**Universidade de Aveiro** Departamento de Eletrónica, Telecomunicações e Informática
**2017**

**Luís Tiago Marques Duarte**

# Comparação e implementação de plataformas de backend para soluções mHealth

# Assessment and implementation of backend frameworks for mHealth applications

**Luís Tiago Marques Duarte**

# Comparação e implementação de plataformas de backend para soluções mHealth

# Assessment and implementation of backend frameworks for mHealth applications

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Ilídio Castro Oliveira, professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

**o júri**

presidente

**Professor Doutor Arnaldo Silva Rodrigues de Oliveira**
Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

vogais

**Professor Doutor Rui Pedro de Magalhães Claro Prior**
Professor auxiliar da Faculdade de Ciências da Universidade do Porto

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

**agradecimentos** Agradeço à minha família, amigos e companheiros o apoio e incentivo necessário para que terminasse esta etapa tão importante.

**Palavras Chave** *mHealth*, Computação Móvel, Plataformas de *backend*, Monitorização fisiológica

**Resumo**

Os dispositivos móveis são cada vez mais utilizados na recolha de dados fisiológicos, quer para uso pessoal ou quer para suporte à prestação de cuidados de saúde. Esta área é mais conhecida como *mHealth*. O programador de aplicações *mHealth* precisa de desenvolver o *front-end* móvel, para os utilizadores finais e o *backend*, para a integração de sistemas e persistência dos dados recolhidos. Existem várias possibilidades de escolha relativamente à arquitetura a utilizar para o *backend*, que vão desde tecnologias empresariais genéricas a tecnologias específicas de determinados domínios da área de saúde. Neste trabalho, estudamos três plataformas para o *backend* de aplicações *mHealth (Open mHealth, HL7 FHIR, Google Fit)*. Foram realizadas implementações exploratórias e aprendidas lições.

De seguida, mostramos a utilização da plataforma *Open mHealth* numa aplicação mais abrangente, cobrindo casos típicos da utilização de *mHealth* num cenário de investigação: criação e gestão de estudos, monitorização ambulatória de participantes e análise dos dados por especialistas em saúde. A arquitetura proposta foi implementada num protótipo funcional, permitindo a monitorização de frequência cardíaca, eletrocardiograma e acelerómetro através de smartphones *Android*, e a revisão dos casos por especialistas numa aplicação *web* dedicada.

**Keywords** mHealth, Mobile computing, Backend platforms, Physiologic monitoring

**Abstract** Mobile devices are increasingly being used in the collection of physiological data, for personal use or health care support. This field of application is known as mHealth. The developer of mHealth applications need to address the mobile front-end, for the final users, and the backend, for systems integration and long-term persistence. Several architectural choices can be adopted for the backend, ranging from general purpose enterprise-level technologies, to health-domain specific frameworks. In this work, we study selected candidate frameworks for the backend of mHealth applications (Open mHealth, HL7 FHIR, Google Fit). Exploratory implementations were conducted and lessons learned.

We then show the use of the Open mHealth framework in a more comprehensive application, covering typical mHealth use cases in a research scenario: studies creation and management, ambulatory monitoring of participants and data analysis by health experts. The proposed architecture was implemented in a functional prototype, allowing for heart rate, electrocardiogram, accelerometer and activity monitoring with Android smartphones, and cases review by experts in a dedicated web application.

# Conteúdo

# Lista de Figuras

# Lista de Tabelas

# Lista de Abreviações e Acrónimos

**API** Application Programming Interface.

**CRUD** Create, Read, Update and Delete.
**CSS** Cascading Style Sheets.

**DSU** Data Storage Unit.

**ECG** Eletrocardiograma.

**FHIR** Fast Healthcare Interoperability Resources.

**HDP** Health Device Profile.
**HL7** Health Level Seven.
**HTML** HyperText Markup Language.
**HTTP** Hypertext Transfer Protocol.

**I&D** Investigação e Desenvolvimento.
**IEETA** Instituto de Engenharia Eletrónica e Telemática de Aveiro.

**JSON** JavaScript Object Notation.
**JVM** Java Virtual Machine.

**mHealth** Mobile Health.

**OMH** Open mHealth.
**OMS** Organização Mundial de Saúde.

**REST** Representational State Transfer.
**RIM** Reference Information Model.

**SDK** Software Development Kit.
**SPP** Serial Port Profile.

**URI** Uniform Resource Identifier.

**XML** eXtensible Markup Language.

# Capítulo 1

# Introdução

## 1.1 Motivação

A grande disponibilidade de dispositivos móveis e a pressão nos sistemas de saúde levaram a uma explosão do número de aplicações móveis para a saúde (*mHealth*). Estas aplicações são cada vez mais relevantes para monitorização e acompanhamento de doentes, especialmente com doenças crónicas(1), mas quando se trata de escolher arquiteturas de sistema para o seu desenvolvimento não há respostas óbvias. Surge então a necessidade de procurar e escolher um *backend* para aplicações *mHealth* com o objetivo de se guardar e posteriormente consultar dados fisiológicos e demográficos dos doentes. Dentro dos *backends* existentes é necessário encontrar um que esteja apto para ser utilizado e que consiga dar suporte a uma aplicação *mHealth*, ou perceber as alterações que seriam necessárias para que fosse possível utilizar um destes *backends* disponíveis. Uma das características a ter em conta ao escolher um destes *backends* disponíveis é perceber se os dados se encontram normalizados para uma possível exportação ou importação de dados clínicos ou hospitalares para uma possível integração num sistema externo.

No Instituto de Engenharia Eletrónica e Telemática de Aveiro (IEETA), a unidade que acolheu este trabalho, a atividade de Investigação e Desenvolvimento (I&D) é constante e o desenvolvimento de aplicações móveis é frequente, e muitas vezes integrado em cenários com experiências que recolhem parâmetros fisiológicos. A identificação de recomendações práticas para a engenharia do *backend* de uma aplicação de *mHealth* tem também interesse prático para as atividades deste instituto.

## 1.2 Objetivos

Para o desenvolvimento desta dissertação temos como objetivo principal encontrar um *backend* que esteja apto a ser utilizado, ou que o possamos utilizar após algumas adaptações, para servir como suporte a aplicações em *mHealth*, explorando e estudando as plataformas

existentes, para perceber o que cada uma delas tem para nos oferecer.

Como segundo objetivo, propomo-nos desenvolver um sistema integrado, servindo como prova de conceito, utilizando um problema típico de um projeto de I&D, que irá gerir a informação fisiológica de pessoas através de uma aplicação móvel. Estas pessoas são participantes de um estudo, podendo estas ser monitorizadas remotamente por especialistas através de uma aplicação *web*. Como suporte destas aplicações deverá ser utilizado o *backend* escolhido no primeiro objetivo.

## 1.3 Estrutura da dissertação

Esta dissertação está organizada por 8 capítulos.

No capítulo 1, apresentamos a motivação e os objetivos para o desenvolvimento da dissertação.

No capítulo 2, revemos alguns conceitos e o estado da arte relativamente às aplicações *mHealth* e arquiteturas relacionadas. Apresentamos ainda algumas plataformas de *backend* que podem vir a ser utilizadas neste tipo de aplicações, realizando uma análise comparativa entre eles.

No capítulo 3, descrevemos os casos de uso do sistema a desenvolver, partindo de um cenário concreto e extraindo os requisitos.

No capítulo 4, fazemos uma avaliação exploratória das várias soluções disponíveis para o *backend* de uma aplicação *mHealth*. Neste capítulo foram contempladas as dificuldades encontradas e as adaptações necessárias relativamente a cada *backend* elaborando uma comparação e concluindo qual o *backend* a ser utilizado.

No capítulo 5, apresentamos uma possível arquitetura para a solução do sistema a desenvolver.

No capítulo 6, descrevemos a implementação do sistema tendo em conta o *backend* escolhido no capítulo 4, e a arquitetura proposta no capítulo 5.

No capítulo 7, discutimos os resultados obtidos e damos algumas recomendações para o desenvolvimento de aplicações de *mHealth*.

No capítulo 8, apresentamos as conclusões obtidas e o trabalho futuro a ser desenvolvido.

# Capítulo 2

# Aplicações *mHealth* e arquiteturas relacionadas

## 2.1 *eHealth*: tecnologias de informação ao serviço da saúde

Nos dias que correm toda a sociedade tem disponível muito facilmente a possibilidade de trocar informação entre si, podendo permanecer numa constante troca e partilha de informação. Com esta realidade têm surgido novas tecnologias, ferramentas e aparelhos que facilitam, tendo em conta a sua implementação e uso, o fortalecimento da informação.

A área da saúde cada vez mais está identificada com esta realidade, as tecnologias da informação e da comunicação têm sido um aliado para aumentar a eficiência e melhorar a qualidade da prestação de serviços na área da saúde, ajudando a população com o seu bem-estar (2).

Surge então o termo *electronic Health* mais conhecido por *eHealth*, inicialmente utilizado por profissionais de saúde, investigadores e no âmbito académico, e está aliado a cenários que envolvem cuidados de saúde, dispositivos eletrónicos e a Internet. O termo que até 1999 pouco era utilizado, atualmente é um termo que não define unicamente a "Medicina na Internet", mas também tudo aquilo que esteja relacionado com computadores e medicina (3). Podemos ver o *eHealth* como designação abrangente para enquadrar a transformação digital da prestação de cuidados de saúde.

A Organização Mundial de Saúde (OMS) define *eHealth* (4) como uma "utilização rentável e segura das tecnologias de informação e comunicação no apoio à saúde e às áreas relacionadas com a saúde, incluindo os serviços de saúde, vigilância na saúde, literatura na saúde, educação na saúde e investigação na área da saúde."

Uma definição apresentada em (3): "*eHealth* é uma área emergente na interceção da informática médica, saúde pública e negócios, referindo-se aos serviços de saúde e entrega de informação através da Internet ou tecnologias semelhantes. Pensando de forma abrangente, o termo não é só um desenvolvimento técnico, mas também uma nova forma de pensar, uma

atitude, um compromisso para a rede, um pensamento global para melhorar o cuidado da saúde local, regional e global com o uso das tecnologias de informação e comunicação".

Alguns requisitos também são colocados (3) para um sistema ser considerado *eHealth*. Entre eles, temos: eficiência; melhoria ao nível de qualidade do cuidado da saúde; sistema baseado em evidências; incentivar uma nova relação entre o utente e o profissional de saúde; ser de fácil uso.

Temos também a prestação de cuidados de saúde a longa distância, mais concretamente a telemedicina. A telemedicina permite a pessoas que estejam localizadas em ambientes mais rurais os cuidados de saúde mínimos. A maior parte dos habitantes dos países em desenvolvimento moram em zonas rurais dificultando o acesso a serviços de saúde, médicos e tratamentos.(5)

A OMS define telemedicina como (6) uma "prestação de cuidados de serviços de saúde em situações em que a distância é um fator crítico, por qualquer profissional de saúde usando tecnologias de informação e comunicação para a partilha de informação válida para ser feito o diagnóstico, o tratamento e a prevenção da doença e danos físicos, pesquisa e avaliação, e para a formação contínua dos prestadores de cuidados de saúde, com o objetivo da melhoria da saúde dos indivíduos e das suas comunidades".

A grande vantagem que existe com a utilização da telemedicina é a eliminação do custo de deslocamento até uma unidade de saúde e a possibilidade de realização de consultas por especialistas de forma remota.

Tendo em conta que a telemedicina e *eHealth* estão diretamente relacionados, surge então a necessidade de existir uma plataforma que possibilite o acesso aos médicos das informações obtidas dos seus utentes a qualquer momento e em tempo real, estando estes geograficamente em locais distintos.

## 2.2 *mHealth*: computação móvel em saúde

O termo *Mobile Health (mHealth)* é definido segundo a OMS como uma componente da *eHealth* (7) sendo esta definida como o recurso a dispositivos móveis (como por exemplo o telemóvel, *tablet* e outros dispositivos sem fios) para a prática de cuidados de saúde e médicos.

Uma aplicação *mHealth* pertence também à componente de *eHealth*, assim como à telemedicina. As aplicações de *mHealth* têm como o objetivo oferecer cuidados de saúde e permitir a médicos a monitorização de diferentes parâmetros desde qualquer lugar e até em movimento, usando dispositivos que fazem parte da componente *eHealth* e transmitindo esses dados através de dispositivos móveis. Como a *mHealth* permite superar as barreiras da localização entre os utentes e os médicos podemos dizer que a telemedicina também está presente nestas aplicações.

Depois das redes móveis começarem a suportar 3G e 4G para transporte de dados, a comu-

nicação móvel tem sido a principal atração de investigadores e de comunidades empresariais. Com esta inovação nas redes móveis a prestação de cuidados de saúde em qualquer momento e em qualquer lugar, superando as barreiras geográficas, temporais e até organizacionais deixou de ser um problema (8).

As áreas abrangidas podem ser mesmo bastantes, basta haver aplicações desenvolvidas para o devido efeito e que estejam preparadas para receber dados dos respetivos dispositivos. Este último ponto não é obrigatório pois os dados podem ser obtidos em dispositivos isoladamente e adicionados manualmente na aplicação.

Tomando por exemplo a diabetes, uma doença crónica, há a necessidade de se monitorizar a concentração de glicose no sangue de doentes com regularidade. Vamos agora analisar um estudo feito nesta área listando algumas aplicações para esta finalidade (8).

| **Aplicação** | **Descrição** | **Monitorização de parâmetros** | **Leitura Autónoma** |
|---|---|---|---|
| Daily Carb (9) | Uma aplicação que possibilita uma monitorização diária dos nutrientes ingeridos, hidratos de carbono, gorduras e água, assim como leituras da glicose, pressão arterial, frequência cardíaca, peso, exercício feito, medicação e insulina ingerida. | Sim | Não |
| Glucose Buddy (10) | Esta aplicação possibilita aos utilizadores a inserção manual de valores da glicose, hidratos de carbono e insulina ingerida, assim como outras atividades. | Sim | Não |
| GoMeals (11) | Esta aplicação foi desenvolvida para ajudar o utilizador na escolha de alimentos, atividade e monitorização da glicose para um estilo de vida saudável. | Sim | Não |

Tabela 2.1: Conjunto de aplicações *mHealth* para ajudar a monitorizar a diabetes

A maioria das aplicações *mHealth* têm como foco a capacidade de monitorização dos utentes remotamente. Um médico ou um utente podem facilmente aceder aos mesmos dados médicos em qualquer momento e em qualquer lugar através de dispositivos móveis como computador, *tablet* ou *smartphone*. Para isto acontecer é necessário os dados estarem guardados num servidor para serem acedidos tanto pelos médicos como pelos utentes.

Dentro dos dados de saúde mais comuns temos a frequência cardíaca, o Eletrocardiograma (ECG) e o SpO2(Oxímetro de pulso) (12). Estes tipos de dados são os mais comuns, pois são os utilizados quando os doentes estão internados e em avaliação e supervisão contínua. Dentro destes tipos de dados, considerando agora no caso das aplicações *mHealth*, podemos verificar que dispositivos que consigam recolher ECG conseguem também recolher frequência cardíaca, que dispositivos que consigam recolher pressão arterial conseguem também recolher frequência cardíaca, ou seja, verificamos que o tipo de dados mais comum poderá ser a frequência cardíaca, pois a percentagem de dispositivos eletrónicos que tem este sensor integrado é mesmo enorme. Leituras contínuas de um ECG serão também bastante importantes nas aplicações de *mHealth* para, por exemplo, a deteção de arritmia.

## 2.3 Padrões de arquitetura em aplicações móveis

Uma aplicação móvel comum é constituída por três camadas principais, entre elas a camada de apresentação que é a camada que compõe a interface com o utilizador da aplicação, a camada de negócio e a camada correspondente aos dados utilizados e guardados pela aplicação. Existe duas perspetivas diferentes ao desenvolver uma aplicação móvel, umas delas é o desenvolvimento de uma aplicação "rica" onde a camada de negócio e a camada de dados estão guardadas no próprio dispositivo. A outra perspetiva é o desenvolvimento de uma aplicação "leve" onde a camada de negócio e a camada de dados está guardada num servidor.

No caso de uma aplicação necessitar apenas de um processamento local num cenário ocasional, considera-se desenvolver uma aplicação "rica", ou seja, uma aplicação que não tem dependências de qualquer tipo de servidor. Quando uma aplicação tem dependências de servidores considera-se uma aplicação "leve". Uma aplicação "rica" será uma aplicação mais complexa e difícil de manter pois todas as alterações terão que ser efetuadas ao nível da aplicação. Na figura 2.1 [1] podemos ver uma arquitetura comum de uma aplicação móvel. (13)

[1]Imagem retirada do livro *Mobile Apllications: Architecture, Design, and Development*

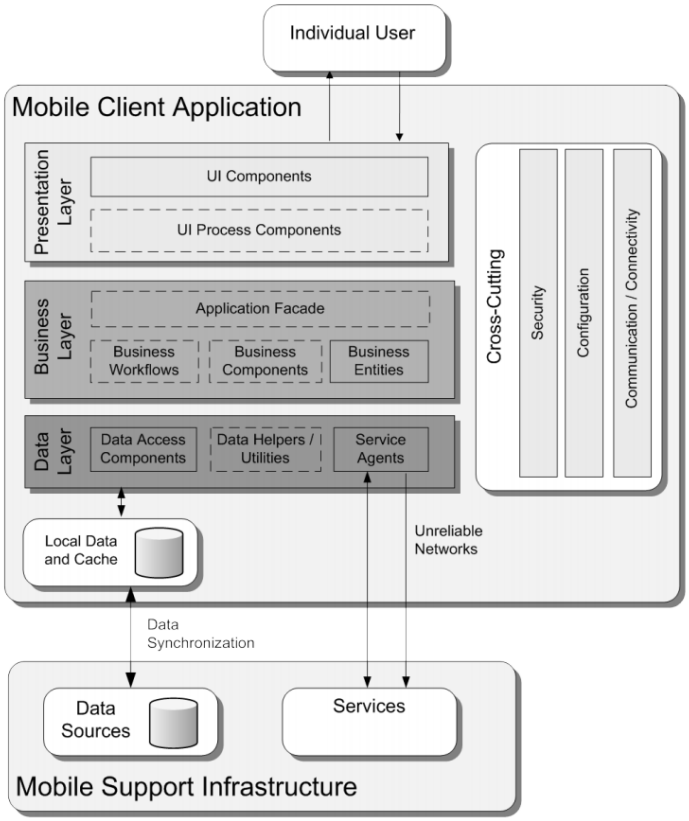


Figura 2.1: Arquitetura típica de uma aplicação móvel. (13)

No desenvolvimento de uma aplicação móvel têm que se ter em conta vários fatores importantes para garantir que a aplicação tem os requisitos necessários e é executável em qualquer *smartphone*. A lista de fatores a ter em conta é bastante alargada, mas entre eles temos: Autenticação e Autorização, Armazenamento em *Cache*, Comunicação e Acesso aos dados (13). Cada um destes fatores será analisado com mais detalhe de seguida.

### 2.3.1 Autenticação e Autorização

Uma estratégia de autenticação e autorização eficaz é importante para a segurança e fiabilidade de uma aplicação. Uma fraca autenticação pode deixar a aplicação vulnerável a uma utilização não autorizada. É necessário perceber que existe uma diferença entre autenticação e autorização. A autenticação é o processo de identificação de um utilizador com um identificador único e um elemento secreto (por exemplo uma palavra-passe). Um processo de autenticação garante, após o mesmo, que se trata de um utilizador específico. A autorização é o processo de controlo de ações sobre um serviço. Não indica um utilizador específico por si só. Para isto existe o protocolo de autorização OAuth (14) que é um protocolo padrão para a autorização. Nos dias de hoje grande parte das aplicações utiliza este protocolo de auto-

rização, atualmente vai na versão 2.0 e utiliza *tokens* para ser dada a permissão de acesso a um determinado recurso (14). De seguida vamos perceber um pouco mais sobre este protocolo de autorização para se perceber a sua utilidade.

### Protocolo de autorização *OAuth*

O *Open Authorization Protocol - OAuth* é um protocolo de autorização que foi desenvolvido com o objetivo de solucionar os problemas relacionados com a gestão de identidades (15).

Na versão 2.0 do protocolo são definidos quatro pontos de contacto necessários para a compreensão do fluxo de execução deste protocolo, são eles: *Resource Owner* - O proprietário do recurso, que é a entidade que tem o poder de conceder a permissão de acesso, aos seus recursos; *Resource Server* - O servidor de recursos, que é o responsável por guardar e responder às solicitações de acesso aos recursos protegidos, utilizando *tokens* de acesso; *Client* - Cliente, que é uma aplicação, que realiza solicitações de acesso aos recursos protegidos, ao servidor de recursos, em nome do proprietário, dono do recurso, após a obtenção de sua autorização; *Authorization Server* - O servidor de autorização, que é responsável por emitir *tokens* de acesso aos clientes, após autenticar e obter autorização do proprietário dos recursos (14). Na figura 2.2 é apresentado o fluxo do protocolo OAuth 2.0 mostrando a interação entre estes quatro pontos de contacto.

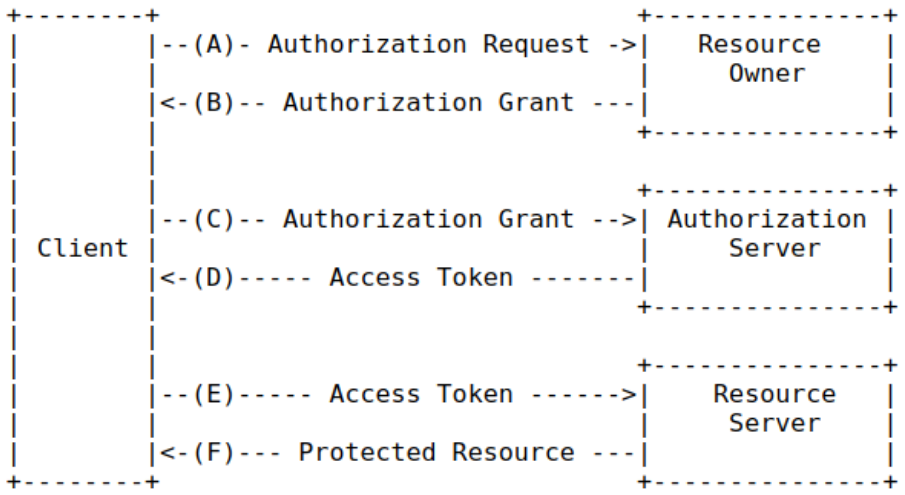


Figura 2.2: Fluxo do Protocolo de autorização OAuth 2.0. (14)

(A) O *Client* pede autorização ao *Resource Owner* para aceder aos seus recursos.

(B) Assumindo que o *Resource Owner* autoriza o acesso, o *Client* recebe um *authorization grant* (garantia de autorização). Essa credencial representa a autorização concedida pelo *Resource Owner*.

(C) O *Client* pede um *token* de acesso ao *Authorization Server*, enviando o *authorization grant*.

(D) Assumindo que o *Client* foi autorizado com sucesso e que o *authorization grant* é válido, o *Authorization Server* gera um *token* de acesso, sendo este enviado para o *Client*.

(E) O *Client* pede acesso a um recurso protegido pelo *Resource Server*, e autentica-se utilizando o *token* de acesso.

(F) Assumindo que o *token* de acesso é válido, o *Resource Server* responde ao pedido do *Client* enviando o recurso pedido.

### 2.3.2 Armazenamento em *Cache*

A utilização da *cache* do dispositivo pode ser muito importante para aumentar o desempenho e a capacidade de resposta de uma aplicação. Quando não existe ligação à internet, a aplicação deve suportar a execução das operações principais, isto é, se uma aplicação tem como objetivo obter a frequência cardíaca de um paciente, é suposto se conseguir obter estes dados dos sensores mesmo sem estes poderem ser guardados no servidor, ou seja, podem ser guardados em *cache* até que se tenha possibilidade de fazer o envio para o servidor.

Recentemente a Google fez recomendações para a utilização de uma componente da arquitetura do Android denominada por Biblioteca de Persistência (16) que é utilizada para a criação de repositórios locais no sentido de *cache*, tornando as aplicações mais resilientes à variabilidade da qualidade do serviço de uma rede.

### 2.3.3 Comunicações

Os dispositivos móveis comunicam essencialmente com tecnologia sem fios. A comunicação com servidores decorre habitualmente sobre IP. Na transmissão entre servidores temos que ter em conta que temos que proteger os dados que estarão a ser transportados contra roubo ou adulteração. Para comunicação entre dispositivos móveis é utilizada a tecnologia *bluetooth*.

#### *Bluetooth*

O *bluetooth* é uma tecnologia que foi criada em 1994, como uma alternativa sem fios para cabos de dados através da utilização de transmissões de rádio para ligar os dispositivos relativamente próximos, e transferir dados entre eles. Esta tecnologia foi criada como um padrão aberto para permitir a conectividade e a colaboração entre diferentes produtos e indústrias (17). Esta tecnologia está presente em grande percentagem de dispositivos eletrónicos vendidos atualmente, e quase todos os *smartphones* têm esta tecnologia.

Para a comunicação entre os sensores e os dispositivos móveis ser estabelecida ambos têm que ter pelo menos uma interface de comunicação em comum, isto é, um sensor *bluetooth* tem

um conjunto de perfis *bluetooth* a partir do qual os dispositivos móveis podem se conectar e estabelecer uma ligação recolhendo os dados. Os perfis disponibilizam padrões que devem ser respeitados para permitir que dispositivos utilizem o *bluetooth* de uma maneira normalizada. (18)

Um dos perfis é denominado de *Health Device Profile (HDP)*, foi um perfil criado com o objetivo de se normalizar a comunicação entre dispositivos e sensores na área da saúde utilizando a tecnologia *bluetooth*. Os mesmo dados podem ser enviados utilizando um perfil mais comum, este perfil é o *Serial Port Profile (SPP)* mas não está configurado para a comunicação de dispositivos na área da saúde (19).

### 2.3.4 Tecnologias para a implementação de serviços

Vamos agora rever tecnologias com interesse direto para a implementação e adaptação da plataforma que vai ser apresentada mais à frente.

#### *Web Services*

Os *Web Services* ou serviços *web* permitem que duas máquinas diferentes comuniquem entre si, ou que dois pedaços de código comuniquem entre eles. Para isto funcionar tem que existir um servidor que disponibilize uma *Application Programming Interface (API)* que é um conjunto de métodos, e então os clientes podem chamar esses métodos e comunicar com o servidor pela Internet por serviços *web*. Uma vantagem da utilização dos serviços *web* é que atualmente é uma tecnologia padrão, ou seja, é uma tecnologia que não é especifica de nenhuma linguagem de programação. Podem estar os serviços *web* desenvolvidos em Java, e os pedidos podem ser efetuados através de um cliente que está a correr em Python. O único formato de dados utilizado por um serviço *web* era o *eXtensible Markup Language (XML)*, e só apenas mais tarde começou a poder ser utilizado o *HyperText Markup Language (HTML)* e o JSON depois do aparecimento do *Representational State Transfer (REST)* (20).

#### REST *Web Services*

O REST é um estilo de arquitetura que utiliza o *Hypertext Transfer Protocol (HTTP)* para fazer chamadas entre máquinas e tem como objetivo fundamental facultar serviços para ajudar no desenvolvimento de aplicações (21). REST caracteriza uma arquitetura centrada em recursos, especificando que cada recurso é identificado por um *Uniform Resource Identifier (URI)*, mediante o qual um conjunto de operações pode ser aplicado através de uma interface uniforme. Esta interface padrão uniforme para a comunicação entre servidores e clientes é o HTTP e, em vez de declarar métodos, são aplicadas ações HTTP, tais como POST, GET, PUT e DELETE. Estas quatro ações podem ser mapeadas para as ações típicas de dados *Create, Read, Update and Delete (CRUD)*. A representação de cada recurso identificado por

um URI, pode variar, podendo ser representado em XML, HTML, JSON, entre muitas outras possibilidades (22).

**Vert.x**

É uma *framework* orientada a eventos e assíncrona, que permite a criação de aplicações facilmente escaláveis de uma forma simples, pode ser utilizada por pequenas aplicações, por aplicações *web* sofisticadas e modernas e ainda para o desenvolvimento de micro serviços HTTP/REST(23). O Vert.x é uma *framework* poliglota, ou seja, tem suporte para várias linguagens de programação, construída segundo o padrão reativo e assente na *Java Virtual Machine (JVM)*. O Vert.x é composto por um conjunto de componentes que são importantes para o desenvolvimento de aplicações. Entre eles temos:

- *Verticle*: Um *verticle* é definido por ter uma função principal (*main*), que corre um *script* específico. Um *verticle* pode ser escrito em múltiplas linguagens (Java, Javacript, Ruby, Groovy, Ceylon, Scala e Kotlin). Vários *verticles* podem ser executados na mesma instância do Vert.x.

- *EventLoops*: São *threads* que estão sempre a correr e estão sempre à escuta, de forma a verificar se existem operações a executar ou dados a tratar. A gestão destas *threads* é feita por uma instância do Vert.x, que aloca um número específico de *threads* a cada núcleo do servidor.

- *Handler*: É uma entidade que recebe mensagens do *eventbus* depois de se registar num determinado endereço.

- *EventBus*: Esta é uma característica do Vert.x bastante importante que permite a comunicação entre vários *verticles*. Para além disso, é possível haver uma comunicação não só entre *verticles* mas entre entidades que registem *handlers* num servidor (ex. clientes). Desta forma é possível a entrega das mensagens a todos os *handlers* que estiverem registados num determinado endereço. Esta característica é um padrão na troca de mensagens conhecido por *publish–subscribe*.

Na figura 2.3 podemos visualizar uma arquitetura simples do Vert.x. (24)

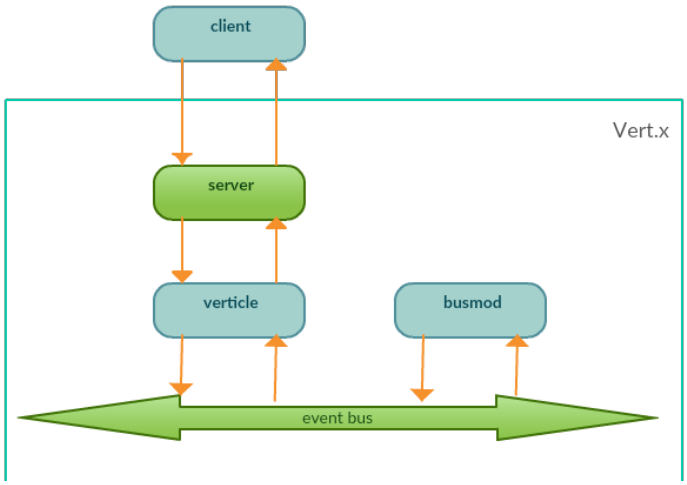


Figura 2.3: Arquitetura simples do Vert.x (Adaptado de: (24))

### JSON

O JSON é um formato de dados baseado em texto, é compacto e utilizado para troca de informação independentemente da linguagem a ser utilizada. O JSON é baseado em pares atributo-valor e a sua sintaxe é composta por quatro tipos de dados primitivos (*strings*, inteiros, *booleans* e *null*) e dois tipos estruturados (objetos e vetores) (25).

É um formato de dados fácil de gerar, compreender e interpretar. Analisando um documento JSON é intuitivo associar a sua aparência, estrutura e sintaxe a muitas linguagens de programação existentes, razão pela qual foi facilmente adotada como uma alternativa ao XML por parte de diversos programadores que viram no novo formato um modo mais natural e simples de definir dados.

## 2.4 O *backend* nas aplicações de *mHealth*

O termo *backend* é utilizado para agrupar tudo aquilo que é executado no lado do servidor e que dá suporte a uma aplicação. Tipicamente a arquitetura das aplicações *mHealth* utiliza a Internet e serviços *web* para disponibilizar uma interação entre os utentes e os médicos (8). Podemos ver na figura 2.4 que do lado do utente temos a aplicação móvel que permite ligar-se a sensores ou dispositivos especializados para fazer a colheita de novos dados fisiológicos e posteriormente fazer o envio para o servidor através de um serviço *web*. Do lado do servidor podemos ter sistemas de monitorização inteligentes e autónomos remotos que podem entrar em contacto com o utente ou médico em situações de emergência.

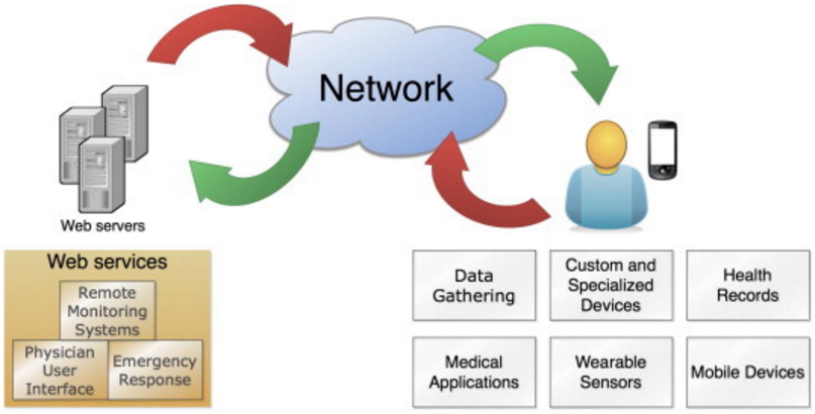


Figura 2.4: Arquitetura típica de uma aplicação *mHealth*. (8)

### 2.4.1 O *backend* no desenvolvimento de uma aplicação *mHealth*

As soluções de *mHealth* que abrangem estes seis princípios têm uma maior probabilidade de sucesso: Interoperabilidade, Integração, Inteligência, Socialização, Resultados e Compromisso (26). Seguindo estes princípios as aplicações desenvolvidas terão a capacidade de ser interoperáveis com diferentes dispositivos móveis e terão a capacidade de importar e exportar registos de saúde eletrónicos com e para outras aplicações. Terá também a capacidade de ser integrada em algumas atividades físicas dos pacientes e de analisar os dados fornecendo uma monitorização em tempo real. Será também importante a partilha dos dados em redes sociais e em plataformas de saúde para receber apoio e algumas recomendações. A aplicação terá também que ter a capacidade de se adaptar à utilização dos pacientes tendo em conta os dados recolhidos para posteriormente preparar a aplicação para um melhor apoio e monitorização(26).

Apesar das aplicações *mHealth* estarem numa crescente popularidade nos últimos anos, muitas delas são rejeitadas ou não utilizadas pelo público alvo pretendido. Para analisar esta realidade foi feito um estudo para se saber os requisitos e as considerações a ter durante e

antes do desenvolvimento de uma aplicação *mHealth*(27).

Depois deste estudo e discussão chegaram à conclusão que o desenvolvimento de uma aplicação deve estar dividida em 4 partes distintas criando uma pipeline de desenvolvimento, entre elas temos: Preparação, Desenvolvimento do *Back-End*, Desenvolvimento do *Front-End* e Lançamento da Aplicação(27).

A secção de Preparação descreve as etapas que estão envolvidas na preparação da aplicação, identificando o problema que vai ser resolvido, uma proposta de solução e o público alvo. É constituída por várias subsecções, entre elas temos: Propósito, Tipo de aplicação e Ética/ Regulamentos (27).

Relativamente ao Desenvolvimento do *Back-End* tem que se ter em consideração que vamos estar a lidar com dados e informações sensíveis e pessoais relacionados com o utente. Por isto deve se ter em atenção o envio, registo e armazenamento dos dados e informações no servidor. Para isso temos que ter em atenção algumas questões relacionadas com a segurança e privacidade dos dados (27).

A secção Desenvolvimento do *Front-End* serve para se decidir o desenvolvimento da aplicação. Para isso deverá-se ter em conta o público alvo, desenvolvendo um ambiente com uma boa experiência de utilização satisfazendo ao máximo, os especialistas e os utentes envolvidos. Um dos requisitos da interface de utilizador deverá ser direcionada para as necessidades do utente e do especialista. Existem também cuidados a ter em conta, mas isso depende da aplicação tendo em conta a sua finalidade e considerações. Nestas considerações podemos incluir que a aplicação pode vir a ser utilizada por pessoas cegas, surdas, pessoas com mobilidade limitada ou dificuldades de aprendizagem, contexto social, contexto cultural e idade (27).

A secção do Lançamento da aplicação descreve o processo de conclusão e lançamento da aplicação. Daqui para a frente pode ser necessário consultar ajuda externa de pessoas especialistas em *marketing* (27).

### 2.4.2 A escolha de um *backend*

Para dar suporte à aplicação móvel é necessário a escolha de um *backend* robusto e seguro para que possamos lidar com dados e informações sensíveis e pessoais relacionados com o utente. Por isto deve ter-se em atenção o envio, registo e armazenamento dos dados e informações no servidor. Para isso temos que ter em atenção algumas questões relacionadas com a segurança e privacidade dos dados. A escolha de um *backend* para dar suporte às aplicações *mHealth* é bastante importante. Ao utilizar uma plataforma já desenvolvida e reconhecida estamos a aumentar as funcionalidades da aplicação pois estamos a permitir uma futura interoperabilidade e troca de dados entre outras aplicações que utilizem a mesma plataforma.

***Open mHealth***

A *Open mHealth (OMH)* foi fundada em 2011 e descreve-se a si própria como "uma *start-up* sem fins lucrativos que quebra as barreiras de integração trazendo significado aos dados clínicos digitais na área da saúde"(28). A OMH trabalha com especialistas da área da saúde e com programadores com o objetivo de tornar os dados de saúde digitais úteis e possíveis de utilizar em diversas plataformas.

Um dos principais objetivos desta organização é que as aplicações na área da saúde não utilizem cada uma delas um formato de dados fechado e próprio, pois desta maneira os dados só são úteis dentro da própria aplicação, ou seja, estes não podem ser exportados para bases de dados de hospitais, ou clínicas, ou ainda outras aplicações na área da saúde. Podemos ver na figura 2.5 a diferença que existe entre cada aplicação utilizar um formato de dados próprio, ou utilizar todas o mesmo formato(29). Do lado esquerdo podemos verificar que existem desenvolvimentos diferentes para cada aplicação, enquanto que utilizando o OMH todas podem utilizar a mesma maneira de guardar os dados, ou até, utilizar o mesmo servidor para guardar os dados.

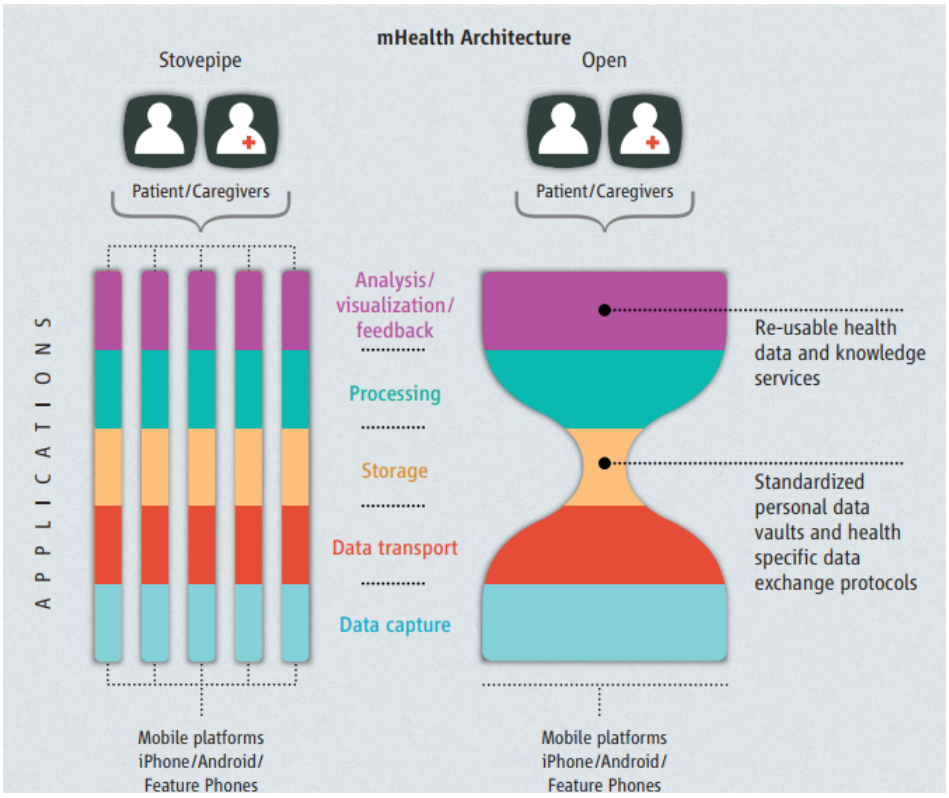


Figura 2.5: Arquitetura *mHealth*: Própria vs *Open* (29). A utilização de uma plataforma aberta reduz o esforço e o desenvolvimento nas tarefas de gestão dos dados

Através desta arquitetura podem ser partilhados diversos recursos entre as aplicações

móveis na área da saúde, entre eles o mais importante é o modo e o local onde os dados são guardados, respeitando um padrão para o formato dos dados, ou seja, se duas aplicações diferentes forem guardar dados de um determinado tipo, como por exemplo frequência cardíaca, esses dados têm que respeitar um determinado formato, sendo estes guardados da mesma maneira independentemente da aplicação que o esteja a fazer.

A OMH tem definido um conjunto de *data schemas* que é um conjunto de esquemas de dados (*schemas*) criados e disponíveis que especificam um formato de dados para um determinado conteúdo como por exemplo a frequência cardíaca(30). Estes *data schemas* estão desenvolvidos em JSON *schema* que serve para descrever um formato de dados, e os programadores têm que ter o cuidado de os dados inseridos e criados sejam compatíveis com o JSON *schema* associado. Na figura 2.7 podemos ver um exemplo de dados que é compatível com o JSON *schema* da figura 2.6 associado que é a frequência cardíaca(31).

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "description": "This schema represents a person's heart rate, either a single heart rate measurement,
                    or the result of aggregating several measurements made over time (see Numeric
                    descriptor schema for a list of aggregate measures)",
    "type": "object",
    "references": [
        {
            "description": "The SNOMED code represents Pulse rate (observable entity)",
            "url": "http://purl.bioontology.org/ontology/SNOMEDCT/78564009"
        }
    ],
    "definitions": {
        "unit_value": { "$ref": "unit-value-1.x.json" },
        "time_frame": { "$ref": "time-frame-1.x.json" },
        "descriptive_statistic": {
            "$ref": "descriptive-statistic-1.x.json"
        },
        "temporal_relationship_to_physical_activity": {
            "$ref": "temporal-relationship-to-physical-activity-1.x.json"
        }
    },
    "properties": {
        "heart_rate": {
            "allOf": [
                {
                    "$ref": "#/definitions/unit_value"
                },
                {
                    "properties": {
                        "unit": {
                            "enum": [
                                "beats/min"
                            ]
                        }
                    }
                }
            ]
        },
        "effective_time_frame": { "$ref": "#/definitions/time_frame"},
        "descriptive_statistic": {
            "$ref": "#/definitions/descriptive_statistic"
        },
        "temporal_relationship_to_physical_activity": {
            "$ref": "#/definitions/temporal_relationship_to_physical_activity"
        },
        "user_notes": {
            "type": "string"
        }
    },
    "required": ["heart_rate"]
}
```

Figura 2.6: JSON *schema* que define o tipo de dados para a frequência cardíaca (31)

```
{
    "heart_rate": {
        "value": 50,
        "unit": "beats/min"
    },
    "effective_time_frame": {
        "date_time": "2013-02-05T07:25:00Z"
    },
    "user_notes": "I felt quite dizzy"
}
```

Figura 2.7: Exemplo de um JSON compatível associado à frequência cardíaca (31)

Existe uma implementação de uma *RESTful API* denominada por *dataPoint API* que suporta a criação, consulta e eliminação de dados inseridos. Esta API permite a autorização utilizando o protocolo de autorização OAuth 2.0. Um *dataPoint* é um documento JSON composto por um cabeçalho e um *body*, em que o *body* representa um tipo de dados e está em conformidade com o *dataschema* definido no cabeçalho.

### FHIR

Como vimos anteriormente para que os sistemas de serviços hospitalares comuniquem e partilhem informação entre si, e com as plataformas online, é imperativo que compreendam o que está a ser comunicado.

Para compreenderem o que está a ser comunicado, os sistemas hospitalares e plataformas devem acordar na norma de comunicação. Existem várias normas para a troca de informação clínica. Existe uma norma que é o *Health Level Seven (HL7)* que é tipicamente utilizada em sistemas hospitalares (32) e tem ganho popularidade na troca de informação clínica estruturada. A organização que definiu o HL7 desenvolveu a sua própria API e formatos de dados denominada de *Fast Healthcare Interoperability Resources (FHIR)* (33). O FHIR consiste numa definição de uma API e conjunto de dados normalizados com o objetivo de fornecer mecanismos de interoperabilidade para o HL7, baseados nas tecnologias existentes na *web*, tais como XML, JSON , HTTP, OAuth, entre outros (33). Este suporta arquiteturas baseadas em REST e é suficientemente flexível para ser utilizado em diversos contextos, tais como aplicações móveis ou partilha de registos clínicos eletrónicos (33).

#### HL7

A norma HL7 é utilizada para troca, integração, partilha e requisição de registos clínicos em formato eletrónico(34). O HL7 encontra-se atualmente na sua versão 3(35), mas a versão 2 do HL7 ainda é bastante utilizada, especialmente em sistemas antigos.

O problema que existia na versão 2 do HL7 é que cada sistema hospitalar ou clínica podia adaptar a norma à sua medida. Como esta versão 2 não possui um modelo explícito de informação(mas definições vagas para muitos campos de dados) e também campos opcionais.

Estas características conferem-lhe uma maior flexibilidade, mas tornam necessários acordos bilaterais detalhados de forma a permitir a interoperabilidade entre os sistemas envolvidos. Quando estes acordos não são efetuados, se formos desenvolver uma aplicação que comunica com vários sistemas hospitalares ou clínicos, é necessário implementar uma interface específica para cada um deles.

Para resolver este problema saiu a versão 3, mas a sua adoção não é ainda generalizada (35). A versão 3 do HL7 foi desenvolvida com base num modelo de dados orientado a objetos, denominado por *Reference Information Model (RIM)*. Este é constituído por 4 classes principais, tendo como objetivo garantir a interoperabilidade semântica que não existia nas versões anteriores. Estas quatro classes principais são: Entidade, Papel/Cargo, Participação e Ato(36).

O modelo RIM é construído à volta de 5 conceitos principais:

- Todo o acontecimento é um ato
- Atos estão relacionados com um participante
- A participação define o contexto de um ato
- Os participantes têm um papel/cargo
- Cada cargo/papel é desempenhado por uma entidade

Na figura 2.8 podemos visualizar a class RIM da versão 3 do HL7.

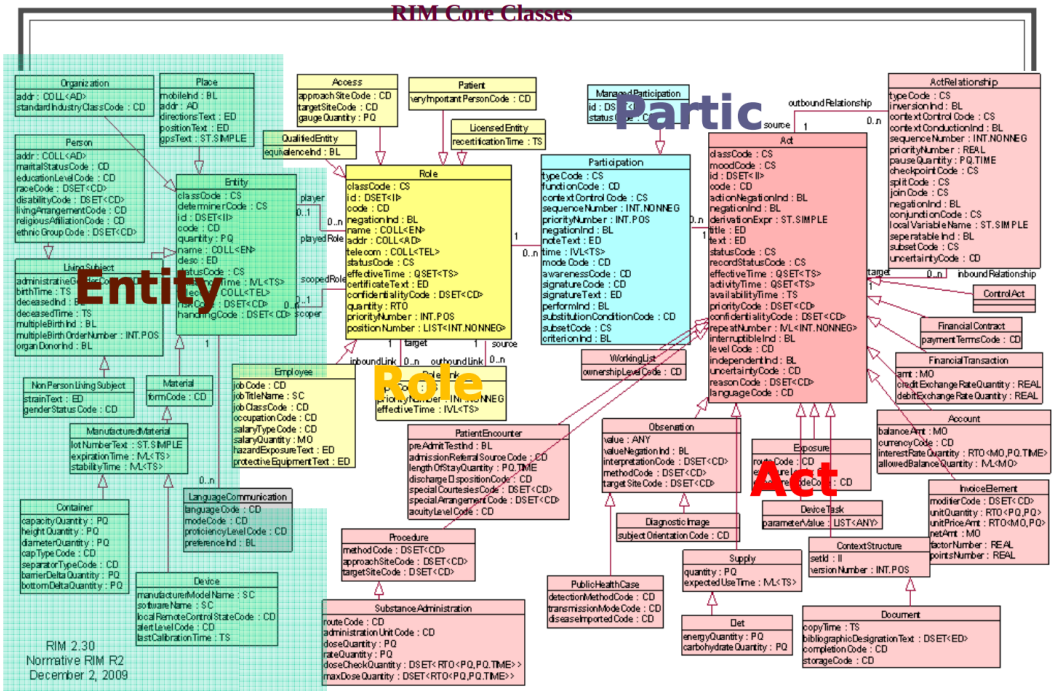


Figura 2.8: Classes principais do RIM (36)

### *Google Fit*

O *Google Fit* é uma API REST desenvolvida pela Google, na área do *fitness*. Permite aos utilizadores armazenar e aceder a informação relativa à sua condição física. Apesar de não ter sido desenvolvido com o objetivo de controlar os dados de saúde dos pacientes, pode ser visto como uma plataforma que cumpre alguns desses objetivos.

O *Google Fit* é mais utilizado com o objetivo de monitorizar a condição física dos seus utilizadores, mas a *Google* permite o acesso aos dados recolhidos, através de API's criadas para esse efeito. Isto permite a criação de aplicações de saúde, visto que também é um dos objetivos da *Google* a integração de qualquer aparelho sensor. Resumindo, são dadas todas as ferramentas necessárias ao programador para aumentar o alcance da plataforma, permitindo-a ser utilizada com finalidades que a *Google* neste momento não cobre. Na figura 2.9 mostramos uma vista geral da plataforma *Google Fit*.

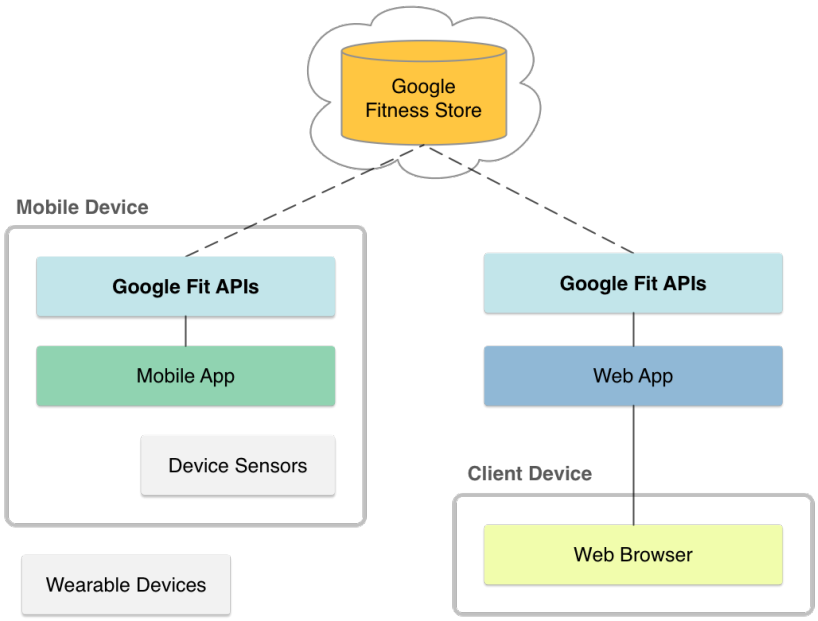


Figura 2.9: Vista geral da plataforma *Google Fit* (37)

Como podemos perceber, existem duas possibilidades diferentes de interagir com a plataforma. Através de uma aplicação para *smartphones* com o sistema operativo *Android*, desenvolvida também pela *Google*. Também é possível a utilização de um *website*, que será um pouco mais limitado, mas se for apenas para visualizar os dados já inseridos na plataforma é bastante viável.

A API REST dá flexibilidade ao sistema. Com a utilização da API passa a ser possível a criação de aplicações para outras plataformas que não o *Android*, tornando o *Google Fit* apelativo para um maior número de utilizadores. Esta API está protegida pelo protocolo de autorização OAuth 2.0 e utiliza para formato dos dados JSON(38).

**Análise Comparativa**

Foi ainda feito um breve estudo em relação a outros três possíveis *backends*, entre eles temos o *Health Vault* [2], *Apple Research Kit* [3] e o *Research Stack* [4]. O estudo não foi tão aprofundado como os apresentados anteriormente mas ainda assim serve para análise comparativa na tabela 2.2 [5]. Os requisitos analisados relativamente a cada possível *backend* foram:

- Possibilidade de guardar dados demográficos dos utentes.
- Análise do grau de complexidade do *backend* para adaptação e implementação de novas funcionalidades.
- Estruturas de dados normalizadas para uma possível importação/exportação dados de/para outros *backends*.
- Possibilidade de adicionar novas estruturas de dados .
- Questões relativas a Segurança e Privacidade sobre os dados recolhidos e demográficos.
- Capacidade de ser integrado em ambientes diferentes, como *android*, iOS ou *web*.
- Código Aberto quanto à própria implementação do *backend*.

| **Plataforma** / **Requisito** | **OMH** | **FHIR** | ***Google Fit*** | ***Health Vault*** | ***Apple Research Kit*** | ***Research Stack*** |
|---|---|---|---|---|---|---|
| **Dados do Paciente** | - | + + | - | + + | + + | + + |
| **Facilidade de Desenvolvimento** | + + | + | + + | - | n/d | n/d |
| **Modelo de Dados Normalizado** | + | + + | - - | - - | n/d | n/d |
| **Extensibilidade do Modelo de Dados** | + + | + | + | + | n/d | n/d |
| **Segurança e Privacidade** | + + | + + | + + | + + | + + | + + |
| **MultiPlataforma** | + + | + + | + + | + + | n/d | n/d |
| **Código Aberto** | + + | + | - | - | n/d | n/d |

Tabela 2.2: Comparação dos diferentes *backends*

[2] https://international.healthvault.com
[3] https://www.apple.com/pt/researchkit/
[4] http://researchstack.org/
[5] - - nenhum suporte; - suporte fraco; + suporte suficiente; ++ muito bem suportado; n/a não definido

Depois de analisados seis serviços de *backend* para dar suporte a uma aplicação móvel na área de saúde, faz-se aqui uma breve análise comparativa das suas características. Cada um apresenta vantagens nuns aspetos e desvantagens noutros.

A grau de complexidade do FHIR é bastante elevado devido à sua grande dimensão. É um *backend* bastante completo apesar da extensibilidade dos tipos de dados não ser trivial. Relativamente aos tipos de dados a ser guardados falta contemplar o ECG e o acelerómetro [6], pois estes não estão contemplados.

Em relação ao *Google Fit* tem uma boa API REST, apesar de não haver nenhum tipo dados onde se pudesse guardar os dados do paciente e o ECG. A extensibilidade do modelo de dados é possível mas existe uma limitação relativamente ao tipo de dados que podem ser utilizados, entre eles temos apenas o *int* e o *float*, ou seja, não existe o tipo objeto e vetor.

O OMH tem uma API REST, assim como o *Google Fit*, e não tem a possibilidade de guardar dados do Paciente, mas o modelo de dados é extensível. A grande vantagem que tem é um conjunto de JSON *schemas* definidos prontos a usar para validar a entrada dos dados no *backend*. Deste modo nenhum tipo de dados vai ser inserido se não respeitar as devidas definições. Ao criarmos novos JSON *schemas* podemos reutilizar os já existentes para definir determinados atributos, o que torna tudo bastante mais fácil.

[6]Estes dados fazem parte do conjunto de dados a ser guardado no sistema mais à frente, neste momento serve também para avaliar a capacidade da cada plataforma para os suportar/integrar

# Capítulo 3

# Requisitos

Neste trabalho, procuramos identificar um *backend* conveniente para o desenvolvimento de aplicações *mHealth*. Para isso, vamos definir um sistema-alvo, que requer uma arquitetura tipica de uma solução de *mHealth*. O objetivo é partir de um cenário concreto para extrair requisitos que possam validar a arquitetura escolhida.

## 3.1 Visão geral do sistema a desenvolver

O produto proposto é um sistema que permite a recolha de dados fisiológicos de pessoas e a respetiva revisão dos dados recolhidos por parte dos profissionais, responsáveis pelo estudo. A recolha pode ocorrer em laboratório ou em ambulatório, isto é, os participantes podem recolher estes dados presencialmente ou remotamente. Um dos objetivos deste sistema é apoiar projetos de I&D que incluem componentes de análise da fisiologia humana.

O sistema será composto por uma aplicação móvel que tem como objetivo principal recolher dados dos sensores, e uma aplicação *web* para visualizar esses dados recolhidos. O sistema irá contemplar então dois atores distintos:

- Revisor/Investigador que tem como objetivo rever dados inseridos pelos diferentes participantes num estudo.
- Participante(alvo de estudo) que tem como função recolher dados vitais e acelerómetro para posteriormente serem revistos pelo revisor/investigador.

O dispositivo com sensores utilizado poderá ser o *VitalJacket* (39) e os dados fisiológicos recolhidos poderão ser a frequência cardíaca, ECG e acelerómetro. Poderá ser utilizado este dispositivo pois como já fez parte de outros estudos, é um equipamento válido e permite recolher os tipos de dados que pretendemos.

A acelerómetro está integrado no *VitalJacket* e a sua integração deve-se à facilidade com que este sensor aparece hoje dia em *smartphones*, *smartwatchs* e *bracelets*. É uma variável adicional, típica das aplicações móveis, mas ausente dos sistemas de dados clínicos clássicos.

## 3.2 Casos de Utilização na Colheita de Dados

Na figura 3.1 podemos visualizar um diagrama com os casos de utilização (cenários-objetivo) da aplicação móvel. O principal ator desta aplicação é o participante que tem que recolher várias sessões de leitura para serem posteriormente analisadas e visualizadas por Investigadores/Revisores. Uma breve descrição de cada caso de utilização é apresentada na tabela 3.1.

O ator é um participante de um estudo e pode ter sido selecionado pelos responsáveis de duas maneiras diferentes, eventualmente como voluntário ou como doente.

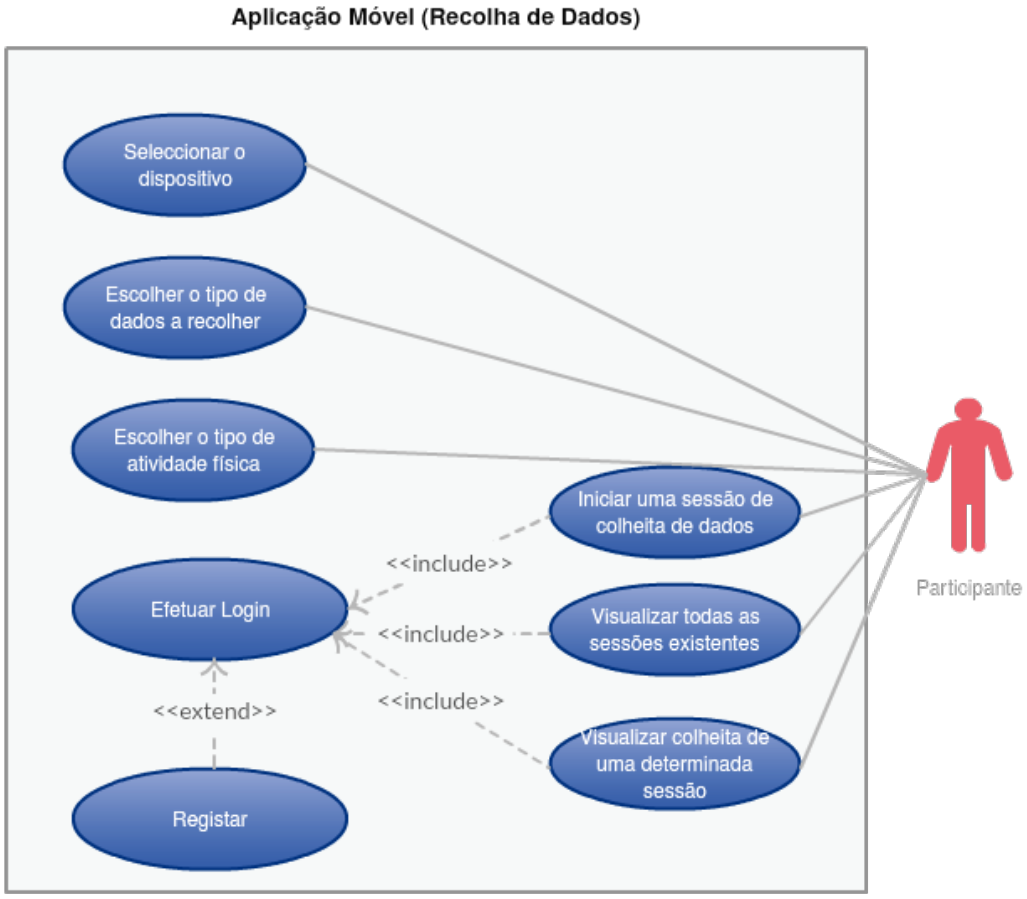


Figura 3.1: Diagrama de casos de uso da aplicação móvel para recolha de dados

| Caso de utilização | Propósito do caso de utilização |
|---|---|
| Efetuar *Login* | O Participante deve conseguir entrar com a sua conta na aplicação móvel. |
| Registar | O Participante que vai participar no estudo pode criar uma nova conta. |
| Escolher a relação temporal relativamente à atividade física | Consoante o tipo de atividade física que estiver a efetuar o participante, pode especificar a sua relação temporal relativamente à atividade física. Tendo como exemplo a corrida, a relação temporal poderá ser entre as disponíveis: antes, depois, durante, em descanso |
| Escolher o tipo de dados a recolher | Tem que existir a possibilidade de filtrar o tipo de dados que são para ser recolhidos numa determinada colheita. Pode ocorrer a situação de recolher todos ao mesmo tempo. |
| Selecionar o dispositivo | Para se conseguir efetuar a colheita dos dados um dispositivo válido tem que ser selecionado. |
| Iniciar uma sessão de colheita de dados | Pode iniciar uma nova colheita de dados. Esta nova colheita pode ser durante vários segundos, minutos e até horas. Apenas os dados escolhidos poderão ser guardados para posterior consulta. A colheita pode ser interrompida quando o participante quiser. |
| Visualizar todas as sessões existentes | O Participante pode visualizar todas as sessões de recolha dados. |
| Visualizar colheita de uma determinada sessão | O Participante ao escolher uma colheita das várias apresentadas e pode visualizar ao detalhe todos os dados recolhidos numa determinada sessão. |

Tabela 3.1: Breve descrição dos casos de utilização da aplicação de colheita de dados

## 3.3 Casos de Utilização na Revisão de Dados

Na figura 3.2 podemos visualizar todos os casos de uso (cenários-objetivo) da aplicação *web*. O principal ator desta aplicação é o revisor/investigador que pode fazer uma revisão das várias sessões de recolha efetuadas pelos participantes. Uma breve descrição de cada caso de utilização é apresentada na tabela 3.2

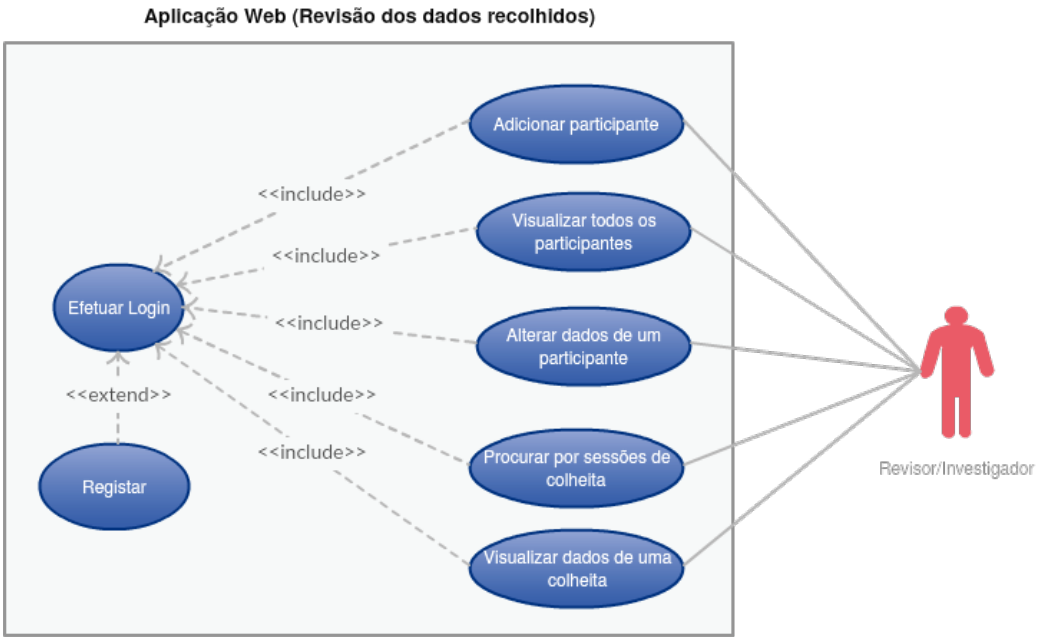


Figura 3.2: Diagrama de casos de uso da aplicação *web*

| **Caso de utilização** | **Propósito do caso de utilização** |
|---|---|
| Efetuar *Login* | O Revisor deve conseguir entrar com a sua conta na aplicação *web*. |
| Registar | O Revisor pode criar uma nova conta. |
| Adicionar participante | O Revisor deve poder adicionar um novo participante como seu alvo de estudo. |
| Visualizar todos os participantes | O Revisor deve conseguir visualizar todos os participantes do seu alvo de estudo. |
| Alterar dados de um participante | A edição dos dados demográficos do participante deve ser possível. |
| Iniciar uma sessão de colheita de dados | Pode iniciar uma nova colheita de dados. |
| Procurar por sessões de colheita | Relativamente a um participante deve ser possível pesquisar colheitas efetuadas num determinado intervalo de tempo. |
| Visualizar dados de uma determinada sessão de colheita | O Revisor depois de pesquisar as sessões para um determinado intervalo, deve poder ver os dados recolhidos para cada uma das sessões. |

Tabela 3.2: Breve descrição dos casos de utilização da aplicação de Revisão

# Capítulo 4

# Avaliação exploratória de soluções para o *backend*

## 4.1 Cenário-tipo selecionado

Antes de avançar com o desenvolvimento do sistema foi necessário escolher um *backend* para dar suporte tanto à aplicação móvel, como à aplicação *web*. Para isso foi desenvolvido um cenário idêntico com os diferentes *backends* em que pudéssemos fazer uma análise comparativa dos pontos fortes e ou fracos entre eles, usando um âmbito mais simples, mas relacionado com o sistema a desenvolver.

O cenário escolhido para esta avaliação exploratória foi o simples ato de recolher a frequência cardíaca do sensor do *VitalJacket* e guardar no *backend* em estudo, para posteriormente ser possível a visualização de um histórico dos dados inseridos. Na figura 4.1 temos uma simples visão das várias partes envolvidas nesta avaliação exploratória.

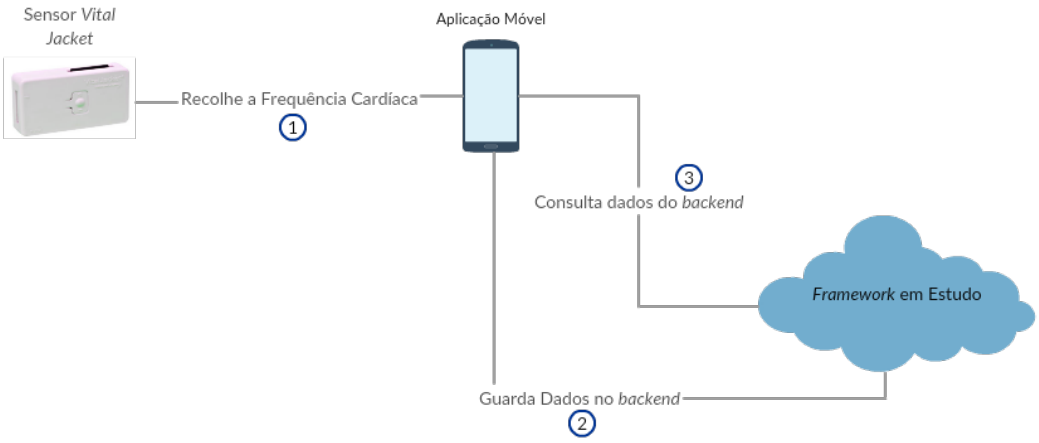


Figura 4.1: Diagrama geral do cenário de estudo

Das várias plataformas disponíveis para um possível *backend* foram selecionadas três para ser desenvolvido o cenário de avaliação com o objetivo de concluir se seria uma boa hipótese para a solução: OMH, FHIR e *Google Fit*.

Para utilizar cada uma destas plataformas, foi desenvolvida uma aplicação móvel, bastante simples. A aplicação móvel tinha várias funcionalidades-chave, entre elas:

- Selecionar o dispositivo com os sensores
- Efetuar *Login* na respetiva plataforma
- Guardar leituras de frequência cardíaca
- Visualizar as leituras inseridas

Tendo em conta que a mesma aplicação conseguia utilizar os três *backends* distintos, esta seleção era feita tendo em conta o tipo de *login* efetuado. De seguida apresentamos as experiências exploratórias com cada plataforma. A frequência cardíaca foi o tipo de dado fisiológico escolhido, pois era aquele tipo de dado que estava disponível por omissão em todos os *backends* escolhidos para esta fase exploratória.

## 4.2 Implementação exploratória: *Open mHealth*

Relativamente a este *backend* para solução do problema proposto, tínhamos disponível um serviço denominado de *Data Storage Unit (DSU)* que disponibiliza um servidor de dados que fornece uma API REST denominada de *dataPoint API*.

A API suporta a criação, consulta e eliminação de dados. Permite a autorização utilizando o protocolo de autorização OAuth 2.0. Em suma este serviço é composto por um servidor de dados e um servidor de autorização e autenticação. O servidor de autorização gere a concessão de *tokens* de acesso. (40)

Com a introdução desta plataforma no caso de estudo exploratório ficamos com uma arquitetura que está representada na figura 4.2.

Desta plataforma foi utilizado o servidor de dados (alguns tipos de dados existentes) e o servidor de autenticação e autorização, toda a implementação adicional é referida em 4.2.1.

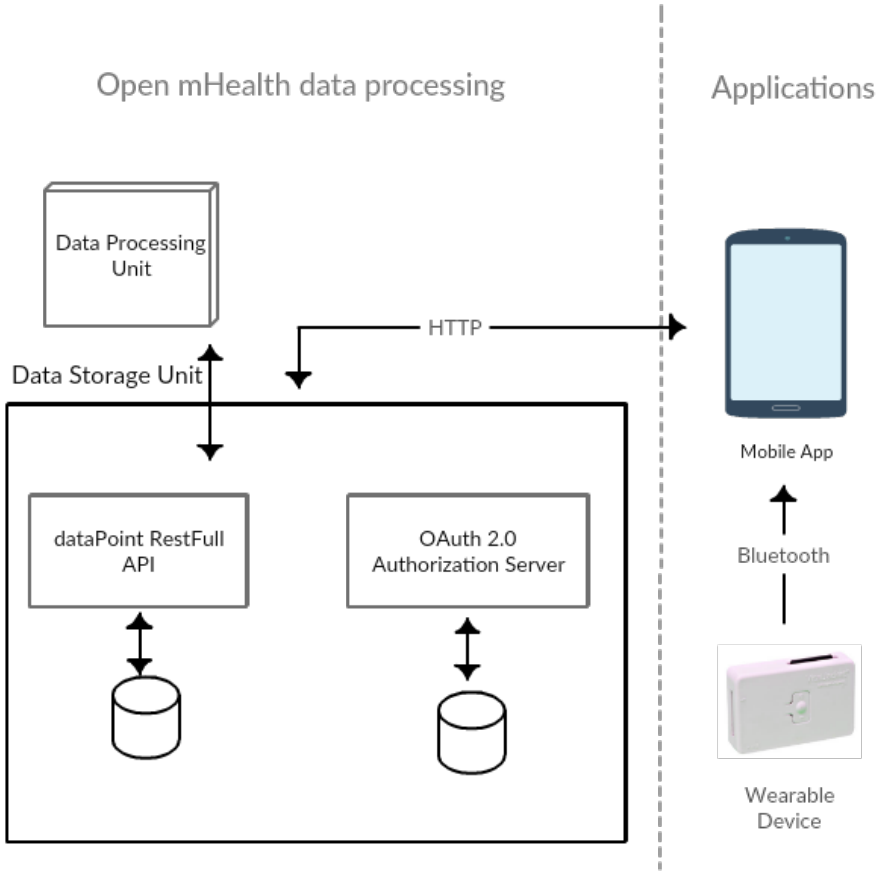


Figura 4.2: Arquitetura do caso exploratório com o *Open mHealth*

### 4.2.1 Implementação adicional

No OMH, um *dataPoint* é um documento JSON composto por um cabeçalho e um *body*, em que o *body* representa um tipo de dados e está em conformidade com o *dataschema* definido no cabeçalho. No processo de criação de um novo *dataPoint* foi adicionada uma validação, para que fosse garantido que o *body* estava corretamente formatado com o tipo de dados definido no cabeçalho. Apesar dos *data schemas* especificarem um formato de dados, esta validação não estava a ser efetuada. Esta validação é importante para posteriores consultas e tratamentos dos dados de maneira igual para cada tipo.

A OMH tem definido um conjunto de *data schemas* (30) que é um conjunto de esquemas de dados criados e disponíveis que especificam um formato de dados para um determinado conteúdo como por exemplo a frequência cardíaca(30). Na tabela 4.2.1 podemos visualizar alguns dos esquemas de dados existentes disponibilizados pelo OMH[1]. Este conjunto de esquemas de dados pode ser estendido. Ao criar um novo esquema de dados(um novo *data schema*) podemos reutilizar outros já existentes, criando um *data schema* que mesmo não sendo normalizado, utiliza para definição de determinadas propriedades outros *data schemas* normalizados. Para experimentar esta funcionalidade foi então adicionado um novo esquema de dados para que o sistema conseguisse suportar ECG. Esta extensão foi conseguida com sucesso, saiu fora do cenário objetivo mas serviu para conseguir perceber melhor a plataforma.

[1] http://www.openmhealth.org/documentation/#/schema-docs/schema-library/

| **Esquema de dados (*dataschema*)** | **Nome do ficheiro e Descrição do esquema de dados** |
|---|---|
| Frequência cardíaca | *heart-rate-1.0.json*<br>Este esquema de dados representa a frequência cardíaca de uma pessoa e a sua relação com a atividade física. O esquema pode ser usado para uma única medição de frequência cardíaca, podendo ser efetuadas várias medições |
| Relação com a atividade física | *temporal-relationship-to-physical-activity-1.0.json*<br>Este esquema de dados representa a relação temporal de uma colheita de dados com a atividade física (por exemplo, em repouso, durante o exercício, antes do exercício). |
| Período de tempo | *time-frame-1.0.json*<br>Este esquema de dados permite que um período de tempo específico seja definido, como uma hora exata ou um intervalo de tempo. |
| Pressão arterial | *blood-pressure-2.0.json*<br>Este esquema de dados representa a pressão arterial de uma pessoa como uma combinação da pressão arterial sistólica com a pressão arterial diastólica. Permite indicar se o paciente estava deitado, sentado ou em pé quando a medição foi obtida. Este esquema pode ser usado para uma medição única da pressão arterial, podendo ser efetuadas várias medições. |
| Temperatura do corpo | *body-temperature-2.0.json*<br>Este esquema de dados representa a temperatura corporal e o local do corpo onde foi efetuada a medição. Este esquema pode ser usado para uma medição única, podendo ser efetuadas várias medições. |
| Glicose no sangue | *blood-glucose-2.0.json*<br>Este esquema de dados representa o nível de glicose no sangue de uma pessoa, e o tipo de amostra do corpo utilizada para efetuar a medição. É ainda apresentada a relação temporal relativamente à refeição e ao sono. Este esquema pode ser usado para uma medição única, podendo ser efetuadas várias medições. |

Tabela 4.1: Tabela com alguns dos esquemas de dados existentes na plataforma *Open mHealth*

### 4.2.2 Dificuldades e Adaptações

As dificuldades encontradas não foram muitas, pois a plataforma estava bastante bem estruturada e o grau de complexidade era aceitável. As possíveis falhas desta plataforma prendiam-se com a falta de esquemas de dados para o acelerómetro, ECG e para dados demográficos dos utilizadores da plataforma.

## 4.3 Implementação exploratória: FHIR

O FHIR ao contrário do *Open mHealth* é apenas uma definição de uma API para armazenamento de informação e troca de dados clínicos entre hospitais e clínicas. Esta API pode ser desenvolvida de diferentes maneiras.

Existe uma implementação de um projeto de código aberto com o nome *Hapi-FHIR* (41). Este projeto suporta todos os dados definidos pelo FHIR e grande parte das operações sobre eles, como por exemplo criar, editar, eliminar, etc.

O projeto *Hapi-FHIR* tem um módulo denominado por *JPAServer* (42). Este módulo pode ser utilizado para criar um servidor FHIR, composto por uma API REST disponibilizando um conjunto de *endpoints* HTTP para que se possa efetuar os pedidos necessários para se guardar os dados no *backend*.

### 4.3.1 Implementação adicional

O módulo utilizado tinha uma limitação relativa à concessão de autorização sobre os pedidos efetuados, não suportando também a gestão de identidades. Para complementar este módulo foi utilizado o servidor de autorização utilizado também na experiência anterior(o servidor de autorização do DSU do *Open mHealth*). No *JPAServer* foi então desenvolvido um Intercetor, cuja função era verificar se o pedido efetuado era acompanhado por um *token* de acesso; caso isso acontecesse a validade do *token* era verificada pelo servidor de autorização e em caso de sucesso era fornecido o acesso ao pedido solicitado. Na figura 4.3 temos então a arquitetura final deste caso exploratório.

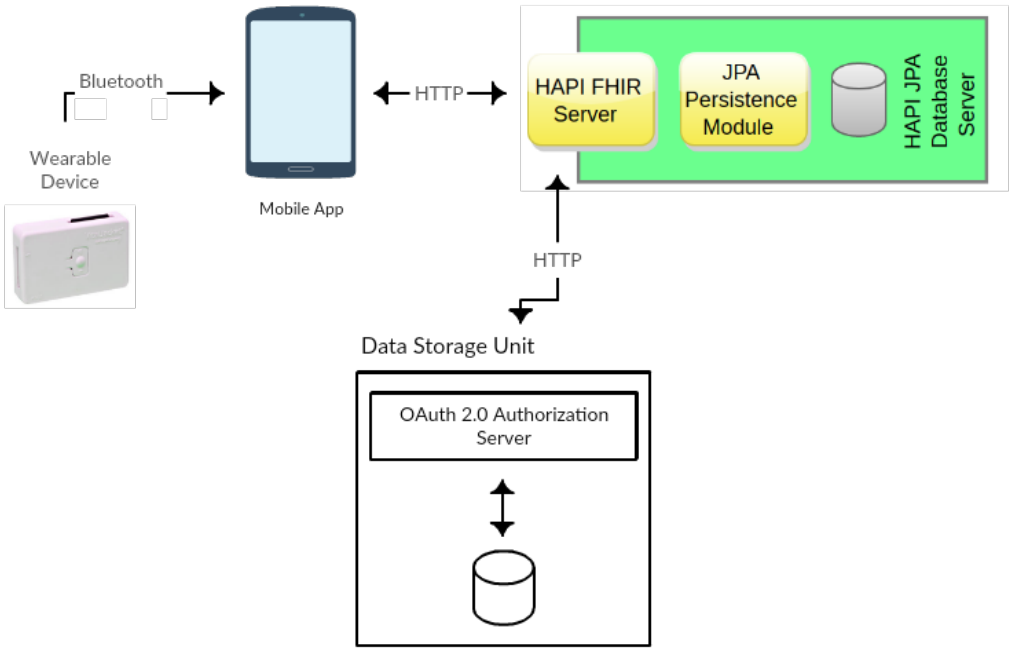


Figura 4.3: Arquitetura do caso exploratório com o FHIR (Adaptado de (41))

### 4.3.2 Dificuldades e Adaptações

As dificuldades encontradas ainda foram algumas, neste caso temos uma definição de uma API em vez de uma plataforma em si, ou seja, foram encontradas várias supostas resoluções, e no caso do *Hapi-FHIR* estava dividida por vários módulos, que por um lado até se pode tornar mais vantajoso porque acabamos por utilizar só aquilo que é necessário. As possíveis falhas desta API estavam relacionadas com a falta de tipos de recursos para o acelerómetro e ECG. Ainda se pode apontar a falta de um servidor de gestão de identidades e a concessão de autorização ao pedidos efetuados.

## 4.4 Implementação exploratória: *Google Fit*

O *Google Fit* disponibiliza uma API REST que permite o armazenamento de dados na nuvem. Esta API permite a criação de *datasources*. Um *datasource* é criado por cada utilizador e representa um conjunto de dados de um determinado tipo. Relativamente à concessão de acesso aos serviços REST é utilizado o serviço de autenticação e autorização da Google baseado em OAuth 2.0.

O cenário-objetivo foi desenvolvido com sucesso e na figura 4.4 podemos ver a arquitetura final deste caso exploratório.

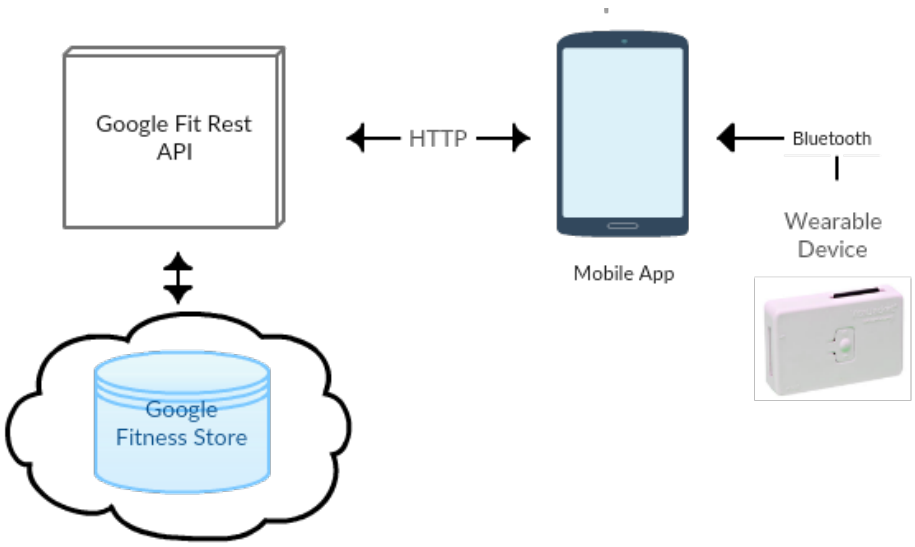


Figura 4.4: Arquitetura do caso exploratório com o *Google Fit*

### 4.4.1 Dificuldades e Adaptações

Um dos problemas encontrados nesta fase de exploração é que apesar do tipo de dados poder ser extensível, apenas permitia para definição das suas propriedades tipo inteiro e *float*, ou seja, não são suportados propriedades do tipo objeto ou vetor. Esta API não tinha suporte para dados do tipo ECG, acelerómetro e dados demográficos.

## 4.5 Resultados e lições aprendidas

***Google Fit***

Esta plataforma tem como um ponto muito positivo utilizar o serviço de autenticação e autorização da *Google* e o facto dos dados ficarem armazenados na nuvem.

O primeiro ponto negativo a apontar é relativo à extensibilidade dos dados, só permitir para definição das suas propriedades, tipo inteiro e *float*. Para exemplificar como isto pode vir a ser um problema, no caso do ECG, por segundo o sensor envia 500 valores, para que se possa formar um ECG com alguma precisão, como esta plataforma não suporta vetores, teriam que ser feitos 500 pedidos ao servidor por segundo para conseguir guardar e posteriormente visualizar um ECG fidedigno. Um outro ponto negativo é o fato de esta plataforma ser proprietária, o que isto quer dizer é que não é possível correr a plataforma de forma nativa e efetuar implementações extra sobre as já existentes.

***Open mHealth***

Como o OMH é um projeto de código aberto um grande ponto forte é a possibilidade da implementação de novas funcionalidades poder ser efetuada com mais facilidade, como

por exemplo a adição da validação dos dados inseridos. Tendo em conta que o projeto foi desenvolvido com o intuito de diminuir a ocorrência de formatos de dados próprios por cada aplicação móvel no âmbito da saúde, esta pode muito bem ser uma solução para o *backend* do produto proposto, dado que a possibilidade da extensibilidade dos dados também é possível através da criação de novos *data schemas* que são posteriormente utilizados para ser efetuada a validação dos dados inseridos.

O conjunto de esquemas de dados criados pela OMH para dar suporte à plataforma foi criado utilizando referências de dados normalizados no ambiente hospitalar e clínico, uma dessas referências utilizada foi a API do FHIR. Aqueles que foram adicionados posteriormente por nós foram também criados utilizando como referência a API do FHIR.

**FHIR**

Um dos pontos fracos do FHIR neste caso em concreto do *Hapi-FHIR* é a falta de serviço de autenticação e autorização. Um outro ponto fraco é a inexistência de um projeto de código aberto "oficial", ou seja, existe bastantes possibilidades diferentes e descobrir qual poderia utilizar e da melhor maneira não foi imediato.

Um dos pontos fortes é saber que esta definição da API é bastante utilizada por hospitais e clínicas.

**Análise Comparativa**

Tendo em conta que o produto proposto tem como um dos objetivos guardar dados relativos ao ECG, o *Google Fit* é limitado relativamente ao tipo de dados, ou seja, não suporta vetores para guardar os dados, ficando logo praticamente excluído como possível escolha para o *backend*. Ficamos agora então com o *Open mHealth* e o FHIR, nenhum deles está apto na totalidade para ser utilizado sem qualquer alteração. Por um lado temos falhas nas duas plataformas que não contemplam todos os tipos de dados que precisam de ser guardados, como por exemplo o acelerómetro e o ECG, apesar do FHIR estar mais completo neste assunto. Para além disto o FHIR não contém nenhum serviço de autenticação e autorização sendo necessário utilizar um serviço externo. Não podemos dizer que o FHIR não servia como solução, mas tendo em conta que nenhum deles é perfeito, e que o *Open mHealth* é um projeto que foi desenvolvido com o objetivo de uniformizar os *backends* na área das aplicações móveis na área de saúde, vamos optar pela utilização desta plataforma, dando uma oportunidade a um projeto de código aberto, com o intuito também de perceber se mais tarde, pode ou não vir a ser utilizado em diferentes projetos como *backend* de aplicações móveis na área da saúde, ou até em projetos de I&D.

# Capítulo 5

# Arquitetura proposta

## 5.1 Visão geral do sistema

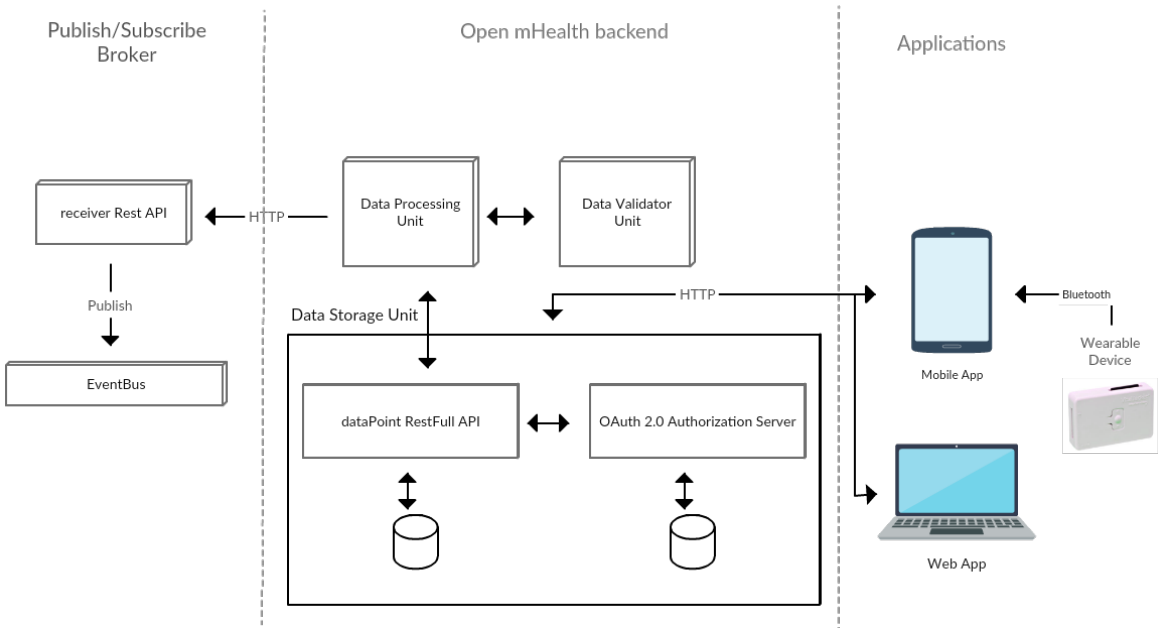


Figura 5.1: Arquitetura proposta para o Sistema

A arquitetura proposta para o sistema está dividida em três partes. Uma delas é a parte das aplicações desenvolvidas, ou seja, a interface com o utilizador, tanto *web* como móvel. Estas duas aplicações comunicarão por pedidos HTTP com o DSU da plataforma do *Open mHealth*. A aplicação móvel ainda terá a responsabilidade de recolher os dados fisiológicos dos participantes como por exemplo o ECG, a frequência cardíaca e o acelerómetro. O sensor

utilizado para recolher os dados é o *VitalJacket* [1] utilizando o perfil de *bluetooth* SPP.

Uma outra parte do sistema já referida é o *backend* do *Open mHealth*. Este *backend* tem um componente principal (DSU) que disponibilizará duas REST API para inserção e fornecimento dos dados e ainda para autenticação e autorização. Ainda terá um componente para processar os dados recebidos. Estes dados passarão por um validador e será adicionada a possibilidade para ser reencaminhados para uma plataforma para posterior publicação das mensagens num *EventBus*, permitindo a extensão com novos módulos interessados em analisar os dados.

A terceira e última parte será composta uma plataforma que irá disponibilizar um *endpoint* para receber os dados e então publicar os mesmos no *EventBus*.

## 5.2 Módulos do sistema

### 5.2.1 Fundações do *backend*

Para o *backend* do sistema será utilizado o *Open mHealth* adaptando-o no necessário e complementado-o. O *backend* deverá ser capaz de fornecer um serviço de autenticação e autorização seguindo o protocolo OAuth 2.0.

Deverá disponibilizar uma REST API para ser efetuado a inserção de dados fisiológicos dos participantes. Esta inserção será apoiada por um validador que irá validar esses dados, para perceber se podem ser inseridos com sucesso.

### 5.2.2 Extensão para suportar *pipelines* de processamento

O *backend* do OMH não suporta uma *pipeline* de processamento dos dados, para este suporte ser possível incluímos uma forma de os dados serem enviados para módulos interessados no seu processamento. A solução proposta passa pela publicação destes dados num *bus* partilhado, permitindo a extensibilidade do sistema com facilidade e um tratamento específico por cada um desses módulos.

### 5.2.3 Integração das Aplicações

As aplicações que forem desenvolvidas utilizando esta arquitetura proposta poderão utilizar, o servidor de autenticação e autorização fornecido pelo *backend* e também uma API para efetuar a inserção e a consulta dos dados inseridos. Esta API terá ainda a capacidade de validar os dados recebidos, não comprometendo posteriormente a consulta dos dados, pois estes só irão ser guardados se respeitarem o esquema de dados respetivo. O conjunto de dados será mais extenso, pois serão adicionados mais esquemas de dados(ECG, dados demográficos

[1] http://www.sdk.vitaljacket.com/?pageid=13

e acelerómetro) e a possibilidade de adicionar novos módulos para processamento de dados recebidos do seu interesse.

## 5.3 Modelo do domínio

Será necessário guardar dados demográficos do participante e os dados fisiológicos como: dados do acelerómetro, frequência cardíaca e ECG. Para isso vamos mostrar um modelo de Entidade-Relação, tendo em conta os dados que precisamos de guardar.

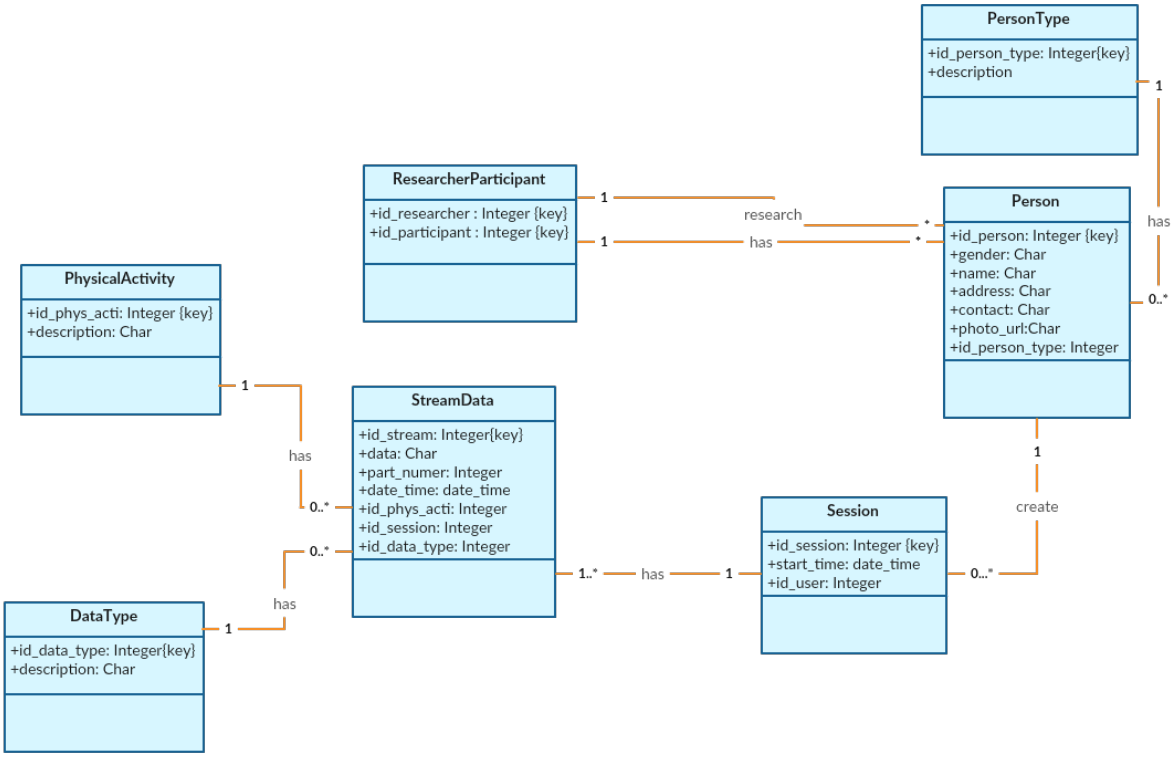


Figura 5.2: Diagrama de classes para a arquitetura proposta

Com este diagrama conseguimos representar pessoas, que podem ser um participante ou um investigador/revisor. Conseguimos associar participantes a revisores, e criar sessões relacionadas com um determinado participante definido a hora de início. Podem ser associadas a estas sessões várias *streams* de dados de um determinado tipo. Cada *stream* de dados tem definida uma relação temporal com a atividade física que esteve a praticar.

Existe ainda a possibilidade de utilizar uma base de dados não relacional, como por exemplo, orientada a documentos. Este tipo de bases de dados não depende de um esquema em específico e como tal suporta dados não estruturados. Este modelo tem uma grande simplicidade permitindo assim aos programadores compreender facilmente o documento e

obter os documentos sob a forma de vetores associativos (43). Uma das grandes vantagens que pode haver ao utilizar um base de dados orientada a documentos, é que a adição de novos esquemas de dados não vai implicar alterações ao esquema da base de dados, mesmo que os dados inseridos não estejam relacionados com os já existentes.

## 5.4 Segurança de dados

Relativamente à segurança dos dados armazenados, esta pode ser visto como uma questão de controlo de acesso, em que é assegurado que os dados estarão acessíveis apenas para pessoas e processos autorizados. Os serviços REST deverão verificar se o pedido efetuado é ou não autorizado, para isso o participante deverá efetuar a autenticação e utilizar o *token* de acesso fornecido. O *backend* deverá utilizar duas bases de dados diferentes, em que numa delas guarda os dados relativos às aplicações cliente e os *tokens* de acesso, e na outra deverá guardar os dados inseridos e os dados relacionados com as contas de utilizador, protegendo assim dados fisiológicos e sensíveis dos participantes, da base de dados onde estão os *tokens*, deste modo conseguimos ter uma segregação de responsabilidade.

A solução devia evoluir para separar a identidade dos participantes do repositório de dados clínicos, guardando por exemplo os dados dos participantes numa base de dados na *cloud*. Tendo em conta que o sistema é uma prova de conceito consideramos algumas simplificações.

# Capítulo 6

# Implementação do sistema

## 6.1 Implementação e adaptação do *backend*

### 6.1.1 Servidor de Autorização e Autenticação

O servidor de autenticação e autorização disponibiliza uma API para criação de novos utilizadores e solicitação de *tokens* de acesso relativamente a várias aplicações. Este serviço utiliza a plataforma *Spring*, mais concretamente a *Spring Security* (44).

No nosso caso vamos apenas ter uma aplicação cliente configurada para posteriormente gerir os *tokens* de acesso para esta aplicação. As credenciais das aplicações cliente vão estar guardadas num base de dados relacional e os utilizadores finais numa base de dados não relacional. Esta opção está relacionada com a segurança de dados referida anteriormente em 5.4. Relativamente à base de dados não relacional para os utilizadores finais são guardados numa coleção do *MongoDB*(45), a coleção é denominada por *endUser*. Este coleção é então composta por vários documentos, em que cada documento corresponde a um utilizador final. Os documentos são formatados como na Figura 6.1.

```
{
        "_id" : "usertest",
        "_class" : "org.openmhealth.dsu.domain.EndUser",
        "password_hash" : "$2a$10$a23I8/KW1KXwHApyd401h./EHFQA2jD6ck8JpgjIz1PiXDbAiD27W",
        "registration_timestamp" : "2017-09-13T00:14:15.703+01:00"
}
```

Figura 6.1: Formato do documento de um utilizador final

Para a base de dados relacional a escolha podia por exemplo ser efetuada entre o *MySQL* (46) e *PostgreSQL* (47), neste caso foi utilizado o *PostgreSQL*, o modelo de dados relacional está representado na Figura 6.2.

Na tabela *oauth_client_details* estão guardadas todas as informações relativas à aplicação como por exemplo a sua identificação, a chave secreta, os tipos de autorização, etc. Rela-

tivamente à tabela *oauth_access_token* estão guardados os *tokens* de acesso por utilizador, relativamente a uma aplicação. Cada *token* de acesso tem relacionado a ele um *refresh token* que está guardado numa tabela diferente, na tabela *oauth_refresh_token*.

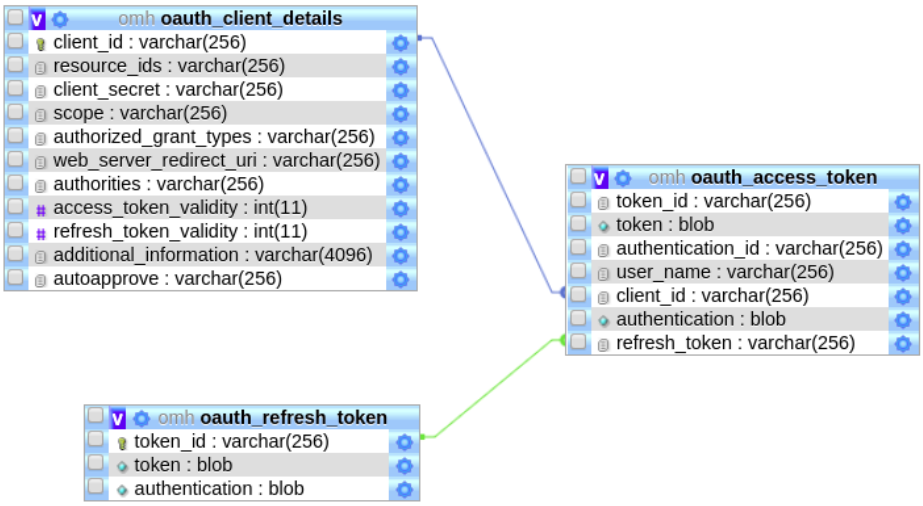


Figura 6.2: Diagrama do modelo de dados para guardar as credenciais dos clientes e os *tokens* de acesso

### API REST

Na tabela 6.1 temos os métodos disponíveis do servidor de autorização e autenticação.

| **domain:port/users** |
|---|
| Serviço POST, permite a criação de novos utilizadores.<br>Recebe um JSON composto por dois campos um *username* e *password*. |
| **domain:port/oauth/token** |
| Serviço POST, permite o pedido de um novo *token* de acesso.<br>Recebe por parâmetros o *username*, *password* e o *grant_type*. |

Tabela 6.1: Tabela com a API REST do servidor de autorização e autenticação

## 6.1.2 Servidor de recursos ( *dataPoint* REST API)

O servidor de recursos disponibiliza uma REST API para criação, consulta e eliminação de *dataPoints*. Estes *dataPoints* estão a ser guardados numa base de dados *MongoDB* numa coleção com o nome *dataPoint*. A maior vantagem de utilizar uma base de dados não relacional é que os esquemas de dados podem ser estendidos de forma mais flexível, não sendo necessário efetuar nenhuma alteração ao esquema da BD.

Como foi referido na fase exploratória do OMH 4.2 estes *dataPoints* são compostos por um cabeçalho e por um corpo. Este corpo é formatado pelo esquema de dados ( *data schema*) associado e definido no cabeçalho do *dataPoint*. Como foi referido também na fase exploratória foi desenvolvido um validador que verifica se o corpo do *dataPoint* está de acordo com o *data schema* definido no cabeçalho do mesmo.

Estavam então em falta alguns esquemas de dados para conseguirmos abranger todos os tipos dados necessários, entre eles: ECG, dados demográficos e acelerómetro. Estes *data schemas* foram criados aproveitando as estruturas da definição da API do FHIR e vou mostrar de seguida como exemplo o esquema de dados criado para definir o formato de dados para as entradas para o ECG.

***data schema* ECG**

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "description": "This schema represents a part of ecg",
  "type": "object",
  "definitions": {
       "time_frame": {
           "$ref": "time-frame-1.x.json"
       },
       "temporal_relationship_to_physical_activity": {
           "$ref": "temporal-relationship-to-physical-activity-1.x.json"
       },
       "values": {
           "type":"array",
           "items":{
             "type": "integer",
             "maximum":250,
             "minimum":0
           },
           "minItems":500,
           "maxItems":500
       },
       "unit":{ "type":"string", "enum":["microVolt", "uV"] },
       "part_number": { "type":"integer", "minimum": 0 },
       "session":{ "type":"integer", "minimum": 0 }
   },
  "properties":{
    "ecg": {
      "type":"object",
      "properties":{
        "values": {
            "$ref": "#/definitions/values"
        },
        "unit": {
            "$ref": "#/definitions/unit"},
        "part_number":{
            "$ref": "#/definitions/part_number"},
        "session":{
            "$ref": "#/definitions/session"}
      },
      "required": ["values", "part_number","unit", "session"]
    },
           "effective_time_frame": {
      "$ref": "#/definitions/time_frame"
    },
    "temporal_relationship_to_physical_activity": {
      "$ref": "#/definitions/temporal_relationship_to_physical_activity"
    }
  },
  "required": ["ecg", "effective_time_frame"]
}
```

Figura 6.3: Novo esquema de dados relativo ao ECG

**API REST**

Na tabela 6.2 temos os métodos disponíveis do servidor de recursos.

| **domain:port/v{apiVersion}/dataPoints** |
|---|
| Serviço POST, para criação de novos *dataPoints*.<br>No *header* recebe o *access_token* para o servidor de autorização lhe dar acesso ao pedido efetuado. Todos os pedidos efetuados a esta API tem que fornecer o *token* de acesso.<br>Recebe um documento JSON constituído por um cabeçalho e um corpo. |
| **domain:port/v{apiVersion}/dataPoints/id** |
| Serviço DELETE, permite a eliminação de um *dataPoint*.<br>O *id* do *dataPoint* tem que ser referido no url. |
| **domain:port/v{apiVersion}/dataPoints/id** |
| Serviço GET, permite a consulta de um *dataPoint*.<br>O *id* do *dataPoint* tem que ser referido no url. |
| **domain:port/v{apiVersion}/dataPoints/caregiver** |
| Serviço GET, permite a consulta dos vários *dataPoints* de um determinado tipo de um participante em específico.<br>Recebe por parâmetros o *schema_namespace*, *schema_name*, *schema_version*, *user_id* e o *CAREGIVER_KEY*. |
| **domain:port/v{apiVersion}/dataPoints** |
| Serviço GET, permite a consulta dos vários dataPoints de um determinado esquema de dados.<br>Recebe por parâmetros o *schema_namespace*, *schema_name* e o *schema_version*. |

Tabela 6.2: Tabela com a API REST do servidor de recursos

Todos os esquemas de dados disponíveis estão no repositório associado à adaptação deste *backend*[1] e estes são os tipos de dados que são utilizados para validar os *dataPoints* inseridos, ou seja, estes são todos aqueles que podem ser inseridos na coleção do *MongoDB*.

### 6.1.3 Comunicação com módulos externos

Para a possível integração com módulos externos foi utilizada a plataforma do Vert.x. No processo de inserção de novos *dataPoints*, depois de validados eles são reencaminhados para um *endpoint* REST. É enviado um documento JSON composto pelo esquema de dados utilizado para validar o documento, como por exemplo *heart-rate* e um corpo do *dataPoint* inserido. Este documento é então inserido no *EventBus* do Vert.x. Neste momento os módulos

[1]Diretório com os Esquemas de dados disponíveis(composto também pelos novos esquemas de dados criados) - https://github.com/luistduarte/omh-dsu-ri/tree/master/resource-server/src/main/resources/schema/omh

que subscreveram o esquema de dados *heart-rate* podem verificar e analisar da maneira que quiserem os dados inseridos. Temos na figura 6.4 um diagrama de sequência com este fluxo.

Isto pode ser importante para por exemplo ser desenvolvido um módulo que tenha a capacidade de verificar se um utilizador que está a fazer a colheita dos dados está a sofrer de arritmia cardíaca. O projeto é então composto por dois *verticles* em que um dos *verticles* cria um *endPoint* REST para receber os dados e os publicar no *Eventbus* assim que os recebe.

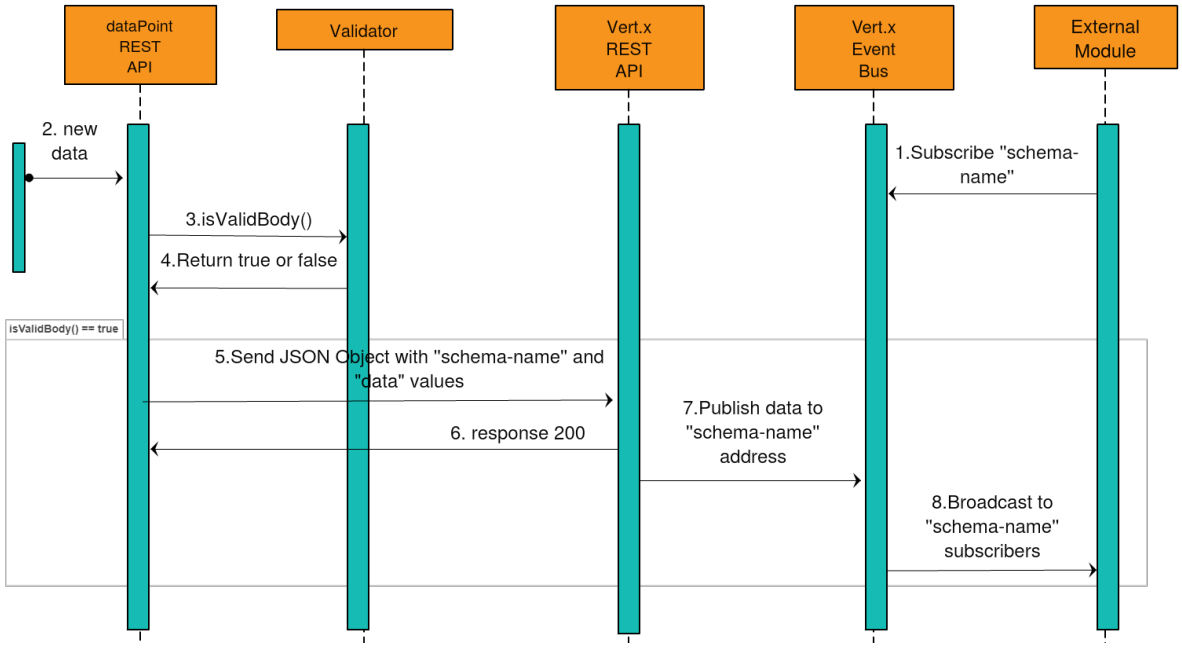


Figura 6.4: Diagrama de sequência com a comunicação com módulos externos

### 6.1.4 Como usar o *backend* desenvolvido num projeto

O sistema foi desenvolvido para ser fácil de utilizar em outros projetos. Tem a capacidade de estar a ser utilizado por diferentes aplicações *mHealth* ao mesmo tempo. Tudo está pronto a utilizar e pode ficar ainda mais completo, pois podem por exemplo ser adicionados novos esquemas de dados.

Relativamente ao servidor de autorização/autenticação e ao servidor de recursos o código está no github [2] e é um *fork* do projeto desenvolvido pela OMH, está documentado como pode ser instalado e executado[3]. O código relativo à extensibilidade com módulos externos também está no github[4], este projeto é composto apenas por dois verticles para os correr pode ser também por terminal e basta correr os comandos:

[2]https://github.com/luistduarte/omh-dsu-ri

[3]https://github.com/luistduarte/omh-dsu-ri/blob/master/README.md#option-2-building-from-source-and-running-nativel

[4]https://github.com/luistduarte/rest.pub.sub

- vertx run SimpleREST.java -cluster
- vertx run Receiver.java -cluster

O Receiver.java pode ser substituído ou adicionado por um outro módulo desenvolvido para receber outro tipo de esquemas de dados e os analisar.

## 6.2 Implementação da aplicação móvel

### 6.2.1 Versões alvo e dependências

Esta aplicação móvel foi desenvolvida com o intuito de conseguir abranger grande maioria dos utilizadores, suporta da versão 16 à versão 25, ou seja, desde o *Android* 4.1 (*Jelly Bean*) até ao *Android* 7.1. De acordo com as informações disponíveis na plataforma de desenvolvimento do *Android* atinge mais de 98% dos utilizadores (48).

O projeto tem um ficheiro por defeito designado por "build.gradle", que é o local onde é efetuada a configuração relativa às versões alvo e às dependências existentes, excertos deste ficheiro de configuração podem ser visualizados nas tabelas 6.3 e 6.4 .

```
defaultConfig {
    applicationId "pt.ua.ieeta.healthintegration"
    minSdkVersion 16
    targetSdkVersion 25
    versionCode 1
    versionName "1.0"
}
```

Tabela 6.3: Excerto da configuração padrão da aplicação móvel relativo às versões

```
dependencies {
    compile fileTree(include: ['*.jar'], dir: 'libs')
    compile 'com.android.support:appcompat-v7:25.3.1'
    compile 'com.android.support:support-v4:25.3.1'
    compile 'com.android.support:design:25.3.1'
    compile files( 'libs/biolib.sdk.jar')
    compile files( 'libs/achartengine-1.1.0.jar')
}
```

Tabela 6.4: Excerto da configuração padrão da aplicação móvel relativo às dependências

Relativamente às dependências temos que ter em atenção que existem duas dependências em especifico que tiveram que ser importadas manualmente. Uma delas é um Software Deve-

lopment Kit (SDK) para android (BioLib) [5] que disponibiliza uma biblioteca para a conexão a um dispositivo VitalJacket e facilita o processamento e a receção dos dados pela aplicação. Uma outra biblioteca que foi utilizada foi o "AChartEngine" [6] que é uma biblioteca para android que permite o desenho de gráficos.

### 6.2.2 Tecnologias integradas e fontes de dados

A aplicação móvel está a utilizar as APIs REST definidas anteriormente em 6.1.1 e em 6.1.2. Utiliza o serviço de autenticação e autorização para todas as interações envolvidas com registo, *login* e acesso aos dados. O serviço de recursos é utilizado para guardar todas as medições efetuadas e também para as consultar posteriormente.

Os dados estão a ser recolhidos através do *VitalJacket* que é um dispositivo médico que conjuga a tecnologia têxtil com soluções avançadas de engenharia biomédica. O dispositivo permite a configuração para adquirir diferentes sinais vitais tais como o ECG, frequência cardíaca e o acelerómetro. A comunicação entre o dispositivo móvel e o sensor é efetuada utilizando o SDK para *android* da biblioteca da BioLib utilizando o perfil de bluetooth SPP.

### 6.2.3 Interações suportadas

Esta aplicação móvel foi desenvolvida com o objetivo principal de efetuar diferentes sessões de medições de dados identificando a relação temporal com a atividade física, todas as funcionalidades que tínhamos como objetivo implementar foram implementadas.

**Menu Lateral de Navegação**

Este menu lateral de navegação permite a um participante efetuar o registo ou efetuar o *Login* na aplicação caso já tenha criado uma conta anteriormente.

**Atividade Principal**

O participante antes de efetuar *Login* tem a atividade Principal praticamente vazia. Pode aceder ao menu lateral de navegação, ou às configurações (Figura 6.5).

[5] http://www.sdk.vitaljacket.com/
[6] https://github.com/ddanny/achartengine

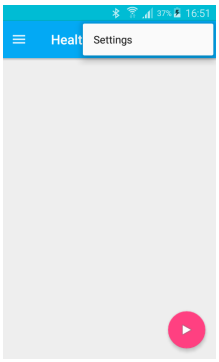


Figura 6.5: Opção na atividade principal para abrir as configurações

Quando o participante efetua o *Login* na aplicação as suas sessões são carregadas e são listadas para que se possa verificar as medições efetuadas correspondentemente a cada uma. Na Figura 6.6 podemos ver do lado esquerdo a situação em que o participante ainda não tem sessões de medições efetuadas e do lado direito quando já efetuou duas sessões de medições.

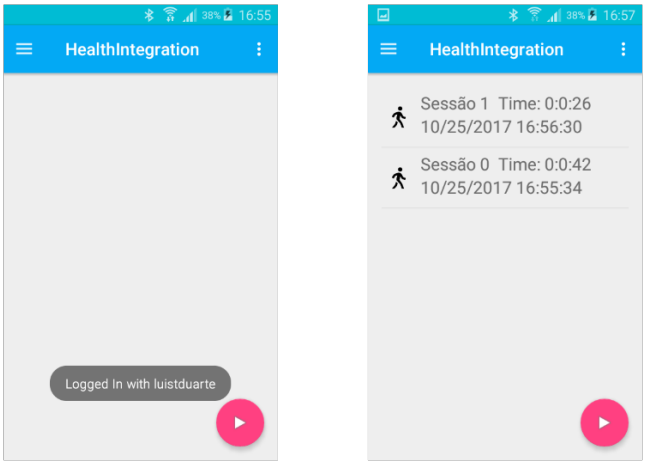


Figura 6.6: Atividade Principal depois do *Login* Efetuado

### Atividade de Configurações

Na atividade de configurações (Figura 6.7) temos a possibilidade de: escolher o dispositivo que vai ser utilizado e ao qual se irá conectar; escolher tanto os tipos de dados que são para recolher como a sua relação temporal em relação à atividade física na altura que a medição está a ser efetuada.

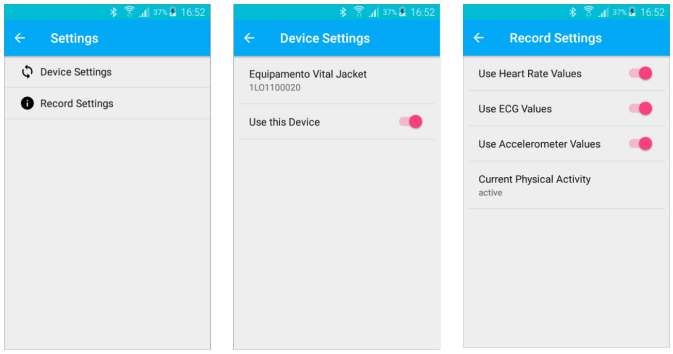


Figura 6.7: Menu de Configurações e as respetivas opções de configuração

### Atividade de Registo

O participante pode ainda não ter uma conta criada, mas a possibilidade de criar uma conta existe na aplicação e está disponível através do menu lateral. No processo de criação de uma nova conta o participante tem que escolher um utilizador que ainda não tenha sido utilizado e escolher uma *password* para essa conta de utilizador(Figura 6.8).

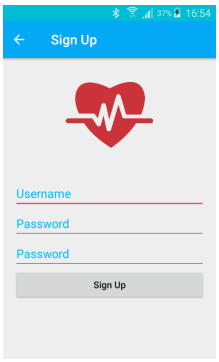


Figura 6.8: Atividade para registo de um novo participante

### Atividade para efetuar o *Login*

Esta atividade (Figura 6.9) pode ser chamada de duas duas maneiras diferentes: ao iniciar uma nova sessão de medições(é verificado que não tem *login* efetuado); escolhendo a opção para efetuar *Login* no menu lateral. É neste momento que o servidor de autorização e autenticação devolve um *token* de acesso.

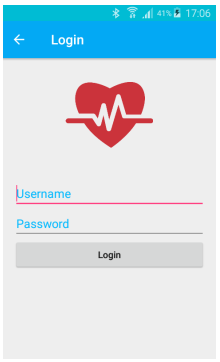


Figura 6.9: Atividade para efetuar o *Login*

### Atividade associada a uma nova sessão de leitura

Depois de efetuar o *login* o participante pode então criar novas sessões de medições (Figura 6.10), pode parar quando quiser e irá voltar a visualizar a sua lista de medições com o acréscimo da última sessão efetuada.


Figura 6.10: Atividade relativa a uma nova sessão de medições

### Atividade Relativa a uma sessão

Depois do participante ter entrado na aplicação com a sua conta pode visualizar as medições relativas a uma sessão. Para isso basta clicar numa das sessões que compõem a lista. Nesta atividade (Figura 6.11) pode visualizar os dados relativos a cada tipo de medição, entre eles a frequência cardíaca, ECG e acelerómetro. Para isso tem disponíveis três diferentes separadores para mudar de tipo de medição.

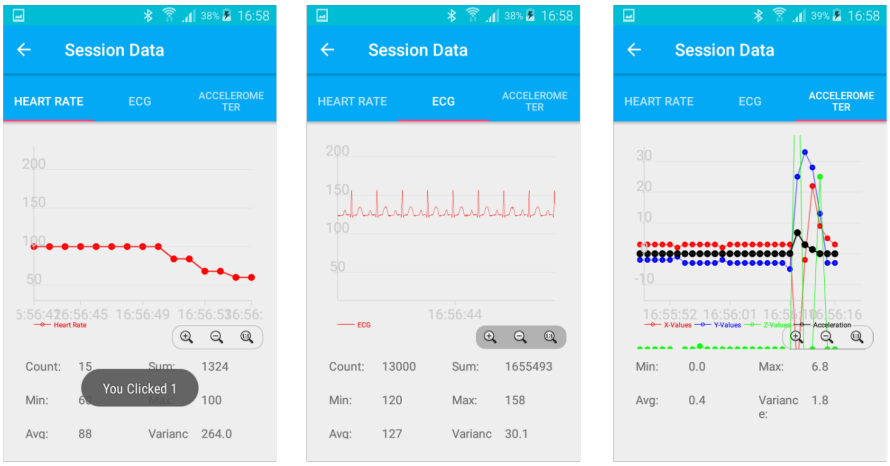


Figura 6.11: Atividade para visualizar os dados relacionados a uma determinada sessão

## 6.3 Implementação da aplicação de revisão

### 6.3.1 Tecnologias integradas e fontes de dados

A aplicação *web* de revisão está a utilizar as APIs REST definidas anteriormente em 6.1.1 e em 6.1.2. Utiliza o serviço de autenticação e autorização para todas as interações envolvidas com registo, *login* e acesso aos dados. O serviço de recursos é utilizado para consultar todas as medições efetuadas pelos participantes em estudo e para guardar os dados demográficos de cada participante adicionado ao seu estudo.

### 6.3.2 Componentes da aplicação

Esta aplicação foi desenvolvida como aplicação *web* com recurso a HTML, *Java Script* e *Cascading Style Sheets (CSS)*. Foi utilizada a plataforma *Bootstrap* (49) da versão 3.3.7 que fornece uma base em HTML e CSS muito popular no mundo das aplicações *web* para desenvolvimento responsivo adaptável a plataformas móveis.

Para a comunicação com a API REST foi utilizado o *jQuery* (50). Relativamente à visualização dos dados para cada sessão foi utilizado a biblioteca *HighCharts* (51) para a visualização dos dados relacionados com o acelerómetro e ECG. Para a frequência cardíaca decidimos utilizar um visualizador padrão desenvolvido pela OMH denominado de *Open mHealth Web Visualizations* (52).

### 6.3.3 Interações suportadas

Esta aplicação *web* foi desenvolvida com o objetivo principal de efetuar a revisão dos dados inseridos pelos participantes. É então possível adicionar vários participantes e verificar as várias sessões de medições de dados.

### Página Principal

A página principal tem uma secção de boas vindas e ainda permite o registo e a entrada na aplicação (Figura 6.12).


Figura 6.12: Página principal da aplicação de revisão

### Página de Registo

O revisor pode ainda não ter uma conta criada, por isso antes de entrar na aplicação terá que efetuar o registo de uma nova conta. Na criação é necessário escolher uma nova conta que ainda não tenha sido utilizada acompanhada de uma password (Figura 6.13).


Figura 6.13: Página de registo na aplicação de revisão

**Página para efetuar o *Login***

Esta página (Figura 6.14) pode ser chamada de duas maneiras diferentes. Ao clicar nos participantes da secção de boas vindas é detetado que o *login* ainda não foi efetuado e é feito o redirecionamento para esta página, ou então ao clicar no botão Entrar na página principal. Depois de efetuar o *login* pode visualizar os seus participantes.


Figura 6.14: Página de *login* na aplicação de revisão

**Página com todos os participantes**

Esta página é composta por uma grelha onde pode visualizar todos os seus participantes e também adicionar novos participantes. Assim que são adicionados novos participantes a grelha vai ficando mais composta, como na Figura 6.15

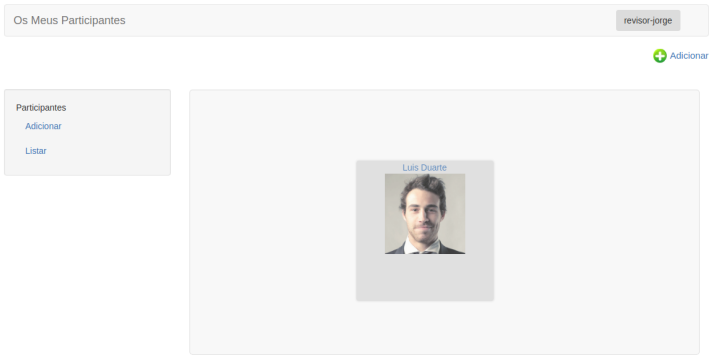


Figura 6.15: Página com grelha de participantes

**Página para adicionar novo participante**

Nesta página (Figura 6.16) pode adicionar um novo participante, para isso tem que identificar o *username* do participante e todos os dados demográficos disponíveis deste participante. Depois de adicionar vai ser redirecionado para a página relativa a todos os participantes.

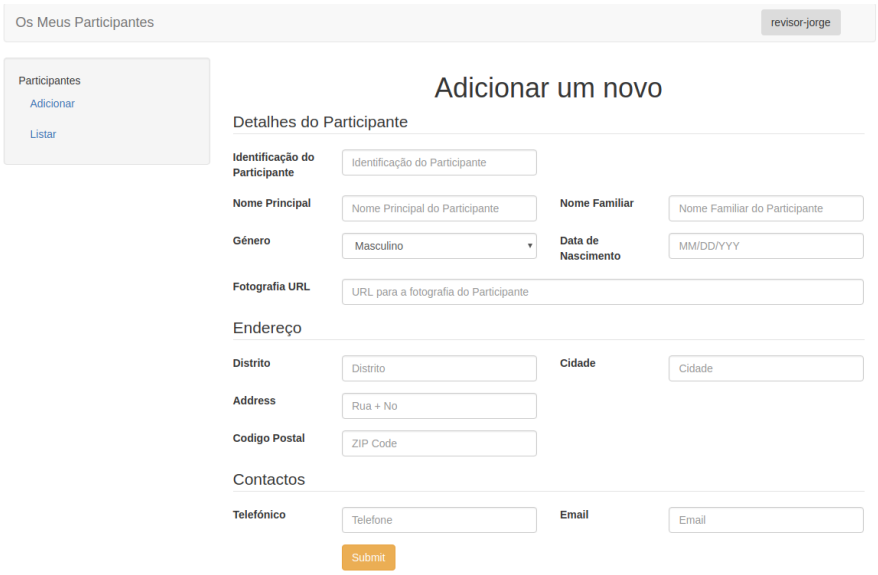


Figura 6.16: Página para adicionar novo participante ao estudo

### Página com leituras do participante

Nesta página pode visualizar todos os dados demográficos e fisiológicos do participante. Os dados demográficos podem ser atualizados e os dados fisiológicos que podem ser consultados são todos aqueles inseridos pelo participante. Para isso terá que escolher um intervalo de datas para procurar por sessões de leitura nesse intervalo, depois disso as várias sessões compõem um *dropdow menu* (Figura 6.17) onde pode escolher a sessão que pretende rever.

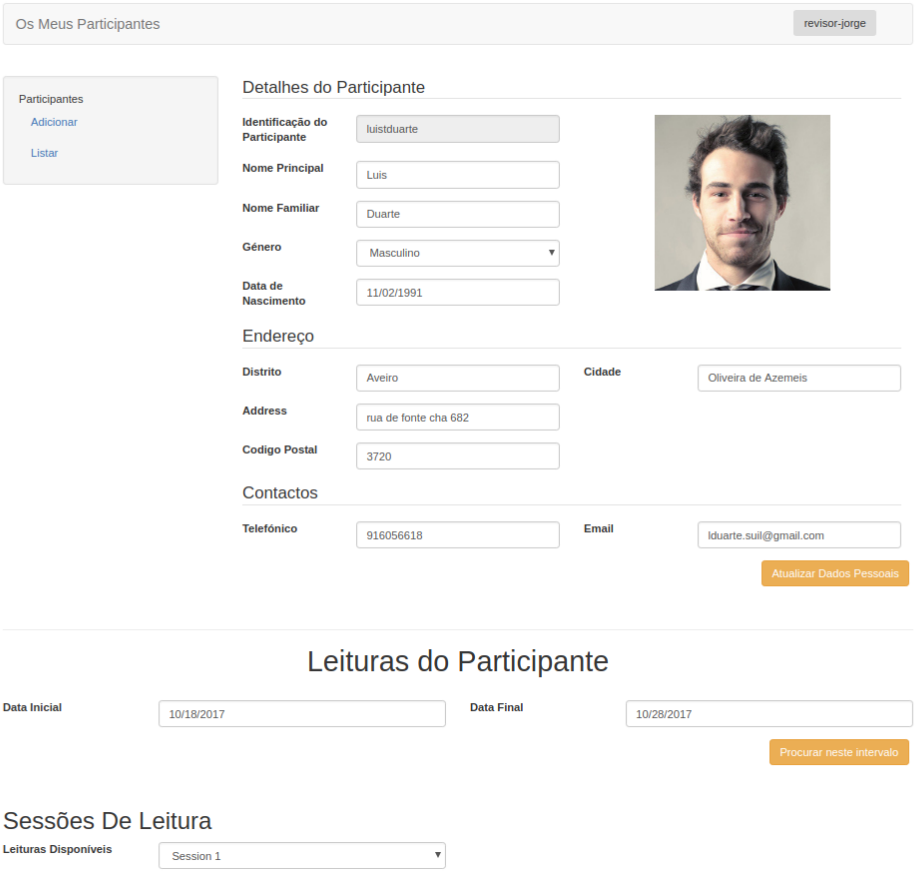


Figura 6.17: Página com dados demográficos e fisiológicos do participante

Podemos então visualizar os dados em gráficos relativos a cada tipo de dados recolhido (Figura 6.18).

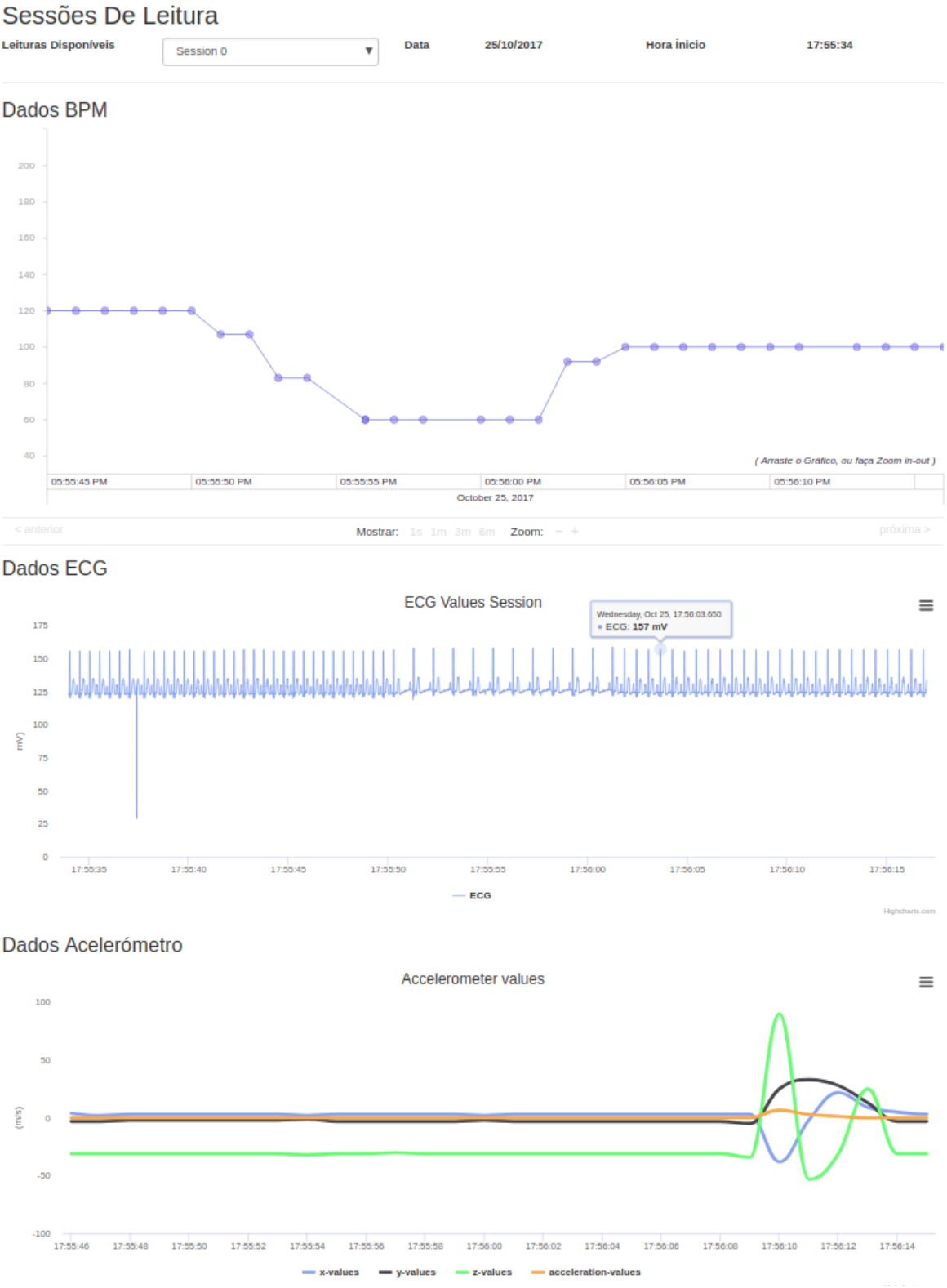


Figura 6.18: Dados fisiológicos do participante numa determinada sessão

Relativamente a cada tipo de dados pode ser efetuado o *zoom* para em cada um dos gráficos, e ainda no caso do acelerómetro escolher os dados a apresentar, neste caso foi seleccionada apenas a aceleração (Figura 6.19).


Figura 6.19: Dados fisiológicos do participante mais detalhados

## 6.4 Como aplicar e estender a plataforma em novos contextos

### 6.4.1 Criar um novo tipo de dados

A criação de um novo tipo de dados pode ser feita com a ajuda do repositório *schemas* (53) que foi criado pela organização da OMH. Para isto, o desenvolvimento do novo esquema de dados deve ser feito em JSON *schema*. Vou descrever agora a lista de tarefas para conseguir efetuar a criação de um novo tipo de dados. Para este tutorial vou criar o tipo de dados acelerómetro. É um dos tipos de dados que são obtidos através do *VitalJacket*.

1. Clonar o repositório correspondente para um diretório à sua escolha De seguida vamos fazer duas coisas principais: uma delas é criar o ficheiro que define o novo tipo de dados;

a outra é criar um ficheiro com uma amostra do novo tipo de dados.

2. Criar um ficheiro que define o tipo de dados em JSON *schema*, para isso, adicione um novo ficheiro no diretório *schema/omh*. O nome deste ficheiro tem que ser composto pelo nome que quer dar ao tipo de dados e a versão correspondente, tem que terminar com a extensão *.json*. No meu caso criei com o nome *accelerometer-1.0.json*

   A versão escolhida aqui é a 1.0 mas podia ser qualquer outra. Pode reparar que no diretório *schema/omh* tem acesso a todos os ficheiros que definem todos os tipos de dados.

3. Adicionar uma nova pasta ao diretório *testdata/omh* como o nome respetivo ao ficheiro criado anteriormente. Neste caso criar uma pasta com o nome *accelerometer*.

4. Vai agora criar uma pasta correspondente à versão introduzida no ponto 2. Neste caso é 1.0 e uma outra pasta dentro da criada anteriormente com o nome *shouldPass*.

5. Dentro do diretório *shouldPass* vai ter que criar um ou vários ficheiros para ser utilizados como amostras do tipo de dados. O objetivo deste diretório é ter várias amostras válidas testando-as com o tipo de dados criado no ponto 2. Para este tutorial criei o ficheiro *example.json*.

   Neste ponto o nome do ficheiro é opcional só tem que acabar com a extensão *.json*

   Como criou o diretório *shouldPass*, pode também criar o diretório *shouldFail* criando também vários ficheiros para testar o tipo de dados criado.

   Neste ponto tem tudo preparado para começar a criar o novo tipo de dados e validar a amostra com a nova definição do novo tipo de dados criados.

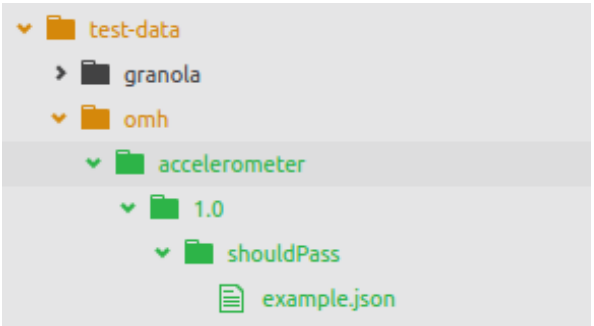


Figura 6.20: Esquema de diretório para o exemplo do novo tipo de dados

6. Vamos agora preencher o ficheiro com a definição do novo tipo de dados, aquele que criou no ponto 2. O ficheiro vai ser criado no formato de JSON *schema*. Para suportar a criação deste ficheiro pode reutilizar *schemas* existentes (54) e referênciá-los, deste modo estará a criar um modelo com tipos de dados normalizados. No meu caso vou reutilizar

um *schema* para definir a data/hora e um outro para definir a relação temporal relativa à atividade física no momento da leitura. Os restantes dados estão relacionados com o acelerómetro em si, a sessão e a leitura respetiva. O ficheiro fica do seguinte modo:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "description": "Accelerometer measure of a person",
    "type": "object",
    "definitions": {
        "time_frame": { "$ref": "time-frame-1.x.json"},
        "temporal_relationship_to_physical_activity": {
          "$ref": "temporal-relationship-to-physical-activity-1.x.json"},
        "part_number": {"type":"integer", "minimum": 0 },
        "session":{"type":"integer", "minimum": 0 },
        "unit":{"type":"string", "enum":["mS"]}
    },
    "properties": {
        "effective_time_frame": { "$ref": "#/definitions/time_frame" },
        "temporal_relationship_to_physical_activity": {
            "$ref": "#/definitions/temporal_relationship_to_physical_activity"},
      "accelerometer": {
        "type": "object",
        "properties": {
          "values": {
            "type": "object",
            "properties": {
              "x": {"type": "integer"},
              "y": {"type": "integer"},
              "z": {"type": "integer"}
            }
          },
          "unit": { "$ref": "#/definitions/unit" },
          "part_number":{ "$ref": "#/definitions/part_number" },
          "session":{ "$ref": "#/definitions/session" }
        },
        "required": ["values", "part_number", "unit", "session"]
      }
    },
    "required": ["accelerometer", "effective_time_frame"]
}
```

Figura 6.21: JSON *schema* para o novo tipo de dados de acelerómetro

7. Preencha o ficheiro com uma amostra do novo tipo de dados, para isso tem que ter em conta a definição utilizada, porque se esta amostra não for compatível não vai passar no validador.

```
{
        "effective_time_frame": {
                "date_time": "2017-09-30T01:31:50Z"
        },
        "accelerometer": {
                "unit": "mS",
                "values": {
                        "x": 3,
                        "y": -4,
                        "z": 33
                },
                "session": 30,
                "part_number": 46
        },
        "temporal_relationship_to_physical_activity": "active"
}
```


Figura 6.22: Exemplo do tipo de dados de acelerómetro

8. Para compilar e executar o validador tem que executar o comando *./gradlew test-data-validator:bootRun* no diretório principal "schemas" ou então execute o comando *./gradlew bootRun* no diretório *schemas/test-data-validator*.

### 6.4.2 Como utilizar o novo tipo de dados criado

Para adicionar o novo tipo de dados criado, tem que o integrar com os restantes tipos de dados existentes.

Tendo em conta que já tem o repositório omh-dsu-ri clonado através do comando "git clone https://github.com/luistduarte/omh-dsu-ri', terá que copiar o seu novo tipo de dados para o diretório "omh-dsu-ri/resource-server/src/main/resources/schema/omh" que é onde se encontram os restantes tipos de dados válidos. A partir deste momento a inserção de novos dados compatíveis com o *data schema* criado já é possível, para isso na criação e inserção de *datapoints* é necessário especificar no cabeçalho o tipo de dados que está a ser inserido, para este ser validado e corretamente criado.

# Capítulo 7

# Resultados

## 7.1 Protótipo integrado

O protótipo que foi construído tem a capacidade de suportar experiências de I&D pois tem a capacidade de gerir contas de utilizadores participantes e revisores que estejam envolvidos num projeto de I&D. Fornece a possibilidade de criação (inserção) de novas medições por parte dos participantes e ainda consultar estas medições inseridas. Os revisores podem consultar todas as medições inseridas pelos participantes envolvidos no seu estudo.

O servidor de recursos suporta a extensibilidade dos dados sem que se tenha de alterar o esquema da base de dados o que é uma grande vantagem. Este servidor utiliza ainda o servidor de autorização e autenticação para permitir o acesso aos pedidos recebidos.

Este protótipo tem então todas as características de uma plataforma *mHealth*, ou seja, permite que exista um participante a recolher dados de dispositivos médicos, encaminhados através do *smartphone*, e que estes estejam a ser monitorizados pelo revisor do estudo sem que eles estejam próximos um do outro, dado que todos os dados recolhidos são guardados no servidor.

## 7.2 Adequação das plataformas estudadas

Na fase exploratória conseguimos encontrar duas plataformas que serviam para ser utilizadas como *backend* do nosso protótipo, para além do OMH que foi a plataforma utilizada conseguimos também concluir que o FHIR servia para as necessidades existentes. O *Google Fit* foi logo excluído pois após um estudo um pouco mais aprofundado percebemos que a definição relativamente às propriedades do tipo de dados, não suporta vetores, ficando logo praticamente excluído como possível escolha para o *backend*.

Apesar de termos duas plataformas que poderiam ser utilizadas como *backend* nenhuma delas se encontrava pronta para utilização sem qualquer adaptação. O FHIR por um lado não suportava gestão de identidades e não controlava o acesso aos pedidos recebidos. Rela-

tivamente ao OMH o que acontecia é que os dados não eram validados, e os dados inseridos eram proprietários, ou seja, apenas a pessoa que fazia a inserção dos dados recolhidos os podia voltar a consultar, isto porque era apenas utilizado o *token* de acesso para identificar de que pessoa seria os dados consultados. Podemos concluir aqui que, nenhuma das plataformas estudadas podia ser utilizada para o desenvolvimento de aplicações *mHealth* sem qualquer tipo de adaptação, o que não impossibilita a sua utilização após algumas adaptações pois até são convenientemente completas.

## 7.3 Recomendações para o desenvolvimento de aplicações de *mHealth*

Face ao trabalho desenvolvido, as recomendações que podemos dar é que a plataforma do OMH é bastante apelativa e de possível solução para um *backend* de uma aplicação *mHealth*. Tem uma boa quantidade de esquemas de dados já disponíveis e é de fácil extensibilidade. Caso a aplicação seja de pequena dimensão esta solução é prática, pois facilmente colocamos o *backend* num servidor, podendo inserir dados e consultar os mesmos. Quando estamos a falar do desenvolvimento de uma aplicação para uma clínica e ou hospital, já não recomendamos o OMH, pois o FHIR é bastante mais completo e serviria muito melhor as necessidades, sendo também mais fácil a posterior exportação e importação de dados clínicos e ou hospitalares.

Ao utilizar o resultado desenvolvido neste trabalho terá o benefício de começar com um *backend* mais robusto e completo, terá a certeza que os dados inseridos respeitam o JSON *schema* associado o que é uma vantagem para uma posterior consulta. Terá ainda a possibilidade de integrar o sistema com módulos externos permitindo uma análise sobre os dados recolhidos.

# Capítulo 8

# Conclusão

Neste trabalho, foi desenvolvida uma investigação de possíveis *backends* para dar suporte a aplicações *mHealth*. Começamos por selecionar três plataformas que se revelaram candidatas a ser utilizadas para este fim. Para se chegar a uma melhor conclusão baseada em factos práticos foi desenvolvido um conjunto de atividades exploratórias relativamente a cada uma das plataformas de *backend* com o objetivo de aprender melhor o funcionamento, os pontos fortes e fracos de cada um.

Chegamos então à conclusão que o *Open mHealth* seria o melhor *backend* dentro dos explorados para dar suporte às aplicações de *mHealth*, para o cenário proposto. Posto isto, foi então desenvolvido um sistema integrado utilizando um problema de I&D para servir como prova de conceito colocando à prova o *backend* escolhido para concluir se este era ou não viável. O sistema foi desenvolvido com sucesso e chegamos à conclusão que o *Open mHealth* pode ser utilizado como *backend* de aplicações *mHealth* tendo em conta que foram apenas necessárias algumas adaptações e implentação extra com o objetivo de enriquecer a plataforma e a completar.

Esta investigação e exploração desenvolvida vem então concluir que o *backend* do *Open mHealth* é fiável e capaz de servir aplicações de *mHealth*.

## 8.1 Trabalho futuro

Tendo em conta o trabalho desenvolvido durante esta dissertação, existem pontos a ser melhorados e complementados, entre eles podemos ter a criação de novos esquemas de dados. Ao estender o conjunto de esquemas de dados do *backend* do *Open mHealth* a plataforma ficará mais completa abrangendo cada vez mais todas as necessidades existentes para se guardar diferentes tipos de dados.

Foi criada a possibilidade de ligação de módulos externos através de um *message bus*. Estes módulos podem subscrever um determinado tipo de dados para posterior análise. Não foi criado nenhum módulo completo para analisar os tipos de dados recolhidos, no entanto

pode ser desenvolvido diferentes módulos com esse objetivo. Um módulo depois de efetuar a subscrição, todos os dados do tipo que foi subscrito vão ser reencaminhados para ele. Existe a possibilidade de diferentes módulos subscreverem o mesmo tipo de dados para efetuar diferentes análises.

Relativamente à segurança e proteção dos dados demográficos e fisiológicos dos participantes no caso de estudo pode também ser melhorado, garantindo a utilização de dados pseudo-anonimizados. Tendo em conta que a comunicação com o *backend* é toda feita através de pedidos HTTP, a utilização deste *backend* com uma aplicação iOS não será problema, mas podem ser desenvolvidos testes com diferentes dispositivos utilizando outros tipos de dados.

# Referências


[1] Saee Hamine, Emily Gerth-Guyette, Dunia Faulx, Beverly B Green, and Amy Sarah Ginsburgh. Impact of mhealth chronic disease management on treatment adherence and patient outcomes: A systematic review. *Journal of Medical Internet Research*, 2015. doi: 10.2196/jmir.3951.

[2] Elske Ammenwerth, Jytte Brender, Pirkko Nykänen, Hans-Ulrich Prokosch, Michael Rigby, and Jan Talmon. Visions and strategies to improve evaluation of health information systems: Reflections and lessons based on the his-eval workshop in innsbruck. *Journal of Medical Informatics*, 2004. doi: 10.1016/j.ijmedinf.2004.04.004.

[3] Gunther Eysenbach. What is e-health? *Journal of Medical Internet Research*, 2001. doi: 10.2196/jmir.3.2.e20.

[4] WHO. ehealth, 2017. URL `http://www.emro.who.int/health-topics/ehealth/`. (Acedido a 1 de Junho de 2017).

[5] Ryhan Ebad. Telemedicine: Current and future perspectives. *Journal of Computer Science*, 2013.

[6] WHO. Oportunities and developments in member states, 2010. URL `http://www.who.int/goe/publications/goe_telemedicine_2010.pdf`.

[7] WHO. New horizons for health through mobile technologies, 2011. URL `http://www.who.int/goe/publications/goe_mhealth_web.pdf`.

[8] Bruno M.C.Silva, Joel J.P.C.Rodrigues, Isabelde la Torre Díez, Miguel López-Coronado, and Kashif Saleem. Mobile-health:a review of current state in 2015. *Journal of Biomedical Informatics*, 2015. doi: 10.1016/j.jbi.2015.06.003.

[9] Maxwell Software. M. software, daily carb – carbohydrate, glucose, medication, blood pressure and exercise tracker, 2014. URL `https://itunes.apple.com/us/app/daily-carb-carbohydrate-glucose/id536425111?mt=8`. (Acedido a 5 de Junho de 2017).

[10] Azumio Inc. Azumio inc., glucose buddy – diabetes logbook manager w/syncing, blood pressure, weight tracking, 2014. URL https://itunes.apple.com/us/app/glucose-buddy-diabetes-logbook/id294754639?mt=8. (Acedido a 5 de Junho de 2017).

[11] Sanofi US. Sanofi-aventis, gomeals, 2014. URL https://itunes.apple.com/us/app/gomeals/id336651139?mt=8. (Acedido a 5 de Junho de 2017).

[12] Hadi Banaee, Mobyen Uddin Ahmed, and Amy Loutfi. Data mining for wearable sensors in health monitoring systems: A review of recent trends and challenges. *Sensors — Open Access Journal*, 2013. doi: 10.3390/s131217472.

[13] Valentino Lee, Heather Schneider, and Robbie Schell. *Mobile Applications: Architecture, Design, and Development*. Prentice Hall PTR Upper Saddle River, NJ, USA 2004, 2004. ISBN 0131172638.

[14] Internet Engineering Task Force. Oauth 2.0, 2012. URL https://tools.ietf.org/html/rfc6749. (Acedido a 7 de Junho de 2017).

[15] Barry Leiba. Oauth web authorization protocol. *IEEE Internet Computing*, 2012. doi: 10.1109/MIC.2012.11.

[16] Android Developers. Room persistence library, 2017. URL https://developer.android.com/topic/libraries/architecture/room.html. (Acedido a 20 de Outubro de 2017).

[17] Bluetooth SIG. what is bluetooth?, 2017. URL https://www.bluetooth.com/what-is-bluetooth-technology/discover-bluetooth. (Acedido a 23 de Outubro de 2017).

[18] K.V.S.S.S.S. Sairam, N. Gunasekaran, and S.R. Redd. Bluetooth in wireless communication. *IEEE Communications Magazine*, 2002. doi: 10.1109/MCOM.2002.1007414.

[19] Jad Noueihed, Robert Diemer, and Samarjit Chakraborty. Comparing bluetooth hdp and spp for mobile health devices. *IEEE*, 2010. doi: 10.1109/BSN.2010.40.

[20] Jakob Jenkove. Web service message formats, 2014. URL http://tutorials.jenkov.com/web-services/message-formats.html. (Acedido a 8 de Junho de 2017).

[21] DR. M. ELKSTEIN. Learn rest: A tutorial, 2014. URL http://rest.elkstein.org/2008/02/what-is-rest.html. (Acedido a 8 de Junho de 2017).

[22] Alex Rodriguez. Restful web services: The basics, 2008. URL https://www.ibm.com/developerworks/library/ws-restful/index.html. (Acedido a 8 de Junho de 2017).

[23] Vert.x, 2017. URL http://vertx.io/. (Acedido a 12 de Outubro de 2017).

[24] Seongmin Woo. Inside vert.x. comparison with node.js, 2017. URL https://www.cubrid.org/blog/inside-vertx-comparison-with-nodejs. (Acedido a 12 de Outubro de 2017).

[25] Internet Engineering Task Force. The javascript object notation (json) data interchange format, 2014. URL https://tools.ietf.org/html/rfc7159. (Acedido a 8 de Junho de 2017).

[26] PwC. How supportive is the regulatory framework for mobile health applications?, 2013. URL https://www.pwc.com/gx/en/healthcare/mhealth/mhealth-insights/assets/pwc-mhealth-how-supportive-is-the-regulatory-framework-for-mobile-health-applications.pdf. (Acedido a 11 de Junho de 2017).

[27] Tamara Vagg, Prof Barry J. Plant, and Dr Sabin Tabirca. A general mhealth design pipeline. *ACM New York*, 2016. doi: 10.1145/3007120.3007147.

[28] Open mHealth. Open mhealth, about, 2015. URL http://www.openmhealth.org/organization/about/. (Acedido a 12 de Junho de 2017).

[29] Deborah Estrin and Ida Sim. Open mhealth architecture: An engine for health care innovation. *Science*, 330:759–760, 2010. doi: 10.1126/science.1196187.

[30] Open mHealth. About schemas, 2015. URL http://www.openmhealth.org/documentation/#/schema-docs/overview. (Acedido a 12 de Junho de 2017).

[31] Open mHealth. Heart rate, json schema, 2015. URL http://www.openmhealth.org/documentation/#/schema-docs/schema-library/schemas/omh_heart-rate. (Acedido a 12 de Junho de 2017).

[32] Keith J. Dreyer. Why ihe? *RadioGraphics*, 20, 2000. doi: 10.1148/radiographics.20.6.g00no381583.

[33] HL7.org. Introducing hl7 fhir, 2011+. URL http://hl7.org/fhir/summary.html. (Acedido a 13 de Junho de 2017).

[34] Health Level Seven International. About hl7, 2017. URL http://www.hl7.org/about/index.cfm?ref=nav. (Acedido a 13 de Junho de 2017).

[35] CorePoint Health. The hl7 evolution, 2010. URL https://corepointhealth.com/resource-center/white-papers/evolution-hl7. (Acedido a 13 de Junho de 2017).

[36] Jr. George W. Beeler. Introduction to: Hl7 reference information model (rim), 2011. URL https://www.hl7.org/documentcenter/public_temp_71980980-1C23-BA17-0CA506478108F246/calendarofevents/himss/2011/HL7%20Reference%20Information%20Model.pdf.

[37] Google. Googlefit platform overview, 2016. URL https://developers.google.com/fit/overview. (Acedido a 15 de Junho de 2017).

[38] Google. Rest api, 2016. URL https://developers.google.com/fit/rest/. (Acedido a 15 de Junho de 2017).

[39] Biodevices. Vitaljacket, 2010. URL http://www.vitaljacket.com/?page_id=790. (Acedido a 23 de Outubro de 2017).

[40] Open mHealth. Start storing data, 2015. URL http://www.openmhealth.org/documentation/#/store-data/storage-overview. (Acedido a 20 de Junho de 2017).

[41] Andrius Velykis. Hapi fhir - some ways you can use hapi fhir, 2017. URL http://hapifhir.io/. (Acedido a 22 de Junho de 2017).

[42] Andrius Velykis. Jpa server, 2017. URL http://hapifhir.io/doc_jpa.html. (Acedido a 22 de Junho de 2017).

[43] Robin Henricsson. Document oriented nosql databases, 2011. URL https://www.diva-portal.org/smash/get/diva2:832580/FULLTEXT01.pdf. (Acedido a 19 de Outubro de 2017).

[44] Ben Alex, Luke Taylor, Rob Winch, and Gunnar Hillert. Spring security reference, 2015. URL https://docs.spring.io/spring-security/site/docs/5.0.0.M5/reference/htmlsingle/. (Acedido a 18 de Outubro de 2017).

[45] MongoDB Inc. Mongodb for giant ideas, 2017. URL https://www.mongodb.com/. (Acedido a 20 de Outubro de 2017).

[46] Oracle Corporation and/or its affiliates. My sql, 2017. URL https://www.mysql.com/. (Acedido a 20 de Outubro de 2017).

[47] The PostgreSQL Global Development Group. Postgresql, 2017. URL https://www.postgresql.org/. (Acedido a 20 de Outubro de 2017).

[48] Android Developers. Android developers - painéis, 2017. URL https://developer.android.com/about/dashboards/index.html. (Acedido a 16 de Outubro de 2017).

[49] Bootsrap Team. Getting started - bootstrap, 2017. URL https://getbootstrap.com/docs/3.3/getting-started/. (Acedido a 19 de Outubro de 2017).

[50] The jQuery Foundation. jquery, 2017. URL https://jquery.com/. (Acedido a 19 de Outubro de 2017).

[51] Highcharts. Highchars - make your data come alive, 2017. URL https://www.highcharts.com/. (Acedido a 19 de Outubro de 2017).

[52] The jQuery Foundation. Open mhealth web visualizations, 2016. URL https://github.com/openmhealth/web-visualizations. (Acedido a 19 de Outubro de 2017).

[53] OMH. Open mhealth schema repository, 2017. URL https://github.com/openmhealth/schemas. (Acedido a 4 de Setembro de 2017).

[54] Open mHealth. Schema library, 2015. URL http://www.openmhealth.org/documentation/#/schema-docs/schema-library. (Acedido a 4 de Setembro de 2017).