 0
            brokerHandler.postDelayed(this, 20000) // 20s
        }
    }
}
```

Figura 6-16 - Serviço de reconexão e parâmetros de ligação ao MessageCollector.

Outra das ligações que é importante de verificar regularmente é o estado de ligação entre os dispositivos com a aplicação VR-Unit e o *MessageCollector*. Desta forma, a partir do *VR-Remote*, o investigador também terá *feedback* sobre a conexão individual de cada cliente ao sistema, podendo verificar no telemóvel em questão, caso tenha perdido a ligação, se de facto tem algum problema de rede. De modo a implementar este mecanismo de verificação, cada VR-Unit em período de recolha terá de enviar uma mensagem de *Keepalive*, através do tópico VR-TOPIC/PING da Tabela 6-3. Esta mensagem deverá ser entregue de 5 em 5 segundos, caso não seja recebida essa mensagem, o *Remote* marcará o dispositivo como "desligado" do grupo de recolha, informando o investigador da falha. A aplicação VR-Unit, entrará no modo de tentativa de restabelecer ligação, quando acontece esse problema, e quando a ligação for restabelecida irá ser sinalizada como "ligada" de novo, dando esse *feedback* ao investigador. Este mecanismo está implementado num Serviço, que permite em *background* fazer a espera por este tipo de mensagens, em caso de falha, o *Eventbus* propagará o erro pela aplicação permitindo criar o *feedback* e registar esse tipo de falhas no histórico de *log* das sessões, que está alojado na base de dados.

6.3.3 Interações suportadas

Preparação e configuração do protocolo de recolha

O primeiro passo na configuração de uma recolha corresponde à definição e configuração do estudo/projeto. Nesta fase, o investigador ou responsável terá de definir o nome do seu estudo e o tipo. Opcionalmente poderá definir o IP do *MessageCollector* e também do *broker* de ligação com o novo sistema de visualização do projeto *VR2Market*. Uma configuração possível encontra-se atribuída na Figura 6-17.

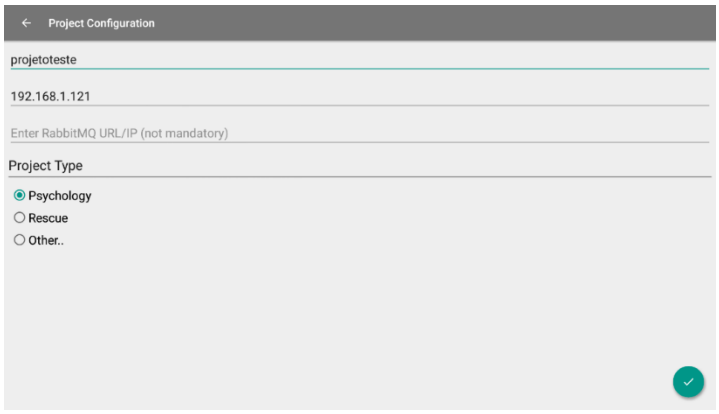


Figura 6-17 - Definição de um projeto.

No final desta fase, como foi abordado na subsecção anterior, o *token* do projeto é criado, podendo então o responsável partir para a definição dos grupos de aquisição onde poderá adicionar os elementos, correspondentes aos participantes.

Os nós correspondentes a dispositivos, ou participantes, com a aplicação VR-Unit podem-se associar ao *MessageCollector*, onde ficam inseridos num grupo pertencente ao estudo configurado. Existem duas formas de uma aplicação *VR-Unit* ficar associada a um *VR-Remote*, a primeira forma é por *Bluetooth* e a segunda por leitura de um código *QR*, ficando o método à escolha do investigador. Quando o responsável decide emparelhar por *Bluetooth*, será aberto um *socket* de *Bluetooth* entre as duas aplicações, de modo a trocarem informações necessárias. Toda esta comunicação é efetuada numa *thread* à parte de modo a não bloquear a *Main thread* da aplicação e após este processo, o cliente liga-se ao *MessageCollector* com os parâmetros de autenticação definidos e o *token* de projeto. A lista de dispositivos possíveis de serem emparelhados por *Bluetooth* é filtrada de modo a aparecer ao utilizador apenas telemóveis, e não outro tipo de dispositivos de *Bluetooth*, como é visível na Figura 6-18.

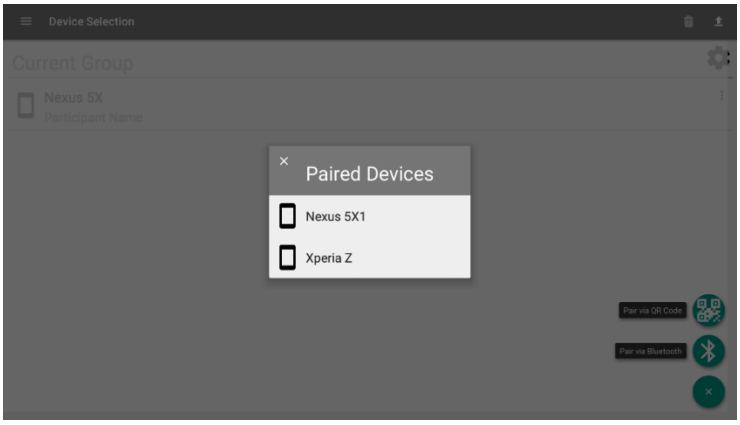


Figura 6-18 - Emparelhamento por Bluetooth.

Caso o responsável decida associar os dispositivos dos participantes por código QR, o código gerado pela aplicação irá conter o endereço IP juntamente com o *token* gerado na criação do projeto. É possível ver o ecrã que exibe o código QR na figura seguinte.

Figura 6-19 - Exemplo do código QR gerado.

Após o investigador ter formado o grupo que irá participar na sessão de recolha, poderá atribuir uma configuração a esse grupo. A configuração permite ao investigador definir o nome do grupo de sessão, o tipo de sensores sobre os quais irá realizar recolha, bem como o número e o nome dos eventos que será possível marcar. O investigador também terá de colocar o seu nome, de modo a poder futuramente aceder aos dados recolhidos na plataforma *web*. Uma configuração possível encontra-se na Figura 6-20. Estas configurações serão enviadas por um tópico no *MessageCollector*, que todos os dispositivos que o investigador adicionou ao grupo irão receber ficando assim completamente associados e configurados de modo a iniciar a recolha.

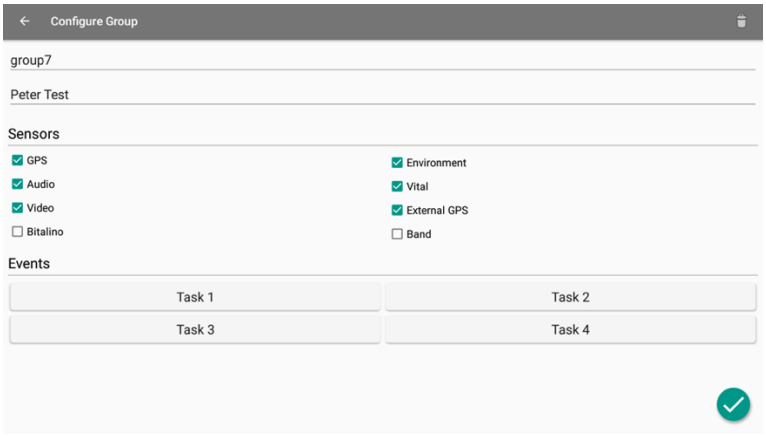


Figura 6-20 - Exemplo de configuração atribuída a um grupo.

A comunicação com a *RESTful API* desenvolvida é realizada nesta fase, visto que é na altura em que todos os parâmetros que definem um estudo e um grupo estão definidos. Para o fazer é utilizado o *Retrofit*, que irá comunicar com essa API de forma assíncrona, devolvendo o resultado da comunicação através de *callbacks*.

Por último, opcionalmente, o investigador poderá definir valores limite sobre o qual poderá ser alertado caso os dispositivos ultrapassem esses valores. Esses valores são configuráveis, podendo ser alterados a meio da recolha, ou entre recolhas, o ecrã de configuração é visível na Figura 6-21. Os valores configurados, são enviados para todos dispositivos através do *MessageCollector*, sendo que esses valores são colocados na configuração dos serviços de ligação ao sensor, produzindo os alertas que serão encaminhados de volta para o *VR-Remote*.

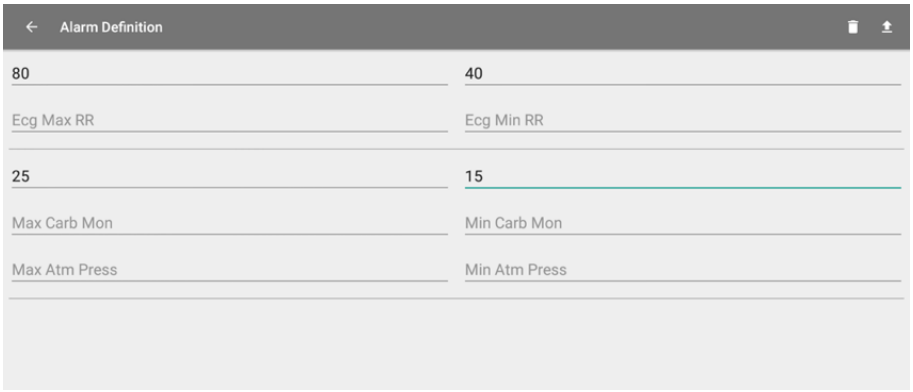


Figura 6-21 - Definição de alarmes.

Monitorização e controlo da recolha

O investigador, na fase de recolha, terá ao seu dispor várias ferramentas e mecanismos de *feedback* de modo a monitorizar e controlar o processo. Do lado esquerdo da interface terá disponíveis os eventos, momentos no protocolo de experiência, que definiu na configuração de sessão, que poderá marcar com o *long click* na célula do evento temporal. Na parte central da interface, terá uma *RecyclerView* composta por células

que simbolizam os participantes da sessão, e na parte inferior, os dois botões de controlos, que ao ser pressionados marcam o início ou o fim do período de recolha. A interface principal pode ser vista na figura seguinte.

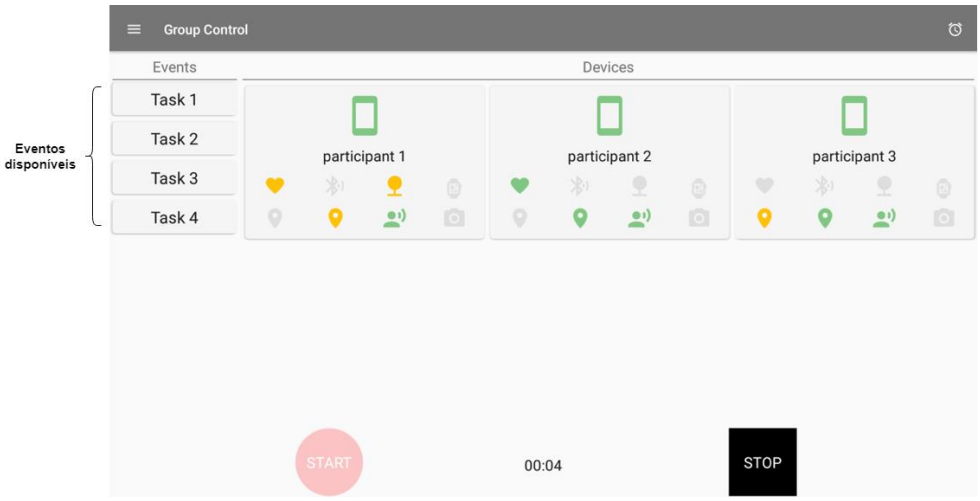


Figura 6-22 - Interface principal de controlo disponível ao investigador.

A grelha de dispositivos permite ao investigador ter acesso ao estado de ligação de cada dispositivo ao sensor que está a recolher dados, mas também, o estado de ligação do *VR-Unit* ao *MessageCollector*. Os códigos de cor serão abordados na tabela seguinte:

Cor	**Ligação ao sensor**	**Ligação ao *MessageCollector***
Verde	Ligado e a recolher	Ligado
Amarelo	A tentar ligar	-----
Vermelho	Desligado	Ligação perdida
Cinzento	Sensor não utilizado	-----

Tabela 6-2 - Códigos de cor do estado de ligação.

O *feedback* visual da cor de cada componente, transita de acordo com a informação recebida no *MessageCollector*. As aplicações *VR-Unit* enviam a informação do estado de ligação dos sensores, e o *EventBus* propaga essa informação até ao *ViewModel* deste fragmento, que terá o trabalho de alterar a informação associada aos dispositivos, alterando a cor dinamicamente. O estado de ligação do VR-Unit ao *MessageCollector*, altera o *feedback* do estado de conexão caso o dispositivo não envie a mensagens de *ping* no intervalo de tempo definido, que neste caso é de 5 segundos.

O investigador, poderá ver com mais detalhe a informação de cada dispositivo, se carregar na célula correspondente ao mesmo. Na Figura 6-23 podemos ver um exemplo de uma *dashboard* individual de um participante, esta *dashboard* apenas mostra os sensores que o participante está a utilizar na recolha, onde também é possível ver o *feedback* visual do estado da conexão. Esta *dashboard* é composta por uma *RecyclerView*, cuja as células se forem clicadas permitem ao investigador navegar para o fragmento de visualização do sensor em questão.

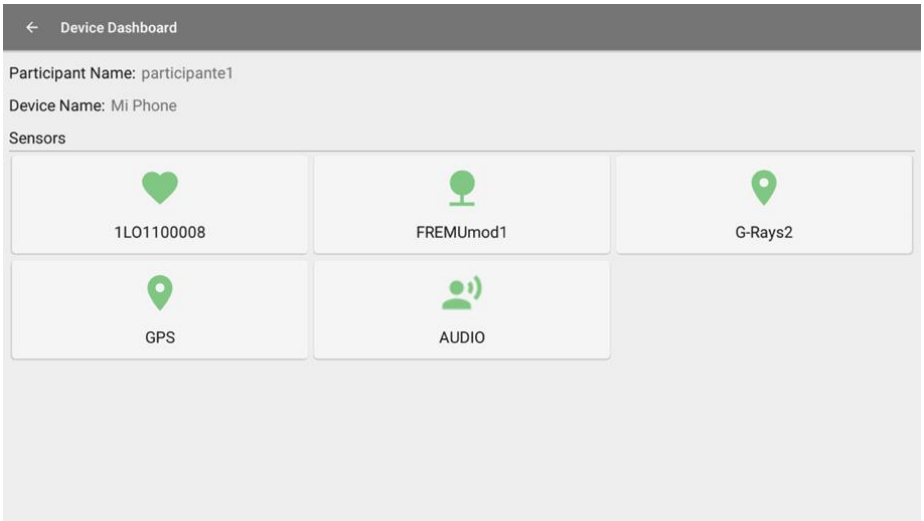


Figura 6-23 - Dashboard individual de um participante.

Durante o período de aquisição, na receção de um alarme sobre os valores definidos como limite, será gerado um *feedback* visual na célula do dispositivo que gerou o alarme juntamente com o aparecimento a vermelho do número de alarmes que já aconteceram. Estes alarmes, são enviados pela aplicação *VR-Unit* que o despoletou, através do *MessageCollector*, internamente, essa mensagem é propagada até ás *dashboards* e fragmento de visualização do sensor em questão. Um exemplo do *feedback* visual da célula a alterar de cor, bem como o número de alarmes que já ocorreram, é visível na figura seguinte.

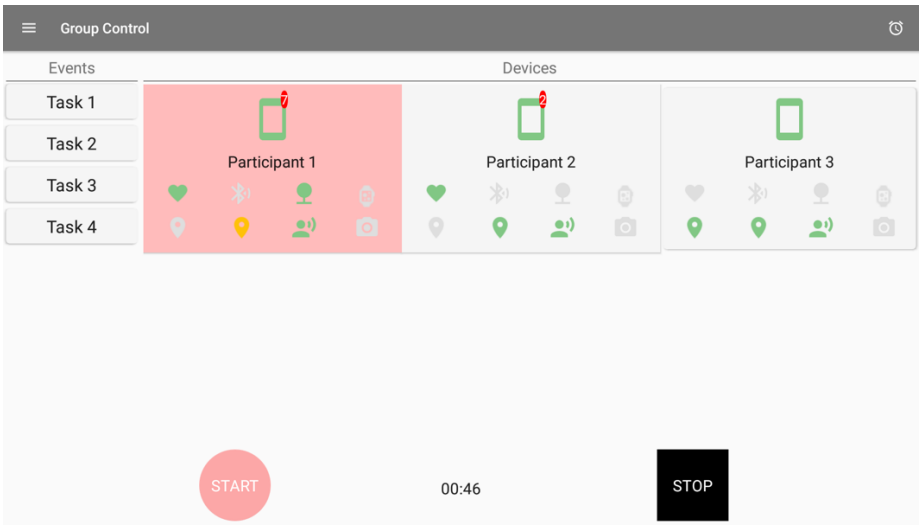


Figura 6-24 - Feedback visual de alarmes.

No final da aquisição, o investigador poderá comandar o *upload* dos dados de aquisição para a *cloud*. Na *dashboard* de controlo, após parar a aquisição, o botão de *upload* irá aparecer possibilitando o envio do comando de *upload* a todos os *VR-Unit*. Na receção as aplicações *VR-Unit* e também o *VR-Remote* comprimem os ficheiros da aquisição e colocam-nos na pasta respetiva do estudo, na *cloud*. Será também disponibilizado ao investigador um ecrã de sumário de aquisição, como é possível ver na figura seguinte,

onde é possível ver o número de eventos marcados, número de alarmes ocorridos, tempo de sessão e número de desconexões.

Figura 6-25 - Sumário de sessão de aquisição.

Visualização de dados

Nesta aplicação é possível visualizar os dados recebidos por cada aplicação *VR*-Unit enquanto decorre a recolha. No fragmento de visualização de cada tipo de sensor, é efetuado um pedido de envio de dados, que terá de ser renovado periodicamente, ao dispositivo a que pertence esse sensor. Os dados são recebidos por subscrição ao tópico de receção de dados por parte do *VR*-Remote, sendo o *EventBus* o responsável por os propagar até ao *ViewModel* de cada fragmento visualizador. Na Figura 6-26 temos um exemplo desse fluxo, a variável *observableGpsData*, no método de subscrição do *EventBus* é preenchida, aparecendo os dados automaticamente nos campos do fragmento.

```
<TextView
    android:id="@+id/altitudeTextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@{viewModel.observableGpsData.alt}"
    android:textColor="@color/abc_primary_text_material_light"
    android:textSize="40sp" />
```

```
class GpsViewerViewModel(application: Application) : AndroidViewModel(application) {
    var observableGpsData = ObservableField(LocationData())
```

Figura 6-26 - Injeção de dados recebidos por DataBinding na TextView.

Figura 6-27 - Fragmento de visualização de sinais vitais.

No fecho deste fragmento, irá ser enviada uma mensagem de controlo de modo à aplicação cliente não enviar mais tranches de dados. O mecanismo de renovar o pedido de dados periodicamente permite manter o fluxo de dados ativo, mas também permite impedir o envio de dados caso ocorra algum problema na conexão entre os dispositivos, ou em caso de ocorrer algum problema com a aplicação controladora.

Consulta do histórico

Na aplicação *VR-Remote*, como suporte ao investigador, é fornecido uma maneira de consultar o histórico de eventos já marcados, organizado por estudo e aquisição, alarmes despoletados organizado por participante, e ainda, configuração e *log* de aquisição, organizado por estudo e aquisição.

Cada um desses elementos de histórico é composto por *RecyclerViews* que podem assumir uma disposição em lista, ou em grelha. Na Figura 6-28 é possível ver dois exemplos de consulta de histórico. Os dados apresentados nestes componentes de visualização de histórico, são provenientes da base de dados local da aplicação, e devido à utilização de *queries* do tipo *LiveData* em cada um destes elementos, é possível visualizar alterações em tempo real nas listas. Por exemplo, quando um alarme é despoletado, é possível verificar o aparecimento dele, sem ser necessário nenhum *refresh* implícito por parte do utilizador.

a)

b)

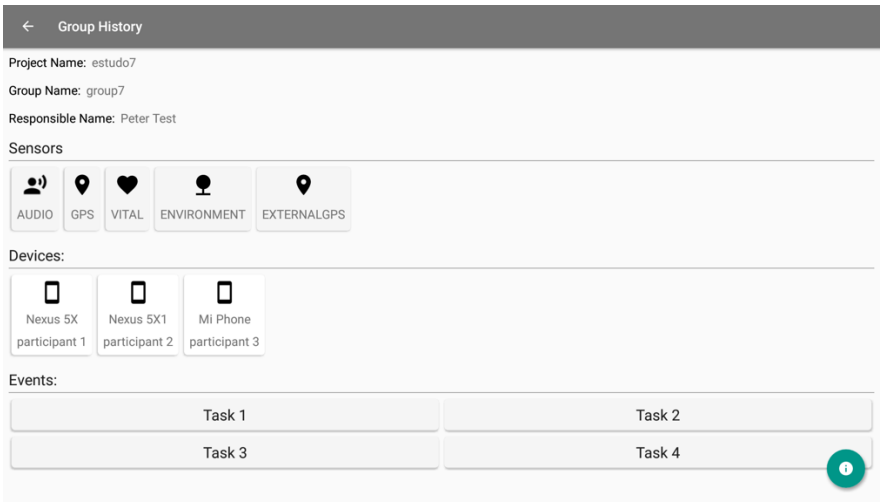


Figura 6-28 - Histórico de estudos, e informação associada a um grupo de recolha.

6.4 Acesso e visualização dos dados recolhidos - backend

Neste conjunto de aplicações agregadoras, os dados são o elemento chave. Cada aplicação *VR-Unit* consegue reunir milhares de amostras de dados, que devem ter uma organização própria, estando também num formato adequado à leitura por outro tipo de *software*. Nesta subsecção irão ser abordados esses pontos, bem como alguns passos de implementação da *Web App* de exportação.

6.4.1 Representação e armazenamento dos dados das recolhas

Os dados recolhidos pela aplicação *VR-Unit* são armazenados em ficheiro à medida que as recolhas são efetuadas. O formato utilizado na escrita destes ficheiros é o formato JSON e CSV, que permite

programaticamente uma leitura fácil por parte de outro *software*, caso seja necessário importar os dados para algum sistema de visualização, ou no caso de *Matlab* em que é mais fácil processar os dados em CSV. Um exemplo de uma parte de um ficheiro gerado é visível na Figura 6-29, onde podemos ver parte de um ficheiro resultante de dados de GPS, cada linha do ficheiro corresponde a uma amostra recolhida.


```
{"alt":"---","date":"13:17:30","deviceID":"bb2e1bc2c4e7e707","lat":"40.63","lon":"-8.66","provider":"network","timestamp":"1522066650195"}
{"alt":"---","date":"13:17:50","deviceID":"bb2e1bc2c4e7e707","lat":"40.63","lon":"-8.66","provider":"network","timestamp":"1522066670133"}
{"alt":"---","date":"13:18:10","deviceID":"bb2e1bc2c4e7e707","lat":"40.63","lon":"-8.66","provider":"network","timestamp":"1522066690254"}
```


Figura 6-29 - Trecho do ficheiro de recolha de GPS.

Todos os dados recolhidos, ou dados referentes a eventos, alarmes e informação, são escritos nos ficheiros respetivos por um Serviço Android, que corre em *background*, sendo um elemento comum ao *VR-Unit* e ao *VR-Remote*. Os dados produzidos são armazenados hierarquicamente consoante o nível de estudo, grupo de aquisição e participante. O diretório de maior nível é o diretório do estudo/, que será nomeado com o nome e o *token* do estudo. Dentro desse diretório estarão vários diretórios identificativos de todas as sessões de recolha desse estudo. Dentro do diretório de cada estudo, existirá o ficheiro de dados de cada sensor, produzido durante aquela sessão, juntamente com ficheiros de áudio produzidos também na mesma sessão. Uma árvore de diretórios desta organização encontra-se representada na figura seguinte.

Figura 6-30 - Organização de ficheiros local, na aplicação VR-Unit.

Na *cloud*, os dados serão organizados na mesma por projeto/estudo, correspondendo esse ao diretório de maior nível, seguido do diretório de todas as sessões desse estudo. Dentro de cada diretório de sessões, existirão ficheiros *.zip* correspondentes aos dados recolhidos por cada participante e também pelo *VR-Remote*. De modo a tornar o *upload* dos dados do telemóvel para a *cloud* o mais rápido possível, os ficheiros dentro de cada telemóvel são comprimidos, de modo a cada telemóvel carregar os ficheiros de cada sessão todos de uma só vez. Esse *upload* é realizado por um Serviço, em *background*, que coloca a operação de *upload* numa *thread* em separado de modo a não bloquear a *Main thread* da aplicação.

6.4.2 API de integração

De modo a poder manter sincronizada a *WebApp* de exportação com o decorrer das aquisições, foi criada uma *API RESTful*[22] que permitisse sincronizar a componente *web* com a componente *mobile*. Na construção da API, foi utilizado *node.Js*, uma *framework* que permite criar servidores web multiplataforma. Para construir o *backend* foi utilizada uma outra *framework*, presente no *npm* (gestor de pacotes do *Node.Js)*, denominada de *Express.Js*, a *API* está diretamente assente nesta camada de *backend*. Como base de dados de suporte a esta API foi utilizado *MongoDB*, uma base de dados não relacional, simples de utilizar na construção de *API's* deste género. A *API* atualiza a base de dados, permitindo inserir os grupos e estudos configurados, apagá-los, ou fazer *queries* sobre esses mesmos dados. As operações possíveis de realizar na API estão descritas na tabela seguinte.

Método HTTP	URL	Operação efetuada
GET	/projects	Listar todos os estudos atuais.
POST	/projects/projectName_token_group_username	Adicionar um estudo.
DELETE	/projects/projectName_token_group_ username	Apagar um estudo.
GET	/projects/projectName_token_group_ username	Devolver informações sobre um estudo.

Tabela 6-3 - Operações suportadas pela API.

O esquema de base de dados utilizado assenta em apenas uma entidade, onde é guardada a informação do nome do estudo, token do estudo e nome do grupo que pertence a esse estudo. De modo a ir buscar informações à base de dados, na *API* desenvolvida existe um *Controller* que contém os métodos que vão tratar das operações de leitura, escrita e remoção na base de dados.

6.4.3 Web app de exportação

De modo a permitir aos intervenientes nas aquisições exportar os dados de forma facilitada, sem necessidade de aceder à Dropbox, que é onde os dados estão localizados após serem enviados dos telemóveis, foi criada uma *Web App* simples que permite transferir os ficheiros resultantes das sessões de recolha. Esta *WebApp* é cliente da *RESTful API* desenvolvida, e também corre sobre o mesmo servidor de *Node.js*

A *WebApp* contém um acesso controlado por *login*, nessa página, os responsáveis pela psicologia devem-se inscrever previamente a realizar o acesso. O *username* que definirem deverá de ser o mesmo que colocam no "Nome do Responsável" na altura da configuração da sessão de recolha, de modo a ficar associado à sua conta na *WebApp* e poderem ser listados os seus estudos.

[22] https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest

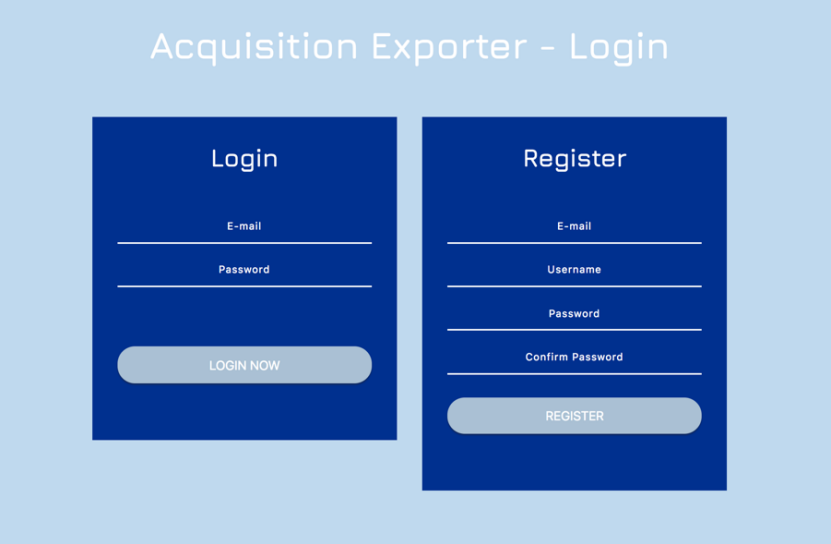


Figura 6-31 - Página de Login da WebApp.

Após o *login*, é apresentado ao utilizador uma página *Web*, onde é possível verificar a lista de estudos/grupos por ele já configurados. Os dados são devolvidos pela API, e consoante essa lista, o utilizador poderá fazer o download dos dados de cada grupo, individualmente, ao carregar no botão de *"download"*. Esta *WebApp*, utiliza a *API* de *Javascript* da *Dropbox* de modo a descarregar os ficheiros das aquisições e os disponibilizar ao utilizador sobre a forma de ficheiro *.zip*. A interface visual da página *web* está representada na Figura 6-32.

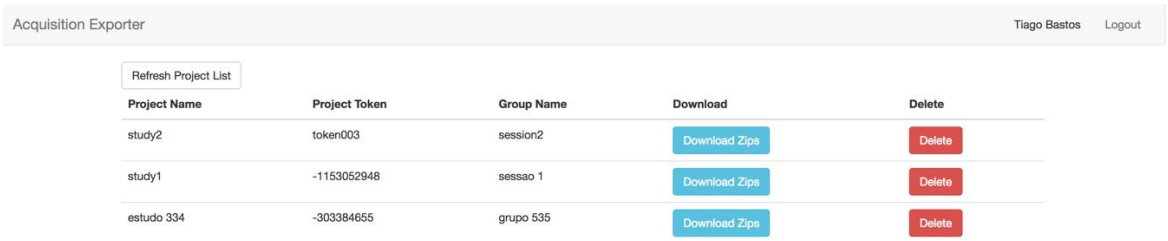


Figura 6-32 - Web App de exportação.

6.5 Discussão de outras estratégias de implementação

Nesta subsecção serão abordados algumas das opções de desenvolvimento que não estão diretamente ligadas à interação do utilizador, mas que, são importantes de referir, devido à sua importância na implementação.

6.5.1 Módulo independente de controlo de sensores

Um dos objetivos iniciais deste projeto era o de tornar o módulo de gestão e ligação a sensores, o mais genérico possível, de modo a poder ser estendido e utilizado em outras aplicações. De modo a criar este módulo, optou-se por desenvolver uma biblioteca para aplicações *Android* que incluísse todo o código necessário para criar conexões, gerir, e obter dados dos sensores. O nome dado a este componente foi de *AcquisitionSensorManager*, tem como principais funcionalidades:

- conectar a sensores;
- receber e processar dados transformando-os num formato programaticamente interpretável;
- realizar processamento de alarmes.

A biblioteca atualmente suporta vários tipos de sensores. Na Tabela 6-4 é possível ver todos os sensores suportados por esta biblioteca, juntamente com a informação da conexão da biblioteca ao sensor.

Sensor	Tipo de Sensor	Ligação	Dados fornecidos
Vital Jacket	Sinais Vitais	API fornecida pelo fabricante (*Bluetooth),*	Sinal ECG, Batimento cardíaco, temperatura corporal, acelerómetro.
GRAYS-2	GPS. Externo	*Sockets* de *Bluetooth*.	*Timestamp*, Latitude, Longitude, Altitude.
FREMU	Ambiental	*Sockets de Bluetooth*.	CO2, Temperatura, Altitude, Pressão atmosférica.
HELMET	Ambiental	*Sockets* de *Bluetooth.*	N2O, CO2, Pressão atm., altitude, humidade, luminosidade, temperatura.
GPS	GPS (Interno)	*Location API* do *Android*.	Latitude, Longitude, distância percorrida.
Áudio	Áudio	*MediaRecorder API* do *Android*.	Áudio.
BITalino	BITalino	*API* fornecida pelo fabricante.	Sinal ECG, EMG, EDA, EEG.
BITalino Low Energy	BITalino	*API* fornecida pelo fabricante.	Batimento cardíaco, sinal ECG, EMG, EDA, EEG.
Xiaomi Mi Band 2	Band	*Sockets* de Bluetooth *Low Energy*.	Batimento cardíaco
Weather Station	Meteorológico	*Sockets* de *Bluetooth*.	Velocidade do vento, direção do vendo, pressão atm., temperatura, luminosidade, humidade.

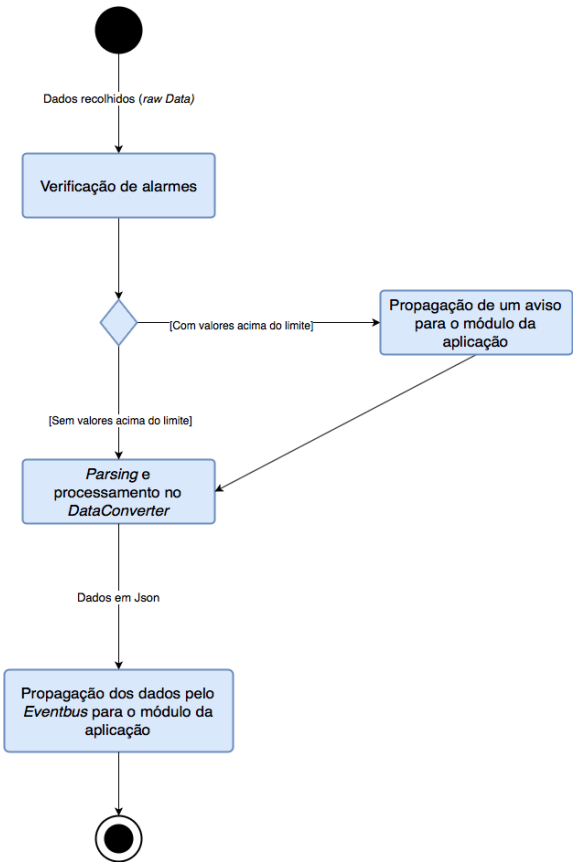
Tabela 6-4 - Tabela de sensores suportados pela biblioteca.

Uma versão esquemática da arquitetura está presente Figura 6-33 b), o fluxo base na Figura 6-33 a), podendo ser descrita desta maneira:

- existem internamente vários serviços que tratam da ligação aos sensores e também da captação dos dados, associado a cada serviço existe um componente denominado de *DataConverter* que recebe os dados recolhidos pelos sensores e os transforma nos tipos de dados definidos nesta biblioteca;

- Os alarmes são despoletados quando são ultrapassados os valores limite configurados pela aplicação, esses valores limite são passados para o Serviço através do *Intent* que o inicia. Para ser despoletado um alarme é necessário que os valores enviados pelo sensor ultrapassem o limite definido 5 vezes num espaço de 1 minuto. Quando esse limite é ultrapassado, a informação sobre o alarme é enviada para a aplicação através do *EventBus*. É possível alterar a forma com que o alarme é acionado, sendo que, o tempo e o número de vezes que o valor tem de ultrapassar o limite configurado é dinâmico.
- após essa passagem pelo *DataConverter*, os dados *raw* transformam-se em dados processados, num processo onde é feito o *parsing* da *string raw* enviada pelo sensor, de seguida é convertido num objeto do tipo de dados associado àquele tipo de sensor, e esse objeto por último é convertido em formato *Json*
- Após esse processo de *parsing* e conversão, os dados serão publicados pelo *EventBus* numa mensagem apropriada a cada tipo de sensor, podendo ser subscrito pelas aplicações que utilizem esta biblioteca e assim receber os dados de modo a fazerem a sua visualização.

a)



b)

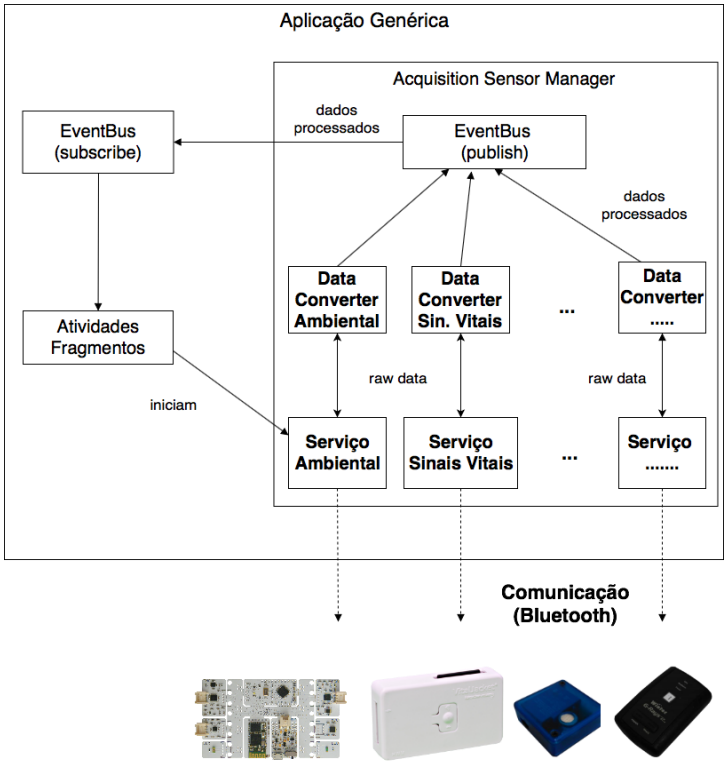


Figura 6-33 - Arquitetura e fluxo interno da biblioteca Sensor Manager.

Os serviços da biblioteca são inicializados diretamente pelas aplicações externas, onde através do *Intent* é possível colocar argumentos, esses argumentos servirão como configuração de alguns parâmetros necessários durante a fase de ligação, ou processamento de dados, ou processamento de alarmes. Todos os serviços da biblioteca estendem um serviço base, o *AcquisitonService*, que contém métodos que obrigatoriamente têm de implementar:

- *getSensorType():* devolve o tipo de sensor, que corresponde a um elemento de um enumerado criado na biblioteca, que distingue tipos de sensores;
- *start()*: responsável por iniciar as recolhas de dados e fazer o envio dos dados recebidos para o *DataConverter* associado;
- *connect()*: responsável por estabelecer a ligação ao sensor;
- *sendState()* que é responsável por publicar os eventos que irão conter os dados já processados no *EventBus*.

Esses serviços, podem ainda implementar uma interface, *IAlarmChecker*, que funciona como "contrato" sobre os serviços que fazem o processamento de alarmes. No diagrama de classes seguinte, estão presentes todas as entidades principais presentes nesta biblioteca, juntamente com dois serviços exemplo de recolha de dados a sensores.

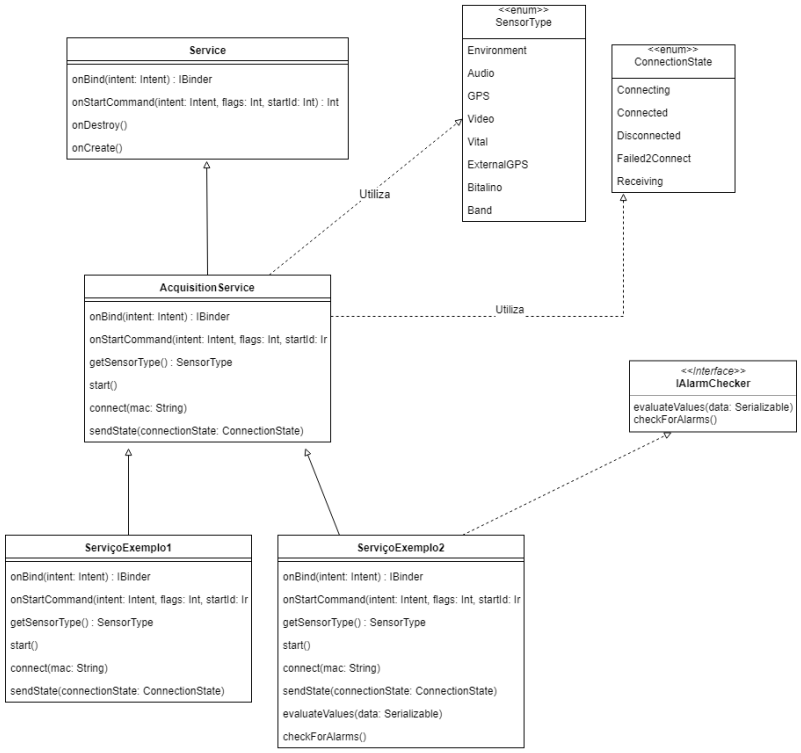


Figura 6-34 - Diagrama de classes da biblioteca.

6.5.2 Segurança e proteção dos dados

Proteção do *MessageCollector*

Neste tipo de recolhas existe a leitura e agregação de dados fisiológicos de participantes reais, logo a proteção dos dados é um componente muito importante no desenvolvimento deste tipo de sistemas. A segurança no armazenamento, comunicação entre nós, e exportação é um ponto crítico que não pode ser ignorado.

Na proteção do canal de comunicação, foram implementados 2 mecanismos de segurança. O primeiro, e o mais natural, é a configuração do *broker* com autenticação por *username* e *password*, de modo a apenas clientes com credenciação terem acesso à troca de mensagens, evitando acessos intrusivos e tentativa de angariação maldosa de dados. Um outro mecanismo que pode ser visto como forma de segurança, é a geração de um *token* aleatório no início de cada recolha, que apenas permite a dispositivos participantes da recolha, subscreverem a tópicos, impedindo o acesso aos tópicos de acessos indesejados.

Proteção no armazenamento

O armazenamento dos dados na *cloud*, está sujeito a segurança que o serviço de armazenamento implementa, neste caso a *Dropbox*. O acesso aos dados de eventuais intrusos sem as credenciais de autenticação da conta utilizada é impedido, bem como o acesso programático aos mesmos sem a utilização do *app secret* associado à conta.

Em relação aos dados em si, pode-se dizer que não são armazenados identificadores pessoais relativos aos participantes. É da responsabilidade dos investigadores e dos utilizadores não atribuírem como nome amigável do participante uma chave que o identifique pessoalmente, sendo essa a única maneira possível de ligar os dados recolhidos a uma pessoa, visto que o nome amigável atribuído identifica os ficheiros relativos aos participantes.

Proteção no acesso aos dados

Outro método de segurança implementado foi o *login* e o acesso controlado à *WebApp* de exportação. De modo a utilizar boas práticas no armazenamento dos dados de acesso dos utilizadores registados, foi utilizada a biblioteca *bcrypt*[23] presente no gestor de pacotes do *Node.js*. Esta, tinha a função de gerar o *hash* das *passwords* dos utilizadores,.

O protocolo *HTTP* é um protocolo *stateless*, o que quer dizer que não é feito uma monitorização de quem visita uma página. Desse modo, foram criados mecanismos de sessões com um ID que identifica cada sessão, funcionando como um *Cookie*. De modo a implementar este mecanismo, foi utilizado o *package Express Session*[24], que permite a criação e gestão de sessões. O ID de sessão criado, é então gravado na base de dados através da configuração atribuída.

6.5.3 Bibliotecas externas integradas

Na realização das aplicações VR-Unit e VR-Remote foram utilizadas várias bibliotecas externas no suporte ao desenvolvimento, a lista das tabelas utilizadas encontra-se na Tabela 6-5.

[23] https://www.npmjs.com/package/bcrypt
[24] https://www.npmjs.com/package/express-session

Nome da biblioteca	Descrição	Utilização na aplicação
EventBus	Biblioteca que contém a API do *EventBus*, permitindo a subscrição, configuração e envio de mensagens através desta ferramenta.	Comunicação interna entre componentes nas aplicações.
Toasty	Biblioteca de *Toasts* com aspeto visual diferente.	Feedback visual ao utilizador.
Alerter	Biblioteca com um elemento visual sob forma de Alerta.	Feedback visual ao utilizador.
Google Gson	Biblioteca de manipulação de objetos *JSON*.	Tradução de *POJOs* para objetos *JSON* e vice-versa.
Number Picker	Biblioteca que contém um *Picker* costumizável.	Utilizado para selecionar o número de eventos.
Android View Animations	API de criação animações em elementos visuais.	Feedback visual ao utilizador.
QR Gen	Biblioteca que permite a criação de códigos QR ou códigos de barras.	Criação de códigos QR.
Dropbox SDK	SDK de interação com a *Dropbox*.	Interação com a Dropbox.
Fabric Crashlytics	Biblioteca do *Fabric*, permite fazer monitorização.	Monitorização da aplicação em dispositivos; distribuição do *apk.*
Firebase API	*API* de autenticação e utilização do serviço *Crashlytics do Firebase.*	Envio de dados de *crash* para a plataforma *crashlytics;* Testes.
Eclipse Paho	API de alto nível para utilização do *broker* Mosquitto em *Android*.	Interação com o *broker*.
Clans Floating Action Button	Biblioteca que fornece uma versão estendida do *FloatingActionButton.*	Utilização gráfica.
MPAndroidChart	Biblioteca que permite o desenho de gráficos de vários tipos.	Desenho de gráficos.
Retrofit	Biblioteca que permite realizar pedidos HTTP.	Comunicação com a RESTful API desenvolvida.
Google Guava	Biblioteca com implementações de diversas estruturas de dados.	Utilização da estrutura de dados MultiMap.
Google Maps Android API	API do Google Maps	Renderização do mapa com a posição atual do utilizador.
Affectiva SDK	SDK que permite o reconhecimento de emoções.	Reconhecimento de emoções na expressão facial.
AndroidPlot	Biblioteca que permite o desenho de gráficos.	Desenho de gráficos em tempo real.
RabbitMQ API	Biblioteca de interação com o *broker RabbitMQ*.	Envio de mensagens pelo broker.

Tabela 6-5 - Quadro de bibliotecas utilizadas.

6.6 *Extensão do sistema*

Um dos pontos fortes de qualquer componente de *Software* deve ser a sua extensibilidade. Um dos objetivos deste projeto era o de fornecer aos programadores um mecanismo fácil de extensão a novos sensores, mas também a extensão a outros sistemas utilizados na psicologia, ou outras áreas. Nas seguintes subsecções serão abordados alguns dos mecanismos que tornam o sistema extensível.

6.6.1 Processo de integração de um novo sensor na biblioteca

Aos sensores já existentes na biblioteca desenvolvida é possível adicionar outros, que recolham o mesmo tipo de variáveis fisiológicas, ou de outro género, ou que recolham novos tipos de variáveis, para cada uma dessas situações a maneira de integração é diferente. Visto de uma maneira geral, serão sempre necessários 3 a 5 passos, sendo que um deles é opcional, descritos na tabela seguinte.

Passo	Tipo de sensor existente	Novo tipo de sensor
1	Alterar o *DataConverter* existente	Acrescentar um novo elemento ao enumerado *SensorType*
2	Alteração do serviço existente: criar código de comunicação com o sensor (*Bluetooth, Zigbee, outro...)*	Criação de uma nova classe de dados para o tipo de dados do sensor, e também para o *Eventbus* interno
3	**Opcional**: Implementar a interface *IAlarmChecker*, e produzir código de verificação de alarmes sobre valores recebidos.	Criação de um *DataConverter*
4		Criação de um serviço para o novo tipo de sensor, com código de comunicação com o sensor
5		**Opcional**: Implementar a interface *IAlarmChecker*, e produzir código de verificação de alarmes sobre valores recebidos.

Tabela 6-6 - Quadro comparativo entre as alterações a efetuar na biblioteca.

No passo 1 e 2, no novo tipo, estão representados os 2 passos iniciais, que criam suporte em termos de modelo de dados ao novo sensor, será então necessário acrescentar o novo tipo de dados ao enumerado *SensorType (*Figura 6-35 *a))*, e criar um modelo de dados adequado à variáveis que o sensor recolhe (*(*Figura 6-35 *b))*, bem como uma nova mensagem no *Eventbus* para esse tipo de sensor *(*Figura 6-35 *c) e d))*.

a)
```
enum class SensorType : Serializable {
    ENVIRONMENT, AUDIO, GPS, VIDEO,
    VITAL, NONE, EXTERNALGPS, BITALINO,
    BAND, NEW_TYPE
}
```

b)
```
data class NewSensorDataModel(private val varXXX: String,
                              private val varYYY: Int,
                              private val varZZZ: Float)
```

c)
```
class NewSensorValuesEvent(var sensorMac: String, var values: String)
```

d)
```
class NewSensorStateEvent(var sensorMac: String, var connectionState: ConnectionState)
```

Figura 6-35 - Passo 1 e 2.

O passo 3, quase comum ao passo 1 no caso de existir o tipo de sensor, corresponde à criação de um novo *DataConverter (*Figura 6-36 a)) ou alteração do existente *(*Figura 6-36 b)), de modo a criar maneira de fazer o *parse* dos dados e respetiva transformação para a classe de dados do sensor.

a)

```
object NewDataConverter {

    private val gson: Gson by lazy {
        Gson()
    }


    fun newDataConverter(location: Location, deviceID: String) : String {
        val df = DecimalFormat( pattern: "#.00")
        val date = Date(location.time)
        var alt = "---"
        if (location.altitude != 0.0)
            alt = df.format(location.altitude).toString()
        val timestamp = Timestamp(System.currentTimeMillis()).time.toString()
        val gpsDataString = LocationData(timestamp = timestamp, date = date.toPortugueseDateWithoutDMY(),
                lat = df.format(location.latitude).toString().replace( oldValue: ",", newValue: "."),
                lon = df.format(location.longitude).replace( oldValue: ",", newValue: "."),
                alt = alt.replace( oldValue: ",", newValue: "."),
                provider = location.provider, deviceID = deviceID)
        return gson.toJson(gpsDataString, LocationData::class.java)
    }

}
```

b)

```
fun newDataConverter(location: Location, deviceID: String) : String {
    val df = DecimalFormat( pattern: "#.00")
    val date = Date(location.time)
    var alt = "---"
    if (location.altitude != 0.0)
        alt = df.format(location.altitude).toString()
    val timestamp = Timestamp(System.currentTimeMillis()).time.toString()
    val gpsDataString = LocationData(timestamp = timestamp, date = date.toPortugueseDateWithoutDMY(),
            lat = df.format(location.latitude).toString().replace( oldValue: ",", newValue: "."),
            lon = df.format(location.longitude).replace( oldValue: ",", newValue: "."),
            alt = alt.replace( oldValue: ",", newValue: "."),
            provider = location.provider, deviceID = deviceID)
    return gson.toJson(gpsDataString, LocationData::class.java)
}
```

Figura 6-36 - Passo 3.

No passo 4, equivalente ao passo 2 no caso de ser existente, corresponde à criação do serviço que irá estender o serviço base *AcquisitionService*, ou criação de código dentro de serviço existente de modo a tornar possível a comunicação com o sensor. Englobará a utilização de uma *API* ou *SDK* fornecido pelo sensor, ou criação de *sockets* de *Bluetooth* ou outro género, de modo a comunicar, e receber dados. Uma implementação exemplo de um serviço deste género é visível na Figura 6-37.

```
class NewSensorService : AcquisitionService() {
    private var state = ConnectionState.DISCONNECTED
    private lateinit var macAddr: String


    override fun onStartCommand(intent: Intent, flags: Int, startId: Int): Int {

        // receber a configuração por intent

        return START_STICKY
    }

    override fun getSensorType(): SensorType = SensorType.NEW_TYPE

    override fun start() {
        // recolha de dados
    }

    override fun connect(mac: String) {
        // ligação ao sensor
    }

    override fun sendState(connectionState: ConnectionState) {
        this.state = connectionState
        EventBus.getDefault().postSticky(NewStateEvent(macAddr, connectionState))
    }

}
```

Figura 6-37 - Criação de um novo serviço de aquisição.

Opcionalmente, temos o último passo em qualquer uma das abordagens, onde é possível implementar a interface que acrescentar o método de avaliação de valores ao sensor, *IAlarmChecker*, que poderá ser implementado de maneira livre pelo programador.

6.6.2 Cenários de integração com módulos externos

Com a utilização do *Broker Mosquitto*, a integração de módulos externos associados a recolhas de psicologia, ou de outro tipo de mecanismos de controlo e agregação de dados é possível. Sistemas externos poderão subscrever a alguns tópicos específicos de modo a sincronizarem-se com as recolhas, podendo então realizar ações na receção de mensagens desses respetivos tópicos.

Um exemplo da facilidade da integração desta solução com módulos externos, com a utilização do *MessageCollector*, foi a integração de um sistema de gravação de áudio desenvolvido por um investigador do IEETA, utilizado em recolhas de psicologia no Departamento de Psicologia. Esse sistema, subscreve ao tópico de envio de eventos, e também ao tópico que sinaliza o início ou o fim de uma sessão de recolha, que são enviados pelo *VR-Remote*. Qualquer módulo externo, poderá ser desenvolvido sobre qualquer linguagem de programação, no caso abordado, o cliente em questão está desenvolvido em C#.

Também foi desenvolvido outro cliente do *MessageCollector* em *Python*, presente num *Raspberry PI*, que processa os eventos e comandos de início e fim de recolha da aplicação anterior, de modo a integrar as aplicações antigas neste processo de comunicação com o sistema de áudio.

Visualização em sistemas externos

Os dados recolhidos pela aplicação *VR-Unit* podem ser observados em tempo real no novo sistema de visualização desenvolvido no âmbito do projeto *VR2Market*. Esse novo visualizador, recebe os dados através de um *Broker RabbitMQ*, sendo que esses dados são transportados com o mesmo formato com o qual são guardados, em *JSON*, de acordo com o modelo de dados.

Nesse sistema é possível observar os dados dos sensores ambientais (FREMU e Helmet), *Weather Station*, *Vital Jacket* e GPS Interno, fornecendo acesso por exemplo a um comandante dos bombeiros a visualizar os dados dos seus operacionais no terreno, permitindo-o tomar decisões e orquestrar as movimentações no terreno. Uma utilização de exemplo da visualização dados recolhidos por uma aplicação agregadora é visível na Figura 6-38.

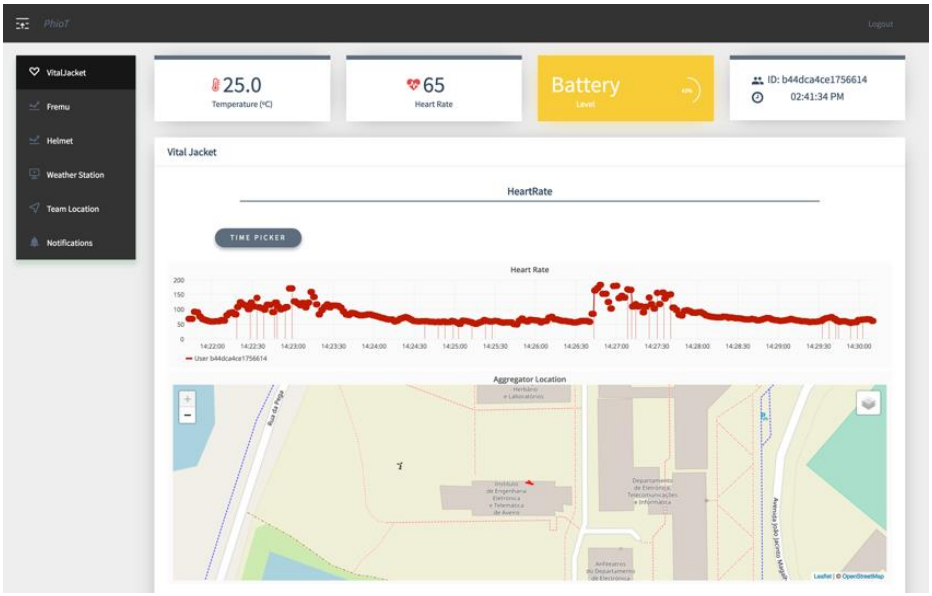


Figura 6-38 - Exemplo da utilização do sistema de visualização, com dados enviados através o VR-Unit.

6.7 Aspetos de garantia e qualidade do software

Durante e após o desenvolvimento das aplicações móveis, devem ser feitos testes de modo a verificar o bom funcionamento dos seus componentes quer individualmente, quer em grupo, de modo a garantir que a aplicação tem o menor número possível de *bugs* em *runtime*.

Neste subcapítulo serão explicadas todas as abordagens e mecanismos de teste utilizados durante e após o desenvolvimento da aplicação.

6.7.1 Análise estática de código

Após a implementação de cada funcionalidade, durante o desenvolvimento, foi utilizado o mecanismo de análise estática de código fornecido pelo *Android Studio*, o *Lint*. [60] Nas definições do *Android Studio* foram ativadas todas as *flags* deste mecanismo de modo à inspeção efetuada fosse o mais extrema e a fundo possível para evitar problemas que surgissem posteriormente em execução.

Esta análise estática permite identificar e corrigir problemas de vários tipos:

- Estrutura e organização de código;
- Potenciais *bugs* através da identificação de código que possa executar de maneira errada;
- Verificação de dependências nos ficheiros *gradle;*
- Problemas nos ficheiros xml (*layout*, recursos linguísticos, recursos de imagens);
- Problemas de internacionalização (por exemplo layouts não preparados para ecrãs árabes);
- Problema em outro tipo de recursos do projeto;
- Problemas no ficheiro de *manifest*;
- Problemas de usabilidade;
- Verificação de ciclos e condições que possam ser executadas de maneira mais simplificada;

Com este mecanismo, é possível analisar a aplicação na totalidade à procura de erros sem executar a aplicação, visto que isto é um programa que é lançado a partir do desenvolvimento de programação sem recurso a qualquer emulador.

6.7.2 Testes

Testes de Memory Leaks

No desenvolvimento de aplicações *Android* é frequente o contexto da aplicação ser utilizado em vários componentes, ou ser necessário em *AsyncTask*, *Adapters*, *Services*, etc.., no entanto uma má gestão do contexto da aplicação poderá dar azo aos chamados *memory leaks*, que poderão levar a aplicação a ser "morta" pelo sistema. Um *memory leak* tecnicamente corresponde a quando um objeto alocado em memória "vive" mais tempo do que o esperado dentro de uma *Activity*, sendo que o *garbage collector* do *Android* não o consegue destruir. [61]

Durante a fase de testes da versão piloto, foi utilizada nas duas aplicações desenvolvidas a biblioteca *leak Canary* que permite ao programador ser notificado de cada vez que acontece um *leak* do contexto da aplicação, identificando a zona de código em que ocorreu, e o tipo de contexto que gerou o problema. Este tipo de funcionalidade é de extrema importância na fase de testes de uma aplicação móvel, visto que no *Android* referências para o contexto são bastante difíceis de gerir e não há garantias que o código produzido não trará problemas de *context leaking* que só poderão ser detetados com uma ferramenta do género.

Na realização deste tipo de testes, que também coincidiu com a realização dos testes de usabilidade, este mecanismo esteve ativo na procura de problemas do género. Não foram detetados problemas de *memory leak* nas duas aplicações desenvolvidas.

Testes de Stress às aplicações

Foram efetuados testes de stress às aplicações através da ferramenta *Monkey* [62] disponibilizada pela plataforma, esta ferramenta sobrecarrega a aplicação com cliques, toques, gestos, ou rotações de ecrã, bem como outras operações de sistema, dando ao programador erros que dificilmente aconteceriam com uma utilização humana, que possivelmente terá poucas utilizações.

Ambas as aplicações foram testadas com 500, 1000 e 5000 eventos aleatórios, sendo que nenhuma delas apresentou problemas no final do desenvolvimento, no entanto, a utilização desta ferramenta de teste permitiu durante o desenvolvimento descobrir alguns erros que com utilização humana normal seriam difíceis de detetar, como por exemplo rotações de ecrã em situações mais críticas como em cenários de recolha, e que seriam difíceis de detetar com a utilização normal humana.

Firebase Test Lab

Através do *Firebase Test Lab* foram efetuados vários *Robo Test* às aplicações, estes testes ao contrário dos testes de *stress* feitos pelo *Monkey*, testam a estrutura da interface visual da aplicação, explorando-a com atividades normais de utilizadores em diferentes dispositivos de diferentes tamanhos, também com diferentes orientações consoante a configuração do utilizador.

Como utilizador gratuito, é possível criar 10 testes deste género por dia, as aplicações eram testadas regularmente neste mecanismo. É possível configurar alguns parâmetros dos testes, como por exemplo o dispositivo onde irá correr, se físico ou virtual, dando ao utilizador escolha de vários modelos de telemóveis com diferentes níveis de API. Nos testes efetuados foram sempre escolhidos 2 dispositivos físicos, um Google Pixel 2 e um Nexus 5, o primeiro por ser um modelo topo de gama atual, e o segundo porque foi utilizado em testes físicos. Os testes efetuados têm uma duração aproximada de dois minutos, no final de cada teste é enviado ao utilizador um e-mail que diz se a aplicação passou ou não passou nos testes, e na plataforma *Firebase* é possível aceder ao vídeo do teste, alguns *screenshots*, o mapa de atividade de utilização, eventuais erros, e alguns gráficos de performance da aplicação.

O *Robo Test*, analisa a estrutura da interface visual da aplicação, onde automaticamente realiza atividades normais de utilizadores. Ao contrário do *Monkey Test*, o *Robo Test* simula sempre as atividades do utilizador na mesma ordem com que um utilizador faria num dispositivo físico [63].

Na Figura 6-39 é possível ver um gráfico da performance de alguns dos componentes do telemóvel, durante um dos *Robo Test* efetuados. Na Figura 6-40 é possível ver parcialmente o diagrama da atividade da aplicação durante a execução desse teste.

Figura 6-39 - Análise da performance da aplicação durante o Robo Test.

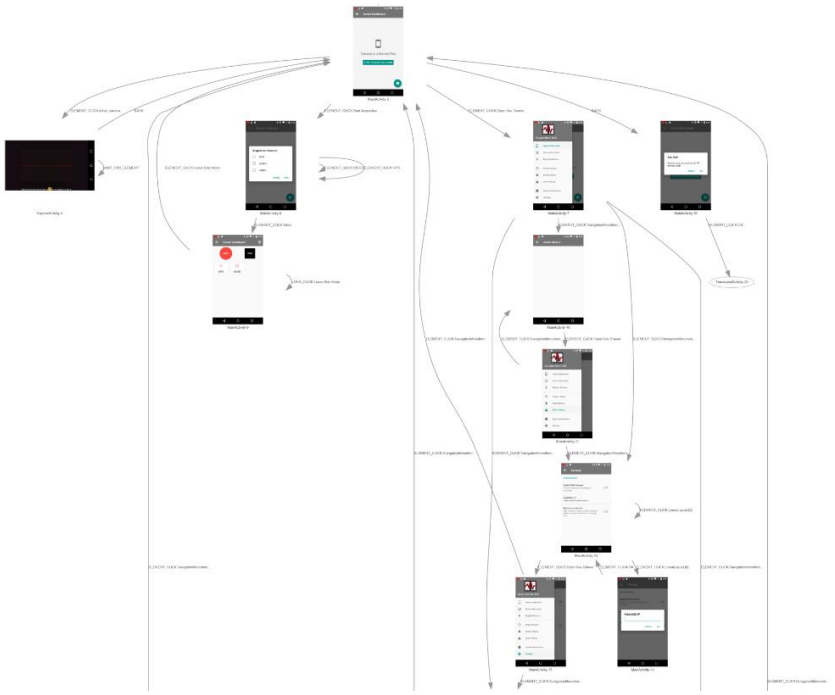


Figura 6-40 - Excerto do diagrama de atividade/interações) gerado pelo Robo Test.

Testes à base de dados local

Quando a base de dados interna foi modelada e estruturada, foram realizados vários testes de inserção e remoção e verificação de alteração desses mesmos dados, antes da utilização da base de dados como suporte à aplicação móvel. Estes conjuntos de testes tinham como objetivo principal testar as *constraints* de chaves entre as tabelas, antes dessas ações serem feitas na aplicação em *runtime*.

Para realizar estes testes foi utilizado *AndroidJUnit* e a aplicação não tinha de correr no dispositivo para os testes decorrerem, apenas precisava da utilização do *adb* para os lançar, sem necessidade de abrir a aplicação efetivamente.

Nas figuras seguintes, é possível ver um dos testes realizados, neste teste foi possível verificar que a *constraints* ON_DELETE_CASCADE funcionava quando era apagada o objeto Projeto de nome "ProjetoTest" que continha o Grupo "Grupo1", mas também, a inserção e remoções de dados na entidade Projeto e Grupo.

```
@RunWith(AndroidJUnit4::class)
class AcquisitionDBTest {
    private var mProjectDao: ProjectDao? = null
    private var mGroupDao: GroupDao? = null
    private var mDb: AppDatabase? = null

    @Before
    fun createDb() {
        val context = InstrumentationRegistry.getTargetContext()
        mDb = Room.inMemoryDatabaseBuilder(context, AppDatabase::class.java).build()
        mProjectDao = mDb!!.projectDao()
        mGroupDao = mDb!!.groupDao()
    }

    @After
    @Throws(IOException::class)
    fun closeDb() {
        mDb!!.close()
    }

    @Test
    @Throws(Exception::class)
    fun writeProjectAndTest() {
        val project = ProjectEntity("ProjetoTest", ProjectType.PSYCHOlOGY)
        val project2 = ProjectEntity("ProjetoTest2", ProjectType.PSYCHOlOGY)
        val group = GroupEntity("Grupo1", "ProjetoTest", "Person1")
        val group2 = GroupEntity("Grupo1", "ProjetoTest2", "Person2")
        mProjectDao!!.insertProject(project)
        mProjectDao!!.insertProject(project2)
        mGroupDao!!.insertGroup(group)
        mGroupDao!!.insertGroup(group2)

        val projectList = mProjectDao!!.getAllProjects()
        val projlGroupList = mGroupDao!!.getAllGroupsOfProject("ProjetoTest")

        assertThat(projectList.size, equalTo(2))
        assertThat(projlGroupList.size, equalTo(1))

        mProjectDao!!.deleteAllProjects()
        val projectList2 = mProjectDao!!.getAllProjects()
        val groupList = mGroupDao!!.getAllGroups()

        assertThat(projectList2.size, equalTo(0))
        assertThat(groupList.size, equalTo(0))
    }
}
```

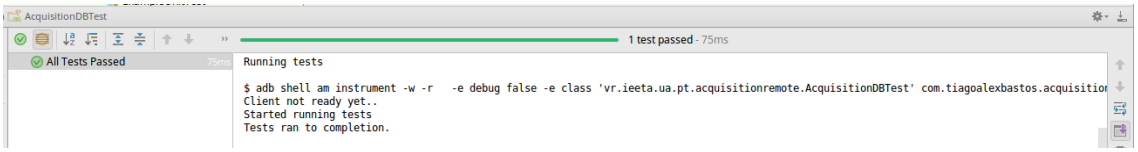


Figura 6-41 - Exemplo de um dos testes realizados e seus resultados.

6.7.3 Ofuscação de código e redução de recursos

O código de ambas as aplicações desenvolvidas encontra-se ofuscado e reduzido através da utilização dos mecanismos de *Proguard* e *code shrinking* do *Android*. Este processo permite ao *apk* final apresentar um tamanho menor do que o suposto, visto que os mecanismos de *code shrinking* retiram do código da aplicação

todos os métodos, variáveis e classes não utilizadas. Para além disso permite acrescentar ao *apk* final uma camada de segurança adicional visto que todo o código fica difícil de ser *reverse engineered*, o que salvaguarda algum código sensível como *passwords*, outras credenciais ou *API keys.* [64]

Este tipo de mecanismo deve ser utilizado em aplicações cujo tamanho do *apk* se torne bastante grande, ou caso seja necessário esconder qualquer tipo de credenciais, a sua utilização em projetos de dimensão reduzida torna-se demasiado excessiva. Como no futuro ambas estas aplicações podem ser lançadas na *PlayStore*, faz sentido o *apk* que será lançado ser o mais curto possível, e também estar com o código completamente ofuscado de modo a prevenir o plágio.

Na Figura 6-42 é possível ver na parte superior, a redução de tamanho do *apk* quando aplicado o mecanismo de *ProGuard*. Na parte inferior da figura, podemos visualizar a ativação deste mecanismo, em modo de *release*.

Figura 6-42 - Canto superior esquerdo, tamanho do apk final da aplicação Remote antes do mecanismo de proguard, e no canto superior direito o depois. Em baixo, a ativação do módulo em modo de release.

6.7.4 Distribuição das aplicações e mecanismos de deteção de erros

De modo a distribuir a aplicação durante a fase de desenvolvimento e teste do protótipo da aplicação aos diversos dispositivos foi utilizado o serviço *Beta* do *Fabric.* Este serviço permite a distribuição de forma gratuita de aplicações *Android* através da partilha do *apk* por uma *mailing list* de testers. Após a receção do *email* que continha o *apk*, cada *tester* apenas tem de abrir a aplicação *Beta* do *Fabric* e poderá instalar e lançar a aplicação normalmente.

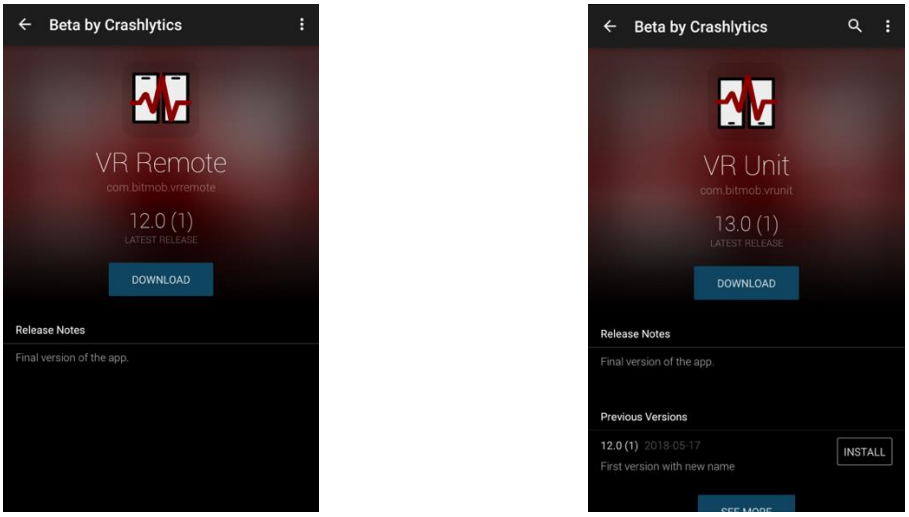


Figura 6-43- Aplicações disponíveis na Beta Store do crashlytics.

Durante o desenvolvimento das aplicações, de modo a de cada vez que ocorresse um problema e a aplicação não estivesse ligada ao computador fosse possível detetar o erro, foram utilizados os mecanismos de deteção de problemas fornecidos pelo *Firebase* e pelo *Fabric*. Estes fornecem ao programador ferramentas que permitem fazer a monitorização de eventuais problemas na execução da aplicação, fornecendo ao programador o *stacktrace* do erro, bem como algumas informações de *hardware* do dispositivo que também possam ser importantes na deteção do erro.

Para além desta visualização e notificação do erro em tempo real é também possível verificar o histórico de erros que já existiram, em ambas as plataformas. A utilização desta plataforma foi fulcral na fase de desenvolvimento e utilização da versão piloto das aplicações, visto que permitia perceber a origem e detetar ocorrências de eventuais erros, que só seriam detetáveis com a aplicação em modo de *debug*.

Esta ferramenta também terá muita importância quando a aplicação estiver efetivamente a ser utilizada em estudos, visto que irá permitir realizar a monitorização de todas as utilizações, permitindo identificar erros que porventura aconteçam, mas também recolher outros dados de utilização, como quantas vezes por dia a aplicação foi aberta, quanto tempo esteve aberta, entre outros. Na Figura 6-44 é possível ver a *dashboard* fornecida ao computador, com erros passados, que ao ser consultados fornecem informações detalhadas sobre esse erro. Para além disso, o programador é notificado em tempo real no e-mail de todos os erros que aconteceram no decorrer da aplicação e que poderão ser consultados no *Firebase*.

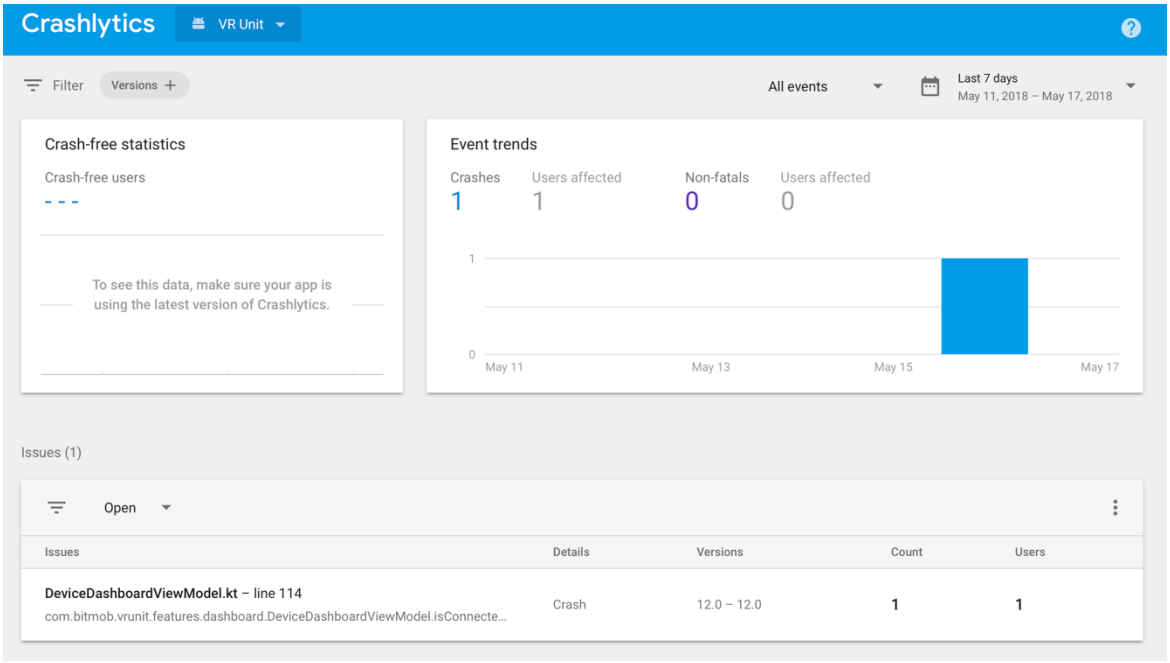


Figura 6-44 - Dashboard de crashlytics do Firebase

7 Resultados e Validação da Solução

7.1 Avaliação de Usabilidade

A aplicação foi sujeita a dois momentos de avaliação de usabilidade, num primeiro momento, foram realizados testes de usabilidade com vários voluntários de áreas diferentes da área destino da aplicação, esses primeiros testes serviram sobretudo para despistar problemas de usabilidade relacionados com a interface visual da aplicação, tais como ícones que não façam sentido, botões posicionados em sítios que não são percetíveis para o utilizador, cores que não façam sentido, tamanho da fonte, ícones e botões, entre outros.

Esse teste foi realizado da seguinte forma: todos os voluntários tinham de executar uma série de tarefas, utilizando as duas aplicações, e no final responderam a um questionário do tipo *Post-study System Usability Questionnaire* [65], dividido em 19 questões, presentes no último anexo do documento, que estão classificadas com uma escala de 1 a 7, correspondendo o valor 1 a "Concordo totalmente" e o valor 7 a "discordo totalmente". Os dados dos questionários foram recolhidos e agrupados de modo a ser possível criar alguns gráficos com os resultados das respostas dadas pelos voluntários. O questionário, do tipo *PSSUQ*, está dividido em 4 grupos [65]:

- Geral: perguntas 1 a 19;
- Usabilidade do Sistema: perguntas 1 a 8;
- Qualidade da informação do Sistema: perguntas 9 a 15;
- Qualidade da interface: perguntas 16 a 18.

As tarefas a realizar durante o teste de usabilidade foram pensadas de modo a criar um fluxo que permitisse aos participantes percorrerem todos os pontos de ambas as aplicações. A lista de tarefas é visível na Figura 7-1.

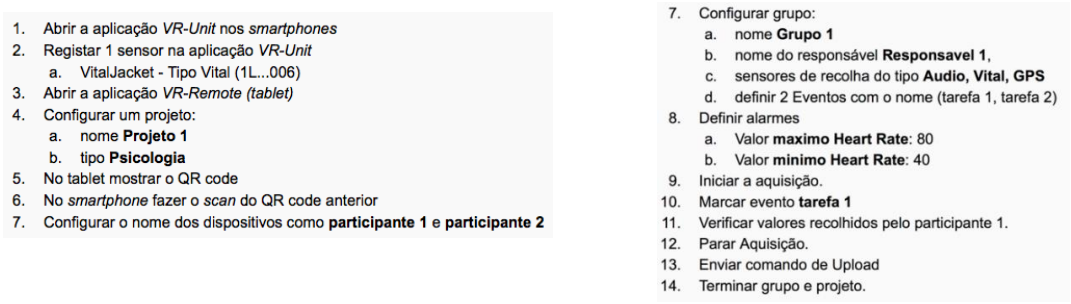


Figura 7-1 - Tarefas realizadas pelos participantes durante os testes de usabilidade.

O *feedback* dos voluntários do primeiro teste foi positivo, todos os participantes cumpriram os objetivos que as tarefas propunham, e serviu para despistar um conjunto de problemas de usabilidade que seriam críticos em cenários reais. Os problemas de usabilidade detetados neste primeiro momento de avaliação estão presentes na tabela seguinte.

Origem do problema	Observações
Botão de configuração de grupo	Botão mal posicionado.
Botão de confirmação de alarmes	Ícone do botão pouco intuitivo.
Lista de eventos	Mal posicionada.
Botão de "terminar grupo"	Mal posicionado.
Atribuir nome a um dispositivo	Falta opção no menu da célula.
Nomes técnicos no menu	"Group Selection" demasiado técnico.
Tamanho dos ícones	Alguns ícones demasiados pequenos para um tablet.
Botão de scan de código QR	Ícone do botão pouco intuitivo.

Tabela 7-1 - Problemas de usabilidade detetados com algumas observações associadas.

A amostra do primeiro teste, como foi dito, não englobava pessoas da área da psicologia, mas sim da área de informática, eletrónica e saúde. Participaram nesta avaliação 13 participantes, onde as idades dos participantes estavam compreendidas entre os 22 e os 32 anos, sendo que 2 dos participantes eram mulheres e 11 homens, dessa amostra, 6 dos participantes eram investigadores ou alunos de doutoramento e os restantes 7 eram alunos de mestrado.

Na seguinte tabela será possível ver em média os resultados atingidos para cada componente do questionário PSSUQ. A média para cada grupo, foi calculada com a soma das médias de todas as questões referentes a cada grupo e a divisão pelo número de questões, sendo que a mediana se baseou também no valor referente a todas as questões do grupo.

Grupo	Resultado (Média)	Mediana
Geral	1.887	2
Usabilidade do Sistema	1.942	2
Qualidade da info. do Sistema	1.978	2
Qualidade da interface	1.589	1

Tabela 7-2 - Resultados do questionário PSSUQ realizado.

Os resultados obtidos no teste são bastante positivos, onde todos os grupos atingem uma média abaixo de 2.0, ou seja, muito próximo do 1.0, valor limite da escala. A mediana de cada grupo também é bastante positiva, atingindo o valor de 2 em 3 grupos e 1 no grupo da qualidade da interface.

A pergunta do questionário que obteve a melhor resposta corresponde à pergunta 16, "Gostei de utilizar a interface deste sistema.", que teve um score de 1.38, a pergunta com o resultado menos positivo, embora que possa ser considerado bom, foi a pergunta 12, "A informação foi eficaz para me ajudar a completar as tarefas e os cenários.", o que foi um indicativo de modo a tentar tornar a aplicação mais rica em informação para o utilizador.

Figura 7-2 - Voluntário a realizar o teste de usabilidade.

7.2 Utilizações com investigadores

Numa segunda fase, foi realizado outra avaliação de usabilidade, desta vez já com os problemas detetados no 1º teste realizado resolvidos. Este teste foi executado por voluntários da área da psicologia, ou seja, uma das áreas de destino do sistema desenvolvido, de modo a comprovar se o sistema era válido ou não na utilização normal de um investigador de psicologia. Neste teste houve uma tentativa de colocar os participantes num cenário de recolha, sendo que as tarefas foram ligeiramente alteradas para esse efeito, por exemplo, a exportação dos dados recolhidos durante o teste foi uma das novas partes abordadas no teste de usabilidade.

No total, 7 responsáveis por estudos no departamento de psicologia da universidade de Aveiro foram angariados para a realização deste teste. Tinham idades compreendidas entre os 22 e os 29 anos, desses 7, 5 eram do género feminino e 2 do masculino. De modo a avaliar a usabilidade do sistema foi utilizado o questionário PSSUQ, também de modo a ter uma maneira de comparar os resultados de cada grupo desse questionário com os resultados obtidos anteriormente.

O *feedback* de todos os participantes foi bastante positivo, após o término das tarefas foi questionado a todos os participantes se achavam que a aplicação era útil na realização dos seus estudos atuais, ou estudos futuros, à qual todos responderam afirmativamente. Também foi questionado aos participantes que já tinham

utilizado a linha antiga de aplicações, que neste caso eram 2, se achavam esta aplicação superior em termos de funcionalidades, interface visual e utilidade nos estudos, ao qual responderam que sim a todos os pontos.

Os resultados do questionário PSSUQ irá ser apresentado na tabela seguinte. Em relação ao questionário realizado pelo primeiro grupo de participantes, os resultados foram ligeiramente mais positivos, melhorando o *score* em todos os grupos do PSSUQ, em termos de média. Pode-se concluir que o resultado é positivo, visto que seria importante no mínimo manter os resultados anteriores com o grupo de utilizador para o qual a aplicação foi destinada, sendo que outro dos pontos muito positivos foi a utilidade que todos os utilizadores conseguiram ver na aplicação, caso fosse utilizada nos seus estudos.

Grupo	Resultado	Mediana	Resultado 1º teste – Média	Resultado 1º teste - Mediana
Geral	1.706	2	1.887	2
Usabilidade do Sistema	1.714	1.5	1.942	2
Qualidade da info. do Sistema	1.918	2	1.978	2
Qualidade da interface	1.285	1	1.589	1

Tabela 7-3 - Resultados do 2º teste de usabilidade comparados com o 1º.

No questionário, a pergunta que obteve o resultado menos positivo, foi a pergunta 9 “O sistema deu mensagens de erros que me indicaram claramente como resolver os problemas.”. A análise feita a este resultado, é a de que até pode ser um ponto positivo, porque nunca se enganaram a preencher nenhum formulário ou configuração, caso existisse erro iriam ter uma mensagem própria. A pergunta com melhor score, foi a pergunta 14 “A informação fornecida pelo sistema foi fácil de entender”, o que demonstra que a aplicação estava adequada a uma das suas finalidades, ser intuitiva e demonstrar bem toda a informação que pretende mostrar, aos olhos de um investigador. Em termos de problemas de usabilidade, nesta amostra, nenhum dos participantes identificou problemas de usabilidade, o que significa que todos os problemas principais foram despistados e corrigidos do 1º para o 2º teste de usabilidade.

Figura 7-3 - Investigador de psicologia a realizar o teste de usabilidade.

7.3 Utilizações no terreno

Durante os finais de março e finais de maio, a aplicação *VR-Unit*, foi utilizada como elemento de recolha de sinais vitais e sinal de GPS por um aluno de Engenharia Mecânica da Universidade de Aveiro. O âmbito destas recolhas era o de recolher dados de sinais vitais de modo a poder verificar alterações cardíacas durante um percurso de bicicleta entre a Universidade de Aveiro e a estação de comboios da cidade, mas também com um trajeto na cidade do porto, essas alterações eram posteriormente correlacionadas com as coordenadas de GPS recolhidas de modo a perceber se as variações ocorriam em subida, descida, ou trajeto plano. Após esses resultados estarem correlacionados, o objetivo do aluno será de construir uma plataforma que permita os utilizadores verificarem os melhores percursos tendo em conta o nível de esforço físico necessário.

A aplicação foi utilizada durante 5 dias, onde foram realizadas diversas aquisições, com os sensores *VitalJacket* e GPS interno do dispositivo móvel. Em nenhuma delas houve registo de problemas, quer por parte do utilizador da aplicação, quer problemas que tenham sido detetados na dashboard de controlo do *Fabric* ou do *Firebase*. Os dados recolhidos foram analisados de modo a serem detetados erros nas recolhas, e também não foram detetados problemas.

Para além desta utilização, a aplicação foi utilizada em testes com o novo sistema de visualização de dados em tempo real desenvolvido no âmbito do projeto *VR2Market* por um aluno de Mestrado. O objetivo destes testes era o de testar a integração entre estes dois projetos, servindo como mecanismo de validação de ambas as soluções. Foi realizada uma recolha experimental, simulada em laboratório, onde com o auxílio do *VR-Remote*, uma aplicação *VR-Unit* foi configurada para uma recolha de sinais vitais e ambientais, onde um simulador de sinais gerava o batimento cardíaco. A visualização dos dados da recolha simulada, pode ser visível na figura seguinte, onde estão presentes as aplicações *VR-Remote* e *VR-Unit*.

Figura 7-4 - Recolha simulada em laboratório, com visualização dos dados em tempo real.

Foi efetuada uma recolha no terreno, num percurso em forma de circuito na Universidade de Aveiro, com a utilização da aplicação *VR-Unit*, com a presença de 3 voluntários. Esse circuito foi repetido 3 vezes, na primeira vez os voluntários deslocavam-se a um ritmo normal, na 2ª vez a ritmo acelerado, e na 3ª vez em ritmo de corrida, de modo a simular eventuais movimentações semelhantes a um operacional no terreno. Nestas recolhas foram recolhidos dados fisiológicos com o auxílio de uma camisola com um *VitalJacket*

integrado, dados de localização com o *GPS* interno e também dados ambientais através da utilização do sensor *FREMU*.

Figura 7-5 - Recolha no terreno.

Nestas recolhas existiu um 4º voluntário, que estava presente em laboratório, no papel de comandante a visualizar os dados recolhidos em tempo real nas diversas *dashboard*, presentes no sistema de visualização externo. Nas figuras seguintes serão visíveis alguns *screenshots* dessas *dashboards*.

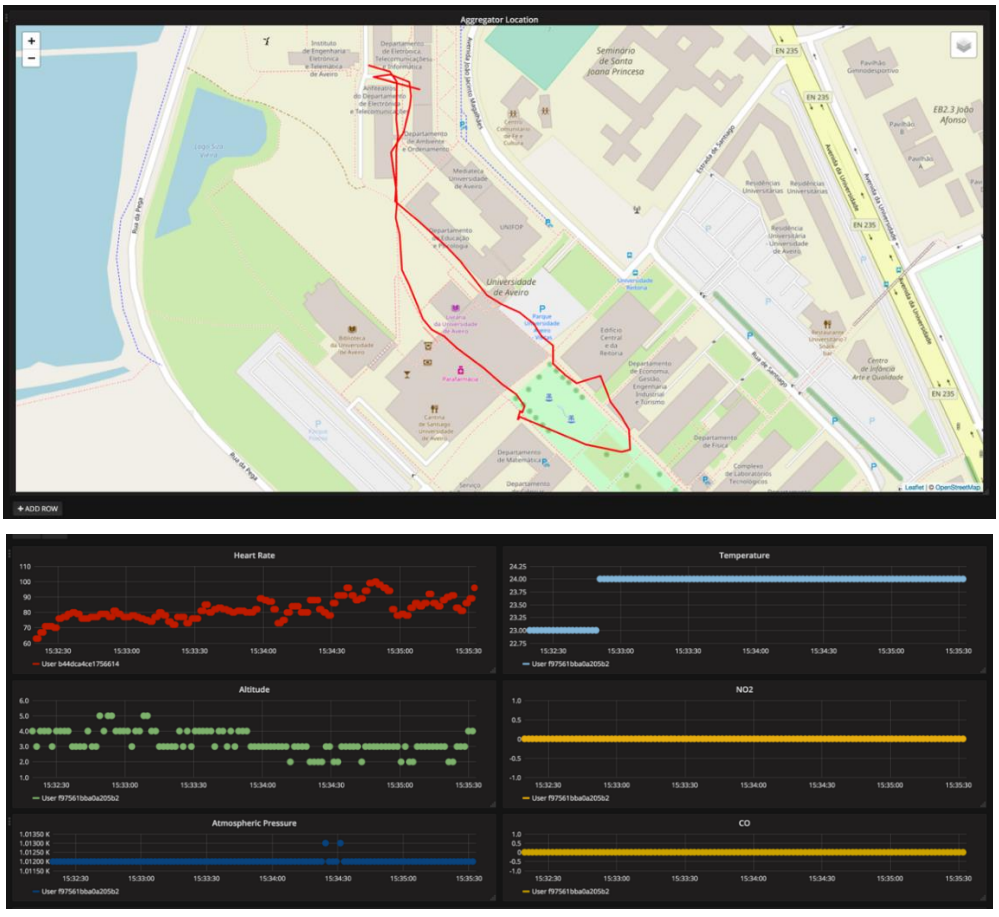


Tabela 7-4 - Dashboards de visualização em tempo real.

7.4 Reengenharia às aplicações existentes

Este trabalho teve como base uma reformulação de um conjunto de aplicações existentes, sendo que as maiores alterações foram ao nível da linguagem de programação utilizada, padrão de arquitetura interno das aplicações, e também comunicação entre componentes. No entanto as aplicações desenvolvidas tiveram mais alguns pontos que sofreram alguma alteração, não ao nível de programação efetiva, mas a outro tipo de metodologias, como organização de código e recursos, e ainda uma componente de testes.

No quadro seguinte será feita uma comparação entre pontos tecnológicos entre as aplicações antigas e o sistema desenvolvido.

Variável Tecnológica	**Vitals Recorder**	**Solução pré-existente**
Linguagem de programação	Kotlin	Java
Motor de Base de dados interno	SQLite c/ *Room*	SQLite nativo
Padrão de Arquitetura	MVVM	MVC
Comunicação Interna	*EventBus*	*Intent & Broadcast Receivers*
Comunicação entre aplicações	*Broker MQTT*	*Bluetooth & Sockets WiFi*
Design para tablet e telemóvel na aplicação controladora	Suportado	Apenas para tablet
Suporte a língua PT e Inglesa	Suportado	Apenas inglês
Ofuscação e redução de código e recursos	Implementado	Não implementado
Testes de *stress* à interface	Realizados	Sem informação
Testes à base de dados	Realizados	Sem informação
Testes de *memory leaks*	Realizados	Sem informação
Mecanismo de *layouts*	*Constraint Layout*	*Nested layouts* com *Relative, Grid e Coordinator Layout*
Utilização de plataformas de *Crashlytics e Analytics*	*Fabric* & *Firebase*	Não existe
Formato dos dados recolhidos	*JSON e CSV*	ASCII

Tabela 7-5 - Quadro de comparação tecnológica entre as aplicações desenvolvidas e as aplicações antigas.

Em termos de funcionalidades e fluxo de recolhas, monitorização e controlo foram acrescentados novos componentes aos já existentes, o seu quadro resumo será apresentado de seguida.

Funcionalidade	**Vitals Recorder**	**Solução pré-existente**
Possibilidade de definição de estudo e grupos de recolha	Suportado	Apenas grupo de recolha
Possibilidade de adicionar e remover sensores dinamicamente	Suportado	Não, sensores estáticos
Visualizar conteúdo em tempo real em ambas as aplicações	Suportado	Apenas na aplicação agregadora
Definição dinâmica de eventos a marcar durante a sessão	Sim	Eventos estáticos
Definição e visualização de alarmes em tempo real	Sim	Não
Consulta de histórico de configurações, eventos, alarmes	Sim	Não
Emparelhamento de Dispositivos por *Bluetooth* e código QR	Sim	Apenas Bluetooth
Mecanismos de reconexão a sensores em caso de falha	Sim	Sim
Mecanismos de reconexão aos dispositivos móveis em caso de falha (aquisição de grupo)	Sim (mecanismo de *keep alive* temporal)	Sim (reconexão explícita do utilizador)
Visualização do estado de conexão a dispositivos móveis e sensores em tempo real (aquisição de grupo)	Sim (automaticamente, sem interação do utilizador, em ambas as aplicações)	No VR-Unit: apenas a sensores No VR-Remote: através de uma ação explícita por clique num botão
Mecanismo de reconexão aos *brokers* em caso de falha	Sim	Não

Tabela 7-6 - Quadro de comparação de funcionalidades.

Na tabela seguinte serão abordados os sensores que foram acrescentados em relação ao sistema antigo.

Sensor	**Variável recolhida**
Bitalino	ECG, EDA, EMG
Bitalino *Low Energy*	Batimento Cardíaco
Xiaomi Mi Band 2	Batimento Cardíaco

Tabela 7-7 - Quadro de novos sensores suportados.

7.5 Visualização de dados

No seguinte subcapítulo serão apresentados alguns ecrãs de visualização de dados, da maioria dos sensores suportados pelas aplicações.

Vital Jacket

O Vital Jacket é um sensor que recolhe sinais vitais, mais propriamente batimento cardíaco, ECG e temperatura corporal. O sensor utilizado nas imagens seguintes não tem a capacidade de recolher temperatura corporal, daí o valor de "-1.0" no ecrã apresentado. O sensor capta os dados da pessoa através da ligação dos elétrodos presentes na camisola ao corpo do utilizador, em 3 pontos no corpo.

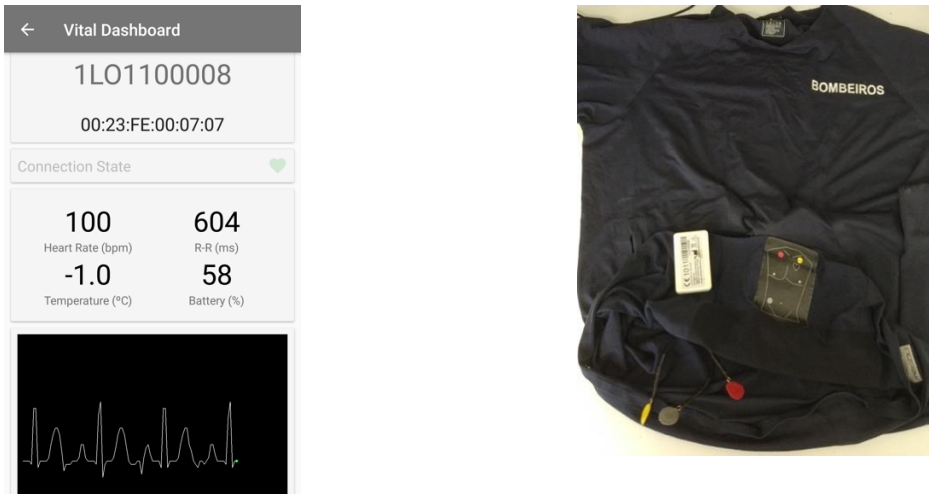


Figura 7-6 - Exemplo de dados do Vital Jacket e setup utilizado.

Mi Band

A Mi Band é uma *Smartband* da Xiaomi que permite a leitura do batimento cardíaco do seu utilizador, nas figuras seguintes é possível ver uma imagem da pulseira e do seu ecrã dedicado na aplicação *VR-Unit*.

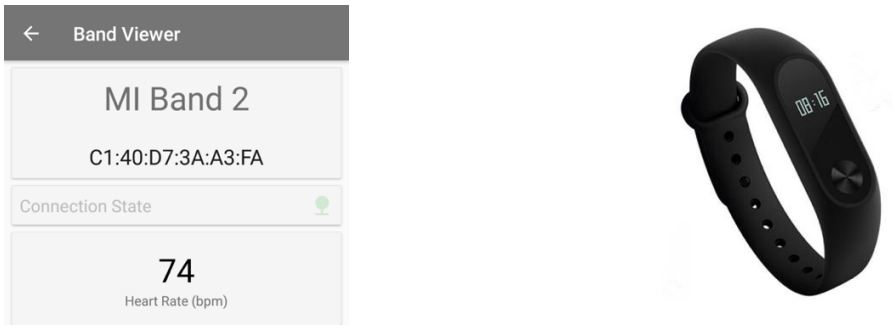


Figura 7-7 - Exemplo de recolha com a Mi Band 2 e respetiva smartband.

Bitalino LE

O Bitalino também tem uma versão *Low Energy*, que não é tão potente em termos computacionais como a versão abordada anteriormente. Com esta versão da placa fez-se uma recolha de batimento cardíaco com um periférico que é colocado ao redor do dedo indicador do participante.

Figura 7-8 - Exemplo de agregação com o Bitalino Low Energy e setup utilizado.

Bitalino

O Bitalino é uma placa semelhante a um microcontrolador com a capacidade de recolher vários dados de sinais vitais como ECG, EMD, EDA ou EEG, com uma gama de valores entre 0 e 1023. No setup utilizado apenas se realizou recolha de ECG, EMG e EDA, devido a não existir em laboratório elétrodos que consigam realizar recolha de EEG.

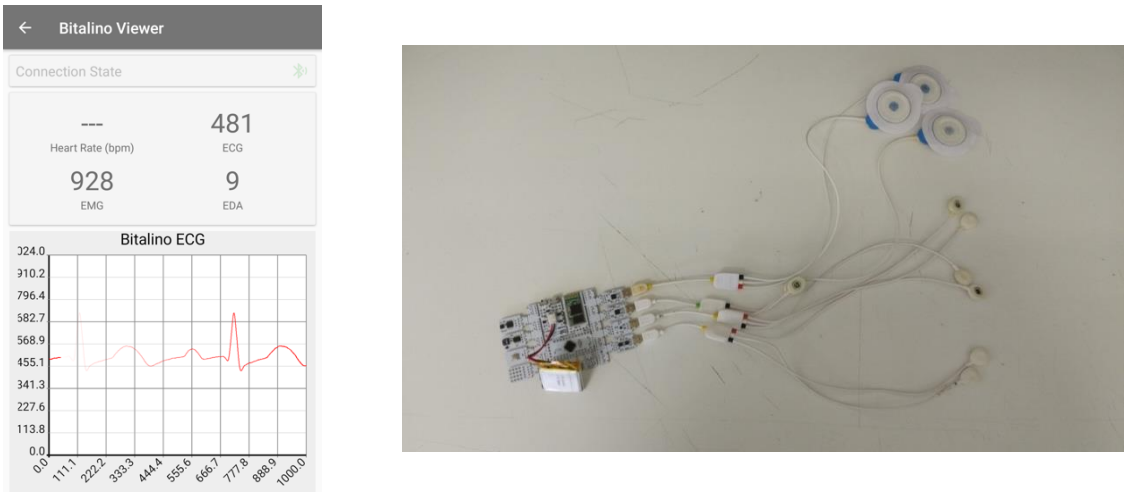


Figura 7-9 - Exemplo de agregação de dados com o sensor Bitalino, e setup utilizado.

Fremu

O Fremu é um sensor ambiental de pequenas dimensões capaz de recolher dados de temperatura, altitude, pressão atmosférica e nível de monóxido de carbono.

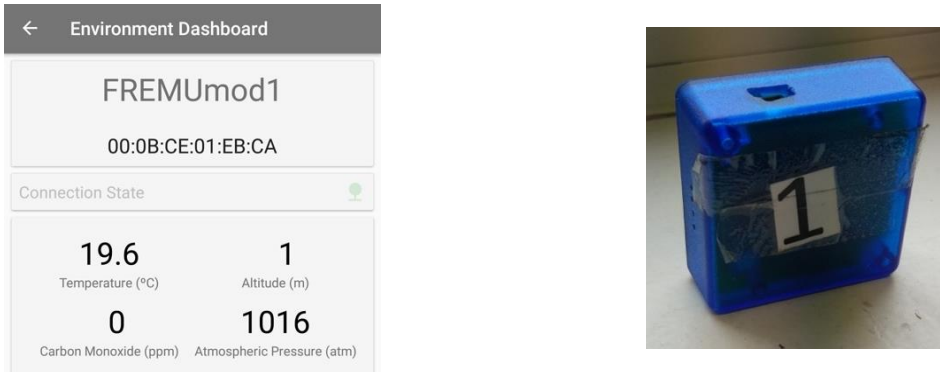


Figura 7-10 - Exemplo de agregação com o sensor Fremu e imagem do sensor.

GPS Interno

O GPS Interno, é como o nome indica um dos sensores internos ao telemóvel, sendo capaz de captar latitude, longitude e altitude, juntamente com uma marca temporal associada aos dados recolhidos. Na imagem seguinte é possível ver uma amostra típica de uma recolha de GPS, onde associado aos dados de localização recolhidos é visível no mapa uma imagem da localização atual.

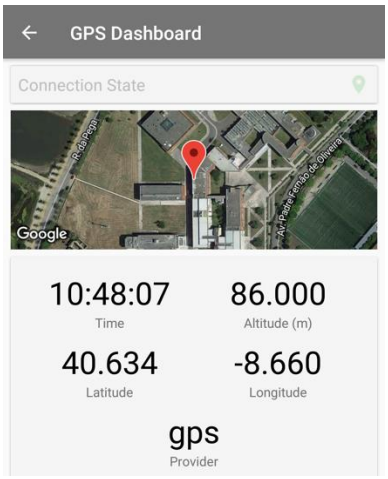


Figura 7-11 - Exemplo de agregação com o GPS Interno do smartphone.

Expressão facial

O reconhecimento da emoção através da análise da expressão facial do utilizador é um dos tipos de dados recolhidos pela a aplicação, no ecrã seguinte é possível ver as 5 variáveis que estão a ser analisadas, que são a alegria, tristeza, desgosto, raiva e medo.

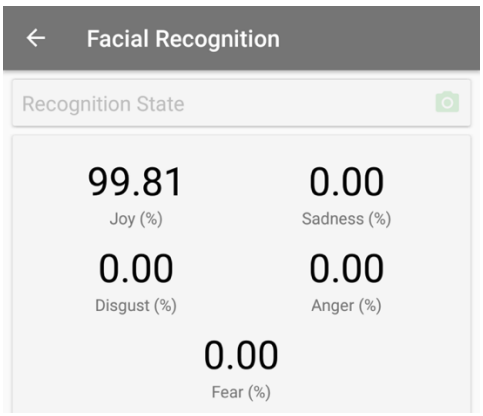


Figura 7-12 - Exemplo de agregação efetuada com o reconhecimento de expressão facial.

8 Conclusão

Neste capítulo serão apresentadas as conclusões desta dissertação, onde será feita uma análise geral sobre o trabalho desenvolvido e o trabalho futuro que ainda poderá ser feito de modo a continuar este projeto.

8.1 Trabalho desenvolvido

O projeto desta dissertação consistia no desenvolvimento de aplicações para agregação de sinais vitais no âmbito do projeto *VR2Market*, no entanto foi percebido que este trabalho poderia ser mais rico se fosse também aplicado a recolhas de psicofisiologia. Desse novo requisito nasceu o produto final do trabalho, um sistema completo de agregação de dados, especialmente de sinais vitais, mas também de outros sensores, em que nesse sistema é possível monitorizar, controlar e exportar os dados de um conjunto de aplicações, através de um ponto central. Apesar de existir um ponto central de controlo, o sistema pode ser caracterizado como móvel, visto que em cenários de recolhas autónomas, ou de recolhas controladas, o *setup* pode ser adaptado a ambientes ecológicos nos quais experiências de psicologia decorrem no terreno.

Era importante que essas as aplicações fossem adaptáveis a vários contextos e esse é um dos pontos que se pode considerar como atingido, visto que o sistema desenvolvido será capaz de ser aplicado ao cenário de psicologia, onde o responsável com a aplicação *VR-Remote* controlará as aplicações *VR-Unit* associadas a cada participante, podendo exportar os dados no final de cada experiência. É adaptável ao cenário de bombeiros, o foco principal do projeto *VR2Market*, onde cada operacional poderá transportar no seu telemóvel a aplicação *VR-Unit* podendo realizar uma recolha durante um teatro de operações de incêndio, podendo no final da sua missão realizar o *Upload* dos dados para o ponto central. Pode ainda ser adaptado a qualquer cenário de equipa, como cenários desportivos ou de *fitness*, onde um utilizador normal, pode realizar uma recolha de sinais vitais enquanto corre, ou enquanto anda de bicicleta, ou na prática de qualquer outro desporto, ou um treinador ou médico desportivo, pode realizar a monitorização dos seus atletas.

Para além da adaptação a vários contextos, era importante a aplicação estar construída de modo a ser fácil adicionar suporte a novos sensores, sendo que foi mais um objetivo realizado, dado que foi desenvolvida uma biblioteca *Android* que poderá ser utilizada em qualquer aplicação através da importação do código dessa biblioteca. Essa biblioteca pode ainda ser estendida de modo a ser possível adicionar novos tipos de sensores, ou novos sensores dos tipos já existentes.

8.2 Trabalho futuro

Os requisitos e objetivos iniciais deste projeto foram conseguidos, no entanto tal como em todos os projetos há sempre espaço para melhorias ou alterações.

A primeira melhoria futura proposta, seria desenvolver uma aplicação *VR-Unit* em *iOS*, ou noutro tipo de tecnologias, como aplicações híbridas com o uso de *React Native*, de modo à aplicação estar presente nos dois maiores sistemas operativos móveis à escala global, sendo que no momento só é suportada em *Android*. A aplicação *Remote*, à partida não necessitará de uma versão numa plataforma móvel, visto que pode controlar dispositivos de qualquer plataforma devido à utilização do *MessageCollector* como canal de comunicação.

Outra das melhorias que é possível de realizar, é desenvolver um *Remote* versão *Web*, ou seja, todo o papel de monitorização e controlo ser possível de fazer numa *WebApp*, onde deverá ser possível também visualizar os dados em tempo real que todas as *Unit* estão a recolher. Mas também, com o objetivo de ser um ambiente integrado de gestão de estudos e participantes, tomando assim o papel de sistema de informação, com a componente de monitorização e controlo.

Em relação às aplicações existentes em si, seria melhor em vez da *Dropbox*, utilizar um armazenamento próprio dos dados recolhidos por todas as aplicações, de modo a ultrapassar eventuais questões de segurança e ética dos dados que possam ser colocadas, aos sistemas utilizados atualmente como forma de armazenamento. Seria também interessante existir suporte a novos sensores, de modo a poder continuar a desenvolver e estender a biblioteca de gestão de sensores desenvolvida neste projeto.

9 Referências

[1] N. Mohd Suki and N. Mohd Suki, “Dependency on Smartphones : An Analysis of Structural Equation Modelling,” *J. Teknol. Full Pap.*, vol. 1, pp. 49–54, 2013.

[2] G. M. Harari, N. D. Lane, R. Wang, B. S. Crosier, A. T. Campbell, and S. D. Gosling, “Using Smartphones to Collect Behavioral Data in Psychological Science: Opportunities, Practical Considerations, and Challenges,” *Perspect. Psychol. Sci.*, 2016.

[3] “Institute of Electronics and Informatics Engineering of Aveiro - IEETA.” [Online]. Available: http://wiki.ieeta.pt/wiki/index.php/Main_Page. [Accessed: 07-May-2018].

[4] “Vital Responder,” 2008. [Online]. Available: http://vitalresponder.inesctec.pt/. [Accessed: 09-Feb-2018].

[5] J. Boase, “Implications of software- based mobile media for social research,” *Mob. Media Commun.*, vol. 1, no. 1, pp. 57–62.

[6] G. Chittaranjan, B. Jan, and D. Gatica-Perez, “Who’s who with big-five: Analyzing and classifying personality traits with smartphones,” in *Proceedings - International Symposium on Wearable Computers, ISWC*, 2011, pp. 29–36.

[7] G. Miller, “The Smartphone Psychology Manifesto,” *Perspect. Psychol. Sci.*, 2012.

[8] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell, “AD HOC AND SENSOR NETWORKS A Survey of Mobile Phone Sensing,” 2010.

[9] A. Rieger, S. Fenger, S. Neubert, M. Weippert, S. Kreuzfeld, and R. Stoll, “Psychophysical workload in the operating room: primary surgeon versus assistant,” *Surg. Endosc. Other Interv. Tech.*, vol. 29, no. 7, pp. 1990–1998, 2015.

[10] V. Pejovic, N. Lathia, C. Mascolo, and M. Musolesi, “Mobile-Based Experience Sampling for Behaviour Research,” 2015.

[11] Z. Yang, · Qihao Zhou, L. Lei, K. Zheng, and W. Xiang, “An IoT-cloud Based Wearable ECG Monitoring System for Smart Healthcare,” *J. Med. Syst.*, vol. 40, 2016.

[12] A. Stopczynski *et al.*, “Smartphones as pocketable labs: Visions for mobile brain imaging and

neurofeedback," *Int. J. Psychophysiol.*, vol. 91, no. 1, pp. 54–66, Jan. 2014.

[13] S. Debener, R. Emkes, M. De Vos, and M. Bleichner, "Unobtrusive ambulatory EEG using a smartphone and flexible printed electrodes around the ear," *Nat. Publ. Gr.*, 2015.

[14] M. Schirmer Jun-Prof Dr-Ing Hagen Höpfner, "Smartphone Hardware Sensors."

[15] "the MyExperience tool." [Online]. Available: http://myexperience.sourceforge.net/. [Accessed: 10-Mar-2018].

[16] J. Hicks *et al.*, "AndWellness: An Open Mobile System for Activity and Experience Sampling."

[17] K. K. Rachuri, M. Musolesi, C. Mascolo, P. J. Rentfrow, C. Longworth, and A. Aucinas, "EmotionSense: A Mobile Phones based Adaptive Platform for Experimental Social Psychology Research."

[18] H. Tangmunarunkit *et al.*, "Ohmage: A General and Extensible End-to-End Participatory Sensing Platform ACM Reference Format," *ACM Trans. Intell. Syst. Technol. ACM Trans. Intell. Syst. Technol*, vol. 6, no. 21, 2015.

[19] "Purple Robot - CBITs TECH Web Site." [Online]. Available: https://tech.cbits.northwestern.edu/purple-robot/. [Accessed: 27-Jun-2018].

[20] S. Saeb *et al.*, "Mobile Phone Sensor Correlates of Depressive Symptom Severity in Daily-Life Behavior: An Exploratory Study."

[21] "Open Data Kit - ODK 1." [Online]. Available: https://opendatakit.org/software/odk1/. [Accessed: 01-Jul-2018].

[22] M. I. Rizqyawan, M. F. Amri, R. P. Pratama, and A. Turnip, "Design and development of Android-based cloud ECG monitoring system," in *Proceedings - 2016 3rd International Conference on Information Technology, Computer, and Electrical Engineering, ICITACEE 2016*, 2017, pp. 1–5.

[23] V. P. Rachim and W. Y. Chung, "Wearable Noncontact Armband for Mobile ECG Monitoring System," *IEEE Trans. Biomed. Circuits Syst.*, vol. 10, no. 6, pp. 1112–1118, 2016.

[24] G. Biagetti, P. Crippa, L. Falaschetti, S. Orcioni, and C. Turchetti, "Wireless surface electromyograph and electrocardiograph system on 802.15.4," *IEEE Trans. Consum. Electron.*, vol. 62, no. 3, pp. 258–266, 2016.

[25] T. Sakai *et al.*, "EDA-BASED ESTIMATION OF VISUAL ATTENTION BY OBSERVATION OF EYE BLINK FREQUENCY," *Int. J. SMART Sens. Intell. Syst.*, vol. 10, no. 2, 2017.

[26] R. Yoshida *et al.*, "Feasibility Study on Estimating Visual Attention using Electrodermal Activity."

[27] C. Chevallier *et al.*, "Measuring social attention and motivation in autism spectrum disorder using eye-tracking: Stimulus type matters," *Autism Res.*, vol. 8, no. 5, pp. 620–628, 2015.

[28] J. P. S. Cunha, B. Cunha, A. S. Pereira, W. Xavier, N. Ferreira, and L. Meireles, "Vital-Jacket: A

wearable wireless vital signs monitor for patients' mobility in cardiology and sports," *Pervasive Comput. Technol. Healthc. (PervasiveHealth), 2010 4th Int. Conf. on-NO Permis.*, pp. 1–2, 2010.

[29] "Vital Jacket Website." [Online]. Available: http://www.vitaljacket.com/. [Accessed: 10-Mar-2018].

[30] J. Guerreiro, R. Martins, H. Silva, A. Lourenço, and A. Fred, "BITalino: A Multimodal Platform for Physiological Computing," *Proc. 10th ICINCO Conf*, pp. 500–506, 2013.

[31] "BITalino - Biomedical Equipment | Low-Cost Toolkit." [Online]. Available: http://bitalino.com/en/. [Accessed: 10-Mar-2018].

[32] "Eye Tracking System for HMD2, Binoc | EYETRAKHMD2BINO, EYETRAKHMD2MONO, EYETRAKHMD3RDBI, EYETRAKHMD3RDMO, EYETRKHMD1B-B90, EYETRKHMD1B-M90 | Research | BIOPAC." [Online]. Available: https://www.biopac.com/product/eye-tracking-systems/#product-tabs. [Accessed: 07-May-2018].

[33] "MP160 Starter Systems | BIOPAC." [Online]. Available: https://www.biopac.com/product-category/research/systems/mp150-starter-systems/. [Accessed: 13-May-2018].

[34] J. P. Higgins, "Smartphone Applications for Patients' Health and Fitness," *The American journal of medicine*, vol. 129, no. 1. pp. 11–19, 2016.

[35] Williams Rhiannon, "Apple iOS: a brief history - Telegraph." [Online]. Available: http://www.telegraph.co.uk/technology/apple/11068420/Apple-iOS-a-brief-history.html. [Accessed: 09-Feb-2018].

[36] Callaham John, "The history of Android OS: its name, origin and more." [Online]. Available: https://www.androidauthority.com/history-android-os-name-789433/. [Accessed: 09-Feb-2018].

[37] C. F. Greer and D. A. Ferguson, "Tablet computers and traditional television (TV) viewing: Is the iPad replacing TV?," *Convergence*, vol. 21, no. 2, pp. 244–256, 2015.

[38] K. Divyap and S. Venkata Krishnakumar, "COMPARATIVE ANALYSIS OF SMART PHONE OPERATING SYSTEMS ANDROID, APPLE iOS AND WINDOWS," *Int. J. Sci. Eng. Appl. Sci.*, no. 22, pp. 2395–3470, 2016.

[39] "Kotlin and Android | Android Developers." [Online]. Available: https://developer.android.com/kotlin/index.html. [Accessed: 10-Feb-2018].

[40] "Design | Android Developers." [Online]. Available: https://developer.android.com/design/index.html. [Accessed: 10-Feb-2018].

[41] "User Interface Guidelines | Android Developers." [Online]. Available: https://developer.android.com/guide/practices/ui_guidelines/index.html. [Accessed: 10-Feb-2018].

[42] C. González García, J. Pascual-Espada, C. Pelayo G-Bustelo, and J. M. Cueva-Lovelle, "Swift vs. Objective-C: A New Programming Language," *Int. J. Interact. Multimed. Artif. Intell.*, vol. 3, no. 3, p. 74, 2015.

[43] “Auto Layout Guide: Understanding Auto Layout.” [Online]. Available: https://developer.apple.com/library/content/documentation/UserExperience/Conceptual/AutolayoutPG/. [Accessed: 10-Feb-2018].

[44] S.-H. Lim, “Experimental comparison of hybrid and native applications for mobile systems,” *Int. J. Multimed. Ubiquitous Eng.*, vol. 10, no. 3, pp. 1–12, 2015.

[45] W. Jobe, “Native Apps Vs. Mobile Web Apps,” *Int. J. Interact. Mob. Technol.*, vol. 7, no. 4, p. 27, 2013.

[46] M. Aniche, G. Bavota, C. Treude, M. A. Gerosa, and A. van Deursen, “Code smells for Model-View-Controller architectures,” *Empirical Software Engineering*, pp. 1–37, 2017.

[47] K. Sokolova, M. Lemercier, and L. Garcia, “Android Passive MVC: a Novel Architecture Model for the Android Application Development,” *Patterns 2013*, no. c, pp. 7–12, 2013.

[48] S. Ng, “A Pattern Language for Mobile Application Development,” *Commun. Comput. Inf. Sci.*, 2015.

[49] M. R. J. Qureshi and F. Sabir, “a Comparison of Model View Controller and Model View Presenter.,” *Sci. Int.*, vol. 25, no. 1, pp. 7–9, 2013.

[50] “Android Architecture Components | Android Developers.” [Online]. Available: https://developer.android.com/topic/libraries/architecture/index.html. [Accessed: 11-Feb-2018].

[51] S. P. Tripathi and T. Narang, “Applying Model View View-Model and Layered Architecture for Mobile Applications,” vol. 20, no. 3, pp. 215–221, 2016.

[52] “Google I/O 2017.” [Online]. Available: https://events.google.com/io2017/. [Accessed: 08-May-2018].

[53] “Handling Lifecycles with Lifecycle-Aware Components | Android Developers.” [Online]. Available: https://developer.android.com/topic/libraries/architecture/lifecycle. [Accessed: 08-May-2018].

[54] “INESC TEC.” [Online]. Available: https://www.inesctec.pt/en. [Accessed: 12-May-2018].

[55] “IT - website.” [Online]. Available: https://www.it.pt/. [Accessed: 12-May-2018].

[56] “CESAM » CESAM - Mission & Goals.” [Online]. Available: http://www.cesam.ua.pt/index.php?menu=89&language=eng&tabela=geral. [Accessed: 12-May-2018].

[57] “EventBus: Events for Android - Open Source by greenrobot.” [Online]. Available: http://greenrobot.org/eventbus/. [Accessed: 18-Feb-2018].

[58] “Data Binding Library | Android Developers.” [Online]. Available: https://developer.android.com/topic/libraries/data-binding/index.html. [Accessed: 03-Mar-2018].

[59] “Material Design for Android | Android Developers.” [Online]. Available: https://developer.android.com/design/material/index.html. [Accessed: 03-Mar-2018].

[60] “Improve Your Code with Lint | Android Studio.” [Online]. Available: https://developer.android.com/studio/write/lint.html. [Accessed: 03-Mar-2018].

[61] H. Shahriar, S. North, and E. Mawangi, “Testing of memory leak in android applications,” in *Proceedings - 2014 IEEE 15th International Symposium on High-Assurance Systems Engineering, HASE 2014*, 2014, pp. 176–183.

[62] “UI/Application Exerciser Monkey | Android Studio.” [Online]. Available: https://developer.android.com/studio/test/monkey.html. [Accessed: 03-Mar-2018].

[63] “Firebase Test Lab Robo Test | Firebase.” [Online]. Available: https://firebase.google.com/docs/test-lab/android/robo-ux-test. [Accessed: 15-May-2018].

[64] “Shrink Your Code and Resources | Android Studio.” [Online]. Available: https://developer.android.com/studio/build/shrink-code.html. [Accessed: 03-Mar-2018].

[65] A. Rosa, A. Martins, V. Costa, A. Queiros, A. Silva, and N. Rocha, “European Portuguese validation of the Post-Study System Usability Questionnaire (PSSUQ),” in *2015 10th Iberian Conference on Information Systems and Technologies (CISTI)*, 2015, pp. 1–5.

Anexo 1 – Especificação dos Casos de Uso

Neste anexo irão estar presentes os casos de uso descritos com detalhe.

Caso de Uso 1 – Registar Sensores

Nome	**Registar sensores**
Ator	Utilizador
Pré-condições	
Sequência de Eventos	1. O utilizador abre a aplicação 2. O utilizador no menu da aplicação seleciona a opção de registar sensores. 3. O utilizador seleciona a opção de adicionar 4. Da lista disponibilizada o utilizador seleciona os vários sensores que pretende e define o seu tipo.
Sequência alternativa	

Tabela 9-1 - Especificação caso de uso 1.

Caso de Uso 2 – Realizar aquisição autónoma

Nome	**Realizar aquisição autónoma**
Ator	Utilizador
Pré-condições	
Sequência de Eventos	1. O utilizador abre a aplicação, e no ecrã principal seleciona o botão que sinaliza a realização de aquisição autónoma. 2. O utilizador seleciona os tipos de sensores que quer utilizar na aquisição. 3. O utilizador inicia a aquisição. 4. O utilizador pode verificar os valores recebidos em tempo real. 5. O utilizador termina a aquisição. 6. O utilizador faz o upload dos dados.
Sequência alternativa	

Tabela 9-2 - Especificação caso de uso 2.

Caso de Uso 3 – Participar em aquisição controlada

Nome	Participar em aquisição controlada
Ator	Utilizador
Pré-condições	Ligação *WiFi* estabelecida.
Sequência de Eventos	1. O utilizador abre a aplicação, e no ecrã principal seleciona o botão simboliza o *scan* de código QR. 2. O utilizador espera pelo início da aquisição. 3. O utilizador pode verificar os valores recebidos em tempo real. 4. O utilizador espera pelo final da aquisição. 5. O utilizador faz o upload dos dados ou espera pelo comando de *upload*.
Sequência alternativa	1. O utilizador abre a aplicação e espera pelo emparelhamento por *Bluetooth*. 2. O utilizador espera pelo início da aquisição. 3. O utilizador pode verificar os valores recebidos em tempo real. 4. O utilizador espera pelo final da aquisição. O utilizador faz o upload dos dados ou espera pelo comando de *upload*.

Tabela 9-1 - Especificação do caso de uso 3.

Caso de Uso 4 – Consultar histórico

Nome	Consultar histórico
Ator	Utilizador
Pré-condições	
Sequência de Eventos	1. Abrir o menu da aplicação e selecionar a opção de histórico. 2. Visualizar o histórico de projetos e escolher o projeto pretendido. 3. Visualizar histórico de grupos do projeto selecionado e selecionar o grupo pretendido. 4. Visualizar informações do grupo.
Sequência alternativa	1. Abrir o menu da aplicação e selecionar a opção de histórico de alarmes. Visualizar o histórico de alarmes que gerou.

Tabela 9-2 - Especificação do caso de uso 4.

Caso de Uso 5 – Configurar estudo/projeto de aquisição

Nome	Configurar projeto de aquisição
Ator	Responsável pela aquisição
Pré-condições	Ligação a uma rede *WiFi*.
Sequência de Eventos	1. O utilizador abre a aplicação. 2. O utilizador escolhe a opção de configurar projeto. 3. O utilizador configura o projeto com os parâmetros pretendidos. 4. O utilizador pressiona o botão de finalizar configuração.
Sequência alternativa	

Tabela 9-5 - Especificação do caso de uso 5.

Caso de Uso 6 – Adicionar elementos ao grupo

Nome	Adicionar elementos ao grupo
Ator	Responsável pela aquisição
Pré-condições	Ligação a uma rede *WiFi*, o projeto tem de estar configurado.
Sequência de Eventos	1. O utilizador seleciona a opção emparelhar por *Bluetooth*. 2. O utilizador seleciona os dispositivos que quer adicionar ao grupo de aquisição. 3. O utilizador define um nome amigável para cada um.
Sequência alternativa	1. O utilizador seleciona a opção emparelhar por código QR. 2. O utilizador espera que todos os dispositivos tenham efetuado a leitura do código. O utilizador define um nome amigável para cada um.

Tabela 9-3 - Especificação do caso de uso 6.

Caso de Uso 7 – Configurar grupo de aquisição

Nome	Configurar grupo de aquisição
Ator	Responsável pela aquisição
Pré-condições	Ligação a uma rede *WiFi, o* grupo de aquisição terá de estar formado
Sequência de Eventos	1. O utilizador seleciona a opção configuração de grupo. 2. O utilizador preenche os campos necessários (nome do grupo e o seu nome) 3. O utilizador define os eventos e os sensores para a aquisição. 4. O utilizador submete as configurações aos constituintes do grupo clicando no botão de *check.*
Sequência alternativa	

Tabela 9-4 - Caso de uso 7.

Caso de Uso 8 – Definir Alarmes

Nome	Definir alarmes
Ator	Responsável pela aquisição
Pré-condições	Ligação a uma rede *WiFi*, o grupo de aquisição terá de estar configurado
Sequência de Eventos	1. O utilizador seleciona a opção de definir alarmes. 2. O utilizador define os valores limite para o qual deverá ser alertado. 3. O utilizador submete as configurações dos alarmes.
Sequência alternativa	

Tabela 9-5 - Especificação do caso de uso 8.

Caso de Uso 9 – Controlar aquisição

Nome	Controlar aquisição
Ator	Responsável pela aquisição
Pré-condições	Ligação a uma rede *WiFi*, o grupo de aquisição terá de estar configurado.
Sequência de Eventos	1. O utilizador clica no botão que simboliza o início da aquisição. 2. O utilizador visualiza os dados recolhidos pelos dispositivos. 3. O utilizador seleciona o botão de paragem de aquisição. 4. O utilizador comanda o *Upload*.
Sequência alternativa	

Tabela 9-6 - Especificação do caso de uso 9.

Caso de Uso 10 – Consultar histórico de estudos

Nome	Consultar histórico de estudos
Ator	Responsável pela aquisição
Pré-condições	O responsável terá de ter realizado pelo menos uma aquisição.
Sequência de Eventos	1. O utilizador abre o menu da aplicação. 2. O utilizador seleciona o item de consultar histório. 3. O utilizador após visualização do histórico de projetos configurados seleciona um. 4. O utilizador após visualização do histórico de grupos configurados seleciona um. 5. O utilizador poderá ver as informações desse projeto e o *log* de interação durante a aquisição.
Sequência alternativa	

Tabela 9-7 - Especificação do caso de uso 10.

Caso de Uso 11 – Consultar histórico de Alarmes

Nome	Consultar histórico de Alarmes
Ator	Responsável pela aquisição
Pré-condições	O responsável terá de ter realizado pelo menos uma aquisição com a ocorrência de alarmes
Sequência de Eventos	1. O utilizador abre o menu da aplicação. 2. O utilizador seleciona a opção de consultar histórico de alarmes. 3. O utilizador seleciona um dos participantes disponíveis. 4. O utilizador verifica os alarmes desse participante.
Sequência alternativa	

Tabela 9-8- Especificação do caso de uso 11.

Caso de Uso 12 – Consultar histórico de Eventos

Nome	Consultar histórico de Eventos
Ator	Responsável pela aquisição
Pré-condições	O responsável terá de ter realizado pelo menos uma aquisição e marcado 1 evento.
Sequência de Eventos	1. O utilizador abre o menu da aplicação. 2. O utilizador seleciona a opção de consultar histórico de eventos. 3. O utilizador verifica o histórico de eventos já marcados em cada grupo de aquisição.
Sequência alternativa	

Tabela 9-9 - Especificação caso de uso 12.

Caso de Uso 13 – Registo na Web App

Nome	Registo na Web App
Ator	Responsável pela aquisição
Pré-condições	O responsável terá de ter acesso à internet
Sequência de Eventos	1. O utilizador abre o browser no computador no endereço http://deti-aulas.ua.pt:8789 2. O utilizador escreve os dados de registo
Sequência alternativa	

Tabela 9-10 - Especificação caso de uso 13.

Caso de Uso 14 – Login na Web App

Nome	Login na Web App
Ator	Responsável pela aquisição
Pré-condições	O responsável terá de ter realizado o registo na *Web App*.
Sequência de Eventos	1. O utilizador abre o browser no computador no endereço http://deti-aulas.ua.pt:8789 2. O utilizador insere os dados de acesso
Sequência alternativa	

Tabela 9-11 - Especificação do caso de uso 14.

Caso de Uso 15 – Exportar ficheiros de aquisição

Nome	Exportar ficheiros de aquisição
Ator	Responsável pela aquisição
Pré-condições	O responsável terá de ter realizado pelo menos uma aquisição.
Sequência de Eventos	1. O utilizador abre o browser no computador no endereço http://deti-aulas.ua.pt:8789 2. O utilizador verifica a lista de grupos sobre o qual é possível realizar *download* dos ficheiros. 3. O utilizador pressiona o botão de *download* associado ao grupo que quer transferir os dados.
Sequência alternativa	

Tabela 9-12 - Especificação caso de uso 15.

Anexo 2 – Poster

O projeto desenvolvido nesta dissertação foi apresentado no evento público *Students@Deti*[25], organizado pelo Departamento de Eletrónica Telecomunicações e Informática. De modo a apresentar o projeto, foi criado o poster visível na figura seguinte.

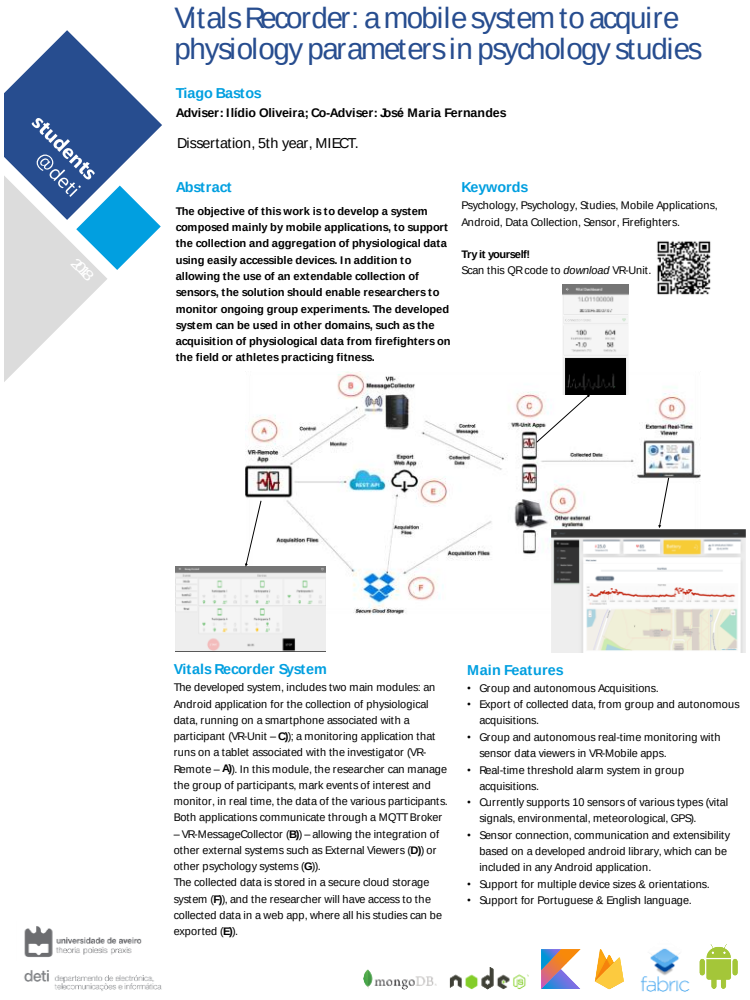


Figura 10-1 - Poster apresentado no evento Students@Deti

[25] http://studentsandteachersdeti.web.ua.pt

Anexo 3 – Questionário PSSUQ

1. **Em geral, estou satisfeito com a facilidade de utilização deste sistema.** *
Mark only one oval.

	1	2	3	4	5	6	7	
Concordo Totalmente	◯	◯	◯	◯	◯	◯	◯	Discordo totalmente

2. **Este sistema foi simples de utilizar.** *
Mark only one oval.

	1	2	3	4	5	6	7	
Concordo Totalmente	◯	◯	◯	◯	◯	◯	◯	Discordo totalmente

3. **Consegui completar as tarefas e os cenários utilizando este sistema.** *
Mark only one oval.

	1	2	3	4	5	6	7	
Concordo Totalmente	◯	◯	◯	◯	◯	◯	◯	Discordo totalmente

4. **Consegui completar rapidamente as tarefas e os cenários utilizando este sistema.** *
Mark only one oval.

	1	2	3	4	5	6	7	
Concordo Totalmente	◯	◯	◯	◯	◯	◯	◯	Discordo totalmente

5. **Consegui completar as tarefas e os cenários com eficiência utilizando este sistema.**
Mark only one oval.

	1	2	3	4	5	6	7	
Concordo totalmente	◯	◯	◯	◯	◯	◯	◯	Discordo totalmente

6. **Senti-me confortável a utilizar este sistema.** *
Mark only one oval.

	1	2	3	4	5	6	7	
Concordo Totalmente	◯	◯	◯	◯	◯	◯	◯	Discordo totalmente

Figura 11-1 - Questionário PSSUQ parte 1

7. **Foi fácil aprender a utilizar este sistema.** *
Mark only one oval.

	1	2	3	4	5	6	7	
Concordo Totalmente	◯	◯	◯	◯	◯	◯	◯	Discordo totalmente

8. **Acredito que me tornaria rapidamente produtivo se utilizasse este sistema.**
Mark only one oval.

	1	2	3	4	5	6	7	
Concordo totalmente	◯	◯	◯	◯	◯	◯	◯	Discordo totalmente

9. **O sistema deu mensagens de erros que me indicaram claramente como resolver os problemas.** *
Mark only one oval.

	1	2	3	4	5	6	7	
Concordo Totalmente	◯	◯	◯	◯	◯	◯	◯	Discordo totalmente

10. **Sempre que cometi um erro durante a utilização do sistema, consegui recuperar de forma fácil e rápida.** *
Mark only one oval.

	1	2	3	4	5	6	7	
Concordo Totalmente	◯	◯	◯	◯	◯	◯	◯	Discordo totalmente

11. **A informação fornecida pelo sistema (como ajuda online , mensagens noecrã ou outra documentação) foi clara.**
Mark only one oval.

	1	2	3	4	5	6	7	
Concordo totalmente	◯	◯	◯	◯	◯	◯	◯	discordo totalmente

12. **Foi fácil de encontrar a informação que precisava.** *
Mark only one oval.

	1	2	3	4	5	6	7	
Concordo Totalmente	◯	◯	◯	◯	◯	◯	◯	Discordo totalmente

Figura 11-2 - Questionário PSSUQ parte 2

13. **A informação fornecida pelo sistema foi fácil de entender.** *
Mark only one oval.

	1	2	3	4	5	6	7	
Concordo Totalmente	◯	◯	◯	◯	◯	◯	◯	Discordo totalmente

14. **A informação foi eficaz para me ajudar a completar as tarefas e os cenários.** *
Mark only one oval.

	1	2	3	4	5	6	7	
Concordo Totalmente	◯	◯	◯	◯	◯	◯	◯	Discordo totalmente

15. **A organização da informação que o sistema transmitiu foi clara**
Mark only one oval.

	1	2	3	4	5	6	7	
Concordo Totalmente	◯	◯	◯	◯	◯	◯	◯	Discordo totalmente

16. **A interface do sistema foi agradável.** *
Mark only one oval.

	1	2	3	4	5	6	7	
Concordo Totalmente	◯	◯	◯	◯	◯	◯	◯	Discordo totalmente

17. **Gostei de utilizar a interface deste sistema.** *
Mark only one oval.

	1	2	3	4	5	6	7	
Concordo Totalmente	◯	◯	◯	◯	◯	◯	◯	Discordo totalmente

18. **Este sistema tem todas as funcionalidades e capacidades que eu esperava.**
Mark only one oval.

	1	2	3	4	5	6	7	
Concordo totalmente	◯	◯	◯	◯	◯	◯	◯	Discordo totalmente

19. **Em geral, estou satisfeito com este sistema.** *
Mark only one oval.

	1	2	3	4	5	6	7	
Concordo Totalmente	◯	◯	◯	◯	◯	◯	◯	Discordo totalmente

Figura 11-3 - Questionário PSSUQ parte 3