



Aplicação de demonstração da tecnologia Hype

Hype Labs

2016 / 2017

1091114 José Carlos Cerqueira da Silva



Aplicação de demonstração da tecnologia Hype

Hype Labs

2016 / 2017

1091114 José Carlos Cerqueira da Silva



Licenciatura em Engenharia Informática

Julho 2017

Orientador ISEP: **Goreti Marreiros**

Supervisor Externo: **José Teixeira**

«Aos pais, irmão, namorada e amigos... »

Agradecimentos

Dirijo os meus agradecimentos a todos aqueles que me ajudaram ao longo do meu percurso académico.

Ao ISEP, pelos conhecimentos que me permitiu adquirir ao longo da minha formação académica em Engenharia Informática.

À Hype Labs e respetiva equipa, pela oportunidade de colaborar com eles, pelo acolhimento fantástico, pelos conhecimentos transmitidos e pelo suporte incondicional, o que permitiu superar todas as dificuldades.

Aos meus orientadores, Eng. José Teixeira da empresa Hype Labs e à professora Goreti Marreiros do ISEP, pela disponibilidade e aconselhamento constante para quaisquer dificuldades que surgissem.

Por último, mas não menos importante, à minha família, namorada e amigos pelo apoio durante todo o percurso.

Resumo

Este relatório descreve o projeto efetuado no âmbito da unidade curricular de Projeto Estágio (PESTI), da Licenciatura em Engenharia Informática do Instituto Superior de Engenharia do Porto (ISEP) no ano letivo 2016/2017. O projeto decorreu na empresa Hype Labs, com escritório no Parque de Ciência e Tecnologia da Universidade do Porto (UPTEC), e teve como objetivo o desenvolvimento de três aplicações para demonstração da tecnologia Hype em sistemas móveis Android. Como tal foram desenvolvidas uma aplicação de alarme, um *chat*, e uma aplicação de recolha de métricas de desempenho da rede.

Na realização do projeto foi utilizada a metodologia ágil Scrum para auxiliar no alcance dos objetivos propostos, o que levou a que existissem várias reuniões com o supervisor da empresa onde foram apresentadas demonstrações das implementações, efetuadas correções e ajustes às mesmas. No final do estágio o trabalho realizado permitiu que todos os objetivos definidos fossem alcançados o que demonstrou ser uma experiência positiva e bastante enriquecedora.

Palavras-chave (Tema):

- Tecnologia
- Hype
- Demonstração
- Dispositivos móveis

Palavras-chave (Tecnologias):

- Android
- Comunicação
- Tecnologia Hype
- Redes em malha
- Wi-Fi
- Bluetooth

Índice

1	<i>Introdução</i>	1
1.1	Apresentação do projeto/estágio	1
1.1.1	Planeamento do projeto	1
1.1.2	Reuniões de acompanhamento	2
1.2	Apresentação da organização	3
1.3	Contributos deste trabalho	4
1.4	Organização do relatório	4
2	<i>Contexto</i>	5
2.1	Enquadramento do problema	5
2.2	Áreas de negócio	5
2.3	Estado da arte	6
2.3.1	Tecnologias de comunicação offline	6
2.3.2	Serialização de dados	7
2.3.3	Android	8
2.3.4	Atividades	9
2.3.5	Fragmentos	11
2.3.6	Android Assets	14
2.4	Contextualização tecnológica	15
•	Android Studio	15
•	GIT	15
•	Scrum	15
•	Java	15
•	UML	16
•	XML	16
•	JSON	16
•	Android	16
2.5	Processo de desenvolvimento e método de trabalho	17
3	<i>Descrição técnica</i>	19
3.1	Análise e Modelação	19
3.1.1	Implementação 01 - configurações	19
3.1.2	Implementação 02 - alarme	20
3.1.3	Implementação 03 - chat	20

3.1.4	Implementação 04 - métricas.....	21
3.2	Engenharia de requisitos.....	23
3.2.1	Stakeholders.....	23
3.2.2	Atores	24
3.2.3	Requisitos funcionais.....	24
3.2.3.1	Implementação 01 - configurações.....	24
3.2.3.2	Implementação 02 - alarme.....	25
3.2.3.3	Implementação 03 - chat	28
3.2.3.4	Implementação 04 - métricas	32
3.2.4	Requisitos não funcionais.....	36
3.3	Hierarquia e estrutura de classes	37
3.3.1	Implementação 02 - alarme	39
3.3.2	Implementação 03 - chat.....	40
3.3.3	Implementação 04 - métricas.....	44
3.4	Diagramas de sequência	47
3.4.1	Implementação 02 – alarme.....	48
3.4.2	Implementação 03 – chat.....	48
3.4.3	Implementação 04 – métricas	49
3.5	Implementação.....	50
3.5.1	Implementação 01 – configurações	50
3.5.1.1	Implementação do menu de navegação.....	51
3.5.1.2	Download e integração da <i>Framework Hype</i>	53
3.5.1.3	Estruturação do código no IDE.....	55
3.5.1.4	Classe DemoDefs.....	56
3.5.1.5	Criação da classe de Logs	56
3.5.1.6	Criação da persistência dos dados	57
3.5.2	Implementação 02 - alarme	57
3.5.2.1	Interface gráfica	58
3.5.2.2	Conteúdo de classes	59
3.5.3	Implementação 03 - <i>chat</i>	62
3.5.3.1	Persistência	63
3.5.3.2	Interface gráfica	64
3.5.3.3	Conteúdo de classes	68
3.5.4	Implementação 04 - métricas.....	80
3.5.4.1	Persistência	82
3.5.4.2	Interface gráfica	84
3.5.4.3	Conteúdo de classes	88

4	<i>Experiências e testes</i>	100
4.1	Testes unitários.....	100
4.2	Testes funcionais	101
4.3	Testes de aceitação	102
4.4	Testes de sistema.....	105
5	<i>Conclusões</i>	109
5.1	Objetivos realizados.....	109
5.2	Outros trabalhos realizados	110
5.3	Limitações e trabalho futuro	110
5.4	Apreciação final	111
	<i>Bibliografia</i>	113
	<i>Anexos</i>	116
	Anexo 1 - Reuniões de acompanhamento.....	116
	Supervisor da Hype Labs.....	116
	Supervisor do ISEP	122
	Anexo 2 – Estados da arte	123
	Anexo 3 – Testes unitários.....	132
	Anexo 4 - Planeamento	135

Índice de Figuras

Figura 1 - Equipa e escritório da Hype Labs	3
Figura 2 – Comparação de performance ao processar JSON e XML	8
Figura 3 - Cotas de mercado das versões Android no dia 3 de Abril de 2017	8
Figura 4 - Ciclo de vida de uma atividade.....	9
Figura 5 - Ciclo de vida de um fragmento	12
Figura 6 - Área de trabalho do Android Assets.....	14
Figura 7 - Resultado de grandes projetos de software	17
Figura 8 - Processo de Scrum	18
Figura 9 - Modelo de domínio da implementação 02 – alarme	20
Figura 10 - Modelo de domínio da implementação 03 - chat	21
Figura 11 - Modelo de domínio da implementação 04 – métricas	22
Figura 12 - Diagrama de casos de uso implementação 02 - alarme	25
Figura 13 - Diagrama de casos de uso implementação 03 - chat	28
Figura 14 - Diagrama de casos de uso implementação 04 - métricas	33
Figura 15 - Diagrama de classes das Interfaces Hype e Main Activity.....	38
Figura 16 - Diagrama de classes da implementação 02 – alarme	39
Figura 17 - Diagrama de classes do model da implementação 03 - chat	40
Figura 18 - Diagrama de classes do controller da implementação 03 - chat.....	42
Figura 19 - Diagrama de classes do model da implementação 04 – métricas.....	44
Figura 20 - Diagrama de classes do controller da implementação 04 - métricas.....	46
Figura 21 - Diagrama sequência Implementação 02 CU 03	48
Figura 22 - Diagrama sequência Implementação 03 CU 08	48
Figura 23 - Diagrama sequência Implementação 03 CU 03	49
Figura 24 - Diagrama sequência Implementação 04 CU 09	49
Figura 25 - Diagrama sequência Implementação 04 CU 04	50

Figura 26 - Navigation Drawer da aplicação.....	51
Figura 27 - Implementação de uma ImageView.....	52
Figura 28 - Declarações no ficheiro string.xml	52
Figura 29 - Declarações no ficheiro colors.xml.....	52
Figura 30 - Declarações dos itens do menu.....	53
Figura 31 - Estrutura de packages	55
Figura 32 – Excerto da classe “Demo.Defs”	56
Figura 33 - Níveis de prioridade dos logs	56
Figura 34 - Layout da implementação 02 "Alarm Button"	58
Figura 35 - Toggle Button "Send Alert"	58
Figura 36 – Ícones apresentados nas ImageViews	59
Figura 37 - Interface "ChatPersistence.java".....	63
Figura 38 - Layout da implementação 03 - "Chat"	64
Figura 39 – Ícone apresentado na imageView	65
Figura 40 – ListView utilizada para representar os contactos.....	66
Figura 41 - Layout janela de conversação	66
Figura 42 - Construtor de contacts.....	69
Figura 43 - Construtor de messages.....	70
Figura 44 - Construtor de TextMessages.....	71
Figura 45 - Método getMessageType() Classe “TextMessage.java”	71
Figura 46 - Construtor de PhotoMessages	72
Figura 47 - Método getMessageType() Classe “PhotoMessage.java”	72
Figura 48 - Enum MessageType.....	73
Figura 49 - Interface "MetricsPersistence.java"	82
Figura 50 - Layout da implementação 04 - “Metrics”	84
Figura 51 – RadioButton usada no layout	85
Figura 52 - Layout das iterações de um teste.....	86

Figura 53 - layout do gráfico das iterações de um teste	87
Figura 54 - Enum TestType	89
Figura 55 - Construtor de Test.....	90
Figura 56 – Construtor de Iteration.....	91
Figura 57 - Construtor Peer	92
Figura 58 - Excerto de código para criação do gráfico	96
Figura 59 – Resultados dos testes unitários	100
Figura 60 - Execução de um teste funcional.....	101

Índice de Tabelas

Tabela 1 – Tarefas por sprint.....	2
Tabela 2 - Comparação de tecnologias offline	6
Tabela 3 - Características de destaque entre JSON e XML.....	7
Tabela 4 - Detalhe dos estados de uma atividade.....	10
Tabela 5 - Prós e contras das atividades	11
Tabela 6 - Detalhe dos estados de um fragmento	12
Tabela 7 - Prós e contras dos fragmentos	13
Tabela 8 – Implementação 02 Caso de uso 01	25
Tabela 9 - Implementação 02 Caso de uso 02	26
Tabela 10 - Implementação 02 Caso de uso 03	26
Tabela 11 - Implementação 02 Caso de uso 04	26
Tabela 12 - Implementação 02 Caso de uso 05	27
Tabela 13 - Implementação 02 Caso de uso 06	27
Tabela 14 - Implementação 02 Caso de uso 07	27
Tabela 15 - Implementação 02 Caso de uso 08	28
Tabela 16 - Implementação 03 Caso de uso 01	29
Tabela 17 - Implementação 03 Caso de uso 02	29
Tabela 18 - Implementação 03 Caso de uso 03	29
Tabela 19 - Implementação 03 Caso de uso 04	30
Tabela 20 - Implementação 03 Caso de uso 05	30
Tabela 21 - Implementação 03 Caso de uso 06	30
Tabela 22 - Implementação 03 Caso de uso 07	30
Tabela 23 - Implementação 03 Caso de uso 08	31
Tabela 24 - Implementação 03 Caso de uso 09	32
Tabela 25 - Implementação 03 Caso de uso 10	32

Tabela 26 - Implementação 04 Caso de uso 01	33
Tabela 27 - Implementação 04 Caso de uso 02	34
Tabela 28 - Implementação 04 Caso de uso 03	34
Tabela 29 - Implementação 04 Caso de uso 04	34
Tabela 30 - Implementação 04 Caso de uso 05	35
Tabela 31 - Implementação 04 Caso de uso 06	35
Tabela 32 - Implementação 04 Caso de uso 07	35
Tabela 33 - Implementação 04 Caso de uso 08	36
Tabela 34 - Implementação 04 Caso de uso 09	36
Tabela 35 - Métodos da interface "StateObserver"	54
Tabela 36 - Métodos da interface "NetworkObserver"	54
Tabela 37 - Métodos da interface "MessageObserver"	55
Tabela 38 – Resumo das classes da implementação 03 - chat	62
Tabela 39 - Resumo das classes da implementação 04 – métricas	81
Tabela 40 -Teste de Aceitação 01 Implementação 02 – alarme.....	102
Tabela 41 - Teste de Aceitação 02 Implementação 03 – chat	103
Tabela 42 - Teste de Aceitação 03 Implementação 04 – métricas	104
Tabela 43 - Teste de Sistema 01 Implementação 02 – alarme	105
Tabela 44 - Teste de Sistema 02 Implementação 03 – chat	105
Tabela 45 - Teste de Sistema 03 Implementação 03 – chat	106
Tabela 46 - Teste de Sistema 04 Implementação 04 – métricas	106
Tabela 47 - Teste de Sistema 05 Implementação 04 – métricas	107
Tabela 48 - Objetivos concluídos.....	109
Tabela 49 - Reuniões com o supervisor da Hype Labs	116
Tabela 50 - Reuniões com o supervisor do ISEP	122

Notação e Glossário

CTO	Chief Technology Officer
Framework	Abstração de software utilizada como funcionalidade genérica a aplicações
IoT	Internet of Things
ISEP	Instituto Superior de Engenharia do Porto
JSON	JavaScript Object Notation
MVC	Model View Controller
SDK	Software Development Kit
UI	User Interface
UML	Unified Modeling Language
UPTEC	Parque de Ciência e Tecnologia da Universidade do Porto
XML	eXtensible Markup Language

1 Introdução

Ao longo deste documento irão ser descritas as tecnologias utilizadas, o problema abordado e o trabalho realizado durante o projeto de estágio, visando a solução final que tem por tema “Aplicação de demonstração da tecnologia Hype”.

Neste primeiro capítulo é apresentado, de forma sucinta e clara, o tema, o trabalho desenvolvido no âmbito do projeto de estágio e a empresa. Inicialmente é feita uma apresentação do projeto, assim como o compromisso assumido no que diz respeito a prazos e planeamentos a cumprir. De seguida é feita uma apresentação da empresa e analisado o benefício que este projeto trará para a mesma. Por fim é relatada a estrutura adotada para o desenvolvimento do presente relatório.

1.1 Apresentação do projeto/estágio

O estágio proposto consiste em criar uma aplicação, para execução em sistemas operativos Android, que permita testar e demostrar a tecnologia Hype em execução.

A tecnologia Hype é uma Framework que permite a comunicação entre dispositivos que se encontrem em proximidade, mesmo sem acesso à internet. Para efeitos de demonstração, foi estabelecido como objetivo a implementação de três aplicações, nomeadamente uma de SOS que despoleta um alarme nos sistemas que se encontram na proximidade, um chat com suporte ao envio de mensagens de texto e imagens e ainda uma aplicação de recolha e apresentação de métricas de desempenho da tecnologia. Todas estas aplicações serão agrupadas numa aplicação “mãe” chamada *Bundle*, e designadas ao longo deste relatório por implementação 01 - configurações (relativo a configurações iniciais), implementação 02 – alarme, respeitante ao alarme, implementação 03 - chat para o chat, e implementação 04 – métricas, para a recolha e análise de métricas.

1.1.1 Planeamento do projeto

Durante o desenvolvimento deste projeto foi utilizada a metodologia ágil designada por *Scrum*, já utilizada pela empresa, pelo que, de acordo com a mesma, o planeamento se encontre estruturado em *sprints*. Apesar de não existir um limite de tempo fixo para cada *sprint*, por norma estes são de 2 a 4 semanas, embora para este projeto se tenha convencionado a que cada *sprint* deveria corresponder a 3 semanas de execução. Na Tabela 1 apresenta-se a distribuição de tarefas por *sprint*, bem como as datas de inicio e conclusão.

Tabela 1 – Tarefas por sprint

	Sprint	Tarefas	Início	Conclusão
Milestone 1	Sprint 1	<ul style="list-style-type: none"> • Adaptação às tecnologias • Configuração do sistema • Planeamento • Relatório 	20 Fev. 2017	10 Mar. 2017
	Sprint 2	<ul style="list-style-type: none"> • Implementação 01 - configurações • Implementação 02 - alarme • Relatório 	13 Mar. 2017	31 Mar. 2017
	Sprint 3	<ul style="list-style-type: none"> • Implementação 03 - chat • Relatório 	03 Mar. 2017	21 Abr. 2017
Milestone 2	Sprint 4	<ul style="list-style-type: none"> • Implementação 03 - chat • Relatório 	24 Abr. 2017	12 Maio 2017
	Sprint 5	<ul style="list-style-type: none"> • Implementação 04 - métricas • Relatório 	15 Maio 2017	02 Jun. 2017
	Sprint 6	<ul style="list-style-type: none"> • Implementação 04 - métricas • Relatório 	05 Jun. 2017	16 Jun. 2017

1.1.2 Reuniões de acompanhamento

Durante o período de estágio, foram realizadas diversas reuniões, quer com o orientador da empresa, quer com o orientador do ISEP. Por norma estabelecida as reuniões na empresa eram realizadas diariamente (desde que oportuno em função dos trabalhos a decorrer), de forma a que o orientador estivesse ocorrente do ponto de situação do desenvolvimento da aplicação. Assim foi possível um melhor acompanhamento da evolução do desenvolvimento do projeto que, quando necessário, proporcionou a discussão de pontos de vista, melhorias e soluções para implementar. Em relação às reuniões com a professora orientadora do ISEP, foi previamente estabelecido o envio semanal de um e-mail com a síntese da semana. Relativamente a reuniões presenciais foram realizadas sempre que uma das partes achou pertinente, sendo que nas mesmas foram discutidos assuntos relativos ao planeamento, datas

de entregas a cumprir, ponto de situação, avaliação e revisão dos progressos a nível de relatório final. No Anexo 1 - Reuniões de acompanhamento, podem ser consultadas com maior detalhe informações relativas a estas reuniões.

1.2 Apresentação da organização

A Hype Labs é uma *Startup*¹ Portuense fundada em 2015, com escritório no Parque da Ciência e Tecnologia da Universidade do Porto (UPTEC), e que foi criada com o objetivo de desenvolver e comercializar a *framework* de comunicação Hype. Esta *framework* recorre a diferentes tecnologias de transporte, como Wi-Fi ou Bluetooth, para a criação de redes ponto-a-ponto entre diferentes dispositivos, independentemente da plataforma, possibilitando aos utilizadores a conectarem-se em qualquer altura e em qualquer lugar. Para tal, a empresa apostou no desenvolvimento de um SDK que permite que os dispositivos comuniquem mesmo quando não há acesso à Internet. Esta tecnologia funciona ligando os dispositivos diretamente uns aos outros, criando uma rede em malha (*mesh*) com os outros dispositivos encontrados na proximidade, permitindo assim aos dispositivos participantes redirecionar tráfego uns através dos outros.



Figura 1 - Equipa e escritório da Hype Labs (Nunes, 2016)

¹ EMPRESA DETENTORA DE UMA IDEIA INOVADORA QUE PROCURA UM MODELO DE NEGÓCIOS REPETÍVEL E ESCALÁVEL, TRABALHANDO EM CONDIÇÕES DE ALGUMA INCERTEZA (MOREIRA, 2016)

1.3 Contributos deste trabalho

O trabalho desenvolvido e analisado ao longo deste relatório vai possibilitar à empresa demostrar a sua tecnologia em execução em tempo real em dispositivos Android, levando assim os clientes a constatarem as potencialidades efetivas da tecnologia concebida, transpondo com isso um maior nível de confiança no produto apresentado.

1.4 Organização do relatório

Este relatório desenvolve-se ao longo de cinco capítulos, sendo que neste primeiro foi feita a abordagem inicial do problema, a apresentação da empresa, do planeamento previsto, e a divisão do tempo, de acordo com as metodologias da empresa.

No capítulo 2 é feita uma contextualização do problema e são apresentadas as tecnologias que podem ser usadas para resolver o problema, assim como um estudo relativo ao estado da arte e à metodologia de trabalho a ser utilizada durante o processo de desenvolvimento.

O capítulo 3 é referente à execução que, com o auxílio de diagramas e outros artefactos, descreve as funcionalidades a serem implementadas.

O capítulo 4 é referente a testes unitários e funcionais. Aqui é apresentada uma análise dos resultados previstos e alcançados.

No capítulo 5 encontram-se as conclusões finais, nomeadamente uma descrição de entraves e problemas encontrados, trabalho futuro, bem como todos os objetivos alcançados durante o período a que este relatório diz respeito.

O presente documento termina com uma secção de referências bibliografias e outra de anexos, onde constam documentos e tabelas relevantes para a compreensão do exposto neste relatório.

2 Contexto

Neste capítulo é feita uma contextualização dos problemas pelo projeto descrito neste documento. Inicialmente é feita uma exposição do problema a solucionar e respetivas áreas de negócio. Um dos focos deste capítulo é referente ao estado da arte, evidenciadas algumas tecnologias utilizadas. No final é apresentada uma contextualização tecnológica e respetivas metodologias de trabalho.

2.1 Enquadramento do problema

A *framework* conta já com suporte para os sistemas operativos Android, iOS, tvOS, macOS e Windows 10. No entanto, neste momento a empresa enfrenta um problema que passa pela incapacidade de demonstração do produto a potenciais e investidores, nomeadamente em conferências, feiras, etc. Com isto surge a aposta de elaboração de uma aplicação para sistemas Android, com o intuito de tornar possível mostrar a tecnologia a funcionar em tempo real. Para tal foram desenvolvidas algumas aplicações: uma de SOS que despoleta um alarme nos sistemas que se encontram na proximidade, um *chat* com suporte ao envio de mensagens de texto e imagens e, finalmente, uma aplicação de recolha e análise de métricas de performance da rede. Estas aplicações foram pensadas com o intuito de demonstrar capacidades diferentes do SDK, como *broadcasting* no caso de um pedido de SOS, ou versatilidade na transferência de diversos tipos de conteúdos com o *chat*. A aplicação relativa à coleta de métricas é mais genérica e serve para demonstrar o desempenho da tecnologia.

2.2 Áreas de negócio

A tecnologia descrita pode ser vantajosa em casos em que o centralismo possa ser prejudicial à segurança dos dados, como bancos, aplicações militares, entre outras, destacando ainda a aplicabilidade a serviços de *IoT*² dado pela componente de interoperabilidade e alcance de rede.

² IoT – INTERNET OF THINGS (INTERNET DAS COISAS) CONSISTE NA CONEXÃO DE VEÍCULOS E EQUIPAMENTOS DO DIA-A-DIA COM RECURSO A SENsoRES E À INTERNET.

2.3 Estado da arte

Para a realização deste projeto de estágio foi feito um estudo sobre o estado da arte de várias tecnologias Android consideradas relevantes para o desenvolvimento. Este estudo tornou possível a tomada de decisões informadas sobre vários aspectos do desenvolvimento, como a escolha das *frameworks* mais indicadas. Os resultados são descritos nas secções que se seguem.

2.3.1 Tecnologias de comunicação offline

Tecnologias de comunicação *offline* são soluções que abdicam de uma ligação de rede de telecomunicações ou ligações à internet e mesmo assim conseguem comunicar. Regra geral estas tecnologias estão munidas de sistemas preparados para se anunciar e procurar os dispositivos vizinhos que possam estar na proximidade. Por forma a ampliar a rede de comunicação criada por estas tecnologias as empresas têm vindo a investir no desenvolvimentos e estabilidade das redes em malha conferindo assim autonomia e maior alcance às mensagens trocadas entre os dispositivos. Atualmente o mercado conta com quatro tecnologias de referência na área das comunicações *offline*, a tecnologia *Hype* (HypeLabs, 2017), o Bridgefy SDK (Bridgefy, 2017), a P2P Kit (P2Pkit, 2017) e a tecnologia *Mesh Kit* desenvolvida pela Open Garden (OpenGarden, 2017). Na Tabela 2 é possível ver uma comparação das diferentes tecnologias referidas. O denominador comum entre todas as tecnologias é o foco na troca de conteúdos através de uma rede criada com recurso aos próprios dispositivos, garantindo comunicações *offline* bidirecionais em ambientes multiplataformas.

Tabela 2 - Comparação de tecnologias offline

Tecnologia	Mesh Network	Broadcast	Plataformas disponíveis	Encriptação
Bridgefy	✓	✓	iOS, Android	✓
Hype	✓	✓	iOS, Android, tvOS, Windows, macOS	✓
OpenGarden (Mesh Kit)	✓	✓	iOS, Android	-
P2P Kit	✓	✓	iOS, Android, macOS	-

2.3.2 Serialização de dados

A serialização de dados é o processo de conversão de um determinado objeto num formato armazenável e transmissível, de tal forma que o processo possa ser invertido para obter o objeto inicial. Estes processos codificam os dados segundo determinados protocolos, de forma a que os dados possam ser interpretados posteriormente.

Associado ao processo de desenvolvimento de aplicações Android surgem dois protocolos de destaque, o JSON e o XML. Para este projeto foi escolhido o protocolo JSON, pelas suas vantagens sobre o XML, como a fácil leitura do código ou o peso mais reduzido em memória, entre outros aspectos, conforme se pode perceber pela Tabela 3 (DevMedia, 2016b).

Tabela 3 - Características de destaque entre JSON e XML

Característica	Melhor Opção
Legível textualmente	Ambos
Simplicidade da análise sintática	JSON
Transporte de pequenos volumes de informação	JSON
Transporte de grandes volumes de informação	XML
Menor utilização de memória	JSON

De acordo com os testes efetuados pela *Infragistics* (Infragistics, 2017) num dispositivo Android, referente ao carregamento e leitura de dois ficheiros (JSON e XML) com sensivelmente o mesmo tamanho (1.3 Mb), apesar de os resultados terem uma variação acentuada, evidenciou-se uma maior rapidez no processo para o formato de dados JSON. Os resultados desse estudo encontram-se resumidos na Figura 2, onde são comparados os tempos de leitura dos dois ficheiros em número de segundos. Como é possível observar, o formato JSON obteve melhores resultados de forma consistente (Betts, 2017).

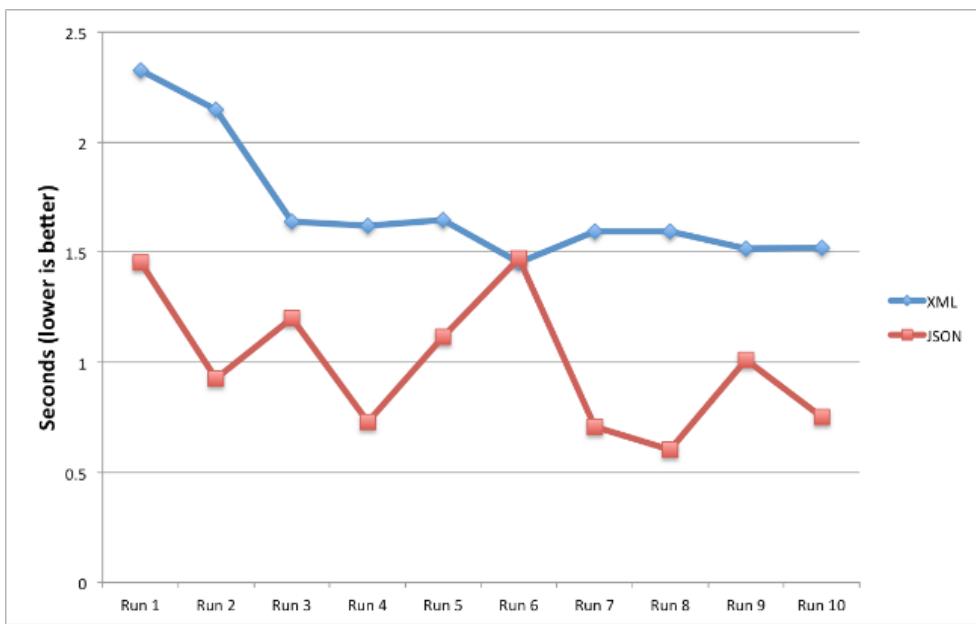


Figura 2 – Comparação de performance ao processar JSON e XML (Betts, 2017)

2.3.3 Android

O Android é um sistema operativo criado pela Google cuja primeira versão comercial, designada Android 1.0 - Astro, data do ano de 2008. Atualmente encontra-se na distribuição da versão 7.1 – Nougat. Segundo a IDC, o sistema operativo da Google detinha, no início de Setembro de 2016, 85.3% de participação de mercado (IDC, 2017).

Relativamente a versões do sistema operativo Android, com dados estatísticos recolhidos pela Google à data de 3 de Abril de 2017, conclui-se que a versão mais utilizada é a versão *Lollipop* com 32.0% da cota de mercado, conforme ilustrado na Figura 3.

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.9%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.9%
4.1.x	Jelly Bean	16	3.5%
4.2.x		17	5.1%
4.3		18	1.5%
4.4	KitKat	19	20.0%
5.0	Lollipop	21	9.0%
5.1		22	23.0%
6.0	Marshmallow	23	31.2%
7.0	Nougat	24	4.5%
7.1		25	0.4%

Figura 3 - Cotas de mercado das versões Android no dia 3 de Abril de 2017 (Google, 2017)

2.3.4 Atividades

Uma atividade consiste num componente que fornece um ecrã para que o utilizador possa desencadear funcionalidades ou interagir com o sistema. A todas as atividades corresponde um ficheiro XML no qual estão contidas as configurações da interface gráfica relativas àquela atividade. Ficheiros de interfaces gráficas encontram-se no pacote “res/layout”. Todos os aplicativos Android tem implicitamente pelo menos uma atividade para ser apresentada ao utilizador quando a aplicação é executada. Após uma atividade estar em execução ela torna-se responsável por ações, como por exemplo criar ou encerrar novas atividades. Neste tópico serão abordadas questões sobre o funcionamento de uma atividade desde que esta é iniciada até chegar o momento em que é destruída. Este processo é designado por “ciclo de vida” (AndroidDevelopers, 2017; Silveira, 2017).

O ciclo de vida de uma atividade representa os estados pelos quais uma atividade passa ao longo da sua execução. Cada ciclo é composto por sete estados, que se relacionam conforme ilustrado na Figura 4.

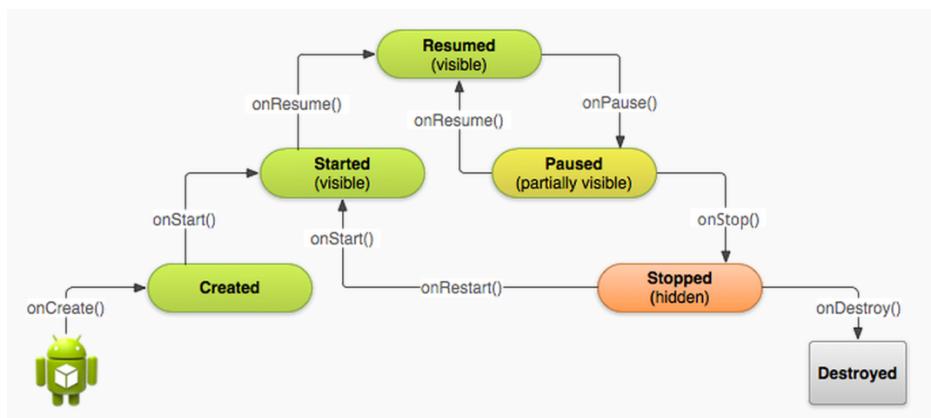


Figura 4 - Ciclo de vida de uma atividade (Reneargento, 2014)

A cada estado corresponde um conjunto de acontecimentos que o fazem transitar para um estado diferente. Esses acontecimentos são despoletados pelo sistema, mediante interação com o utilizador. Na Tabela 4 encontram-se listados todos os métodos de transição de estados das atividades (AndroidDevelopers, 2017; TutorialsPoint, 2017b).

Tabela 4 - Detalhe dos métodos de transição de estados

Método	Descrição
onCreate()	Chamado quando a atividade é criada pela primeira vez
onRestart()	Chamado se a atividade entrou em onStop()
onStart()	Chamado quando a atividade se torna visível para o utilizador Seguido de onResume() se a atividade for para segundo plano ou onStop() se ficar oculta
onResume()	Chamado quando a atividade inicia a interação com o utilizador
onPause()	Chamado quando a atividade atual está a entrar em onPause() porque outra atividade está a entrar em onResume() Seguido de onResume() se a atividade retornar para a frente ou de onStop() se ficar invisível ao utilizador
onStop()	Chamado quando a atividade deixa de estar visível ao utilizador Seguido de onRestart() se a atividade voltar a interagir com o utilizador ou onDestroy() se estiver a encerrar
onDestroy()	Chamado quando a atividade está a ser destruída, ou porque o sistema teve necessidade por uma questão de memória por exemplo, ou porque a atividade está realmente a ser destruída por ação do utilizador ou de outra atividade.

A utilização de atividades está associada a algumas vantagens e desvantagens. Conforme exposto na Tabela 5 percebemos que a simplicidade de construção, a facilidade de gestão do ciclo de vida e de navegação entre as atividades são enaltecidas como principais vantagens da sua utilização. Em contrapartida a sua utilização implica *layouts* mais complexos o que torna o que dificulta o dinamismo necessários em algumas situações.

Tabela 5 - Prós e contras das atividades

Prós	Contras
Simplicidade de construção	<i>Layouts</i> personalização mais complexa
Facilidade de gerir o ciclo de vida	Adaptação a <i>layouts</i> dinâmicos ³
Facilidade de navegar entre atividades	

2.3.5 Fragmentos

Fragmentos são módulos de atividades que servem para representar uma parte dos comportamentos da interface gráfica do utilizador. Esta funcionalidade surgiu na versão 3.0. do Android. Os fragmentos surgem também para auxiliar os programadores a contornar o problema das diversas resoluções e tamanhos dos diferentes ecrãs. Comparando um *Smartphone* e um *Tablet* torna-se possível perceber que as diferenças de tamanho do ecrã podem ser significativas, devido à disparidade da quantidade de espaço livre entre os dois. Os fragmentos permitem gerir os projetos de forma a evitar alterações complexas na modulação das diferentes visualizações de dispositivos diferentes.

Tal como no processo das atividades os fragmentos também são geridos por um processo transitório de estados, ou seja, cada fragmento tem também um ciclo de vida próprio com uma associação direta ao ciclo de vida da atividade que o criou. Essa particularidade leva a que as alterações desse ciclo da atividade tenham efeitos nos fragmentos. Por exemplo, quando uma atividade entra em pausa, todos os fragmentos a ela associados também são colocados em pausa, e quando uma atividade é destruída todos os fragmentos também são destruídos. O ciclo de vida de um fragmento transita por onze fases conforme representado na Figura 5 (Cordeiro, 2017).

³ LAYOUTS DINÂMICOS SÃO AQUELES QUE ADQUIREM DIFERENTES COMPORTAMENTOS EM FUNÇÃO DA RESOLUÇÃO E/OU ORIENTAÇÃO DO ECRÃ DO DISPOSITIVO.

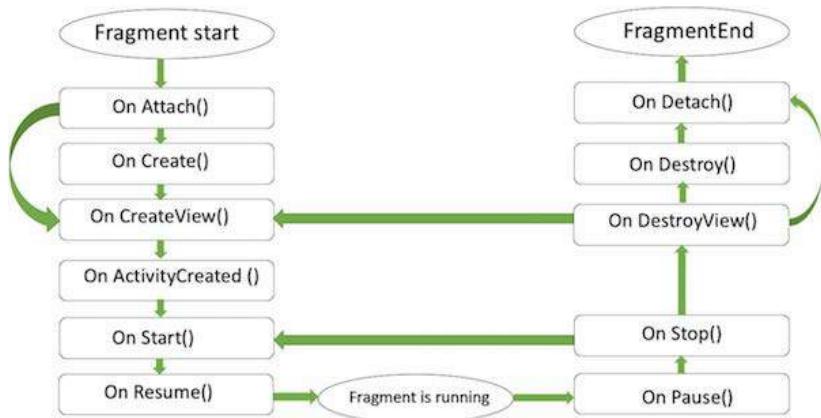


Figura 5 - Ciclo de vida de um fragmento (TutorialsPoint, 2017a)

Na Tabela 6 estão listadas os métodos de transições de estados possíveis de ocorrer no ciclo de vida de um fragmento (Cordeiro, 2017; TutorialsPoint, 2017a).

Tabela 6 - Detalhe dos métodos de transição de estados

Método	Descrição
onAttach()	Chamado para associar o fragmento à atividade
onCreate()	Chamado para criar o fragmento
onCreateView()	Chamado para desenhar a UI do fragmento pela primeira vez
onActivityCreated ()	Chamado quando a atividade do fragmento foi criada e a hierarquia de exibição desse fragmento foi instanciada. Pode ser usado para fazer a inicialização final para recuperar pontos de vista ou restaurar o estado atual
onStart()	Chamado quando o fragmento fica visível, geralmente está associado ao onStart() da atividade que contém o fragmento
onResume ()	Chamado quando o fragmento está visível e inacessível. Geralmente está associado ao onResume() da atividade que contém o fragmento
onPause()	Quando este método é chamado é a primeira indicação que o utilizador está a abandonar o fragmento. Geralmente está associado ao onPause() da atividade que contém o fragmento
onStop()	Chamado quando o fragmento já não é iniciado, geralmente está associado ao onStop() da atividade que contém o fragmento

Estado	Descrição
onDestroyView()	Chamado quando a visualização criada anteriormente pelo <code>onCreateView()</code> foi desassociada do fragmento
onDestroy()	Chamado quando o fragmento não já não voltará novamente a ser utilizado
onDetach()	Chamado para notificar que o fragmento foi desassociado da atividade

Na Tabela 7 encontram-se as principais vantagens e desvantagens da utilização de fragmentos. Os fragmentos denotam pontos fortes na elaboração de *layouts* que requerem dinamismo e reutilização de código. Em contrapartida a utilização estes componentes requerem maiores níveis de conhecimentos, por forma a controlar os estados e transições do seu ciclo de vida assegurando que estes não se confundem com os estados e transições das atividades.

Tabela 7 - Prós e contras dos fragmentos

Prós	Contras
Facilidade de reutilização	Gerenciamento do ciclo de vida mais complexo
Facilidade de construção de <i>layouts</i> dinâmicos ⁴	Comunicação entre fragmentos requer a intervenção de uma atividade
Estável durante as alterações dos <i>layouts</i> dinâmicos	

⁴ LAYOUTS DINÂMICOS SÃO AQUELES QUE ADQUIREM DIFERENTES COMPORTAMENTOS EM FUNÇÃO DA RESOLUÇÃO E/OU ORIENTAÇÃO DO ECRÃ DO DISPOSITIVO

2.3.6 Android Assets

Android Assets é uma ferramenta de grande utilidade para gerar ícones usados nas implementações. Esta ferramenta, desenvolvida por Roman Nurik veio solucionar os problemas que os programadores encontravam com os elementos visuais ao nível dos tamanhos e resoluções. Com o recurso ao *Android Assets* é possível gerar recursos para os projetos Android a partir de elementos existentes nas bibliotecas próprias da ferramenta, ou ainda carregar elementos, como imagens, ícones e outros, para que sejam geradas versões finais otimizadas para os dispositivos, conforme se pode observar pelo ambiente de trabalho da ferramenta presente na Figura 6.

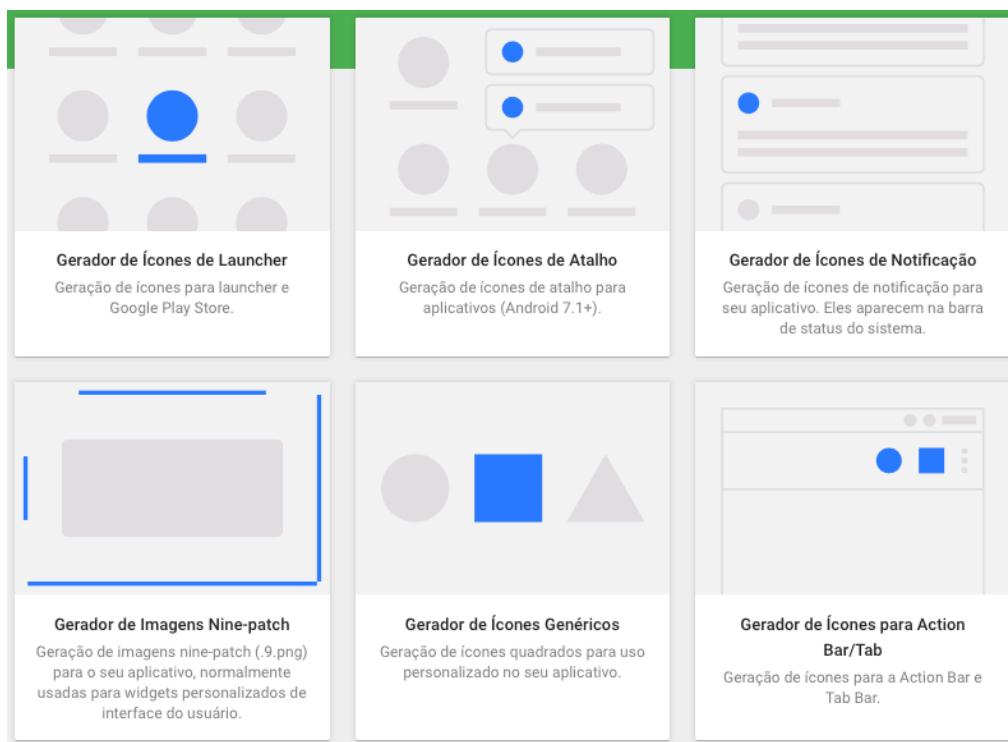


Figura 6 - Área de trabalho do Android Assets

2.4 Contextualização tecnológica

Este projeto foi projetado com recurso à modelação em UML e desenvolvido em Android o que utiliza JAVA como linguagem de programação dos seus produtos. A aplicação criada neste estágio foi desenvolvido com recurso ao IDE oficial, o Android Studio, utilizando ferramentas de gestão como é o caso do GIT e do Scrum. Relativamente à formatação de dados utilizou-se JSON e XML. Estas tecnologias serão descritas sucintamente ao longo desta secção e aprofundadas no estado da arte.

- **Android Studio**

O Android Studio é o ambiente de desenvolvimento integrado (IDE) para *Android*, criado pela Google para auxiliar no desenvolvimento de aplicativos para todos os tipos de *Android* (Google, 2016).

- **GIT**

O *GIT* é um sistema de controlo de versões *open source* distribuído gratuitamente. Foi criado em 2005 por *Linus Torvalds*, para inicialmente auxiliar no desenvolvimento do *kernel* do *Linux*. No entanto, expandiu-se rapidamente a imensos outros projetos de desenvolvimento de *software* (Atlassian, 2017).

- **Scrum**

Scrum, trata-se de uma *framework* que permite agilizar o processo de desenvolvimento de *software* (e não só), gerindo, controlando, reduzindo a complexidade, e aumentando a concentração na execução das determinadas tarefas inerentes ao processo em curso (Scrum, 2017).

- **Java**

O java é uma linguagem de programação multiplataforma orientada a objetos criada em 1995 por uma equipa de programadores na empresa *Sun Microsystems*. Java difere das linguagens de programação convencionais por ser compilada para um *bytecode* que é interpretado por uma máquina virtual. Hoje em dia essa máquina virtual encontra-se presente em inúmeros dispositivos que fazem parte do nosso dia-a-dia, permitindo assim que um programa criado uma única vez possa ser executado infinitamente em qualquer plataforma que suporte esta tecnologia (Oracle, 2016).

- **UML**

O UML (**Unified Modeling Language**), é uma linguagem para especificar, visualizar, construir e documentar modelos de sistemas de *software*, inclusive permite projetar a estrutura e o design das soluções a implementar, partindo de uma análise aos requisitos da aplicação futura (UML, 2017b).

- **XML**

O XML (**eXtensible Markup Language**), é uma recomendação da W3C para gerar linguagens de marcação que tem como principais características a simplicidade e flexibilidade oriunda do SGML. Assume atualmente um papel cada vez mais importante no que diz respeito à troca de dados na Web (W3C, 2016).

- **JSON**

O JSON (**JavaScript Object Notation**), é um formato leve de troca de dados, derivado do JavaScript, que permite o intercâmbio de dados de fácil leitura e escrita para seres humanos, fazendo do JSON a linguagem de intercâmbio ideal (JSON, 2017).

- **Android**

Android, é o sistema operativo da *Google* criado em parceria com a *Open Handset Alliance*, para dispositivos móveis baseado em Linux. Este gera todas as tarefas do dispositivo no qual se encontra instalado apresentando uma interface gráfica para que seja possível interagir com o dispositivo.

Aliado a isto, o Android está munido de um ambiente de desenvolvimento bastante poderoso e flexível que permite aos utilizadores criarem e executarem as suas aplicações nos seus dispositivos (Lecheta, 2013).

2.5 Processo de desenvolvimento e método de trabalho

Existe uma constante pressão associado ao desenvolvimento de projetos e/ou produtos, e à permanente necessidade de entregar rapidamente ao (s) cliente (s) aquilo que eles procuram para satisfazerem as suas necessidades. Devido à necessidade de continuidade de satisfação, é necessário despender do menor tempo e recursos possíveis, o que por vezes leva algumas etapas do processo de desenvolvimento serem quebradas, resultando em taxas de insucesso mais elevadas. Conforme ilustrado na Figura 7, na grande maioria das vezes os projetos das empresas que operam com as tecnologias da informação não são entregues ou são entregues com falhas. É este tipo de situações que as metodologias ágeis pretendem resolver. Estas surgem para introduzir uma forma diferente de construir *software*, recorrendo a planeamentos prévios, otimizando tempo e os recursos e dividindo um dado problema em vários subproblemas de complexidade mais baixa, levando a que as taxas de falha diminuam acentuadamente.

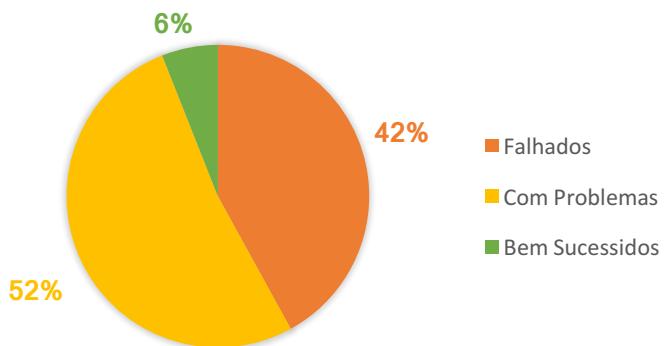


Figura 7 - Resultado de grandes projetos de software (The Standish Group International, Inc., 2014)

No sentido de evitar potenciais desvios e por sua vez consequências desastrosas para as empresas surgem as ferramentas de desenvolvimento iterativo, como é o caso do *Scrum*. *Scrum* é uma metodologia ágil e flexível que tem por objetivo definir um processo iterativo e incremental. Esta pode ser aplicada a qualquer atividade de maior complexidade, como é o caso do desenvolvimento de *software*. Destaca-se das demais metodologias pelo maior enfase atribuído à gestão do projeto, dado que estabelece ações de monitorização de evoluções e *feedback*, com o intuito de identificar e corrigir potenciais impedimentos ou desvios. Projetos

organizados com *Scrum* são divididos em partes designadas por *sprints*, em que cada uma contém uma lista de atividades a executar, chamada *backlog de sprint*. Cada *sprint* dura, por norma, entre duas a quatro semanas conforme se pode observar pela Figura 8. A cada *sprint* corresponde um pequeno incremento de produto, que pode ser testado e validado por gestores e clientes de forma iterativa. Diariamente são efetuadas pequenas reuniões de controlo com o intuito de detetar bloqueios ou desvios à solução ótima. Cada *sprint* inicia com uma reunião onde o *Product Owner* atribui recursos e prioridades para cada tarefa. No processo de *Scrum* existe ainda o *Scrum Master*, que é o responsável por garantir uma correta aplicação da metodologia e suporte à equipa de desenvolvimento, assim como executar funções de controlo, no sentido de impedir que a equipa se comprometa com *sprints* impossíveis de realizar (Scrum, 2017)(Vieira, 2014).

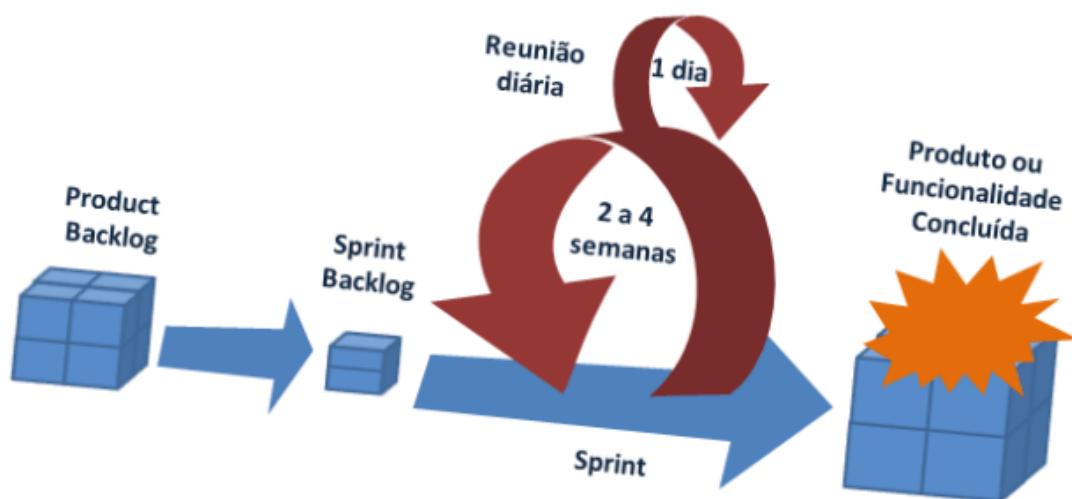


Figura 8 - Processo de Scrum (Vieira, 2014)

3 Descrição técnica

Conforme sugerem as boas práticas de engenharia de software é extremamente recomendado que antes de iniciar uma implementação sejam efetuados estudos prévios das mesmas. Esses estudos assentam sobre o levantamento dos requisitos e planeamento de interações do sistema quer seja com o utilizador quer com outros sistemas. Ao longo deste capítulo será apresentada com mais detalhe a análise das implementações desenvolvidas, incluindo requisitos e modelação. Para tal, utilizam-se diagramas construídos com recurso à linguagem UML. O capítulo termina com uma discussão mais aprofundada sobre a implementação efetuada.

3.1 Análise e Modelação

Sendo a análise da arquitetura de *software* uma etapa essencial no processo de desenvolvimento de soluções, neste capítulo encontram-se quatro subcapítulos, um por cada implementação. Cada um contém uma simples introdução de cada uma das implementações desenvolvidas (alarme, chat e métricas), respetivos modelos de domínio e uma breve descrição de cada classe.

Relativo à modelação e por forma a introduzir previamente os elementos comuns entre todas as implementações, é importante denotar que, em todas as implementações existe a biblioteca “framework” que contempla a seguinte descrição:

- **Framework** – Diz respeito ao produto comercializado pela Hype Labs e que os programadores vão utilizar para enviar dados entre os dispositivos. A biblioteca Framework é autónoma no que respeita à gestão dos equipamentos, envio de dados, notificações aos utilizadores e etc.

3.1.1 Implementação 01 - configurações

O objetivo para esta implementação consistia na preparação do desenvolvimento do projeto, como a criação dos *packages* de cada implementação a desenvolver, a integração da *Framework Hype*, a criação e configuração do repositório *git*, a criação de classes para *logs*, a criação de uma estrutura de persistência de informação e implementação de um sistema de navegação na aplicação.

3.1.2 Implementação 02 - alarme

Para a implementação 02 planeou-se uma aplicação que permitisse ao utilizador despoletar alarmes nos equipamentos que se encontrem na periferia. Esta implementação é composta pelas instâncias da periferia e pelos alarmes que estas criam. Ao longo desta seção é efetuada uma descrição das classes do modelo de domínio presente na Figura 9 onde se representa um alarme que é criado por uma instância que por sua vez contém a *framework* (*Hype*).

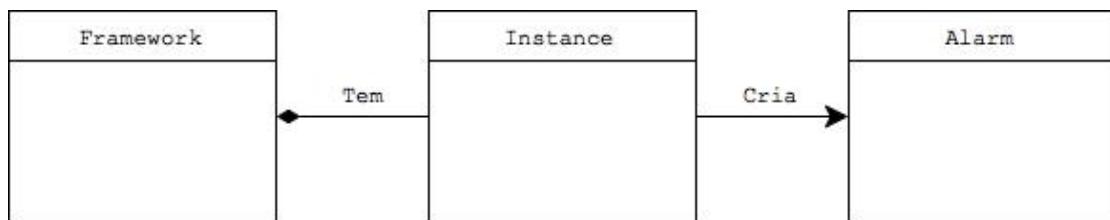


Figura 9 - Modelo de domínio da implementação 02 – alarme

- **Instance** – Interpreta-se como um dispositivo que utiliza a *Framework Hype* e que se encontra em estado que permita iniciar o processo de envio/recepção de dados.
- **Alarm** – Classe que representa o que o utilizador executa ao recorrer a esta implementação, ou seja, é a classe responsável por criar e enviar alarmes para outros dispositivos que estejam na rede, bem como executar alarmes vindos de outra *instance*.

3.1.3 Implementação 03 - chat

Na implementação 03 foi determinado a implementação de um chat que desse suporte ao envio de mensagens de texto e de imagens. Este chat está dividido em duas partes, a primeira referente aos contactos disponíveis na periferia, ou seja, diz respeito aos dispositivos com a *Framework Hype* em execução, a segunda parte, referente à janela de conversação de dois dispositivos. No decorrer desta seção é efetuada a descrição das classes do modelo de domínio apresentado pela Figura 10.

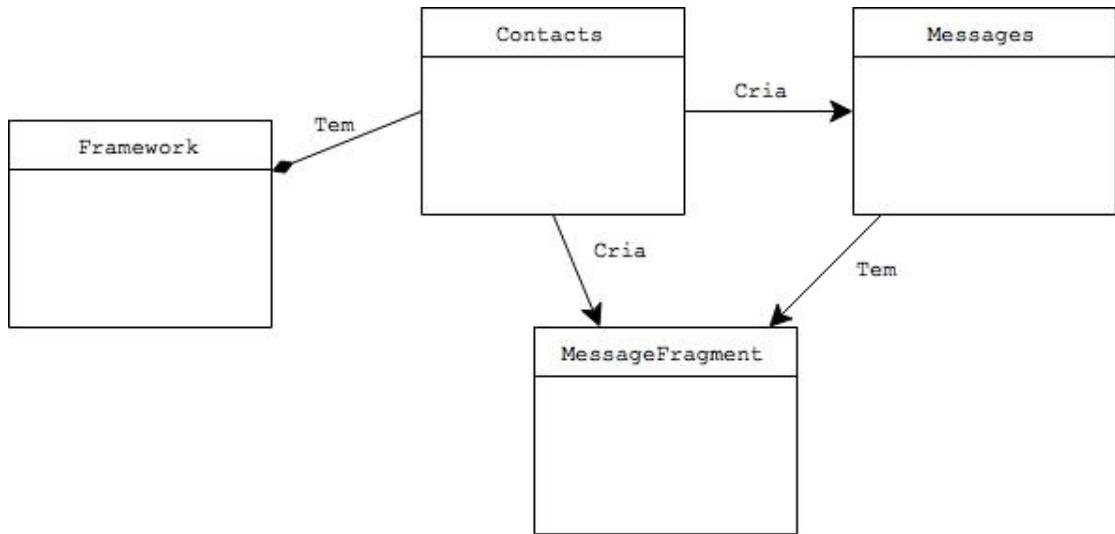


Figura 10 - Modelo de domínio da implementação 03 - chat

- **Contacts** – Esta classe representa os utilizadores *online*, entenda-se por *online* aqueles que se encontram na periferia e com a *framework Hype* em execução, ou seja disponíveis para iniciar uma conversão.
- **Messages** – Classe *Messages* representa os objetos criados pelos *Contacts* que irão ser alvo de troca entre dispositivos.
- **MessageFragment** – Representa para a janela de conversação entre dois contactos. Existirá uma *MessageFragment* para cada par de contactos (emissor e destinatário), onde serão apresentadas as mensagens por eles trocadas.

3.1.4 Implementação 04 - métricas

Para esta implementação foi requerido uma aplicação que permitisse efetuar testes à rede. Para tal designou-se que os testes consistiriam no envio de pacotes de dados pela rede, para a partir daí ser possível efetuar diferentes cálculos de velocidade, sendo possível variar o tamanho de dados enviados com o objetivo de gerar conclusões elucidativas. Na Figura 11 é apresentado o modelo de domínio que apresenta a relação entre as entidades desta implementação. Um *peer* cria testes (*test*) que geram iterações (*iteration*), um teste depois de obter um conjunto de iterações pode, ou não, gerar um gráfico.

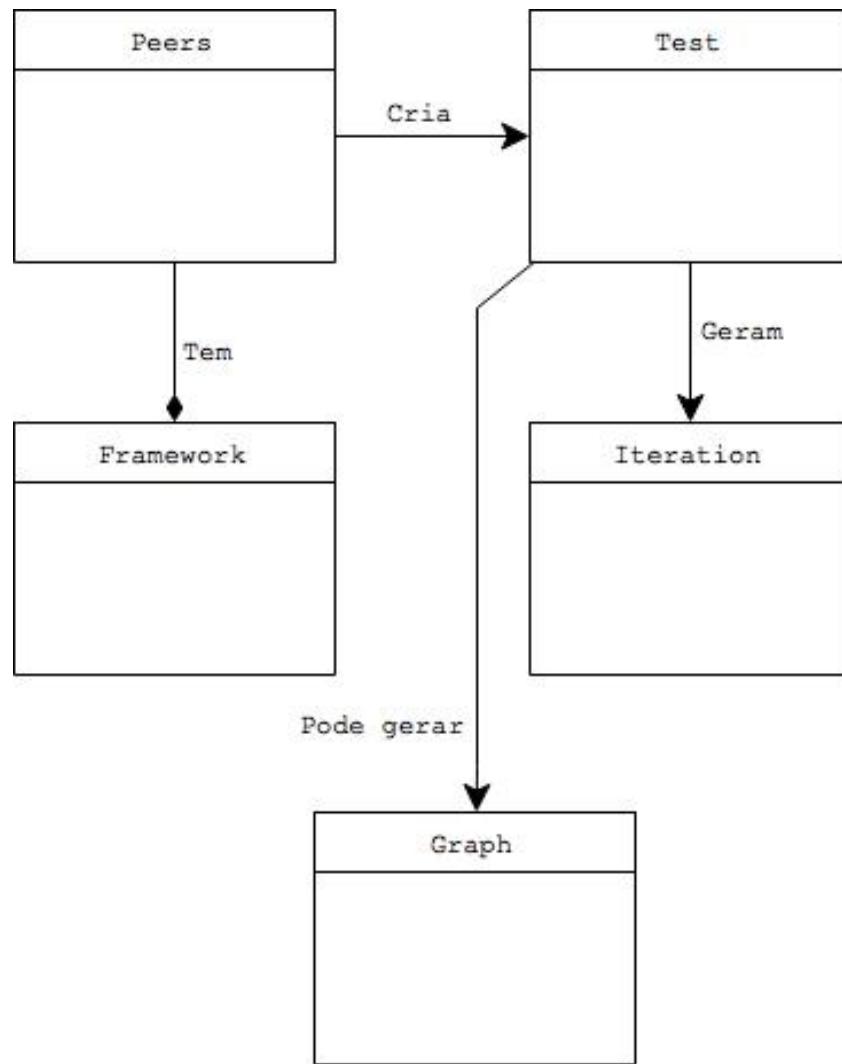


Figura 11 - Modelo de domínio da implementação 04 – métricas

- **Peer** – Esta classe representa os utilizadores *online*, entenda-se por *online* aqueles que se encontram na periferia e com a *framework Hype* em execução, ou seja disponíveis para receber dados que visão testar parâmetros da rede.
- **Test** – Classe referente aos testes criados entre dois *Peers* dos quais se vão obter métricas de rede mediante os parâmetros que o utilizador definiu para o teste.
- **Iteration** - Referente aos resultados dos testes num determinado momento, efetuados entre dois *Peers*.
- **Graph** – Representa o gráfico que pode ser gerado a partir de um conjunto de iterações.

3.2 Engenharia de requisitos

O desenvolvimento de software com qualidade, aliado a prazos rigorosamente definidos, comporta sempre complexidade. No sentido de suavizar a dificuldade inerente a este processo e rentabilizar ao máximo a equipa de desenvolvimento, é fundamental que se efetue um levantamento prévio dos requisitos. Um levantamento de requisitos permite atingir o maior número de objetivos sem comprometer custos e prazos. Como tal, ao longo deste subcapítulo, irão ser identificados os *stakeholders*, os atores, requisitos funcionais e os requisitos não funcionais para este projeto (Ferreira, 2015).

3.2.1 Stakeholders

Os *stakeholders* são as pessoas ou organizações com interesse no projeto que podem ser afetadas direta ou indiretamente de forma positiva ou negativa, não significando com isto que existe por parte das referidas um investimento no projeto/empresa (Research to Action, 2012).

No planeamento deste projeto foram identificados os seguintes *stakeholders*:

- **Programador** – Aquele que utiliza a *framework Hype* no desenvolvimento de um projeto.
- **Gestor de projetos** – Aquele que reconhece as vantagens da utilização da *framework Hype* e quer usufruir das suas potencialidades para o seu projeto.
- **Utilizador final** - Aquele que usa um produto de *software* onde a *Framework Hype* está inserida.
- **Investidor** – Uma pessoa ou entidade com interesse financeiro na empresa

3.2.2 Atores

Atores são utilizadores do sistema que podem ser um elemento humano ou externo e que detêm um papel em relação ao sistema. No caso dos elementos externos considera-se *software*, módulos de *hardware* ou ainda sistemas que enviam ou recebem informações do sistema (UML, 2017a). No caso concreto deste projeto os atores identificados são:

- Utilizador
- Sistema (*Framework Hype*)

Entenda-se por utilizador o ser humano que recorre à aplicação desenvolvida para usufruir das suas potencialidades, por exemplo, o envio de mensagens de texto, imagens, para executar de um pedido de auxílio, ou para obter métricas de desempenho da rede. Por sistema entenda-se a *Framework Hype* que dá suporte estrutural a todo o processo de comunicação existente na aplicação desenvolvida.

3.2.3 Requisitos funcionais

Os requisitos funcionais traduzem-se pelas informações e comportamento que o sistema deve adotar em função de determinadas ações. Essas ações podem ser despoletadas pelos atores envolvidos ou pelo sistema. É prática comum que a descrição destes requisitos seja efetuada com recurso a diagramas de caso de uso nos quais é estabelecido o que o sistema realiza do ponto de vista do utilizador e do sistema. É também dado como tarefa implícita a descrição pormenorizada de cada caso de uso em tabelas (DevMedia, 2016a).

Este capítulo encontra-se dividido por implementações, onde dentro de cada uma constam o respetivo diagrama e as tabelas para cada caso de uso.

3.2.3.1 Implementação 01 - configurações

Conforme exposto anteriormente no tópico 3.1.1, a esta implementação diz respeito tarefas relacionadas à organização do processo de desenvolvimento, pelo que não existem requisitos funcionais considerados.

3.2.3.2 Implementação 02 - alarme

Esta implementação tem como objetivo o desenvolvimento de uma aplicação que permita ao utilizador despoletar alarmes nos equipamentos que se encontrem na periferia.

Os vários casos de uso estão ilustrados na Figura 12.

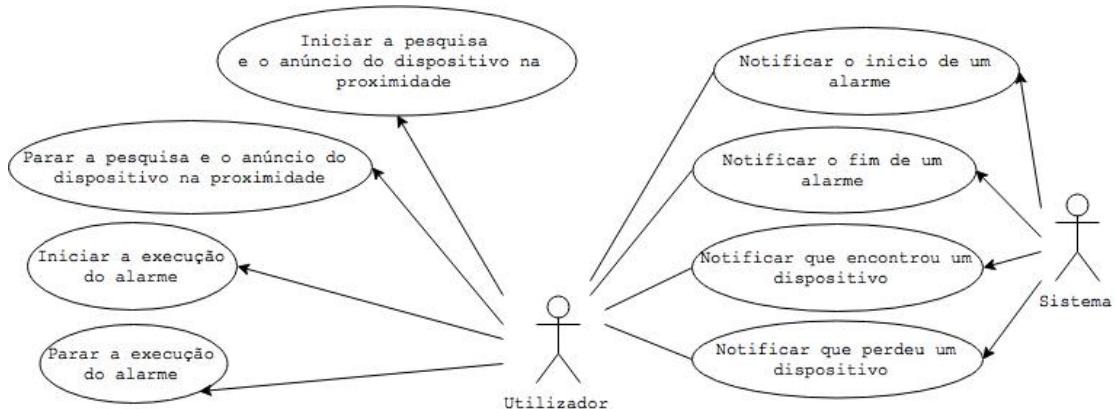


Figura 12 - Diagrama de casos de uso implementação 02 - alarme

No decorrer desta secção encontram-se diversas tabelas onde estão descritos cada caso de uso em maior detalhe. Para cada um são apresentadas as pré-condições necessárias, o ator interveniente e respetivo fluxo de eventos. Devido à natureza descritiva dos casos de uso, são utilizadas tabelas como artefacto, já que permitem um sumário compacto daquilo que se pretende expor.

CU 01 – Iniciar a pesquisa e o anúncio do dispositivo na proximidade

Tabela 8 – Implementação 02 / Caso de uso 01

Objetivo	Iniciar a pesquisa e o anúncio do dispositivo na proximidade
Pré-condições	<ul style="list-style-type: none"> O adaptador de Wi-Fi ou Bluetooth estarem ligados
Atores	<ul style="list-style-type: none"> Utilizador
Fluxo de eventos	<ol style="list-style-type: none"> O utilizador abre a aplicação que por sua vez inicia automaticamente o anúncio do próprio dispositivo, e a procura de dispositivos na proximidade

CU 02 – Parar a pesquisa e o anúncio do dispositivo na proximidade

Tabela 9 - Implementação 02 / Caso de uso 02

Objetivo	Parar a pesquisa e o anúncio do dispositivo na proximidade
Pré-condições	<ul style="list-style-type: none"> • Utilizador ter iniciado o anúncio e pesquisa de dispositivos na proximidade
Atores	<ul style="list-style-type: none"> • Utilizador
Fluxo de eventos	<ol style="list-style-type: none"> 1. O utilizador encerra a aplicação que por sua vez termina automaticamente o anúncio do próprio dispositivo e a procura de dispositivos na proximidade

CU 03 – Iniciar a execução do alarme

Tabela 10 - Implementação 02 / Caso de uso 03

Objetivo	Iniciar a execução do alarme
Pré-condições	<ul style="list-style-type: none"> • Existir pelo menos um dispositivo encontrado na proximidade
Atores	<ul style="list-style-type: none"> • Utilizador
Fluxo de eventos	<ol style="list-style-type: none"> 1. -Aceder ao menu e selecionar “Alarm” 2. -Aguarda que sejam encontrados dispositivos na proximidade 3. -Clicar no botão “Send Alert” para enviar o alarme

CU 04 – Parar a execução do alarme

Tabela 11 - Implementação 02 / Caso de uso 04

Objetivo	Parar a execução do alarme
Pré-condições	<ul style="list-style-type: none"> • Existir uma execução de alarme em curso
Atores	<ul style="list-style-type: none"> • Utilizador
Fluxo de eventos	<ol style="list-style-type: none"> 1. Aceder ao menu e selecionar “Alarm” 2. Clicar no botão “Stop Alert” para cancelar o envio do alarme

CU 05 – Notificar o inicio de um alarme

Tabela 12 - Implementação 02 / Caso de uso 05

Objetivo	Notificar o inicio de um alarme
Pré-condições	<ul style="list-style-type: none"> • Ter encontrado pelo menos um dispositivo na proximidade
Atores	<ul style="list-style-type: none"> • Sistema
Fluxo de eventos	<ol style="list-style-type: none"> 1. Sistema deteta que existiu um pedido para iniciar o alarme 2. Inicia a reprodução do som de alarme

CU 06 – Notificar o fim de um alarme

Tabela 13 - Implementação 02 / Caso de uso 06

Objetivo	Notificar o fim de um alarme
Pré-condições	<ul style="list-style-type: none"> • Existir um alarme em execução
Atores	<ul style="list-style-type: none"> • Sistema
Fluxo de eventos	<ol style="list-style-type: none"> 1. Sistema deteta que existiu um pedido para terminar o alarme 2. Cancela a reprodução do som de alarme

CU 07 – Notificar que encontrou um dispositivo

Tabela 14 - Implementação 02 / Caso de uso 07

Objetivo	Notificar que encontrou um dispositivo
Pré-condições	<ul style="list-style-type: none"> • Ter sido iniciado a execução do alarme
Atores	<ul style="list-style-type: none"> • Sistema
Fluxo de eventos	<ol style="list-style-type: none"> 1. Após detetar que um dispositivo entrou da proximidade o sistema chama um método para notificar que um dispositivo ficou <i>online</i>

CU 08 – Notificar que perdeu um dispositivo

Tabela 15 - Implementação 02 / Caso de uso 08

Objetivo	Notificar que perdeu um dispositivo
Pré-condições	<ul style="list-style-type: none"> Ter encontrado pelo menos um dispositivo na proximidade
Atores	<ul style="list-style-type: none"> Sistema
Fluxo de eventos	<ol style="list-style-type: none"> Após detetar que um dispositivo saiu da proximidade o sistema chama um método para notificar que um dispositivo ficou <i>offline</i>

3.2.3.3 Implementação 03 - chat

Para a implementação 03 o objetivo a cumprir passava pela implementação de um *chat* que desse suporte ao envio de mensagens de texto e de imagens. Assim sendo, após devida análise foi criado o diagrama de casos de uso da Figura 13.

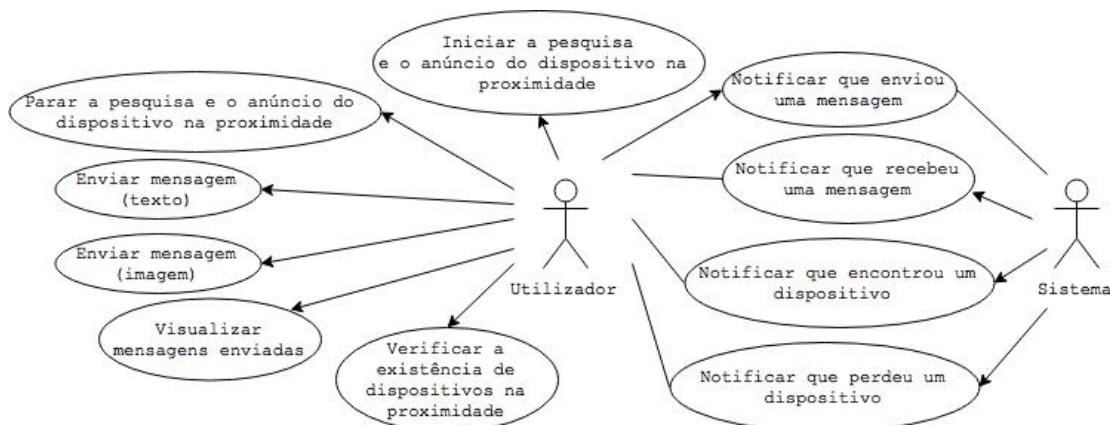


Figura 13 - Diagrama de casos de uso implementação 03 - chat

De maneira a que seja melhor perceptível o que cada caso de uso representa, seguem-se diversas tabelas. Cada tabela representa um caso de uso e contém informações detalhadas acerca do mesmo. Tal como na implementação anterior, para cada tabela são apresentadas as pré-condições necessárias, o ator interveniente e o respetivo fluxo de eventos.

CU 01 – Iniciar a pesquisa e o anúncio do dispositivo na proximidade

Tabela 16 - Implementação 03 / Caso de uso 01

Objetivo	Iniciar a pesquisa e o anúncio do dispositivo na proximidade
Pré-condições	<ul style="list-style-type: none">O adaptador de Wi-Fi ou Bluetooth estarem ligados
Atores	<ul style="list-style-type: none">Utilizador
Fluxo de eventos	<ol style="list-style-type: none">O utilizador abre a aplicação que por sua vez inicia automaticamente o anúncio do próprio dispositivo, e a procura de dispositivos na proximidade

CU 02 – Parar a pesquisa e o anúncio do dispositivo na proximidade

Tabela 17 - Implementação 03 / Caso de uso 02

Objetivo	Parar a pesquisa e o anúncio do dispositivo na proximidade
Pré-condições	<ul style="list-style-type: none">Utilizador ter iniciado o anúncio e pesquisa de dispositivos na proximidade
Atores	<ul style="list-style-type: none">Utilizador
Fluxo de eventos	<ol style="list-style-type: none">O utilizador encerra a aplicação que por sua vez termina automaticamente o anúncio do próprio dispositivo e a procura de dispositivos na proximidade

CU 03 – Enviar mensagem (texto)

Tabela 18 - Implementação 03 / Caso de uso 03

Objetivo	Enviar mensagem (texto)
Pré-condições	<ul style="list-style-type: none">Existir pelo menos um dispositivo encontrado na proximidade (contacto)
Atores	<ul style="list-style-type: none">Utilizador
Fluxo de eventos	<ol style="list-style-type: none">Aceder ao menu e selecionar “Chat”Aguardar que sejam encontrados dispositivos na proximidadeSelecionar o dispositivo para o qual deseja enviar o textoDigitar textoClicar no botão para enviar

CU 04 – Enviar mensagem (imagem)

Tabela 19 - Implementação 03 / Caso de uso 04

Objetivo	Enviar mensagem (imagem)
Pré-condições	<ul style="list-style-type: none"> • Existir pelo menos um dispositivo encontrado na proximidade
Atores	<ul style="list-style-type: none"> • Utilizador
Fluxo de eventos	<ol style="list-style-type: none"> 1. Aceder ao menu e selecionar “Chat” 2. Aguardar que sejam encontrados dispositivos na proximidade (contacto) 3. Selecionar o dispositivo para o qual deseja enviar a imagem 4. Clicar no botão para aceder à galeria de imagens 5. Selecionar a imagem a enviar

CU 05 – Visualizar mensagens enviadas

Tabela 20 - Implementação 03 / Caso de uso 05

Objetivo	Visualizar mensagens enviadas
Pré-condições	<ul style="list-style-type: none"> • Existir pelo menos um dispositivo encontrado na proximidade • Existir mensagens previamente enviadas
Atores	<ul style="list-style-type: none"> • Utilizador
Fluxo de eventos	<ol style="list-style-type: none"> 1. Aceder ao menu e selecionar “Chat” 2. Selecionar o dispositivo para o qual pretende abrir a conversação

CU 06 – Verificar a existência de dispositivos na proximidade

Tabela 21 - Implementação 03 / Caso de uso 06

Objetivo	Verificar a existência de dispositivos na proximidade
Pré-condições	<ul style="list-style-type: none"> • Ter sido iniciada a execução do chat
Atores	<ul style="list-style-type: none"> • Utilizador
Fluxo de eventos	<ol style="list-style-type: none"> 1. Aceder ao menu e selecionar “Chat”

CU 07 – Notificar que enviou uma mensagem

Tabela 22 - Implementação 03 / Caso de uso 07

Objetivo	Notificar que enviou uma mensagem
Pré-condições	<ul style="list-style-type: none"> • Existir pelo menos um dispositivo encontrado na proximidade • Botão de enviar ter sido clicado
Atores	<ul style="list-style-type: none"> • Utilizador
Fluxo de eventos	<ol style="list-style-type: none"> 1. Aceder ao menu e selecionar “Chat” 2. Seleciona um contacto 3. Digitar texto ou selecionar imagem e clicar no botão enviar 4. Sistema adiciona a mensagem à persistência 5. Sistema deteta mensagem enviada e executa um método que notifica utilizador de destino.

CU 08 – Notificar que recebeu uma mensagem

Tabela 23 - Implementação 03 / Caso de uso 08

Objetivo	Notificar que recebeu uma mensagem
Pré-condições	<ul style="list-style-type: none"> • Existir pelo menos um dispositivo encontrado na proximidade • Utilizador ter enviado uma mensagem
Atores	<ul style="list-style-type: none"> • Sistema
Fluxo de eventos	<ol style="list-style-type: none"> 1. Deteta a receção de envio de mensagem por parte de um utilizador 2. Adiciona a mensagem à persistência 3. Apresenta a mensagem na janela de mensagens (se o utilizador de destino possuir a respetiva janela aberta)

CU 09 – Notificar que encontrou um dispositivo

Tabela 24 - Implementação 03 / Caso de uso 09

Objetivo	Notificar que encontrou um dispositivo
Pré-condições	<ul style="list-style-type: none"> • Ter sido iniciado a execução do chat
Atores	<ul style="list-style-type: none"> • Sistema
Fluxo de eventos	<ol style="list-style-type: none"> 1. Após detetar que um dispositivo entrou da proximidade o sistema invoca um método para notificar que um dispositivo ficou <i>online</i>

CU 10 – Notificar que perdeu um dispositivo

Tabela 25 - Implementação 03 / Caso de uso 10

Objetivo	Notificar que perdeu um dispositivo
Pré-condições	<ul style="list-style-type: none"> • Ter encontrado pelo menos um dispositivo na proximidade
Atores	<ul style="list-style-type: none"> • Sistema
Fluxo de eventos	<ol style="list-style-type: none"> 1. Após detetar que um dispositivo saiu da proximidade sistema chama um método para notificar que um dispositivo ficou <i>offline</i>

3.2.3.4 Implementação 04 - métricas

A implementação 04 tinha como objetivo o desenvolvimento de uma aplicação que permitisse ao utilizador obter métricas de desempenho de rede. Para isso o utilizador efetua testes na rede criada entre os dispositivos disponíveis. Para realizar esses testes o utilizador acede à aplicação e define alguns parâmetros essenciais para a realização do teste e após isso é iniciado um processo de troca de dados entre dois pontos da rede. Mediante o exposto foi criado o diagrama de casos de uso da Figura 14.



Figura 14 - Diagrama de casos de uso implementação 04 - métricas

Conforme ocorrido anteriormente, e de maneira a tornar melhor perceptível o que cada caso de uso representa, ao longo desta secção encontram-se diversas tabelas com informações detalhadas acerca de cada um. Em cada tabela são apresentadas as pré-condições necessárias, o ator interveniente e respetivo fluxo de eventos.

CU 01 – Selecionar um *peer* para destino do teste

Tabela 26 - Implementação 04 | Caso de uso 01

Objetivo	Selecionar um <i>peer</i> para destino do teste
Pré-condições	<ul style="list-style-type: none"> Terem sido encontrados dispositivos na proximidade
Atores	<ul style="list-style-type: none"> Utilizador
Fluxo de eventos	<ol style="list-style-type: none"> Aceder ao menu e selecionar “Metrics” Selecionar um dos <i>peers</i> disponíveis

CU 02 – Definir tipo de teste

Tabela 27 - Implementação 04 | Caso de uso 02

Objetivo	Definir tipo de teste
Pré-condições	<ul style="list-style-type: none"> • Não existem pré-condições notáveis
Atores	<ul style="list-style-type: none"> • Utilizador
Fluxo de eventos	<ol style="list-style-type: none"> 1. Aceder ao menu e selecionar “Metrics” 2. Selecionar um tipo, “RTT” ou “Metrics”

CU 03 – Definir tamanho para pacote de informação

Tabela 28 - Implementação 04 | Caso de uso 03

Objetivo	Definir tamanho para pacote de informação
Pré-condições	<ul style="list-style-type: none"> • Não existem pré-condições notáveis
Atores	<ul style="list-style-type: none"> • Utilizador
Fluxo de eventos	<ol style="list-style-type: none"> 1. Aceder ao menu e selecionar “Metrics” 2. Introduzir valor superior a 60

CU 04 – Iniciar teste

Tabela 29 - Implementação 04 | Caso de uso 04

Objetivo	Iniciar teste
Pré-condições	<ul style="list-style-type: none"> • Utilizador ter selecionado um <i>peer</i> • Utilizador ter definido o tamanho para o pacote de informação • Utilizador ter definido um tipo para o teste
Atores	<ul style="list-style-type: none"> • Utilizador
Fluxo de eventos	<ol style="list-style-type: none"> 1. Aceder ao menu e selecionar “Metrics” 2. Clicar no botão “BEGIN TEST”

CU 05 – Iniciar a pesquisa e o anúncio do dispositivo na proximidade

Tabela 30 - Implementação 04 / Caso de uso 05

Objetivo	Iniciar a pesquisa e o anúncio do dispositivo na proximidade
Pré-condições	<ul style="list-style-type: none"> O adaptador Wi-Fi ou Bluetooth estarem ligados
Atores	<ul style="list-style-type: none"> Utilizador
Fluxo de eventos	<ol style="list-style-type: none"> O utilizador abre a aplicação que por sua vez inicia automaticamente o anúncio do próprio dispositivo, e a procura de dispositivos na proximidade

CU 06 – Parar a pesquisa e o anúncio do dispositivo na proximidade

Tabela 31 - Implementação 04 / Caso de uso 06

Objetivo	Parar a pesquisa e o anúncio do dispositivo na proximidade
Pré-condições	<ul style="list-style-type: none"> Utilizador ter iniciado o anúncio e pesquisa de dispositivos na proximidade
Atores	<ul style="list-style-type: none"> Utilizador
Fluxo de eventos	<ol style="list-style-type: none"> O utilizador encerra a aplicação que por sua vez termina automaticamente o anúncio do próprio dispositivo e a procura de dispositivos na proximidade

CU 07 – Notificar que encontrou um dispositivo

Tabela 32 - Implementação 04 / Caso de uso 07

Objetivo	Notificar que encontrou um dispositivo
Pré-condições	<ul style="list-style-type: none"> Ter sido iniciada com sucesso a Framework Hype.
Atores	<ul style="list-style-type: none"> Sistema
Fluxo de eventos	<ol style="list-style-type: none"> Após detetar que um dispositivo entrou da proximidade o sistema chama um método para notificar que um dispositivo ficou <i>online</i>

CU 08 – Notificar que perdeu um dispositivo

Tabela 33 - Implementação 04 | Caso de uso 08

Objetivo	Notificar que perdeu um dispositivo
Pré-condições	<ul style="list-style-type: none"> Ter encontrado pelo menos um dispositivo na proximidade
Atores	<ul style="list-style-type: none"> Sistema
Fluxo de eventos	<ol style="list-style-type: none"> Após detetar que um dispositivo saiu da proximidade o sistema chama um método para notificar que um dispositivo ficou <i>offline</i>

CU 09 – Notificar da percentagem de entrega

Tabela 34 - Implementação 04 | Caso de uso 09

Objetivo	Notificar da percentagem de entrega
Pré-condições	<ul style="list-style-type: none"> Utilizador ter iniciado um teste
Atores	<ul style="list-style-type: none"> Sistema
Fluxo de eventos	<ol style="list-style-type: none"> Após detetar que um pacote de dados, ou parte dele, chegou ao destino o sistema chama um método para notifica o utilizador acerca da percentagem dos dados entregues

3.2.4 Requisitos não funcionais

Os requisitos não funcionais são aqueles que descrevem os atributos de qualidade do sistema desenvolvido. Estes requisitos têm um papel de grande importância durante o desenvolvimento que se podem refletir no momento da escolha das alternativas a seguir.

Estes requisitos servem para acautelar e aproximar o interesse do utilizador da solução desenvolvida. Um sistema que não descarte este tipo de requisitos deve estar preparado para suportar e controlar situações que comprometam o sistema. Por exemplo, no caso de um utilizador mal-intencionado o sistema deve reunir protocolos e abordagens que lhe confirmam diferentes níveis de segurança, dificultando e limitando as intenções dissimuladas do utilizador. Este sistema deve ser sensível a desperdícios por exemplo de memória, bateria, etc.

No que respeita à usabilidade, o sistema deve primar por uma fácil utilização com recursos gráficos harmoniosos para que um utilizador com conhecimentos reduzidos se sinta rapidamente familiarizado e satisfeito com a solução encontrada, excluindo assim o interesse por outras soluções concorrentes de finalidade semelhante (Celestino, 2013) Assim sendo para este sistema foram definidos os seguintes requisitos não funcionais:

- Usabilidade - uma interface gráfica apelativa e de fácil utilização que siga os padrões mais comuns das aplicações mobile em Android.
- Portabilidade - atendendo à forma como o código se encontra organizado, é possível desacoplar uma das demos da aplicação principal com alguma facilidade o que permite que esta seja lançada como uma aplicação principal e independente.
- Interoperabilidade - dada a utilização de JSON como ferramenta de formatação de dados permite as aplicações a troca de dados entre diferentes sistemas, por exemplo, é possível que no futuro as implementações desenvolvidas para Android possam comunicar com sistemas iOS.

3.3 Hierarquia e estrutura de classes

No presente capítulo pretende-se representar a hierarquia e estrutura de classes de cada implementação. Para efetuar essa representação foram utilizados diagramas de classes e posteriormente efetuada uma explicação das classes de cada diagrama. Em conformidade com o exposto anteriormente no tópico 3.1.1, à implementação 01 – configurações, dizem respeito tarefas relacionadas à organização do processo de desenvolvimento, pelo não existe uma estrutura de classes relativa à mesma. Esta seção encontra-se dividida em três subcapítulos referentes à implementação do alarme, *chat* e métricas. Acerca da estrutura de classes das diferentes implementações existe um ponto comum entre todas, a reescrita dos métodos das interfaces da *Framework Hype*. As ações de reescrita de métodos surgem implícitas ao efetuar o “*implements*” das classes “*MessageObserver*”, “*StateObserver*”, e “*NetworkObserver*” por parte da “*MainActivity*”. A estrutura de classes referente à interação das classes da *framework* com a *MainActivity* encontra-se representada na Figura 15 e detalhada ao longo desta secção.

- **MessageObserver** - Responsável pelas notificações relativas à receção, envio ou entrega das mensagens.
- **StateObserver** - Responsável pelas notificações relativas ao estado que a Framework se encontra, podendo transitar entre os estados *Idle*, *Starting*, *Running* ou *Stopping*.
- **NetworkObserver** - Responsável pelas notificações de eventos que acontecem na rede relativas a dispositivos que entram ou saem da periferia, estas notificações acontecem quando um dispositivo é encontrado (*found*) ou quando é perdido (*lost*).
- **MainActivity** - Implementa os *observers* da *Framework Hype* para que desta forma possa ser notificada pela *framework* dos eventos ocorridos e também para que seja possível enviar dados para a rede. Além das interfaces da *framework* esta classe vai implementar outros métodos próprios de cada implementação que serão apresentados ao longo deste capítulo.

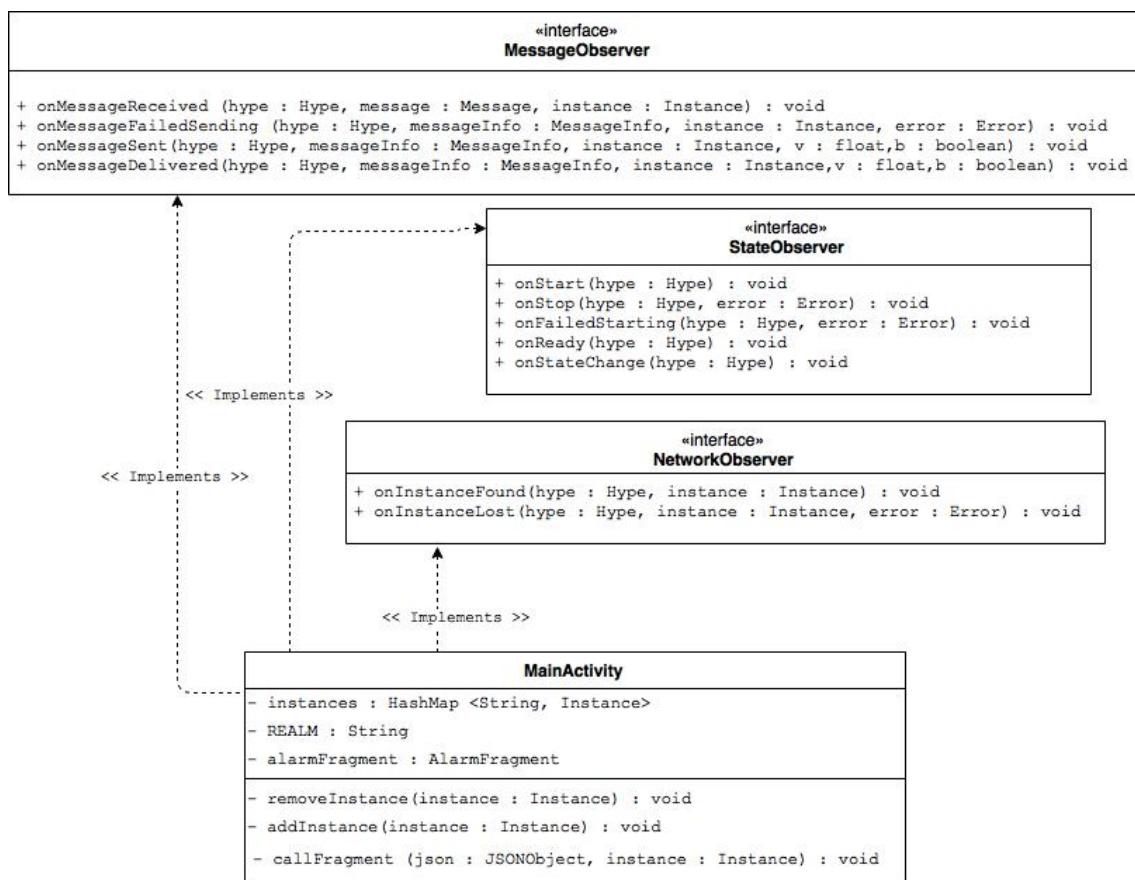


Figura 15 - Diagrama de classes das Interfaces Hype e Main Activity

3.3.1 Implementação 02 - alarme

Neste tópico é possível observar na Figura 16 o diagrama de classes da implementação 02 e respetiva introdução das suas classes.

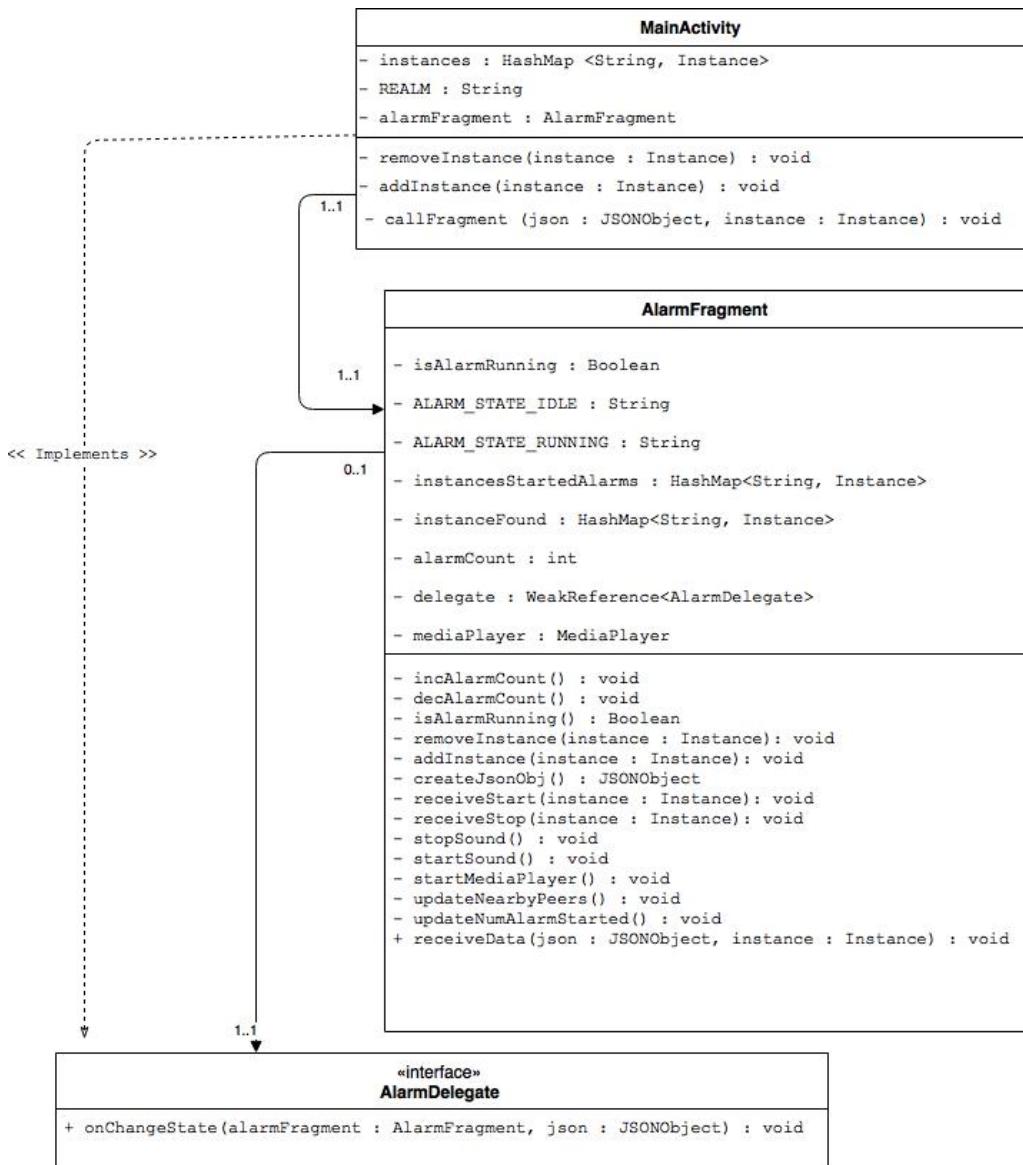


Figura 16 - Diagrama de classes da implementação 02 – alarme

- **MainActivity** – Além de implementar as interfaces da *Framework* (conforme descrito em 3.3) esta classe implementa a interface “*AlarmDelegate*” para ser notificada sempre que um alarme transita de estado.
- **AlarmDelegate** – Permite notificar a *MainActivity* sempre que um alarme permuta de estado, por exemplo, de alarme em execução para alarme parado.
- **AlarmFragment** – Classe responsável pela gestão dos alarmes e da UI. Sempre que recebe uma notificação para início ou fim de alarme, verifica o emissor (instância) da

notificação e em função da existência, ou não, de um pedido de alarme dessa instância executa ou termina a reprodução de um som.

3.3.2 Implementação 03 - chat

Na Figura 17 pode-se observar o diagrama de classes do *model* da implementação 03 – chat.

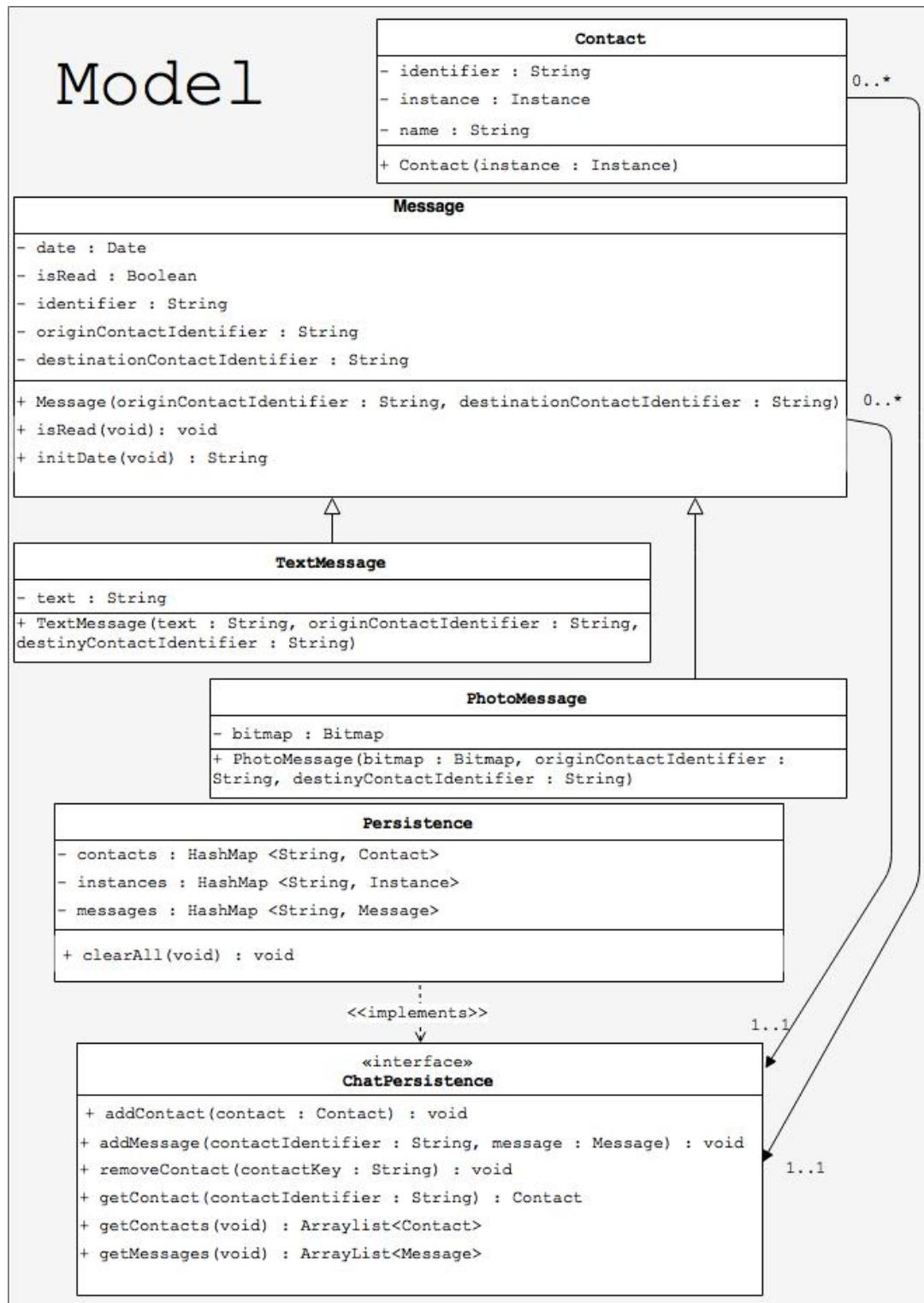


Figura 17 - Diagrama de classes do *model* da implementação 03 - chat

- **Contact** – Representa uma instância⁵ que ficou disponível para comunicar, ou seja, que iniciou a execução da *Framework Hype* e entrou na periferia.
- **Message** – Representa aquilo que é trocado entre dois objetos do tipo “*Contact*”.
- **TextMessage** – Subclasse de “*Message*”, com atributos específicos para representar as mensagens de texto.
- **PhotoMessage** - Subclasse de “*Message*”, com atributos específicos para representar as mensagens que são imagens.
- **Persistence** – Classe que funciona como “base de dados” para armazenar e garantir persistência dos dados desde que a aplicação é executada até que seja encerrada. Esta classe implementa interfaces específicas da implementação de *chat* garantido assim controlo no acesso aos dados por parte das implementações.
- **ChatPersistence** – Interface responsável por permitir o acesso controlado aos dados da persistência relativos à implementação de *chat*, bem como garantir modificações aos mesmos, como por exemplo a adição e remoção de contactos e mensagens.

⁵ INSTÂNCIA, REPRESENTA UM DISPOSITIVO QUE SE CONECTA À REDE.

Na presente secção é possível observar pela Figura 18 o diagrama de classes do *controller* da implementação 03 – chat e ainda uma introdução das suas classes.



Figura 18 - Diagrama de classes do controller da implementação 03 - chat

- **MainActivity** – Implementa os *observers* da *Framework Hype* (conforme descrito em 3.3) para que desta forma possa receber notificações da *framework* e enviar dados para os dispositivos da rede. Além das interfaces da *framework*, esta classe implementa a interface “*ChatManagerDelegate*”.
- **ChatManagerDelegate** – Permite notificar a *MainActivity* quer de pedidos a efetuar à *framework*, quer de solicitações de envio de mensagens a processar.
- **ChatManager** – Funciona como ponte entre as classes “*ContactFragment*” e “*MessageFragment*”. Esta classe gestora implementa os *delegates*⁶ de cada uma por forma a receber as notificações provindas dessas classes, e por sua vez, notificar a “*MainActivity*” das necessidades das classes por ela geridas. É ainda a Classe responsável pela propagação de notificações vindas da “*MainActivity*” para a classe correta em função do tipo de notificação que recebe.
- **MessageDelegate** – Recebe notificações para propagar o envio de mensagens e notifica a classe “*ChatManager*”.
- **MessageFragment** – Classe responsável pela gestão de mensagens e da UI. Acede aos valores inseridos pelo utilizador preparando a mensagem e por sua vez inicia o pedido de propagação da mesma.
- **ContactDelegate** – Recebe notificações para invocar fragmentos, solicita pedidos à *framework* e notifica a classe “*ChatManager*” dos pedidos a propagar.
- **ContactFragment** – Responsável por criar objetos do tipo “*Contact*” mediante notificações de novas instâncias encontradas sendo também sua função criar e atualizar a UI.
- **Utils** – Classe de apoio onde se encontram métodos de funcionamento isolado e específico que dão suporte a necessidades de uma ou mais classes, por exemplo, métodos de codificação e descodificação de *strings*⁷.

⁶ TIPO DE VARIÁVEL QUE PERMITE FAZER REFERENCIA A MÉTODOS QUE SE ENCONTREM EM CLASSES ACIMA, COMO SE UM MÉTODO DE UMA SUBCLASSE ESTIVESSE A INVOCAR MÉTODOS DA SUA SUPERCLASSE. (BALBO, 2016)

⁷ TIPO DE DADOS PRIMITIVO DO JAVA MUITO UTILIZADO PARA REPRESENTAR SEQUÊNCIAS DE CARACTERES ASCII (FRANCISCO, 2003)

3.3.3 Implementação 04 - métricas

Na Figura 19 é apresentado o diagrama de classes referente ao *model* da implementação 04 – métricas.

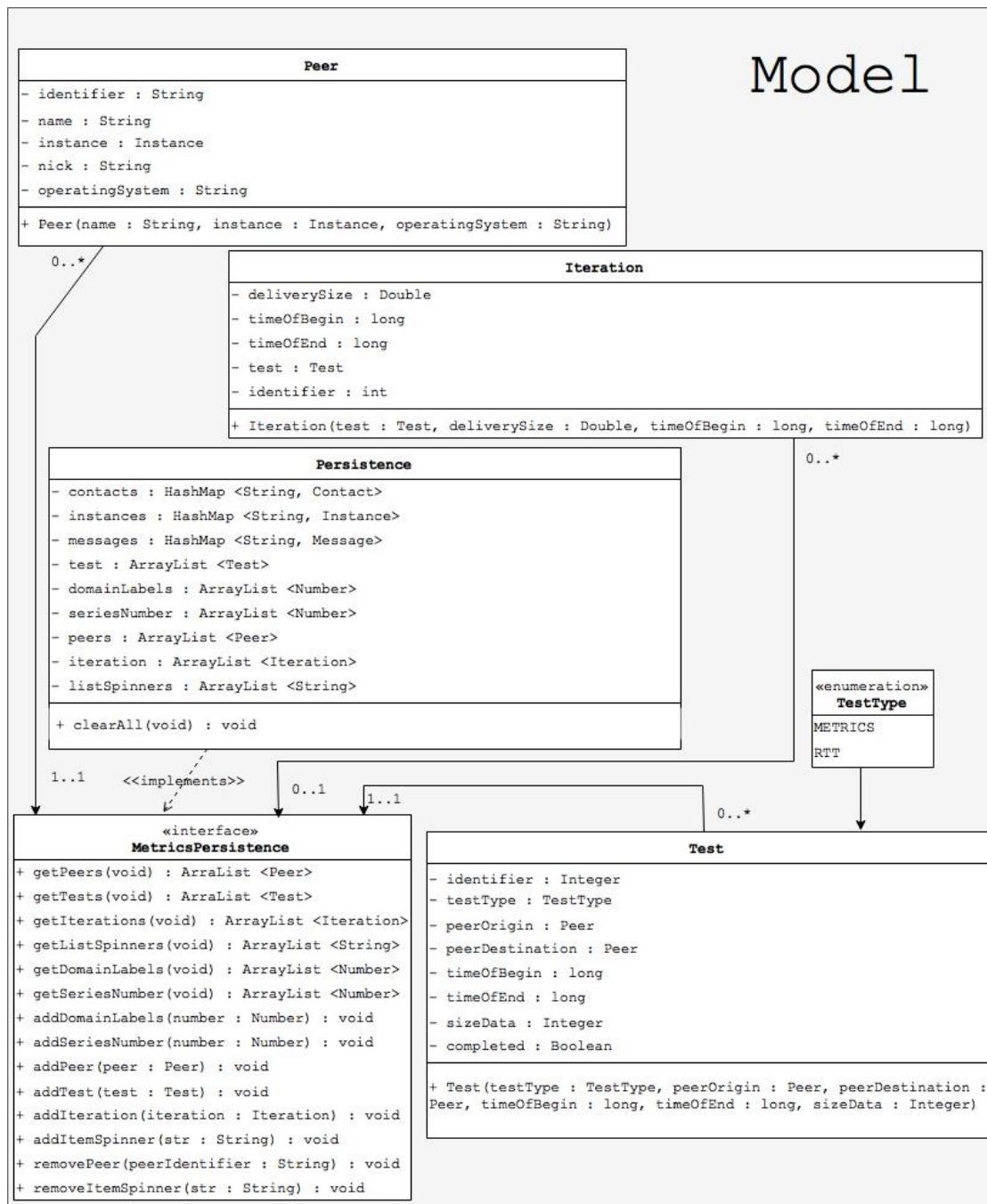


Figura 19 - Diagrama de classes do *model* da implementação 04 – métricas

- **Peer** – Representa uma instância que ficou disponível para comunicar, ou seja, que iniciou a execução da *Framework Hype* ou entrou na periferia.
- **TestType** – É um tipo de enumerado que representa os tipos de testes possíveis de serem criados.
- **Test** – Representa aquilo que é criado pelo utilizador, com o objetivo de obter conclusões acerca do desempenho da rede.
- **Iteration** – Elemento de um teste, representa o envio de um pacote de dados do “cliente” (emissor do teste) para o “servidor” (recetor do teste).
- **Persistence** – Classe que funciona como “base de dados” para armazenar e garantir persistência dos dados desde que a aplicação é executada até que seja encerrada. Esta classe implementa interfaces específicas da implementação de métricas.
- **MetricsPersistence** – Responsável por permitir o acesso controlado aos dados da persistência, bem como garantir modificações aos mesmos, como por exemplo a adição de um “*Test*” e respetivas “*Iteration*”.

De seguida na Figura 20 pode observar-se o diagrama de classes referente ao *controller* da implementação 04 – métricas.

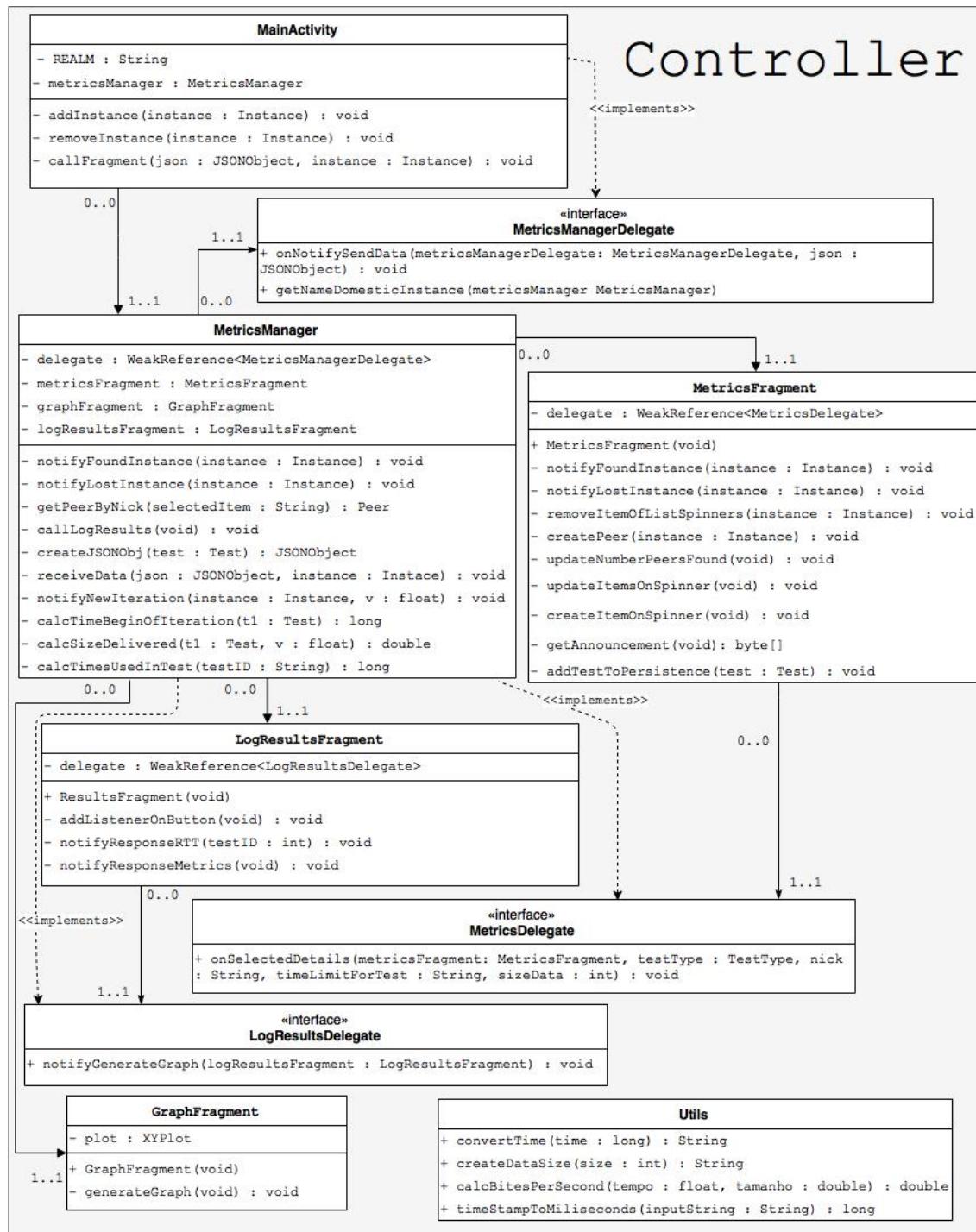


Figura 20 - Diagrama de classes do controller da implementação 04 - métricas

- **MainActivity** – Esta classe implementa os *observers* da *Framework Hype* (conforme descrito em 3.3), implementa ainda a interface “*MetricsManagerDelegate*” para possa ser notificada de envios de dados a processar.

- **MetricsManagerDelegate** – Permite notificar a “*MainActivity*” de testes prontos a processar, e solicitações de pedidos a efetuar à *framework*.
- **MetricsManager** – Esta classe funciona como ponte entre as classes “*MetricsFragment*”, “*LogResultsFragment*” e “*GraphFragment*”. Esta classe implementa os *delegates* das classes “*MetricsDelegate*” e “*LogResultDelegate*” por forma a receber as notificações provindas destas classes. É ainda a classe responsável pela propagação das notificações vindas da “*MainActivity*” para a classe correta em função do tipo de notificação recebida.
- **MetricsDelegate** - Permite notificar a “*MetricsManager*” de testes prontos a executar.
- **MetricsFragment** - Classe responsável pela criação de testes e da UI. Acede aos valores inseridos pelo utilizador na UI criando o teste em função dos mesmos e por sua vez inicia o pedido de propagação do teste.
- **LogResultsDelegate** – Permite notificar a “*MetricsManager*” quando o utilizador pretende a criação de um gráfico.
- **LogResultsFragment** – Responsável pela criação de “*Iterations*” e pela gestão da UI referente aos resultados dos testes.
- **GraphFragment** – Classe responsável pela criação de gráficos. Neste gráfico é representada a velocidade em *Mbits* de cada iteração do teste.
- **Utils** - Classe de apoio onde se encontram métodos de funcionamento isolado e específico que dão suporte a necessidades das classes desta implementação.

3.4 Diagramas de sequência

Os diagramas de sequência são gráficos que procuram representar o fluxo de eventos que ocorrem num processo ao longo do tempo. Estes diagramas identificam os métodos que são invocados entre os atores e as classes envolvidas. Os diagramas deste tipo baseiam-se num caso de um e nas classes presentes dos diagramas de classe. A presente secção encontra-se dividida pelas três implementações, onde em cada uma é apresentado o (s) diagrama (s) do caso de uso mais complexo (Microsoft, 2016).

3.4.1 Implementação 02 – alarme

Na Figura 21 observa-se o diagrama de sequência do caso de uso 03 “Iniciar a execução do alarme” da implementação 02, onde o utilizador acede ao sistema para enviar um alarme aos dispositivos da periferia.

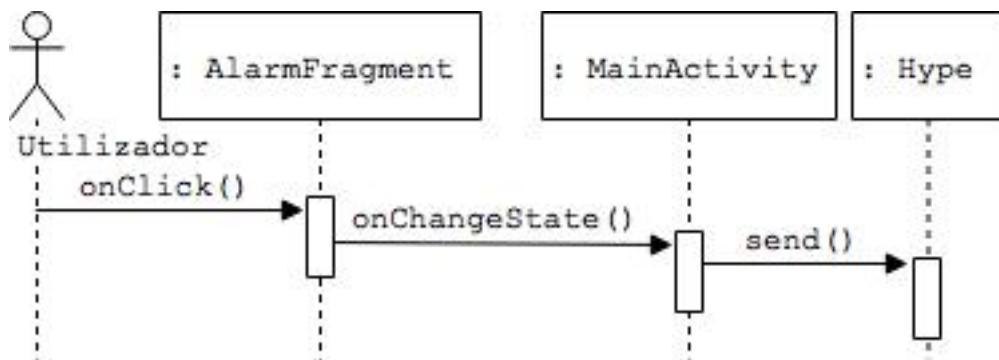


Figura 21 - Diagrama sequência Implementação 02 / CU 03

3.4.2 Implementação 03 – chat

Na Figura 22 observa-se o diagrama de sequência do caso de uso 08 da implementação 03, “Notificar que recebeu uma mensagem”. Neste diagrama é apresentado o fluxo efetuado pelo sistema para notificar o utilizador de que este recebeu uma mensagem.

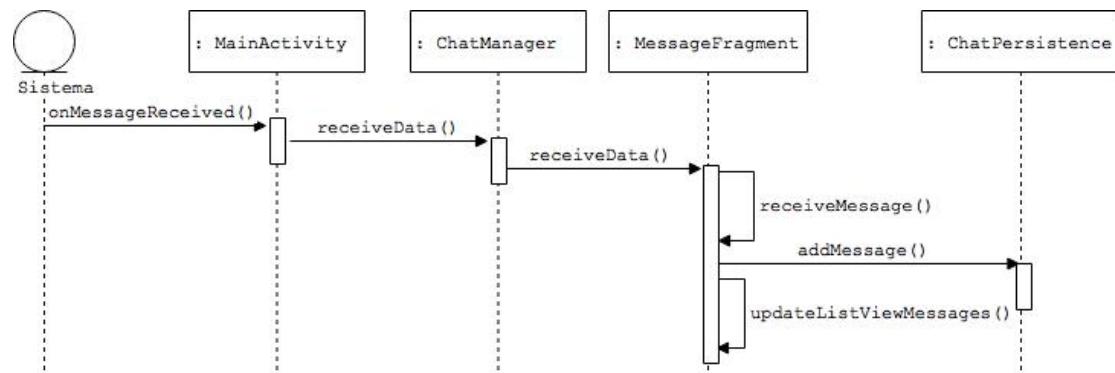


Figura 22 - Diagrama sequência Implementação 03 / CU 08

Na Figura 23 é apresentado o diagrama de sequência do caso de uso 03 da implementação 03, “Enviar mensagem (texto). Neste diagrama é representado o fluxo de envio de uma mensagem por parte do utilizador.

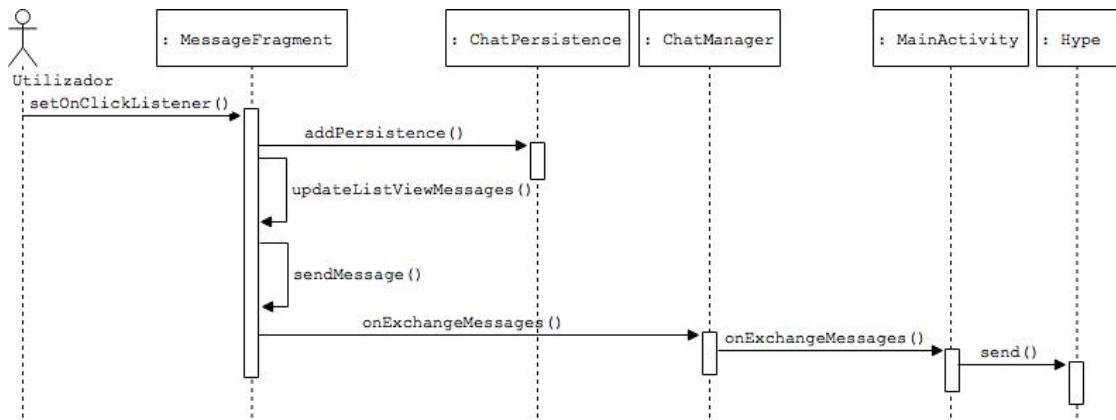


Figura 23 - Diagrama sequência Implementação 03 / CU 03

3.4.3 Implementação 04 – métricas

Na Figura 24 observam-se o diagrama de sequência do caso de uso 09 “Notificar da percentagem de entrega” da implementação 04. Neste diagrama o sistema notifica o utilizador de foi enviada uma percentagem do pacote de dados.

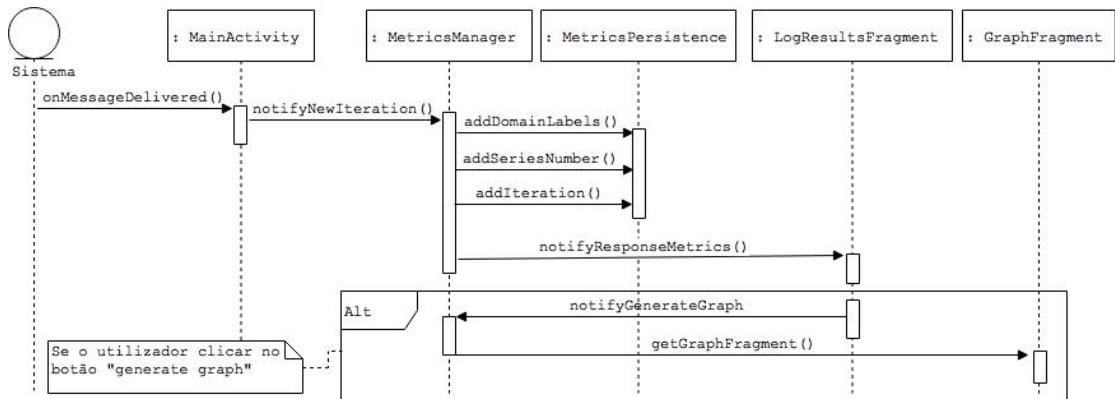


Figura 24 - Diagrama sequência Implementação 04 / CU 09

Na Figura 25 observa-se o diagrama de sequência do caso de uso 04 “Iniciar Teste” da implementação 04 da implementação 04. Neste diagrama é representado o fluxo para a criação de um teste do tipo “Metrics” por parte do utilizador.

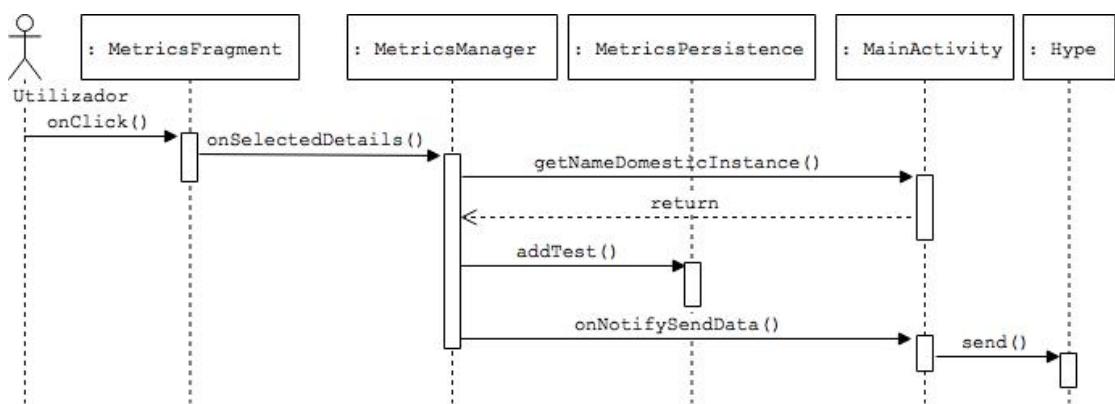


Figura 25 - Diagrama sequência Implementação 04 / CU 04

3.5 Implementação

Esta secção tal como as anteriores também se encontra dividida em quatro implementações. Ao longo de cada implementação será feito um enquadramento do problema, descrito o processo de desenvolvimento e respetivas abordagens para cada uma das implementações. De notar que por questões relativas à melhor otimização da *Framework Hype* e por também neste momento ser a API mais utilizada, esta aplicação foi desenvolvida para a API 21: Android 5.0 (*Lollipop*) do Android.

3.5.1 Implementação 01 – configurações

Conforme descrito anteriormente em 3.1.1, esta implementação é composta por diversas tarefas, que vislumbravam dar suporte ao processo de desenvolvimento futuro. Como tal foram consideradas nesta fase efetuar-se as seguintes tarefas:

- Implementação do menu de navegação
- *Download* e integração da *Framework Hype*
- Estruturação do código no *IDE*⁸
- Criação da classe de *Logs*
- Criação da persistência dos dados

⁸ PROGRAMA DE COMPUTADOR QUE REÚNE CARACTERÍSTICAS E FERRAMENTAS DE APOIO AO DESENVOLVIMENTO DE SOFTWARE

3.5.1.1 Implementação do menu de navegação

Para a criação do menu de navegação convencionou-se a criação de um *Navigation Drawer*. Este componente trata-se um de menu que surge da esquerda para a direita, do lado esquerdo do ecrã que cobre o ecrã da aplicação. Para aceder a este menu basta clicar no ícone (*hamburguer button*) no canto superior esquerdo do ecrã (Developers, 2015).

Este tipo de funcionalidade é nativa do Android Studio pelo que não foi necessário efetuar a sua programação estrutural, no entanto existiu a necessidade de remover e adicionar os conteúdos pretendidos para o menu de acordo com as especificações do projeto. Para o desenvolvimento da aplicação de demonstrações existiu, a necessidade de criar um projeto no IDE de suporte ao desenvolvimento para Android, o Android Studio. Já com o IDE aberto, selecionou-se “*New Project*” em seguida atribuiu-se um nome para a aplicação “*bundle*”, um domínio “*demo.hypelabs.com*” e um local para guardar o projeto. Posto isto foi necessário escolher qual o tipo de equipamento a que a aplicação a desenvolver se destinava, para este caso, foi selecionada apenas a primeira “*Phone and Tablet*” com “*API 21: Android 5.0 (Lollipop)*” para o campo “*Minimum SDK*”. Posto isto foi necessário escolher uma “*Activity*” para adicionar ao projeto, é aqui que entra o “*Navigation Drawer Activity*” como uma das opções disponíveis para adicionar, como tal selecionou-se a atividade referida, fazendo “*next*” e “*finish*” concluindo assim o processo de criação do “*Navigation Drawer*” conforme-se se pode observar pela Figura 26.

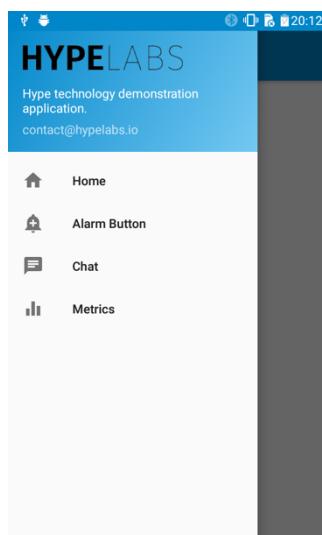


Figura 26 - *Navigation Drawer* da aplicação

Criado o menu foi necessário remover os elementos criados automaticamente pelo IDE e personalizar os restantes elementos conforme o solicitado, como tal foi necessário proceder aos seguintes ajustes.

- Ficheiro “content_main.xml” - adicionada uma “*ImageView*⁹” na qual é apresentado o logotipo da empresa. Na Figura 27 pode-se observar o excerto de código XML que permite a criação da referida *ImageView*.

```
<ImageView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@mipmap/hype_labs_png"  
    app:layout_constraintBottom_toBottomOf="parent"  
    app:layout_constraintLeft_toLeftOf="parent"  
    app:layout_constraintRight_toRightOf="parent"  
    app:layout_constraintTop_toTopOf="parent" />
```

Figura 27 - Implementação de uma *ImageView*

- Ficheiro “string.xml” – conforme se observa na Figura 28, foram adicionadas *strings* que vão ser utilizadas ao longo de todo o projeto, com a vantagem de as deixar concentradas em apenas uma classe, o que permite alterações fáceis e de grande impacto em todo o projeto.

```
<string name="btn_help">SOS</string>  
<string name="toggle_off">Send Alert</string>  
<string name="toggle_on">Stop Alert</string>  
<string name="not_found"><i>NOT FOUND</i></string>  
<string name="my_nick_in_chat">My user:</string>
```

Figura 28 - Declarações no ficheiro string.xml

- Ficheiro “colors.xml” – definidas novas cores para usar ao longo da aplicação conforme se observa na Figura 29.

```
<color name="colorText">#757575</color>  
<color name="colorBlueApp">#0086c9</color>  
<color name="colorWhite">#ffffff</color>
```

Figura 29 - Declarações no ficheiro colors.xml

- Ficheiro “activity_main_drawer.xml” – é neste ficheiro que são definidos os elementos do menu, como tal e conforme apresentado na Figura 30, foram criados quatro itens, um para cada implementação. “*Home*” para voltar ao ecrã inicial,

⁹ COMPONENTE DO ANDROID QUE PERMITE EXIBIR IMAGENS

“Alarm Button” para a aplicação de alarme, “Chat” para a aceder à aplicação de *chat* e “Metrics” para a aplicação de métricas de rede.

```
<group android:checkableBehavior="single">
    <item
        android:id="@+id/nav_home"
        android:icon="@drawable/ic_menu_home"
        android:title="Home" />
    <item
        android:id="@+id/nav_alarm"
        android:icon="@drawable/ic_menu_alarm"
        android:title="Alarm Button" />
    <item
        android:id="@+id/nav_chat"
        android:icon="@drawable/ic_menu_chat"
        android:title="Chat" />
    <item
        android:id="@+id/nav_metrics"
        android:icon="@drawable/ic_menu_metrics"
        android:title="Metrics" />
</group>
```

Figura 30 - Declarações dos itens do menu

3.5.1.2 Download e integração da *Framework Hype*

A *Framework Hype* não é um *software* de distribuição livre para se tornar um utilizador da mesma existe um conjunto de regras a seguir. Em traços gerais é necessário efetuar um registo no *website* da empresa para que possam ser atribuídas permissões que permitam efetuar download da Framework bem como gerar uma credencial de identificação (*REALM*) dentro da rede *Hype*. O *download* consiste num ficheiro no formato “.aar” o qual tem que ser importado para o Android Studio criando-se assim um package próprio para a “Framework Hype”.

Por forma a utilizar a “*Framework Hype*” a classe “*MainActivity*” tem que implementar da *framework* os *observers* “*StateObserver*”, “*NetworkObserver*” e “*MessageObserver*”. Assim sendo existe a necessidade de fazer *override*¹⁰ dos métodos de cada *interface* anteriormente referida e expresso na Tabela 35, Tabela 36 e Tabela 37.

¹⁰ ESCREVER O MESMO MÉTODO, COM OS MESMOS PARÂMETROS, PODE SER VISTO COMO O ANULAR DE UM MÉTODO HERDADO PARA FAZER O SEU ESPECÍFICO.

➤ *StateObserver*

Tabela 35 - Métodos da interface "StateObserver"

Método	Funcionalidade
onStart()	Invocado quando a "Framework Hype" é colocada em execução
onStop()	Invocado quando é terminada a execução da "Framework Hype", isto é, é parado o anúncio do dispositivo na rede e a pesquisa por outros dispositivos.
onFailedStarting()	Invocado quando não é possível iniciar a execução da framework, geralmente porque não existem antenas ligadas (wi-fi, bluetooth)
onReady()	Invocado sempre que a framework voltar a estar disponível para executar. Por exemplo, depois de falhar a iniciar por não ter adaptadores Wi-Fi ou Bluetooth ligados, quando um deles é ligado este método é invocado.
onStateChange()	Invocado quando a framework transita de estado, sendo possível transitar entre "idle", "starting", "running" e "stopping"

➤ *NetworkObserver*

Tabela 36 - Métodos da interface "NetworkObserver"

Método	Funcionalidade
onInstanceFound()	Invocado quando a framework encontra um dispositivo na rede
onInstanceLost()	Invocado quando a framework perde um dispositivo da rede

➤ *MessageObserver*

Tabela 37 - Métodos da interface "MessageObserver"

Método	Funcionalidade
onMessageFailedSending()	Invocado pela <i>framework</i> quando o envio da mensagem falha
onMessageSent()	Invocado pela <i>framework</i> quando a mensagem é enviada (não garante que foi entregue). Indica a percentagem de dados que foram enviados.
onMessageReceived()	Invocado pela <i>framework</i> quando recebe uma mensagem
onMessageDelivered()	Invocado pela <i>framework</i> para notificar que entregou parte ou a totalidade de mensagem. Indica a percentagem de dados que foram entregues.

3.5.1.3 Estruturação do código no IDE¹¹

É característico de um projeto bem estruturado e organizado a divisão das classes por pacotes (*packages*). Os pacotes costumam agrupar classes de funcionalidades similares ou relacionadas. Assim sendo no desenvolvimento deste projeto foram considerados a criação de cinco pacotes, conforme se pode observar pela Figura 31. Sendo que existe um para cada implementação, designados por *Alarm*, *Chat* e *Metrics*, um outro para a persistência dos dados, denominado por *database*, e o ultimo referente as classes de apoio ao projeto, designado por *utils*, o qual contém por exemplo a classe de *logs* e a classe “*Demo.Defs*”.

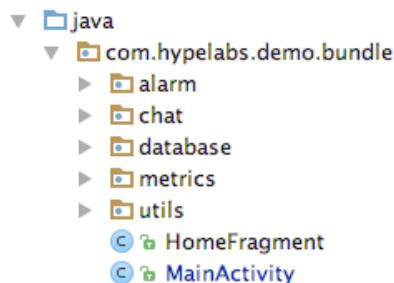


Figura 31 - Estrutura de packages

¹¹ PROGRAMA DE COMPUTADOR QUE REÚNE CARACTERÍSTICAS E FERRAMENTAS DE APOIO AO DESENVOLVIMENTO DE SOFTWARE

3.5.1.4 Classe DemoDefs

A classe “*DemoDefs.java*” funciona de certa forma como classe de apoio. Nesta classe encontram-se várias constantes, conforme excerto da Figura 32, e como tal declaradas como “*final static*” para serem utilizadas ao longo do projeto e impedir a sua modificação.

```
public final static String HOME_TAG = "HOME_FRAG";
public final static String CONTACT_TAG = "CONTACT_FRAG";
public final static String MESSAGE_TAG = "MESSAGE_FRAG";
public final static String ALARM_TAG = "ALARM_FRAG";
public final static String METRIC_TAG = "METRIC_FRAG";
public final static String GRAPH_TAG = "GRAPH_FRAG";
public final static String LOG_RESULTS_TAG = "RESULTS_FRAG";
public final static String OPERATING_SYSTEM_TAG = "OS";
```

Figura 32 – Excerto da classe “*Demo.Defs*”

3.5.1.5 Criação da classe de Logs

A classe “*Log.java*” recorre-se da classe de *logs* nativa do java para permitir o envio de mensagens ao programador durante a execução do programa. Neste caso concreto o envio de mensagens para o terminal do *IDE*. Esta classe contém funcionalidades idênticas as do *System.out.println()* repensadas para conferir uma organização mais profissional separando os *logs* por níveis de importância conforme representado na Figura 33.

```
package com.hypelabs.demo.bundle.utils;

public enum LogLevel
{
    LevelEmergency (0),
    LevelAlert      (1),
    LevelCritical   (2),
    LevelError      (3),
    LevelWarning    (4),
    LevelNotice     (5),
    LevelInfo       (6),
    LevelDebug      (7);

    private int value;
    LogLevel(int value) {
        this.value = value;
    }

    public int getValue() {
        return value;
    }
}
```

Figura 33 - Níveis de prioridade dos logs

3.5.1.6 Criação da persistência dos dados

A persistência de *dados* consiste no armazenamento confiável, coerente e consistente das informações de um sistema para que estas existam e estejam acessíveis a qualquer momento durante a execução da aplicação. Geralmente ao falar de persistência de dados, associa-se o termo à ligação a uma Base de Dados. No entanto, para evitar as ligações às redes de dados, visto que *Framework Hype* tem como forte característica a comunicação *offline*, convencionou-se a implementação de uma persistência volátil comum a todas as implementações. Essa convenção garante a manipulação e funcionamento em pleno das diferentes implementações armazenando os dados gerados até que a atividade principal seja destruída. Ao longo de todas as implementações o acesso à persistência será feito com recurso aos métodos *getAlarmPersistence()*, *getChatPersistence()* e *getMetricsPersistence()* conforme a implementação de onde está a ser acedida.

3.5.2 Implementação 02 - alarme

A implementação 02 refere-se ao desenvolvimento de uma aplicação de alarme. Para esta aplicação é pretendido que o utilizador aceda ao sistema (aplicação *bundle*) e efetue uma solicitação de ajuda aos dispositivos na periferia. Do ponto de vista de utilização final esta implementação poderá ter utilidade em algumas atividades do dia-a-dia, por exemplo um grupo de amigos que parte para um trilho na natureza e que se perde do restante grupo, sabendo de antemão que todos têm a aplicação em execução a qualquer momento podem ser trocados alarmes entre eles. Para que esta funcionalidade seja testada é necessário que o dispositivo tenha a aplicação aberta, que é sinónimo de execução da *Framework Hype*, com isto o utilizador encontra-se num estado de escuta e análise de ações na rede e aquando da receção de um pedido de alarme de outra instância será acionado um alarme sonoro e são efetuadas alterações visuais.

Para o desenvolvimento desta aplicação foram criadas duas classes dentro do pacote “*Alarm*”, a *AlarmDelegate* e a *AlarmFragment*. Conforme referido anteriormente em 3.5.1.2, após integrar a “*Framework Hype*” é necessário proceder à implementação dos métodos *override*. O método *onInstanceFound* invoca o método “*addInstance*” que adiciona a instância encontrada à persistência e de seguida atualiza o número de instâncias encontradas na UI.

A nível de persistência de dados foram criados dois *HashMap* um para coletar as instâncias encontradas pela *framework* e o outro para guardar as instâncias que iniciaram pedidos de execução de alarmes.

3.5.2.1 Interface gráfica

Considerando que o utilizador acede ao menu e seleciona a implementação “*Alarm*” é-lhe apresentado no ecrã o *layout* que se pode observar pela Figura 34.

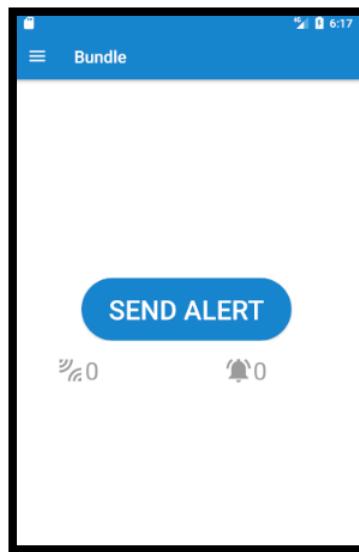


Figura 34 - Layout da implementação 02 "Alarm Button"

Ao invocar do método “*onCreateView*” é colocado em primeiro plano o *layout* do *fragment* referente à implementação selecionada, que é composto pelos seguintes componentes:

- Um *button* (“SEND ALERT”),
- Duas *textView* para os contadores
- Duas *imageView* para apresentação de ícones.

De notar que o *layout* desta implementação encontra-se definido num ficheiro *.xml* designado por “*fragment_alarm.xml*”. Este ficheiro é responsável pela composição da UI da implementação. Para os elementos da UI que permitem dinamismo é criados os respetivos métodos *get[...]()* por forma a permitir o acesso aos componentes abaixo descritos.

- Botão “SEND ALERT”

Trata-se de um *toggle button* nativo do Android cuja sua vantagem consiste na transição entre dois estados. O botão observado na Figura 35 contém dois estados (*isChecked == false* / *isChecked == true*), como se tratasse de dois botões agrupados num só.



Figura 35 - Toggle Button "Send Alert"

➤ **imageView**

Componente nativo do Android apropriado para representar imagens. Neste caso, considerando a observação da Figura 36, à esquerda observa-se o ícone relativo ao número de instâncias encontradas na periferia, e à direita o ícone do número de alarmes em execução no dispositivo.



Figura 36 – Ícones apresentados nas ImageViews

➤ **TextViews**

Permitem a apresentação de textos compostos por caracteres alfanuméricicos e especiais. Aqui são apresentados os contadores de instâncias encontradas na periferia (número zero à esquerda) e o contador do número de alarmes recebidos e em execução no dispositivo (número zero à direita).

3.5.2.2 Conteúdo de classes

Ao longo desta secção é efetuada a descrição das classes referentes à implementação 02 – alarme. Mediante esta descrição pretendesse entender o funcionamento de cada uma bem como o conhecer os seus métodos. Dentro de cada classe são apresentados os respetivos métodos.

AlarmDelegate.java

A interface¹² “AlarmDelegate.java” contém a assinatura do método “onChangeState”. Este recebe como parâmetros um “AlarmFragment” e um objeto JSON. Esta classe é implementada pela “MainActivity” que por isso vê o seu método aqui a ser reescrito. Assim sendo a criação desta classe permite que a classe “AlarmFragment” possa notificar a “MainActivity” que existiu uma alteração de estado do alarme. Este tipo de classe é invocada pela declaração de um objeto do tipo “weak reference”, isto é, uma referência fraca para um objeto. Supondo que um objeto cria um segundo objeto ficando estes ligados por uma “weak reference”, quando o primeiro objeto é libertado da memória, é quebrada a ligação entre eles, o que leva

¹² É UM RECURSO QUE PERMITE QUE UM DETERMINADO GRUPO DE CLASSES POSSA TER MÉTODOS OU PROPRIEDADES EM COMUM DENTRO DE UM DETERMINADO CONTEXTO, CONTUDO OS MÉTODOS PODEM SER IMPLEMENTADOS EM CADA CLASSE DE UMA MANEIRA DIFERENTE. (ALBERTTANURE, 2012)

a que o segundo objeto também seja libertado. Referências fracas permitem aproveitar as potencialidades do “*garbage collector*” e assim o programador não tem que se preocupar com a gestão dos objetos em memória.

AlarmFragment.java

Ao longo deste subcapítulo é realizada uma abordagem à classe “*AlarmFragment*” que é a responsável pelo *controller* de toda a implementação. A par das tarefas executadas no *onCreateView* apresentadas anteriormente, são também implementados outros métodos imprescindíveis para o pleno funcionamento da implementação, os métodos referidos encontram-se descritos nos tópicos abaixo.

- `getBtnAlarm().setOnClickListener(this)`

A invocação deste método representa a definição de eventos associados ao clicar do botão. Para permitir este tipo de operações é mandatório que a classe faça o *implements* do *View.OnClickListener*, e como tal é necessário efetuar o *override* do método *onClick(View view)*. É neste método que são programadas as ações para o botão, neste caso, essas ações permitem verificar o número de instâncias existentes na periferia e efetuar pedidos de execução ou cancelamentos dos pedidos já existentes. Caso o numero de instâncias seja superior a zero, é verificado o estado atual do alarme com recurso ao método “*isChecked()*”, isto permite que a ação do botão invoque o delegate que notifica a classe “*MainActivity*” de que existiu uma alteração ao estado atual do alarme dos equipamentos encontrados na periferia levando assim à execução ou cancelamento do(s) alarme(s).

- `updateNearbyPeers()`

Método responsável por atualizações a nível de UI, neste caso acede à persistência e verifica qual o tamanho do *HashMap* de instâncias e escreve o seu tamanho na referente *textView*.

- `updateNumAlarmStarted()`

Método responsável por atualizações a nível de UI. Acede à persistência e verifica qual o tamanho do *HashMap* de instâncias que solicitaram a execução de alarmes e escreve o seu tamanho na respetiva *textView*.

➤ `startMediaPlayer()`

Executado para que seja possível colocar um som em reprodução aquando da solicitação de início de alarme de uma instância. Para isso necessário criar um objeto do tipo `MediaPlayer`. Como tal foi efetuado o *implements* na classe do “`MediaPlayer.OnPreparedListener`” e efetuado o respetivo *override* dos método “`onPrepared(MediaPlayer mp)`” que recebe como parâmetro um `MediaPlayer` e executa o método `startMediaPlayer()` responsável por colocar o `MediaPlayer` em prontidão para executar sons.

➤ `startSound()`

Método que invoca a reprodução do som com recurso ao método `startMediaPlayer()`.

➤ `stopSound()`

Método que termina a reprodução do som e coloca o `MediaPlayer` no seu estado inicial (preparado para nova execução).

➤ `receiveStop(Instance instance)`

Método invocado quando uma instância termina o pedido de execução de alarme ou quando uma instância fica *offline* (saí da periferia ou termina a execução da aplicação). Este método é responsável por atualizações de UI, por terminar a execução de um alarme, por remover a instância perdida da persistência de instâncias que colocaram alarmes em execução e por atualizar os contadores de instâncias na periferia e de alarmes em execução.

➤ `receiveStart(Instance instance)`

Método invocado quando uma instância solicita um pedido de execução de alarme.

Este método coloca a instância que solicitou a execução de um alarme na persistência e atualiza o contador de alarmes em execução.

➤ `createJsonObj()`

Método do protocolo de comunicação invocado para enviar mensagens com notificações de alterações de estado do alarme para os outros dispositivos da rede. Este método retorna um objeto `JSON` criado a partir de um `HashMap<String, String>` por ele criado tendo como chave a `String “ALARM_SOUND_KEY”` definida na classe `Demo.Defs` e como conteúdo da chave o estado em que deve colocar o alarme do dispositivo que recebeu a mensagem.

- `receiveData(JSON Object json, Instance instance)`

Método do protocolo de comunicação invocado quando a implementação recebe uma mensagem. Este método recebe um objeto JSON que representa o conteúdo da mensagem e a instância que iniciou o envio da mensagem pela *framework*. O conteúdo da mensagem é acedido com recurso à mesma chave usada na criação do objeto *Json* (*ALARM_SOUND_KEY*) e analisada a informação nela contida, em função disso será ativada ou desativada a reprodução do alarme, ou seja, verifica se a instância que enviou a mensagem, já efetuou um pedido de execução de alarme, em caso afirmativo termina a sua reprodução, em caso negativo inicia-a.

3.5.3 Implementação 03 - *chat*

Na implementação 03 é pretendido que o utilizador aceda ao sistema (aplicação *bundle*) para enviar mensagens aos dispositivos na periferia. As mensagens podem ser compostas por texto (caracteres alfanuméricos) ou imagens. Do ponto de vista de utilização final esta implementação poderá ser útil num contexto de festivais ou espetáculos nos quais as pessoas podem comunicar umas com as outras. No desenvolvimento desta aplicação foram criadas várias classes dentro do pacote “*Chat*” conforme exposto na Tabela 38, onde estas são expostas resumidamente e por sua vez explicadas com maior detalhe ao longo deste capítulo.

Tabela 38 – Resumo das classes da implementação 03 - chat

Classe	Síntese
Utils	Classe com métodos de apoio ao package da implementação 03
Contact	Classe <i>model</i> referente aos objetos “ <i>contact</i> ”
Message	Classe <i>model</i> referente aos objetos “ <i>message</i> ”
TextMessage	Subclasse de <i>Message</i> , representa o tipo de mensagem referente a texto (caracteres alfa números).
PhotoMessage	Subclasse de <i>Message</i> , representa o tipo de mensagem referente a imagens
ContactDelegate	Interface com funções de <i>delegate</i> para notificar a classe “ <i>ChatManager</i> ”

Classe	Síntese
ContactFragment	Classe <i>controller</i> dos contact's
MessageDelegate	Interface com funções de <i>delegate</i> para notificar a classe “ChatManager”
MessageFragmant	Classe <i>controller</i> das messages
ChatManager	Classe <i>controller</i> que faz a gestão para/entre as classes “ContactFragment” e “MessageFragmant”
ChatDelegate	Interface com funções de <i>delegate</i> para notificar a classe “MainActivity”

3.5.3.1 Persistência

Relativamente à persistência de dados, conforme se pode observar pela Figura 37, criou-se uma *interface* (ChatPersistence.java) que contém os métodos relativos a esta implementação, métodos esses que são reescritos na classe Persistence.java

```
public interface ChatPersistence {
    ArrayList<Contact> getContacts();

    ArrayList<Message> getMessages(String contactIdentifier);

    void addMessage(String contactIdentifier, Message message);

    void addContact(Contact contact);

    void removeContact(String contactKey);

}
```

Figura 37 - Interface "ChatPersistence.java"

A nível da classe “Persistence.java” foram reescritos os métodos da interface e criados dois contentores de objetos. Um *ArrayList* para armazenar os contactos (um contacto representa uma instância “online”), e um *HashMap* para coletar as mensagens de cada contacto tendo este como chave um identificador do contacto e como conteúdo da key um *ArrayList* de mensagens associada aquela key. Os métodos da interface *ChatPersistence* foram reescritos de forma a tomar os comportamentos descritos nos tópicos abaixo.

➤ `getContacts()`

Método que permite o acesso ao *ArrayList* de contactos.

➤ `getMessages(String contactIdentifier)`

Método que permite o acesso ao *ArrayList* de mensagens, acedendo ao *HashMap* de mensagens. Procura no *HashMap* de mensagens pelo *contactIdentifier* passado por parâmetro e retorna um *ArrayList* com o conjunto de mensagem daquele contacto.

➤ `addMessage(String contactIdentifier, Message message)`

Método que adiciona a mensagem passada por parâmetro ao *ArrayList* de mensagens associado ao contacto passado por parâmetro (*contactIdentifier*).

➤ `addContact(Contact contact)`

Método que adiciona o contacto, passada por parâmetro ao *ArrayList* de contactos.

➤ `removeContact(String contactKey)`

Método que remove o contacto identificado por “*contactIdentifier*”, passado por parâmetro, do *ArrayList* de contactos.

3.5.3.2 Interface gráfica

Relativamente à interface gráfica, considerando que o utilizador acede ao menu e seleciona a implementação “Chat”, é-lhe apresentado no ecrã o *layout* que se pode observar pela Figura 38.

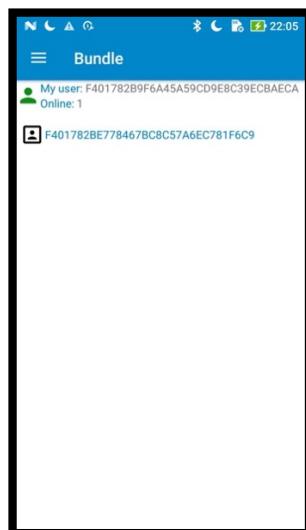


Figura 38 - Layout da implementação 03 - "Chat"

Com o invocar do método “*onCreateView*” é colocado em primeiro plano o *layout* do *fragment* referente aos contactos disponíveis para comunicar. Este *layout* é do tipo “*TableLayout*”¹³ e contem os seguintes componentes:

- Uma *imageView* para apresentação do ícone do utilizador
- Quatro *textView's* para apresentação de informações de texto relativas a identificações, *nicknames* e contadores de contactos disponíveis.
- Uma *listView* para apresentação dos contactos disponíveis na periferia

De notar que o *layout* desta implementação encontra-se definido num ficheiro .xml designado por “*fragment_chat.xml*” que é responsável pela composição da UI desta implementação. Para os elementos da UI que permitem dinamismo é criado o respetivo método *get[...]()* de maneira a permitir o acesso controlado aos componentes que abaixo se encontram descritos.

➤ *imageView*

Trata-se de um componente nativo do Android apropriado para representar imagens, neste caso e conforme podemos observar pela Figura 39, vemos o ícone apresentado como forma de dar *feedback* ao utilizador de que se encontra em estado de iniciar uma conversação.



Figura 39 – Ícone apresentado na *imageView*

➤ *textViews*

Permitem a apresentação de textos compostos por caracteres alfanuméricos e especiais. Nesta implementação é aqui que é apresentado o contador de instâncias encontradas na periferia (contactos), o *nickname* do utilizador, e os títulos referentes a cada campo apresentado anteriormente.

➤ *listView*

Trata-se de um componente para apresentar dados em lista, graças as suas potencialidades relacionadas com a listagem de dados fazem deste um componente

¹³ COMPONENTE DE EXIBIÇÃO DO ANDROID CARACTERÍSTICO PELA DISPOSIÇÃO SEMELHANTE A UMA TABELA (LINHAS E COLUNAS).

muito utilizado em Android para fins relacionados com a apresentação de dados. Na Figura 40 pode observar-se uma *listview* preparada para receber dados.

As *listView* em Android por norma tem associado um *adapter*, que é um elemento especializado para trabalhar no acesso, tratamento e apresentação de dados, visto que ele acede e armazena (não necessariamente) os dados a serem impressos na *listview*. Nesta implementação a *listview* apresenta os dados armazenados na persistência contidos no *arraylist* de contactos evitando assim replicação de informação.

Item 1
Sub Item 1

Item 2
Sub Item 2

Item 3
Sub Item 3

Item 4
Sub Item 4

Figura 40 – ListView utilizada para representar os contactos

Considerando que o utilizador seleciona um contacto para iniciar uma conversação é-lhe apresentado o *layout* da Figura 41.



Figura 41 - Layout janela de conversação

Este *layout*, da janela de conversação, é do tipo “*TableLayout*”¹⁴ e contém os seguintes componentes:

- Quatro *textView's* para apresentação das identificações dos contactos que estão a comunicar (emissor e destinatário).
- Uma *listView* para apresentação das mensagens trocadas entre os contactos.
- Um *editText* onde o utilizador digita o texto a enviar na mensagem.
- Duas *imageButton* para permitir ao utilizador enviar as mensagens (imagens ou texto).

Abaixo encontram-se descritos os componentes relativos ao *layout* “*fragment_message.xml*” responsável pela composição da UI relativa à janela de conversação.

➤ ***textView***

Componentes utilizados para apresentar informações ao utilizador, neste caso apresentam as identificações dos contactos que estão a trocar mensagens (emissor e receptor).

➤ ***listView***

Como apresentado anteriormente, trata-se de um componente para apresentar dados em lista. Nesta implementação a *listview* apresenta as mensagens armazenadas na persistência contidas no *ArrayList* de mensagens relativas aquele contacto.

➤ ***editText***

Componente utilizado para permitir ao utilizador digitar o conteúdo da mensagem a processar, este campo aceita caracteres alfanuméricos.

➤ ***imageButton***

Componente com funções semelhante a um botão, com a particularidade de apresentar uma imagem em vez de um texto, é responsável por iniciar o envio da mensagem.

¹⁴ COMPONENTE DE EXIBIÇÃO DO ANDROID QUE ALINHA TODOS OS ELEMENTOS FILHOS NUMA ÚNICA DIREÇÃO (VERTICAL OU HORIZONTAL)

3.5.3.3 Conteúdo de classes

No decorrer desta seção serão apresentados os métodos de cada classe referentes à implementação 03. Com isto pretendesse perceber em concreto o funcionamento de cada método bem como quem o invoca e quem é notificado mediante cada execução dos respetivos métodos, como tal a presente seção encontra-se dividida por classes e dentro de cada uma é feita uma divisão por tópicos onde cada tópico representa um método.

Utils.java

Neste tópico é efetuada uma abordagem à classe “*Utils.java*”, esta classe contém métodos de apoio ao desenvolvimento da implementação 03. Esta classe foi criada com vista a reutilização e organização do código. Para melhor entender o processo serão abaixo expostos os métodos contidos na classe efetuando uma descrição de cada.

➤ `base64encode(Bitmap bm)`

Método que efetua a conversão de um *bitmap* passado por parâmetro para uma *String* formatada em *Base64*¹⁵. Para isso recorre-se do método “*encodeToString()*” da classe nativa “*Base64.java*”.

➤ `base64decode(String str)`

Método que efetua a conversão de uma *String* formatada em *Base64* passada por parâmetro para um *bitmap*. Para isso recorre-se do método “*decodeByteArray()*” da classe nativa “*BitmapFactory.java*”.

➤ `scale(Bitmap bitmap, int max_height, int max_width)`

Método que transforma o tamanho do *bitmap*, que contém uma imagem, passado por parâmetro para as dimensões altura (*max_height*) e largura (*max_width*) também passadas por parâmetro. Para efetuar essas transformações o método verifica qual é o lado maior do *bitmap*, por forma a manter a proporção original da imagem, e gera um novo *bitmap* com recurso ao método “*createScaledBitmap()*” da classe nativa “*Bitmap.java*”.

¹⁵ BASE64 – CONVENÇÃO DE MÉTODO DE CODIFICAÇÃO DE DADOS PARA CARACTERES.

➤ `initDate()`

Método que devolve uma *String* com uma data no formato “dd/MM/yy HH:mm”, sendo que “dd” representa o dia, “MM” o mês, “yy” o ano em dois dígitos, “HH” as horas e “mm” os minutos. Para isso é criado um objeto do tipo *Date* e outro do tipo *SimpleDateFormat* com o tipo pretendido (dd/MM/yy HH:mm), posto isto é feito o *format* para atribuir o tipo *Date* ao *SimpleDateFormat* criado.

Contact.java

Neste tópico é efetuada uma abordagem à classe “*Contact.java*”, esta classe representa o *model* para os contactos, e como tal é responsável por definir qual a estrutura que um contacto deve seguir. Por forma a depreender a estrutura de um objeto deste tipo, abaixo é analisado o construtor referente aos objetos “*Contact*”.

➤ `Contact(Instance instance)`

Para este construtor foram definidas três variáveis sendo elas inicializadas no construtor partindo de uma instância passada por parâmetro. Um objeto do tipo “*Contact*” contém uma instância (*instance*), um nome (*name*) e um identificador (*identifier*) conforme é possível observar pela Figura 42.

- A instância representa o próprio objeto *instance* passado por parâmetro.
- O nome é um conjunto aleatório de caracteres gerado pelo java, recorrendo ao método “*randomUUID()*” da classe nativa do java “*UUID.java*”.
- O identificador é o mesmo identificador da instância passada por parâmetro obtido através de “*instance.getStringIdentifier()*”

```
public Contact(Instance instance) {  
    this.instance = instance;  
    this.name = UUID.randomUUID().toString();  
    this.identifier = instance.getStringIdentifier();  
}
```

Figura 42 - Construtor de contacts

Message.java

Neste tópico é efetuada uma abordagem à classe “*Message.java*” que representa o *model* para as mensagens, e como tal é responsável por definir qual a estrutura que uma mensagem deve seguir, esta classe funciona como superclasse de *TextMessage* e de *PhotoMessage*. Por forma a depreender a estrutura de um objeto deste tipo, abaixo é analisado o construtor referente aos objetos “*Message*”.

➤ Message(Instance instance)

Para este construtor foram definidas cinco variáveis e um método abstrato que retorna o tipo de mensagem. As *strings* são inicializadas no construtor partindo de outras duas *strings* passadas por parâmetro. Um objeto do tipo “*Message*” contém um identificador (*identifier*), um contacto de origem (*originContactIdentifier*), um contacto de destino (*destinyContactIdentifier*), uma data (*date*) e uma variável *boolean* (*isRead*) conforme se pode observar pela Figura 43.

- O identificador é constituído por um conjunto aleatório de caracteres gerado pelo java, recorrendo ao método “*randomUUID()*” da classe nativa do java “*UUID.java*”.
- O contacto de origem é a String “*originContactIdentifier*” passada por parâmetro.
- O contacto de destino é a String “*destinyContactIdentifier*” passada por parâmetro.
- A data é obtida através da invocação do método “*initDate()*” declarado na classe “*Utils.java*”.
- A variável *boolean* (*isRead*) representa o estado de leitura da mensagem. Visto que a mensagem no momento em que é construída ainda não foi lida/listada é declarada como *false*.

```
public Message(String originContactIdentifier, String destinyContactIdentifier){  
    this.identifier = UUID.randomUUID().toString();  
    this.originContactIdentifier = originContactIdentifier;  
    this.destinyContactIdentifier = destinyContactIdentifier;  
    this.date = Utils.initDate();  
    this.isRead = false;  
}
```

Figura 43 - Construtor de messages

TextMessage.java

Neste tópico é efetuada uma abordagem à classe “*TextMessage.java*”, esta classe é subclasse de “*Message*”, como tal estende a classe “*Message.java*”, sendo assim responsável por definir a estrutura própria de uma mensagem do tipo “texto”. Abaixo é analisado o construtor referente aos objetos “*TextMessage*”.

- `TextMessage(String text, String originContactIdentifier, String destinyContactIdentifier)`

Para esta classe foi definida uma variável do tipo *String* sendo que esta será inicializada no construtor partindo de um parâmetro que é passado simultaneamente com os restantes parâmetros a ser “endereçados” para a superclasse. Assim sendo um objeto do tipo “*TextMessage*” contém um texto (*text*), um contacto de origem (*originContactIdentifier*) e um contacto de destino (*destinyContactIdentifier*).

- O texto da mensagem é a *String* “*text*” passada por parâmetro.
- Os parâmetros “*originContactIdentifier*” e “*destinyContactIdentifier*” são enviados para superclasse conforme se pode observar pela Figura 44.

```
public TextMessage(String text, String originContactIdentifier, String destinyContactIdentifier) {  
    super(originContactIdentifier, destinyContactIdentifier);  
    this.text = text;  
}
```

Figura 44 - Construtor de *TextMessages*

- `getMessageType()`

Método declarado na superclasse como *abstrato*¹⁶ e como tal obrigatoriamente rescrito nas suas subclasses. Este método acede à classe do tipo enumerado¹⁷ para retornar o valor referente á variável caso “*TEXT*” conforme apresentado na Figura 45.

```
@Override  
public int getMessageType() {  
    return MessageType.TEXT.getValue();  
}
```

Figura 45 - Método `getMessageType()` / Classe “*TextMessage.java*”

¹⁶ MÉTODO DEFINIDO COM RECURSO À PALAVRA RESERVADA *ABSTRACT*, ESTES MÉTODOS NÃO POSSUEM IMPLEMENTAÇÃO COM TAL REQUEREM QUE SEJAM IMPLEMENTADOS NAS CLASSES FILHO.

¹⁷ TIPO DE CLASSE COMPOSTA POR CONSTANTES ESTÁTICAS INTEIRAS PARA REPRESENTAR UM CONJUNTO FINITO DE TIPOS.

PhotoMessage.java

Neste tópico é efetuada uma abordagem à classe “*PhotoMessage.java*”, esta classe é subclasse de “*Message*”, como tal estende a classe “*Message.java*”, ficando assim responsável por definir a estrutura própria de uma mensagem do tipo “*Photo*”. Por forma a depreender a estrutura de um objeto deste tipo, abaixo é analisado o construtor referente aos objetos “*PhotoMessage*”.

- `PhotoMessage(Bitmap bitmap, String originContactIdentifier, String destinyContactIdentifier)`

Para esta classe foi definida uma variável do tipo *bitmap* sendo que esta será inicializada no construtor partindo de um parâmetro que é passado simultaneamente com os restantes parâmetros a ser “endereçados” para a superclasse. Assim sendo um objeto do tipo “*PhotoMessage*” contém um *bitmap*, um contacto de origem (*originContactIdentifier*) e um contacto de destino (*destinyContactIdentifier*).

- O conteúdo da mensagem é o *bitmap* passada por parâmetro.
- Os parâmetros “*originContactIdentifier*” e “*destinyContactIdentifier*” são enviados para superclasse conforme se pode observar pela imagem Figura 46.

```
public PhotoMessage(Bitmap bitmap, String originContactIdentifier, String destinyContactIdentifier) {  
    super(originContactIdentifier, destinyContactIdentifier);  
    this.bitmap = bitmap;  
}
```

Figura 46 - Construtor de PhotoMessages

- `get MessageType()`

Método declarado na superclasse como *abstrato* e como tal imperiosamente rescrito nas suas subclasses. Este método acede à classe do tipo Enumerado para retornar o valor referente á variável caso “*PHOTO*” conforme se pode observar pela Figura 47.

```
@Override  
public final int getMessageType() {  
    return MessageType.PHOTO.getValue();  
}
```

Figura 47 - Método `get MessageType()` | Classe “*PhotoMessage.java*”

MessageType.java

A “*MessageType.java*” é uma classe do tipo enumerada, conforme se pode constatar pela Figura 48, na qual se encontram representados alguns valores fixos para representar o tipo de mensagens. Esta classe torna-se útil no momento de acesso aos objetos do tipo de mensagens permitindo facilmente comparar o tipo da mensagem que estamos a aceder com os tipos que são permitidos criar.

```
public enum MessageType {
    TEXT    (0),
    PHOTO   (1);

    private final int value;

    MessageType(int newValue) {
        this.value = newValue;
    }

    public final int getValue() {
        return value;
    }
}
```

Figura 48 - Enum MessageType

ContactDelegate.java

A “*ContactDelegate.java*” consiste numa interface¹⁸ com as assinaturas de dois métodos, o método “*callMessageFragment*”, que recebe como parâmetros um objeto do tipo *ContactFragment* e outro do tipo *Contact*, implementa também a assinatura do método “*getNameDomesticInstace*”, que recebe como parâmetros um objeto do tipo “*ContactFragment*”. Esta classe é implementada pela “*ChatManager*” que por isso vê os seus métodos aqui serem reescritos. Assim, sendo a criação desta classe permite que a classe “*ContactFragment*” possa notificar a classe gestora “*ChatManager*” que existiu um contacto selecionado pelo utilizador e por isso é necessário invocar “*MessageFragment*” para o contacto que fora selecionado. Este tipo de classe é invocada pela declaração de um objeto do tipo “*weak reference*”, que consiste na criação de uma referência fraca para um objeto. Supondo que um objeto crie um segundo objeto estes ficam ligados por uma “*weak reference*”. Quando o primeiro objeto é libertado da memória, quebra-se a ligação entre eles e consequentemente ambos são eliminados. Referências fracas permitem aproveitar as

¹⁸ É UM RECURSO QUE PERMITE QUE UM DETERMINADO GRUPO DE CLASSES POSSA TER MÉTODOS OU PROPRIEDADES EM COMUM DENTRO DE UM DETERMINADO CONTEXTO, CONTUDO OS MÉTODOS PODEM SER IMPLEMENTADOS EM CADA CLASSE DE UMA MANEIRA DIFERENTE. (ALBERTTANURE, 2012)

potencialidades do “*garbage collector*” e retirado preocupações ao programador relativas à gestão da memória.

ContactFragment.java

Neste tópico é efetuada uma abordagem à classe “*ContactFragment.java*”, que se trata da classe *controller* relativa aos contactos. Para melhor entender o processo abaixo são expostos os métodos contidos na classe efetuando uma descrição de cada um.

- `createListViewContactsOnLine()`

Método que cria e coloca em prontidão um componente de UI onde serão listadas as instâncias disponíveis para o início de uma conversação.

- `updateListViewContactsOnLine()`

Método responsável por atualizar a listView dos contactos disponíveis sempre que surja uma alteração relativa aos contactos.

- `updateNewMessages()`

Método invocado pelo no método “*ReceiveData*” que atualiza a UI com a nova mensagem recebida.

- `onSelectedContact(Contact contact)`

Método que invoca o método “*callMessageFragment*” do *delegate* “*ContactaDelegate*” para notificar que um contacto foi selecionado pelo utilizador, com o objetivo que seja carregada para a UI a janela de conversação daquele contacto.

- `updateNumberOnInstancesFound()`

Método invocado pelo “*onCreateView*” e sempre que seja adicionada ou removida uma instância à persistência. Este método atualiza na UI o contador de instâncias encontradas na periferia.

- `updateDomestic()`

Método invocado no “*onCreateView*” responsável por atualizar na UI a *textview* onde é apresento o “nick do meu equipamento”, para isso o método invoca o método “*getNameDomesticInstance*” do *delegate* “*ContactDelegate*”.

- `createContact(Instance instance)`

Método invocado pelo método “*notifyFoundInstance*” para criar um novo objeto do tipo *contact* e de seguida adiciona-lo à persistência.

➤ `notifyLostInstance(Instance instance)`

Método invocado pela “*MainActivity*” quando a *framework* deteta que uma instância se perdeu. Este método acede à persistência referente a esta implementação para eliminar a instância do *ArrayList* de *contacts*, e de seguida solicita atualizações de UI invocando para isso os métodos “*updateNumberOfInstancesFound()*” e “*updateListViewContactsOnline()*”. A instância sob a qual existem alterações é passada por parâmetro para este método.

➤ `notifyFoundInstance(Instance instance)`

Método invocado pela “*MainActivity*” quando a *framework* deteta que uma instância foi encontrada, ou porque entrou na periferia ou porque iniciou a execução da *Framework Hype*. Este método invoca o método “*createContact()*” e de seguida solicita atualizações de UI invocando para isso os métodos “*updateNumberOfInstancesFound()*” e “*updateListViewContactsOnline()*”. Tal como no método “*notifyLostInstance()*” a instância sob a qual existem alterações é passada por parâmetro para este método.

MessageDelegate.java

A “*MessageDelegate.java*” trata-se de uma interface que desempenha funções semelhantes à “*ContactDelegate.java*”. Esta interface contém a assinatura de um método, “*onExchangeMessage*” que recebe como parâmetros um “*MessageFragment*”, um Objeto JSON e um contacto. Esta classe é implementada pela “*ChatManager*” que por isso vê os seus métodos aqui serem reescritos. A criação desta classe permite que a classe “*MessageFragment*” possa notificar a classe gestora “*ChatManager*” que existem alterações a serem processadas. Pelo invocar do método “*onExchangeMessage*” é a notificada a classe “*ChatManager*” de que existe uma mensagem a enviar para um determinado destino. Este tipo de classe é invocada pela declaração de um objeto do tipo “*weak reference*” que consiste na associação de uma referência fraca para um objeto.

MessageFragment.java

Neste tópico é abordada a classe “*MessageFragment.java*”, que se trata da classe *controller* relativa as mensagens. De maneira a entender o processo serão abaixo expostos os métodos contidos na classe efetuando uma descrição de cada um.

- `sendMessage(Message message, Contact contact)`

Método invocado pelas ações definidas para os botões de envio de mensagens, responsável por notificar a classe “*ChatManager.java*” de que existe uma mensagem a propagar na rede, para tal o método invoca o método “*onExchangeMessges*” do *delegate* “*MessageDelegate*”.

- `createListViewMessages()`

Método que cria e coloca em prontidão um componente de UI, designada por *ListView*, onde serão listadas as mensagens trocadas entre dois dispositivos.

- `updateListViewMessages()`

Método responsável por atualizar a *ListView* das mensagens sempre que uma mensagem seja criada.

- `createJsonObj(Message message)`

Método cujo funcionamento se assemelha ao método “*createJsonObj()*” das implementações anteriores, no entanto para esta implementação este método retorna um *HashMap<String, String>* com dois elementos, um deles a chave *String* “MESSAGE_TYPE_KEY”, definida na classe *Demo.Defs*, tendo no conteúdo o tipo mensagem, o outro elemento do *HashMap* tem como chave a *String* “MESSAGE_CONTENT_KEY” tendo com conteúdo, a mensagem a enviar.

- `receiveData(JSON Object json, Instance instance)`

Método do protocolo de comunicação invocado quando a implementação recebe uma mensagem. Este método recebe um objeto JSON que representa o conteúdo da mensagem e a instância que iniciou o envio da mensagem pela *framework*. O conteúdo da mensagem é acedido com recurso à mesma chave usada na criação do objeto JSON (*MESSAGE_CONTENT_KEY*) e analisada a informação nela contida, em função disso e apenas no caso do conteúdo ser diferente de “*null*”, é criada uma nova mensagem tendo como base a verificação do tipo de mensagem efetuada previamente com recurso à chave definida na sua criação (*MESSAGE_TYPE_KEY*), posto isto é invocado o método “*receiveMessage()*” sendo-lhe passado o

contacto, obtido através da instância passada para o método “*receiveData*”, que enviou a mensagem e a mensagem criada.

➤ `addListenerOnButton()`

Método que trata as ações de todos os componentes do tipo “*Button*” presentes na IU. Este método acede ao botão através do seu método *get*, invoca o método nativo “*.setOnClickListener*” (*resultado do extends View.OnClickListener efetuado à classe*) e é aqui que é declarado o código referente as ações do botão que está a ser programado.

Para esta implementação existem dois botões, um para o envio de mensagens de texto, e outro para o envio de imagens. Este botões tem associados os métodos *get* designados por “*getIbSendMessage()*” e “*getIbSendImage()*” respetivamente.

Relativo a “*getIbSendMessage*” esta declaração cria uma nova *intent*¹⁹ que torna possível o acesso à galeria do telefone onde o utilizador seleciona uma imagem, após isto e de forma implícita é invocado o método “*onActivityResult*”.

➤ `onActivityResult (int requestCode, int resultCode, Intent data)`

Este método verifica se a seleção do utilizador na galeria de imagens do dispositivo tem como *resultCode == RESULT_OK*, e em caso afirmativo, retorna a imagem selecionada num *bitmap*, recorrendo-se para a conversão, do método nativo “*decodeStream*” da classe “*BitmapFactory.java*”.

De seguida é efetuado um escalonamento da imagem para 500 x 750 por forma a tornar a imagem mais pequena e menos pesada para proceder ao seu envio pela rede, para tal é invocado o método “*scale()*” da classe “*Utils.java*”.

Findas estas operações é criada uma mensagem do tipo *PhotoMessage* e adicionada à persistência, de seguida é invocado o método “*sendMessage()*” e efetuadas atualizações na UI através da invocação do método “*updateListViewMessages()*”.

¹⁹ “INTENTS SÃO MENSAGENS ASSÍNCRONAS QUE PERMITEM QUE UM COMPONENTES SOLICITE FUNCIONALIDADE DE OUTROS COMPONENTES, EM UMA LINGUAGEM MAIS SIMPLES É A INTENÇÃO DE SE REALIZAR UMA AÇÃO” (MACEDO, 2015)

- `receiveMessage(Contact contact, Message message)`

Método invocado pelo método “`receiveData()`” responsável por adicionar a mensagem, recebida por parâmetro, à persistência e de seguida invocar o método “`updateListViewMessages()`”.

ChatManagerDelegate.java

A “`ChatManagerDelegate.java`” tem um tipo de comportamentos semelhantes à “`MessageDelegate.java`” ou seja, consiste numa interface. Esta interface contém as assinaturas de dois métodos, “`onExchangeMessages()`” que recebe como parâmetros um objeto “`ChatManagerFrgament`”, um objeto JSON e um objeto do tipo *instance*. A classe implementa ainda a assinatura do método “`getNameDomesticInstance()`”, que recebe como parâmetro um objeto “`ChatManager`”. Esta classe é implementada pela “`MainActivity`” que por isso vê os seus métodos aqui a serem reescritos. Com a criação desta classe torna possível que a classe “`ChatManager`” notifique a classe “`MainActivity`” de que existem alterações a serem processadas. Pelo invocar do método “`onExchangeMessage()`” está a ser notificada a classe “`MainActivity`” que existe uma mensagem a enviar para um determinado destino. Ao invocar do método “`getNameDomesticInstance()`”, está a ser solicitado que a classe “`MainActivity`” questione à *Framework Hype* qual é instância domestica do próprio dispositivo. Conforme exposto anteriormente nas classes do mesmo tipo, estas classes são invocadas pela declaração de um objeto do tipo “`weak reference`” que consiste na associação de uma referência fraca para um objeto.

ChatManager.java

Neste tópico é efetuada uma abordagem à classe “`ChatManager.java`”. Trata-se da classe gestora de toda a implementação funciona como a ponte estabelecida entre as classes “`ContactFragment`” e “`MessageFrgament`”. Para melhor entender o processo e o seu funcionamento abaixo serão expostos os métodos contidos na classe efetuando uma descrição de cada um.

- `callMessageFrgament(ContactFragment contactFragment, Contact contact)`

Implementação do método da interface “`ContactDelegate`”.

Método que cria um objeto do tipo `MessageFrgament` e outro do tipo `FragmentManager`, com o objetivo de efetuar uma transição de fragmentos,

passando assim a despoletar um fragmento com o conteúdo de “*fragment_message.xml*”.

- `notifyLostInstance(Instance instance)`

Método invocado pela “*MainActivity*” para notificar a classe “*ContactFragment.java*” de que uma instância se perdeu.

- `notifyFoundInstance(Instance instance)`

Método invocado pela “*MainActivity*” para notificar a classe “*ContactFragment.java*” de que uma instância foi encontrada.

- `onExchangeMessages(MessageFragment messgeFragment, JSONObject json, Contact contact)`

Implementação de método da interface “*MessageDelegate*”.

Método usado para receber a notificação de que existe uma mensagem a propagar na rede, e por sua vez, invoca o *delegate* “*ChatMessageDelegate*” para notificar a “*MainActivity*” de que existe a mensagem (objeto *json*) para enviar ao contacto (*contact*).

MainActivity.java

Neste tópico é efetuada uma abordagem à classe “*MainAcitivity.java*”. Trata-se da classe principal e comum a todas as implementações, esta classe além de responsável pelas tarefas apresentadas em 3.5.1.2, também implementa os métodos da interface “*ChatManagerDelegate*” assim sendo implementa os métodos:

- `getNameDomesticInstance(ChatManager chatManager)`

Solicita à *Framework Hype* qual é a identificação da instância doméstica do próprio dispositivo.

- `onExchangeMessage (ChatManager chatManager, JSONObject json, Instance destiny)`

Notifica a *framework Hype* de que existe uma mensagem (*json*) a enviar para um determinado destino (*destiny*).

3.5.4 Implementação 04 - métricas

Na implementação 04 é pretendido que o utilizador aceda ao sistema (aplicação *bundle*) para obter métricas de desempenho da rede. Esta implementação vem colmatar uma necessidade da organização por forma a ter um recurso que permita à equipe e clientes, perceber quais as velocidades e tempo utilizado na transferência de pacotes de dados pela rede criada entre os dispositivos. Foi projetada a realização de uma aplicação que permita ao utilizador efetuar dois tipos de testes, os testes de *RTT* e os testes de métricas. Os testes de *RTT* consistem no envio, entre dois dispositivos, de um pacote de dados de tamanho definido pelo utilizador. O dispositivo 1 inicia o envio do pacote de dados para o dispositivo 2, este por sua vez ao receber o pacote de dados completo devolve-o para o dispositivo emissor, com este tipo de teste. Com este comportamento é possível perceber o tempo despendido para propagar uma mensagem e receber *feedback*. Quanto aos testes do tipo “*Metrics*” estes consistem também no envio de um pacote de dados entre dois dispositivos, sendo que neste teste não existe uma devolução do pacote ao emissor. Para perceber este tipo de teste importa denotar que a *Framework Hype* detém um algoritmo que detetando um grande pacote de dados fragmenta-os em vários pacotes de menor tamanho, e que a mesma está munida de um sistema que notifica o programador a cada fragmento de pacote enviado. Neste tipo de teste é criado a cada notificação de envio um objeto representativo do pacote de dados propagado designado por *iteration*, permitindo assim analisar quanto tempo foi gasto no envio, qual o volume de dados enviado e volume que falta enviar.

De acordo com o referido anteriormente em 3.5.1.2, por forma a integrar a “*Framework Hype*” é necessário proceder à implementação dos métodos das interfaces da *Framework*. No desenvolvimento desta aplicação foram criadas várias classes dentro do pacote “*Metrics*” conforme exposto na Tabela 39 onde são expostas as classes resumidamente e explicadas com maior detalhe ao longo deste capítulo.

Tabela 39 - Resumo das classes da implementação 04 – métricas

Classe	Síntese
Utils	Classe com métodos de apoio ao pacote da implementação 04
Test	Classe <i>model</i> referente aos objetos “test”
Peer	Classe <i>model</i> referente aos objetos “peer”
Iteration	Classe <i>model</i> referente aos objetos “iteration”
TestType	Classe do tipo enumerado que define um tipo de <i>test</i> (<i>RTT/Metric</i>)
GraphFragment	Classe <i>controller</i> geradora de gráficos
LogResultsDelegate	Interface com funções de <i>delegate</i> para notificar a classe “MetricsManager”
LogResultsFragment	Classe <i>controller</i> para os resultados dos testes efetuados
MetricsDelegate	Interface com funções de <i>delegate</i> para notificar a classe “MetricsManager”
MetricsFragment	Classe <i>controller</i> dos <i>tests</i>
MetricsManagerDelegate	Interface com funções de <i>delegate</i> para notificar a classe “MainActivity”
MetricsManager	Classe <i>controller</i> que faz a gestão para/entre as classes “MetricsFragment”, “LogResultsFragment”, “GraphFragment”

3.5.4.1 Persistência

Relativamente à persistência de dados tal como nas implementações anteriores, foi criada uma *interface* (*MetricsPersistence.java*) que contém os métodos alusivos a esta implementação, conforme se pode observar na Figura 49. Os métodos definidos nesta classe são reescritos na classe “*Persistence.java*”

```
public interface MetricsPersistence {  
    ArrayList<Peer> getPeers();  
    ArrayList<Test> getTests();  
    ArrayList<Iteration> getIterations();  
    ArrayList<String> getListSpinners();  
    ArrayList<Number> getDomainLabels();  
    ArrayList<Number> getSeriesNumber();  
    void addDomainLabels(Number number);  
    void addSeriesNumber(Number number);  
    void addPeer(Peer peer);  
    void addTest(Test test);  
    void addIteration(Iteration iteration);  
    void addItemSpinner(String str);  
    void removePeer(String peerIdentifier);  
    void removeItemSpinner(String str);  
}
```

Figura 49 - Interface “*MetricsPersistence.java*”

Quanto à classe “*Persistence.java*” foram reescritos os métodos da interface e criados seis contentores de objetos do tipo *ArrayList*. Um *ArrayList* para armazenar os *peers* (um *peer* representa uma instância “online”), um para coletar os *tests*, outro para as *iterations* (*feedback* de receção de cada pacote no destinatário), dois para apoio a criação de gráficos onde são armazenados os dados a representar no gráfico designados por *domainLabels* (identificação das iterações – eixo XX) e *seriesNumber* (velocidade de transferência de dados – eixo YY) e um último designado por *listSpinners* que se trata de um *ArrayList* auxiliar para permitir que a UI carregue conteúdos num determinado componente designado por *spinner*.

Os métodos da interface *MetricsPersistence* foram reescritos de forma a tomar os comportamentos descritos nos tópicos abaixo.

- `getPeers()`

Método que permite o acesso ao *ArrayList* de *peers*.

- `getTests()`

Método que permite o acesso ao *ArrayList* de *tests*.
- `getIterations()`

Método que permite o acesso ao *ArrayList* de *Iterations*.
- `getListSpinners()`

Método que permite o acesso aos elementos a carregar no elemento de UI (Spinner).
- `addIteration(Iteration iteration)`

Método que adiciona uma *iteration*, passada por parâmetro, ao *ArrayList* de *Iterations*.
- `addPeers(Peer peer)`

Método que adiciona um *peer*, passado por parâmetro, ao *ArrayList* de *Peers*
- `addTest(Test test)`

Método que adiciona um *test*, passado por parâmetro, ao *ArrayList* de *Tests*
- `addItemSpinner(String str)`

Método que adiciona uma *string* que contém o *nickname* do *peer*, ao *ArrayList* (*listSpinners*) por forma a permitir que este seja listado no elemento de UI (Spinner)
- `removePeer(String peerIdentifier)`

Método que remove o *peer* identificado por *contactIdentifier*, passado por parâmetro, do *ArrayList* de *peers*.
- `removeItemSpinner(String str)`

Método que remove o item com o *nickname* passado por parâmetro, do *ArrayList* *listSpinners*.
- `addDomainSeries (Number number)`

Método que adiciona um valor, passado por parâmetro, ao *ArrayList* *domainSeries* que representa os identificadores de cada iteração a mostrar no gráfico.

- addSeriesNumber(Number number)

Método que adiciona um valor, passado por parâmetro, ao *ArrayList* seriesNumber que representa o valor espelho (velocidade de transferência de dados) da iteração representada no eixo das abcissas.

3.5.4.2 Interface gráfica

Considerando que o utilizador acede ao menu e seleciona a implementação “Metrics” é-lhe apresentado no ecrã o *layout* observado na Figura 50.

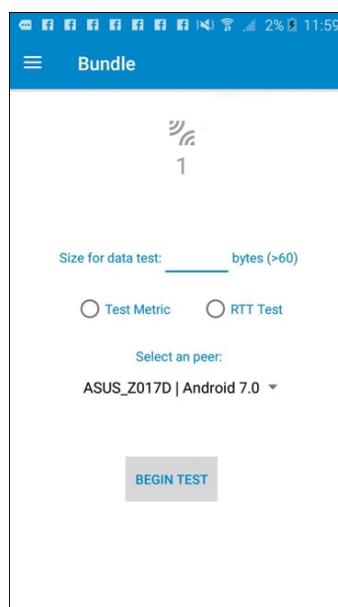


Figura 50 - Layout da implementação 04 - “Metrics”

Este *layout*, de definição de parâmetros para o teste, é do tipo “*TableLayout*”²⁰ e contém os seguintes componentes:

- Uma *imageView* para apresentação do ícone do utilizador.
- Três *textView's* para apresentação de informações de texto relativas a identificações e contadores de *peers* disponíveis.

²⁰ COMPONENTE DE EXIBIÇÃO DO ANDROID CARACTERÍSTICO PELA DISPOSIÇÃO SEMELHANTE A UMA TABELA (LINHAS E COLUNAS).

- Dois *radioButtons* para seleção do tipo de teste.
- Uma *spinner* para listagem dos *peers* disponíveis.
- Uma *editText* para permitir ao utilizador inserir o tamanho do pacote de dados a usar no teste.
- Um *button* para permitir iniciar o teste.

Abaixo encontram-se descritos os componentes relativos ao layout “*fragment_metrics.xml*” responsável pela composição da UI relativa à seleção dos parâmetros do teste.

- *imageView*
Trata-se de um componente nativo do Android apropriado para representar imagens, neste caso o ícone é apresentado como forma de dar *feedback* ao utilizador de quantos *peers* se encontram disponíveis para comunicar.
- *textViews*
Permitem a apresentação de textos compostos por caracteres alfanuméricos e especiais. Nesta implementação é aqui que é apresentado o contador de *peers* encontradas na periferia, e as legendas para o atributo *editText*.
- *radioButton*
Componente apresentado com o tipo de teste, característico por permitir efetuar uma seleção restrita, neste caso, permitem selecionar apenas e só um tipo de teste conforme observado na Figura 51.



Figura 51 – RadioButton usada no layout

- *spinner*
Componente que permite listar conteúdos, e efetuar a seleção de apenas um.
Torna-se útil para economizar espaço para uma apresentação mais agradável de conteúdo, pois trata-se de um elemento que expande ao ser clicado para mostrar o conteúdo.

➤ **editText**

Usada para introdução de caracteres numéricos que representam o tamanho do pacote de dados a ser processado no teste.

➤ **Button “BEGIN TEST”**

Botão utilizado para aceder e obter os parâmetros definidos na UI para iniciar a execução do teste.

Após seleção dos parâmetros e clicado o botão “*BEGIN TEST*” é apresentado ao utilizador o ecrã da Figura 52 com o resultado das iterações do teste.

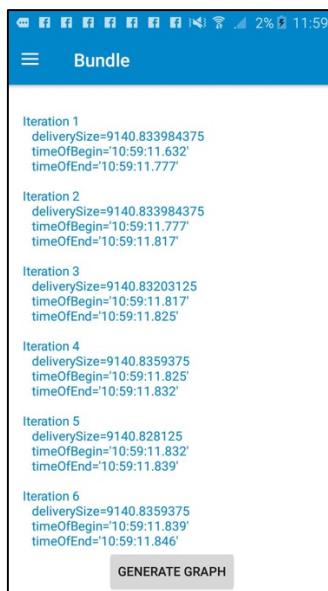


Figura 52 - Layout das iterações de um teste

Este *layout*, de definição de parâmetros para o teste, é do tipo “*LinearLayout*”²¹ e contém os seguintes componentes:

- Uma *textView’s* para apresentação os resultados das iterações.
- Um *scrollView* para permitir ao utilizador navegar dentro da *textview* onde aparecem os resultados das iterações.
- Um *button “GENERATE GRAPH”* que fica visível quando se trata de um teste do tipo *metrics* que permite gerar um gráfico.

²¹ COMPONENTE DE EXIBIÇÃO DO ANDROID QUE ALINHA TODOS OS ELEMENTOS FILHOS NUMA ÚNICA DIREÇÃO (VERTICAL OU HORIZONTAL)

Abaixo encontram-se descritos os componentes relativos ao *layout* “*fragment_log_results.xml*” responsável pela composição da UI relativa aos resultados das iterações de um teste.

➤ Button “*GENERATE GRAPH*”

Botão utilizado para gerar um gráfico onde é representada a velocidade de envio de cada iteração. Este botão no caso dos testes de RTT não se encontra visível e como tal não é possível gerar gráficos para este tipo de teste.

➤ *textViews*

Permitem a apresentação de textos compostos por caracteres alfanuméricos e especiais. Nesta implementação é aqui que são apresentados os resultados das iterações referentes ao teste em execução.

O último ecrã que poderá ser apresentado nesta implementação é relativo ao gráfico gerado a partir das iterações de um teste do tipo “*metrics*”. Assim sendo caso o utilizador prima o botão “*GENERATE GRAPH*” é apresentado o *layout* da Figura 53.

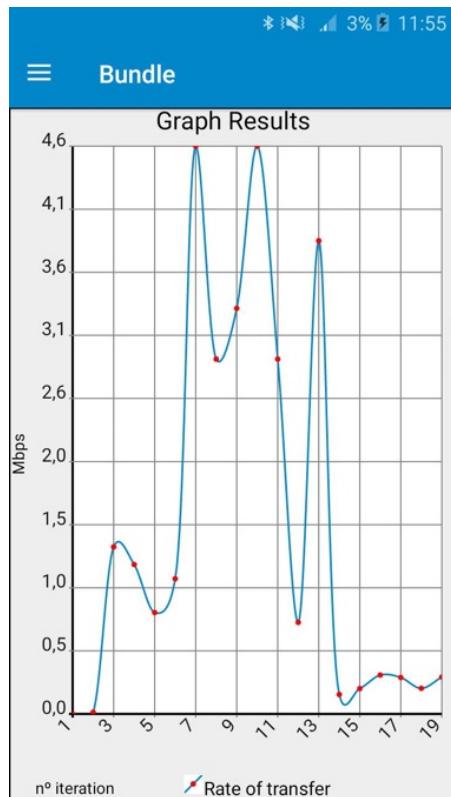


Figura 53 - layout do gráfico das iterações de um teste

Este *layout* que apresenta o gráfico, é do tipo “*LinearLayout*” e contém um componente, “*com.androidplot.xy.XYPLLOT*” que se trata de uma API externa de distribuição livre para geração de gráfico em Android. Esta API será apresentada detalhadamente ao longo do tópico 3.5.4.3.

3.5.4.3 Conteúdo de classes

No decorrer desta seção serão apresentados os métodos de cada classe referentes à implementação 04. Com isto pretendesse perceber em concreto o funcionamento de cada método bem como quem o invoca e quem é notificado mediante cada execução dos respetivos métodos, como tal, a presente seção encontra-se dividida por classes e dentro de cada uma é feita uma divisão por tópicos onde cada tópico é um método.

Utils.java

Neste tópico é efetuada uma abordagem à classe “*Utils.java*”, esta classe contém métodos de apoio ao desenvolvimento da implementação 04. Criada visando a reutilização e organização do código tendo sido implementados os seguintes métodos:

- convertTime(long time)

Método que converte uma variável do tipo *long*, passada por parâmetro, para o formato de hora no tipo “HH:mm:ss.SSS”, sendo que “HH” representa as horas, “mm” os minutos, “ss” os segundos e “SSS” os milissegundos. Para tal é criado um objeto do tipo *Date* e outro do tipo *SimpleDateFormat* com o tipo pretendido (HH:mm:ss.SSS), posto isto é efetuado o *format* para atribuir o tipo *Date* ao *SimpleDateFormat* criado.

- createDataSize(int size)

Método que gera uma *string* com o tamanho passado por parâmetro na variável *size*. Para isso é criado um objeto do tipo *StringBuilder* que posteriormente será inicializado em todas as posições com o carácter ‘a’.

- calcBitesPerSecond(float time, double tamanho)

Método que calcula quantos bites foram transferidos por segundo a partir do tempo e do tamanho passado por parâmetro.

- timeStampToMiliseconds(String inputString)

Método que devolve o tempo em milissegundos a partir do tempo passado como parâmetro no formato *string*.

TestType.java

A classe “*TestType.java*” é uma classe do tipo enumerada, conforme se pode observar pela Figura 54, que representa o tipo de testes que o utilizador pode criar. Esta classe torna-se útil no momento de acesso aos objetos do tipo de “*TestType*” no sentido de facilitar a comparação do tipo de teste a que se está a aceder.

```
public enum TestType {
    RTT      (0),
    METRIC  (1);

    private final int value;

    TestType(int newValue) { this.value = newValue; }

    public final int getValue() { return value; }

    public static TestType getTestType(String type)
    {
        return TestType.values()[Integer.parseInt(type)];
    }
}
```

Figura 54 - Enum *TestType*

Test.java

Ao longo deste tópico é efetuada uma abordagem à classe “*Test.java*”, esta classe representa o *model* relativo aos testes e assim sendo é responsável por definir qual a estrutura que um teste deve seguir. Abaixo é analisado o construtor referente aos objetos “*test*” que se pode observar na Figura 55.

- `Test(TestType testType, Peer peerOrigin, Peer peerDestination, long timeOfBegin, long timeOfEnd, int sizeData)`

Para este construtor foram definidas as variáveis convenientes para caracterizar um objeto deste tipo. Assim sendo estes objetos são compostos por um identificador (*identifier*), um tipo (*testType*), um emissor (*peerOrigin*), um destinatário (*peerDestination*), uma hora de início (*timeOfBegin*), uma hora de fim (*timeOfEnd*), um tamanho do pacote usado para o teste (*sizeData*) e ainda por uma variável booleana (*completed*).

- O identificador (*identifier*) é constituído por contador sem repetições inicializado a zero.
- *testType* representa o tipo de teste a ser criado.
- O emissor (*peerOrigin*) é a instância “*online*” que cria e executa o teste, pode ser interpretado como “cliente”.
- O destinatário (*peerDestination*) representa a instância que vai receber o teste, pode ser interpretado como “servidor”.
- A hora de início (*timeOfBegin*) é a hora do sistema em que o utilizador cria o teste. Obtida através do método “*currentTimeMillis()*” da classe nativa “*System.java*”
- A hora de fim (*timeOfEnd*) é a hora a que o teste ficou concluído, tal como a hora de inicio esta também é obtida a partir da invocação do método “*Sysyem.currentTimeMillis()*”.
- O tamanho do pacote usado no teste (*sizeData*) representa o tamanho do volume de dados que o utilizador define para processar durante a execução do teste.
- A variável booleana (*completed*) representa o estado de conclusão do teste.

```
public Test(TestType testType, Peer peerOrigin, Peer peerDestination,  
           long timeOfBegin, long timeOfEnd, int sizeData) {  
    this.identifier = ID++;  
    this.testType = testType;  
    this.peerOrigin = peerOrigin;  
    this.peerDestination = peerDestination;  
    this.timeOfBegin = timeOfBegin;  
    this.timeOfEnd = timeOfEnd;  
    this.sizeData = sizeData;  
    this.completed = false;  
}
```

Figura 55 - Construtor de Test

Iteration.java

Neste tópico é efetuada uma abordagem à classe “*Iteration.java*” esta classe representa o *model* relativo as iterações, de salientar que um teste é comporto por varias iterações entre os dois dispositivos (associação à resposta enviada do servidor para o cliente pela invocação do método “*onMessageDelivered*”). Abaixo é analisado o construtor apresentado na Figura 56 referente aos objetos do tipo “*iteration*”.

➤ Iteration(Test test, Double deliverySize, long timeOfBegin, long timeOfEnd)

Para este construtor foram definidas cinco variáveis para caracterizar um objeto do tipo *iteration*. Assim sendo estes objetos são compostos por um identificador (*identifier*), um teste (*test*), um tamanho enviado na iteração (*deliverySize*), uma hora de início (*timeOfBegin*) e uma hora de fim (*timeOfEnd*).

- O identificador (*identifier*) é constituído por contador sem repetições inicializado a zero.
- O teste (*test*) é um objeto do tipo “*Test*” ao qual a iteração é associada, ou seja, representa o teste que originou a iteração.
- O tamanho enviado na iteração (*deliverySize*) representa o quanto foi enviado na iteração, este valor é calculado a partir tamanho total do pacote de dados e da percentagem de envio retornada pela *framework* a cada invocação do método “*onMessageDelivered*”.
- A hora de inicio (*timeOfBegin*) é a hora à qual foi criada a iteração, esta hora é obtida do sistema através do método “*currentTimeMillis()*” da classe nativa “*System.java*”
- A hora de fim (*timeOfEnd*) é a hora a que a iteração foi concluída, tal como a hora de inicio esta também é obtida do sistema a partir da invocação do método “*Sysyem.currentTimeMillis()*”.

```
public Iteration(Test test, Double deliverySize, long timeOfBegin, long timeOfEnd) {  
    this.identifier = ID++;  
    this.test = test;  
    this.deliverySize = deliverySize;  
    this.timeOfBegin = timeOfBegin;  
    this.timeOfEnd = timeOfEnd;  
}
```

Figura 56 – Construtor de Iteration

Peer.java

A classe “*Peer.java*” representa o *model* relativo aos *peers* que representam no sistema os dispositivos “*online*” para efetuar testes. O construtor de objetos do tipo “*peer*” é apresentado na Figura 57.

➤ Peer(Instance instance, String name, String operatingSystem)

Para este construtor foram definidas cinco variáveis para caracterizar um objeto do tipo *peer*. Estes objetos são compostos por um identificador (*identifier*), um nome (*name*), uma instância (*instance*), um sistema operativo (*operatingSystem*) e um *nickname* (*nick*).

- O identificador (*identifier*) é construído a partir da instância passada como parâmetro, accedendo ao identificador da instância.
- O nome (*name*) representa o nome do dispositivo que está a criar o objeto, obtido com acesso as informações do dispositivo.
- O sistema operativo (*operatingSystem*) é referente ao sistema operativo do dispositivo que está a criar o objeto, obtido com acesso as informações do dispositivo.
- O *nickname* (*nick*) trata-se de uma variável auxiliar para a UI, onde é feita uma compilação do *nickname* e do sistema operativo.

```
public Peer(Instance instance, String name, String operatingSystem) {  
    this.identifier = instance.getStringIdentifier();  
    this.name = name;  
    this.instance = instance;  
    this.operatingSystem = operatingSystem;  
    this.nick = name + " | Android " + operatingSystem;  
}
```

Figura 57 - Construtor Peer

MetricsDelegate.java

A “MetricsDelegate.java” é uma interface²² com a assinatura do método “onSelectedDetails”, que recebe como parâmetros um objeto do tipo *MetricsFragment*, um outro do tipo *TestType*, duas *Strings* e um inteiro.

Esta classe é implementada pela “MetricsManager” que por isso vê os seus métodos aqui serem reescritos. A criação desta classe permite que a classe “MetricsFragment” possa notificar a classe gestora “MetricsManager” de que o utilizador selecionou todos os detalhes para o teste e que se encontra em posição de iniciar o teste. Estes tipos de classe são invocados pela declaração de um objeto do tipo “weak reference” que consiste na associação de uma referência fraca para um objeto.

²² É UM RECURSO QUE PERMITE QUE UM DETERMINADO GRUPO DE CLASSES POSSA TER MÉTODOS OU PROPRIEDADES EM COMUM DENTRO DE UM DETERMINADO CONTEXTO, CONTUDO OS MÉTODOS PODEM SER IMPLEMENTADOS EM CADA CLASSE DE UMA MANEIRA DIFERENTE. (ALBERTTANURE, 2012)

MetricsFragment.java

Neste tópico é abordada a classe “*MetricsFragment.java*” trata-se da classe *controller* relativa aos componentes de um teste. De maneira a entender o processo serão abaixo expostos os métodos contidos na classe efetuando uma descrição de cada um.

- `updateNumberPeersFound()`

Método responsável por atualizações a nível de UI, acede à persistência e verifica qual o tamanho do *HashMap* de *peers* e escreve o seu tamanho na referente *textView*.

- `getAnnouncement()`

Método invocado pela “*MainActivity*” para propagar na rede a identificação do “meu dispositivo”, esta identificação é composta pelo modelo do equipamento e pela versão do sistema operativo.

- `createItemsOnSpinner()`

Método invocado para criar o spinner onde serão apresentados os peers disponíveis na periferia. Este método é invocado pelo “*onCreateView()*” colocando o componente pronto para receber interações.

- `updateItemsOnSpinner()`

Método que atualiza o conteúdo da lista de *spinners*.

- `removeItemFromListSpinners(Instance instance)`

Este método acede à persistência para remover o *peer* que representa a instância passada por parâmetro.

- `notifyLostInstance(Instance instance)`

Método invocado pela classe “*MetricsManager.java*” quando uma instância é perdida. Este método remove da persistência o *peer* representado pela instância passada por parâmetro, invoca o método “*removeItemFromListSpinners()*” e efetua ainda as respetivas atualizações à UI “*updateItemOnSpinner()*”, “*updateNumberPeersFound()*”.

➤ `notifyFoundInstance(Instance instance)`

Método invocado pela classe “*MetricsManager.java*” quando uma instância é encontrada. Invoca o método “*createPeer()*” e efetua ainda as respetivas atualizações à UI “*updateItemOnSpinner()*”, “*updateNumberPeersFound()*”.

➤ `createPeer(Instance instance)`

Método invocado pelo “*notifyFoundInstance()*” que cria um *peer* a partir de uma instância passada por parâmetro. Após criado o objeto este método adiciona-o à persistência e atualiza a UI “*updateItemOnSpinner()*”.

➤ `onclick(View v)`

A invocação deste método representa a definição de eventos associados ao clicar do botão. Para permitir este tipo de operações é mandatório que a classe faça o *implements* do *View.OnClickListener*, e como tal é necessário efetuar o *override* deste método. É neste método onde estão programadas as ações para o botão, neste caso em concreto é usado o número de *peers* disponíveis para iniciar as ações do botão. As ações do botão passam por aceder aos dados que o utilizador definiu para o teste na UI. O tamanho para o teste, o tipo de teste e um *peer*, são os dados que o utilizador tem que definir para o botão conseguir notificar a classe gestora que de o utilizador definiu todos os parâmetros e que é necessário iniciar o teste.

LogResultsDelegate.java

A “*LogResultsDelegate.java*” é uma classe do tipo interface²³ que contém a assinatura do método “*notifyGenerateGraph*” que recebe como parâmetro um objeto do tipo *LogResultsFragment*.

Também esta é uma classe implementada pela “*MetricsManager*”, que por se tratar de uma interface tem os seus métodos aqui reescritos. A sua criação permite à classe “*LogResultsFragment*” notificar a classe gestora “*MetricsManager*” da intenção do utilizador em gerar um gráfico com os resultados das iterações criadas no teste. Estes tipos de classe são invocados pela declaração de um objeto do tipo “*weak reference*” que consiste na associação de uma referência fraca para um objeto.

²³ É UM RECURSO QUE PERMITE QUE UM DETERMINADO GRUPO DE CLASSES POSSA TER MÉTODOS OU PROPRIEDADES EM COMUM DENTRO DE UM DETERMINADO CONTEXTO, CONTUDO OS MÉTODOS PODEM SER IMPLEMENTADOS EM CADA CLASSE DE UMA MANEIRA DIFERENTE. (ALBERTTANURE, 2012)

LogResultsFragment.java

A classe “*LogResultsFragment.java*” trata-se da classe *controller* relativa as iterações de um teste. Por forma a entender o seu funcionamento, abaixo estão explícitos os métodos contidos na classe.

- `notifyResponseMetrics()`

Método responsável por atualizações a nível de UI que escreve o resultado de uma iteração de um teste do tipo “*metrics*” na *textView* correspondente.

- `notifyResponseRTT(int testID)`

Método responsável por atualizações a nível de UI que escreve o resultado de um teste do tipo “*RTT*” na *textView* correspondente.

- `addListenerOnButton()`

Método que implementa as ações do botão “*BEGIN TEST*”. As ações do botão são a invocação do *delegate* para notificar a classe gestora “*MetricsManager*” que existe um teste para iniciar.

GraphFragment.java

A classe “*GraphFragment.java*” trata-se da classe *controller* responsável pela criação de gráficos para esta implementação. Os gráficos são criados no método “*onCreateView()*” da classe. Na implementação deste método é estabelecido o acesso aos dados da persistência criada para este fim (*ArrayList domainLabels e seriesNumber*) para que os dados sejam apresentados no gráfico. Para a criação de gráficos convencionou-se a utilização de uma API externa de distribuição livre designada “*AndroidPlot*”(Fellows, 2017) já conhecida e utilizada pela empresa noutrios projetos. Para que a utilização dessa API seja possível é necessário efetuar a adição da dependência, *compile “com.androidplot:androidplot-core:1.4.3”*, ao ficheiro “*build.gradle*”. A nível de UI é necessário efetuar a declaração do componente relativo ao gráfico conforme excerto de código da Figura 58 no ficheiro “*fragment_graph.xml*”. É também necessário criar um objeto do tipo “*SimpleXYSeries*” que fará a associação dos *ArrayList* que guardam dos dados a apresentar.

```
<com.androidplot.xy.XYPlot  
    style="@style/APDefacto.Light"  
    android:id="@+id/plot"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    ap:title="Graph Results"  
    ap:rangeTitle="Mbps"  
    ap:domainTitle="nº iteration"  
    ap:lineLabels="left|bottom"  
    ap:lineLabelRotationBottom="-45"/>
```

Figura 58 - Excerto de código para criação do gráfico

MetricsManagerDelegate.java

A “*MetricsManagerDelegate.java*” é uma interface²⁴ com a assinatura dos métodos “*getNameDomesticInstance*”, que recebe por parâmetro um objeto do tipo “*MetricsManager*” e do método “*onNotifySendData*”, que recebe como parâmetro um objeto do tipo *MetricsManager*, um objeto JSON e uma instância.

Esta interface é implementada pela “*MainActivity*” e como tal, é aqui que os seus métodos são reescritos. A criação desta classe permite que a classe “*MetricsManager*” possa notificar e solicitar informações à *framework* utilizando com intermediaria a classe “*MainActivity*”.

MetricsManager.java

Neste tópico é efetuada uma abordagem à classe “*MetricsManager.java*”. Trata-se da classe gestora de toda a implementação funciona como a ponte estabelecida entre as classes “*LogResultsFragment*”, “*MetricsFragment*” e “*GraphFragment*”. Para melhor entender o processo e o seu funcionamento abaixo serão expostos os métodos contidos na classe efetuando uma descrição de cada um.

➤ notifyFoundInstance(Instance instance)

Método invocado pela “*MainActivity*” que notifica a classe “*MetricsFragment.java*” de que uma instância foi encontrada.

²⁴ É UM RECURSO QUE PERMITE QUE UM DETERMINADO GRUPO DE CLASSES POSSA TER MÉTODOS OU PROPRIEDADES EM COMUM DENTRO DE UM DETERMINADO CONTEXTO, CONTUDO OS MÉTODOS PODEM SER IMPLEMENTADOS EM CADA CLASSE DE UMA MANEIRA DIFERENTE. (ALBERTTANURE, 2012)

- `notifyLostInstance(Instance instance)`

Método invocado pela “*MainActivity*” que notifica a classe “*MetricsFragment.java*” de que uma instância foi perdida.
- `getPeerByNick(String selectedItem)`

Acede à persistência e retorna um *peer* a partir da *string* passada como parâmetro, nessa *string* está o *nickname* do elemento selecionado na lista de *spinners*.
- `callLogResults()`

Coloca em exibição o fragmento com o conteúdo de “*fragment_log_results.xml*”. Para isso cria um objeto do tipo “*LogResultsFragment*” e outro do tipo “*FragmentManager*”, com o objetivo de efetuar uma transição de fragmentos.
- `onSelectedDetails(MetricsFragment metricsFragment, TestType testType, String nick, String timeLimitForTest, int sizeData)`

Implementação do método da interface “*MetricsDelegate*” responsável por criar um teste, adiciona-lo à persistência e propagar à *framework* a execução do teste. Após a criação do teste, este método invoca o método “*callLogResults()*”.
- `notifyGenerateGraph()`

Método com funções semelhantes a “*callLogResults*”, que cria dois objetos do tipo “*GraphResults*” e outro “*FragmentManager*”, para colocar em execução um fragmento com o conteúdo de “*fragment_graph.xml*”.
- `createJsonObj(Test test)`

Método cujo funcionamento se assemelha ao método “*createJsonObj()*” das implementações anteriores, no entanto para esta implementação é criado um *HashMap* que é convertido num objeto *JSON* para ser retornado. Na criação deste objeto são adicionadas quatro entradas ao *HashMap*, sendo elas o tipo de teste passado por parâmetro (*Demo.Defs.TESTE_TYPE_KEY*), o *identifier* do teste (*Demo.Defs.TESTE_IDENTIFIER*), uma variável de controlo que indica se a mensagem já foi recebida pelo servidor, que no momento da criação é sempre zero (*Demo.Defs.ENTER_ON_SERVER*) e por último o pacote de dados a ser processado (*Demo.Defs.TEST_CONTENT_KEY*).

➤ calcTimeBeginOfIteration(Test t1)

Calcula a hora de início de uma iteração. Para efetuar o calculo acede à persistência de testes através do teste passado como parâmetro para verificar se se trata da primeira iteração, em caso afirmativo a hora de inicio da iteração será a mesma do teste, caso não seja a primeira iteração, acede à ultima iteração daquele teste e obtém a sua hora de fim.

➤ calcSizeDelivered(Test t1, float v)

Calcula o tamanho de dados enviados. Para tal acede à persistência de testes através do teste *t1* passado como parâmetro para verificar qual o tamanho total do teste, de seguida com a percentagem de envio contida na variável “v” também passada por parâmetro efetua o cálculo do tamanho.

➤ calcTimeUsedInTest(String testID)

Método que calcula o tempo decorrido numa interação, acede à persistência com recurso ao *testID* passado por parâmetro e calcula o tempo decorrido desde a última iteração.

➤ notifyNewIteration(Instance instance, float v)

Método que cria uma iteração a cada notificação da *framework* através do método “*onMessageDelivered()*”. Este método acede à persistência para que a partir da instância passada por parâmetro encontre o teste referente à iteração que vai criar. Posto isto e com recurso aos métodos “*calcTimeBeginOfIteration()*” e “*calcSizeDelivered()*” efetua o calculo do tempo decorrido e o tamanho enviado na iteração. Por consequente cria a iteração que é adicionada à persistência.

Relativamente à criação dos gráficos, no caso dos testes do tipo “Metrics” este método denota um papel muito importante, ao colocar na persistência os dados a apresentar nos gráficos a criar posteriormente. Este método calcula os bites transferidos por segundo em cada iteração adicionando esses dados ao *ArrayList* “seriesNumber” da persistência, acrescenta ainda ao *ArrayList* “DomainLabel” o identificador da instância criada.

➤ receiveData(JSONObject json, Instance instance)

Método do protocolo de comunicação invocado quando a implementação recebe um teste. Este método recebe um objeto *JSON* com detalhes acerca do teste e uma instância que iniciou o teste. O conteúdo da mensagem é acedido com recurso as

mesmas chaves usadas na criação do objeto JSON (*TESTE_IDENTIFIER*, *TEST_TYPE_KEY*, *ENTER_ON_SERVER*, *TEST_CONTENT_KEY*). Acedido o conteúdo e analisado o tipo de teste, se se tratar de um teste do tipo *RTT*, e no caso do valor de *ENTER_ON_SERVER* se manter a zero, significa que é a primeira vez que a mensagem está no destino. Quando isso se verifica, a variável *ENTER_ON_SERVER* é atualizada para um, e a mensagem reencaminhada para a origem pois um teste de *RTT* consiste na verificação do tempo que uma mensagem necessita para chegar ao servidor e voltar ao ponto de origem. No caso do campo *ENTER_ON_SERVER* se encontrar a um, significa que a mensagem já foi ao destino e voltou para a origem, como tal o teste está concluído e é necessário colocar no teste, representado por *TEST_IDENTIFIER*, a hora da sua conclusão, obtida através da hora de sistema com recurso ao método “*System.currentTimeMillis()*”. Se se tratar de um teste do tipo *METRICS* a mensagem é ignorada, pois para este tipo de teste só são considerados as notificações dadas pela *framework* para o equipamento de origem.

MainActivity.java

Neste tópico é efetuada uma abordagem à classe “*MainAcitivity.java*”. Trata-se da classe principal e comum a todas as implementações, esta classe além de responsável pelas tarefas apresentadas em 3.5.1.2 também implementa os métodos da interface “*MetricsManagerDelegate*” assim sendo reescreve os métodos:

- `getNameDomesticInstance(MetricsManager metricsManager)`
Solicita à *Framework Hype* qual é a identificação da instância doméstica do próprio dispositivo.
- `onNotifySendData (MetricsManager metricsManager, JSONObject json, Instance instance)`
Notifica a *framework Hype* de que existe um teste (*json*) a enviar para um determinado destino (*instance*).

4 Experiências e testes

Neste capítulo é referente aos testes que cada vez mais representa uma parte importante do processo de desenvolvimento de *software*. Apesar dos testes não fazerem parte das práticas comuns da empresa, mesmo assim foram desenvolvidos testes unitários, funcionais, de aceitação e também de integração.

4.1 Testes unitários

O teste unitário ou de Unidade é a fase do processo de teste em que se testam as pequenas partes ou módulos do sistema, normalmente dentro de uma classe, procurando provocar falhas ocasionadas por defeitos de lógica e de implementação. O objetivo passa por encontrar falhas de funcionamento dentro de uma pequena parte do sistema independentemente do todo. Cada parte do programa é isolada e testada para validar que funciona individualmente.

Dado que este projeto contém imensa dependências entre classes não foi possível efetuar um elevado número de testes. Assim sendo foram efetuados testes as classes de apoio e de persistência conforme se pode observar pelos resultados da Figura 59 e dos excertos de código presentes no Anexo 3 – Testes unitários.

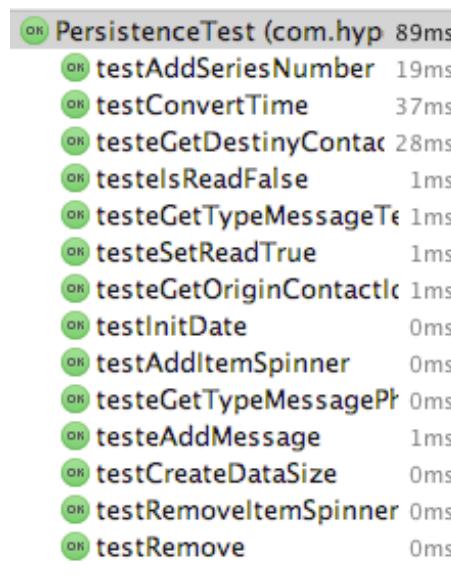


Figura 59 – Resultados dos testes unitários

4.2 Testes funcionais

Os testes funcionais são testes que permitem perceber qual o comportamento da aplicação face a uma utilização não controlada, como por exemplo o utilizador inserir letras no número de telefone. Este tipo de testes levam o programador a prever o tratamento de exceções e controlo de potenciais falhas provocadas, voluntária ou involuntariamente, pelo utilizador (Testesdesoftware.com, 2016).

O “Monkey Test Tool” é uma ferramenta que envia para a aplicação desenvolvida um determinado número de eventos que são interpretados pela aplicação como se fossem ações externas do utilizador. As ações variam desde toques no ecrã da aplicação, a toques simulados nas teclas de volume do equipamento são executadas de forma aleatória sequenciais e intensivas simulando o comportamento de um utilizador que não sabe o está a fazer no sistema.(AndroidStudio, 2017). Para este tipo de teste não é necessário efetuar qualquer alteração de código na aplicação pois esta ferramenta corre via linha de comandos, conforme se pode observar na Figura 60. De notar que para cada implementação foram efetuados testes com recurso a esta ferramenta, simulando 500 cliques em cada teste (*count=500*) e que todas as implementações obtiveram aprovação nos referidos testes.

```
MacBook-Pro-de-JSilva:platform-tools K_A_S_T_O_R$ ./adb shell monkey -p com.hypelabs.demo.bundle -v 500
:Monkey: seed=1496120476514 count=500
:AllowPackage: com.hypelabs.demo.bundle
:IncludeCategory: android.intent.category.LAUNCHER
:IncludeCategory: android.intent.category.MONKEY
// Event percentages:
//  0: 15.0%
//  1: 10.0%
//  2: 2.0%
//  3: 15.0%
//  4: -0.0%
//  5: -0.0%
//  6: 25.0%
//  7: 15.0%
//  8: 2.0%
//  9: 2.0%
// 10: 1.0%
// 11: 13.0%
```

Figura 60 - Execução de um teste funcional

4.3 Testes de aceitação

Os testes de aceitação são desenvolvidos tendo como foco os cenários de utilização do sistema com o intuito de testar funcionalidades associadas a um caso de uso, como tal e em função da implementação, é pertinente que possa existir mais que um teste para cada caso de uso. Nas tabelas abaixo são apresentados os testes efetuados para cada implementação. Para cada teste é apresentado o conjunto de critérios que devem ser cumpridos por forma a superar o teste com sucesso.

TA 01 – Despoletar inicio de alarme nos dispositivos da rede

Tabela 40 -Teste de Aceitação 01 | Implementação 02 – alarme

Referente à implementação 02 - alarme Caso de Uso 03	
Cenário:	<ul style="list-style-type: none"> Despoletar início de alarme nos dispositivos da rede
Procedimento de teste:	<ul style="list-style-type: none"> Iniciar propagação do pedido com recurso ao botão “SEND ALERT”
Critérios:	<ol style="list-style-type: none"> É iniciada a pesquisa e anúncio de dispositivos na rede É encontrado pelo menos um dispositivo na rede É enviado o “SEND ALERT” pelo dispositivo 1 É iniciada a reprodução do alarme no dispositivo 2 É enviado o “STOP ALERT” pelo dispositivo 1 É terminada a reprodução do alarme no dispositivo 2
Resultados dos critérios:	<ol style="list-style-type: none"> Cumpre Cumpre Cumpre Cumpre Cumpre Cumpre

Tabela 41 - Teste de Aceitação 02 | Implementação 03 – chat

Referente à implementação 03 - chat Caso de Uso 03	
Cenário:	<ul style="list-style-type: none"> • Iniciar comunicação com dispositivo na rede
Procedimento de teste:	<ul style="list-style-type: none"> • Enviar mensagem de texto para dispositivo na rede
Critérios:	<ol style="list-style-type: none"> 1. É iniciada a pesquisa e anúncio de dispositivos na rede 2. É encontrado pelo menos um dispositivo na rede 3. É selecionado um dispositivo disponível na rede 4. É criada uma mensagem de texto pelo dispositivo 1 5. É enviada a mensagem do dispositivo 1 para o dispositivo 2 (dispositivo selecionado anteriormente) 6. É recebida a mensagem no dispositivo 2 7. Dispositivo 2 notifica o utilizador que recebeu uma nova mensagem 8. É apresentada a mensagem no ecrã do dispositivo 2
Resultados dos critérios:	<ol style="list-style-type: none"> 1. Cumpre 2. Cumpre 3. Cumpre 4. Cumpre 5. Cumpre 6. Cumpre 7. Não cumpre 8. Cumpre

TA 03 – Iniciar comunicação com dispositivo na rede

Tabela 42 - Teste de Aceitação 03 | Implementação 04 – métricas

Referente à implementação 04 - métricas Caso de Uso 04	
Cenário:	<ul style="list-style-type: none"> • Iniciar teste que permite o cálculo de métricas de rede
Procedimento de teste:	<ul style="list-style-type: none"> • Enviar pacote de dados na rede para um dispositivo
Critérios:	<ol style="list-style-type: none"> 1. É iniciada a pesquisa e anúncio de dispositivos na rede 2. É encontrado pelo menos um dispositivo na rede 3. É selecionado um dispositivo disponível na rede 4. É introduzido um tamanho para o pacote de dados a propagar na rede 5. É introduzido um tempo limite para a execução do teste 6. É propagado o pacote de dados na rede 7. Dispositivo 1 é informado da receção do pacote no dispositivo 2 8. Dispositivo 2 reencaminha o pacote recebido para o dispositivo 1 9. Dispositivo 1 efetua os cálculos convenientes relativos ao tempo e tamanho de dados enviados/recebidos 10. Dispositivo 1 gera gráfico dos cálculos efetuados
Resultados dos critérios:	<ol style="list-style-type: none"> 1. Cumpre 2. Cumpre 3. Cumpre 4. Cumpre 5. Cumpre 6. Cumpre 7. Cumpre em função do tipo de teste selecionado pelo utilizador 8. Cumpre em função do tipo de teste selecionado pelo utilizador 9. Cumpre em função do tipo de teste selecionado pelo utilizador 10. Cumpre em função do tipo de teste selecionado pelo utilizador

4.4 Testes de sistema

Os testes de sistema pretendem testar a implementação desenvolvida como um todo na procura de falhas de utilização no uso das funcionalidades implementadas. O objetivo destes testes passa por verificar se o produto satisfaz os requisitos funcionais estabelecidos. Nas tabelas seguintes pode-se verificar alguns testes de sistema efetuados para as diferentes implementações.

TS 01 – Iniciar alarme

Tabela 43 - Teste de Sistema 01 | Implementação 02 – alarme

Referente à implementação 02 - alarme	
Descrição:	<ul style="list-style-type: none">• Criar uma solicitação de alarme, e enviar a mesma para os dispositivos na periferia
Resultado esperado:	<ul style="list-style-type: none">• Criação do alarme no dispositivo 1• Receção do alarme no dispositivo 2• Execução de som no dispositivo 2
Resultado alcançado:	Sim (*)

TS 02 – Enviar mensagem de texto

Tabela 44 - Teste de Sistema 02 | Implementação 03 – chat

Referente à implementação 03 - chat	
Descrição:	<ul style="list-style-type: none">• Enviar mensagem de texto
Resultado esperado:	<ul style="list-style-type: none">• Criação da mensagem de texto no dispositivo 1• Receção da mensagem de texto no dispositivo 2• Apresentação do conteúdo da mensagem no ecrã correspondente do dispositivo 2
Resultado alcançado:	Sim (*)

(*) O resultado satisfatório do teste está sujeito a que pelo menos uma instância seja encontrada pelo dispositivo 1

TS 03 – Enviar imagem

Tabela 45 - Teste de Sistema 03 | Implementação 03 – chat

Referente à implementação 03 - chat	
Descrição:	<ul style="list-style-type: none"> Enviar de uma imagem
Resultado esperado:	<ul style="list-style-type: none"> Acede à galeria e seleciona a imagem no dispositivo 1 Receção da imagem no dispositivo 2 Apresentação da imagem no ecrã dispositivo 2
Resultado alcançado:	Sim (*)

TS 04 – Efetuar teste de métricas

Tabela 46 - Teste de Sistema 04 | Implementação 04 – métricas

Referente à implementação 04 - métricas	
Descrição:	<ul style="list-style-type: none"> Iniciar envio de pacote de dados para testar métricas de rede
Resultado esperado:	<ul style="list-style-type: none"> Selecionar tamanho do pacote de dados no dispositivo 1 Propagação pelo dispositivo 1 do pacote de dados para o dispositivo 2 Receção do pacote de dados pelo dispositivo 2 Receber feedback da <i>Framework Hype</i> no dispositivo 1 Apresentar resultados do teste no dispositivo 1
Resultado alcançado:	Sim (*)

(*) O resultado satisfatório do teste está sujeito a que pelo menos uma instância seja encontrada pelo dispositivo 1

TS 05 – Efetuar teste de RTT

Tabela 47 - Teste de Sistema 05 | Implementação 04 – métricas

Referente à implementação 04 - métricas	
Descrição:	<ul style="list-style-type: none"> Iniciar envio de pacote de dados para testar RTT da rede
Resultado esperado:	<ul style="list-style-type: none"> Selecionar tamanho do pacote de dados no dispositivo 1 Propagação pelo dispositivo 1 do pacote de dados para o dispositivo 2 Receção do pacote de dados pelo dispositivo 2 Propagação pelo dispositivo 2 do mesmo pacote de dados para dispositivo 1 Apresentar resultados do teste no dispositivo 1
Resultado alcançado:	Sim (*)

(*) O resultado satisfatório do teste está sujeito a que pelo menos uma instância seja encontrada pelo dispositivo 1

5 Conclusões

No presente capítulo encontram-se os resultados desenvolvidos ao longo deste estágio, bem como os aspetos que se revelaram limitadores ou bloqueantes à sua realização. Pode-se ainda perceber quais os objetivos atingidos e, para o caso de existirem, quais as oportunidades a explorar. Por fim encontra-se uma visão pessoal que enquadra as experiências relativas a este estágio.

5.1 Objetivos realizados

No primeiro capítulo deste relatório foram introduzidos os objetivos para este estágio, esses objetivos passavam pela implementação de quatro aplicações para o sistema operativo Android. Com essas aplicações era pretendido demonstrar o funcionamento da Tecnologia *Hype*. Como tal foram visadas a implementação de uma aplicação para acoplar todas as outras (*bundle*), a implementação de um botão de alarme, de um *chat* de mensagens de texto e imagem e por fim uma aplicação de aquisição de métricas. O desenvolvimento das aplicações referidas tornou possível à Hype Labs demonstrar o seu produto em execução aproximando assim potenciais clientes e investidores. As vantagens da conclusão deste projeto com sucesso não foram só evidenciadas só a nível externo, com aplicação de demonstração a clientes/investidores, mas também a nível interno, dado que, com a aplicação de métricas a empresa está munida de um *software* que permite obter parâmetros mais detalhados e realistas acerca dos processos de transferência de dados entre dispositivos. Atendendo a que todas as funcionalidades previstas para cada uma das aplicações foram implementadas na totalidade e de forma funcional, conforme se pode observar pela Tabela 48, consideram-se que os objetivos propostos foram cumpridos.

Tabela 48 - Objetivos concluídos

Objetivos	Nível de conclusão
Aplicação “mãe”	Objetivo concluído
Botão de alarme	Objetivo concluído
Chat (texto, imagem)	Objetivo concluído
Métricas	Objetivo concluído

5.2 Outros trabalhos realizados

No decorrer deste estágio surgiram inúmeros desafios intermédios, no sentido de os vencer de forma satisfatória e na tentativa de acompanhar as melhores abordagens do mercado, surgiu a necessidade de efetuar alguns estados da arte. Esses documentos embora sejam bastantes simples e redigidos de forma pouco formal, permitiram enquadrar com as tecnologias e as metodologias mais recentes e desenvolvidas que o mercado dispõe para oferecer. Os mesmos podem ser consultados no Anexo 2 – Estados da arte.

5.3 Limitações e trabalho futuro

Com a constante evolução do mercado é provável que continuem a aparecer novas tecnologias de comunicação, novos sistemas operativos, novos equipamentos com diversas características e novos ecrãs. Sendo que os sistemas operativos, atendendo a sua vasta gama de versões, poderão descontinuar determinadas funcionalidades, ou até mesmo denotar problemas ao nível de compatibilidade com as versões mais antigas. Quanto aos ecrãs estes poderão causar problemas a nível da otimização do *design* implementado nas aplicações desenvolvidas, resultado das diferentes dimensões e resoluções de ecrã que surge a cada novo dispositivo. Apesar da programação em causa utilizar tecnologias já abordadas ao longo da Licenciatura em Engenharia Informática do ISEP, como é o caso da linguagem Java e XML, estudar e dominar as API's disponibilizadas pelo Android revelou ser uma tarefa difícil e que requer tempo de aprendizagem. Inicialmente grande parte do tempo disponível foi utilizado para pesquisar e aprofundar o conhecimento nesta plataforma. Relativamente ao trabalho futuro fará todo o sentido equacionar algumas melhorias e implementação de novas funcionalidades. Essas melhorias passam por implementar, no caso da aplicação de *chat*, o envio de documentos, envio de ficheiros de som (*push-to-talk*), e ainda melhorar e implementar um sistema de notificações para os eventos que ocorrem ao nível da Framework e das demos desenvolvidas. Quanto à implementação de métricas será pertinente a implementação de outros tipos de teste, nomeadamente teste de *stress* enviando recursivamente dados pela rede durante um tempo determinado pelo utilizador.

5.4 Apreciação final

Este estágio foi um marco essencial na minha curva de aprendizagem, visto que ao longo do tempo que estive inserido na organização, tive oportunidade de solidificar e reter inúmeros conhecimentos. Os conhecimentos adquiridos não se predem só a nível tecnológico como a nível de empreendedorismo e do mercado de uma empresa tecnológica permitindo assim perceber o quanto exigente este se torna. Vivenciar de perto o ótimo e pleno espirito de equipa que se vive numa *startup* onde a pressão na busca da perfeição e anciã de vencer os objetivos são uma constante, leva-me a sentir um elevado nível de satisfação com a escolha que fiz ao aceitar este estágio. Poder aplicar os conceitos aprendidos ao longo da licenciatura no mercado de trabalho, num ambiente onde as ideias e os objetivos se vão alterado com frequência e mesmo assim ter o constante apoio de uma equipe solida e motivada é de fato diferente e aliciante ao ponto de assim poder apelidar de fantásticos os últimos meses vividos nesta organização.

Bibliografia

- Alberttanure. (2012, Setembro 21). O que são Delegates (C#)? - parte 1 - Blog Framework. Obtido 12 de Junho de 2017, de <http://www.frameworksystem.com/o-que-sao-delegates-c-net1/>
- AndroidDevelopers. (2017, Junho 1). Atividades. Obtido 1 de Junho de 2017, de <https://developer.android.com/guide/components/activities.html>
- AndroidStudio. (2017, Junho 13). UI/Application Exerciser Monkey [UI/Application Exerciser Monkey]. Obtido 13 de Junho de 2017, de <https://developer.android.com/studio/test/monkey.html>
- Atlassian. (2017). What is Git: become a pro at Git with this guide | Atlassian Git Tutorial. Obtido 8 de Março de 2017, de <https://www.atlassian.com/git/tutorials/what-is-git>
- Balbo, W. (2016, Fevereiro). Conceitos do Delegate e Métodos Anônimos – Delegates, Events e Generics: Estrutura da Linguagem – Parte 1. Obtido 5 de Junho de 2017, de <http://www.devmedia.com.br/conceitos-do-delegate-e-metodos-anonimos-delegates-events-e-generics-estrutura-da-linguagem-parte-1/20060>
- Betts, T. (2017, Maio 28). Mobile Performance Testing - JSON vs XML. Obtido 28 de Maio de 2017, de <https://www.infragistics.com/community/blogs/torrey-betts/archive/2016/04/19/mobile-performance-testing-json-vs-xml.aspx>
- Bridgefy. (2017, Maio 28). Bridgefy: The SDK that makes apps work offline. Obtido 28 de Maio de 2017, de <https://bridgefy.me/learn-more.php>
- Celestino, A. (2013, Fevereiro 6). A importância dos requisitos não-funcionais. Obtido 24 de Maio de 2017, de <https://www.profissionaisti.com.br/2013/02/a-importancia-dos-requisitos-nao-funcionais/>
- Cordeiro, F. (2017, Junho 1). Aprenda a Usar os Fragments em 4 Passos. Obtido 1 de Junho de 2017, de <http://www.androidpro.com.br/fragments/>
- Developers. (2015, Agosto). Creating a Navigation Drawer. Obtido 8 de Junho de 2017, de <https://developer.android.com/training/implementing-navigation/nav-drawer.html>
- DevMedia. (2016a). Introdução a Requisitos de Software. Obtido 16 de Maio de 2017, de <http://www.devmedia.com.br/introducao-a-requisitos-de-software/29580>
- DevMedia. (2016b). Use a serialização em Java com segurança! Obtido 1 de Junho de 2017, de <http://www.devmedia.com.br/use-a-serializacao-em-java-com-seguranca/29012>

- Fellows, N. (2017). *androidplot: Charts and plots for Android*. Java. Obtido de <https://github.com/halfhp/androidplot> (Original work published 1 de Março de 2011)
- Ferreira, C. (2015). Modelos de Desenvolvimento de Software - Revista .Net Magazine 101. Obtido 29 de Junho de 2017, de <http://www.devmedia.com.br/modelos-de-desenvolvimento-de-software-revista-net-magazine-101/26747>
- Francisco, O. (2003, Outubro). Trabalhando com string: String em Java. Obtido 6 de Junho de 2017, de <http://www.devmedia.com.br/trabalhando-com-string-string-em-java/21737>
- Google. (2016). Android Studio. Obtido 3 de Abril de 2017, de <https://developer.android.com/studio/index.html>
- Google. (2017, Abril 18). Dashboards. Obtido 18 de Abril de 2017, de <https://developer.android.com/about/dashboards/index.html>
- HypeLabs. (2017, Maio 28). HypeLabs. Obtido 28 de Maio de 2017, de <https://hypelabs.io>
- IDC. (2017, Abril 12). Flat Smartphone Growth Projected for 2016 as Mature Markets Veer into Declines, According to IDC. Obtido 12 de Abril de 2017, de <http://www.idc.com/getdoc.jsp?containerId=prUS41699616>
- Infragistics. (2017). Infragistics UI Controls and Tools For Developers and UX Pros. Obtido 27 de Junho de 2017, de <https://www.infragistics.com>
- JSON. (2017). Introducing JSON. Obtido de <http://www.json.org>
- Lecheta, R. R. (2013). *Google Android - 3^a Edição: Aprenda a criar aplicações para dispositivos móveis com o Android SDK*. Novatec Editora.
- Macedo, R. (2015, Maio 28). O que é um Intent? | Android na veia. Obtido 18 de Junho de 2017, de <http://www.androidnaveia.com.br/2015/04/o-que-e-um-intent.html>
- Microsoft. (2016). Diagramas de sequência UML: referência. Obtido 29 de Junho de 2017, de <https://msdn.microsoft.com/pt-br/library/dd409377.aspx>
- Moreira, D. (2016, Fevereiro 3). O que é uma startup? | EXAME.com - Negócios, economia, tecnologia e carreira. Obtido 4 de Junho de 2017, de <http://exame.abril.com.br/pme/o-que-e-uma-startup/>
- Nunes, D. (2016, Novembro 26). Hype Labs. Comunicar com todos os objetos mesmo sem ter internet. Obtido 1 de Junho de 2017, de <https://www.dinheirovivo.pt/fazedores/hype-labs-comunicar-com-todos-os-objetos-mesmo-sem-ter-internet/>
- OpenGarden. (2017, Maio 28). Mesh networking made easy. Obtido 28 de Maio de 2017, de <http://www.opengarden.com/meshkit.html>

- Oracle. (2016). The History of Java Technology. Obtido de <http://www.oracle.com/technetwork/java/javase/overview/javahistory-index-198355.html>
- P2Pkit. (2017, Maio 28). Welcome to p2pkit. Obtido 28 de Maio de 2017, de <http://p2pkit.io/developer/>
- Reneargento. (2014, Novembro 5). Android – Managing the activity lifecycle. Obtido 1 de Junho de 2017, de <https://mobiledevnews.wordpress.com/2014/11/05/android-managing-the-activity-lifecycle/>
- Research to Action, R. to A. A. rights reserved L. (2012, Maio 1). Stakeholder Analysis: A basic introduction. Obtido 11 de Maio de 2017, de <http://www.researchtoaction.org/2012/05/stakeholder-analysis-a-basic-introduction/>
- Scrum. (2017). What is Scrum? Obtido 8 de Março de 2017, de <http://www.scrum.org/resources/what-is-scrum>
- Silveira, F. (2017, Junho 1). Activity - o que é isto? | Desenvolvendo para Android | Felipe Silveira fala sobre Android, Java e desenvolvimento de software em geral. Obtido 1 de Junho de 2017, de <http://www.felipesilveira.com.br/2010/05/activity-o-que-e-isso/>
- Testesdesoftware.com. (2016, Junho 29). Testes funcionais de software. Obtido 13 de Junho de 2017, de <http://testesdesoftware.com/testes-funcionais/>
- The Standish Group International, Inc. (2014). Big Bang Boom.
- TutorialsPoint. (2017a, Junho 1). Android Fragments. Obtido 1 de Junho de 2017, de https://www.tutorialspoint.com/android/android_fragments.htm
- TutorialsPoint. (2017b, Junho 2). Android Activities. Obtido 2 de Junho de 2017, de https://www.tutorialspoint.com/android/android_acitivities.htm
- UML. (2017a). Diagrama de Casos de Uso. Obtido 17 de Maio de 2017, de <http://www.dsc.ufcg.edu.br/~sampaio/cursos/2007.1/Graduacao/SI-II/Uml/diagramas/usecases/usecases.htm>
- UML. (2017b). WHAT IS UML. Obtido 8 de Março de 2017, de <http://www.uml.org/what-is-uml.htm>
- Vieira, D. (2014, Junho 26). Scrum: A Metodologia Ágil Explicada de Forma Definitiva. Obtido 19 de Abril de 2017, de <http://www.mindmaster.com.br/scrum/>
- W3C. (2016, Novembro 10). Extensible Markup Language (XML). Obtido 8 de Março de 2017, de <http://www.w3.org/XML/>

Anexos

Anexo 1 - Reuniões de acompanhamento

Supervisor da Hype Labs

Nas tabelas seguintes são apresentadas as reuniões efetuadas entre o estagiário e o orientador na organização. Em cada reunião foi efetuado um pequeno registo de forma a ter uma linha orientadora das atividades desenvolvidas. Para tal foi assumida uma metodologia que se resume na divisão da reunião em três momentos, o ponto de situação neste momento (representado pela letra “S” – *Situation*), o que irá ser desenvolvido hoje ou até à próxima reunião (representado pela letra “T” - *Today*), o que está em atraso, entenda-se aquilo que surgiu e que não estava previsto ser assim (representado pela letra “B” - *Backlog*).

Tabela 49 - Reuniões com o supervisor da Hype Labs

Reunião	S/T/B	Assuntos tratados	Data
Reunião 1	S.	- Estudo de padrões de desenho (MVC) em Android - Convenções em java + Android - Testes convenientes em Android	17/02/17
	T.	- Convenções em java + Android	
	B.	- n/a	
Reunião 2	S.	- Comunicação entre <i>activity's</i> a funcionar com utilização de serialize ou parcelable - SOA acerca da criação de UI	21/02/17
	T.	- SOA UI em desenvolvimento	
	B.	- n/a	
Reunião 3	S.	- Reunião para inicio de <i>sprint</i> Seg. 13 de Março - Estrutura do relatório em desenvolvimento - Experimentação com as tecnologias	07/03/17
	T.	- Revisão da estrutura atual do relatório	
	B.	- n/a	

<i>Reunião</i>	<i>S/T/B</i>	<i>Assuntos tratados</i>	<i>Data</i>
Reunião 4	S.	- Progressos a nível de relatório - Alterações estruturais revistas ontem	08/03/17
	T.	- Revisão das <i>timelines</i> e planeamento - Início de arquitetura da solução	
	B.	- n/a	
Reunião 5	S.	- Planeamento concluído - Rever formatações - Efetuadas algumas notas para revisão de conteúdo	13/03/17
	T.	- Início da implementação 01 – configurações - Notas de configurações iniciais, por exemplo, <i>namespaces</i> sem estarem de acordo com as especificações da empresa.	
	B.	- Corrigir <i>namespaces</i> - Corrigir Convenções de nomes	
Reunião 6	S.	- <i>Hamburguer button</i> concluído - Navegação entre janelas concluída - <i>Fragments</i> criados	20/03/17
	T.	- Início da implementação 02 - alarme - Integração da <i>Framework Hype</i> - SOA protocolo de comunicação Json	
	B.	- Considerar <i>assets</i> ²⁵ - Configurar projeto para receber a <i>framework</i> - Importar assets	

²⁵ RECURSOS DO PROJETO (IMAGENS/ÍCONES /ETC.)

<i>Reunião</i>	<i>S/T/B</i>	<i>Assuntos tratados</i>	<i>Data</i>
Reunião 7	S.	<ul style="list-style-type: none"> - SOA protocolo de comunicação concluído - Utilização de um gerador de <i>assets</i> - Demonstração de alarmes da implementação 02 – alarme - Várias correções apontadas 	21/03/17
	T.	<ul style="list-style-type: none"> - SOA de assets - Desenvolvimento da implementação 02 – alarme 	
	B.	<ul style="list-style-type: none"> - Revisão das correções apontadas 	
Reunião 8	S.	<ul style="list-style-type: none"> - Revisão das correções apontadas ontem em desenvolvimento - Implementações de UI relativas aos contadores - Revisão de algumas correções ao nível de nomenclaturas 	22/03/17
	T.	<ul style="list-style-type: none"> - Revisão das correções apontadas 	
	B.	<ul style="list-style-type: none"> - n/a 	
Reunião 9	S.	<ul style="list-style-type: none"> - Demonstração funcional - Falhas funcionais detetadas 	23/03/17
	T.	<ul style="list-style-type: none"> - Revisão de paradigmas de <i>model</i> e <i>controller</i>. 	
	B.	<ul style="list-style-type: none"> - n/a 	

<i>Reunião</i>	<i>S/T/B</i>	<i>Assuntos tratados</i>	<i>Data</i>
Reunião 10	S.	- Revisão de nomenclaturas para métodos imperativos - Revisões relativas à documentação	28/03/17
	T.	- SOA de testes unitários em Android até ao final da semana para terminar	
	B.	- Equacionar <i>copyright</i> (MIT)	
Reunião 11	S.	- Testes funcionais não são adequados á implementação por ser demasiado simples - Testes unitários não parecem adequados dado que as operações se executam praticamente todas a nível de UI - SOA testes unitários em desenvolvimento	03/04/17
	T.	- <i>Timeline</i> atrasada em um dia - Início da implementação 03 - chat	
	B.	- n/a	
Reunião 12	S.	- SOA de testes unitários concluído e depreende-se que não são aplicáveis à solução desenvolvida dada a sua simplicidade - Implementação 03 - chat em desenvolvimento	06/04/17
	T.	- Desenhar <i>model</i> e <i>controller</i>	
	B.	- n/a	

<i>Reunião</i>	<i>S/T/B</i>	<i>Assuntos tratados</i>	<i>Data</i>
Reunião 13	S.	<ul style="list-style-type: none"> - Implementação 03 – chat em desenvolvimento - Apresentação do <i>design</i> da solução - Implementação da UI - Revisões à arquitetura do <i>model</i> e <i>controller</i> - Progressos no relatório e implementações 	17/04/17
	T.	<ul style="list-style-type: none"> - Fechar arquitetura <i>model</i> e <i>controller</i> - Fechar UI 	
	B.	<ul style="list-style-type: none"> - n/a 	
Reunião 14	S.	<ul style="list-style-type: none"> - Revisão de artefactos UML (modelação) - SOA de UI - Ajustes à abordagem efetuada. A implementação da UI estaria a ser manual e foi alterada para de tecnologias existentes 	02/05/17
	T.	<ul style="list-style-type: none"> - SOA de UI 	
	B.	<ul style="list-style-type: none"> - n/a 	
Reunião 15	S.	<ul style="list-style-type: none"> - SOA de UI concluída - <i>Frameworks</i> existentes não são suficientes ou suficientes adaptáveis para o foco da aplicação e empresa - Conclusão: regresso à implementação manual da UI - Trabalho futuro: <i>framework</i> de <i>chat</i> UI desacoplada - Implementação de <i>layout</i> de balões à direita 	04/05/17
	T.	<ul style="list-style-type: none"> - Implementação de <i>layout</i> de balões à esquerda - Implementação da UI 	
	B.	<ul style="list-style-type: none"> - n/a 	

<i>Reunião</i>	<i>S/T/B</i>	<i>Assuntos tratados</i>	<i>Data</i>
Reunião 16	S.	<ul style="list-style-type: none"> - Implementação de <i>layout</i> de balões concluída (já com comunicação a funcionar) - Implementações de UI concluídas 	9/05/17
	T.	<ul style="list-style-type: none"> - Atualizar a UI quando é utilizado o botão físico <i>back</i> - <i>Possíveis erros a corrigir (testar)</i> 	
	B.	<ul style="list-style-type: none"> - Documentação - Testes funcionais e de aceitação 	
Reunião 17	S.	<ul style="list-style-type: none"> - Erros detetados anteriormente corrigidos - Possível exaustão de memória detetada - Suporte para imagens em <i>Base64</i> suportada - Erros de persistência detetados 	18/05/17
	T.	<ul style="list-style-type: none"> - Corrigir erros 	
	B.	<ul style="list-style-type: none"> - n/a 	
Reunião 18	S.	<ul style="list-style-type: none"> - Início da implementação 04 – métricas - Desenhar <i>model</i> e <i>controller</i> 	07/06/17
	T.	<ul style="list-style-type: none"> - Desenhar <i>model</i> e <i>controller</i> 	
	B.	<ul style="list-style-type: none"> - n/a 	
Reunião 19	S.	<ul style="list-style-type: none"> - Desenho do <i>model</i> e <i>controller</i> concluído - Implementação 04 – métricas em desenvolvimento 	13/06/17
	T.	<ul style="list-style-type: none"> - Utilização de tecnologias existentes para criação de gráficos 	
	B.	<ul style="list-style-type: none"> - n/a 	

<i>Reunião</i>	<i>S/T/B</i>	<i>Assuntos tratados</i>	<i>Data</i>
Reunião 20	S.	<ul style="list-style-type: none"> - Evoluções ao relatório com duas versões enviadas para revisão da orientadora do ISEP - Testes de RTT concluídos - Testes de métricas em desenvolvimento 	20/05/17
	T.	<ul style="list-style-type: none"> - Gerar gráficos com valores obtidos pelos testes - Possíveis erros a corrigir (testar) 	
	B.	- n/a	

Supervisor do ISEP

Na Tabela 50 constam os resumos das tabelas realizadas entre o aluno e o orientador do ISEP.

Tabela 50 - Reuniões com o supervisor do ISEP

<i>Reunião</i>	<i>Assuntos tratados</i>	<i>Data</i>
Reunião 1	<ul style="list-style-type: none"> - Apresentação da empresa (Hype Labs) - Explicação dos objetivos para o estágio - Esclarecimento de <i>timings</i> - Assinatura de ata 	07/03/17
Reunião 2	<ul style="list-style-type: none"> - Estado atual do projeto (aplicação) - Demonstração da aplicação em execução - Esclarecimento de abordagens relativas ao relatório - Agendamento de revisão de relatório 	25/05/17

Anexo 2 – Estados da arte

- SOA @ UI Android

SOA @ UI [1]- Android

View é uma área da tela que mostra conteúdos, pode ser um botão, um pedaço de texto, uma imagem, um ícone, etc.. uma ou mais juntas formam o Layout da aplicação que estamos a criar.

Em suma, tudo aquilo que vemos ou interagimos numa aplicação é chamado interface de utilizador ou UI (do inglês, *User Interface*).

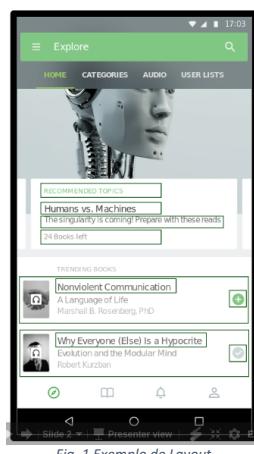


Fig. 1 Exemplo de Layout

✓ FUNCIONAMENTO DE UMA VIEW

Uma view é formada internamente por código XML, ou seja, utilizamos a sintaxe dessa tecnologia para criar os componentes Android.

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">

    <TextView
        android:text="Olá Androideiro!"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="24sp" />

    <TextView
        android:text="Bora criar apps"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="24sp" />

</LinearLayout>
```

Fig. 2 Código XML

Na da Fig.2 podemos ver um LinearLayout que contém duas TextViews, às quais chamamos de elementos Filho, e ao LinearLayout chamamos de elemento Pai. Uma sequencia de código XML começa e termina com o carater <,> (maior e menor) respetivamente, seguido de uma Tag que pode ou não conter atributos (elementos adicionais dentro da Tag) que conferem a aparência pretendida ao ser executada no Android.

- `android:text="HypeLabs!"`
- `android:background="@android:color/blue"`
- `android:layout_width="150dp"`
- `android:layout_height="750dp"`

Analisando os atributos, vemos a vermelho e à esquerda do operador “=” (igual) o nome do atributo que se inicia por “`android:`”, à direita escrito a verde, temos os valores referentes ao atributo sempre escritos dentro de “” (aspas).

✓ PRINCIPAIS VIEW NO ANDROID

- **TextView** – Exibir texto
- **ImageView** – Exibir imagens
- **Button** – Utilizado para executar uma ação ao ser clicado
- **ImageButton** – Exibe uma imagem com comportamento de um botão
- **EditText** – Input de texto
- **ListView** – Lista de itens que contém outras Views

✓ TIPOS DE LAYOUT'S

- Linear Layout
 - O layout linear é dividido em layout horizontal e vertical. Isso significa que pode organizar View's numa única coluna ou numa única linha.
- Absolute Layout
 - O AbsoluteLayout permite especificar a localização exata dos filhos.
- Table Layout
 - O TableLayout agrupa exibições em linhas e colunas Criamos uma linha com `<TableRow>` em que cada view dentro do TableRow representa uma coluna.
- Frame Layout
 - O FrameLayout é um espaço reservado no ecrã que pode ser usado para exibir uma única View.
- Relative Layout
 - O RelativeLayout permite especificar como as View Filho são posicionadas em relação umas às outras.
- Grid Layout
 - Permitir posicionar as Views numa disposição idêntica a uma grelha. Basicamente consiste num número de linhas de horizontais e verticais que dividem a visualização do layout em forma de “grelha”, com cada linha e coluna formando uma célula que pode conter uma ou mais Views.

As linhas e colunas são definidas utilizando os atributos **android:columnCount** e **android:rowCount**, define o numero de linhas e colunas respetivamente.

Cada linha do GridLayout é referenciada por **índices**, que são numerados a partir de **0** contando de baixo para cima. As **células** (linha/coluna) também tem numeração e começam em **0** a partir da célula no canto superior esquerdo do GridLayout.

É possível definir em qual linha e coluna que cada **View** vai ficar utilizando os atributos **android:layout_column** e **android:layout_row**, onde podemos dizer ao **GridLayout** a posição exata de cada componente dentro da grelha.

Posicionamento de uma View dentro da célula do GridLayout é dado por **android:layout_gravity** que pode receber como atributo (left/right/top/bottom/fill [preenche toda a célula]/center)

✓ ATRIBUTOS MAIS COMUNS

○ **Layout_width**

Especifica a largura do View ou ViewGroup

○ **Layout_height**

Especifica a altura do View ou ViewGroup

○ **Layout_marginTop**

Especifica espaço extra na parte superior do View ou ViewGroup

○ **Layout_marginBottom**

Especifica espaço extra na parte inferior do View ou ViewGroup

○ **Layout_marginLeft**

Especifica espaço extra no lado esquerdo do View ou ViewGroup

○ **Layout_marginRight**

Especifica espaço extra no lado direito do View ou ViewGroup

○ **Layout_gravity**

Especifica como as Vistas filho são posicionadas

○ **Layout_weight**

Especifica a quantidade de espaço extra no layout deve ser alocada para a View

✓ VALORES DE ATRIBUTOS MAIS COMUNS

PARA DEFINIR ALTURA E/OU LARGURA

Wrap_content – Ajusta-se ao conteúdo utilizado dentro da View

Match_parent – Expande para o tamanho disponível no layout pai

PARA ACRESCENTAR ESPAÇAMENTOS:

android:padding – Espaço dentro da view/viewGroup entre a borda e o conteúdo

android:layout_margin – Espaço entre a parte de fora e os outros elementos próximos da View

android:textSize – Tamanho do texto

android:color/black – Cor do texto ou do background, aceita hexadecimal ou o nome da cor.

✓ LISTVIEWS

O ListView é uma View que vai constituir um determinado layout de uma tela.

Assim como declaramos um Button, TextView, EditText, ou qualquer outro componente gráfico no Layout XML, também declaramos o ListView, que é, possivelmente, o componente de UI mais usado no desenvolvimento de APPs, permite a listagem de dados, p.e. a Lista de contactos.

✓ RECYCLEVIEW

O RecyclerView, surge como uma melhoria ao ListView e ao GridView, é também utilizado para listar dados para o utilizador, no entanto com algumas melhorias ao nível da performance e velocidade de execução, permitindo ainda mais possibilidades ao programador.

O RecyclerView, obriga a implementar o padrão ViewHolder que impede a invocação de métodos findViewById desnecessariamente.

- Internamente tem um nível de abstração maior entre o modelo (lista de dados) e view (o Adapter), ações como animações não ficam a cargo do Adapter deixando-o assim com maior performance.
- Resumindo oferece mais possibilidades ao programador e é mais rápido.

✓ RECICLAGEM DE VIEWS

Surge como solução com vista a otimização da memória.

Com estas, podemos mostrar listas longas dentro das App reutilizando as Views que não estão visíveis. Para permitir esta técnica necessitamos de um Adapter.

Adapter:

Responsável por gerir e adaptar os dados nas Views, e por criar uma View para cada item do conjunto de dados a ser “impresso”.

Exemplos de adapters fornecidos pelo Android, e por sua vez, os mais importantes, *SimpleCursorAdapter*, *ArrayAdapter* e *CursorAdapter*.

ArrayAdapter: Usado para manipular estruturas baseadas em Arrays ou java.util.List
SimpleCursorAdapter e CursorAdapter: Utilizado para manipular dados oriundos de Base de Dados Android SQLite.

Métodos para fazer atualização de dados e notificar as ListViews que algo mudou:
notifyDataSetChanged(): É chamado se os dados tiverem sido alterados ou se houver novos dados disponíveis.

notifyDataSetInvalidated(): É chamado se os dados não estiverem mais disponíveis.

[MVC em Android:](#)

O padrão Model, View, Controller, é um padrão de fácil implementação que permite dividir em camadas fases as funcionalidades do software.

A camada *Model*, é utilizada para conferir informações com mais detalhe, sendo assim utilizada sempre que é necessário efetuar operações relacionadas à camada de negócios.

A camada *View*, utilizada para interagir com o usuário, é responsável por tudo o que ele visualiza, esta camada não deve conter regras de negócio, visto que isso é responsabilidade da camada *Model*.

A camada *Controller*, utilizada para gerir o fluxo de informação com origem na *Model* e passando-a até a *View* para que o usuário visualize. [1]

[NOTAS:](#)

onCreateView – Apresenta o layout definido no file .xml, guardado em R.layout.nome

[FONTES:](#)

Fonte: <http://www.devmedia.com.br/linear-table-e-relative-layouts-com-android-studio/34127>

Fonte: <http://www.devmedia.com.br/android-layouts-aprendendo-tecnicas-de-layout-no-android/30790>

Fonte: <http://www.androidpro.com.br/android-layouts-viewgroups-intro/>

Fonte: <http://www.androidpro.com.br/android-views-intro/>

- SOA @ Network Test

SOA Testes de Network

PING

(Packet InterNet Grouper) criado por Michael Muuss o nome surge da analogia ao som de um sonar. O motivo dessa analogia veio do fato do Ping agir de forma semelhante a um sonar, porém, com foco no mundo virtual. Com este comando, o computador é capaz de medir quantos milissegundos (ms) um pacote de informações leva para chegar a um destino e voltar. e forma simples, quanto menor o valor que ele retornar, mais rápida é a ligação.

O teste de *download* consiste em transferir um conjunto de dados, gerados de forma aleatória, do servidor, com ligação de alto débito, para o computador do utilizador. O teste de *ping*, ou latência, consiste em efetuar o envio e receção de mensagens e verificar os tempos de resposta.

O teste de *upload* consiste em transferir um conjunto de dados, gerados de forma aleatória, para servidor, com ligação de alto débito, do computador do utilizador.

Throughput

Throughput é a medida do número de mensagens que um sistema pode processar num determinado período de tempo. No software, o throughput é geralmente medido em "requests / second" ou "transactions / second".

- Iniciar envio de ficheiros com o envio de um ficheiro de 10Kb para evitar low-start
- Usar transferências que demorem entre 10 a 20 segundos de forma a ter uma largura de banda media.

FONTES:

<https://stackoverflow.com/questions/23764826/bandwidth-speed-testing>

<https://stackoverflow.com/questions/13709118/java-is-there-a-way-to-calculate-the-download-throughput-over-a-specific-inter>

<http://www.javidamae.com/2005/06/20/calculating-throughput-and-response-time/>

Casos de uso:

- Utilizador define tamanho do pacote de informação a enviar
- Utilizador define tempo limite para o envio do pacote de informação
- Utilizador define tecnologia a usar no teste
- Utilizador inicia o teste com tamanho por defeito (*)

(*)

“Teste por defeito”: envio de pacote de informação de diferentes tamanhos
(iniciar com 10Kb, e ir duplicando o tamanho automaticamente)

- SOA @ Testes Unitários

SOA Testes unitários

O QUE SÃO?

Imagine por exemplo, se um avião só fosse testado após a conclusão da sua construção, com certeza isso seria um verdadeiro desastre, é nesse aspecto que a engenharia aeronáutica é uma ótima referência em processos de desenvolvimento de projetos de software, principalmente em sistemas de missão crítica, pois durante a construção e montagem de um avião todos os seus componentes são testados isoladamente até a exaustão, e depois cada etapa de integração também é devidamente testada e homologada.

O teste unitário, baseia nessa ideia, pois são modalidade de testes que se concentra na verificação do projeto de software. É realizado o teste de uma unidade lógica, com uso de dados suficientes para se testar apenas à lógica da unidade em questão.

Em sistemas construídos com uso de linguagens orientadas a objetos, essa unidade pode ser identificada como um método, uma classe ou mesmo um objeto.

PORQUE?

Lista dos principais fatores de motivam o uso sistemático e continuo de testes unitários:

- Prevenir o aparecimento de BUG's com origem em código mal escrito
- Elevar a confiança no código
- Testar situações de sucesso e de falha
- Funciona como métrica do projeto (teste == requisitos)
- Cria e preserva um conhecimento aprofundado sobre as regras de negocio do projeto

QUANDO?

Continuamente...

No inicio, projetar e escrever as classes de testes,

Como ?

O JUnit é a plataforma mais popular para gestão e execução de testes unitários em Java.

Conhecido pela sua elevada utilização na industria, e por ser possível utiliza-lo na linha de comando ou integrado no próprio IDE. JUnit pode ser utilizado para testar, um objeto, parte de um objeto, um ou mais métodos ou a interação entre vários objetos.

Existe documentação que propõem o uso de bibliotecas para execução de testes, tais como: Espresso, Monkey, UI Automator, Android JUnitRunner, Robotium, Appium, etc..

NOTAS:

- Em JUnit4, a classe de teste não precisa ser instanciada
- Em Junit4, o nome do método não precisa de iniciar por “test”
- Package 1 – Código
- Package 2 (Android Test) – Testes de instrumentação UI
- Package 3 – Testes unitários

Testes de Interação (Funcionais) utilizando Espresso:

<https://www.youtube.com/watch?v=LnjzoHGAYe0>

Testes unitários com JUnit3:

<https://www.youtube.com/watch?v=OLID9D5kCbk>

<http://www.vogella.com/tutorials/AndroidTesting/article.html>

Anexo 3 – Testes unitários

```
@Test
public void testInitDate() {
    String date = "sem data";
    date = Utils.initDate();

    assertEquals(date, "sem data");
}

@Test
public void testRemove() {
    Persistence p = new Persistence();
    p.addDomainLabels(123);
    int size = p.getDomainLabels().size();

    assertEquals(size, 1);
}

@Test
public void testeAddMessage() {
    Persistence p = new Persistence();
    Message m1 = new TextMessage("ola", "1", "3");
    p.addMessage("a", m1);
    int size = p.getMessages().size();

    assertEquals(size, 1);
}

@Test
public void testeGetDestinyContactIdentifierOfMessage() {
    Persistence p = new Persistence();
    Message m1 = new TextMessage("ola", "origin", "destin");
    p.addMessage("a", m1);

    String dest = m1.getDestinyContactIdentifier();

    assertEquals(dest, "destin");
}
```

```
@Test
public void testeGetOriginContactIdentifierOfMessage() {
    Persistence p = new Persistence();
    Message m1 = new TextMessage("ola", "orig", "destin");
    p.addMessage("a", m1);

    String orig = m1.getOriginContactIdentifier();
    assertEquals(orig, "orig");
}

@Test
public void testeGetTypeMessageText() {
    Persistence p = new Persistence();
    Message m1 = new TextMessage("ola", "orig", "destin");
    p.addMessage("a", m1);

    int type = m1.getMessageType();
    assertEquals(type, MessageType.TEXT.getValue());
}

@Test
public void testeGetTypeMessagePhoto() {
    Persistence p = new Persistence();
    Bitmap bm = null;
    Message m1 = new PhotoMessage(bm, "orig", "dest");
    p.addMessage("a", m1);

    int type = m1.getMessageType();
    assertEquals(type, MessageType.PHOTO.getValue());
}

@Test
public void testeIsReadFalse() {
    Persistence p = new Persistence();
    Bitmap bm = null;
    Message m1 = new PhotoMessage(bm, "orig", "dest");
    p.addMessage("a", m1);

    boolean isRead = m1.isRead();
    assertEquals(isRead, false);
}

@Test
public void testeSetReadTrue() {
    Persistence p = new Persistence();
    Bitmap bm = null;
    Message m1 = new PhotoMessage(bm, "orig", "dest");
    p.addMessage("a", m1);
    m1.setRead(true);
    boolean isRead = m1.isRead();

    assertEquals(isRead, true);
}

@Test
public void testCreateDataSize() {
    String a = com.hypelabs.demo.bundle.metrics.Utils.createDataSize(10);
    int size = a.length();

    assertEquals(size, 10);
}
```

```
    @Test
    public void testConvertTime() {
        long time = 123456789;
        String res = com.hypelabs.demo.bundle.metrics.Utils.convertTime(time);
        assertEquals(res, "10:17:36.789");
    }

    @Test
    public void testAddSeriesNumber() {
        Persistence p = new Persistence();
        p.addSeriesNumber(123);
        int size = p.getSeriesNumber().size();
        assertEquals(size, 1);
    }

    @Test
    public void testAddItemSpinner() {
        Persistence p = new Persistence();
        p.addItemSpinner("test1");
        p.addItemSpinner("test2");
        int size = p.getListSpinners().size();
        assertEquals(size, 2);
    }

    @Test
    public void testRemoveItemSpinner() {
        Persistence p = new Persistence();
        p.addItemSpinner("test");
        p.removeItemSpinner("test");
        int size = p.getListSpinners().size();
        assertEquals(size, 0);
    }
```

Anexo 4 - Planeamento

The cover page features a large orange background with a white diagonal stripe. At the top, the text "Projetos/Estágios" and "Ano letivo de 2016-2017" is centered in blue. Below this, in the center, is the title "PONTO DE SITUAÇÃO" in bold black capital letters. Underneath it, the project number "PROJETO Nº: 20172192" and student number "ALUNO Nº: 1091114" are listed. To the left, "ORIENTADOR (docente): Eng. Goreti Marreiros" and "ORGANIZAÇÃO: Hype Labs" are mentioned. To the right, "SUPERVISOR (Organiz.): Eng. José Teixeira" is listed. In the bottom left corner, there is a logo for "25 ANOS DEPARTAMENTO DE INFORMÁTICA" and in the bottom right corner, the ISEP logo with the text "Instituto Superior de Engenharia do Porto".

25
ANOS
DEPARTAMENTO DE INFORMÁTICA

PONTO DE SITUAÇÃO

PROJETO Nº: 20172192
ALUNO Nº: 1091114

ORIENTADOR (docente): Eng. Goreti Marreiros
ORGANIZAÇÃO: Hype Labs
SUPERVISOR (Organiz.): Eng. José Teixeira

LEI-ISEP

iseplnstituto Superior de
Engenharia do Porto

The main content page has a similar layout to the cover page, with an orange background and a white diagonal stripe. The title "Projetos/Estágios 2016-2017" is at the top in blue. Below it, under the heading "■ Problema a resolver", there is a bulleted list: "– Software mobile que permita a visualização da tecnologia Hype em funcionamento." In the bottom left corner, there is a logo for "25 ANOS DEPARTAMENTO DE INFORMÁTICA" and in the bottom right corner, the ISEP logo with the text "Instituto Superior de Engenharia do Porto".

25
ANOS
DEPARTAMENTO DE INFORMÁTICA

Projetos/Estágios 2016-2017

■ Problema a resolver

– Software mobile que permita a visualização da tecnologia Hype em funcionamento.

LEI-ISEP

iseplnstituto Superior de
Engenharia do Porto

Projetos/Estágios 2016-2017

■ Solução pretendida

- Desenvolvimento de uma aplicação mobile (Android) que permita a execução da tecnologia criada pela empresa evidenciando as suas funcionalidades.

25
ANOS
DEPARTAMENTO DE INGENHARIA INFORMÁTICA

LEI-ISEP

isept Instituto Superior de
Engenharia do Porto 3

Projetos/Estágios 2016-2017

■ Análise e implementação da solução

- Integração com a tecnologia Hype
- Aprendizagem de paradigmas de programação mobile (Android)
- Pesquisa sobre Estado da Arte Android

25
ANOS
DEPARTAMENTO DE INGENHARIA INFORMÁTICA

LEI-ISEP

isept Instituto Superior de
Engenharia do Porto 4

Projetos/Estágios 2016-2017

■ Execução da implementação

- Neste momento está a decorrer a implementação da aplicação 04, (4/5)
- A *timeline* apresentada no planeamento está a ser cumprida sem atrasos (disponível no documento de planeamento)



Projetos/Estágios 2016-2017

■ Problemas surgidos no projeto

- Não foram efetuadas alterações ao planeamento inicial.
- Os problemas surgidos tem sido relacionados com os paradigmas específicos do Android, que rapidamente tem sido ultrapassados graças ao excelente acompanhamento e suporte dado pelo CTO e orientador na empresa.



Projetos/Estágios 2016-2017

■ Observações

- Projeto aliciante tendo em conta as potencialidades da tecnologia Hype cujas principais características são,
 - Contacto com dispositivos na proximidade
 - Comunicação offline e gratuita entre pessoas
- Quanto à aplicação em desenvolvimento, existe potencial futuro no que diz respeito à apresentação da tecnologia a clientes em reuniões/feiras/exposições/etc.

