

**Aplicação YAMA**

**(Yet Another Messaging Application)**

**Alunos**

Luís Gordete, nº 33685

Jorge Oliveira, nº 36929

Samuel Costa, nº 43552

**Professor**

Eng.º João Trindade

Relatório realizado no âmbito da disciplina de Programação de Dispositivos Móveis,  
do curso de Licenciatura em Engenharia Informática e de Computadores  
Semestre de Inverno 2018/2019

Dezembro de 2018

# Índice

[Índice ii](#_Toc532148097)

[Lista de Figuras ii](#_Toc532148098)

[1. Introdução 1](#_Toc532148099)

[2. Arquitetura da aplicação 2](#_Toc532148100)

[3. Descrição da solução proposta 4](#_Toc532148101)

[4. Avaliação experimental 5](#_Toc532148102)

[5. Conclusão 7](#_Toc532148103)

# Lista de Figuras

[Figura 1 - Arquitetura da aplicação (Inspirado no guia para arquitetura de aplicações) 2](#_Toc532148104)

[Figura 2 – fluxo de navegação dentro da aplicação 3](#_Toc532148105)

[Figura 3 – ecrã de login (primeira utilização) 5](#_Toc532148106)

[Figura 5 – Detalhe de perfil do utilizador activo 5](file:///D:\18_19SI\PDM\Projecto\1819i-LI51N-G01\Relatorio\Relatorio_PDM_1819i_LI51N_G01_v2.docx#_Toc532148107)

[Figura 4 – Lista de equipas 5](file:///D:\18_19SI\PDM\Projecto\1819i-LI51N-G01\Relatorio\Relatorio_PDM_1819i_LI51N_G01_v2.docx#_Toc532148108)

[Figura 6 – Lista de membros de uma equipa 6](file:///D:\18_19SI\PDM\Projecto\1819i-LI51N-G01\Relatorio\Relatorio_PDM_1819i_LI51N_G01_v2.docx#_Toc532148109)

[Figura 7 – sala de chat de uma equipa 6](file:///D:\18_19SI\PDM\Projecto\1819i-LI51N-G01\Relatorio\Relatorio_PDM_1819i_LI51N_G01_v2.docx#_Toc532148110)

1. Introdução

Este relatório foi elaborado no âmbito do trabalho prático da disciplina de Programação de Dispositivos Móveis e acompanha a implementação de uma aplicação Android de mensagens dentro de equipas de desenvolvimento de software. A aplicação desenvolvida é compatível com dispositivos desde a API 23 a 28.

Ao longo do documento apresenta-se a exposição da arquitetura da aplicação, com a explicação da forma como os módulos interagem entre si; a descrição da solução proposta; a avaliação experimental; e por fim, uma breve reflexão em que se discutem os resultados obtidos, explicitando-se conclusões referentes ao trabalho desenvolvido.

Este trabalho é realizado na sequência da introdução de componentes de software inserida no *Android Jetpack* e procura viver dentro desse contexto. Peças como ViewModel, WorkManager, Room, Livedata, Lifecycles, gestão de permissões e notificações foram apresentadas pela *Google* como “a próxima geração de componentes, ferramentas e instruções de arquitetura para acelerar o desenvolvimento de aplicativos Android.”[[1]](#footnote-1) Estas alterações visam tornar o desenvolvimento de software mais fácil para o programador, uma vez que adicionamcamadas de abstração na realização de tarefas como o agendamento de tarefas periódicas, a persistência de dados da aplicação, etc., que podem ser testadas independentemente.

Em virtude de se viver um momento de transformação na plataforma, a documentação atualizada é escassa, pelo que foram úteis na realização do trabalho e do relatório os exemplos fornecidos na aula da disciplina e o guia para arquitetura da aplicação[[2]](#footnote-2).

O trabalho foi elaborado em duas fases: na primeira foi criada a estrutura base de *Activities*, *ViewModels* e *Repository* que permitiu organizar o código escrito e dividir responsabilidades. Ainda nesta fase, foi feita a ligação à API GitHub, através da biblioteca *Volley*, para obtenção dos dados das equipas e dos seus membros. Já na segunda fase do trabalho a aplicação passou a suportar persistência de dados através da biblioteca *Room*, usando a base de dados SQL Lite, sincronizada com dados remotos via API GitHub e coleções *Firestore*.

1. Arquitetura da aplicação

Considere-se o seguinte diagrama, que apresenta a organização dos módulos que compõem a aplicação.



Figura 1 - Arquitetura da aplicação (Inspirado no guia para arquitetura de aplicações)

A aplicação está organizada por camadas. Cada camada depende da que lhe está diretamente abaixo.Uma *Activity* define o layout do ecrã apresentado ao utilizador e regista-se para observar mudanças em *ViewModel*. Eventualmente faz sentido que uma *Activity* propague os dados resultantes de interação com o utilizador no *ViewModel* associado como por exemplo no Login inicial. É garantido que, para várias instâncias da mesma *Activity* haja apenas uma instância de *ViewModel* correspondente.

A aplicação contém uma *activity* de login, que é usada apenas quando não existem dados associados ao utilizador armazenados em *shared preferences*, uma *Activity main* composta por dois fragmentos, referentes ao ecrã que lista as equipas da organização e ao ecrã de detalhe do perfil do utilizador ativo, uma *activity* de detalhe da equipa que contem os membros dessa equipa a qual permite navegar para o ecrã de chat. O fluxo de navegação dentro da aplicação é apresentado na Figura 2.



Figura 2 – fluxo de navegação dentro da aplicação

Sobre os ciclos de vida das activities, acresce dizer que é no método onResume da activity main (fragmento *TeamFragment*) que se procede ao registo da observação da lista de equipas e seu carregamento, pois se verificou que se se fizesse tal operação no método onCreate, as lista de equipas não era carregada corretamente depois da navegação para a activity Login, não chegando a ser carregada também quando não existem credenciais válidas.

Uma instância de *ViewModel* contém como campos as instâncias de *LiveData* necessárias para modelar os dados associados a um ecrã. Estas instâncias serão observadas (pelo observe declarada na Activity desse ViewModel) e sempre que existem alterações serão apresentadas ao utilizador automaticamente.

O repositório é um contentor para a informação ser disponibilizada a *ViewModel* e existe um repositório por aplicação. Por forma a diminuir a complexidade existente no repositório, foi criada uma classe *TeamRepository* e *MemberRepository*. O repositório geral contém uma instância de cada uma destas classes.

As entidades a modelar foram:

- Equipa, identificável univocamente por um id;

- Utilizador enquanto membro de uma equipa, caracterizado também por um id único;

- Mensagem, identificável por um campo texto, o identificador do remetente e o identificador da equipa, por forma a poder obter-se todas as mensagens de uma equipa.

Os dados da aplicação persistem localmente por recurso a base de dados *SQL lite através da biblioteca Room*.

A sincronização com a fonte de dados remota é feita uma vez por dia para a lista de equipas e para a lista de membros de uma equipa através de um Worker. Da primeira vez que o utilizador pede a lista de grupos e a lista dos membros da sua organização por ter navegado para essa vista da aplicação, é colocado na *queue* do *WorkManager* um *WorkRequest* periódico – TeamWorker e *MemberWorker*, configurado para que a tarefa corra uma vez por dia. O identificador desse pedido é colocado nas *Shared Preferences* de modo que não sejam multiplicadas as configurações de WorkRequest para a mesma tarefa, nem os pedidos à rede.

De cada vez que é enviada uma mensagem para um dado chat, os utilizadores que se registaram para obter notificações relativas ao chat de dada equipa as mudanças são publicadas.

1. Descrição da solução proposta

A aplicação (*Messaging App*) tem uma instância do repositório, da base de dados SQL Lite que armazena a lista de equipas e a lista de membros de uma equipa, bem como da coleção *Firestore* que contém as mensagens da aplicação.

Na criação do Repositório é passada no construtor a aplicação, pois tornou-se necessário obter o contexto na criação de um *WorkRequest* e no acesso às shared preferences. O repositório sincroniza-se com a fonte de dados remota por meio da criação de um *WorkRequest*, que é posteriormente colocado na *queue* do WorkManager. *WorkManager* é uma biblioteca usada para colocar numa *queue* trabalho deferível. Essa API permite criar tarefas e especificar quando devem correr, bem como agendar tarefas para correrem de forma assíncrona. É tomado em conta o nível de API do dispositivo e o estado da aplicação e é possível executar trabalho numa nova *thread* no processo da aplicação. Quando são pedidos elementos de uma equipa ou as listas de equipas, é criado e colocado em *queue* um *WorkRequest* de *TeamWorker* ou de *MemberWorker*.

O *Android* *Jeckpack* introduziu duas peças de software que têm noção de *lifecycle* e que podem ser usadas para partilhar informação através de diferentes módulos. *LiveData*, assim como *MutableLiveData*, pode ser observada, sendo o *LifecycleOwner (componente que tem noção de estado e que podem ser actualizados sem necessidade de escrita de código na activity)*  associado notificado das mudanças ocorridas se estiver no estado STARTED ou RESUMED, isto é, se estiver ativo. Em *LiveData*, *postvalue* e *setvalue* não são métodos públicos, ao contrário do que acontece em *MutableLiveData*. *LiveData* não pode ser modificada fora da classe *ViewModel*, mas *ViewModel* pode modificá-la como seja necessário. Se se quiser modificar dados fora de *ViewModel*, usa-se *MutableLiveData*, que é modificável e *thread-safe*. Foi necessária utilização de MutableLiveData no armazenamento de cache de mensagens.

Relativamente à funcionalidade de chat foi usado o serviço *Firebase - Cloud Firestore*. Este garante o armazenamento das mensagens do chat, podendo depois ser filtradas por equipa garantindo um bom desempenho de funcionamento para os utilizadores. A cada equipa foi atribuído um documento na coleção *messages*, com o nome “team<team\_id>”. Cada um desses documentos tem uma coleção “*chats*”, que contem as mensagens indivuais, que são documentos. Um utilizador apenas tem que se registar para receber mudanças ao documento da sua equipa e após se ligarem ao serviço de *messaging* serão exibidos resultados atualizados em tempo real das mensagens enviadas por outros utilizadores. Esta capacidade é garantida pela propriedade Server Side que garante que notifica os dispositivos que estão à escuta, ficando essa funcionalidade garantida pela chamada do modo *setListener* em *ChatViewModel*, o qual se apresenta a seguir.

Figura 3 – detalhe do método setListener(idTeam: Int) da classe ChatViewModel

fun setListener(idTeam: Int)  
{  
 this.idTeam=idTeam  
 teamChatCollection.addSnapshotListener**{** doc, ex **->** if(ex!=null) {  
 Log.w("Yama chat", "Listen failed:", ex)  
 }  
 else if(doc != null){  
 loadMessages(idTeam)  
 }  
 **}** //registerForMessagesChange()  
}

O método loadMessages que obtém a coleção das mensagens e adiciona as mensagens novas à lista local é apresentado a seguir. Essa lista local funciona como cache das mensagens enviadas e recebidas.

Figura 4 – detalhe do método loadMessages(idTeamLoad : Int) da classe ChatViewModel

val db = FirebaseFirestore.getInstance()  
val masterMessagesCollection = db.collection("messages")  
val TeamDocument by *lazy* **{** masterMessagesCollection.document("team"+idTeam) **}**val teamChatCollection by *lazy* **{** TeamDocument.collection("chats") **}**

fun loadMessages(idTeamLoad:Int) {  
 teamChatCollection.get()  
 .addOnCompleteListener **{** task **->** if (task.*isSuccessful*) {  
 for (document in task.*result*!!) {  
 val msg = MessageSummary(  
 document.get("message").*toString*(),  
 document.get("senderId").*toString*(),  
 document.get("teamId").*toString*().*toInt*(),  
 document.get("time").*toString*()  
 )  
 if(!app.*messageList*.contains(msg))  
 app.*messageList*.add(msg)  
 Log.d(*TAG*, document.*id* + " ====> " + document.*data*)  
 }  
 messages.*value* = app.*messageList*.*filter* **{ it**.teamId==idTeamLoad **}** } else {  
 Log.d(*TAG*, "Error getting documents: ", task.*exception*)  
 }  
 **}**}

1. Avaliação experimental

É apresentado na *Figura 5* o ecrã em que os utilizadores podem colocar os seus dados de acesso. Este ecrã aparece na primeira vez que a aplicação é executada e deixa de ser visível nas restantes utilizações da aplicação. Os seus dados serão guardados em shared preferences que serão consultados nos seguintes acessos.

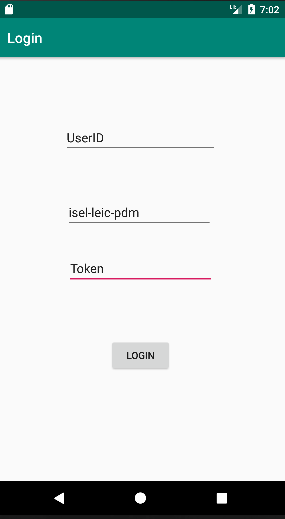


Figura 5 – ecrã de login (primeira utilização)

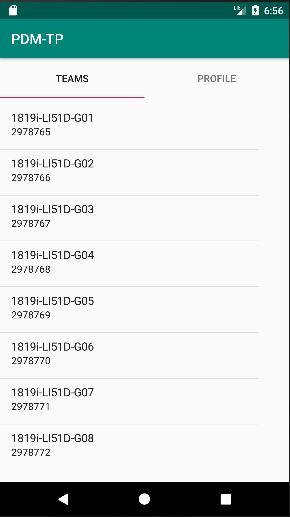
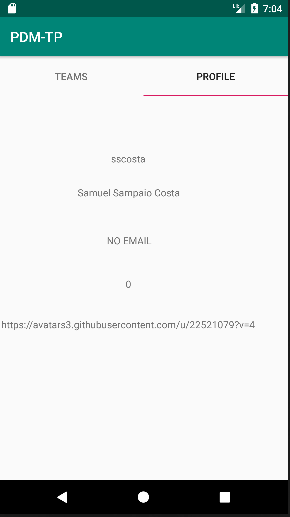


Figura 6 – Detalhe de perfil do utilizador activo

Figura 7 – Lista de equipas

A *Figura 6* mostra o perfil do utilizador ativo em detalhe, incluindo o seu nome de utilizador, nome, email, número de seguidores e url de avatar. Na *Figura 7*, apresenta-se a lista de equipas da organização do utilizador ativo. Após clicar na Equipa o utilizador será redirecionado para a lista de Membros dessa Equipa.

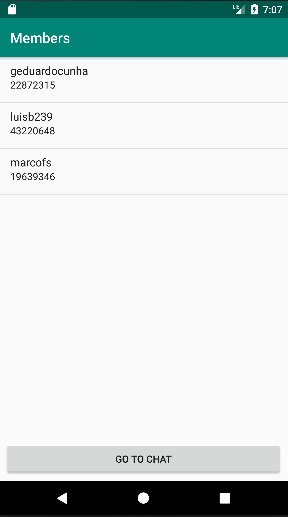


Figura 8 – Lista de membros de uma equipa

A *Figura 9* mostra os elementos de uma equipa selecionada, o seu nome de utilizador e id único. Quer através do Membro ou do botão “GO TO CHAT” o utilizador será enviado para a página de envio de mensagens.



Figura 9 – sala de chat de uma equipa

Na *Figura 7* apresenta-se a sala de chat de uma equipa, incluindo as mensagens enviadas pelos elementos dessa equipa.

1. Conclusão

Considerou-se que este trabalho prático permitiu compreender e integrar conceitos relativamente novos que foram introduzidos nas aulas, como por exemplo o tópico do *WorkManager*, a sua concretização para as tarefas de carregamento de equipas e membros, e execução de tarefas em segundo plano, designadamente tarefas de inserção e remoção em base de dados em threads secundarias. Permitiu também abordar a peça *LiveData* e integrar essa informação observável numa utilização de forma transparente numa *Activity*. Para além disso, exigiu uma organização do código de forma compacta. Permitiu, finalmente, a adaptação do mesmo esquema, baseado em duas entidades distintas: a obtenção da lista de equipas da organização e de membros de uma equipa.

No global, apesar de introdutório à temática do desenvolvimento de aplicações móveis, considera-se um exercício pleno de desafio e concluído com sucesso. Destacamos em particular, a utilização da fonte de dados remota *FireStore*, que armazena dados em coleções, que requereu de entre todos os tópicos uma investigação mais aprofundada e tornou significativos e perenes os exercícios feitos nas aulas.

Destacam-se dois desenvolvimentos futuros para a aplicação, nomeadamente que no carregamento de equipas fossem apenas inseridas as equipas que constituem diferença face ao que está armazenado na base de dados SQLite em vez de se proceder a remoção de todos os elementos e posterior adição. Também seria desejável que fossem enviadas notificações para os utilizadores interessados sempre que se verificasse existência de novas mensagens numa sala de chat.

1. Fonte:https://developers-br.googleblog.com/2018/05/use-o-android-jetpack-para-acelerar-o.html [↑](#footnote-ref-1)
2. https://developer.android.com/jetpack/docs/guide [↑](#footnote-ref-2)