Mestrado em Engenharia Informática

Relatório de Estágio

Monitorização em tempo-real para ambientes PPDR

Pedro Lopes Gonçalves

plgonc@student.dei.uc.pt

Orientador DEI:

Prof. Fernando Boavida

Orientador BlueTalent:

Eng. Luís Cordeiro

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Monitorização em tempo-real para ambientes PPDR

Relatório submetido no âmbito do estágio curricular do Mestrado em Engenharia Informática

2014/2015

Autor: Pedro Lopes Gonçalves

Orientador DEI: Professor Doutor Fernando Boavida
Orientador BlueTalent: Engenheiro Luís Cordeiro

Júri Arguente: Professor Doutor Edmundo Monteiro
Júri Vogal: Professor Doutor Hugo Gonçalo Oliveira

Resumo

A expressão PPDR (*Public Protection and Disaster Relief*) inclui qualquer atividade de proteção pública desde bombeiros a apagar fogos a polícias a perseguir criminosos e profissionais de saúde a providenciar assistência médica.

É no contexto dos ambientes referidos que surge o projeto descrito no presente documento e que foi levado a cabo no âmbito do estágio curricular do Mestrado em Engenharia Informática da FCTUC, durante o ano lectivo de 2014/15. O objectivo central deste trabalho é criar um sistema que permita realizar a monitorização em tempo-real de sinais vitais dos intervenientes envolvidos em contextos PPDR. Uma vez que se trata de um sistema inovador, as componentes de investigação, especificação e implementação estão bem vincadas ao longo deste projeto.

O trabalho apresentado neste relatório incide particularmente sobre quatro pilares essenciais da Engenharia de Software: definição de requisitos, especificação da arquitetura, implementação e testes de sistema. O estágio focou-se fundamentalmente nas questões relativas ao software implementado, considerando, contudo, alguns aspetos técnicos relacionados com hardware que são aplicados no desenvolvimento do sistema.

Palavras-Chave: *Control Center*, *Server*, *Device*, *PPDR*, Monitorização, Sinal Vital

Abstract

PPDR refers to any public protection activity, such as firemen putting out fires, policemen chasing criminal or even health professionals providing medical assistance.

The project presented in this document emerges in the context of the mentioned environments and was inserted in the Masters Degree curricular internship during the academic year of 2014/2015. The main goal of this project is to create a system that allows real-time monitoring of vital signs of the patients involved in PPDR environments. Since it is an innovator task, the research, specification and implementation components are very noticeable throughout the development of this work.

The project presented in this document is mostly about four essential subjects of Software Engineering: requirement definition, architecture specification, implementation and system test. The curricular internship focused mainly in issues surrounding the implemented software, considering, however, some technical aspects related to hardware, which are applied in the development of the system.

Keywords: Control Center, Server, Device, PPDR, Monitoring, Vital Sign

Índice

Índice de Figuras

Índice de Tabelas

Glossário

BITalino – Plataforma de *hardware* usada para a aquisição de sinais vitais e a sua transmissão em tempo-real. É fácil de usar, versátil e escalável e existe em três versões: *free style*, onde todos os blocos de *hardware* (cada sensor) estão separados providenciando uma maior customização; *plugged*, onde os sensores contêm uma ligação *RJ22*; *board*, onde o BITalino é apresentado como uma placa tudo-em-um. Todos possuem sensores para medir sinais do coração, músculos, sistema nervoso, movimento e luz ambiente. Inclui também um microcontrolador, *Bluetooth* e um módulo de carregamento.

SALUS Message Broker – Servidor de mensagens utilizado no seio do projeto SALUS. À altura da elaboração deste trabalho de estágio, este servidor tinha já uma biblioteca para aplicações externas poderem utilizá-lo.

Critical Communications World – Evento internacional que junta vários oradores que falam sobre temas relativos às comunicações em situações críticas, focando principalmente os utilizadores da rede móvel neste tipo de comunicações e os utilizadores da tecnologia TETRA. Existem também vários *stands* com várias empresas onde são mostrados os produtos mais inovadores na área (redes, terminais, aplicações, sistemas, etc.).

Alcatel-Lucent – Empresa francesa de equipamentos de telecomunicações. Oferece vários tipos de *hardware* de rede fixa e móvel, tecnologias IP, *software* e serviços. A *Alcatel-Lucent* resulta de uma fusão entre duas empresas: a *Alcatel* e a *Lucent-Techonologies*. Esta fusão ocorreu em 2006. Em Abril de 2015, esta empresa foi adquirida pela Nokia por 15.6 biliões de dólares.

Acrónimos

PPDR – *Public Protection and Disaster Relief*

TETRA – *Terrestrial Trunked Radio*

LTE – *Long-Term Evolution*

RMI – *Remote Method Invocation*

CORBA – *Common Object Request Broker Architecture*

NIO – *Non-blocking Input/Output*

TLV – *Type, Lenght Value*

ECG – *Electrocardiogram*

HR – *Heart Rate*

RR – *Respiration Rate*

TEMP – *Temperature*

Capítulo 1 – Introdução

Com vista à obtenção do grau de Mestre em Engenharia Informática pela Faculdade de Ciências e Tecnologias da Universidade de Coimbra, surge um estágio na empresa BlueTalent Lda., na área de saúde e com a duração de um ano letivo. Este estágio foi executado sob orientação do Engenheiro Luís Cordeiro por parte da empresa. O orientador indicado pelo Departamento de Engenharia Informática (DEI) foi o Professor Doutor Fernando Boavida.

Este documento serve para apresentar e descrever o trabalho realizado, desde a análise inicial até à fase de testes e validação do sistema.

Neste capítulo pretende-se contextualizar o estágio e apresentar os objetivos gerais que serviram de ponto de partida.

1.1. Contexto

PPDR (*Public Protection and Disaster Relief)* refere-se a qualquer atividade de proteção pública como, por exemplo, bombeiros a apagar fogos, polícias a perseguir criminosos ou médicos a controlar algum tipo de doença. Neste tipo de situações, é necessário que as comunicações usadas entre os intervenientes sejam de segurança extrema e completamente dedicadas a este tipo de cenários. Atualmente, este tipo de comunicações é feito através da utilização, maioritariamente, da tecnologia TETRA (*Terrestrial Trunked Radio*) desenhada especificamente para agências governamentais, serviços de emergência, forças policias, quartéis de bombeiros, serviços de transporte e em ações militares, e que apenas permitem transmitir voz e dados (pequenas mensagens). Um dos principais desafios neste campo é usar as redes IP, como o 3G ou 4G, disponíveis de forma pública, e conseguir nestas redes um nível de segurança equivalente ao TETRA. Outro aspecto importante é o facto de em redes IP ser possível a transmissão de vídeo. [1]

Este trabalho insere-se num projeto europeu, apoiado pelo fundo da Comissão Europeia, intitulado *SALUS*, que tem como principal objetivo a implementação e avaliação de sistemas PPDR de última geração, tendo em foco aspetos tanto tecnológicos, como económicos de forma a oferecer aos utilizadores PPDR uma solução eficiente em termos de custo e operação, criando assim um modelo de negócio sustentável para a indústria e os operadores da área.

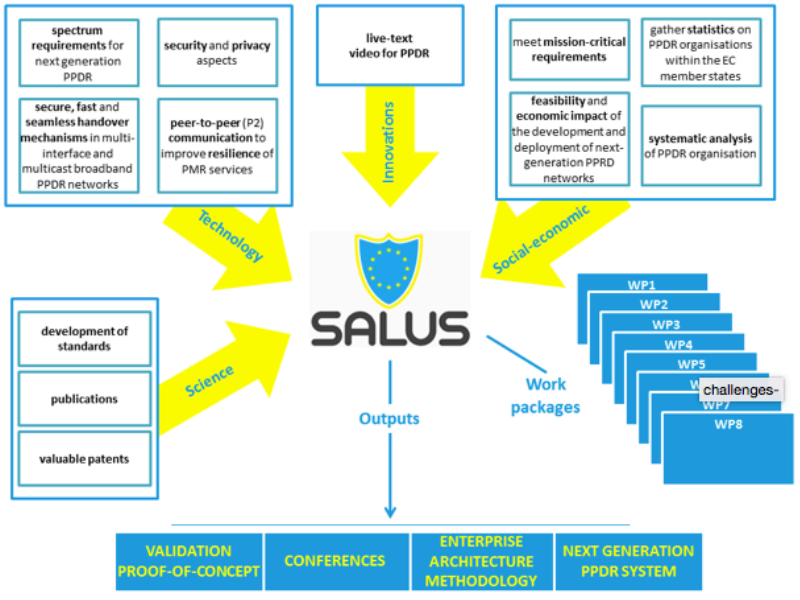

Figura 1. Projeto SALUS (retirada de www.sec-salus.eu)

Na Figura 1 está exemplificado um esquema relativo à forma como o *SALUS* pretende alcançar os objetivos a que se propõe.

No seio do projeto SALUS, a BlueTalent tem como principal papel contribuir com a experiência em segurança, comunicação e gestão de redes e serviços, incluindo aplicações móveis, gestão de recursos, redes de sensores e monitorização de redes e serviços. Esta experiência prévia foi construída, em parte, noutros projetos de investigação do mesmo género. Contribui também na integração de sistemas e no desenvolvimento das componentes da plataforma *SALUS*.

O trabalho que este relatório apresenta é parte da contribuição da BlueTalent no projeto SALUS, focando a contribuição do estagiário para o mesmo.

1.2. Objetivos

O principal objetivo deste trabalho da BlueTalent para o projeto SALUS é o de desenhar e implementar um sistema de monitorização de sinais vitais em tempo-real. Mais em concreto, vários atores, em ambientes PPDR, irão transportar consigo um conjunto de sensores incorporados num único dispositivo e medirão um determinado número de sinais vitais. Esses sinais vão ser monitorizados em vários postos de controlo que poderão visualizar a informação para várias pessoas ao mesmo tempo ou para uma pessoa com maior detalhe. Na Figura 2 é apresentado um esquema simples que representa este objetivo e demonstra as componentes deste projeto.

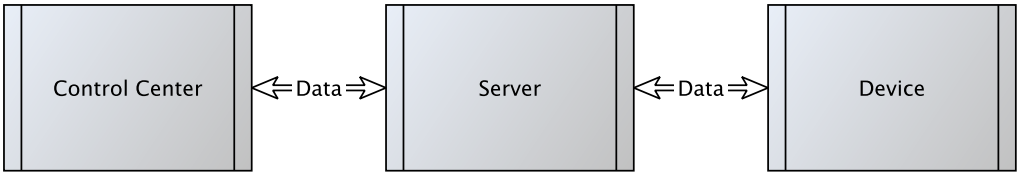

Figura 2. Esquema Global

Sendo este projeto de dimensão considerável, foi necessário definir e esclarecer quais as componentes que estariam no escopo do estágio curricular e que iriam ser alvo de estudo, definição e implementação e onde o estagiário iria ser um dos intervenientes.

Device	**No âmbito?**	**Observações**
Hardware a ser utilizado (single board PC, dispositivo de sensores e acessório *wearable*)	Não	Decisão interna do projeto SALUS e da empresa
Aspetos técnicos relativos aos sinais vitais a serem medidos	Não	Fazem parte do projeto parceiros da área de saúde e toda a informação técnica necessária relativa a sensores e sinais vitais será fornecida pela empresa
Tratamento dos dados vindos dos sensores	Sim	Identificação e envio de alertas caso os dados demonstrem algum problema
Mecanismos para o envio dos dados para o Servidor	Sim	-

Tabela 1. Âmbito – *Device*

Server	No âmbito?	Observações
Mecanismos de recepção, tratamento e envio dos dados vindos dos dispositivos	Sim	-
Armazenamento dos dados para histórico	Sim	-
Definição de infraestrutura e hardware	Não	Todas as decisões sobre hardware do lado do servidor são tomadas no seio do projeto.

Tabela 2. Âmbito – *Server*

Control Center	No âmbito?	Observações
Visualização dos dados em tempo-real	Sim	-
Definição de hardware	Não	Todas as decisões sobre hardware são tomadas no seio do projeto.

Tabela 3. Âmbito – *Control Center*

Outros Aspetos	No âmbito?	Observações
Funcionamento em redes 4G, LTE e Wi-Fi	Sim	O sistema terá de ser preparado para funcionar nestas redes e obedecer a requisitos de segurança que serão definidos. No entanto, o estudo destas tecnologias não é do âmbito do estágio.
Integração com projeto SALUS (Message Broker)	Sim	É necessária a integração com um servidor de mensagens do projeto SALUS intitulado Message Broker.

Tabela 4. Âmbito - Outros

Tendo isto em conta, o principal objetivo deste trabalho é o desenho e implementação de um sistema de monitorização de sinais vitais e a integração com um serviço de captação e transmissão de sinais de voz e vídeo em tempo-real já implementado. Pretende-se que este sistema seja usado em ambientes de PPDR (*Public Protection and Disaster Relief*), isto é, em cenários de emergência. Como tal, o sistema irá ser composto por duas principais componentes: um dispositivo com vários sensores ligados a uma pessoa (utilizador), que terá a função de captar os sinais vitais e enviá-los para um pequeno computador onde este dispositivo se conecta e onde se procederá ao envio dos dados para vários centros de controlo para estes serem analisados; os centros de controlo, onde será feita a monitorização em tempo-real de todos os dados de todos os utilizadores conectados e ativos, isto é, que estejam efetivamente a enviar dados. Será possível visualizar dados em detalhe para um único paciente ou monitorizar vários pacientes ao mesmo tempo, onde os dados terão forçosamente de ser mais resumidos.

O dispositivo será incorporado num acessório que contém uma câmara e um microfone para captação de sinais de áudio e vídeo. O mecanismo de transmissão de voz e vídeo já se

encontra implementado e o objetivo passa por integrá-lo com o sistema de monitorização de sinais vitais a implementar e descrito neste relatório.

Os sinais vitais a monitorizar pelo sistema descrito são:

- Atividade elétrica gerada pelo coração – a monitorização deste sinal corresponde ao Eletrocardiograma (ECG) que se refere a um exame de saúde da área de cardiologia.
- Taxa de respiração – refere-se ao número de respirações (ciclos de inspiração e expiração) que um individuo efetua por minuto.
- Frequência cardíaca – quantidade de vezes que o coração bate por minuto.
- Temperatura – temperatura corporal expressa em graus centígrados.

São ainda monitorizados os sensores acelerómetro que indica a aceleração própria da pessoa e o sensor de GPS, que devolve a posição do individuo. Todos estes sensores têm de ser compostos num único dispositivo que será incorporado numa *backpack*.

1.3. Metodologia

Tendo como base o contexto descrito na secção anterior, optou-se, enquanto modelo de trabalho, por uma metodologia em cascata. Neste tipo de metodologia percorre-se um conjunto de etapas bem definidas e no final de cada uma das fases é alcançado um resultado que pode ser exposto sob forma de documento, protótipo de software ou como um produto, considerado final. Cada um dos resultados tem de ser revisto, avaliado e o produto alterado caso seja necessário.

A Figura 3 ilustra o esquema com as etapas definidas para este projeto:

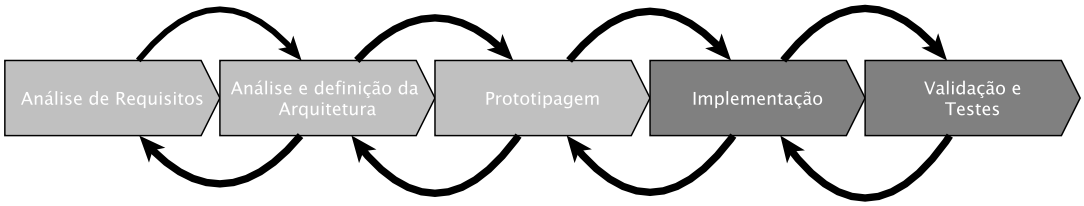

Figura 3. Metodologia utilizada no desenvolvimento do projeto

Como é possível observar, todo o processo se inicia com uma análise detalhada dos requisitos, definindo-se a prioridade de cada um deles. Quando completada, prossegue-se para uma análise dos elementos que têm de constar da arquitetura do sistema e faz-se a definição dessa mesma arquitetura. Tratando-se de um projeto com uma vertente forte de investigação, estas duas fases revestem-se de um constante debate de ideias e definição de soluções para os problemas que vão surgindo. Embora não se possa considerar que estas duas primeiras fases estão claramente fechadas rumo a um produto final, é possível definir algo passível de sustentar um protótipo que represente um exemplo das funcionalidades pretendidas.

Após as duas primeiras fases, surge então a prototipagem onde se define um modelo que simula todas as funcionalidades de um produto final, respondendo aos requisitos identificados anteriormente. Depois disto, segue-se a implementação do protótipo em causa. Após a implementação ter sido considerada concluída, entramos na fase de validação e testes. Como apresentado, em todas as fases é possível o recuo à fase anterior quando nos deparamos com testes falhados, validações não concluídas ou qualquer alteração necessária de outra ordem. É de esperar que o "ciclo" que engloba as duas últimas fases se repita mais do que uma vez até ser

alcançado um produto final. É importante referir também que todo este processo é acompanhado de reuniões semanais onde se discutem as questões e desafios que vão surgindo.

1.4. Planeamento

O estágio curricular onde o trabalho descrito neste relatório se insere tem a duração de um ano letivo, com início no dia 15 de Setembro de 2014 e fim no dia 26 de Junho de 2015. Antes da apresentação final do relatório deste estágio, foi requerida a entrega de uma versão intermédia do mesmo, objetivada para o dia 27 de Janeiro de 2015. Como se pode observar pela Figura 4, este trabalho iniciou-se com uma semana e meia reservada para a ambientação ao tema do trabalho seguindo-se uma fase de análise dos requisitos do sistema a implementar. Após serem definidos os requisitos, foi feita uma análise e definição da arquitetura necessária para o problema em questão, que compreendeu o período entre meados do mês de Outubro e inicio do mês de Dezembro, tendo sido reservado cerca de oito semanas, por ser uma fase revestida de momentos de discussão de diferentes possibilidades e procura do melhor caminho a tomar. Seguiu-se a fase da prototipagem inicial, onde o intento foi construir um protótipo a nível de design e de funcionalidades básicas requeridas. Esta fase decorreu em paralelo com a anterior por ser possível conciliar a prototipagem com as decisões relativas à arquitetura. Depois do protótipo definido foi necessária a implementação final desse protótipo com todas as funcionalidades necessárias. Esta fase tem a duração de três meses, compreendidos entre o fim de Novembro de 2014 e meados de Março de 2015, existindo um trabalho paralelo de preparação e escrita de documentação detalhada como, por exemplo, um relatório de estágio intermédio. Dando por concluída a implementação do sistema, este foi submetido a uma demonstração no âmbito do projeto SALUS. Outras demonstrações deste género foram ocorrendo mas a que foi indicada no planeamento foi a demonstração definida como sendo a principal e a mais importante. Na sequência dessa demonstração foi necessária uma revisão de todas as alterações que se teriam de fazer. Este estádio compreende o período entre o fim de Março e o inicio de Maio. Este tempo é também reservado para melhoramentos que poderão ser necessários e que são decorrentes do resultado dos testes e validações.

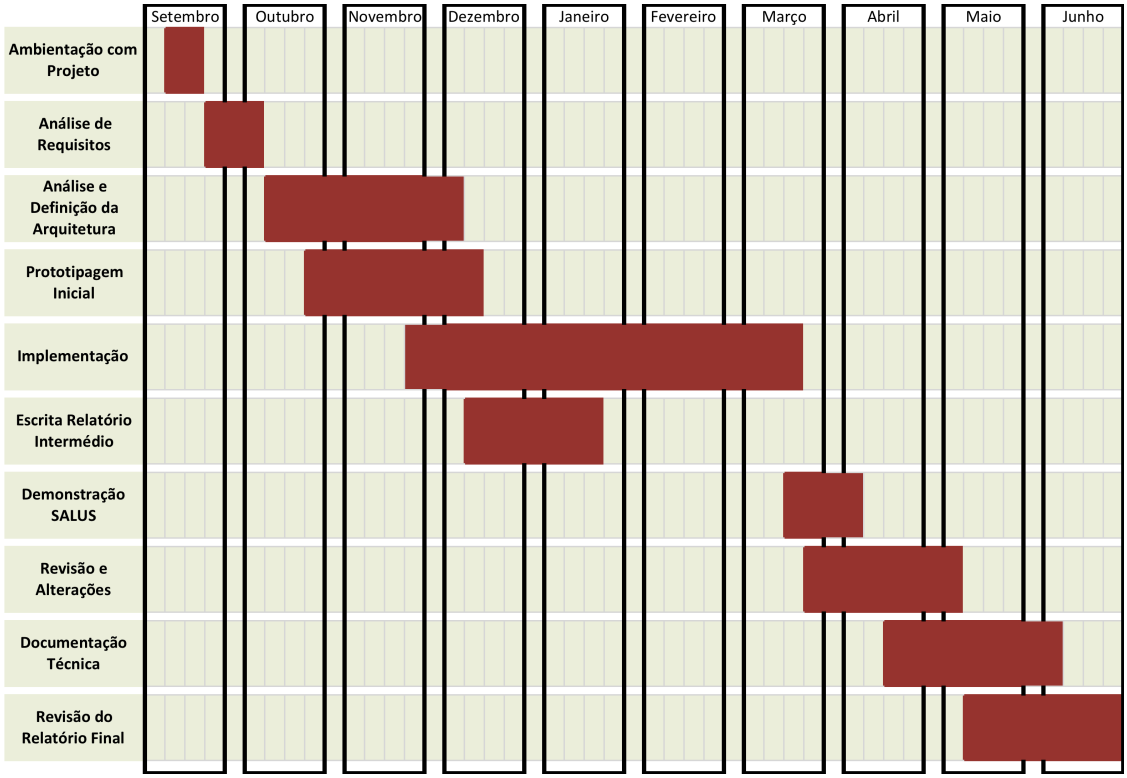

Figura 4. Planeamento do Ano Letivo

1.5. Trabalho Realizado

Esta secção serve para detalhar qual foi o trabalho realizado pelo estagiário ao longo do ano e quais as tarefas que desenvolveu tanto a nível individual mas também quando foi inserido numa equipa de trabalho responsável pelo projeto.

Numa primeira fase do projeto (durante a ambientação ao tema, análise de requisitos, definição da arquitetura e prototipagem), o estagiário trabalhou de forma autónoma. Como o âmbito do projeto tem uma componente de inovação bastante acentuada e como o ponto de partida eram apenas os objetivos definidos, foi necessário fazer um trabalho de investigação prévio como a análise de requisitos e um estudo com o intento de saber quais as melhores tecnologias a utilizar para implementar o sistema. Como exemplo, foi necessário definir quais as *frameworks* de desenvolvimento a utilizar e que poderiam responder melhor aos requisitos definidos. Depois disto foi apresentada pelo estagiário uma primeira análise e definição de uma arquitetura para o sistema (incluindo, por exemplo, um primeiro modelo da base de dados) e feita a prototipagem inicial.

Depois desta fase, o estagiário foi inserido num grupo de trabalho composto por mais quatro pessoas: João Gonçalves, Pedro Borges, Hugo Fonseca e Patrício Baptista. Esta equipa foi coordenada e gerida pelo Engenheiro Luís Cordeiro.

É apresentado de seguida um diagrama (Figura 5) que demonstra as várias fases do projeto, referindo quem foram os membros da equipa que participaram em cada uma delas, focando o trabalho que foi realizado pelo estagiário.

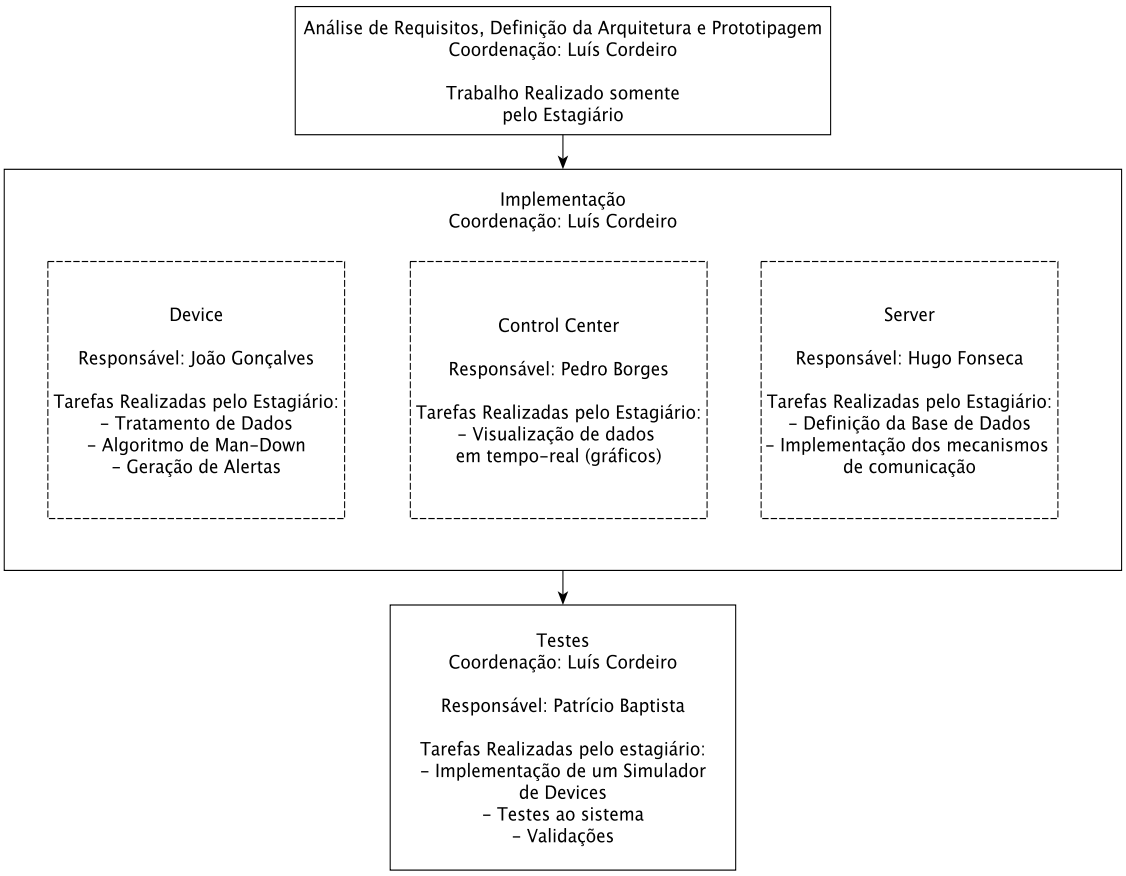

Figura 5. Trabalho realizado pelo estagiário

1.6. Estrutura do Documento

Este documento está estruturado em capítulos, secções e subsecções. Os capítulos são os elencados de seguida:

- **Capítulo 1.** Introdução: capítulo atual que introduz o documento, abordando a instituição e o estágio, apresentando o contexto, metodologia e planeamento.
- **Capítulo 2.** Estado de Arte: apresenta os resultados da análise ao estado de arte onde é esmiuçado o mais alto nível de desenvolvimento dos sistemas de monitorização de sinais vitais e onde são exploradas tecnologias passíveis de serem usadas no desenvolvimento do sistema.
- **Capítulo 3.** Requisitos: apresenta três cenários onde o sistema pode ser usado e revela os requisitos (funcionais e não-funcionais) do sistema, ou seja, é feito o levantamento e análise de requisitos.
- **Capítulo 4.** Arquitetura e especificação: apresenta com detalhe a solução a implementar, descrevendo a arquitetura e os vários módulos da aplicação. Aborda os métodos de comunicação, estrutura das mensagens e *frameworks* e tecnologias utilizadas.
- **Capítulo 5.** Implementação: descreve em detalhe todos os passos do desenvolvimento da aplicação, explicando o que foi feito em cada componente do sistema, com especial foco para os segmentos onde o trabalho do estagiário teve mais influência.
- **Capítulo 6.** Verificação e Validação: detalha como foi feito o processo de validação e descreve os testes realizados ao sistema.
- **Capítulo 7.** Conclusão: capítulo para apresentação das conclusões relativas a este trabalho e apresenta os possíveis passos futuros no projeto.
- **Capítulo 8.** Referências: apresenta todas as referências usadas para apresentação de resultados ou descrições.

O documento contém também um conjunto de anexos que contém informação diversa sobre o trabalho realizado:

- **Anexo A –** Modelo de Mensagens TLV
- **Anexo B –** Modelo de Mensagens *Protobuffer*
- **Anexo C –** Testes de Sistema

Capítulo 2 – Estado da Arte

Neste capítulo é apresentado um estado da arte acerca de sistemas de monitorização de sinais vitais. É feito também um estudo acerca das tecnologias de maior relevo passíveis de serem utilizadas ao longo do trabalho.

O termo *Mobile Health* (mHealth) refere-se à prática da medicina e proteção da saúde pública usando dispositivos móveis. Os sistemas e aplicações mHealth baseiam-se no uso de dispositivos móveis para recolha de dados clínicos e para um possível envio posterior para equipas médicas ou de saúde, para análise em tempo-real de sinais vitais dos utilizadores das aplicações ou sistemas. Também é possível, em certos casos, proceder a um tratamento ou ajuste imediatamente após a leitura dos dados. Com estes sistemas, além de ser possível reduzir os riscos de complicações de saúde para os pacientes, tornam-se as respostas médicas aos eventuais problemas que surjam muito mais rápidas e eficazes. [2]

Na primeira secção, são apresentados sistemas de monitorização de sinais vitais capazes de envolver equipas médicas e pacientes, em larga escala. São também apresentadas outras aplicações semelhantes para dispositivos móveis, sendo estas bem mais simples e de uso mais pessoal e personalizado. Nas secções seguintes é apresentado um estado da arte sobre tecnologias passíveis de ser usadas na implementação do sistema que é proposto.

Não são estudados outros aspetos relativos ao desenvolvimento do sistema por estes não fazerem parte do âmbito do estágio curricular. De seguida, são apresentados alguns exemplos de estudos fora do âmbito do estágio:

- Estudo de características técnicas relativas a cada sinal vital: é importante saber algo sobre os sinais vitais (como, por exemplo, a frequência ideal de amostragem) mas não é do âmbito o estudo aprofundado destes.
- Hardware passível de ser utilizado pelo dispositivo: decisão interna do projeto e da empresa que não se encontra no âmbito do estágio. O estagiário tem acesso a um dispositivo chamado *BITalino* (e suas API's) que lê os dados dos sensores e que se encontra conectado a um mini computador. Todo este hardware está inserido numa mochila. A câmara e o microfone são incorporados num par de óculos. [3]

2.1. Sistemas de monitorização de sinais vitais

Nesta secção são apresentados vários sistemas e plataformas de monitorização de sinais vitais. São analisados sistemas comparáveis com o apresentado neste relatório, mas é feito também um estudo relativo a aplicações especificas para *smartphones* e *tablets*, capazes de executar a monitorização de bio-sinais, embora em menor escala ou que contêm limitações quando comparadas com as primeiras.

2.1.1. Sistemas de monitorização de sinais vitais.

Para a descrição e análise destes sistemas de monitorização de sinais vitais, o critério utilizado foi o de identificar os sistemas mais conhecidos a nível mundial e que melhor se possam comparar com o sistema descrito neste relatório. Para cada um dos sistemas de monitorização de sinais vitais listados, é feita uma análise das respectivas vantagens e eventuais desvantagens provenientes da sua utilização.

2.1.1.1. Visi Mobile (Sotera Wireless Inc.)

Esta plataforma de monitorização de sinais vitais, foi desenhada para manter o pessoal clínico mais próximo dos pacientes, quer na cama do hospital, quer nas ambulâncias.

Este sistema é composto por um dispositivo que se coloca no pulso e que contém um pequeno ecrã onde são visualizados os valores dos sinais vitais (ECG, pulsações, nível de

saturação de oxigénio, pressão arterial, temperatura da pele). Também fazem parte do sistema todos os sensores que permitem medir estes sinais. São utilizados computadores ou *tablets* para visualizar remotamente os dados do dispositivo e para receber todo o tipo de notificações/alertas que possam surgir. [4]

Esta plataforma serve-se das infra-estruturas de rede Wi-fi hospitalares já existentes e é capaz de apresentar a informação em formato digital ou imprimi-la em papel.

No futuro, o Visi Mobile pretende incluir também a postura e o movimento como novos "sinais vitais" a serem detectados.

Figura 6. Visi Mobile (retirada de www.visimobile.com/overview)

Vantagens: visualização remota em tempo-real dos dados de um paciente;

Desvantagens: funciona apenas em redes wi-fi hospitalares.

2.1.1.2. CareLink System (Medtronic)

Este sistema visa ajudar pessoas com doenças crónicas como a diabetes ou doenças de coração.

O CareLink System tem duas componentes:

1. OptiVol Monitoring: é colocado nos dispositivos electrónicos que ajudam o coração humano a manter-se em normal funcionamento e serve para detectar a formação de fluídos que possam provocar doenças de coração. O médico tem acesso a um relatório derivado deste dispositivo, podendo desta forma alterar a dieta do paciente ou mesmo a medicação;

2. Diabetes Management: é o projeto futuro deste sistema e visa permitir que a informação acerca dos níveis de glicose dos pacientes seja enviada em tempo-real, para que possa ajudar a prever alguma complicação que possa advir de níveis de glicose excessivamente altos ou baixos. Este projeto ainda se encontra num estado de desenvolvimento muito embrionário, não tendo o seu sucesso ou risco sido testado; [5]

Vantagens: monitoriza o coração de forma detalhada;

Desvantagens: limitado em quantidade de dados que monitoriza; não permite monitorização fora do hospital e em tempo-real;

2.1.1.3. Telcare

O sistema Telcare permite monitorizar o registo de todas as leituras de glicose para doentes com diabetes. Desta forma, é possível dar uma resposta imediata, a qualquer hora, após cada leitura do nível de açúcar no sangue, sem ter de recorrer a leituras periódicas manuais dos registos. Esta gestão da condição do indivíduo em tempo real é feita com recurso a gráficos e tabelas de fácil compreensão. [6]

O Telcare intitula-se como a primeira solução do género que é compatível com redes móveis, de forma a conectar todos os intervenientes que podem gerir a condição de uma pessoa: profissionais de saúde, recursos para ensinar os melhores comportamentos aos pacientes e a rede de contactos de família e amigos da própria pessoa.

Para alcançar o objetivo proposto, o Telcare envia mensagens e lembretes ao paciente com diretrizes personalizadas sobre, por exemplo, a sua alimentação, e mantém os profissionais de saúde envolvidos com atualizações periódicas sobre o estado de saúde do utilizador. Se o paciente tiver algum familiar que seja também médico ou assistente de saúde, pode também partilhar com esse familiar dados críticos sobre o seu estado de saúde. Tudo isto pode ser gerido pelo próprio dispositivo sem ser necessário nenhum smartphone ou hardware adicional.

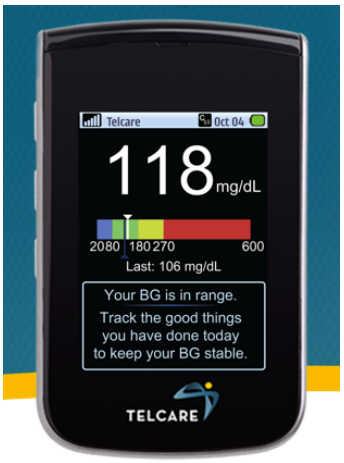

Figura 7. Telcare (retirada de www.telcare.com)

2.1.1.4. 2net (Qualcomm Life)

A plataforma 2net é um sistema baseado na cloud e que tem como objetivo a interoperabilidade de diferentes aplicações e dispositivos médicos, permitindo uma comunicação sem fios, ponto-a-ponto que possibilita o fácil acesso a dados biométricos por parte, tanto dos utilizadores como dos médicos ou assistentes de saúde. Existem quatro possíveis formas de se estabelecer esta comunicação: um dispositivo externo, o 2net Hub, que permite a comunicação máquina-a-máquina entre dispositivos médicos tanto dentro como fora de casa; um módulo de software que conecta os diferentes dispositivos móveis, o 2net Mobile Core, que permite que tanto smartphones como tablets e outros dispositivos similares sirvam de gateway (ou router) para a plataforma 2net. Esta solução permite que os dados biométricos recolhidos sejam visualizados nos smartphones ou tablets e que sejam transmitidos de forma segura via WWAN (wireless wide area network) ou Wi-Fi para o Data Center da plataforma 2net. O 2net permite então a conexão entre uma variedade de sensores biométricos e de dispositivos móveis; dispositivos médicos que tenham um sistema de comunicação móvel incorporado; integração de serviços entre a plataforma 2net e uma determinada plataforma com uma determinada tecnologia usando APIs (Application Programming Interfaces) especificas. [7]

Figura 8. 2net (retirada de www.qualcommlife.com/wireless-health)

Vantagens: sistema baseado na *cloud,* que possibilita um acesso rápido e fácil aos dados*;*

Desvantagens: não opera sobre redes 3G ou 4G; necessário dispositivo externo para poder enviar os dados.

2.1.1.5. AirStrip ONE (AirStrip Technologies)

O AirStrip ONE é uma solução de mobilidade clinica que permite que todo o pessoal de saúde aceda à informação de um paciente em qualquer altura e em qualquer lugar. Possibilita a combinação de fontes de dados que se encontram separadas, permitindo que esses dados sejam visualizados num único software. Este sistema permite visualizar sinais vitais na forma de ondas muito próximas do tempo-real, EMR (Electronic Medical Record) bem como informação do paciente proveniente de diferentes fontes, fornecendo uma comunicação segura para o efeito. Este sistema é constituído por: Core Mobility Platform, a plataforma que permite a conexão dos dados vitais clínicos provenientes de diferentes fontes e os diferentes dispositivos das diferentes marcas; Clinical Capabilities, que fornece informação histórica e em tempo-real de um determinado paciente, sendo essa informação adaptada às necessidades clínicas especificas de cada um; AirStrip Accelerator Services, que serve para conduzir e manter um determinado tipo de tratamento e para transformar os tratamentos existentes, medindo o impacto dos mesmos. Desta forma, o AirStrip ONE é independente do fornecedor ou do dispositivo que transmite os dados e ao mesmo tempo é agnóstico quanto às fontes de dados. [8]

Figura 9. AirStrip ONE (retirada de www.venturebeat.com/2014/08/26/airstrip-raises-25m-for-mobile-clinical-platform)

Vantagens: permite o uso de diferentes sensores de diferentes fabricantes;

Desvantagens: não opera em redes 3G ou 4G;

2.1.2 Outras aplicações de monitorização de sinais vitais

A escolha das aplicações apresentadas e descritas nesta secção tem por base alguma característica particularmente interessante e inovadora. Estas aplicações diferem das apresentadas em 2.1.1. pelo facto de serem mais simples, de uso mais pessoal e conterem

algumas limitações quer a nível de número de sensores monitorizados, quer em propósito de utilização.

2.1.2.1. AliveCor

A AliveCor é uma aplicação para dispositivos móveis para monitorizar os batimentos cardíacos e apresentar um Electrocardiograma (ECG). Esta aplicação vem acompanhada por um acessório que se coloca na parte de trás do smartphone (como podemos verificar na Figura 8) e que serve para capturar o ECG e o batimento cardíaco, colocando para isso o dispositivo junto ao peito ou dispondo os dedos sobre ele. Esta aplicação permite que seja visto um ECG em apenas 30 segundos e que sejam detetados problemas de fibrilação atrial. Esta aplicação está disponível para o Iphone (iOS) e para os smartphones Android. [9]

Figura 10. AliveCor (retirada de www.medgadget.com/2014/02/fda-over-the-counter-approval-for-alivecor-heart-monitor-interview-with-ceo-euan-thomson.html)

Vantagens: medição do ECG de forma simples e prática.

Desvantagens: não permite obter o feedback necessário caso detecte algum problema.

2.1.2.2. Instant Heart Rate (Azumio, Inc.)

O Instant Heart Rate é uma aplicação existente para iOS, Android e Windows Phone que permite a medição dos batimentos cardíacos. Esta medição é feita colocando o dedo indicador sobre a câmara do smartphone. A aplicação apresenta a medição do batimento cardíaco, gráficos de PPG (Photoplethysmogram) que mostra o batimento cardíaco ao longo do tempo, permite guardar dados e *tags* relacionadas e possibilita também a exportação desses dados para utilizadores registados na plataforma. Esta aplicação chegou a ser considerada a melhor aplicação Health & Fitness nos Mobile Premier Awards, onde o júri era composto por várias personalidades experientes na indústria. [10]

Figura 11. Instant Heart Rate (retirada de www.transforminghealth.org/stories/2013/02/app-of-the-week-instant-heart-rate.php)

Vantagens: obtenção do batimento cardíaco de forma inovadora.

Desvantagens: em caso de alerta, não é possível qualquer tipo de socorro.

2.1.2.3. Vital Signs Camera (Philips)

A Vital Signs Camera é uma aplicação disponível para iOS que permite a medição do batimento cardíaco e do ritmo respiratório usando para isso a câmara do próprio iPhone ou iPad. Basta para isso captar a imagem da pessoa e através dela a aplicação identifica pequenas mudanças na cor da pele da cara para medir o ritmo cardíaco e detecta também os movimentos do peito para medir o ritmo respiratório exato. Possibilita ainda que duas pessoas possam obter uma medição ao mesmo tempo e para isso basta que as duas pessoas estejam a ser captadas pela câmara. Esta foi a primeira aplicação capaz de medir o ritmo cardíaco e respiratório à distância. [11]

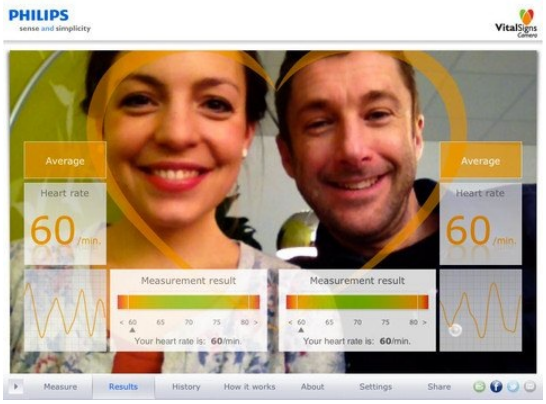

Figura 12. Vital Signs Camera (retirada de www.ipadinsight.com/ipad-apps/vital-signs-camera-for-ipad-2-updated-adds-history-and-more)

Vantagens: medição do batimento cardíaco de duas pessoas ao mesmo tempo.

Desvantagens: esta medição baseia-se em factores pouco comuns neste tipo de sinais como cor da pele.

2.1.2.4. iHealth (iHealth Labs, Inc.)

iHealth é uma aplicação que monitoriza vários sinais vitais, dados biométricos relativos à atividade física e informações como peso e nível de glicose no sangue. Para isso esta aplicação serve-se de vários acessórios vendidos separadamente, que vão desde um medidor de glicose no sangue, até uma balança sem fios ou um medidor de ritmo cardíaco sem fios e uma pulseira para controlar a atividade física do utilizador. [12]

Figura 13. iHealth e o componente de pressão arterial (retirada de www.carevium.com/carevium-assisted-living-software-health-monitoring-devices-ihealth-integration)

Vantagens: medição de vários dados biométricos.

Desvantagem: necessidade de comprar vários acessórios em separado para a medição de diferentes sinais vitais.

2.1.3. Sumário

Para sumarizar este estudo é importante fazer uma comparação entre os sistemas apresentados na secção 2.1., por serem as opções que mais se podem comparar com o sistema que se pretende implementar. Esta comparação serve também para identificar as eventuais lacunas das soluções existentes.

Sistemas	Apto para ambientes PPDR	Monitorização em tempo-real	Monitorização de vários sensores	Monitorização por vários intervenientes	Dispositivos *wearable*
Visi Mobile	✗	✓	✓	✓	✓
CareLink	✗	✓	✗	✓	✗
Telcare	✗	✓	✗	✗	✗
2net	✗	✓	✓	✓	✗
AirStrip	✗	✓	✓	✓	✗
AliveCor	✗	✓	✗	✗	✗
Instant Heart Rate	✗	✓	✗	✗	✗
Vital Signs Camera	✗	✓	✓	✗	✗
iHealth	✗	✓	✓	✗	✗

Tabela 5. Comparação entre aplicações

Como é possível observar pela Tabela 5, a principal lacuna apresentada por todos os sistemas analisados é o facto de não estarem aptos a operar em ambientes PPDR. Isto deve-se ao facto de não terem sido construídos especificamente para esse fim. Por exemplo, em ambientes de emergência não é apenas necessário que o sistema seja *wearable* mas também que seja resistente.

Relativamente às outras aplicações apresentadas e apesar destas conterem particularidades interessantes como a medição do ECG através da colocação do dedo indicador na câmara fotográfica do *smartphone* ou apenas através da cara usando a câmara do telefone, não poderá haver uma comparação direta com o que se pretende desenvolver. Isto porque este tipo de aplicações são de uso mais pessoal e são-lhes identificadas características muito distintas e com apenas um ponto em comum: a monitorização em tempo-real de um bio-sinal.

O sistema que se pretende implementar e descrito neste relatório quer ser uma junção de todas estas funcionalidades num só sistema. Ou seja, é suposto ser um sistema que monitorize um variado número de sinais vitais de uma pessoa em tempo-real e que essa informação e essa monitorização sejam passíveis de ser acedidas por um variado rol de intervenientes como, por exemplo, um médico, um especialista de saúde ou outros.

2.2. Comunicação

Nesta secção são abordadas as tecnologias de rede que poderão ser usadas na implementação de mecanismos de comunicação entre os vários componentes que farão parte do sistema.

2.2.1. Métodos

Nesta secção são apresentados dois métodos de comunicação diferentes, passíveis de serem usados na implementação do sistema. Estes métodos foram escolhidos com base no conhecimento prévio e na experiência prática que se possui com as mesmas.

2.2.1.1. Sockets

Os *Sockets* são uma forma de comunicação dentro de uma rede e base das comunicações IP. Um *Socket* é identificado segundo três parâmetros: endereço local do *socket*, ou seja, o endereço de IP local e o número da porta; o endereço remoto, apenas usado em *sockets* TCP com a função de suportar vários clientes num servidor ao mesmo tempo; protocolo de transporte, como por exemplo, TCP, UDP ou raw IP.

Existem vários tipos de *Socket*: *datagram socket*, que usam o protocolo UDP (*User Datagram Protocol)* e são não-conectados, isto é, não é necessária nenhuma preparação da mensagem aquando do seu envio; *stream sockets*, que são *sockets* orientados para a conexão, onde as mensagens tem uma estrutura definida e necessitam de preparação antes do envio. Os protocolos usados nos *stream sockets* podem ser o TCP (*Transmission Control Protocol*) ou o SCTP (*Stream Control Transmission Protocol*); *raw sockets*, disponíveis em routers ou qualquer outro equipamento de rede. Neste tipo de *sockets* a camada de transporte é abstraída, tendo as aplicações acesso aos cabeçalhos dos pacotes. [13]

2.2.1.2. RPC's (Remote Procedure Call)

O Java RMI (*Remote Method Invocation)* é uma API Java que possibilita a transferência direta de classes de Java serializadas entre um cliente e um servidor e é uma possível representação de um RPC.

Aquando da sua implementação inicial, esta API dependia dos mecanismos usados para a representação de classes presentes na JVM (*Java Virtual Machine*), e apenas suportava chamadas entre JVM's. O protocolo usado nesta primeira implementação era o JRMP (*Java Remote Method Protocol*).

Numa versão posterior, uma versão CORBA foi desenvolvida para suportar a execução de código fora da JVM. [14]

2.2.1.3. Comparação

Nesta subsecção é apresentada uma tabela comparativa entre os métodos apresentados. São tidos em conta fatores como o que se pode enviar em cada um dos métodos.

Método	Formato dos dados a enviar
Socket	Qualquer formato de dados
RPC	Dados especificamente formatos

Tabela 6. Comparação métodos comunicação

Esta é a principal diferença entre a utilização de Sockets e RPC e é o principal fator de escolha na implementação do sistema. É necessário ter a flexibilidade de construir as mensagens a enviar da forma que for mais conveniente e por essa razão os Socket simples são a melhor opção.

2.3. Sistemas de Gestão de Base de Dados

Nesta secção, são apresentados os sistemas de gestão de bases de dados que foram considerados para este projeto, tendo em conta que a base de dados será uma parte fulcral no sistema. São escolhidas para análise as bases de dados relacionais e as bases de dados não relacionais. Esta

análise é feita de uma forma generalizada e procurando encontrar a melhor solução entre estes dois tipos.

2.3.1. Bases de Dados relacionais

Os sistemas de gestão de bases de dados é a escolha mais comum para guardar os registos de vários dados como dados pessoais, informação logística, dados financeiros ou outro tipo de informações desde 1980.

Cada base de dados relacional é composta por um conjunto de tabelas que contêm relações entre elas. Estas tabelas são representações metafóricas de uma entidade ou de um objeto que consistem em colunas e linhas. As colunas são os atributos de cada objeto e as linhas são os vários registos das tabelas.

Para estas tabelas funcionarem de forma eficiente, tem de satisfazer as propriedades ACID(*Atomicity, Consistency, Isolation, Durability*): [15]

- *Atomicity:* quando uma transação é iniciada tem de ser terminada na totalidade, ou seja, se uma parte da transação falhar, toda a transação falha e o estado da base de dados não se modifica.
- *Consistency:* assegura que qualquer transição leva a que uma base de dados passe de um estado válido para outro diferente mas igualmente válido.
- *Isolation:* se várias transações estiverem a ser executadas de forma concorrente, esta propriedade garante que são executadas em cadeia, ou seja, umas seguidas das outras.
- *Durability:* depois de uma transação ser confirmada, esta continua confirmada mesmo ue haja algum erro como perda de energia.

2.3.2. NoSQL

Uma base de dados NoSQL fornece mecanismos de armazenamento e busca diferentes das bases de dados relacionais. As estruturas de dados de uma base de dados NoSQL diferem das bases de dados relacionais o que permite que as bases de dados NoSQL sejam mais rápidas que as relacionais em certas tarefas mas também mais lentas noutras. Por exemplo, para o processo de armazenamento de dados, as bases de dados não relacionais são mais rápidas. Por outro lado, este tipo de base de dados não garante um conjunto importante de propriedades, ACID (*Atomicity, Consistency, Isolation, Durability*), regendo-se, no entanto, pelo teorema de Brewers CAP (*Consistency, Availability and Partition tolerance*). Este tipo de bases de dados está a ser cada vez mais usado em aplicações que contenham um fluxo de uma quantidade de dados muito grande e em aplicações *web* em tempo-real.

Embora existam várias abordagens para a classificação de bases de dados NoSQL, a classificação básica passa pelo modelo de dados:

- Coluna, onde são exemplos: Accumulo, Cassandra. Druid, HBase ou Vertica.
- Documento, onde são exemplos: Clusterpoint, Apache CouchDB, Couchbase, MarkLogic ou MongoDB.
- Chave-valor, onde são exemplos: Dynamo, FoundationDB ou Redis.
- Grafo, onde são exemplos: Allegro, Neo4J ou Virtuoso.
- Multi-modelo: OrientDB ou CortexDB.

2.3.3. Comparação

Nesta subsecção é apresentada uma tabela comparativa entre os diferentes sistemas de bases de dados apresentados (SQL e NoSQL). [16]

SGBD	Modelo de Dados	Propriedades	Escalabilidade
SQL	Tabelas	ACID	Vertical
NoSQL	Documentos, pares chave-valor, grafos	CAP	Horizontal

Tabela 7. Comparação de SGBD's

Feita uma comparação entre os dois sistemas de bases de dados, a principal razão para a escolha de bases de dados SQL é o facto de cumprir e garantir as propriedade ACID. Em contra partida a escalabilidade das bases de dados NoSQL é horizontal o que permite que seja apenas necessária a inclusão de mais servidores para poder gerir as grandes quantidades de tráfego. Tendo uma escalabilidade vertical, as bases de dados SQL necessitam de um incremento de CPU e RAM num único servidor para poder sustentar o aumento de carga.

Para o desenvolvimento do sistema, e tendo em conta que é primordial a garantia das propriedades ACID, as bases de dados SQL revelam-se como as apropriadas, embora haja a possibilidade de, no futuro, comparar a performance entre os dois tipos de bases de dados.

2.4. Frameworks de Desenvolvimento

Esta secção serve para apresentar um estudo feito sobre *frameworks* de desenvolvimento na linguagem Java. São abordadas as *frameworks NIO* e as aplicacionais. Este estudo foi feito para decidir quais a passíveis de serem usadas no desenvolvimento do sistema em causa no estágio.

2.4.1. NIO

A implementação de aplicações de servidor na linguagem Java sempre for algo difícil de fazer. Antes do aparecimento do *Java New I/O (NIO)*, os problemas relacionados com a gestão de *threads* faziam com que fosse impossível criar aplicações robustas capazes de escalar para centenas ou milhares de utilizadores. Nesta secção, são apresentadas *frameworks* que foram desenhadas para ajudar no desenvolvimento de aplicações de servidor altamente escaláveis em Java e que tiram partido de todas as vantagens que o *Java NIO API* trouxe. [17]

2.4.1.2. Project Grizzly

O *Grizzly* é uma *framework NIO* desenvolvida pela *Oracle*. Está disponível para Java 6 ou superior. A parte principal da *framework* inclui: gestão de memória, estratégias de *Input* e *Output*, unificação de portas e várias configurações. Esta *framework* é usada em vários projetos *java.net* incluindo o *GlassFish*, *Shoal* e *Jersey*. [18]

2.4.1.2. CoralReactor3

O *CoralReactor* é uma *framework NIO* desenvolvida pela empresa Coral Software LLC. Esta *framework* está inserida num pacote intitulado *CoralBlocks.* Esta é a framework mais recente de todas as analisadas. No site oficial da *framework* existem vários artigos que comprovam que esta é a mais rápida de todas as *frameworks* no mercado. Um desses exemplos é uma comparação de performance com o *Netty*, outra das ferramentas analisadas neste estado da arte. [19]

2.4.1.3. Netty

O *Netty* é uma *framework* Java NIO desenvolvida pela *Netty Project Community* em que a última versão estável foi lançada em Outubro de 2014. Esta ferramenta é coberta pela licença *Apache* 2.0. Possui vários guias de utilizador e uma documentação recheada de exemplos. [20]

2.4.1.4. Comparação

Nesta subsecção é apresentada uma tabela comparativa das *frameworks* apresentadas e feita uma análise.

Framework	Documentação	Última Versão	Data de Lançamento
Grizzly	Tutoriais simples e alguns exemplos	2.3	Setembro 2013
CoralReactor	Tutoriais simples e *live chat* no site	1.0	Abril 2014
Netty	Detalhada com vários exemplos	4.0	Junho 2014

Tabela 8. Comparação de Frameworks NIO

Comparando as três *frameworks* analisadas, podemos afirmar que a *Grizzly* é aquela que é utilizada há mais tempo quando comparada com as restantes e foi implementada em projetos de grande importância como o *Glassfish* (implementação oficial dos *Servlets* 3.0). Por sua vez, a *CoralReactor* é a mais recente de todas as *frameworks* analisadas e apresenta no seu próprio *website* um artigo que a compara diretamente com o *Netty*, afirmando ter uma melhor performance. O *Netty* possui uma documentação bastante detalhada e vários exemplos do seu funcionamento e da sua implementação.

2.4.2. Frameworks Aplicacionais JAVA

Nesta subsecção é apresentado um estudo feito sobre frameworks aplicacionais para a linguagem Java. O levantamento das mais adequadas foi feito com base em critérios relativos ao projeto: a framework a utilizar teria de ter uma componente de desenho de gráficos que fosse altamente personalizável e era necessário que tivesse todas as componentes de interação com o utilizador.

2.4.2.1. Swing

O Java Swing é uma framework Java e independente da plataforma onde é utilizada. Faz parte do JFC (Java Foundation Classes) da Oracle, uma API que providencia interfaces de utilizador gráficas para programas Java. É altamente configurável e modular. Trabalha sobre o modelo MVC (Model View Controller). Encontra-se neste momento num processo de substituição pelo JavaFX. Nesta framework, a construção de gráficos tem de ser feita recorrendo a bibliotecas externas como, por exemplo, o JFreeChart. [21]

2.4.2.2. JSF (Java Server Faces)

O JSF é uma framework Java que tem como principal foco o desenho de interfaces de utilizador que sejam baseadas na Web. É baseado no modelo de design *component-driven* e usa ficheiros XML aos quais chama de *view templates* ou *Facelets*. Os *FaceServlets* são responsáveis por processar os pedidos, carregar a *view* mais apropriada, construir uma estrutura com as várias componentes e reenviar a resposta para o cliente. O estado de cada componente é guardado no fim de cada pedido e tanto o cliente como o servidor podem guardar objetos e estados. Nesta framework é necessária a utilização de bibliotecas externas para o desenho de gráficos 2D como o PrimeFaces ou o Apache myFaces. [22]

2.4.2.3. JavaFX

O JavaFX é uma framework Java que permite criar RIAs (Rich Internet Applications) capazes de correr numa grande variedade de dispositivos. Esta framework foi construída com o intento de substituir o a framework Swing como a biblioteca standard para construir interfaces gráficas na versão standard do Java (Java SE) estando neste momento as duas disponíveis. Esta framework tem suporte tanto para computadores como para browsers web, ou seja, com ela podemos construir aplicações baseadas na Web ou aplicações *standalone*. Tem suporte para Linux, Microsoft Windows e Mac OS. O JavaFX contém uma componente própria para o desenho de gráficos 2D e várias opções para a sua personalização. [23]

2.4.2.4. Comparação

Nesta subsecção é feita uma comparação entre as frameworks aplicacionais analisadas e é apresentada uma análise a esta comparação.

Framework	Web/Standalone	Gráficos de Linhas 2D	Última Versão	Data de Lançamento
Swing	Ambas	Biblioteca Externa	Incluído no Java SE 8	18-03-2014
JSF	Web	Biblioteca Externa	2.2	21-05-2013
JavaFX	Ambas	Incluído na framework	Incluído no Java SE 8	18-03-2014

Tabela 9. Comparação entre frameworks

Analisando esta tabela comparativa podemos concluir que o JavaFX e o Swing são as mais indicadas para o desenvolvimento do sistema, tendo em conta que além de serem as ferramentas mais recentes, são também usadas tanto em aplicações *desktop* como em aplicações baseadas na Web. Como o JavaFX é a framework que está programada para substituir de vez o Swing no futuro e considerando que não é necessário usar bibliotecas externas para o desenho dos gráficos de linhas 2D, parece ser a mais indicada para ajudar a cumprir os requisitos de todo o sistema.

Capítulo 3 – Requisitos

Neste capítulo irá ser feito o levantamento e a análise de requisitos do sistema a implementar. Para fazer este estudo foi usada a engenharia de requisitos baseada em cenários, ou seja, eventos hipotéticos. Este tipo de análise permite que se imagine o comportamento de um determinado sistema e que os seus utilizadores possam comentar sobre a interação que esperam poder ter com ele. Para isso, começa-se por identificar três cenários PPDR distintos onde o sistema poderá ser usado. Estes cenários foram fornecidos pela empresa e explorados/especificados pelo estagiário. Esta especificação de cenários serve de suporte para a análise de requisitos que é feita em seguida. Todos os requisitos apresentados foram definidos pelo estagiário e aprovados pela empresa, tendo em conta as necessidades do sistema. Por fim, são apresentados os desafios reconhecidos que terão de ser enfrentados tanto na especificação da solução a implementar como do desenvolvimento do sistema.

3.1. Levantamento de Requisitos

Para o processo de levantamento de requisitos, começou-se por pesquisar e definir cenários onde o sistema fosse utilizado e trouxesse uma mais-valia para os seus intervenientes. Este processo teve inicio com o contacto da empresa com os diversos parceiros presentes no projeto, onde estão incluídas diferentes áreas como a saúde ou as telecomunicações. Depois disto foi dado ao estagiário um quadro geral sobre as possíveis situações que caracterizam os ambientes *PPDR* para posterior análise e detalhe sobre cada um deles. Foram fornecidos também os primeiros *mockups* para a interface dos *Control Center*, como é apresentado na Figura 14. Este artefacto foi elaborado no seio do projeto SALUS e revelou-se bastante útil para o trabalho de levantamento de requisitos.

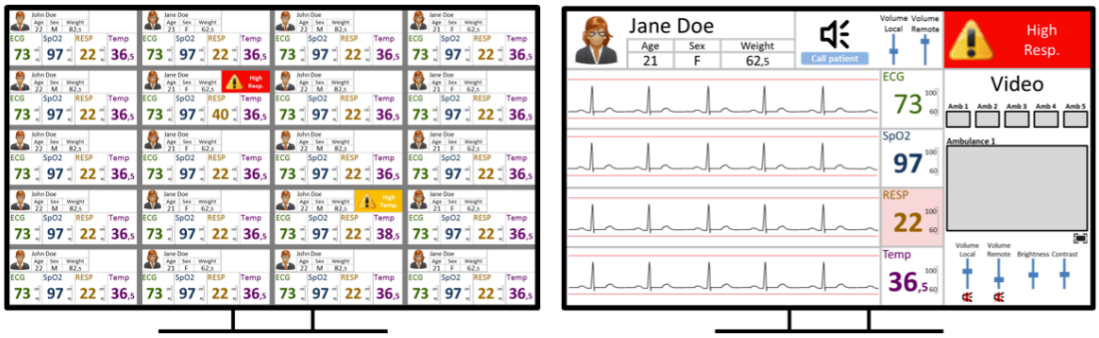

Figura 14. Primeiros mockups do *Control Center*

Depois deste levantamento de requisitos, foi possível fazer uma análise mais consciente de todos os detalhes que deviam fazer parte do sistema a implementar.

3.1.1. Cenários

Nesta secção são descritos três cenários onde o sistema poderá ser usado. Em cada um dos cenários é feita uma descrição do mesmo e definidos os intervenientes.

A definição dos cenários parte de um diagrama explicativo representado na Figura 15 e que foi fornecido pela empresa ao estagiário. Este diagrama apresenta três principais aspetos a ter em conta ao construir os cenários. Em primeiro lugar existe a monitorização em tempo-real de vários ou de um determinado paciente, representada por *in loco diagnosis*. Quando surge algum problema em relação aos dados que estão a ser visualizados e analisados tem de ser disparado um alerta (*Emergency Alert*). Por fim tem de haver a resposta à emergência identificada. A análise em tempo-real dos dados torna esta resposta mais optimizada.

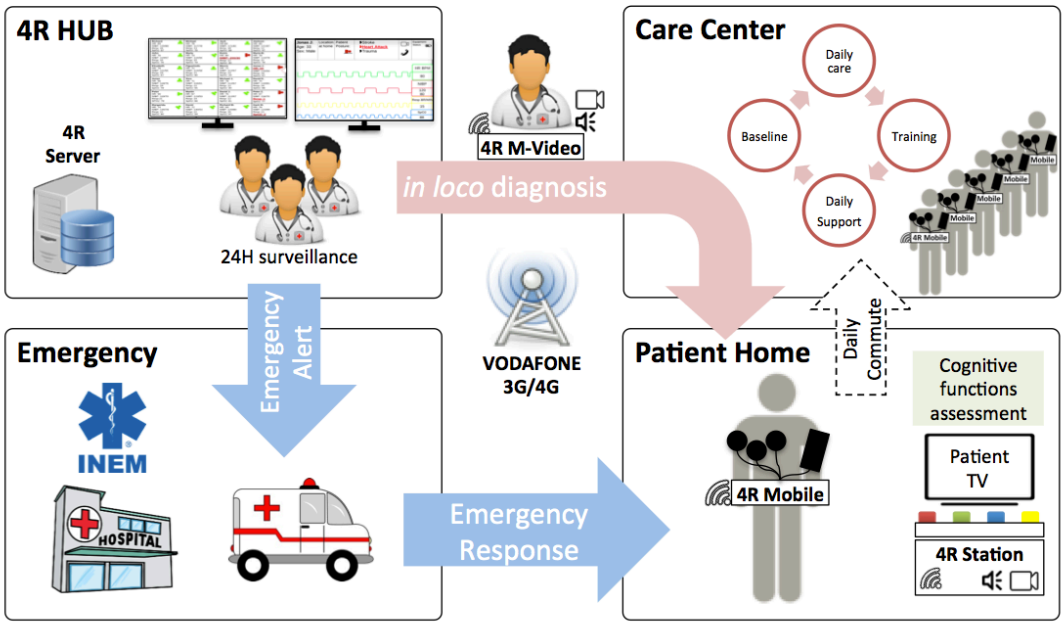

Figura 15. Representação geral de um cenário.

Depois de analisada a representação geral de um cenário, partiu-se então para a definição de cada um deles, mais em detalhe.

3.1.1.1. Cenário A: Paciente com problemas cardíacos.

Manuel Ribeiro é um homem de 39 anos que sofre de problemas cardíacos graves. Num determinado dia começou-se a sentir mal e ligou para a emergência médica. Ao receber o pedido de socorro, é colocado um dispositivo no paramédico responsável pela resposta a esta emergência. Este dispositivo contém vários sensores que medem sinais vitais como o electrocardiograma (ECG) e a taxa de respiração mas também uma câmara e um microfone para captação de áudio e vídeo diretamente do local da emergência. Desta forma, o médico responsável e que se encontrará no hospital à espera da vítima, poderá ter uma melhor percepção do que se passa e administrar os medicamentos e procedimentos que mais se adequam ao incidente em questão. Ao mesmo tempo poderá controlar a ansiedade dos paramédicos e analisar vários detalhes através da visualização do vídeo e de escuta do sinal de áudio.

Intervenientes:

- Manuel Ribeiro – paciente que sofre de problemas cardíacos.
- Médico – será o responsável pelo centro de controlo e por monitorizar os dados enviados pelo Paramédico.
- Paramédico – é o responsável por ocorrer à emergência. Será ele a transportar o dispositivo.

Ao analisar este cenário, pode-se concluir que usando um sistema como o que se pretende implementar neste trabalho é possível monitorizar os dados em tempo-real e ao mesmo tempo ter acesso aos sinais de vídeo e áudio. Desta forma, tanto a resposta a qualquer emergência como a seguimento do estado clinico do paciente serão feitos de uma forma mais eficaz.

3.1.1.2. Cenário B: Equipa de polícia em operação de resgate.

Uma equipa da polícia é chamada a uma fábrica que foi assaltada por um grupo de terroristas que mantiveram várias pessoas como reféns. Chegando ao local, vários agentes são equipados com

uma *backpack* composta por sensores de sinais vitais como o ECG, pressão arterial e taxa de respiração e também por uma câmara e um microfone para captar imagem e som. Além destes sensores é colocado também um sensor de GPS e um acelerómetro. Outro agente é colocado na carrinha que transporta todo o material necessário para monitorizar os dados enviados por cada um dos agentes e para dar as ordens mais indicadas para cada situação. Desta maneira, será possível controlar a posição de cada um dos agentes e controlar a operação da melhor maneira. Para complementar este aspeto, será também possível visualizar em tempo-real os dados vitais dos agentes para identificar possíveis situações de emergência verificando, por exemplo, se existe ansiedade que indique problemas no local. Para complementar tudo isto existe também a transmissão de imagem e som quando possível, o que facilita a percepção acerca do que está a acontecer no local. Por fim, o acelerómetro permite detetar quedas ou *Man-Down*.

Intervenientes:

- Agentes da polícia – portadores da *backpack* com os sensores, câmara e microfone.
- Coordenador da operação – responsável pelas rédeas do centro de controlo, monitorizando todos os agentes envolvidos na operação. Quando for necessário, poderá monitorizar um agente em específico.

Neste cenário a incorporação do serviço de voz e vídeo é essencial para o sucesso da utilização do sistema. Em situações de tensão em que não seja possível ao coordenador perceber bem o que se está a passar apenas através da transmissão de som e imagem, os dados dos sinais de ECG e taxa de respiração têm um papel fundamental.

3.1.1.3. Cenário C: Bombeiros a socorrer pessoas presas em prédio cercado por chamas.

Um incêndio deflagra num prédio de três andares. Uma equipa de bombeiros é chamada ao local para resgatar as pessoas presas devido ao fogo. Cada bombeiro vai equipado com uma mochila que contém uma câmara, um microfone, sensores biométricos e um sensor de níveis de CO2 no ar. Ao entrar no edifício, será possível obter imagens e som do seu interior e também os níveis de CO2 no ar. Desta forma, será possível dar indicações a cada um dos bombeiros para que possam ser tomadas as devidas precauções em cada situação.

Intervenientes:

- Bombeiro – Será o portador do dispositivo incorporado numa mochila com uma câmara, um microfone e um micro PC.
- Chefe dos Bombeiros – O chefe dos bombeiros será o responsável por gerir o Centro de Controlo para monitorizar todos os dados biométricos e de CO2 enviados por cada bombeiro. É o responsável por tomar as decisões em momentos críticos.

3.2. Análise de Requisitos

Nesta secção são apresentados os requisitos do sistema a implementar, que resultam do levantamento descrito na secção anterior. Nas tabelas apresentadas é dado um código ao requisito (para ser de mais fácil referência ao longo do documento) e uma descrição. O código de cada um dos requisitos segue a seguinte nomenclatura: "REQ_Módulo_ID". O módulo é representado pela primeira letra de cada uma das componentes do sistema, ou seja: "CC" refere-se ao *Control Center*; "S" refere-se a servidor; "D" refere-se a dispositivo. O ID é um identificador único de cada um dos requisitos de cada componente (começa em "01" para cada uma das componentes) e a ordem não espelha qualquer tipo de prioridade ou importância. Para a classificação de cada requisito é usado o classificador "Prioridade" que pode ter vários níveis: Baixa, Normal, Alta ou Muito Alta. Neste primeira definição de requisitos foi seguido o modelo apresentado:

- Muito Alta: corresponde a um requisito que é considerado fundamental para o desenvolvimento de um protótipo funcional ou de um *POC (Proof of Concept).*

- Alta: esta prioridade é dada a requisitos que sejam necessários de implementar para uma apresentação de um protótipo a pessoas externas (a um cliente, por exemplo).
- Normal: corresponde a um requisito que apenas precisa de ser implementado para atingir um produto final, capaz de desempenhar todas as funcionalidades esperadas.
- Baixa: esta prioridade identifica o requisito como sendo apenas necessário numa aplicação que se pretenda robusta e que para além das funcionalidades esperadas implemente algo extra que conceda ao sistema uma maior robustez.

Para a definição dos requisitos funcionais é feita uma divisão pelos componentes do sistema apresentados no Capítulo 1 para que fique bem delineado o que é necessário implementar para cada uma das partes. Estão assim definidos os três componentes: *Control Center*, *Server* e *Device*. Para o levantamento dos requisitos não funcionais, deixa-se de parte esta divisão, encarando o sistema como um todo por estes representarem os requisitos mais genéricos.

3.2.1 Requisitos Funcionais

São apresentados nesta secção os requisitos funcionais identificados para cada um dos componentes do sistema: *Control Center*, *Server* e *Device*. De notar que, tendo em conta a metodologia de desenvolvimento utilizada, todos estes requisitos podem vir a sofrer alteração, seja porque a definição inicial não foi a mais correta ou por se verificar que algo não é possível de se concretizar.

3.2.1.1. Control Center

Na Tabela 10 são apresentados os requisitos funcionais para o *Control Center*. De notar que a maior parte destes requisitos referem-se a visualização de dados, principal função do utilizador deste componente do sistema.

Código	Descrição	Prioridade
REQ_CC_01	Autenticar-se com sucesso no sistema através de utilizador e palavra-chave	Alta
REQ_CC_02	Estabelecer ligação com o servidor, com sucesso	Muito Alta
REQ_CC_03	Visualizar todos os dispositivos ativos	Alta
REQ_CC_04	Monitorizar vários dispositivos ao mesmo tempo	Muito Alta
REQ_CC_05	Monitorizar um dispositivo em detalhe	Muito Alta
REQ_CC_06	Apresentar fotografia de cada individuo na lista de dispositivos	Normal
REQ_CC_07	Apresentar nome de cada individuo na lista de dispositivos	Alta
REQ_CC_08	Apresentar sexo de cada individuo na lista de dispositivos	Alta
REQ_CC_09	Apresentar a idade de cada individuo na lista de dispositivos	Alta
REQ_CC_10	Apresentar o valor para o sensor de ECG para cada individuo na lista de dispositivos	Alta
REQ_CC_11	Apresentar o valor para o sensor de temperatura para cada individuo na lista de dispositivos	Normal
REQ_CC_12	Apresentar o valor da taxa de respiração para cada individuo na lista de dispositivos	Normal

REQ_CC_13	Ao selecionar um dispositivo, a informação sobre o individuo deve aparecer na secção de detalhe	Muito Alta
REQ_CC_14	Apresentar nome do individuo na secção de detalhe	Muito Alta
REQ_CC_15	Apresentar idade do individuo na secção de detalhe	Muito Alta
REQ_CC_16	Apresentar sexo do individuo na secção de detalhe	Muito Alta
REQ_CC_17	Apresentar a localização do individuo na secção de detalhe	Baixa
REQ_CC_18	Apresentar o gráfico com os dados do ECG em tempo-real na secção de detalhe	Muito Alta
REQ_CC_19	Visualizar o gráfico do ECG aumentado na secção de detalhe	Baixa
REQ_CC_20	Apresentar o gráfico com os dados da taxa de respiração em tempo-real na secção de detalhe	Alta
REQ_CC_21	Visualizar o gráfico da taxa de respiração aumentado na secção de detalhe	Baixa
REQ_CC_22	Apresentar os valores das coordenadas x, y e z para o acelerómetro na secção de detalhe	Normal
REQ_CC_23	Apresentar uma figura que represente a posição do individuo na secção de detalhe	Normal
REQ_CC_24	Redimensionar janela manualmente	Normal
REQ_CC_25	Redimensionar janela de acordo com o tamanho do ecrã	Normal
REQ_CC_26	Visualizar histórico na secção de detalhe	Normal
REQ_CC_27	Apresentar uma secção reservada para os alertas	Alta
REQ_CC_28	No caso de haver algum alerta, este tem de ser apresentado na secção reservada para o efeito	Alta
REQ_CC_29	Apresentar uma secção reservada para a transmissão de voz e vídeo	Muito Alta
REQ_CC_30	Possibilitar o inicio de vídeo	Alta
REQ_CC_31	Possibilitar a paragem da visualização do sinal de vídeo	Alta
REQ_CC_32	Regular a qualidade do sinal de vídeo	Normal
REQ_CC_33	Possibilitar o inicio do sinal de áudio	Alta
REQ_CC_34	Possibilitar a paragem do sinal de áudio	Alta
REQ_CC_35	Regular o volume do sinal de áudio	Normal
REQ_CC_36	Ao sucederem, visualizar alertas amarelos	Normal
REQ_CC_37	Ao sucederem, visualizar alertas vermelhos	Normal
REQ_CC_38	Cada alerta amarelo deve destacar o dado a que se refere a amarelo	Normal
REQ_CC_39	Cada alerta vermelho deve destacar o dado a que se refere a	Normal

	vermelho	
REQ_CC_40	Possibilitar o descarte de um determinado alerta	Normal
REQ_CC_41	Apresentar o tipo de rede que está a ser utilizado pelo dispositivo, no ecrã de detalhe	Alta
REQ_CC_42	Apresentar a qualidade de rede do dispositivo no ecrã de detalhe	Alta
REQ_CC_43	Permitir visualização de vídeo em ecrã inteiro	Alta
REQ_CC_44	Ordenar lista de utilizadores por nome	Normal
REQ_CC_45	Pesquisar utilizador na lista de utilizadores	Normal
REQ_CC_46	Visualizar o estado da bateria do dispositivo	Normal
REQ_CC_47	Apresentar uma secção para receber notificações	Baixa
REQ_CC_48	Captar *screenshot* a partir do sinal de vídeo	Baixa

Tabela 10. Requisitos Funcionais *Control Center*

3.2.1.2. Server

Na Tabela 11 são apresentados os requisitos funcionais do Servidor. Todos estes requisitos referem-se a funcionalidades que decorrem em *background* e que para serem testados e validados terão de ser analisados *logs* e *outputs* do sistema.

Código	Descrição	Prioridade
REQ_S_01	Aceitar a conexão de um dispositivo no sistema	Muito Alta
REQ_S_02	Aceitar a conexão de vários dispositivos no sistema	Muito Alta
REQ_S_03	Autenticar um dispositivo no sistema	Alta
REQ_S_04	Aceitar a conexão de um *Control Center* no sistema	Muito Alta
REQ_S_05	Aceitar a conexão de vários *Control Center* no sistema	Muito Alta
REQ_S_06	Autentica um *Control Center* no sistema	Alta
REQ_S_07	Receber os dados de um dispositivo	Muito Alta
REQ_S_08	Enviar dados de um dispositivo para os *Control Center* que os pretendam visualizar	Muito Alta
REQ_S_09	Analisar os dados que chegam ao servidor	Normal
REQ_S_10	Enviar alertas para os *Control Center* caso haja um problema	Alta
REQ_S_11	Guardar dados na base de dados para efeitos de histórico	Alta
REQ_S_12	Alterar configurações da base de dados	Alta
REQ_S_13	Ler da Base de Dados os dados relativos ao histórico	Alta
REQ_S_14	Enviar lista de dispositivos configurados como visíveis para um determinado *Control Center*	Muito Alta

Tabela 11. Requisitos Funcionais *Server*

3.2.1.3. Device

De seguida é apresentada uma tabela de requisitos funcionais para o *Device*.

Código	Descrição	Prioridade
REQ_D_01	Conectar-se ao servidor	Muito Alta
REQ_D_02	Autenticar-se com sucesso no servidor	Alta
REQ_D_03	Captar os dados dos sensores	Muito Alta
REQ_D_04	Analisar os dados	Alta
REQ_D_05	Encriptar os dados para envio	Alta
REQ_D_06	Enviar os dados para o servidor	Muito Alta
REQ_D_07	Em caso de detetar problema, enviar alerta para o servidor	Alta
REQ_D_08	Em caso de detetar problema, enviar alerta para o Message Broker do projeto SALUS	Alta
REQ_D_09	Enviar dados do sensor de ECG	Muito Alta
REQ_D_10	Enviar dados do sensor de temperatura	Muito Alta
REQ_D_11	Enviar dados do sensor de taxa de respiração	Muito Alta
REQ_D_12	Enviar dados do sensor acelerómetro	Muito Alta
REQ_D_13	Enviar sinal de vídeo	Alta
REQ_D_14	Enviar sinal de voz	Alta

Tabela 12. Requisitos Funcionais *Device*

3.2.2. Requisitos Não-Funcionais

Na tabela seguinte são apresentados os requisitos não-funcionais do sistema como um todo.

Código	Descrição	Prioridade
REQ_NF_01	Funcionar em redes Wi-Fi	Alta
REQ_NF_02	Funcionar em redes móveis como 4G e LTE	Alta
REQ_NF_03	A mochila que transporta os sensores tem de ser resistente	Alta
REQ_NF_04	A mochila que transporta os sensores tem de ser leve	Alta
REQ_NF_05	Todo o sistema tem de ser resiliente	Muito Alta
REQ_NF_06	As ligações entre servidor e outros componentes devem ser repostas em caso de perda	Alta
REQ_NF_07	Suportar até dez *Control Center* como de dispositivos	Muito Alta
REQ_NF_08	Suportar até vinte dispositivos conectados e a enviar dados ao mesmo tempo	Muito Alta

Tabela 13. Requisitos Não-Funcionais do Sistema

Capítulo 4 – Arquitetura e Especificação

Neste capítulo será feita uma especificação da solução a implementar. Primeiramente é apresentada uma arquitetura global do sistema e de seguida é feita a especificação de cada uma das partes integrantes.

4.1. Arquitetura Global do Sistema

Nesta secção é apresentada a arquitetura global do sistema seguido da especificação da arquitetura de cada componente.

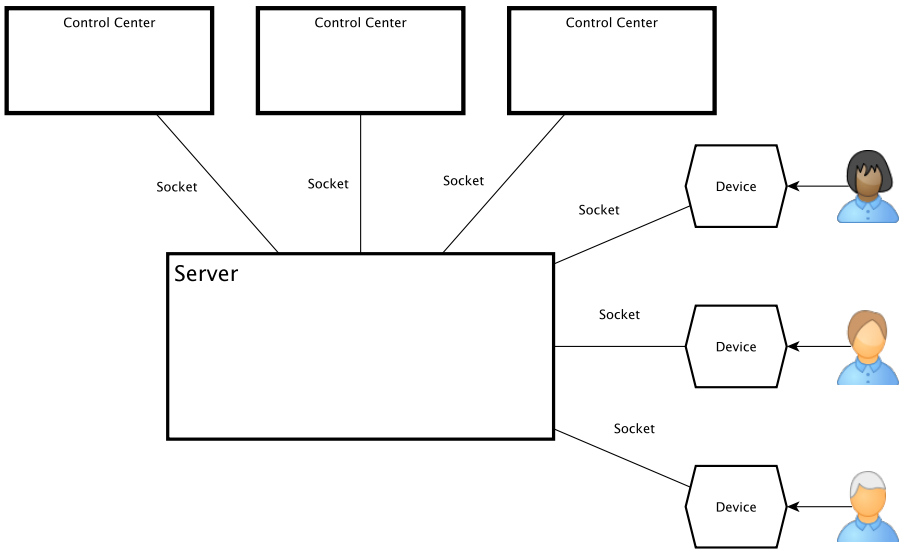

Figura 16. Arquitetura Global do Sistema

No desenho da arquitetura apresentado na Figura 16 podem ser identificadas as três principais partes integrantes do sistema: *Device*, *Server* e *Control Center*. Apresenta-se de seguida uma descrição de cada uma delas.

Device

O *Device* representa o dispositivo que incorpora os sensores que medirão os sinais vitais da pessoa que os utilize. Este dispositivo é responsável por enviar os dados que recebe para um micro-computador.

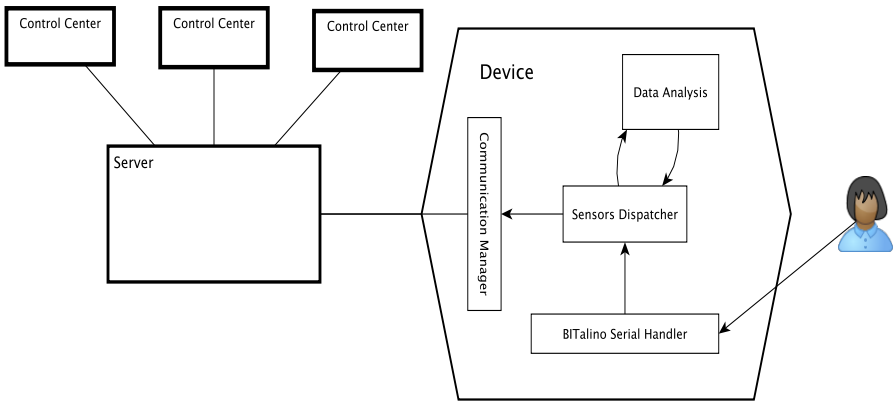

Figura 17. Arquitetura *Device*

Como se pode observar pela Figura 17, os dados vitais do portador do dispositivo são enviados pelo *BITalino* para um componente da aplicação intitulado de *BITalino Serial Handler*

que é o responsável por gerir todos os dados enviados pelo *BITalino* e para fazer o tratamento necessário antes dos dados serem enviados para o *Sensors Dispatcher*, componente responsável por fornecer os dados ao *Communication Manager* e ao *Data Analysis*. O *Communication Manager* é a classe responsável por gerir a comunicação do dispositivo com o servidor, tratando de enviar os dados para o socket inicializado aquando do emparelhamento inicial, seguindo assim para o servidor. O módulo de *Data Analysis* está incubido de analisar todos os dados no dispositivo para, caso haja alguma anomalia relativa a esses dados, enviar um alerta para o servidor. No âmbito do projeto SALUS, estes alertas irão passar por um *Message Broker*, que será detalhado no capítulo que relata a implementação.

Server

O *Server* é o módulo responsável por gerir comunicações, base de dados e pedidos assim como por processar dados.

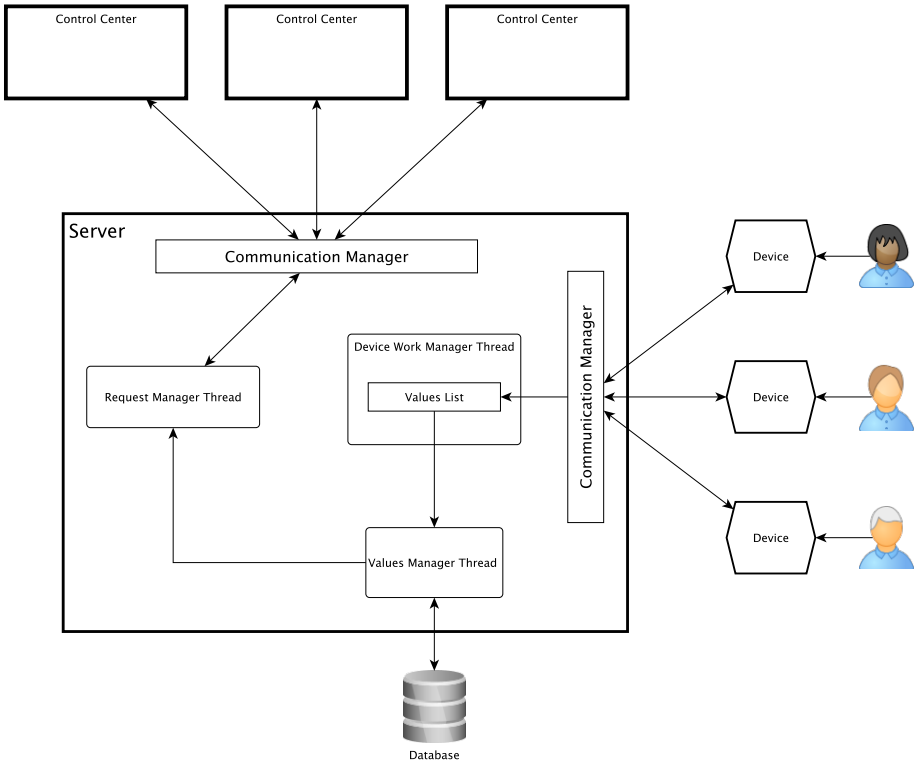

Figura 18. Arquitetura *Server*

Como se pode observar pela Figura 18, no servidor existem dois módulos para gestão das comunicações apelidados de *Communication Manager*, um para controlar as comunicações com os diferentes dispositivos e outro para gerir as ligações aos diversos centros de controlo. Depois de receber os dados vindos dos dispositivos, o gestor de comunicações envia os dados para uma *Thread* especifica que é responsável por tratar desses mesmos dados e enviá-los para uma *Thread* que os analisa. Esta última *Thread* (representada na imagem como *Values Manager Thread*) vai fazer a conexão à Base de Dados para obter a informação sobre quais os dispositivos que cada centro de controlo vai visualizar. Com esta informação será possível selecionar os dados que são necessários enviar. Estes dados são então enviados para uma outra *Thread* que se responsabiliza por agregar todos os dados, construir as mensagens e fornece-las ao gestor de comunicações dos centros de controlo para posterior envio e visualização.

Control Center

O *Control Center* é o módulo responsável por fazer a gestão das janelas correspondentes à visualização de vários pacientes ao mesmo tempo e de detalhe de um só paciente.

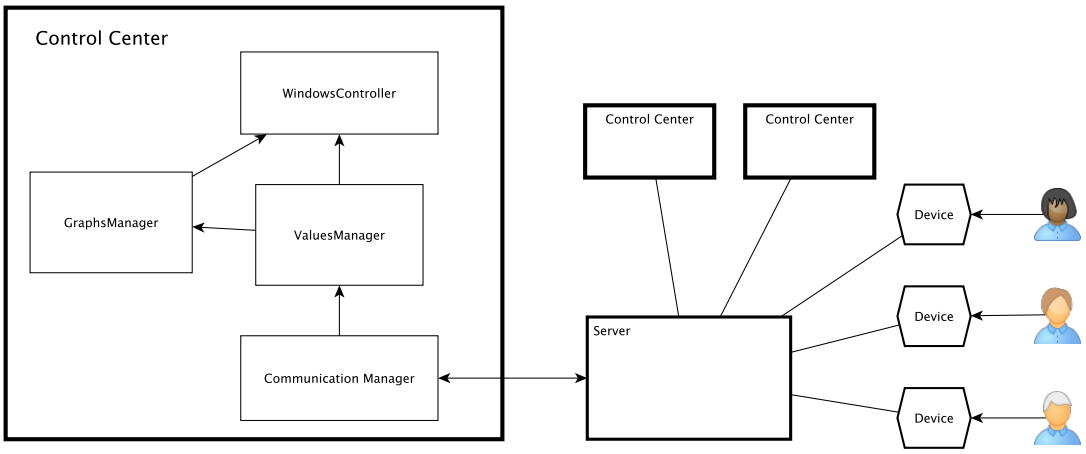

Figura 19.Arquitetura *Control Center*

Ao receber os dados dos sensores dos utilizadores configurados, é feito o direcionamento destes para um módulo que se intitulou de *Values Manager*. Este é o responsável por analisar os valores de cada sensor que compõem cada mensagem. Se for um tipo de dados que deve ser visualizado sob a forma de um gráfico em tempo-real (por exemplo, o ECG), são enviados para o *Graphs Manager*, responsável por fazer a gestão de todos os gráficos. Esta gestão inclui a criação e o preenchimento em tempo-real de todos os gráficos que sejam necessários. Por fim, existe o *Windows Controller* que vai fazer toda a gestão das janelas, atualizar os valores para casa sinal vital e tratar de toda a interação com o utilizador.

4.3. Comunicação

Nesta secção é apresentada uma solução para a comunicação entre os vários componentes do sistema, respondendo da melhor forma possível ao desafio relativo à sincronização apresentado no Capítulo 3. A comunicação entre os diversos componentes do sistema (PC-Servidor e Servidor-Centro de Controlo) é feita recorrendo a *Sockets*, apesar de terem sido identificadas outras hipóteses possíveis como Java RMI e REST usando JSON. Após a análise das vantagens e desvantagens de cada uma, foi decidido que a forma de comunicação utilizada seria *Sockets*. Em comparação com REST, *Socket* permite uma transferência de dados mais rápida para quantidades de dados significativas. Como se lida com uma grande quantidade de dados a ser transferida por segundo, o tempo de transferência é importante para se conseguir alcançar o menor atraso possível na leitura dos dados em tempo-real. Quando comparado com Java RMI, os *Sockets* podem possuir um *overhead* mais reduzido, uma vez que no Java RMI uma grande parte da mensagem é reservada para cabeçalhos e informação que não os dados que pretendemos enviar. Como no Java RMI é abstraído o tipo de protocolo ou o processo de autenticação, não é possível defini-los. Tendo em conta que o objetivo é construir uma aplicação o máximo optimizada possível, é conveniente poder definir os protocolos e os processos de autenticação para tentar que o *overhead* dos processos de troca de mensagens seja o menor possível.

4.3.1. Netty

O Netty foi a framework escolhida para operar e gerir todas as comunicações entre clientes (PC e Centro de Controlo) e servidor. Foi escolhido o Netty por ser uma framework baseada em Java e que permite uma gestão de *Sockets* de uma forma não-bloqueante (NIO, Non-blocking Input/Output), ou seja, um programa não necessita de bloquear/esperar quando escreve algo num *Socket,* funcionando assim com base em eventos. Isto é, quando existe algo para ser lido ou escrito, isso é feito com transparência para o sistema, não afetando em nada a execução do mesmo. Este factor é fundamental, tendo em conta que a quantidade de pedidos e respostas de clientes e servidores é significativa. De referir também que a API do Netty é compatível com vários tipos de protocolos de transporte e tem um modelo de *threads* personalizável.

Na sua estrutura interna, o Netty contém, do lado do servidor, um *NioServerSocketFactoryChannel* que utiliza o modo não-bloqueante, de forma a servir vários clientes eficientemente e que cria um *ServerSocketChannel* baseado nesse NIO.

Existem dois tipos de *threads* num *NioServerSocketFactoryChannel*:

- *Boss thread*: cada *ServerSocketChannel* possui a sua própria *boss thread*, isto é, se, por exemplo, forem abertas duas portas no servidor (8000 e 8080), irão existir duas *threads* deste tipo. Esta *thread* é responsável por aceitar as conexões ao servidor até que a porta seja desacoplada. Quando uma conexão é aceite com sucesso, a *boss thread* passa o *channel* aceite para uma das *worker thread* que são geridas pelo *NioServerSocketFactoryChannel*.

- *Worker thread*: podem existir uma ou mais *threads* deste tipo. Esta *thread* é responsável por executar as escritas e leituras não-bloqueantes para um ou mais *Channels* no modo não-bloqueante. Esta *thread* contém um *Selector* que é usado para registar os canais. O número de *threads* que podem existir deste tipo é configurável sendo por defeito duas vezes o número de *cores* do cpu.

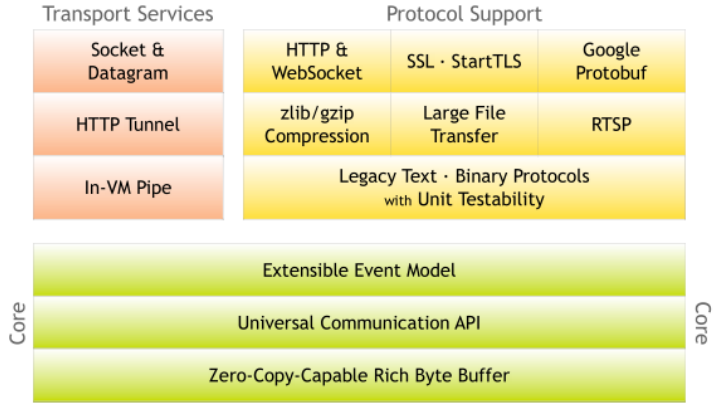

Figura 20. Netty Overview

Para cada *Channel* criado no servidor, e consequentemente para cada cliente, existe um *pipeline* responsável por conter os *handlers* necessários ao processamento das mensagens. Neste *pipeline* podem existir também um codificador e descodificador de mensagens. No sistema a implementar o *pipeline* de cada canal (tanto do lado do servidor como do lado do cliente) vai ter a estrutura apresentada:

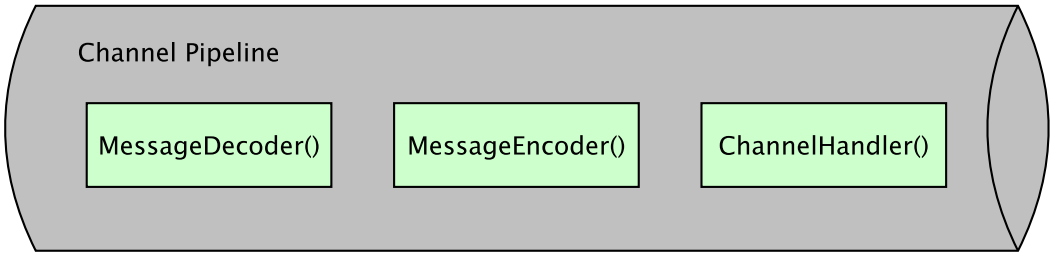

Figura 21. Channel Pipeline Netty

Como se pode observar pela Figura 21, existe primeiramente um descodificador de mensagens (*MessageDecoder*) que permite transformar a mensagem que chega, representada num conjunto de *bytes* (num *ByteBuf*), no objeto ao qual se refere. O *MessageEncoder* é

responsável pelo processo inverso ao do descodificador, codificando a mensagens para ser enviada aquando da execução de uma escrita, transformando um objeto num conjunto de *bytes*. Por fim, existe o *ChannelHandler*, onde serão tratadas as mensagens. Deve ser apenas verificado o tipo da mensagem e enviá-la para outro fluxo de execução responsável pelo real tratamento da mesma. Isto deve ser feito para evitar o aparecimento de exceções no *handler* e consequentemente no *pipeline*. Se alguma exceção ocorrer dentro do *pipeline*, o canal é fechado, não sendo possível captar ou obter informação sobre a exceção em causa e perdendo-se assim a ligação. Como explicado na documentação do *Netty*, a ordem de inserção destes *handlers* no *pipeline* tem de ser feita na ordem descrita na figura através do comando *addLast(MessageDecoder(), MessageEncoder(), ChannelHandler()).*

A framework Netty está muito bem documentada, incluindo um Javadoc detalhado, manuais de utilizador e vários exemplos que servem de apoio à implementação. Para além disso, alguns colaboradores da OneSource trabalharam com a framework num passado recente e têm, por isso, o conhecimento suficiente para fornecerem o apoio necessário.

4.3.2. Estrutura das Mensagens

A estrutura das mensagens sofre alterações ao longo do desenvolvimento do sistema, sempre sem busca da melhor performance possível. Começou-se por definir mensagens recorrendo a uma codificação análoga ao TLV e utilizando Byte Buffers. Por fim, acabou por se utilizar o Protobuf, um mecanismo de serialização de dados estruturados desenvolvido pela Google. Esta secção pretende explicar estas duas formas de estruturação das mensagens e trocar pelos vários componentes do sistema.

4.3.2.1. ByteBuffer - TLV

Para optimizar a troca de mensagens entre os três componentes do sistema foi escolhido em primeiro lugar o método de codificação de mensagens TLV (*Type*, *Length* e *Value*), onde as mensagens são representadas em conjuntos de bytes. Este método permite dividir uma mensagem em três partes:

Type: normalmente é representado por um código alfanumérico e que indica o tipo dos dados presentes na mensagem (TLV);

Length: indica o tamanho da mensagem, ou seja, quantos registos únicos têm de ser lidos;

Value: é uma série de bytes de medida variável que contém os dados para o TLV em que estão inseridos (estes dados podem ser eles próprios outro TLV).

Existem várias vantagens para o uso dos TLV:

- As mensagens TLV são facilmente codificadas e descodificadas no emissor e no receptor;
- Não existe uma ordem específica para a disposição dos vários TLV numa mesma mensagem.
- São usados no formato binário o que facilita o processo de análise e torna os dados mais pequenos.

De seguida são apresentados exemplos das estruturas das mensagens seguindo o método TLV, adaptado internamente para este sistema, em fase de protótipo, e que serviu para a troca de mensagens entre os diferentes componentes do sistema.

De *Control Center* para *Server* (Request)

Este request pode ser de vários tipos:

Tipo 1 – *Control Center* pede os dados dos sensores dos utilizadores que pretende que sejam visualizados no ecrã com todos os utilizadores.

TLV:

Type – Inteiro, representando o tipo da mensagem.
Length – Inteiro, representando o número de utilizadores a visualizar.
Value – Cada "value" é composto por um inteiro que representa o ID de cada utilizador.

Tipo 2 – *Control Center* pede os dados relativos a um utilizador específico para serem visualizados no ecrã de detalhe.

TLV:

Type – Inteiro, representando o tipo da mensagem.
Length – Neste tipo de mensagem, este campo não é usado tendo em conta que é sempre um utilizador.
Value – Inteiro que representa o id do utilizador em questão.

Tipo 3 – O *Control Center* pede ao *Server* os utilizadores que estão ativos e a enviar dados.

TLV:

Type – Inteiro, representando o tipo da mensagem.
Length – Este campo não é usado.
Value – Este campo não é usado.

De *Server* para *Control Center* (Packet)

O pacote pode ser de vários tipos e conter informação diversa:

Tipo 1 – *Server* envia os dados dos sensores relativos aos utilizadores que o centro do controlo pretende.

TLV:

Type – Inteiro, representando o tipo da mensagem.
Length – Inteiro, representando a quantidade de informações que compõem o pacote.
Value – Cada "value" é representado por um conjunto de três valores. Esses três valores são: um inteiro que representa o id do utilizador, um inteiro que representa o id do sensor e um double que representa o valor emitido pelo respectivo sensor.

Tipo 2 – *Server* envia dados para um utilizador especifico. Estes dados são enviados de duzentos em duzentos milissegundos.

TLV:

Type – Inteiro, representando o tipo da mensagem.
Length – Inteiro, representando o número de sensores ligados ao utilizador e a emitir dados.
Values – Cada value é composto por um TLV (TLV 2).

TLV 2:

Type – Inteiro, representando o id do sensor.
Length – Inteiro, representando o número de valores a ler para o sensor especifico.
Values – Cada "value" é composto por um Long que representa o timestamp do PC que está ligado ao utilizador e um Double que representa o valor emitido pelo sensor.

Tipo 3 – *Server* envia para o *Control Center* os dados dos pacientes ativos.

TLV:

Type – Inteiro, representando o tipo da mensagem.
Length – Inteiro, representando o número de utilizadores que estão ativos.
Value – Cada "value" é representado por um conjunto de cinco valores. Esses cinco valores são: um inteiro que representa o id do utilizador, um inteiro que indica o número de bytes a ler a seguir, os bytes a ler para formar uma String com o nome do utilizador, um char que representa o sexo do utilizador e um inteiro que representa a idade do utilizador.

4.3.2.2. Google Protobuf

O Protobuf, ou Protocol Buffer, é um método para serializar dados estruturados, como classes. Foi desenvolvido pela Google e é bastante útil para desenvolver sistemas que necessitem de comunicação entre diversos componentes. Foi por decisão interna do projeto que se resolveu atualizar o método de comunicação para este meio.

Existe primeiramente um ficheiro de terminação ".proto" onde é definida a estrutura das mensagens. Essa estrutura será depois convertida para classes em Java através de um compilador próprio criado pela Google para a linguagem Java. De seguida é apresentado um exemplo da utilização do *protobuf* no projeto.

```
message BaseMessage {
    required Type type = 1;
    required string idHash = 2;
    optional Authentication authMessage= 3;
    repeated SensorValue sensorValueMessage = 4;
    optional Command commandMessage = 5;
    optional Configuration configurationMessage = 6;
    repeated SensorInfo sensorInfoMessage = 7;
    repeated TextMessage textMessage = 8;
    enum Type {
        AUTHENTICATION = 1;
        SENSOR = 2;
        COMMAND = 3;
        NOTIFY = 4;
        CONFIGURATION = 5;
        INFO = 6;
        MESSAGE = 7;
        PING = 8;
    }
}

message SensorValue {
    required Type type = 1;
    repeated DataPoint value = 2;
    optional DeviceData dData = 3;
    enum Type{
        ECG = 1;
        EMG = 2;
        EDA = 3;
        TEMP = 4;
        RESPRATE = 5;
        LIGHT = 6;
        PULSEO2 = 7;
        MOTION = 8;
        GPS = 10;
        DEADDEVICE = 11;
        HR = 12;
        RR = 13;
        DEVICEINFO = 14;
    }
}

message DeviceData{
    optional double cpu = 1;
    optional double memory = 2;
    optional int64 uptime = 3;
    optional double battery = 4;
    optional string networkType = 5;
    optional double wirelessSignalStrength = 6;
    optional double cellSignalStrength = 7;
}

message DataPoint {
    required int64 time = 1;
    optional double value = 2;
    optional double motionX = 3;
    optional double motionY = 4;
    optional double motionZ = 5;
    optional double latitude = 6;
    optional double longitude = 7;
}
```

Figura 22. Protobuf

Nesta figura estão representadas as quatro principais estruturas do tipo *message*. Todas as restantes são apresentadas no

Anexo B – Modelo de Mensagens Protobuffer.

DataPoint será a estrutura que representa cada valor de cada sensor. Nele estão os seguintes atributos: time, que se refere ao instante de tempo em que o valor foi captado e que é definido como *required*, o que significa que é obrigatório este valor estar preenchido; *value*, onde é armazenado o valor *double* captado extraído do sensor; *motionX*, *motionY* e *motionZ* que são utilizados para armazenar as coordenadas X, Y e Z quando se trata do sensor acelerómetro; latitude e longitude que servem para guardar os valores do sensor de GPS. Todos estes valores, excepto o instante de tempo, são definidos como *optional* para permitir o preenchimento daqueles que forem estritamente necessários para um determinado sensor.

DeviceData é a estrutura usada para enviar informações diversas relativas ao dispositivo. Todos os seus atributos são opcionais para possibilitar que se envie apenas a informação que se pretende a cada instante. Entre as possíveis informações estão o *cpu* que apresenta a percentagem de processamento utilizada, a *memory* que representa a quantidade de memória que está a ser ocupada, o *uptime* que se refere ao tempo que passou desde que o dispositivo se conectou, a *battery* que apresenta a percentagem de bateria do dispositivo, *networkType* que indica qual o tipo de rede que está a ser utilizado pelo dispositivo (pode ser, por exemplo, "*wi-fi*" ou "4G"), o *wirelessSignalStrength* que representa a força do sinal sem fios e o *cellSignalStrength* que indica a força do sinal da rede móvel.

SensorValue é a estrutura que representa cada sensor e o seu conjunto de valores para um determinado período de tempo. Cada *SensorValue* contém um *type* que define qual é o sensor que está a ser representado. Este tipo é obrigatório e está identificado por uma enumeração que associa cada inteiro a um determinado sensor. Existe também um *value* que é do tipo *DataPoint* e que, por sua vez, está definido como *repeated*, ou seja, pode existir mais do que um *DataPoint* em cada *SensorValue*. Existe também um bloco *dData* do tipo *DeviceData* que serve para enviar a informação do dispositivo quando assim se pretender.

BaseMessage é a estrutura da mensagem principal, ou seja, daquela que serve efetivamente para comunicar, a que será enviada através do *socket*. Para diferenciar as mensagens entre si, existe um atributo *type* do tipo *Type*, uma enumeração que indica qual o tipo da mensagem que está a ser enviada. Esse tipo pode variar, por exemplo, entre uma mensagem de autenticação ou de de configuração até uma mensagem do tipo sensor que representa os dados que são captados do dispositivo. O outro campo obrigatório que existe é o *idHash* que identifica o dispositivo que está a enviar a mensagem. Todos os restantes campos são opcionais para poder preencher de acordo com o tipo da mensagem que está a ser enviada. O *authMessage* é do tipo autenticação e serve para as mensagens desse tipo. O *sensorValueMessage* é do tipo *SensorValue* e pode ser repetido tendo em conta que podem ser enviados dados de vários sensores numa mesma mensagem. A *commandMessage* é do tipo *Command* e serve para enviar mensagens desse mesmo tipo, usadas para os *Control Center* pedirem ações aos dispositivos (por exemplo, iniciar e parar vídeo e voz). A *configurationMessage* serve para o servidor enviar mensagens de configuração aos dispositivos, que servem para definir, por exemplo, de que sensores e com que frequência determinado dispositivo deve enviar os seus dados. A *sensorInfoMessage* é do tipo *SensorInfo* e permite enviar informação referente aos sensores. Por fim, existe uma *textMessage* que serve para enviar um tipo variado de mensagens do tipo *MESSAGE*.

4.4. Base de Dados

O sistema de gestão de Base de Dados escolhido para o desenvolvimento foi o MySQL. Foi escolhida a abordagem que permitisse uma maior rapidez de implementação. Tendo em conta que, para uma base de dados não relacional, a curva de aprendizagem seria grande, era necessário despender bastante tempo, primeiro para a aprendizagem e de seguida para os testes que permitissem perceber se seria uma melhor solução que uma base de dados relacional. No entanto, este estudo reveste-se de grande interesse e será tido em conta num trabalho futuro deste sistema.

Segue um esquema do modelo ER definido para o sistema em causa e a descrição de cada uma das tabelas:

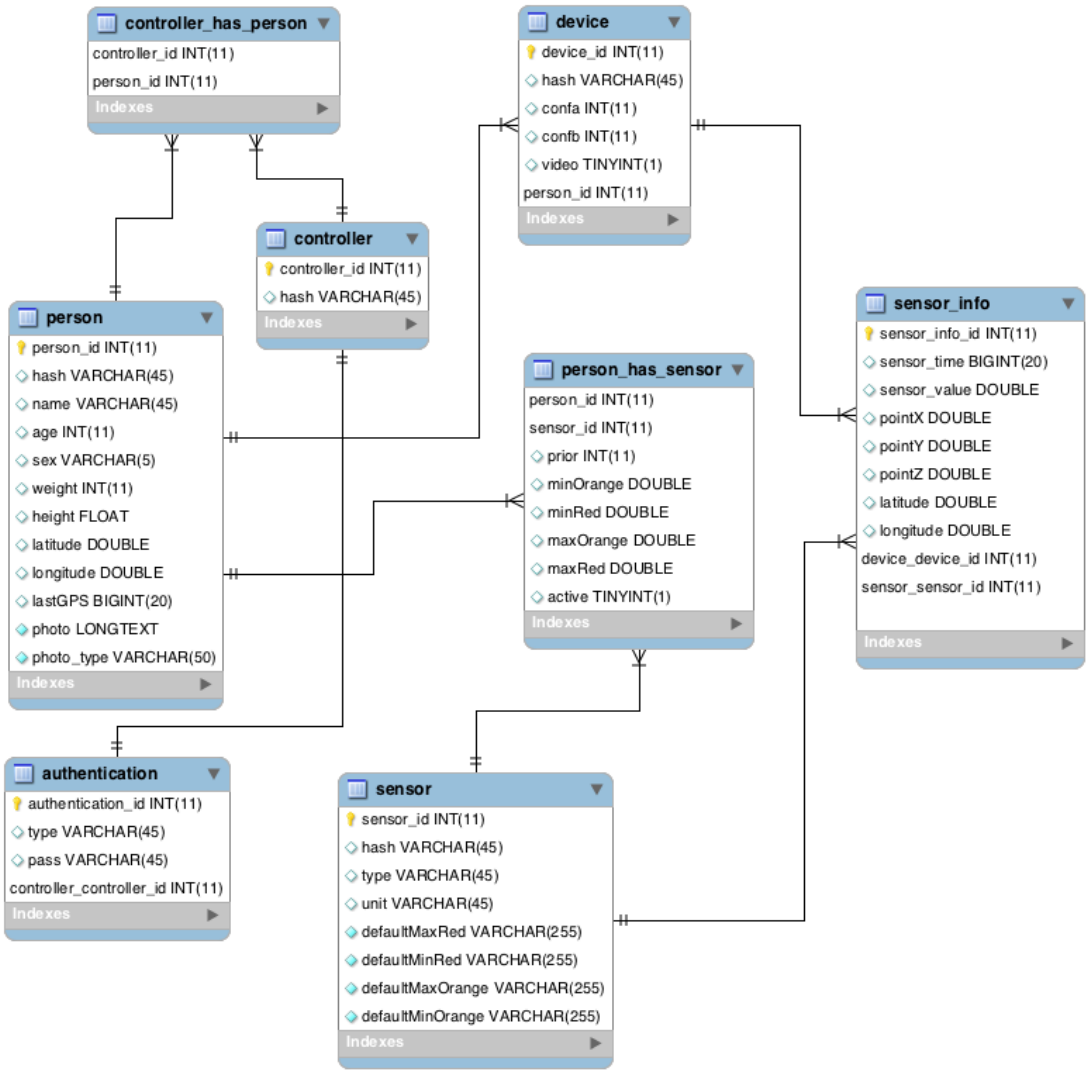

Figura 23. Modelo Base de Dados

person: esta tabela representa a pessoa que irá transportar o dispositivo com sensores. Tendo em conta os requisitos apresentados no capítulo 3, está incluída nesta tabela informação que vai desde o nome e idade até à fotografia ou a altura e peso.

controller: esta tabela representa cada um dos centros de controlo e apenas necessita de dois dados, um identificador e uma *hash* para a autenticação no sistema.

controller_has_person: tabela que representa a ligação *Many to Many* entre centros de controlo e pessoas, controlando que pessoas cada centro de controlo está configurado para ver.

authentication: tabela usada com o intuito de autenticar o acesso ao sistema dos centros de controlo.

sensor: tabela que armazena os dados de cada um dos sensores. Nela estão contidos os valores máximos e mínimos admitidos para cada um dos sinais vitais e indispensáveis para disparar alertas no sistema.

person_has_sensor: relaciona os sensores com as pessoas, indicando quais os sensores conectados a cada individuo.

device: tabela que se refere às configurações de cada dispositivo. Contem informação como o identificador único, uma *hash*, duas configurações e uma *flag* que indica a existência de vídeo.

sensor_info: tabela onde são armazenados todos os dados vindos dos sensores. Para isto é associado um determinado dispositivo representado pelo seu identificador e o mesmo acontece para o sensor a que se refere o valor. É também armazenado um timestamp que corresponde ao instante em que foi capturado o dado relativo a um sinal vital e o seu valor, representado por um *double*. Se for um dado relativo ao sensor acelerómetro, existem três valores *double* para armazenar a coordenada X, Y e Z desse sensor. Existem também os valores de latitude e longitude para o caso do sensor ser o GPS.

4.4.1. Performance

Este modelo de base de dados apresentado demonstrou um problema de performance. Tendo em consideração que cada dispositivo pode enviar dados de mais do que um sensor com frequências que podem ascender aos 250 pontos por segundo, a quantidade de inserções na base de dados, em particular na tabela *sensor_info* será muito significativo. Isto trouxe problemas na velocidade de inserção de registos tendo em conta que a cada iteração era necessário que o motor de base de dados verificasse a integridade dos dados, confirmando todas as relações existentes entre tabelas.

Para evitar este problema, foram retiradas as relações existentes da tabela *sensor_info*. Desta forma, e tendo controlo sobre o resto da base de dados, foi garantida internamente a integridade de todos os dados inseridos.

Tendo isto em conta, a tabela *sensor_info* ficou com a seguinte estrutura:

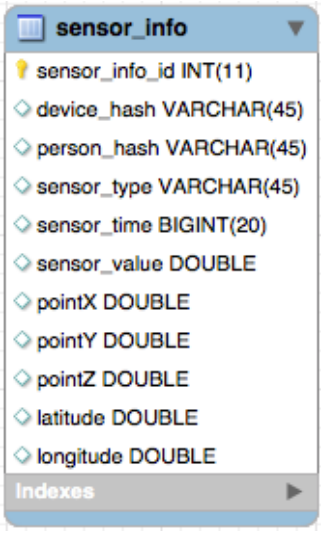

Figura 24. Tabela sensor_info

Como se pode observar pela Figura 24, foi adicionado à tabela dois atributos: *device_hash* e *person_hash* que identificam assim a pessoa e o dispositivo a que o registo do sensor se refere.

4.5. Framework de Desenvolvimento

Durante esta fase de especificação foi necessário escolher qual a *framework* aplicacional Java a utilizar. Tendo em conta a experiência por parte dos elementos da empresa que faziam parte do projeto e analisando algumas características base necessárias (gráficos de linhas 2D, por exemplo), ficou decidido que a *framework* a utilizar seria o *JavaFX*. Esta *framework*, além de possibilitar o desenho de gráficos 2D com uma biblioteca própria, é a mais recente no mercado e

está projetada para ser a substituta da mais utilizada até aos dias de hoje, o Swing, e por isto prevê-se que seja a mais utilizada no que toca a linguagem Java.

Outro factor importante é o facto de esta *framework* possibilitar uma metodologia de desenvolvimento análoga ao *MVC (Model View Controller)*. Para controlar toda a interação com o utilizador, ou seja, tudo o que possa gerar um *output* visível para o utilizador são usadas as *views*. No *JavaFX* estas *views* podem ser definidas recorrendo a ficheiros *FXML*. O *Model* pode ser identificado como as classes que controlam o conteúdo das janelas e do que as compõem e o *Controller* pode ser definido pelas classes que contêm toda a lógica do programa.

4.6. Prototipagem

Nesta secção são apresentados os protótipos referentes ao sistema. Estes protótipos refletem o trabalho que foi realizado em termos de prototipagem e que coincidiu com o término do primeiro semestre de trabalho.

4.6.1. Protótipo *Control Center*

Numa primeira parte, é apresentado o protótipo do ecrã que mostra em detalhe todos os dados referentes a um único utilizador/dispositivo. Na segunda parte é mostrado um protótipo do ecrã onde é possível monitorizar vários utilizadores ao mesmo tempo. É apresentado também um protótipo que serviu de alternativa para as duas janelas e que contém toda a informação (tanto de detalhe como com vários utilizadores ao mesmo tempo) numa só janela. Esta última solução acabou por ser a adotada para o produto final.

4.6.1.1. Utilizador em Detalhe

Nesta subsecção é apresentado um protótipo do *Control Center*, mais especificamente do ecrã que mostra as informações de um utilizador em detalhe. Este ecrã é composto pelas informações pessoais e pela medição dos dados vindos de um dispositivo. Estes dados podem ser apresentados na forma de valor ou através de um gráfico.

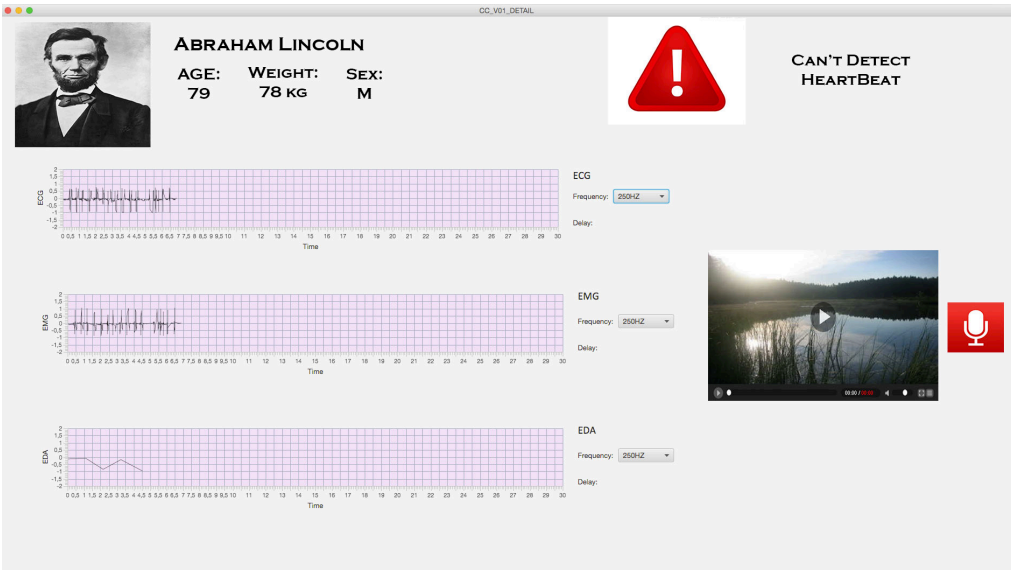

Figura 25. Protótipo *Control Center* – ecrã detalhe

A Figura 25 representa um protótipo do ecrã dos dados em detalhe de um dispositivo. Neste é possível ver o nome do utilizador. No canto superior esquerdo é suposto vir a aparecer a fotografia do utilizador. Do lado direito do ecrã onde se encontra "*Video*" é suposto aparecer a secção de vídeo e opções para o mesmo *stream* de vídeo e voz.

Os dois primeiros gráficos correspondem a dois sensores distintos e recebem dados a uma frequência de 250 pontos por segundo. O terceiro gráfico corresponde a um exemplo de um

sensor em que a sua frequência de amostragem é de um ponto por segundo . O eixo dos yy de cada gráfico representa o valor numérico que é debitado por cada sensor. O eixo dos xx representa o tempo em segundos. Neste protótipo, este tempo está apresentado em segundos, sendo o segundo "0" o início da leitura (momento em que o utilizador em questão foi selecionado).

Do lado direito de cada gráfico está o nome do sensor em questão, um *drop-down* onde poderá ser escolhida a frequência e um campo para ser apresentado o atraso relativo à apresentação dos dados. O campo onde é escolhida a frequência deixou de existir numa versão final, sendo que o servidor irá receber os dados à frequência óptima, ou seja, a melhor frequência possível tendo em conta a qualidade de rede em cada momento.

De notar que esta é uma versão inicial que foi implementada com o principal propósito de testar o desempenho dos gráficos do Java FX para apresentação de dados em tempo real.

4.6.1.2. Vários utilizadores em simultâneo

Nesta secção é demonstrado um protótipo inicial do ecrã onde é possível monitorizar vários utilizadores ao mesmo tempo.

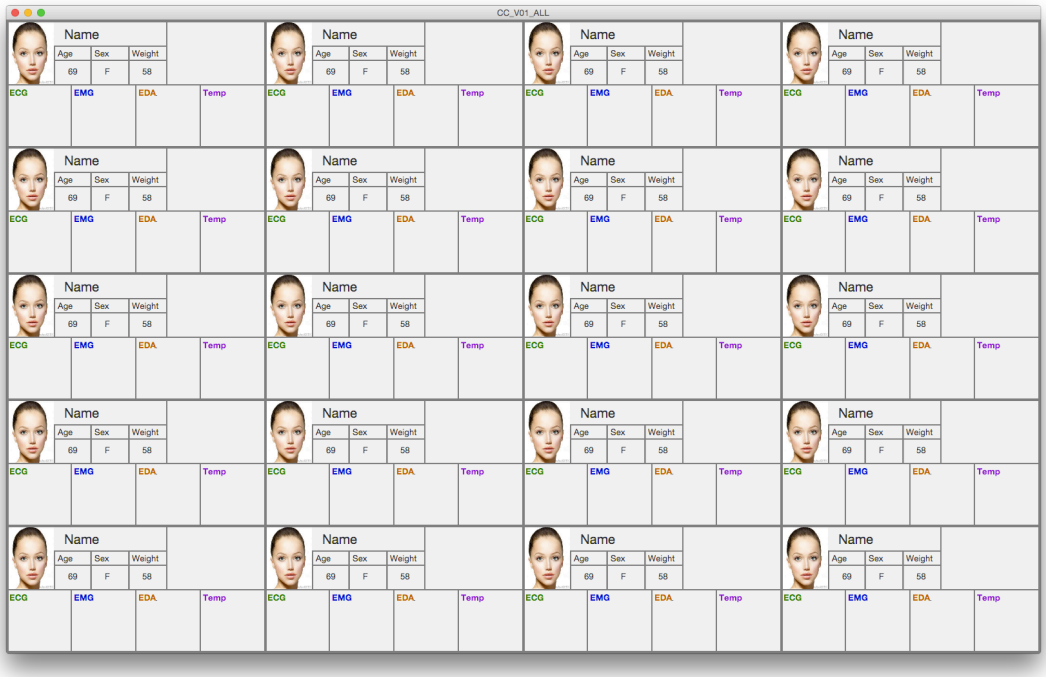

Figura 26. Protótipo *Control Center* - ecrã vários utilizadores

A Figura 26 representa o protótipo do ecrã onde é possível monitorizar vários utilizadores ao mesmo tempo. A tela é dividida em vinte secções (rectângulos) com as mesmas dimensões. Neles encontra-se uma foto do utilizador, o nome e três dados pessoais como idade, sexo e peso. Do lado direito destas informações, encontra-se uma secção vazia que servirá para disparar alertas quando estes sucederem. Na parte inferior de cada paciente, encontram-se quatro secções (quadrados) que representam quatro sensores distintos que estão ligados ao utilizador. É supoo que os valores apareçam no interior dessas secções e sejam atualizados a cada segundo. Do lado do nome do utilizador encontra-se também um botão "*Detail*" que ao ser clicado seleciona este paciente para ser monitorizado com mais detalhe no ecrã demonstrado na subsecção anterior.

4.6.1.3. Protótipo alternativo

No decorrer da implementação deste protótipo inicial, surgiu a ideia da construção de um protótipo onde as duas janelas supra referidas se unissem numa só. Nesta secção é apresentado esse mesmo protótipo.

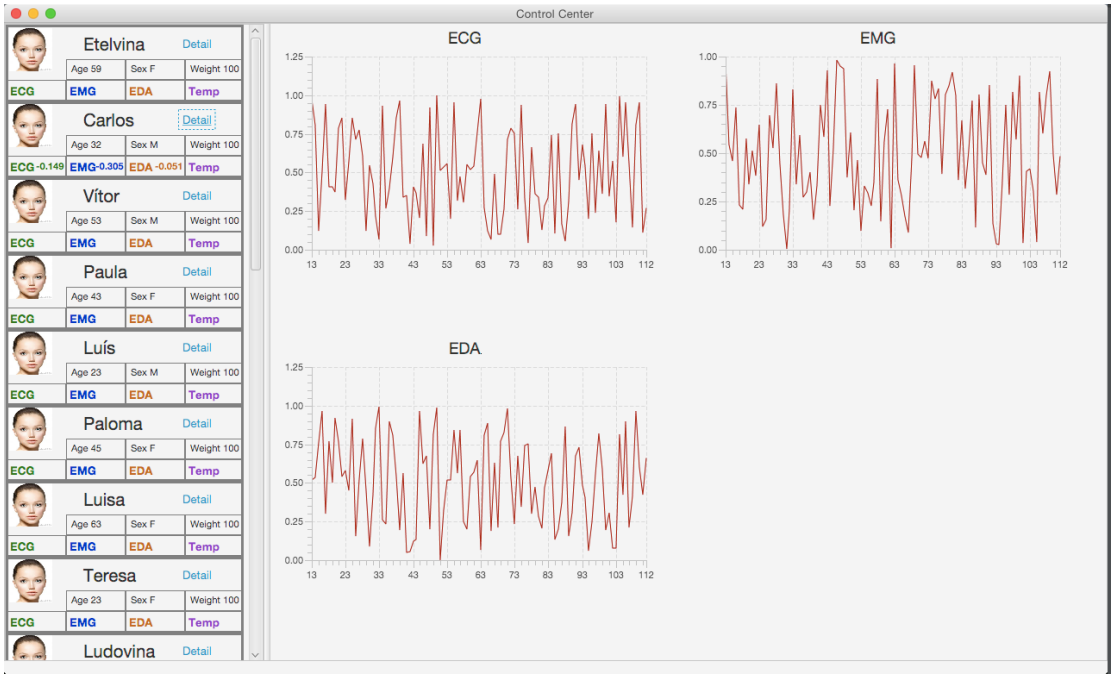

Figura 27. Protótipo *Control Center* - alternativa

A Figura 27 demonstra o possível protótipo alternativo. Do lado esquerdo da janela é acondicionada, numa lista vertical, a janela que monitoriza vários utilizadores ao mesmo tempo. Ao ser selecionado um deles, a secção da parte direita da janela é preenchida com a informação detalhada e a monitorização em tempo-real.

Esta janela mostra-se como sendo uma alternativa à solução inicial. Depois de se submeter as duas soluções a testes de usabilidade e a revisões no seio do projeto SALUS, ficou decidido e especificado que a solução tinha como base apenas uma janela que acondicionava tanto a monitorização de vários utilizadores ao mesmo tempo como a monitorização de um utilizador em detalhe.

Capítulo 5 – Implementação

Neste capítulo é apresentado o trabalho de implementação para satisfazer os requisitos identificados no realizado no âmbito do estágio curricular. Será apresentado o trabalho em cada um dos componentes do sistema (*Control Center*, Servidor e Dispositivo), dando foco ao trabalho realizado pelo estagiário, integrando com toda a implementação do projeto. Toda esta fase de implementação foi coordenada pelo Engenheiro Luís Cordeiro, havendo um responsável interno por cada uma das componentes que formam o sistema.

Para clarificar o que se pretende alcançar nesta fase de implementação é apresentado de seguida um diagrama de atividade onde estão englobados os três componentes do sistema e ambos os utilizadores (cliente do *Control Center* e o utilizador do *Device*).

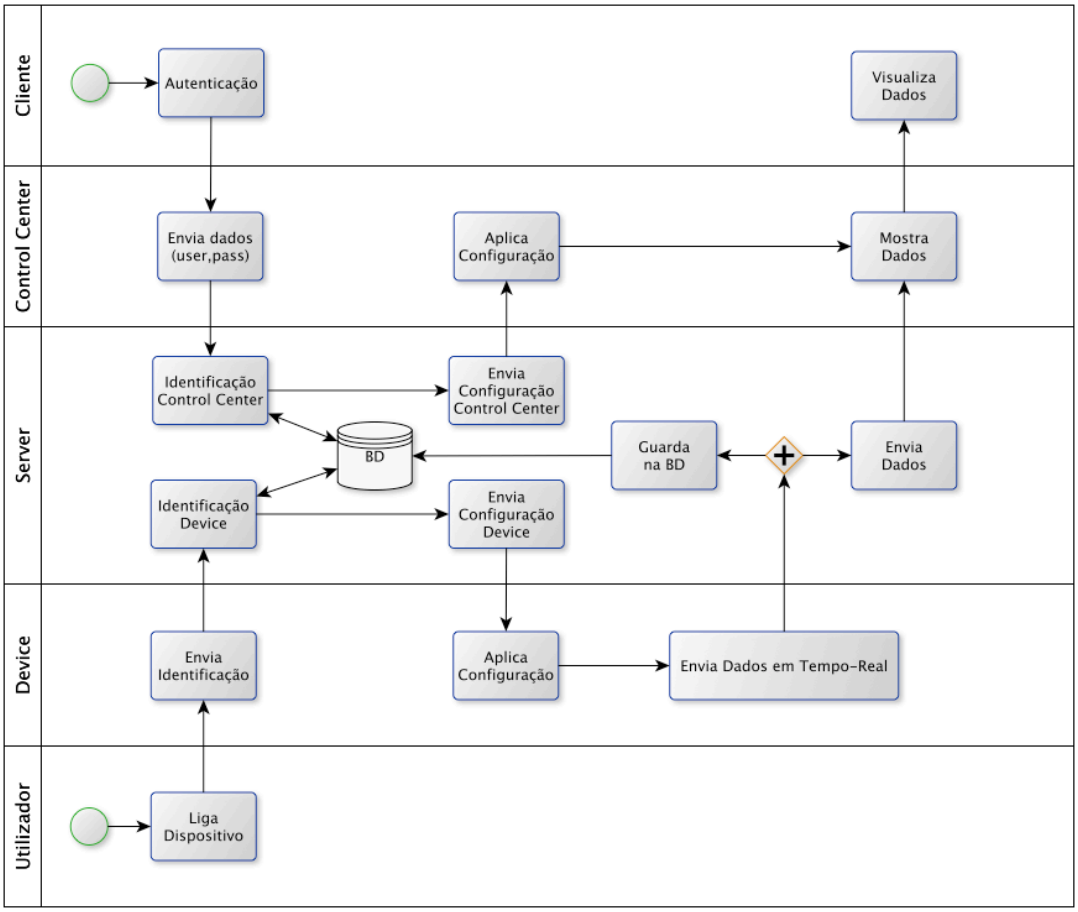

Figura 28. Diagrama de atividade

Do lado do *Control Center*, a atividade é despoletada pela autenticação por parte do cliente e envio de dados para o servidor por parte do *Control Center*, o que, ao nível do servidor, leva à identificação do *Control Center* em questão, recorrendo a uma ligação à Base de Dados. Após esta identificação, é retornado pela Base de Dados uma configuração própria daquele *Control Center*. Essa configuração é então enviada para o *Control Center*. Depois de receber a configuração, que consiste nos utilizadores a visualizar por aquele *Control Center*, são preparados os dados para apresentar ao cliente. Por fim esses dados são mostrados e o cliente pode assim visualizá-los.

Por outro lado, o utilizador despoleta a atividade ligando o dispositivo. O *Device*, por sua vez, envia a identificação do dispositivo. Recebendo a identificação, o servidor conecta com a base de dados e, depois de autenticar o dispositivo, recebe a configuração relativa a este e envia-a. Esta configuração tem informações como a pessoa que o dispositivo está a controlar, quais os

sensores e as portas que tem de ativar. Ao receber esta configuração, o dispositivo aplica-a e começa a enviar os dados para o servidor. No servidor os dados tomam dois caminhos em simultâneo: são enviados diretamente para os *Control Center* e ao mesmo tempo são colocados na Base de Dados para efeito de histórico.

5.1. *Control Center*

Nesta secção é apresentada a forma como o desenvolvimento do projeto na componente do *Control Center* decorreu, focando os aspetos técnicos mais relevantes. Como foi referido no Capítulo 1, o trabalho do estagiário nesta componente ficou centrado em dois principais campos: a visualização em tempo-real de dados através de gráficos e a gestão dos alertas. Esta componente esteve sob a responsabilidade de Pedro Borges.

De seguida, são apresentadas e descritas imagens do *Control Center* resultantes da implementação dos requisitos definidos.

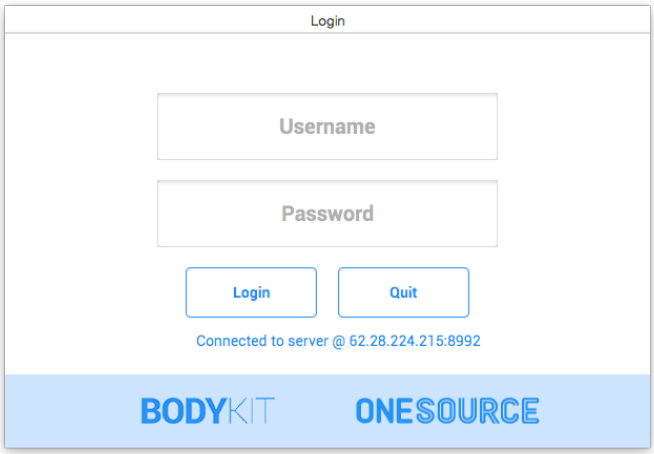

Figura 29. *Control Center* - Ecrã de login

Esta figura representa o ecrã de login da aplicação. Tem um campo para colocar o username e outro para a password. Através do botão Login é dada a entrada no sistema e feita a autenticação. Este ecrã surge como resultado da implementação do requisito com o código **REQ_CC_01**. Nota: para fazer referência ao requisitos nesta secção são apresentados apenas os números dos códigos, ficando pressuposto que o código se inicia em "REQ_CC".

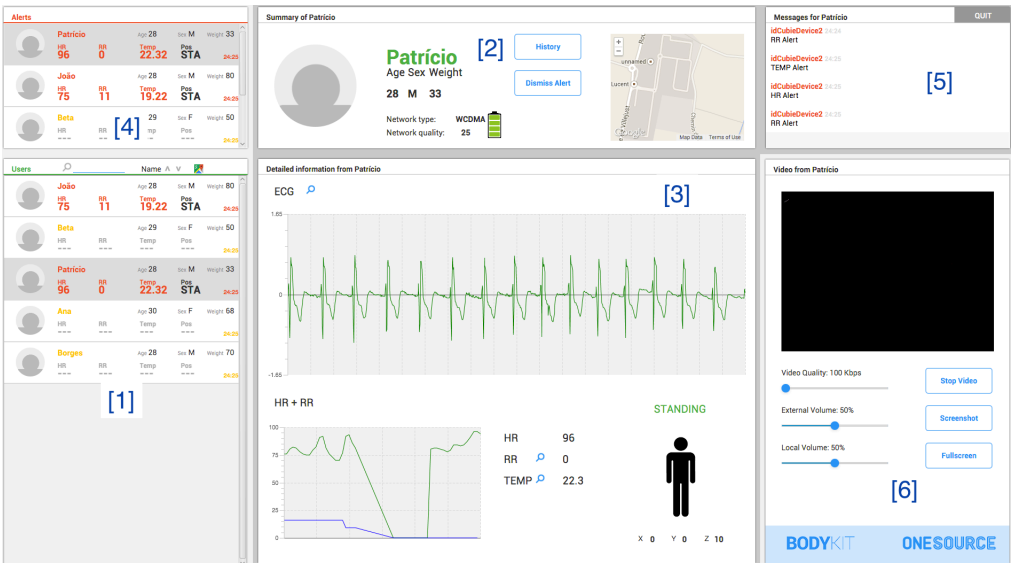

Figura 30. *Control Center* – Dashboard

A figura anterior apresenta a *dashboard* implementada para o *Control Center*. Esta apresentação divide-se em várias secções que estão representadas na imagem com os números de um a seis.

(1) Listagem de utilizadores – esta secção serve para apresentar a lista de utilizadores de dispositivos que este *Control Center* tem configurados para visualizar. Como se pode observar, cada entrada na lista corresponde a um utilizador e tem informação como a fotografia, o nome, idade, sexo e peso assim como valores para o batimento cardíaco (retirado da análise do ECG) e para os sensores de taxa de respiração, temperatura e acelerómetro (posição). No topo da lista pode-se pesquisar por utilizador ou ordenar a lista por nome.

Requisitos implementados nesta secção: 03, 04, 06-13, 44-46.

(2) Informação sumária sobre utilizador – esta secção contém um sumário sobre o utilizador que está a ser monitorizado em detalhe. Contém uma fotografia e informação pessoal. É possível observar também o tipo de rede, a qualidade desta e um indicador da bateria do dispositivo. Para além disso, é aqui que poderemos ver a localização do utilizador e aceder ao histórico de dados. Ao pressionar o botão "history" é apresentada a seguinte janela:

Figura 31. *Control Center* – Histórico

Requisitos implementados nesta secção: 05, 14-17, 40-42.

(3) Informação detalhada sobre utilizador – é nesta secção que se encontram os gráficos dos sensores a monitorizar para um determinado utilizador. Contém os valores e gráficos para os sensores de ECG, batimento cardíaco, taxa de respiração e acelerómetro. A posição do utilizador é apresentada recorrendo a uma figura, como se pode ver no canto inferior direito.

Requisitos implementados nesta secção: 05, 18-23, 26.

(4) Alertas – secção do ecrã reservada à apresentação dos alertas. É apresentada uma lista com os alertas disparados e cada entrada dessa lista é um utilizador que disparou um alerta. Os valores dos sensores que apresentam problemas são colocados a vermelho ou laranja, consoante a gravidade do problema ocorrido.

Requisitos implementados nesta secção: 27,28,36-39.

(5) Mensagens – esta secção é reservada para mensagens diversas que possam aparecer, como por exemplo, o aviso de que houve um alerta. Esta notificação é complementada com a informação mais detalhada na secção de alertas descrita anteriormente

Requisitos implementados nesta secção: 47.

(6) Integração voz e vídeo – esta secção representa a integração com a transmissão de sinal de voz e vídeo. Como se pode observar existe um quadrado preto onde aparece o sinal de vídeo bem como três *sliders* para controlar a qualidade de vídeo e o volume de áudio. Existem também três botões para começar/parar o vídeo, tirar um *sreenshot* e colocar o vídeo em *fullscreen*.

Requisitos implementados nesta secção: 29-35, 43, 48.

Nesta componente do sistema, o trabalho do estagiário esteve focado nas secções 2, 3 e 4. Trabalho esse que é detalhado de seguida.

5.1.1. Visualização de dados em tempo-real

A visualização de dados em tempo-real, em gráficos, revelou-se desde inicio como um desafio. A principal preocupação é a de conseguir o menor atraso possível entre o envio dos dados pelo dispositivo para o servidor e a visualização dos mesmos no *Control Center*. No caso específico do *Control Center*, é fundamental que o intervalo entre a chegada dos dados provenientes do servidor e a apresentação destes seja o menor possível.

Na fase de desenho do protótipo, este fator foi o principal foco no desenho do ecrã de detalhe. Foi necessário perceber qual a melhor forma de conseguir este objetivo.

O *JavaFX* contém uma biblioteca de gráficos interna personalizável que contém um gráfico de linhas intitulado *LineGraph*. Os pontos que preenchem o gráfico estão contidos numa estrutura de dados (pares X,Y) chamada *DataSeries*. Numa primeira abordagem, à medida que os valores iam chegando, eram colocados no *DataSeries* os valores em intervalos de 200 milissegundos. No início do processamento, a *Thread* que trata do desenho do gráfico fica em espera (*sleep*) durante 500 milissegundos (os 300 milissegundos a mais que o intervalo de 200 milissegundos definido prendem-se com uma espera extra para garantir que chegaram dados, ou seja, para compensar o tempo de transmissão). Depois disto, a *DrawThread* verifica se existem dados novos para apresentar e, caso hajam, coloca os valores correspondentes aos últimos 200 milissegundos no *DataSeries* e são automaticamente apresentados no gráfico. A partir daqui o ciclo repete-se: a *DrawThread* fica em espera durante 200 milissegundos e apresenta os dados que chegaram nesses 200 milissegundos que passaram. À medida que os pontos iam sendo adicionados ao *DataSeries*, iam sendo retirados os pontos que chegaram em primeiro lugar.

Após vários testes a esta solução, ficou patente um problema: com o decorrer do tempo o atraso na apresentação dos dados ia aumentando. Foi necessário perceber qual era o problema e procurar uma solução.

Depois de analisar o problema, verificou-se que este ocorria devido ao facto de a função *Thread.sleep(200)* (função usada para fazer com que a *Thread* fique em espera durante um período de tempo, neste caso, 200 milissegundos) não corresponder exatamente ao tempo que se define. Uma espera de 200 milissegundos pode não corresponder exatamente a esse tempo e pode ter uma variação até 10 milissegundos. Isto deve-se ao facto do método *sleep* não ser preciso e depender muito da máquina onde está a ser executado. Este problema afetou o desempenho do gráfico no desenho em tempo-real. Ao apresentar os dados que eram supostamente de 200 milissegundos, eram apresentados no gráfico como sendo, por vezes, referentes a 210 milissegundos, o que torna os dados pouco fiáveis, o que no caso deste sistema se figurava como um problema delicado. Com o decorrer do tempo o atraso vai aumentando. Por exemplo, ao fim de dois segundos já existiria um atraso médio de 100 milissegundos e ao fim de dez segundos já existia um atraso médio de meio segundo e assim sucessivamente.

Outro dos problemas para a baixa performance na produção dos gráficos foi a inserção direta no *DataSeries* e o preenchimento automático a partir deste, que criava uma baixa de performance e o aumento do atraso.

Para solucionar o primeiro problema, foi desenvolvida uma função que ajusta o tempo de espera da *Thread* consoante o tempo que realmente passou. Desta forma é possível controlar e contornar este problema.

Para solucionar o outro problema identificado foi utilizado no desenvolvimento a interface *Animation,* fornecida pelo JavaFX. Esta interface possui outra interface chamada *KeyFrame*. Assim, quando se pretende adicionar valores ao gráfico, inicializa-se uma *KeyFrame,* com a duração definida e regulada pelo método descrito no parágrafo anterior, responsável por aceder à estrutura onde os dados que chegam ao *Control Center* são colocados e colocar esses dados da *DataSeries*. Desta forma garante-se a separação das operações: receber os dados que chegam é da responsabilidade de uma *Thread* enquanto que toda a gestão do gráfico é feita por outra *Thread* recorrendo a *KeyFrames.*

5.2. *Server*

Nesta secção é apresentado o desenvolvimento que foi feito na componente do Servidor. Este trabalho teve como responsável o Hugo Fonseca. As tarefas do estagiário foram: definição do modelo de base de dados e a gestão de todas as comunicações.

De seguida é apresentada uma estrutura relativa à implementação desta componente.

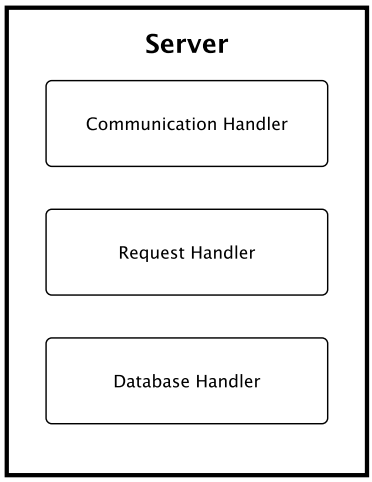

Figura 32. Estrutura *Server*

Como se pode observar pela figura, existem três principais divisões lógicas dentro do servidor: um gestor de comunicações, um tratador de pedidos e um gestor da base de dados e suas comunicações.

Comunnication Handler – divisão responsável por tratar das comunicações com os *Device*s e os *Control Center*. É aqui que é inicializado o *Netty*, descrito no capítulo 4, e são recebidas e encaminhadas as mensagens para serem posteriormente tratadas.

Request Handler – é nesta divisão que são tratadas as mensagens que chegam ao servidor.

Database Handler – divisão responsável por tratar de todas as comunicações e operações com a Base de Dados.

O estagiário esteve envolvido num processo de implementação transversal a todas as divisões do servidor, ou seja, desde a chegada de mensagens ao servidor até ao seu tratamento, armazenamento de dados e envio.

5.2.1. Gestão das comunicações

A gestão das comunicações tem por base a inicialização da *framework Netty*, descrita no capítulo 4. É necessário indicar o endereço IP e a porta onde os dispositivos se poderão conectar. Depois de definidos estes parâmetros são inicializadas a *Boss* e a *Worker Threads* e são associados os diferentes *Handlers*.

Depois desta fase de inicialização dos *Sockets* e ao conectar-se tanto um dispositivo como um *Control Center*, é inicializado o processo de recepção de mensagens. Este processo é apresentado através do diagrama seguinte e explicado de seguida.

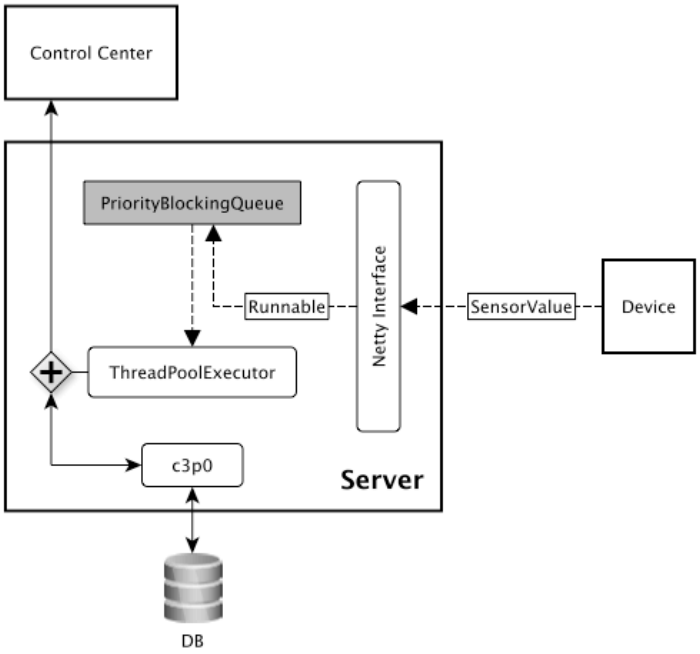

Figura 33. Processamento *Server*

A Figura 33 demonstra o processo de envio de uma mensagem do tipo *SensorValue* por parte do *Device* para o *Server*. O *Device* começa por enviar a mensagem sendo esta recebida pelo servidor usando a função *ChannelRead* da *framework* Netty. De seguida é criado um *Runnable* para esta mensagem que indica o tipo de processamento e o tipo de dados que esta contém (se é um alerta ou um bloco de dados de um ou mais sensores, por exemplo). Esse *Runnable* é então colocado numa estrutura de dados do tipo *PriorityBlockingQueue*. Esta estrutura, do tipo fila, é *thread-safe*, ou seja, pode ser acedida por várias *threads* sem que haja falhas na integridade dos dados. Para além disto é possível definir a prioridade dos elementos na estrutura (neste caso cada elemento é um *Runnable*). Por exemplo, se chegar uma mensagem do tipo alerta, esta terá a prioridade máxima e ao ser inserida na fila vai ser colocada no inicio desta.

Para fazer o tratamento dos objetos *Runnable* inseridos na fila existe um *ThreadPoolExecutor* que é um conjunto de *Threads* em que uma delas será responsável por tratar cada um dos *Runnables*. Esta interface permite definir qual o número de *Threads* que se perfila como o ideal para cada caso e um número máximo de *Threads* possíveis. Se o número de *Threads* criadas dentro da *pool* for inferior ao número ideal no momento de tratar uma mensagem, é criada nova *Thread* mesmo que haja na *pool* alguma que esteja inativa. Se existem mais tarefas a serem executadas e todas as *Threads* ideais estiverem a correr e se esse número for inferior ao número máximo definido, uma nova *Thread* é criada para executar a tarefa.

No caso da mensagem ser do tipo *SensorValue*, como é o caso da figura, ao executar a tarefa relativa a uma mensagem deste tipo, existem dois processos a decorrer em paralelo: o primeiro é o de enviar os dados para os *Control Center* que os pretenderem visualizar e o outro é

o de guardar na Base de Dados toda a informação. Na próxima secção é abordado o c3p0, utilizado para realizar as operações relativas à Base de Dados.

5.2.2. Gestão da Base de Dados

A gestão da Base de Dados do lado do servidor é feita recorrendo a uma biblioteca externa intitulada de *c3p0*. Esta biblioteca é fácil de usar e permite criar *drivers* JDBC, possibilitando as funcionalidades introduzidas na especificação do *jdbc4*. [24]

O *c3p0* fornece vários serviços: uma classe que adapta o tradicional *DriverManager* para o novo *javax.sql.DataSource* para obter as várias conexões a Base de Dados. Existem também um número ideal e um número máximo de conexões. Neste caso, estes número são iguais aos números de *Threads* definidos para a *ThreadPoolExecutor*. Isto faz com que, no caso em que todas as *Threads* disponíveis estejam a tentar aceder a Base de Dados, possam ter cada uma a sua conexão.

5.3. *Device*

Esta secção é reservada à apresentação do trabalho realizado na componente do *Device*. O trabalho de implementação do *Device* teve como responsável João Gonçalves. As tarefas do estagiário focaram-se nos seguintes aspetos: tratamento de dados, algoritmo de Man-Down e geração de alertas.

A placa para captura de sensores definida para o dispositivo foi o *BITalino*. Esta definição foi feita por parte da empresa no âmbito do projeto. A equipa de desenvolvimento deste componente teve acesso à API para comunicação com as portas da placa de captura de sinais de sensores. [25]

Os sensores usados para o desenvolvimento foram: electrocardiograma, taxa de respiração, acelerómetro e temperatura.

Na figura seguinte é apresentado o processamento implementado para esta componente do sistema.

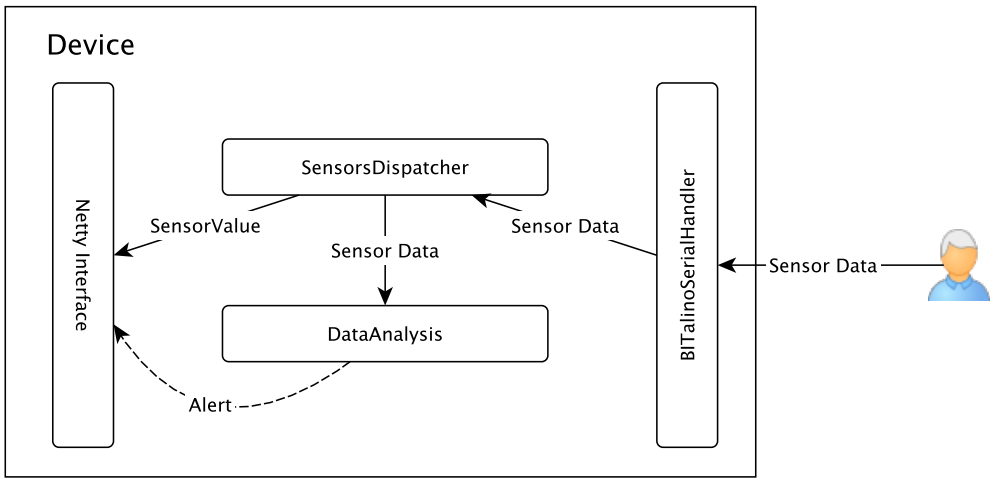

Figura 34. Processamento *Device*

O processamento demonstrado na Figura 34 tem inicio com a captação dos dados dos sensores conectados à pessoa. A gestão de todas as comunicações é feita pela *Thread BITalinoSerialHandler* que, como o próprio nome indica, é o responsável por gerir a comunicação com a placa *BITalino*. Ao receber os dados, estes são colocados na *Thread SensorsDispatcher* para serem transformados numa mensagem *protobuf* do tipo *SensorValue* e para serem enviadas. O *SensorsDispatcher* é também responsável por enviar os dados para outra *Thread* intitulada

DataAnalysis. Esta última é responsável por fazer a análise dos dados que estão a ser enviados pelo *Device* e enviar alertas caso seja detetada alguma anomalia.

Nesta componente, o estagiário esteve envolvido de forma mais acentuada no desenvolvimento do módulo de análise dos dados e envio de alertas. Na análise de dados, o foco centrou-se principalmente no algoritmo *Man-Down* para a detecção da posição em que se encontra o portador do dispositivo.

5.3.1. Análise de dados e Geração de Alertas

O processo de análise inicializa-se com a verificação do tipo de sensor que se está a tratar. Existe depois uma função na *Thread DataAnalysis* para cada um dos tipos e que é responsável por analisar os dados desse sensor.

Caso seja detetado algum problema, tem de ser enviado um alerta de duas formas:

1. Através do servidor do sistema – neste caso os alertas são enviados como uma mensagem normal do sistema, com o tipo alerta.

2. *Message Broker* do projeto SALUS – no âmbito do projeto SALUS, foi pedido que os alertas fossem enviados também via um servidor próprio do projeto, o *Message Broker*. Foi fornecida ao estagiário documentação sobre o mesmo e uma API para ser utilizada no desenvolvimento. Neste caso foi necessário definir a estrutura da mensagem de alerta, partindo do pressuposto que a mensagem seria enviada no formato JSON.

De seguida é apresentado um exemplo da mensagem a enviar através do *Message Broker*:

```
JSONObject message = constructMessageMB("TEMP", "ORANGE",
sensor.getValue(i).getValue(), sensor.getValue(i).getTime());
```

O método *constructMessageMB* é o responsável por construir a mensagem e devolver o objeto JSON a ser enviado. Recebe quatro parâmetros como argumentos: o tipo de sensor a que se refere o alerta que, neste exemplo, é o sensor de temperatura; o tipo de alerta que neste caso é um alerta laranja; o valor debitado pelo sensor que disparou o alerta e o instante de tempo em que o alerta ocorreu.

5.3.2. Algoritmo Man-Down

O algoritmo de Man-Down serve, principalmente, para detetar quedas do portador do dispositivo. Além disto, serve para analisar a posição (deitado, em pé ou a andar). Este algoritmo teve como base a análise de um já explorado em *smartphones Android*. [26]

Para isto, é utilizado o sensor acelerómetro que consegue medir a aceleração linear nos eixos do xx, yy e zz. Assume-se que este sensor é colocado na *backpack* que contém o *Device* e que é transportada pelo utilizador. Estando o sensor na posição vertical, a aceleração é sentida no eixo dos yy. Como o sensor, ao ser acoplado à *backpack*, ficará na posição horizontal, terá de haver uma calibração inicial do algoritmo para definir que a aceleração inicial (e estando o utilizador em pé, na vertical) é aplicada sobre o eixo dos zz.

Existem dois módulos principais neste algoritmo:

1. *Posture Recognition* – serve para detetar, a cada instante de tempo, qual a posição do portador do dispositivo. Existem três posições possíveis: em pé, deitado e a andar. Aos valores debitados pelo eixo dos zz são aplicados vários limites para definir a orientação do sensor. Estes limites foram sendo calibrados para produzir os resultados e a precisão esperados.

Funcionamento do *Posture Recognition*:

- Este método começa por receber um conjunto de valores, a que se dá o nome de Janela. Esta Janela é recebida pelo algoritmo, a cada segundo, e tem de conter no mínimo cinquenta valores representativos do cálculo da aceleração normal para cada conjunto de valores do acelerómetro (xx, yy, zz). É analisada essa Janela, definindo a aceleração normal máxima e mínima para cada conjunto de valores, guardando o último valor do eixo dos zz.
- De seguida é feito um método ZRC (*Zero Crossing Rate*) que deteta quantas variações significativas da aceleração normal existem na janela em questão. Se não detetar nenhuma variação significativa, assume-se que existem dois estados possíveis: em pé ou deitado (para a definição deste estado recorre-se ao último valor do eixo dos zz guardado na etapa anterior). Se existirem variações suficientes o estado "a andar" é identificado.
- Este método retorna então a posição para cada Janela analisada.

2. *Fall Detection* – este módulo é responsável por detetar a queda. É procurado um padrão particular em que a diferença entre máximos e mínimos consecutivos seja maior que 3.5G, ou seja, um impacto. No algoritmo para *Android* analisado este valor era de 2G mas verificou-se que, nestas condições (ou seja, com o sensor acelerómetro usado), as quedas eram facilmente detetadas com impactos relativamente pequenos e sem realmente terem existido.

- No processamento de cada Janela, este método é executado logo após o *Posture Recognition* e limita-se a verificar se a diferença entre o valor da normal máximo e mínimo é grande o suficiente para detetar queda (a força do impacto é calculada pela diferença das acelerações normais).

A decisão final relativamente á existência de queda é baseada no *output* gerado pelos dois métodos descritos anteriormente (*Posture Recognition* e *Fall Detection*). Quando é detetada uma queda, a decisão obtida pelo *Posture Recognition* é que vai reconhecer se é um falso alarme ou não. Por exemplo, se depois de detetada uma queda, a posição é "a andar", a queda detetada pode ser automaticamente descartada. A tabela seguinte apresenta as várias decisões finais possíveis em relação a uma queda.

Fall Detection	Posture Recognition	Queda?
Queda	A andar	Não
Queda	Em pé	Não
Queda	Deitado	Sim
Queda	Null	Sim

Tabela 14. Decisão sobre queda

No *Control Center* são apresentadas diferentes imagens para cada uma das posições. A Figura 35 demonstra as várias posições possíveis e as respetivas imagens.

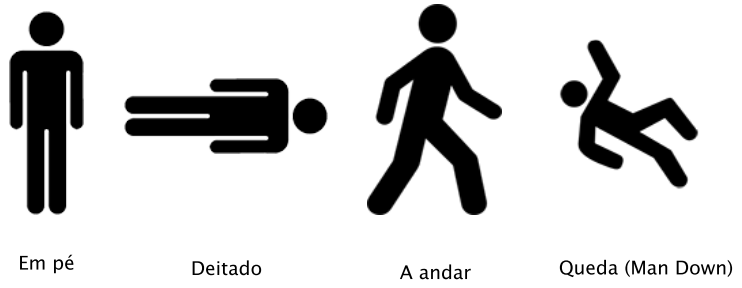

Figura 35. Posições identificadas

O desenvolvimento deste algoritmo começou com a adaptação de um algoritmo já existente para dispositivos *Android*. Foi necessário perceber em que medida a posição do acelerómetro influenciava o comportamento do algoritmo e fundamentalmente foi necessário fazer vários testes para calibrar os limites definidos para cada uma das situações. A título de exemplo, existiam várias vezes algumas incoerências no que toca à detecção de queda: algum impacto menos significativo (pequena colisão do dispositivo com uma parede, por exemplo) detetava logo quedas. Por isto foi necessário a calibração dos valores dos limites.

5.4. Sumário

Nesta secção é apresentado um sumário da fase de implementação. Para isso, são relembradas as tabelas de requisitos definidas no Capítulo 3 e adicionadas a estas duas colunas, uma que indica se o requisito foi ou não implementado e outra que indica quem foi que implementou. Nesta última, podem ser colocadas as seguintes indicações:

- 'E' – Somente estagiário
- 'PE' – Parcialmente estagiário
- 'O' – Outros elementos da equipa

Código	Descrição	Prioridade	Implementado?	Por quem?
REQ_CC_01	Autenticar-se com sucesso no sistema através de utilizador e palavra-chave	Alta	Sim	O
REQ_CC_02	Estabelecer ligação com o servidor, com sucesso	Muito Alta	Sim	E
REQ_CC_03	Visualizar todos os dispositivos ativos	Alta	Sim	PE
REQ_CC_04	Monitorizar vários dispositivos ao mesmo tempo	Muito Alta	Sim	PE
REQ_CC_05	Monitorizar um dispositivo em detalhe	Muito Alta	Sim	PE
REQ_CC_06	Apresentar fotografia de cada individuo na lista de dispositivos	Normal	Sim	O
REQ_CC_07	Apresentar nome de cada individuo na lista de dispositivos	Alta	Sim	PE
REQ_CC_08	Apresentar sexo de cada individuo na lista de dispositivos	Alta	Sim	PE
REQ_CC_09	Apresentar a idade de cada individuo na lista de dispositivos	Alta	Sim	PE
REQ_CC_10	Apresentar o valor para o sensor de ECG para cada individuo na lista de dispositivos	Alta	Sim	PE
REQ_CC_11	Apresentar o valor para o sensor de temperatura para cada individuo na lista de dispositivos	Normal	Sim	PE
REQ_CC_12	Apresentar o valor da taxa de respiração para cada individuo na lista de dispositivos	Normal	Sim	PE
REQ_CC_13	Ao selecionar um dispositivo, a informação sobre o individuo deve aparecer na secção de detalhe	Muito Alta	Sim	PE
REQ_CC_14	Apresentar nome do individuo na secção de detalhe	Muito Alta	Sim	PE

REQ_CC_15	Apresentar idade do individuo na secção de detalhe	Muito Alta	Sim	PE
REQ_CC_16	Apresentar sexo do individuo na secção de detalhe	Muito Alta	Sim	PE
REQ_CC_17	Apresentar a localização do individuo na secção de detalhe	Baixa	Sim	O
REQ_CC_18	Apresentar o gráfico com os dados do ECG em tempo-real na secção de detalhe	Muito Alta	Sim	E
REQ_CC_19	Visualizar o gráfico do ECG aumentado na secção de detalhe	Baixa	Sim	O
REQ_CC_20	Apresentar o gráfico com os dados da taxa de respiração em tempo-real na secção de detalhe	Alta	Sim	PE
REQ_CC_21	Visualizar o gráfico da taxa de respiração aumentado na secção de detalhe	Baixa	Sim	O
REQ_CC_22	Apresentar os valores das coordenadas x, y e z para o acelerómetro na secção de detalhe	Normal	Sim	E
REQ_CC_23	Apresentar uma figura que represente a posição do individuo na secção de detalhe	Normal	Sim	E
REQ_CC_24	Redimensionar janela manualmente	Normal	Sim	O
REQ_CC_25	Redimensionar janela de acordo com o tamanho do ecrã	Normal	Não	-
REQ_CC_26	Visualizar histórico na secção de detalhe	Normal	Sim	O
REQ_CC_27	Apresentar uma secção reservada para os alertas	Alta	Sim	O
REQ_CC_28	No caso de haver algum alerta, este tem de ser apresentado na secção reservada para o efeito	Alta	Sim	PE
REQ_CC_29	Apresentar uma secção reservada para a transmissão de voz e vídeo	Muito Alta	Sim	O
REQ_CC_30	Possibilitar o inicio de vídeo	Alta	Sim	O
REQ_CC_31	Possibilitar a paragem da visualização do sinal de vídeo	Alta	Sim	O
REQ_CC_32	Regular a qualidade do sinal de vídeo	Normal	Sim	O
REQ_CC_33	Possibilitar o inicio do sinal de áudio	Alta	Sim	O
REQ_CC_34	Possibilitar a paragem do sinal de áudio	Alta	Sim	O
REQ_CC_35	Regular o volume do sinal de áudio	Normal	Sim	O
REQ_CC_36	Ao sucederem, visualizar alertas amarelos	Normal	Sim	PE
REQ_CC_37	Ao sucederem, visualizar alertas vermelhos	Normal	Sim	PE
REQ_CC_38	Cada alerta amarelo deve destacar o dado a que se refere a amarelo	Normal	Sim	PE
REQ_CC_39	Cada alerta vermelho deve destacar o dado a que se refere a vermelho	Normal	Sim	PE
REQ_CC_40	Possibilitar o descarte de um determinado alerta	Normal	Sim	O
REQ_CC_41	Apresentar o tipo de rede que está a	Alta	Sim	O

	ser utilizado pelo dispositivo, no ecrã de detalhe			
REQ_CC_42	Apresentar a qualidade de rede do dispositivo no ecrã de detalhe	Alta	Sim	O
REQ_CC_43	Permitir visualização de vídeo em ecrã inteiro	Alta	Sim	O
REQ_CC_44	Ordenar lista de utilizadores por nome	Normal	Sim	O
REQ_CC_45	Pesquisar utilizador na lista de utilizadores	Normal	Sim	O
REQ_CC_46	Visualizar o estado da bateria do dispositivo	Normal	Sim	O
REQ_CC_47	Apresentar uma secção para receber notificações	Baixa	Sim	O
REQ_CC_48	Captar *screenshot* a partir do sinal de vídeo	Baixa	Sim	O

Tabela 15. Requisitos implementados *Control Center*

Código	Descrição	Prioridade	Implementado?	Por quem?
REQ_S_01	Aceitar a conexão de um dispositivo no sistema	Muito Alta	Sim	E
REQ_S_02	Aceitar a conexão de vários dispositivos no sistema	Muito Alta	Sim	E
REQ_S_03	Autenticar um dispositivo no sistema	Alta	Sim	PE
REQ_S_04	Aceitar a conexão de um *Control Center* no sistema	Muito Alta	Sim	E
REQ_S_05	Aceitar a conexão de vários *Control Center* no sistema	Muito Alta	Sim	E
REQ_S_06	Autentica um *Control Center* no sistema	Alta	Sim	PE
REQ_S_07	Receber os dados de um dispositivo	Muito Alta	Sim	PE
REQ_S_08	Enviar dados de um dispositivo para os *Control Center* que os pretendam visualizar	Muito Alta	Sim	PE
REQ_S_09	Analisar os dados que chegam ao servidor	Normal	Sim	E
REQ_S_10	Enviar alertas para os *Control Center* caso haja um problema	Alta	Sim	E
REQ_S_11	Guardar dados na base de dados para efeitos de histórico	Alta	Sim	PE
REQ_S_12	Alterar configurações da base de dados	Alta	Sim	O
REQ_S_13	Ler da Base de Dados os dados relativos ao histórico	Alta	Sim	PE
REQ_S_14	Enviar lista de dispositivos configurados como visíveis para um determinado *Control Center*	Muito Alta	Sim	PE

Tabela 16. Requisitos implementados *Server*

Código	Descrição	Prioridade	Implementado?	Por quem?
REQ_D_01	Conectar-se ao servidor	Muito Alta	Sim	PE
REQ_D_02	Autenticar-se com sucesso no servidor	Alta	Sim	PE
REQ_D_03	Captar os dados dos sensores	Muito Alta	Sim	PE
REQ_D_04	Analisar os dados	Alta	Sim	E
REQ_D_05	Encriptar os dados para envio	Alta	Sim	O
REQ_D_06	Enviar os dados para o servidor	Muito Alta	Sim	PE
REQ_D_07	Em caso de detetar problema, enviar alerta para o servidor	Alta	Sim	E
REQ_D_08	Em caso de detetar problema, enviar alerta para o Message Broker do projeto SALUS	Alta	Sim	E
REQ_D_09	Enviar dados do sensor de ECG	Muito Alta	Sim	PE
REQ_D_10	Enviar dados do sensor de temperatura	Muito Alta	Sim	PE
REQ_D_11	Enviar dados do sensor de taxa de respiração	Muito Alta	Sim	PE
REQ_D_12	Enviar dados do sensor acelerómetro	Muito Alta	Sim	E
REQ_D_13	Enviar sinal de vídeo	Alta	Sim	O
REQ_D_14	Enviar sinal de voz	Alta	Sim	O

Tabela 17. Requisitos implementados *Device*

Analisando as tabelas apresentadas, é possível concluir que o estagiário esteve envolvido mais vincadamente em tarefas relativas ao *backend* do sistema. Muitos dos requisitos relativos ao *Control Center* estão relacionados diretamente com apresentação de dados, ou seja, interação com o utilizador, e o estagiário esteve envolvido em questões relativas ao processamento para obter um determinado resultado, sem ser ele a focar-se em questões de design.

Capítulo 6 – Verificação e Validação

De forma a validar e avaliar o sistema desenvolvido foi necessária a criação de um plano de testes capaz de detetar falhas na implementação das funcionalidades, perceber se o sistema se comporta de acordo com os requisitos definidos no capítulo 3 e avaliar a qualidade do produto desenvolvido.

Tendo em conta que estava apenas disponível um dispositivo para testes, foi necessário arranjar uma forma de simular vários dispositivos a enviarem dados próximos dos reais e a poderem comportar-se de acordo com a realidade. Foi a criação deste simulador que arrancou todo o processo de testes e validações. Depois disto foi necessário o desenho de um plano de testes que se dividiu em duas fases: testes funcionais e testes não-funcionais. Os primeiros pretendem examinar os resultados obtidos pelo sistema e confrontá-los com o que é esperado. Os testes não-funcionais relacionam-se com o comportamento do sistema relativamente aos requisitos não-funcionais.

Esta foi a abordagem foi escolhida pelo facto de esta fase pretender analisar e verificar se os requisitos que foram estipulados foram ou não cumpridos. Como esses requisitos se dividiram desde inicio numa componente funcional e numa não funcional, considerou-se também que a fase de testes e validações de deveria subdividir da mesma forma.

Toda esta fase teve como responsável Patrício Baptista e o estagiário trabalho com este no desenvolvimento dos testes.

6.1. Simulador

Para iniciar e suportar toda a fase de testes e validações foi necessário implementar um simulador de dispositivos capaz de criar um número variável de dispositivos enviando dados próximos do real e simulando comportamentos como, por exemplo, alertas. Esta necessidade partiu do facto do hardware existente ser bastante limitado. O simulador possibilitou assim simular grandes cenários que de outra forma não era possível. Este simulador foi usado para a realização de várias demonstrações do produto por parte da empresa.

Este simulador foi desenvolvido em Python e implementado inteiramente pelo estagiário. É inicializado com o comando "python sim.py <número de dispositivos>". Através deste comando podemos inicializar o número de dispositivos que pretendamos.

Para ser feita a comunicação entre o simulador e os dispositivos, é criado um ficheiro intitulado "test_fifo<id_device>" para cada dispositivo simulado. Existe uma *Thread* do lado do dispositivo que está permanentemente a ler os comandos que são escritos neste ficheiro. Este *Thread* que é tem o nome de "TestFifo" e controla, ou seja, executa métodos de outras *Threads* como a "FeedServer", responsável por enviar dados simulados e a "BitalinoSerialHandler", responsável por gerir as conexões e enviar os dados dos sensores reais.

De seguida seguem exemplo da interface criada para este simulador.

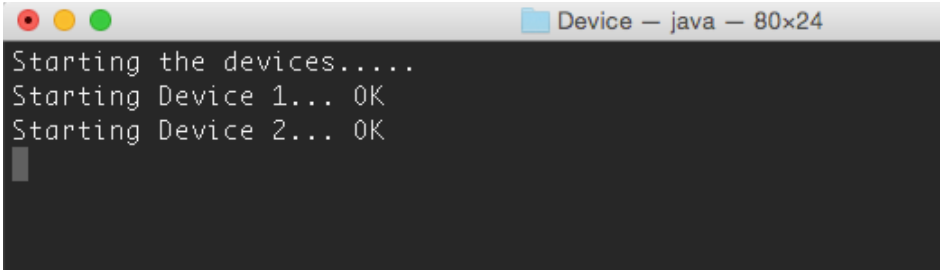

Figura 36. Simulador - Inicialização de *Devices*

Este ecrã demonstra a inicialização de dois dispositivos. Este processo tem um tempo fixo de dez segundos para o simulador poder criar os ficheiros de configuração necessários para

os dispositivo se poderem autenticar no servidor, entre outras coisas. Além disto, é necessário que todos os dispositivos inicializem, se conectem ao servidor e comecem a enviar os dados.

Para isto tudo acontecer escrevem-se então alguns comandos para o "test_fifo" destinado a cada dispositivo. Os comandos definidos foram:

- 'F' – Iniciar/parar a execução da *Thread* FeedServer, responsável por criar e enviar dados simulados, próximos dos reais para os sensores de ECG, temperatura e taxa de respiração. Além destes são enviados também os valores do acelerómetro para uma determinada posição do utilizador do dispositivo.
- 'B' – Iniciar/parar a execução do BitalinoSerialHandler que tem como função enviar os dados reais vindos da placa BITalino.
- 'A1O' – Simular alerta laranja do sensor ECG.
- 'A1R' – Simular alerta vermelho do sensor ECG.
- 'A2O' – Simular alerta laranja para o sensor de temperatura.
- 'A2R' – Simular alerta vermelho para o sensor de temperatura.
- 'A30' – Simular alerta laranja do sensor de taxa de respiração.
- 'A3R' – Simular alerta vermelho do sensor de taxa de respiração.
- 'ACCP' – Simular a posição "em pé" do utilizador que usa o dispositivo.
- 'ACCL' – Simular a posição "deitado" do utilizador que usa o dispositivo.
- 'ACCF' – Simular a queda do utilizador que usa o dispositivo.

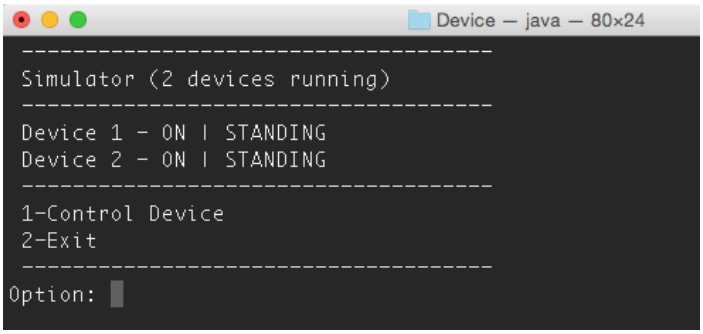

Figura 37. Simulador - Menu inicial

Este ecrã mostra o menu criado para o simulador. Na parte superior é indicado o numero de dispositivos que estão ativos e logo por baixo o estado de cada um deles: "ON" para indicar que está a enviar dados e "STANDING" referindo-se a posição de pé. São apresentadas de seguida duas opções: a primeira serve para controlar um determinado dispositivo; a segunda serve para terminar o simulador e desligar todos os dispositivos.

Ao selecionarmos a opção '1', é pedido o identificador numérico do dispositivo que pretendemos controlar. De seguida surge o seguinte ecrã.

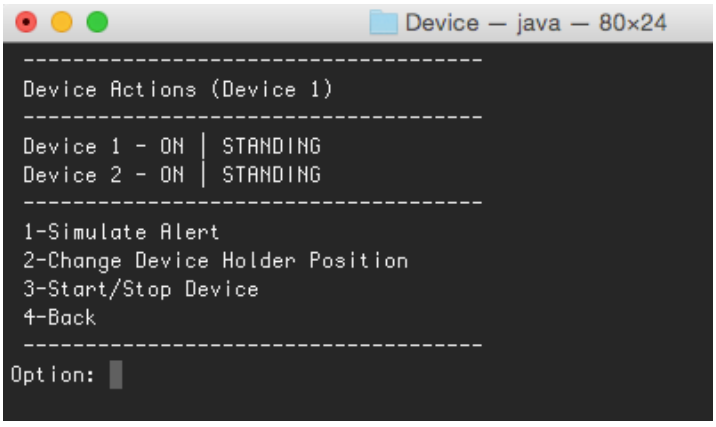

Figura 38. Simulador - Menu ações

Neste ecrã surgem então as seguintes opções: '1' para simular um determinado alerta; '2' para mudar a posição do utilizador do dispositivo; '3' para parar ou reiniciar o envio de dados por parte de um dispositivo e '4' para voltar ao menu anterior. Surgem dois novos menus ao selecionar a opção '1' e '2', volta-se ao menu inicial se selecionarmos a opção '3' e volta-se ao menu anterior de selecionarmos a opção 4.

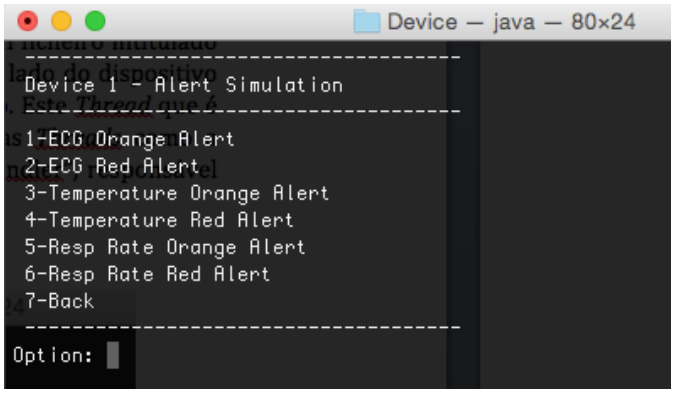

Figura 39. Simulador - Alertas

Neste ecrã poderemos simular os alertas laranja e vermelho para cada um dos três sensores simulados: eletrocardiograma, temperatura e taxa de respiração.

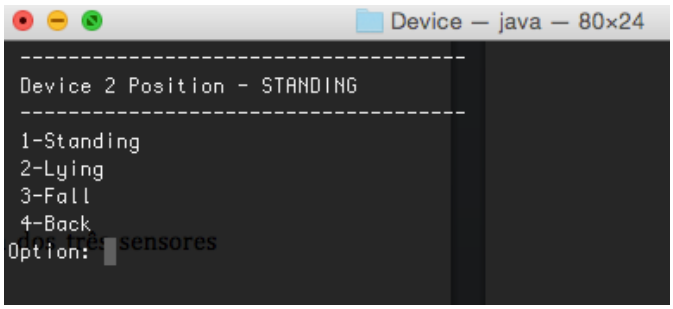

Figura 40. Simulador - Posição

Este ecrã possibilita a alteração da posição do utilizador. Existe a posição "de pé", "deitado" e "queda", que neste caso simula o "Man-Down".

6.2. Testes Funcionais

Nesta secção é descrito o que foi feito relativamente a testes funcionais, dividindo estes em duas secções: testes unitários e testes de sistema.

6.2.1. Testes Unitários

Ao longo do desenvolvimento de todo o sistema, as diversas unidades criadas para cada componente foram sendo testadas, tendo sido corrigidos desde logo as falhas detetadas em algumas funcionalidades e métodos. Estes testes ao longo do desenvolvimento permitiram validar algumas decisões tomadas relativamente à arquitetura e evitou que se gerassem falhas maiores e com impacto no produto final.

Para a realização deste tipo de testes ficou estipulado que para cada funcionalidade desenvolvida haveria uma ação com um resultado esperado por parte do sistema. Nos casos em que se verificou que o resultado obtido por uma determinada ação não era o esperado, foram feitas as alterações e correções necessárias e testado de novo até o resultado obtido ser igual ao esperado.

Numa fase mais avançada do sistema, em que as várias componentes e funcionalidades especificas estavam já implementadas e testadas, foi necessária a realização de testes de

integração de forma a compreender como é que os diferentes componentes interagiam entre si e constatar que essa interação ocorreria sem falhas e de acordo com o esperado.

Nesta última fase foi necessário o envolvimento de todos os elementos do grupo de trabalho para os esclarecimentos que fossem necessários.

6.2.2. Testes de Sistema

Depois de todas as funcionalidades do sistema terem sido desenvolvidas, segue-se o processo de validação da implementação. Neste tipo de testes são confrontados os requisitos funcionais definidos no Capítulo 3 com os resultados produzidos pelo sistema.

O registo de cada teste de sistema foi feito de acordo com a tabela especificada de seguida.

ID Teste	
Descrição	Pequena descrição do Teste
Prioridade	Elevada, média ou baixa
Requisito(s)	Código(s) do(s) Requisito(s)
Condições	Condições necessárias para a realização do teste
Ações/Dados necessários	Ações ou dados de entrada necessários para a realização do teste
Resultado Esperado	Define-se qual o comportamento esperado pelo sistema
Estado	Passou ou não passou
Detalhes	Observações relativas ao teste

Tabela 18. Tabela descritiva de um teste de sistema

A definição desta tabela foi feita pelo estagiário e validada pela empresa.

Este processo possibilitou verificar se as funcionalidades implementadas respondem da melhor forma ao requisitos estipulados.

No Anexo 3 estão detalhados todos os testes efetuados e registados segundo este modelo.

Analisando o resultado dos testes efetuados, pode concluir-se que todos eles foram executados com sucesso. Em cada teste foi possível testar vários requisitos ao mesmo tempo. Por exemplo, num teste intitulado “envio de dados pelo dispositivo”, podem ser testados requisitos funcionais do dispositivo (envio de dados para o servidor) e do servidor (recepção de dados).

Capítulo 7 – Conclusão

O presente trabalho desenvolveu-se destacando um objetivo principal: desenvolver um sistema de monitorização de sinais vitais em tempo-real para ambientes PPDR e que integrasse com um módulo de transmissão de voz e vídeo já implementado.

Este objetivo dividiu-se de forma a tornar mais perceptível qual o âmbito específico do trabalho de estágio e que etapas este iria ter. Desta divisão resultaram as seguintes fases:

- Análise de Requisitos
- Especificação da Arquitetura
- Desenvolvimento
- Validações e testes

Depois de estudadas e exploradas as competências necessárias para o desenvolvimento do trabalho, foi possível começar por definir os Requisitos. Os requisitos foram delimitados em funcionais, para cada um dos componentes do sistema, e não-funcionais, para o sistema como um todo. A análise de requisitos iniciou-se com a definição de três cenários PPDR, onde o sistema seria útil, salientando os possíveis intervenientes: polícias, bombeiros e profissionais de saúde. Foi com base nestes cenários que foram construídas as tabelas de requisitos funcionais e não-funcionais, e definida a prioridade para cada requisito. Estas tabelas foram sendo atualizadas e revistas ao longo do projeto, tendo em consideração diretivas resultantes das reuniões internas e no seio do projeto SALUS.

Na definição da arquitetura foi feito um trabalho de investigação que procurou encontrar a melhor forma de resolver os problema que se apresentavam. Esta procura incidiu sobre temas como qual a base de dados a utilizar e qual o seu modelo. Dado o número elevado de transações por segundo com que a base de dados se iria deparar e a obrigatoriedade de garantir a consistência dos dados (sinais vitais são dados muito sensíveis que requerem essa consistência), foi estudado e desenvolvido um modelo de base de dados capaz de responder a estes requisitos. Este modelo foi evoluindo, procurando sempre tentar obter a melhor performance possível. Nesta fase, foi necessário definir também as *frameworks* a utilizar para as comunicações entre componentes e um modelo das mensagens a serem enviadas entre pares de componentes, ou seja, *Device-Server* e *Server-Control Center*.

Terminado o primeiro passo, o estagiário foi integrado num grupo de trabalho onde se deu início ao desenvolvimento de todas as funcionalidades identificadas anteriormente. Nesta fase, o trabalho do estagiário foi transversal a todas as componentes que fazem parte do sistema, uma vez que desempenhou tarefas específicas em cada uma delas. Na componente do *Control Center*, o estagiário teve um papel mais relevante no desenvolvimento de um mecanismo que permitisse a visualização de dados através de gráficos 2D e com a melhor performance possível. Quanto à componente do servidor, foi responsável por tarefas como a implementação de um modelo de base de dados e a gestão das comunicações com os restantes componentes. Relativamente ao dispositivo, o estagiário foi mais interventivo aquando da análise dos dados e da implementação de um algoritmo para detecção de quedas.

Por fim, teve lugar a fase de validação e testes. Parte da validação ocorreu ao longo da implementação através de testes unitários e de integração entre as várias componentes. De forma a garantir que o sistema cumpre os requisitos definidos, foram realizados testes de sistema com o intuito de verificar se o mesmo estava implementado de acordo com o esperado e estipulado. Nesta fase, o estagiário foi responsável por implementar um simulador de dispositivos e por realizar os testes de sistema. Este simulador permitiu criar condições que de outra forma não seriam possíveis. Sem o simulador não teria sido possível testar cenários em que mais de que um dispositivo estivesse conectado ao sistema e a enviar dados ou simular os diferentes alertas que podiam suceder, uma vez que havia a restrição de só existir um dispositivo físico.

Fazendo uma análise ao trabalho desenvolvido, é possível concluir que os objetivos definidos no início deste estágio foram plenamente alcançados. A Análise de Requisitos e a Especificação da Arquitetura permitiram estudar as várias possibilidades para resolver os problemas que foram surgindo e comprovaram que era possível implementar este sistema. A fase de desenvolvimento cumpriu com o planeamento definido e decorreu de acordo com a metodologia acordada. Finalmente, a fase de testes e validações foi fundamental para garantir que tudo estava correctamente implementado.

O resultado final do projeto onde este trabalho esteve inserido pode considerar-se como sendo um produto funcional capaz de cumprir os objetivos propostos. Foi submetido a várias demonstrações, quer no âmbito do projeto SALUS, onde foi apresentado aos vários parceiros do projeto, por diversas ocasiões, e onde foram sendo relatadas as melhorias a serem feitas; quer na feira CCW 2015 (*Critical Communications World*), onde o sistema foi integrado nos produtos de demonstração do sistema 4G da Alcatel-Lucent.

7.1. Trabalho Futuro

Uma vez que estamos perante um produto funcional, o trabalho seguinte relativo a este sistema será a entrada em produção, ou seja, é necessária a construção de mais dispositivos físicos e a comercialização do sistema como um todo. Desta forma, será possível mudar a maneira como são tratadas as situações em cenários como os apresentados neste relatório. À medida que se dê o crescimento do sistema, é necessário também procurar reduzir ao máximo o custo na produção dos dispositivos.

Relativamente a questões funcionais, as sugestões para trabalho futuro prendem-se com os seguintes aspectos:

- Estudar a possível migração para Bases de Dados não relacionais e os benefícios que isso traria em termos de performance para o sistema.
- Criar interfaces do *Control Center* especificas para cada interveniente em cenários PPDR (bombeiros, polícias, médicos, etc.) . Para isto, seria necessário definir novas funcionalidades especificas e a forma mais conveniente de apresentar os dados.

Capítulo 8 – Referências

[1] **Alan R. Jamieson**, *Radiocommunication for public protection and disaster relief*, , 2006, disponível em https://www.itu.int/itunews/manager/display.asp?lang=en&year=2006&issue=03&ipage=publicProtection&ext=html.

[2] **World Health Organization**, *mHealth New horizons for health through mobile technologies*, 2011, disponível em http://www.who.int/goe/publications/goe_mhealth_web.pdf.

[3] **José Guerreiro, Raúl Martins, Hugo Silva, André Lourenço, Ana Fred**, *BITalino: A Multimodal Platform for Physiological Computing*, 2013, disponível em http://www.researchgate.net/publication/259177489_BITalino_A_Multimodal_Platform_for_Physiological_Computing.

[4] VisiMobile, "Overview", http://www.visimobile.com/overview/ [Última visita: 7 de Janeiro de 2015].

[5] Medtronic, "CareLink Systems", http://www.medtronic.com/innovation/connected-carelink.html [Última visita: 7 de Janeiro de 2015].

[6] Telcare, "How it works", https://www.telcare.com/how-it-works [Última visita: 7 de Janeiro de 2015].

[7] Qualcommlife, "What is 2net?", http://www.qualcommlife.com/wireless-health [Última visita: 7 de Janeiro 2015].

[8] AirStrip, "Airstrip ONE", http://www.airstrip.com/solutions/airstrip-one [Última visita: 8 de Janeiro 2015].

[9] AliveCor, "AliveCor Mobile ECG", http://www.alivecor.com/home [Última visita: 8 de Janeiro de 2015].

[10] Top Fitness Apps, "Instant Heart Rate", http://www.topfitnessapps.com/iphone-app-reviews/instant-heart-rate/ [Última visita: 8 de Janeiro de 2015].

[11] Philips, "Vital Signs Camera", http://www.vitalsignscamera.com/index.html [Última visita: 8 de Janeiro de 2015].

[12] iHealth, "iHealth", http://www.ihealthlabs.com/ [Última visita: 8 de Janeiro de 2015].

[13] IBM Knowledge Center, "How sockets work", http://www-01.ibm.com/support/knowledgecenter/ssw_ibm_i_71/rzab6/howdosockets.htm [Última visita: 8 de Janeiro de 2015].

[14] **David Curtis**, *Java, RMI and CORBA*, 1997, disponível em http://www.omg.org/library/wpjava.html.

[15] **Shiwei Yu**, *ACID Properties in Distributed Databases*, 2009, disponível em http://www.cs.helsinki.fi/group/cinco/teaching/2009/advanced-businesstransactions-seminar/papers/ACID_in_Distributed_Database_Shiwei_Yu.pdf.

[16] **Luke P. Issac**, *SQL vs NoSQL Database Differences Explained with few Example DB*, 2014, disponível em http://www.thegeekstuff.com/2014/01/sql-vs-nosql-db/.

[17] **Ronny Standtke, Ulrich Ultes-Nitsche**, *Java NIO Framework, Introducing a high-performance I/O framework*, disponível em http://nioframework.sourceforge.net/NIO_Paper.pdf.

[18] **Dio Synodinos**, *Grizzly and the New Atmosphere Comet Framework: Q&A with Project Lead Jean-Francois Arcand*, 2008, disponível em http://www.infoq.com/news/2008/06/grizzly-atmosphere.

[19] **User "cb" in CoralReactor**, *CoralReactor vs Netty Performance Comparison*, 2014, disponível em http://www.coralblocks.com/index.php/2014/04/coralreactor-vs-netty-performance-comparison/.

[20] **John Boardman**, *Netty: A Different Kind of Web(Socket) Server*, 2015, disponível em https://keyholesoftware.com/2015/03/16/netty-a-different-kind-of-websocket-server/.

[21] **John O'Conner**, *Using the Swing Application Framework (JSR 296)*, 2007, disponível em http://www.oracle.com/technetwork/articles/javase/index-141957.html.

[22] **Daniel Rubio**, *JSF framework: Shale and Seam*, 2006, disponível em http://www.javaworld.com/article/2071762/java-web-development/jsf-frameworks--shale-and-seam.html.

[23] **Carl Dea, Mark Heckler, Gerrit Grunwald, Jose Pereda, Sean Phillips**, *JavaFX 8: Introduction by Example*, 2014.

[24] **Lance Andersen**, *JDBC 4.1 Specification*, 2011, disponível em http://download.oracle.com/otn-pub/jcp/jdbc-4_1-mrel-spec/jdbc4.1-fr-spec.pdf?AuthParam=1436112834_03b2d532112fca00dd4f89647aea9d40.

[25] **Ana Priscila Alves, Hugo Silva, André Lourenço, Ana Fred**, *BITalino: A Biosignal Acquisition System based on the Arduino*, 2013, disponível em http://www.lx.it.pt/~afred/papers/BITalino.pdf.

[26] **Bharadwaj Sreenivasan**, *Fall Detection with Posture recognition on Android Smartphone*, 2012, disponível em https://github.com/BharadwajS/Fall-detection-in-Android.

[27] **James Weaver,** *Pro JavaFX 8: A Definitive Guide to Building Desktop, Mobile and Embedded Java Clients,* 2014.

[28] **Richard Warburton,** *Java 8 Lambdas, Pragmatic Functional Programming,* 2014.

[29] **Dion Almaer,** *Protocol Buffers, our serialized strutctured data, released as Open Source,* 2008, disponível em http://googlecode.blogspot.pt/2008/07/protocol-buffers-our-serialized.html.

[30] **Uma Bhat, Shraddha Jadhav,** *Moving Towards Non-Relational Databases,* 2010, disponível em http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.206.3532&rep=rep1&type=pdf.

Anexos

Anexo A – Modelo de Mensagens TLV

Neste anexo são apresentadas as mensagens TLV definidas no inicio do desenho do protótipo da aplicação.

De *Control Center* para *Server* (Request)

Este request pode ser de vários tipos:

Tipo 1 – *Control Center* pede os dados dos sensores dos utilizadores que pretende que sejam visualizados no ecrã com todos os utilizadores.

TLV:

Type – Inteiro, representando o tipo da mensagem.
Length – Inteiro, representando o número de utilizadores a visualizar.
Value – Cada "value" é composto por um inteiro que representa o ID de cada utilizador.

Tipo 2 – *Control Center* pede os dados relativos a um utilizador específico para serem visualizados no ecrã de detalhe.

TLV:

Type – Inteiro, representando o tipo da mensagem.
Length – Neste tipo de mensagem, este campo não é usado tendo em conta que é sempre um utilizador.
Value – Inteiro que representa o id do utilizador em questão.

Tipo 3 – O *Control Center* pede ao *Server* os utilizadores que estão ativos e a enviar dados.

TLV:

Type – Inteiro, representando o tipo da mensagem.
Length – Este campo não é usado.
Value – Este campo não é usado.

De *Server* para *Control Center* (Packet)

O pacote pode ser de vários tipos e conter informação diversa:

Tipo 1 – *Server* envia os dados dos sensores relativos aos utilizadores que o centro do controlo pretende.

TLV:

Type – Inteiro, representando o tipo da mensagem.
Length – Inteiro, representando a quantidade de informações que compõem o pacote.
Value – Cada "value" é representado por um conjunto de três valores. Esses três valores são: um inteiro que representa o id do utilizador, um inteiro que representa o id do sensor e um double que representa o valor emitido pelo respectivo sensor.

Tipo 2 – *Server* envia dados para um utilizador especifico. Estes dados são enviados de duzentos em duzentos milissegundos.

TLV:

Type – Inteiro, representando o tipo da mensagem.
Length – Inteiro, representando o número de sensores ligados ao utilizador e a emitir dados.
Values – Cada value é composto por um TLV (TLV 2).

TLV 2:

Type – Inteiro, representando o id do sensor.

Length – Inteiro, representando o número de valores a ler para o sensor especifico.

Values – Cada "value" é composto por um Long que representa o timestamp do PC que está ligado ao utilizador e um Double que representa o valor emitido pelo sensor.

Tipo 3 – *Server* envia para o *Control Center* os dados dos pacientes ativos.

TLV:

Type – Inteiro, representando o tipo da mensagem.

Length – Inteiro, representando o número de utilizadores que estão ativos.

Value – Cada "value" é representado por um conjunto de cinco valores. Esses cinco valores são: um inteiro que representa o id do utilizador, um inteiro que indica o número de bytes a ler a seguir, os bytes a ler para formar uma String com o nome do utilizador, um char que representa o sexo do utilizador e um inteiro que representa a idade do utilizador.

De *Server* para *Device* (RequestConfig)

Este pedido é feito sempre que o *Server* pretende alterar ou controlar as configurações relativas ao envio dos dados por parte do *Device*.

Tipo 1 - *Server* envia este tipo de mensagem para indicar que está preparado para que certo *Device* comece a enviar dados.

TLV:

Type – Inteiro, representando o tipo da mensagem.

Length – campo não usado.

Value – id do utilizador (isto não é necessário visto que cada *Device* sabe o id do utilizador que ta a monitorizar).

Tipo 2 – A mensagem deste tipo é usada para proceder a quaisquer alterações relativas ao envio dos dados por parte do *Device*.

TLV:

Type – Inteiro, representando o tipo da mensagem.

Length – Inteiro, representando o número de configurações a ler.

Value – Cada "value" composto por dois inteiros, um indicando o id da configuração e outro a indicar o valor da configuração (por exemplo, o servidor (por qualquer razão) pretende que o *Device* passe a enviar dados de segundo a segundo para um determinado sensor).

Tipo 3 – Esta mensagem é enviada após a conexão de um determinado *Device*. Serve para "perguntar" ao *Device* as informações iniciais (id do utilizador que está a monitorizar, por exemplo).

TLV:

Type – Inteiro, representando o tipo da mensagem.

Length – Não utilizado.

Value – Não utilizado.

De *Device* para *Server* (MessagePC)

Esta mensagem é utilizada para enviar os dados monitorizados pelos sensores relativos a um paciente.

Tipo 1 – Mensagem deste tipo é usada para enviar as configurações iniciais de cada *Device* (nomeadamente qual o utilizador a que está associado).

TLV:

Type – Inteiro, representando o tipo da mensagem.

Length – Não utilizado.

Value – Inteiro, representando, por exemplo, o id do utilizador que está a ser monitorizado.

Tipo 2 – Esta mensagem serve para enviar os dados dos sensores do utilizador que o *Device* está a monitorizar.

TLV:

Type – Inteiro, representando o tipo da mensagem.

Length – Inteiro, representando o número de registos a ler.

Value – Cada "value" é representado por quatro valores: um inteiro que representa o id do utilizador, um inteiro que representa o id do sensor, um long que representa o timestamp do *Device* e um double que representa o valor emitido pelo sensor.

D

Anexo B – Modelo de Mensagens Protobuffer

```
message Device {
      required string name = 1;
      required string sex = 2;
      required int32 age = 3;
      required int32 weight = 4;
      required string id = 5;
      required bool video = 6;
      repeated SensorConfig availableSensors = 7;
      required string IP = 8;
}

message SensorConfig {
      required SensorValue.Type sensorType = 1;
      required int32 priority = 2;
}

message SensorInfo {
      required string idDevice = 1;
      optional double sensor1 = 2;
      optional double sensor2 = 3;
      optional double sensor3 = 4;
      optional double sensor4 = 5;
      optional double sensor5 = 6;
      optional double latitude = 7;
      optional double longitude = 8;
      optional string lastGPSUpdate = 9;

}

message Authentication {
      required string key = 1;
      optional Response response = 2;
      enum Response {
            OK = 1;
            NOK = 2;
      }
}

message DeviceData{
      optional double cpu = 1;
      optional double memory = 2;
      optional int64 uptime = 3;
      optional double battery = 4;
      optional string networkType = 5;
      optional double wirelessSignalStrength = 6;
      optional double cellSignalStrength = 7;
}

message Configuration {
      required Type type = 1;
      repeated int32 attribute = 2 [packed=true];
      repeated Device devices = 3;
      enum Type{
            USERLIST = 1;
```

```
            DEVICECONFIG = 2;
      }
}

message Command {
      required Task command = 1;
      optional string param1 = 2;
      optional string param2 = 3;
      optional string param3 = 4;
      required string destinationIdHash = 5;
      repeated SensorValue.Type sensorAlert = 6;
      optional SensorValue sensorHistoryValue = 7;

      enum Task {
            STARTVIDEO = 1;
            STARTAUDIO = 2;
            STARTVIDEOAUDIO = 3;
            STOPVIDEO = 4;
            STOPAUDIO = 5;
            STOPVIDEOAUDIO = 6;
            AUDIOVOLUME = 7;
            ALERT = 8;
            HISTORY = 9;
            TERMINATED = 10;
            CONNECTED = 11;
      }
}

message TextMessage{
      required int64 time = 1;
      required string text = 2;
      required string idMonitoredDevice = 3;
}

message BaseMessage {
      required Type type = 1;
      required string idHash = 2;
      optional Authentication authMessage= 3;
      repeated SensorValue sensorValueMessage = 4;
      optional Command commandMessage = 5;
      optional Configuration configurationMessage =
6;
      repeated SensorInfo sensorInfoMessage = 7;
      repeated TextMessage textMessage = 8;
      enum Type {
            AUTHENTICATION = 1;
            SENSOR = 2;
            COMMAND = 3;
            NOTIFY = 4;
            CONFIGURATION = 5;
            INFO = 6;
            MESSAGE = 7;
            PING = 8;
      }
}

message SensorValue {
```

```
        required Type type = 1;
        repeated DataPoint value = 2;
        optional DeviceData dData = 3;
        enum Type{
              ECG = 1;
              EMG = 2;
              EDA = 3;
              TEMP = 4;
              RESPRATE = 5;
              LIGHT = 6;
              PULSEO2 = 7;
              MOTION = 8;
              GPS = 10;
              DEADDEVICE = 11;
              HR = 12;
              RR = 13;
              DEVICEINFO = 14;
        }
}

message DeviceData{
        optional double cpu = 1;
        optional double memory = 2;
        optional int64 uptime = 3;
        optional double battery = 4;
        optional string networkType = 5;
        optional double wirelessSignalStrength = 6;
        optional double cellSignalStrength = 7;
}

message DataPoint {
        required int64 time = 1;
        optional double value = 2;
        optional double motionX = 3;
        optional double motionY = 4;
        optional double motionZ = 5;
        optional double latitude = 6;
        optional double longitude = 7;
}
```

H

Anexo C – Testes de Sistema

Neste anexo são apresentados os vários testes realizados ao sistema.

1	
Descrição	*Login* no Sistema por parte de dois *Control Center*
Prioridade	Elevada
Requisito(s)	**REQ_CC_01**, **REQ_CC_02**, **REQ_S_04**, **REQ_S_05**
Condições	Servidor iniciado *Control Center* iniciado e indicação de ligação ao servidor
Ações/Dados necessários	Introduzir Nome de Utilizador, Palavra-Pass e clicar no botão de *Login*
Resultado Esperado	Entrada no sistema e redireccionamento para o ecrã principal em cada um dos *Control Center*
Estado	Passou
Detalhes	-

2	
Descrição	Monitorizar vários utilizadores num *Control Center*
Prioridade	Elevada
Requisito(s)	**REQ_CC_03, REQ_CC_04, REQ_CC_06-12, REQ_S_08, REQ_S_14**
Condições	Mais do que um dispositivo configurado como visível para o *Control Center* usado no teste Login efetuado com sucesso Visualizar ecrã inicial do *Control Center*
Ações/Dados necessários	-
Resultado Esperado	Do lado esquerdo do ecrã é possível visualizar uma lista com os vários utilizadores e as informações para cada um deles
Estado	Passou
Detalhes	-

3	
Descrição	Visualização de um utilizador em detalhe num *Control Center*
Prioridade	Elevada
Requisito(s)	**REQ_CC_13-23, REQ_S_08**
Condições	Dispositivo configurado como visível para o *Control Center* usado no teste Login efetuado com sucesso Visualizar ecrã inicial do *Control Center*
Ações/Dados necessários	Selecionar o utilizador da lista que se pretende ver Clicar sobre o utilizador pretendido
Resultado Esperado	Informação do utilizador selecionado surge nas secções de detalhe reservadas para o efeito
Estado	Passou
Detalhes	A informação pessoal está visível no topo central da janela e a informação dos sensores está imediatamente abaixo.

4	
Descrição	Ligação de um dispositivo ao sistema
Prioridade	Elevada

Requisito(s)	**REQ_D_01, REQ_D_02, REQ_S_01, REQ_S_03**
Condições	Servidor iniciado
Ações/Dados necessários	Ligar o dispositivo
Resultado Esperado	Conexão do dispositivo físico ao servidor com sucesso
Estado	Passou
Detalhes	A confirmação deste teste é feita através de dados de saída que são apresentados numa consola

5	
Descrição	Ligação de cinco dispositivo ao sistema
Prioridade	Elevada
Requisito(s)	**REQ_D_01, REQ_D_02, REQ_S_01, REQ_S_02, REQ_S_03**
Condições	Servidor iniciado
Ações/Dados necessários	Ligar o dispositivo físico Iniciar o simulador com quatro dispositivos simulados
Resultado Esperado	Conexão do dispositivo físico ao servidor com sucesso Conexão dos quatro dispositivos simulados com sucesso
Estado	Passou
Detalhes	A confirmação da ligação do dispositivo físico é feita através de dados de saída que são apresentados numa consola A confirmação da conexão dos dispositivos simulados é feita no próprio simulador

6	
Descrição	Envio de dados pelo dispositivo físico
Prioridade	Elevada
Requisito(s)	**REQ_D_03, REQ_D_06, REQ_S_07**
Condições	Servidor iniciado Dispositivo conectado e autenticado no servidor
Ações/Dados necessários	Conectar o conjunto dos sensores ao corpo
Resultado Esperado	Servidor a receber os dados com sucesso
Estado	Passou
Detalhes	A confirmação deste teste é feita através de dados de saída que são apresentados numa consola do lado do servidor Uma consola do lado do dispositivo demonstra os dados que estão a sair e outra consola do lado do servidor demonstra que este os está a receber

7	
Descrição	Envio de dados por cinco dispositivos ao mesmo tempo
Prioridade	Elevada
Requisito(s)	**REQ_D_03, REQ_D_06, REQ_S_07**
Condições	Servidor iniciado Dispositivo conectado e autenticado no servidor Simulador ligado com quatro dispositivos simulados e conectados ao servidor
Ações/Dados necessários	Conectar o conjunto dos sensores ao corpo para o dispositivo físico Iniciar envio de dados por parte dos dispositivos simulados
Resultado Esperado	Servidor a receber os dados dos vários dispositivos com sucesso
Estado	Passou

Detalhes	A confirmação deste teste é feita através de dados de saída que são apresentados numa consola do lado do servidor O simulador cria um ficheiro de *log* para cada dispositivo onde se pode verificar os dados de saída de cada um deles Neste caso, são também guardados *logs* do servidor para uma mais fácil análise dos dados

8	
Descrição	Envio de dados por dez dispositivos ao mesmo tempo
Prioridade	Elevada
Requisito(s)	**REQ_D_03, REQ_D_06, REQ_S_07**
Condições	Servidor iniciado Dispositivo conectado e autenticado no servidor Simulador ligado com nove dispositivos simulados e conectados ao servidor
Ações/Dados necessários	Conectar o conjunto dos sensores ao corpo para o dispositivo físico Iniciar envio de dados por parte dos dispositivos simulados
Resultado Esperado	Servidor a receber os dados dos vários dispositivos com sucesso
Estado	Passou
Detalhes	A confirmação deste teste é feita através de dados de saída que são apresentados numa consola do lado do servidor O simulador cria um ficheiro de *log* para cada dispositivo onde se pode verificar os dados de saída de cada um deles Neste caso, são também guardados *logs* do servidor para uma mais fácil análise dos dados

9	
Descrição	Envio de alerta amarelo de temperatura para o *Control Center*
Prioridade	Elevada
Requisito(s)	**REQ_S_10, REQ_CC_27, REQ_CC_28, REQ_CC_36, REQ_CC_38, REQ_D_08, REQ_D_10**
Condições	Servidor iniciado *Control Center* conectado Um dispositivo simulado a enviar dados
Ações/Dados necessários	Simular um alerta amarelo de temperatura para o dispositivo
Resultado Esperado	No *Control Center* deve aparecer, na secção de alertas, um alerta amarelo com o valor da temperatura destacado
Estado	Passou
Detalhes	Este teste só foi possível executar usando o simulador criado A temperatura vai aumentando até estabilizar entre 37,5 e 39 graus Os limites estão definidos na Base de Dados

10	
Descrição	Envio de alerta vermelho de taxa de respiração para o *Control Center*
Prioridade	Elevada
Requisito(s)	**REQ_S_10, REQ_CC_27, REQ_CC_28, REQ_CC_37, REQ_CC_39, REQ_D_08, REQ_D_11**
Condições	Servidor iniciado

	Control Center conectado Um dispositivo simulado a enviar dados
Ações/Dados necessários	Simular um alerta vermelho do sensor de taxa de respiração para o dispositivo
Resultado Esperado	No *Control Center* deve aparecer, na secção de alertas, um alerta vermelho com o valor da taxa de respiração destacado
Estado	Passou
Detalhes	Este teste só foi possível executar usando o simulador criado A taxa de respiração vai aumentando até estabilizar entre os valores 20 e 25 Os limites estão definidos na Base de Dados

11	
Descrição	Envio de alerta vermelho de temperatura para o *Message Broker* do *SALUS*
Prioridade	Elevada
Requisito(s)	**REQ_D_08, REQ_D_10**
Condições	*Message Broker* funcional Um dispositivo simulado a enviar dados
Ações/Dados necessários	Simular um alerta vermelho do sensor de temperatura para o dispositivo
Resultado Esperado	O *Message Broker* deverá receber o alerta vindo do dispositivo
Estado	Passou
Detalhes	Este teste só foi possível executar usando o simulador criado A temperatura vai aumentando até atingir valores superiores a 39 graus Os *logs* do *Message Broker* estão incluídos na própria API do mesmo

12	
Descrição	Envio de alerta vermelho de temperatura para o *Message Broker* do *SALUS*
Prioridade	Elevada
Requisito(s)	**REQ_D_08**
Condições	*Message Broker* funcional Um dispositivo simulado a enviar dados
Ações/Dados necessários	Simular um alerta vermelho do sensor de temperatura para o dispositivo
Resultado Esperado	O *Message Broker* deverá receber o alerta vindo do dispositivo
Estado	Passou
Detalhes	Este teste só foi possível executar usando o simulador criado A temperatura vai aumentando até atingir valores superiores a 39 graus Os *logs* do *Message Broker* estão incluídos na própria API do mesmo

13	
Descrição	Posição “de pé” do utilizador do dispositivo
Prioridade	Elevada
Requisito(s)	**REQ_D_12, REQ_CC_22, REQ_CC_23**
Condições	Dispositivo físico ligado e carregado pelo utilizador

Ações/Dados necessários	O utilizador tem de ficar parado, em pé e direito
Resultado Esperado	No *Control Center* é esperado que apareça a imagem relativa ao utilizador de pé
Estado	Passou
Detalhes	Qualquer movimento pode resultar numa análise indevida da posição do utilizador, podendo detetar que este está a andar

14	
Descrição	Posição "a andar" do utilizador do dispositivo
Prioridade	Elevada
Requisito(s)	**REQ_D_12, REQ_CC_22, REQ_CC_23**
Condições	Dispositivo físico ligado e carregado pelo utilizador
Ações/Dados necessários	O utilizador tem de andar, numa postura direita
Resultado Esperado	No *Control Center* é esperado que apareça a imagem relativa ao utilizador estar a andar
Estado	Passou
Detalhes	Caso haja inclinações do corpo, podem ser detetadas posições erradas ou, em alguns casos, nenhuma posição No geral e depois de realizar este teste várias vezes, as falhas ou imprecisões são mínimas, considerando o teste passado.

15	
Descrição	Posição "deitado" do utilizador do dispositivo
Prioridade	Elevada
Requisito(s)	**REQ_D_12, REQ_CC_22, REQ_CC_23**
Condições	Dispositivo físico ligado e carregado pelo utilizador
Ações/Dados necessários	O utilizador tem de ficar deitado
Resultado Esperado	No *Control Center* é esperado que apareça a imagem relativa à posição horizontal do utilizador
Estado	Passou
Detalhes	-

16	
Descrição	Queda do utilizador do dispositivo
Prioridade	Elevada
Requisito(s)	**REQ_D_07, REQ_D_12, REQ_CC_22, REQ_CC_23**
Condições	Dispositivo físico ligado e carregado pelo utilizador
Ações/Dados necessários	Criar o impacto do dispositivo numa superfície
Resultado Esperado	No *Control Center* é esperado que apareça a imagem relativa à queda É esperado também o disparo de um alerta no *Control Center*
Estado	Passou
Detalhes	Este teste foi realizado várias vezes para calibra o algoritmo de *Man Down* para detecção da queda Diferentes limites definidos no algoritmo, produziam resultados diferentes

17	
Descrição	Visualização de sinal de vídeo em tempo-real
Prioridade	Elevada

Requisito(s)	**REQ_D_13, REQ_CC_30, REQ_CC_31, REQ_CC_32**
Condições	Servidor ligado *Control Center* ligado e autenticado Dispositivo ligado com câmara acopolada
Ações/Dados necessários	No *Control Center* é necessário clicar no botão para inicio de vídeo na secção que serve para o efeito
Resultado Esperado	Deve ser inicializado o vídeo no espaço reservado para tal na secção de vídeo
Estado	Passou
Detalhes	Existe um pequeno período de tempo para ser feita a ligação antes de se iniciar o vídeo Após a inicialização do sinal de vídeo, poderá ser definida a qualidade desse mesmo sinal

18	
Descrição	Comunicação via áudio do utilizador do dispositivo
Prioridade	Elevada
Requisito(s)	**REQ_D_14, REQ_CC_33, REQ_CC_34, REQ_CC_35**
Condições	Servidor ligado *Control Center* ligado e autenticado Dispositivo ligado com microfone acopolado
Ações/Dados necessários	No *Control Center* é necessário clicar no botão para inicio transmissão de audio
Resultado Esperado	Deve ser inicializado o sinal de aúdio no espaço reservado para áudio e vídeo
Estado	Passou
Detalhes	Existe um pequeno período de tempo para ser feita a ligação antes de se iniciar o aúdio Depois de iniciado, pode ser regulado o volume do sinal

19	
Descrição	Guardar os dados na Base de Dados
Prioridade	Elevada
Requisito(s)	**REQ_S_11**
Condições	Servidor ligado Base de Dados operacional Dispositivo ligado e a enviar dados
Ações/Dados necessários	-
Resultado Esperado	Deve ser verificado na Base de Dados que os dados foram inseridos com sucesso
Estado	Passou
Detalhes	Teste feito com o dispositivo físico ao dispor A verificação do resultado foi feita recorrendo a *logs* e à análise dos dados contidos na BD usada

20	
Descrição	Visualizar histórico relativo a um utilizador
Prioridade	Média
Requisito(s)	**REQ_S_13, REQ_CC_26**
Condições	Servidor ligado Base de Dados operacional

	Dispositivo ligado e a enviar dados *Control Center* inicializado, autenticado e a monitorizar o utilizador
Ações/Dados necessários	Clicar no botão de histórico situado na secção de sumário do utilizador Definir data de inicio e data de fim para visualização de histórico
Resultado Esperado	Deve ser apresentado o histórico dos dados para o período de tempo definido
Estado	Passou
Detalhes	Teste feito com o dispositivo físico ao dispor Foram usados *logs* da Base de Dados para verificar que os dados lidos eram corretos

21	
Descrição	Visualizar o gráfico aumentado que apresenta os valores do electrocardiograma
Prioridade	Baixa
Requisito(s)	**REQ_S_13**
Condições	Dispositivo ligado e a enviar dados *Control Center* inicializado, autenticado e a monitorizar o utilizador
Ações/Dados necessários	Junto ao gráfico do ECG, clicar no botão de *zoom*
Resultado Esperado	Deve ser apresentado o gráfico do ECG aumentado, sobreposto ao ecrã principal do *Control Center*
Estado	Passou
Detalhes	-

22	
Descrição	Visualizar o gráfico aumentado que apresenta os valores de taxa de respiração
Prioridade	Baixa
Requisito(s)	**REQ_S_13**
Condições	Dispositivo ligado e a enviar dados *Control Center* inicializado, autenticado e a monitorizar o utilizador
Ações/Dados necessários	Junto ao gráfico da taxa de respiração, clicar no botão de *zoom*
Resultado Esperado	Deve ser apresentado o gráfico da taxa de respiração aumentado, sobreposto ao ecrã principal do *Control Center*
Estado	Passou
Detalhes	-

23	
Descrição	Analisar o estado do dispositivo
Prioridade	Média
Requisito(s)	**REQ_CC_41, REQ_CC_42**
Condições	Dispositivo ligado e a enviar dados *Control Center* inicializado, autenticado e a monitorizar o utilizador
Ações/Dados necessários	-
Resultado Esperado	Deve ser possível a visualização de informações relativas ao dispotivo na secção de sumário (bateria, tipo e qualidade de rede

	usada)
Estado	Passou
Detalhes	-

24	
Descrição	Dez *Control Center* e vinte dispositivos a funcionar ao mesmo tempo
Prioridade	Elevada
Requisito(s)	**REQ_NF_07, REQ_NF_08**
Condições	Dispositivo físico ligado e a enviar dados Servidor ligado
Ações/Dados necessários	Simular vinte dispositivos usando o simulador criado Inicializar dez *Control Center*'s
Resultado Esperado	O sistema deve ser capaz de operar sem qualquer tipo de problema
Estado	Passou
Detalhes	Foi dada especial atenção à performance do desenho de gráficos nos vários *Control Center*