

**Licenciatura em Engenharia Informática e de Computadores**

**Semestre de Inverno 2014/2015**

Programação em

Dispositivos Móveis

2ª & 3ª Série de exercícios

*Trabalho elaborado por:*

* Flávio Cadete

Nº 35383

*Engenheiro:*

* Pedro Félix

Índice

[**AnniversaryReminder** 3](#_Toc407562707)

[Enunciado 3](#_Toc407562708)

[AlarmStartupReceiver 3](#_Toc407562709)

[AlarmNotificationReceiver 4](#_Toc407562710)

[Notificações 5](#_Toc407562711)

[Preferências 6](#_Toc407562712)

[**ThothNews** 7](#_Toc407562713)

[Enunciados 7](#_Toc407562714)

[ContentProvider 8](#_Toc407562715)

[IntentService 9](#_Toc407562716)

[BroadcastReceiver 9](#_Toc407562717)

[CursorLoaders 10](#_Toc407562718)

[CursorAdapters 10](#_Toc407562719)

[Notificações 11](#_Toc407562720)

[Selecção de turmas 12](#_Toc407562721)

[Procura na selecção de turmas 12](#_Toc407562722)

[Turmas escolhidas 14](#_Toc407562723)

[Lista de Notícias - Introdução 15](#_Toc407562724)

[Lista de Notícias – Phone 16](#_Toc407562725)

[Lista de Notícias – Tablet 17](#_Toc407562726)

[Estudantes inscritos 18](#_Toc407562727)

[Avatar – Handlers + AsyncTask 19](#_Toc407562728)

[Funcionalidades Adicionais 20](#_Toc407562729)

[Multi-Língua 20](#_Toc407562730)

[Enviar correio electrónico 20](#_Toc407562731)

[AutoLink 20](#_Toc407562732)

[Ler todas as notícias 21](#_Toc407562733)

[SwipeRefreshLayout 21](#_Toc407562734)

[Esconder Fragmento em Landscape 21](#_Toc407562735)

# **AnniversaryReminder**

### Enunciado

##### 2ª Série de exercícios

2. Adicione à aplicação de gestão de aniversários, realizada na primeira série, a capacidade de criar notificações da proximidade de um evento. Por exemplo, a aplicação deverá enviar uma notificação:

* Quando o aniversário de um contacto ocorrer a menos de uma semana.
* No dia de aniversário de um contacto.

Em ambos os casos deverá ser possível, a partir da notificação, visualizar a informação do contacto.

### AlarmStartupReceiver

Esta classe extende de ***BroadcastReceiver*** e tem como objectivo detectar quando foi feito o ***BOOT*** do dispositivo ou quando o número de semanas a filtrar encontrado preferências foi alterado.

Depois de feita a confirmação irá requisitado um serviço de ***AlarmManager*** que já é intrínseco á plataforma Android.

É adicionado um *PendingIntent* ao **AlarmManager** criado e que contém um *Intent* para fazer broadcast à classe **AlarmNotificiationReceiver** (que é o receiver responsável do envio das notificações)*.*

Neste momento só falta recolher das ***DefaultSharedPreferences*** a hora defenida nas preferências ([***Figura 6***](#_Preferências)), e passar ao setRepeat do ***AlarmManager*** criado.

*AlarmManager.setRepeat*

*PendingIntent*

*AlarmNotificationReceiver*

**BroadcastReceiver**

*BOOT\_COMPLETED*

*PREFS\_UPDATE*

ALARM\_SERVICE

**Service***(Started Service)*

**Figura 1**

Alterações necessárias no Manifesto para que o componente BroadcastReceiver em cima descrito tenha funcionalidade pretendida:

<receiver android:name=".broadcastreceivers.AlarmStartupReceiver"

android:enabled="true"

android:permission="android.permission.RECEIVE\_BOOT\_COMPLETED">

<intent-filter>

<action android:name="android.intent.action.BOOT\_COMPLETED" />

<action android:name="com.starlon.froyvisuals.PREFS\_UPDATE"/>

<category android:name="android.intent.category.DEFAULT" />

</intent-filter>

</receiver>

<uses-permission android:name="android.permission.RECEIVE\_BOOT\_COMPLETED" />

### AlarmNotificationReceiver

Como o próprio nome deixa induzir, esta classe extende do componente ***BroadcastReceiver***.

Tem como objectivo receber os broadcasts do ***AlarmManager*** criado em ***AlarmStartupReceiver*** e enviar as notificações com a informação necessária para que o utilizador consiga distinguir os aniversariantes, além de poder abrir a aplicação de contactos com o contacto aniversariante selecionado na notificação.

Foram adicionados às notificações pequenas funcionalidade como vibração, luzes led e som ao Notification.Builder, sendo o mesmo passado ao método *notify* ***NotificationManager***, além do ID da notificação a enviar (é enviada um ID diferente por cada contacto aniversariante).

O método ***notify*** enviará assim a notificação activando as funcionalidades passadas.

ALARM\_SERVICE

**Service***(Started Service)*

*NotificationManager.notify*

**BroadcastReceiver**

**Figura 2**

Alterações necessárias no Manifesto para que o componente BroadcastReceiver em cima descrito tenha funcionalidade pretendida:

<receiver android:name=".broadcastreceivers.AlarmNotificationReceiver"

android:enabled="true">

<intent-filter>

<action android:name="android.intent.action.SCREEN\_ON"/>

<action android:name="android.intent.action.VIEW" />

</intent-filter>

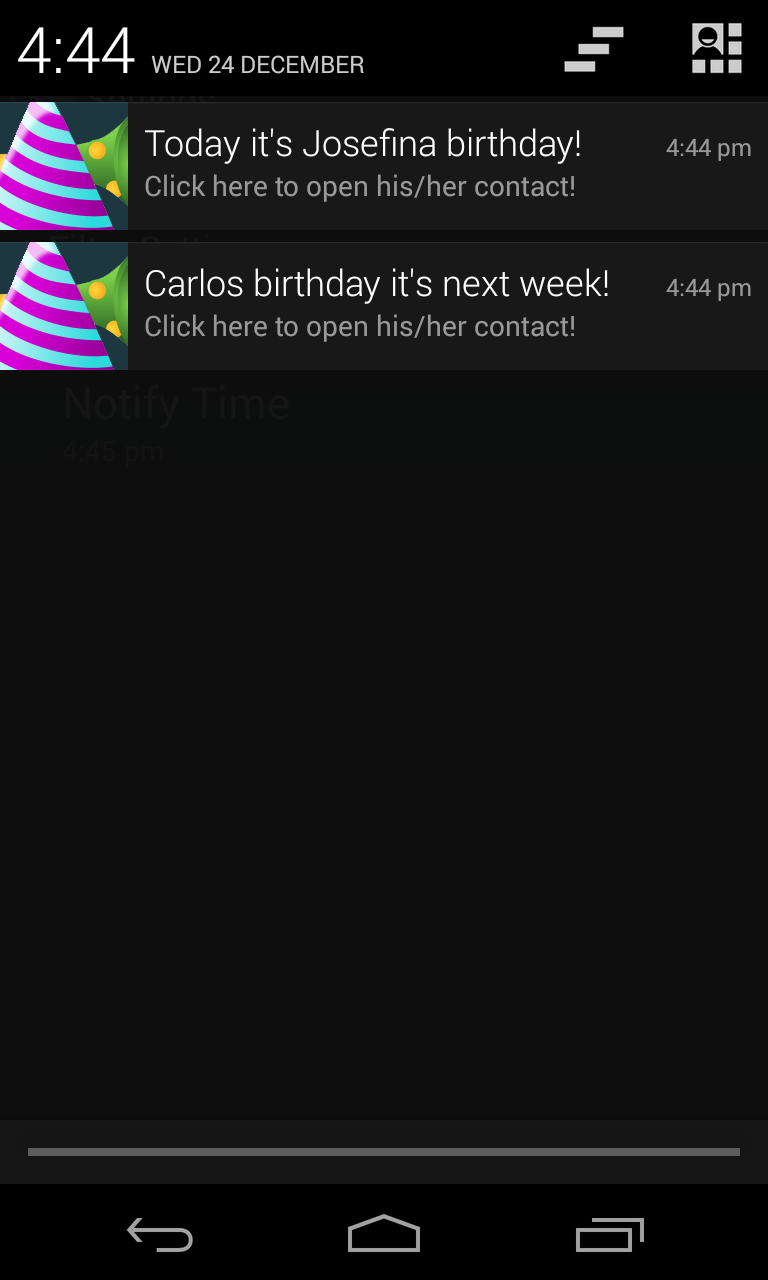
</receiver>

<uses-permission android:name="android.permission.VIBRATE" />

### Notificações

Na ***Figura 3*** é possível verificar as ***notificações*** recebidas. Cada contacto tem a sua respectiva notificação. Ao clicar na notificação irá abrir a aplicação responsável pela gestão dos contactos, com a vista do contacto selecionado na notificação e exemplificado na ***Figura 4***, podendo aí ligar/enviar SMS ao contacto aniversariante.

Também é possível com um ***long* *click*** num contacto da lista de contactos filtrados pela nossa aplicação, abrir a vista responsável pela informação do contacto e presente na ***Figura 4***.



**Figura 3**: Notificações (própria dia e dentro de uma semana)



**Figura 4**: Informação do Contacto (Genymotion Contacts Application)

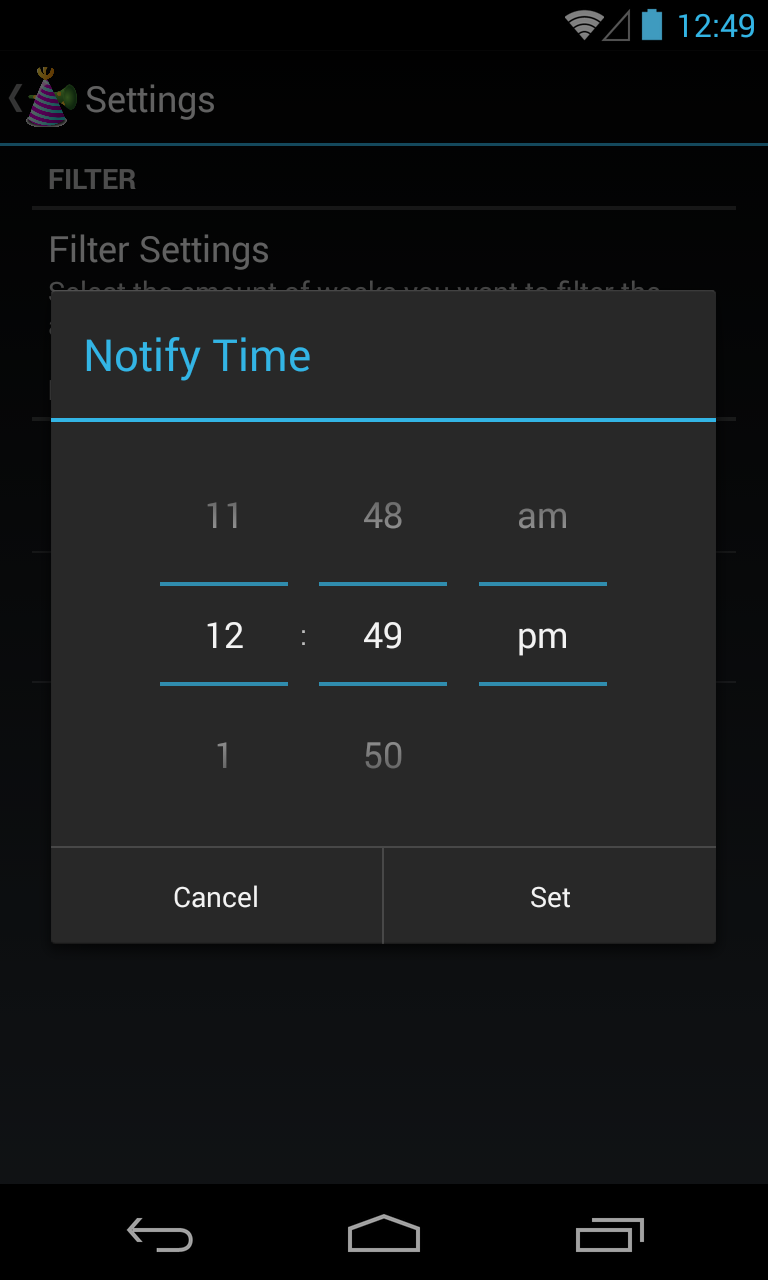
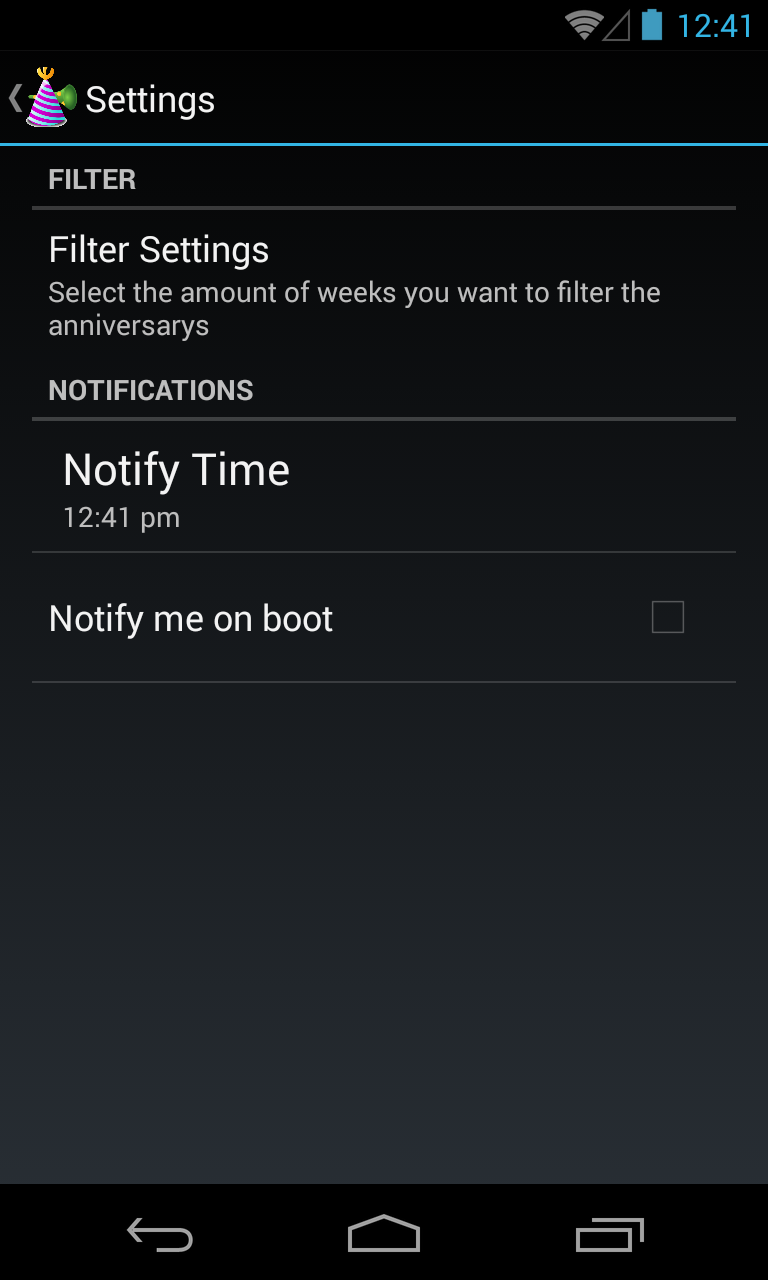
### Preferências

Alteração das preferências face á série anterior . Foi adicionado um ***DialogPreference*** (***Figura 6***) que dá a possibilidade ao utilizador de indicar a que hora pretende receber as notificações. Esta alteração será depois visivel ao utilizador nas preferências (***Figura 5***).

Quando é seleccionado o botão ***Set*** exemplificado na ***Figura 6***, é guardado em milisegundos a hora escolhida nas ***DefaultSharedPreferences*** da aplicação e enviado um broadcast com um Intent com a acção de ***PREFS\_UPDATE***, fazendo com que a classe ***AlarmStartupReceiver*** possa criar um novo ***AlarmManager*** com a nova a hora que irá fazer os brodcasts.

*(Na 2ªSérie ainda não tinha sido leccionado os Fragmentos, por isso tenho consiência de que irei perder o dialog quando são feitas rotações com o dispositivo e que face ao pedido no momento esta requisito não era obrigatório)*

**Figura 5**: Preferências



**Figura 6**: DialogPreference para escolher quando pretende receber as notificações

# ThothNews

### Enunciados

#### 2ª Série de exercícios

1. Adicione à aplicação de notícias do Thoth as seguintes características:

* Existência dum content provider para armazenamento e gestão da seguinte informação:

1. Conjunto de turmas disponíveis no Thoth.
2. Conjunto de turmas selecionadas.
3. Conjunto de notícias das turmas selecionadas, incluindo o estado de visualização.

Este content provider deve usar uma base de dado SQLite para persistência da informação.

* Existência dum serviço para actualização periódica da informação, apenas realizada quando existir conectividade via WiFi.

#### 3ª Série de exercícios

1. Altere a aplicação de notícias do Thoth para usar fragmentos na implementação da user interface. Nomeadamente, acrescente suporte para:

* Interface master-details com dois painels.
* Navegação entre notícias através de swipe.

2. Adicione à aplicação do Thoth a capacidade de apresentar os participantes de uma turma. Esta apresentação deve incluir a fotografia do participante. Realize o necessário para minimizar o consumo de recursos necessários para esta tarefa.

### ContentProvider

##### ThothProvider

Classe que extende ***ContentProvider*** e tem a responsabilidade fazer match das rotas recebidas com as rotas a que consegue responder, assegurando segurança nas alterações feitas á base de dados.

Essa resposta consiste em fazer o comando requisitado á base de dados e:

* Notificar essas alterações (***insert***, ***update*** ou ***delete***) a todos os cursores registados e interessados nessa alteração;
* Registar o cursor no caso de retorno de um cursor (***query***). (Mais informações em [***CursorLoaders***](#_CursorLoaders))

##### ThothContract

Esta classe é responsável por definir o contracto com DB, ou seja:

* Indicar a Autoridade responsável por responder aos pedidos á DB;
* Definir campos para cada tabela;
* Definir a query de ***Create*** com os nomes das colunas e respectivos tipos, além da chave primária;

Cada classe estática e interna a ThothContract representa uma tabela, sendo que essa classe implementa BaseColumns que nos dá a definição de um id único que irá representar cada tuplo (*row*) na tabela. Este id único tem o nome de coluna **‘\_id’** e será a chave primária da tabela.

##### ThothDBHelper

Classe que extende de ***SQLiteOpenHelper*** e responsável por criar a base de dados, tabelas e respectivas colunas, através das ***Create Query*** disponibilizadas por cada tabela em ***ThothContract***.

Embora não o esteja a suportar, também seria expectável nesta classe disponibilizar alterações de estrutura da DB.

#### Tabelas Criadas:

* ***classes*** (Tabelas das Turmas)
* ***news*** (Tabela das Notícias)
* ***students*** (Tabela dos Estudantes)
* ***teachers*** (Tabela dos Docentes)
* ***classes***\_***students*** (Tabela de **Associação** **entre Turmas e Estudantes**)

Manifesto:

<provider

android:name="pt.isel.pdm.grupo17.thothnews.data.ThothProvider"

android:authorities="pt.isel.pdm.grupo17.thothnews"

android:enabled="true"

android:exported="false" >

</provider>

### IntentService

##### ThothUpdateService

O nosso serviço é responsável pelos pedidos HTTP à plataforma Thoth, recebendo os dados em forma de objectos ***JSON*** e introduzir esses dados na minha SqLite DB via ThothProvider.

Apenas se encontram públicos 4 métodos, sendo que cada método lança um serviço com um *intent* com uma acção diferente, além de um extra com a classId para pedido espcíficos. Esse *intent* é recebido pelo serviço via método ***onHandleIntent(Intent intent)***, e extraído a acção do *intent*.

Seguem-se as 4 acções possíveis:

* ***ACTION\_CLASSES\_UPDATE:*** Actualização de todas turmas e respectivos docentes;
* ***ACTION\_NEWS\_UPDATE***: Actualização das notícias todas turmas seleccionadas/matriculadas;
* ***ACTION\_CLASS\_NEWS\_UPDATE***: Actualização das notícias da turma com um id específico;
* ***ACTION\_CLASS\_PARTICIPANTS\_UPDATE***: Actualização dos participantes da turma com id específico.

Manifesto:

<service

android:name="pt.isel.pdm.grupo17.thothnews.services.ThothUpdateService"

android:exported="false" >

</service>

### BroadcastReceiver

##### NetworkReceiver

Este BroadcastReceiver tem a missão de detectar os momentos em que existe conexão WiFi (Wireless ou Dados móveis) ou deixa de a ter.

Do ponto de vista de utilização com dispositivos Android, decidi não utilizar um AlarmManager para actualização periódica da informação devido a dois motivos:

* Não é um dispositivo que se encontre muitas horas ligado á internet, desligando a mesma quando não se encontra em uso. Ao voltar acordar, é feita a tentativa automática de conexão e nesse momento o broadcast recebe essa informação e faz o pedido de actualização;
* O componente necessário para actualização periódica já tinha sido desenvolvida por mim na aplicação AnniversaryReminder, preferindo assim dedicar o meu tempo ao desenvolvimento de outras componentes.

Manifesto:

<uses-permission android:name="android.permission.INTERNET" />

<uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />

<uses-permission android:name="android.permission.RECEIVE\_BOOT\_COMPLETED" />

<receiver android:name=".broadcastreceivers.NetworkReceiver" >

<intent-filter>

<action android:name="android.net.conn.CONNECTIVITY\_CHANGE" />

<action android:name="android.net.wifi.STATE\_CHANGE" />

</intent-filter>

</receiver>

### CursorLoaders

Um CursorLoader em geral, não faz detecção de alterações nos dados da DB e afetação **automática** nos dados presentes na vista que se encontra visível ao utilizador. Para que isso aconteça é necessário que utilizemos o ***padrão*** ***Observer***, seguindo os passos:

1. Registar um ***Observer*** no ***ContentResolver*** através do cursor:

* *cursor.setNotificationUri(getContext().getContentResolver(), uri);*

1. Quando existem alterações feitas a dados, ou seja, sejam executados os comandos insert()/delete()/update(), temos de notificar o ContentResolver com:

* getContext().getContentResolver().notifyChange(insertedId, null);

1. A notificação é recebida pelo observer registado:

* Internamente chama o ***ContentProvider*** com a query() e o uri passado no passo 1, retornando um novo cursor ao ***LoaderManager***;
* O ***LoaderManager*** chama o ***onLoadFinished*** passando esse novo cursor, junto com o CursorLoader da vista que terá de fazer update;
* Para que seja possível fazer update do seu conteúdo é utilizado o método ***swapcursor*** do CursorAdapter dessa vista (fragmento), que limpa a sua lista e introduz os novos dados.

#### Métodos de LoaderManager.LoaderCallbacks:

* ***onCreateLoader***: Instancia e retorna um novo Loader que está a ser carregado;
* ***onLoadFinished***: Chamado após um Loader ter sido criado e carregado;
* ***onLoaderReset***: Chamado quando é necessário invalidar a data presente no Loader.

Os seguintes Fragmentos contém uma instância de um ***CursorAdapter*** e implementam LoaderManager.LoaderCallbacks <Cursor>:

* ***ClassesFragment*** |***ClassesPickFragment*** | ***NewsListFragment*** | ***ParticipantsFragment***
* **Inicializar** explicitamente o cursor: ***getLoaderManager().initLoader(LOADER\_ID, null, this);***
* **Reiniciar** explicitamente o cursor: ***getLoaderManager().restartLoader(LOADER\_ID, null, this);***

### CursorAdapters

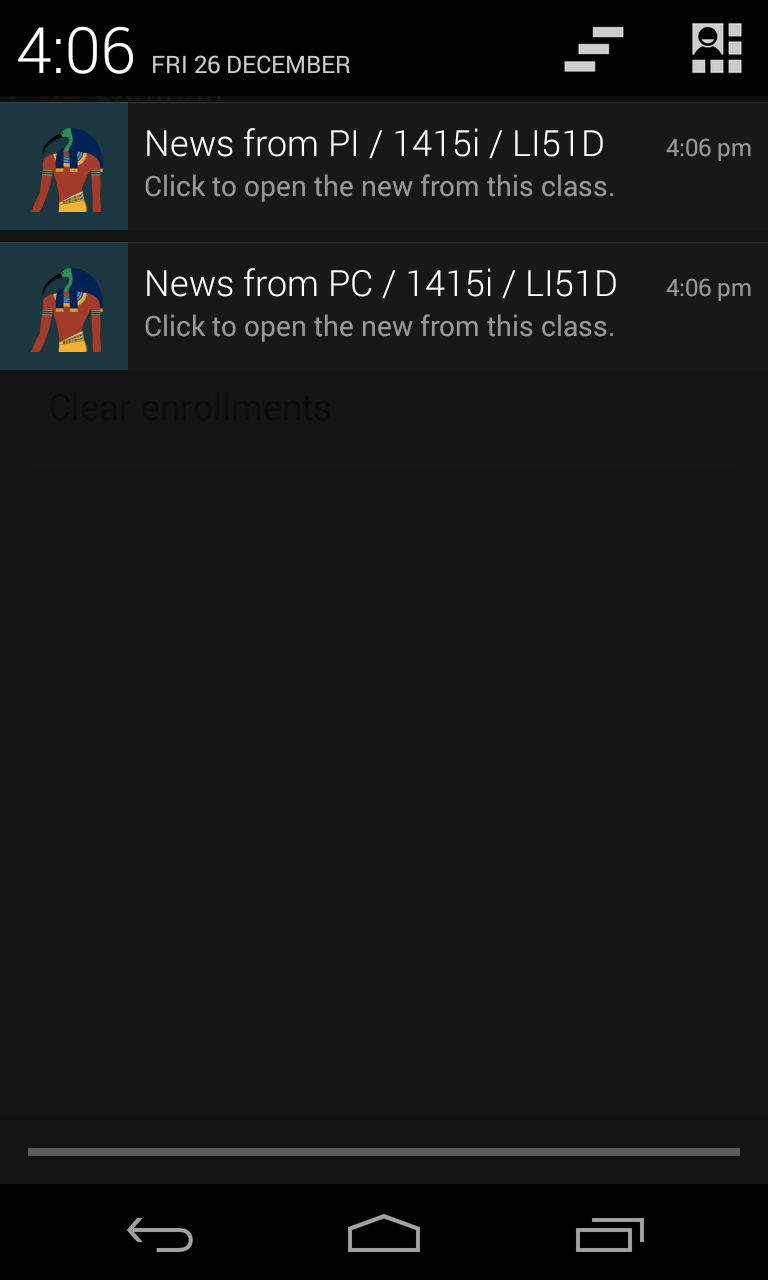
O cursor passado ao adapter tem de conter uma coluna com o nome **‘\_id’**,sendo este definido em BaseColumns e implementado por todas as tabelas.

***CursorAdapters*** existentes e em que cada um define o método ***swapcursor***, além de ***getItem***, ***newView*** e ***bindView***:

* ***ClassesAdapter*** -> ClassesFragment
* ***ClassesPickAdapter*** -> ClassesPickFragment
* ***NewsAdapter*** -> NewsListFragment
* ***ParticipantsAdapter*** -> ParticipantsFragment

### Notificações

Na ***Figura 7*** é possível ver 2 notificações, uma para cada turma com novas notícias.



**Figura 6**: Notificação para cada turma com novas notícias

As notícias são lançadas quando é feita a chamada ao ***handleClassNewsUpdate (classId)*** em ***ThothUpdateService,*** e só depois de ter a confirmação de que foram realmente adicionadas notícias á base de dados, é que irá lançar a notificação com a informação necessária para que o utilizador consiga identificar a turma com a (s) nova (s) notícia (s) e adicionado um ***PendingIntent*** para ***ClassSectionsActivity.class*** com um extra da turma serializado, lançando a vista com as notícias pretendidas. *(método sendNotification (long classID))*

A chamada a ***handleClassNewsUpdate*** tanto pode vir da acção de Intent ***ACTION\_NEWS\_UPDATE*** ou pela acção ***ACTION\_CLASS\_NEWS\_UPDATE*** feitas ao nosso serviço ***ThothUpdateService***.

Case o tipo de acção seja ***ACTION\_NEWS\_UPDATE***, chamará ***handleNewsUpdate*** que irá percorrer um cursor com os ids de todas as turmas ***matriculadas,*** passando assim o id da classe a *handleClassNewsUpdate (classId).*

Estas acções podem ocorrer em três tipos de situação:

* Selecção (matricular) de turmas
* Momento de ligação WiFi
* Pedido de refresh na lista de notícias

(pedido explícito pelo utilizador)

###### Código em **ThothUpdateService**:

private void **sendNotification** (long classID) {

Cursor classInfo = getContentResolver().query(UriUtils.Classes.parseClass(classID), null,null,null,null);

if(classInfo.moveToNext()){

**ThothClass thothClass = ThothClass.fromCursor(classInfo);**

Intent intent = new Intent(this.getApplication(), **ClassSectionsActivity.class**);

**intent.putExtra(TagUtils.TAG\_SERIALIZABLE\_CLASS, thothClass**);

intent.setFlags(**Intent.FLAG\_ACTIVITY\_NEW\_TASK** | Intent.FLAG\_ACTIVITY\_CLEAR\_TOP);

PendingIntent pIntent = PendingIntent.getActivity(this.getApplication(), **NOTIFICATION\_ID**, intent, PendingIntent.**FLAG\_CANCEL\_CURRENT**);

if(builder == null)

**builder =** **new Notification.Builder**(getApplicationContext()) (…);

builder.setContentTitle("News from "+ thothClass.getFullName().setContentIntent(pIntent);

if(notificationManager == null)

notificationManager = (NotificationManager) this.getApplicationContext().**getSystemService(Context.NOTIFICATION\_SERVICE)**;

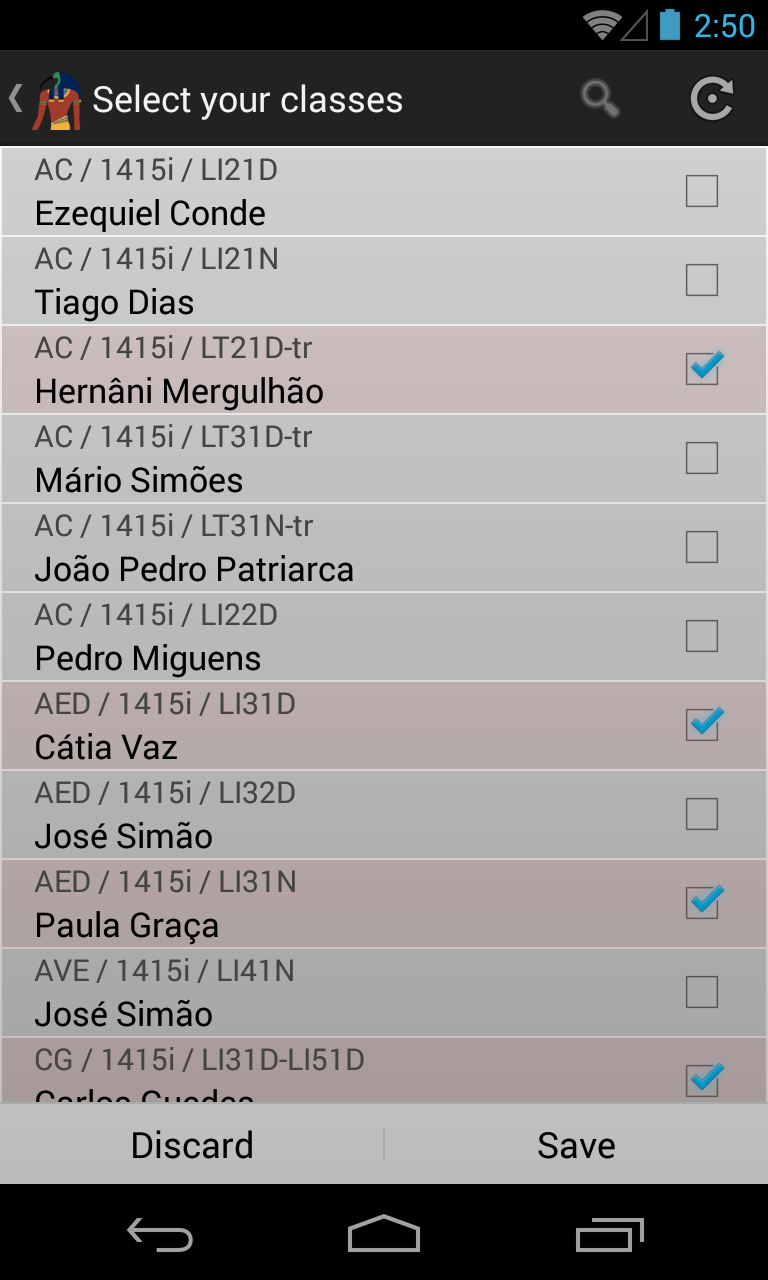
**notificationManager**.**notify**(**NOTIFICATION\_ID++,** **builder.build()**);

}

}

### Selecção de turmas

##### ClassesPickActivity & ClassesPickFragment



**Figura 7**: Lista da selecção de classes – Phone

Esta vista já tinha sido construída para a 1ª Série, mas foram necessárias algumas mudanças.

Além das óbvias alterações visuais, foi corrigido o botão de ***Discard/Descartar***. A tática utilizada foi guardar no adapter deste fragmento apenas as alterações (estado *incial & final*) feitas pelo utilizador e retomando o seu estado inicial, caso o utilizador queira descartar o que alterou (com novo update á DB, com o estado inicial, por cada turma).

Como implementa LoaderCallbacks, cada vez que é selecionado uma turma, além de alterar a *checkbox* visível ao utilizador, é necessário fazer um **update** à *DB* para que essa alteração não se perca ao fazer **scrolling** da lista ou **rotação** do dispositivo.

Quando é iniciado esta vista é feita a verificação se já se encontram turmas na *DB*. Caso seja verdade é feita logo inicialização da lista com ***initLoader*** (pág.10).

Caso contrário é feita verificação de existência de conectividade activa, extracção de turmas e respetivos docentes via *HTTP à framework Thoth* e adicionado á base de dados. Este processo, visto que a base de dados encontra-se totalmente vazia, demora alguns segundos, pedindo ao utilizador para que seja paciente *(via dialog - Toast)*.

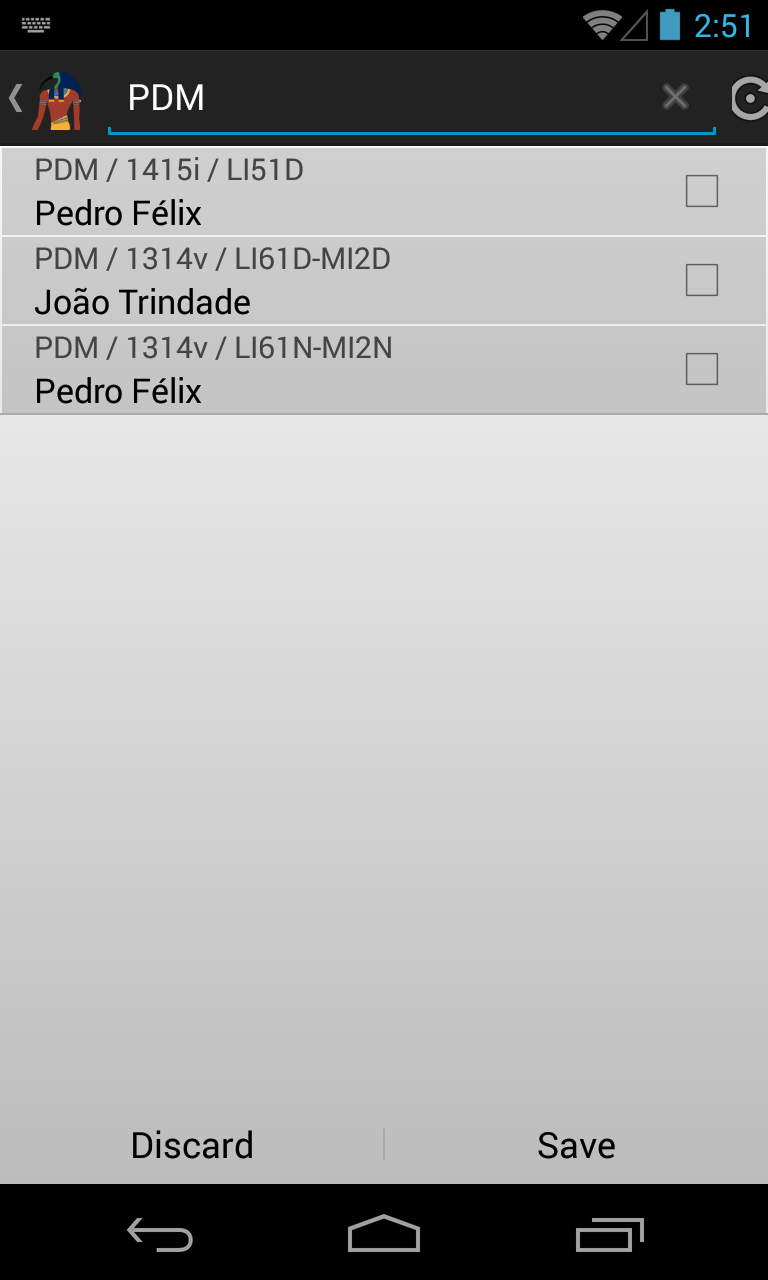
### Procura na selecção de turmas



**Figura 8:** Grelha da selecção de classes - Tablet

##### ClassesPickActivity & ClassesPickFragment

Na vista de selecção de turma, foi adicionado um SearchView que penso que é algo essencial, numa lista com muitos elementos.



**Figura 9**: SearchView - Procura de classes – Phone

Para conseguir obter este resultado foi preciso fazer os seguintes passos:

1. Implementar ***SearchView.OnQueryTextListener*** e os seus métodos ***onQueryTextSubmit*** e ***onQueryTextChange*** no fragmento ***ClassesPickFragment.class***;
2. Adicionar um ***MenuItem*** ao *Menu* do fragmento a partir do método ***onCreateOptionsMenu*** ***(Menu menu)***;
3. Adicionar uma route especial no ThothProvider e retornar um novo cursor com a respectiva selecção:

\_matcher.addURI(CONTENT\_AUTHORITY, "classesSearch/\*", ROUTE\_CLASSES\_SEARCH);

1. Reiniciar o cursor Loader passando o texto adicionado ao SearchView

Código adicionado a **ClassesPickFragment**:

public void myCreateOptionsMenu(Menu menu) { // chamado em ***onCreateOptionsMenu*** *da* actividade

**MenuItem item = menu.add("Search");**

item.setIcon(android.R.drawable.ic\_menu\_search);

item.setShowAsAction(MenuItem.SHOW\_AS\_ACTION\_ALWAYS);

**SearchView sv = new SearchView(getActivity());**

sv.setOnQueryTextListener(this);

int id = **sv.getContext().getResources().getIdentifier("android:id/search\_src\_text",**

null, null);

((TextView) sv.findViewById(id)).setTextColor(Color.WHITE);

**item.setActionView(sv);**

}

@Override public boolean **onQueryTextSubmit**(String query) {

return true;

}

@Override public boolean **onQueryTextChange**(String newText) {

mCurFilter = !TextUtils.isEmpty(newText) ? newText : null;

**getLoaderManager().restartLoader(CLASSES\_SELECTION\_CURSOR\_LOADER\_ID, null, this);**

return true;

}

@Override public Loader<Cursor> ***onCreateLoader***(int id, Bundle args) {

**Uri baseUri;**

**if (mCurFilter != null) {**

**baseUri = Uri.withAppendedPath(ThothContract.Classes.SEARCH\_URI,**

**Uri.encode(mCurFilter));**

**} else {**

**baseUri = ThothContract.Classes.CONTENT\_URI;**

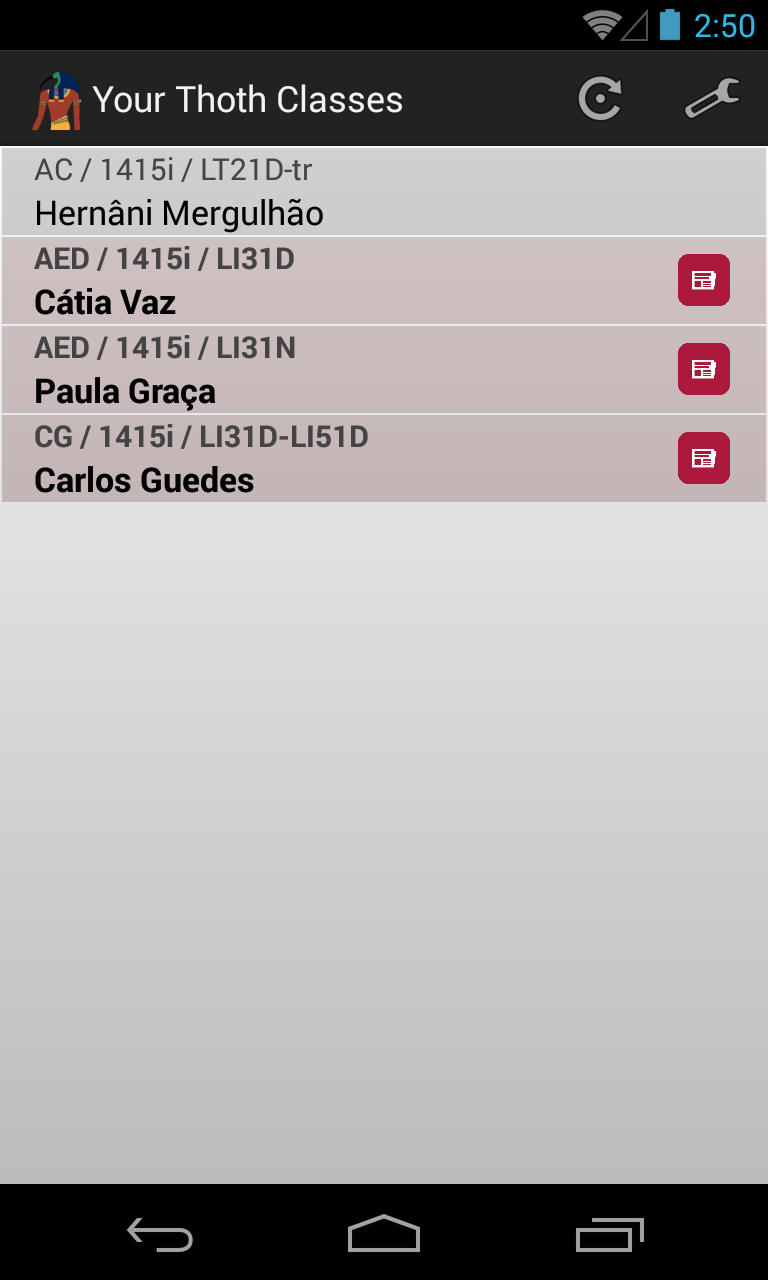
**}**

return new CursorLoader(getActivity(), **baseUri**, CURSOR\_COLUMNS, null, null, ORDER\_BY);

}

### Turmas escolhidas

##### ClassesActivity & ClassesFragment



**Figura 10**: Lista das classes escolhidas - Phone

Vista com as turmas previamente escolhidas nas definições.

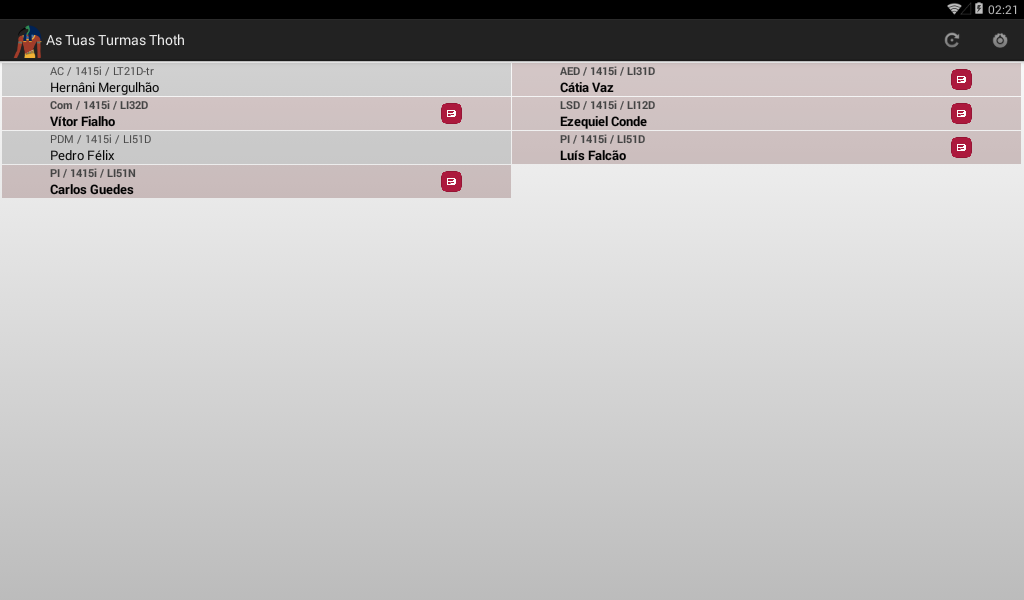
Contém um LoaderCursor que que contém um URI que faz match com ***ROUTE\_CLASSES\_ENROLLED***.

O ThothProvider retorna então um *Cursor* com as turmas em que a coluna ***ENROLLED*** da tabela ***classes*** estejam a “***1”*** (ou seja a ‘*true’*).

Além do ***ClassesAdapter***, responsável pela gestão da lista mostrada ao utilizador, o fragmento também contém uma instância de ***GridView***. Essa instância corresponde a um ***GridView*** definido no *layout* utilizado pelo fragmento.

Sobre a ***GridView*** foi implementado o método observer ***setOnItemClickListener ()***, que irá detectar qual o elemento da lista foi selecionado e inicia uma nova actividade, passando no seu *Intent* um extra com um objecto da turma serializado.

D:\ISEL\14_15_SInv\PDM\MyRep\src\ThothNews\app\src\main\res\drawable\ic_new_news.pngAlém diferenciação por cores de cada elemento da lista do adapter, foi introduzido o símbolo que indica que a classe contém notícias por ler.

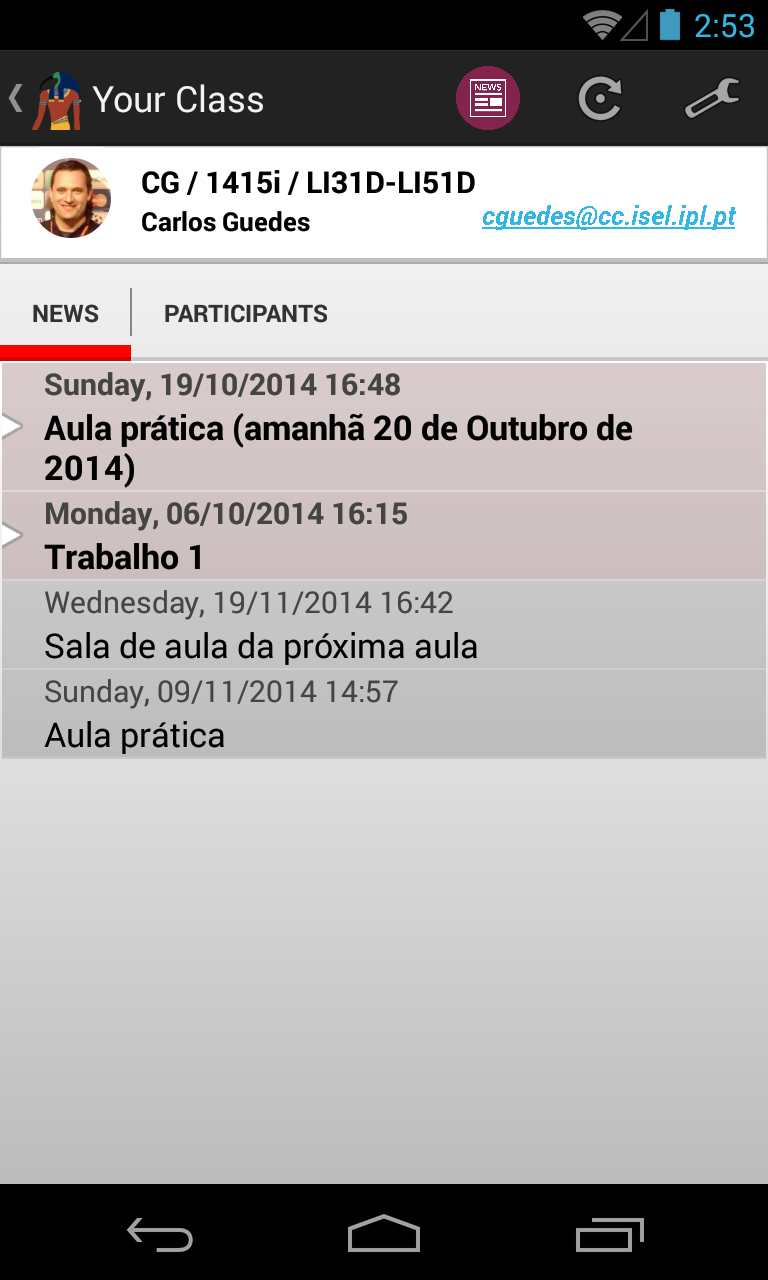


**Figura 11**: Grelha das classes escolhidas - Tablet

### Lista de Notícias - Introdução

##### ClassSectionsActivity

Dada a quantidade de componentes concentrados nesta actividade, para conseguir obter o resultado final esperado foi feita a seguinte figura descritiva:



**Figura 12:** Constituição da vista das secções de uma classe

**FrameLayout** composto por um **HorizontalScrollView** e um **ViewPager** que têm a responsabilidade de detectar os movimentos feitos pelo utilizador (clique no tab, ou transação horizontal na ViewPager).

Para gerir a adição de informação nas tabs e instanciação dos diferentes fragmentos presentes no ViewPager, foi utilizado o Fragmento ***SlidingTabsColorsFragment.class****, que além disso contém um FragmentPagerAdapter responsável por enviar um fragmento, dependente da tab selecionada.*

**FrameLayout** com a informação da turma escolhida e o docente dessa turma.

Tabs associadas á turma selecionada (*SlidingTabLayout.class*)

**ViewPager** com o fragmento *NewsFragment.class* visível

Fragmentos utilizados nesta actividade:

* *SlidingTabsColorsFragment*
* *NewsListFragment*
* *SingleNewFragment* ***(Utilizado de maneira diferente na versão Tablet vs Phone)***
* *ParticipantsFragment.*

Fragmentos instanciados e utilizados pelo Fragmento SlidingTabsColorsFragment:

* *NewsListFragment*
* *ParticipantsFragment*

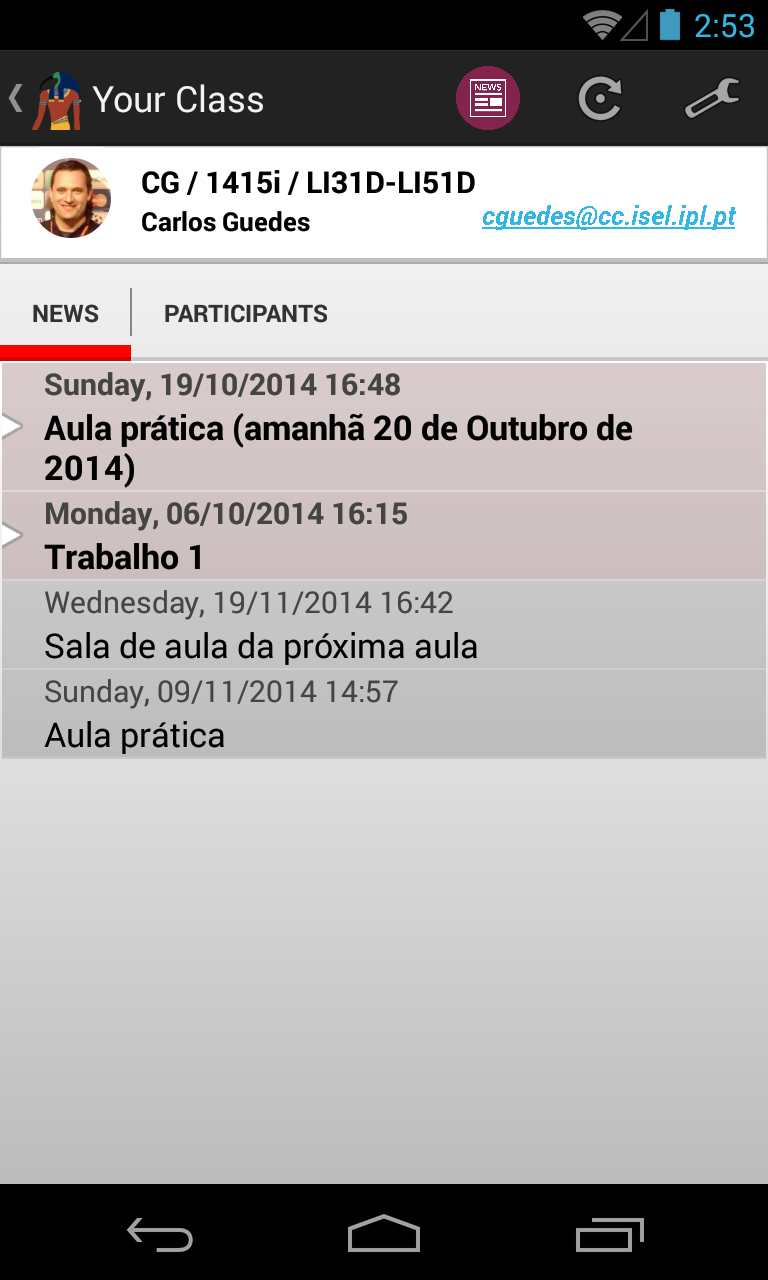
No Fragmento ***NewsListFragment*** foi definido uma propriedade booleana estática (**sTwoPane**), para que esta actividade saiba se o dispositivo sobre o qual a aplicação está a correr no momento, é um Telemóvel ou Tablet.

Ao clicar numa notícia irá:

* *Tablet* ***(sTwoPane == True)****:* Criar instância de ***SingleNewFragment***, passando como argumento um objecto serializado com o conteúdo da notícia seleccionada *(afectando o FrameLayout* *apenas existente no Layout para versões sw600p)*
* *Phone* ***(sTwoPane == False)****:* Lançar nova actividade ***(SingleNewActivity)*** com um objecto serializado, de **todas** as notícias da turma

### Lista de Notícias – Phone

##### NewsListFragment

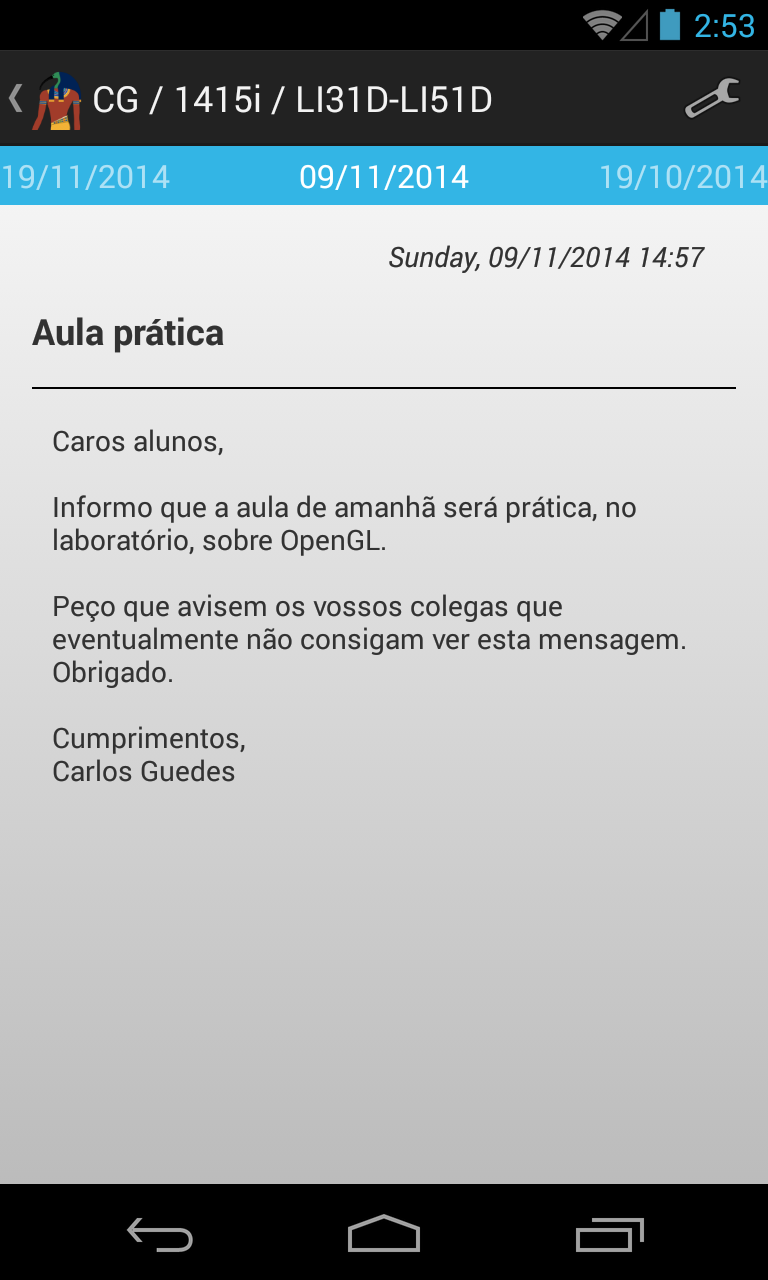


**Figura 13**: Lista de notícias da classe escolhida - Phone

Este Fragmento contém apenas, a lista das notícias criadas na plataforma Thoth pelo docente da turma seleccionada.

*NewsListFragment* implementa *ListFragment* e por isso é possível utilizar o método herdado ***onListItemClick (…)***, que a partir da propriedade estática *sTwoPane == false* lançará uma nova actividade ***(SingleNewActivity)***, mostrado na *Figura 14****.***

##### SingleNewActivity & SingleNewFragment



**Figura 14**: Detalhe da notícia escolhida - Phone

O *Layout* desta actividade além de conter um ***ViewPager***, contém um ***PagerTitleStrip*** (barra azul) que dá ao utilizador uma breve informação sobre o conteúdo das notícias adjacentes. Neste caso optei por utilizar a data em que a notícia foi criada.

A actividade extraí dos extras do *Intent* o objecto serializado com todas as notícias e a posição seleccionada e afecta o adapter da instância de ViewPager como é possível verificar no código mais em baixo.

int thothNewPosition = intent.getExtras().getInt(TagUtils.TAG\_SELECT\_NEW\_POSITION, 0);

ThothNewsList list = (ThothNewsList) intent.getExtras().getSerializable(TagUtils.TAG\_SERIALIZABLE\_LIST);

sThothNewList = list.getItems();

mViewPager.setAdapter(new FragmentStatePagerAdapter(getSupportFragmentManager()) {

@Override public Fragment getItem(int position) {

return SingleNewFragment.newInstance(sThothNewList.get(position));

}

@Override public int getCount() {

return sThothNewList.size();

}

@Override public CharSequence getPageTitle(int position) {

return sThothNewList.get(position).getShortWhen();

}

});

mViewPager.setCurrentItem(thothNewPosition);

### Lista de Notícias – Tablet

##### NewsListFragment & SingleNewFragment

Para que este fragmento suporte duplo painel, sendo que o 2º contém informação dependente do 1º foi necessário criar um sistema de callback. Ou seja ***NewsListFragment*** criou uma interface de nome ***Callbacks*** e com o método ***onItemSelected(ThothNew)***, sendo que a actividade tenha de implementar esta interface e assim poder passar o objecto pretendido à instância criada de ***SingleNewFragment*** (já descrito na *página 15*).

Código adicionado em **NewsListFragment**:

public interface Callbacks {

public void onItemSelected(ThothNew thothNew);

}

private static Callbacks sDummyCallbacks = new Callbacks() {

@Override

public void onItemSelected(ThothNew thothNew) {

}

};

Código adicionado em **ClassSectionsActivity**:

… implements NewsListFragment.Callbacks{

@Override public void onItemSelected(ThothNew thothNew) {

if (NewsListFragment.isTwoPane()) {

Bundle arguments = new Bundle();

arguments.putSerializable(TagUtils.TAG\_SERIALIZABLE\_NEW, thothNew);

SingleNewFragment fragment = new SingleNewFragment();

fragment.setArguments(arguments);

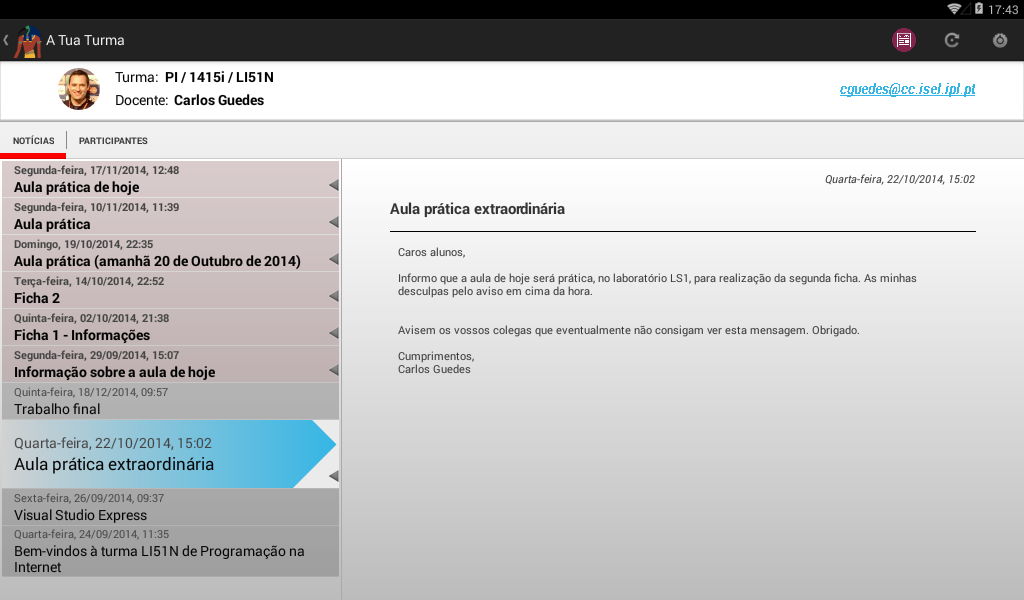
getSupportFragmentManager().beginTransaction()

.replace(R.id.fragment\_container\_detail\_new, fragment)

.commit();

}

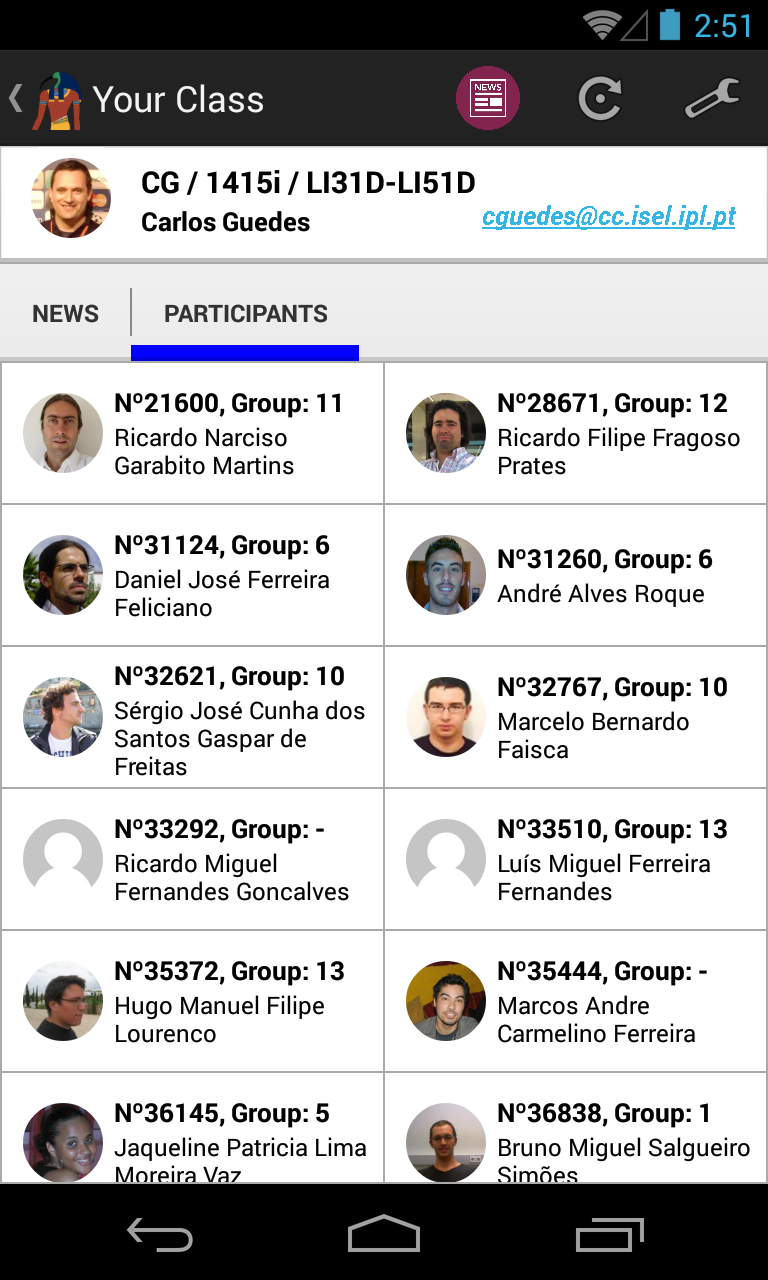
}



**Figura 15**: 2 Painéis - Lista de notícias da classe & detalhe da notícia escolhida - Tablet

### Estudantes inscritos

##### ParticipantsFragment



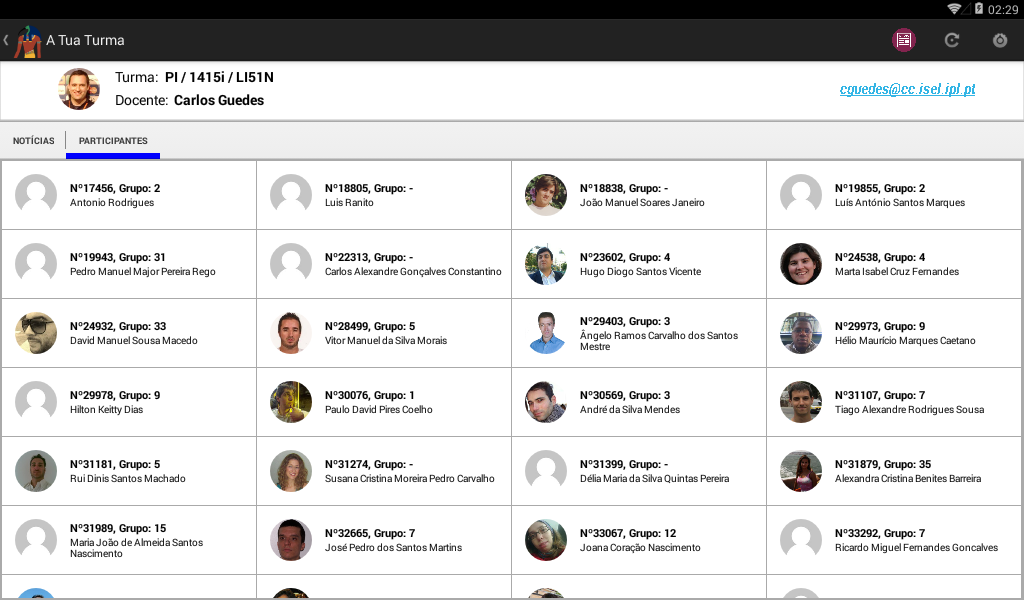
**Figura 16**: Participantes da turma - Phone

Fragmento responsável por mostrar os estudantes inscrito na turma seleccionada. Além da informação como o nome e o número, também é feita a extracção do avatar que cada estudante submeteu na *plataforma* ***Gravatar*** e guardado o URL na *plataforma* ***Thoth***(mais detalhe na Página 20).

Os estudantes estão ordenados por número de estudante e atribuído pelo ISEL.

Os avatars só são carregados quando é feito *swip horizontal* para este fragmento (ocorre ao momento da criação e binding das vistas dos participantes no ***ParticipantsAdapter***).

Como este processo de carreamento de muitos avatars consome alguns recursos foi necessário utilizar mecanismos de assincronismo na afectação das ***ImageView’s***, para que a utilização seja o mais fluído possível.



**Figura 17**: Participantes da turma - Tablet

### Avatar – Handlers + AsyncTask

Tanto a tabela de docentes como na tabela de estudantes têm duas colunas para armazenar os caminhos para obtenção do avatar, um externo e outo interno. Sendo esses:

* ***AVATAR\_URL***: caminho URL para o *gravatar*, extraído da plataforma *Thoth*
* ***AVATAR\_PATH***: caminho para o ficheiro com o bitmap pretendido e armazenado no ***SD*** do dispositivo

No momento de introdução do bitmap é feita uma query a questionar se já existe caminho interno (ou seja, existe um ficheiro com o avatar já criado) e utiliza esse caminho para afectar a ***ImageView*** pretendida com um ***AsyncTask***.

Exemplo de utilização do AsyncTask:

new BitmapUtils.***LoadBitmapTask***(ivStudentAvatar).***execute***(avatarPath);

Caso contrário será necessário a partir do URL, que foi guardado na DB no momento de carregamento do objecto JSON do docente / estudante, fazer:

1. Pedido ***HTTP GET*** ao ***Gravatar***;
2. *Decoding* para *Bitmap*, do ***InputStream*** obtido no passo 1;
3. Afectar ***ImageView*** com o *Bitmap*;
4. Criar um ***ImageFile*** com o *Bitmap* e armazena-lo no SD;
5. Fazer update ao campo ***AVATAR\_PATH***, com o caminho para o ficheiro armazenado.

Exemplo de utilização dos Loopers e Handlers necessários:

String storagePath = BitmapUtils.initStoragePath(mContext, DIR\_PATH\_STUDENT);

String avatarUrl = cursor.getString(cursor.getColumnIndex(ThothContract.Avatars.AVATAR\_URL));

***SetViewAndUpdateHandler*** svh = new SetViewAndUpdateHandler(Looper.getMainLooper(), mContext.getContentResolver());

***ImageHandlerThread*** th = new ***ImageHandlerThread***();

th.start();

***ImageHandler*** ih = new ***ImageHandler***(svh, th.getLooper());

ih.***fetchImage***(ivStudentAvatar, avatarUrl, UriUtils.Students.parseStudentID(id), storagePath);

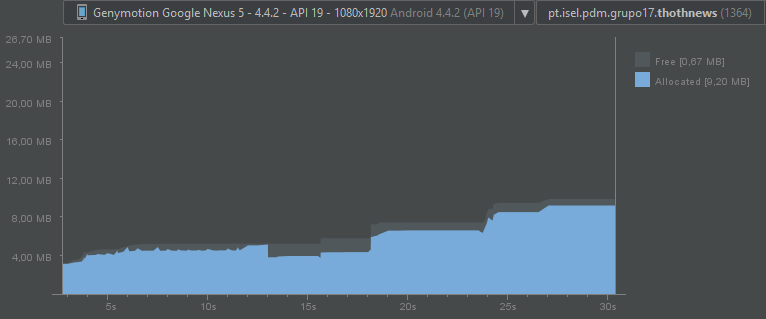
* ***Não foi dado suporte para o cancelamento destas acções***
* ***Pastas/Directórios diferentes para avatars de Docentes e Estudantes***
* ***Diferenciação dos nomes dos ficheiros pelo número de docente/estudante atribuído pelo ISEL***

Manifesto:

<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />

<uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE" />

**Figura 18**: Consumo de memória desde o momento da selecção de turmas até à criação e binding das vistas dos estudantes



ClassesPickActivity

ClassesActivity

ClassSectionsActivity

Handlers

## Funcionalidades Adicionais

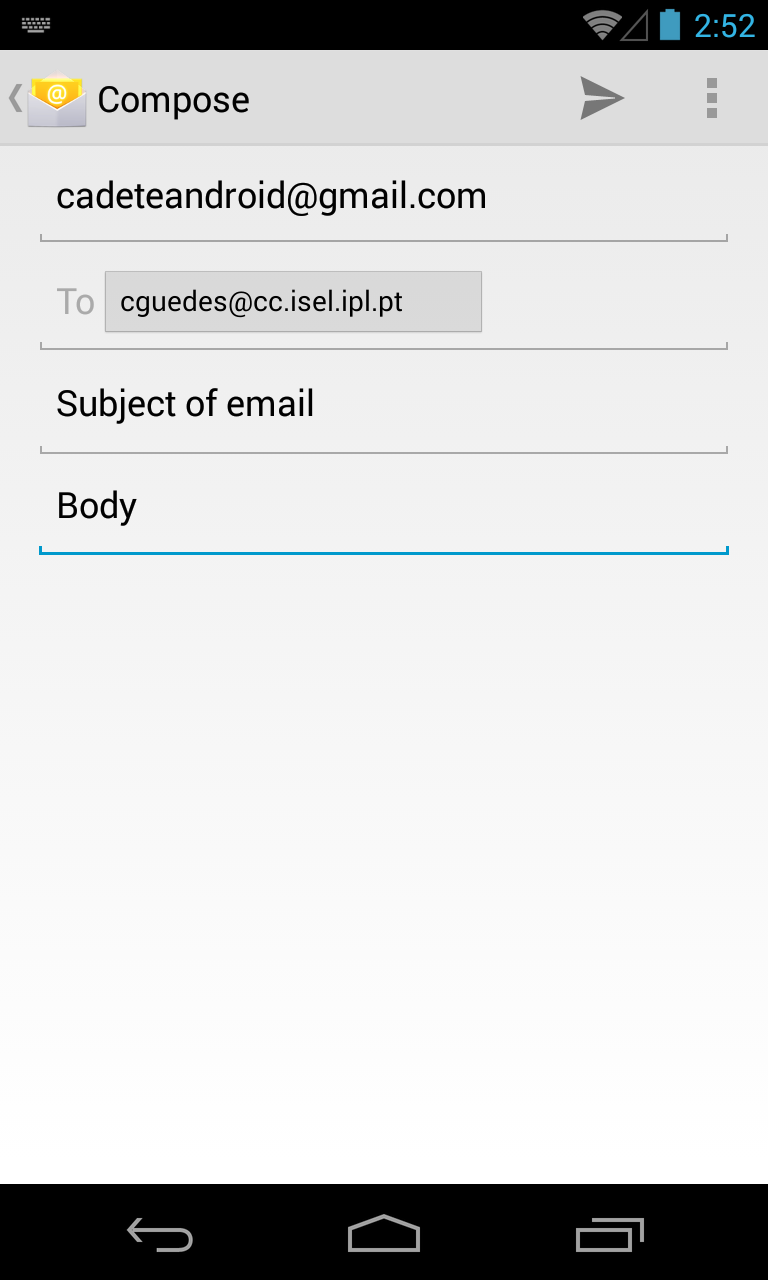
### Multi-Língua

Aplicação ***ThothNews*** suporta *língua* ***Portuguesa*** e *língua* ***Inglesa***. É possível verificar isso nas figuras deste relatório em que o dispositivo virtual de ***telemóvel*** encontra-se configurado com língua ***Inglesa*** e o dispositivo ***Tablet*** como língua preferencial a ***Portuguesa***. Porém, conteúdo retirado da ***framework*** ***Thoth*** encontra-se na língua disponível pela mesma, ou seja a Portuguesa.

### Enviar correio electrónico

Ao clicar sobre o correio electrónico do ***docente*** a azul, que se encontra no topo da vista de uma turma, abrirá (se o utilizador já não tiver definido) um dialog para escolher a aplicação que pretende para enviar o mail. Assim que escolher a aplicação o campo ***To***/***Para*** será preenchido com o correio electrónico do docente.

O mesmo acontecerá se for feito um ***Long Click*** sobre um ***participante*** da grelha de participantes da turma seleccionada.



**Figura 19**: Enviar Email para um docente/participante

view.setOnLongClickListener(new View.OnLongClickListener() {

@Override

public boolean onLongClick(View v) {

Intent i = new Intent(Intent.ACTION\_SEND);

**i.setType("message/rfc822");**

**i.putExtra(Intent.EXTRA\_EMAIL, new String[] {studentEmail});**

i.putExtra(Intent.EXTRA\_SUBJECT, mContext.getString(R.string.send\_email\_subject));

i.putExtra(Intent.EXTRA\_TEXT, mContext.getString(R.string.send\_email\_body));

try {

mContext.startActivity(**Intent.createChooser**(i, mContext.getString(R.string.send\_mail\_to) + studentName));

} catch (android.content.ActivityNotFoundException ex) {

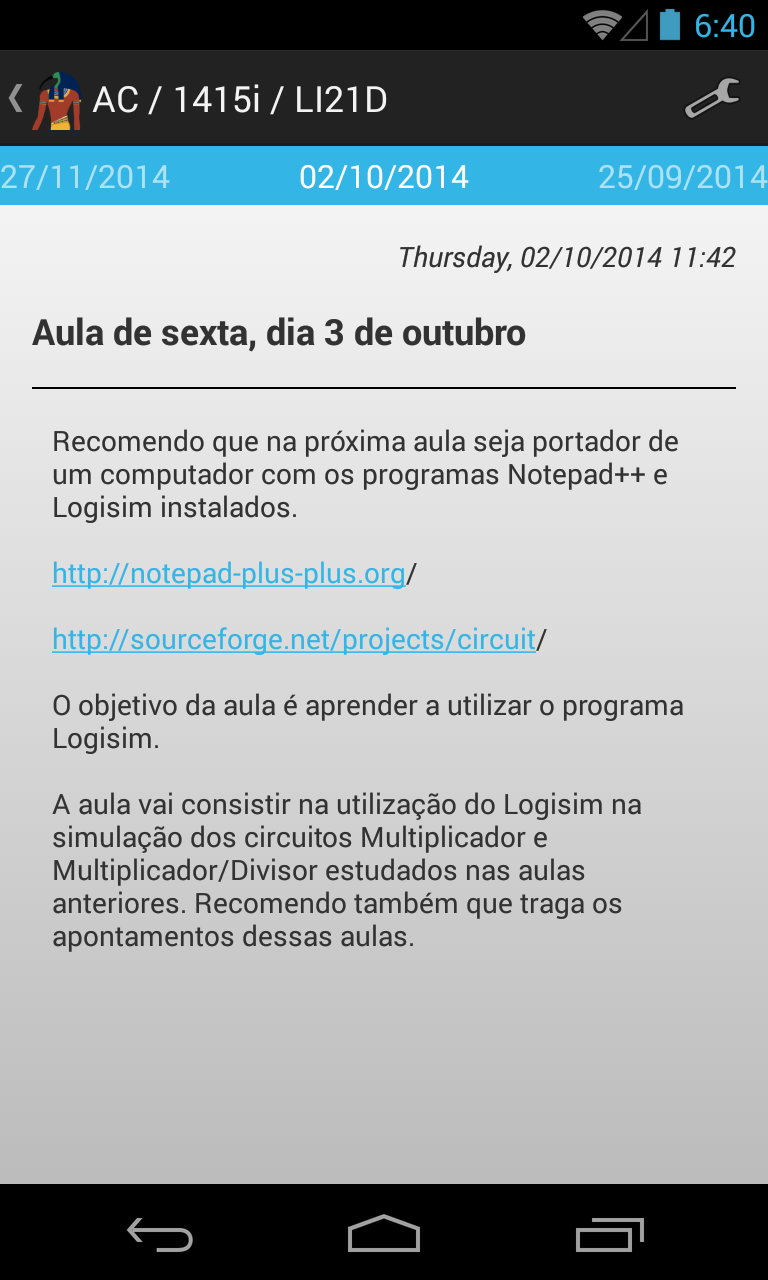
Toast.makeText(mContext, mContext.getString(R.string.send\_mail\_fail\_no\_app), Toast.LENGTH\_SHORT).show();

}

return true;

}

});

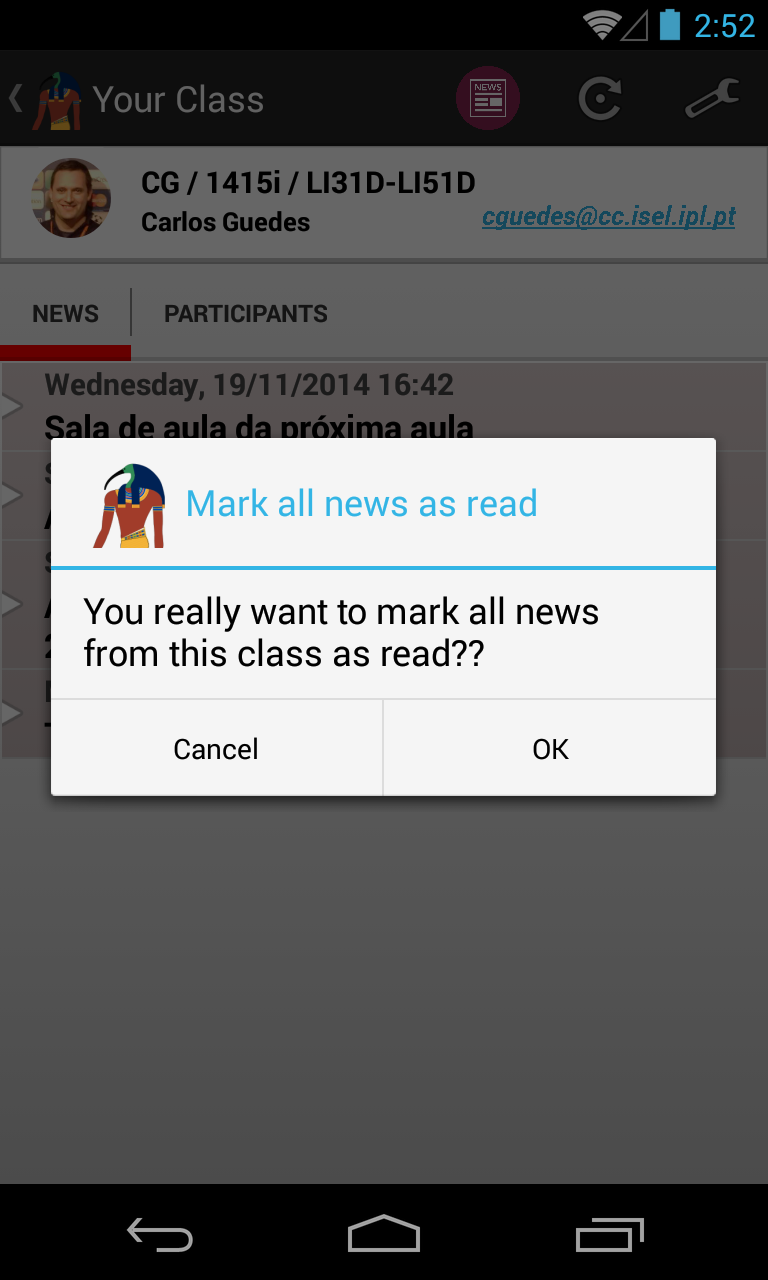


**Figura 20**: demonstração de clickable links

### AutoLink

Adicionando o atributo ***android:autoLink="web"*** ao *TextView* que terá o conteúdo detalhado da notícia e se encontra no Layout utilizado por ***SingleNewFragment***, faz com que a plataforma *Android* consiga converter ***URL’s*** e ***endereços de correio electrónico***, em **Links** que possam ser *clicados* pelo utilizador, abrindo a aplicação responsável para esse tipo de pedido.

### Ler todas as notícias



**Figura 21**: DialogFragment para confirmação

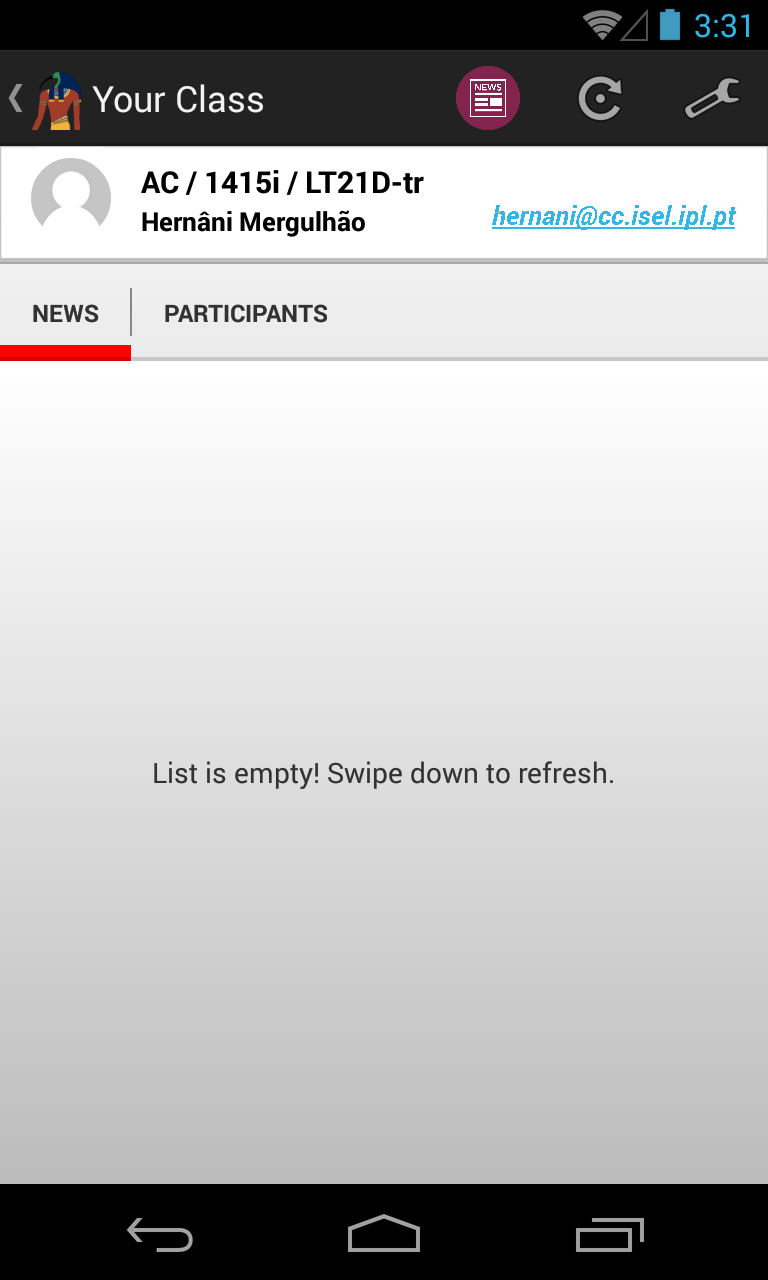
D:\ISEL\14_15_SInv\PDM\MyRep\src\ThothNews\app\src\main\res\drawable\ic_tab_news.png

Ao clicar no símbolo visível na ActionBar aquando o utilizador encontra-se a navegar sobre as secções de uma turma, irá ser mostrado ao utilizador um Dialog para que confirme se quer marcar todas as notícias da turma selecionada como lidas.

É possível cancelar esta operação carregando no botão ***Cancel/Cancelar*** ou fora do dialog.

### SwipeRefreshLayout

##### MultiSwipeRefreshLayout



**Figura 22**: Texto a indicar que a lista se encontra vazia

Possibilidade de o utilizador fazer *swipe down* para que se faça refresh dos dados *(Request HTTP)* e cursor em causa. *(O mesmo acontece ao carregar no do botão de refresh da ActionBar)*

Definido no *layout* e criada uma instância de *MultiSwipeRefreshLayout* no Fragmento a utilizar.

Fragmentos que utilizam esta funcionalidade:

* *ClassesFragment*
* *ClassesPickFragment*
* *NewsListFragment*
* *ParticipantsFragment*

### Esconder Fragmento em Landscape



**Figura 23**: ActionBar Escondida com Phone em Landscape

Para dar maior espaço de usabilidade aos utilizadores de telemóvel, quando o dispositivo encontra-se no estado de ***Landscape*** ou é feita uma rotação para esse estado, é escondido o Fragmento (***layout.setVisibility(View.GONE)*** em ***onConfigurationChanged(Configuration)***) que contém a informação do Docente da turma escolhida.

Manifesto:

<activity android:name=".activities.ClassSectionsActivity"

android:label="@string/label\_class\_section\_view"

android:configChanges="orientation|screenSize|keyboardHidden"/>