**I**nstituto **S**uperior de **E**ngenharia de **L**isboa

Licenciatura em Engenharia Informática e de Computadores

Semestre de Inverno 2014/2015



Projeto Final

PDM

Programação em Dispositivos Móveis

Eng.º Pedro Félix

Data de entrega: 19.01.2015

**Trabalho elaborado por:**

Pedro Lima

Nº. 33684

Sílvia Ganhão

N.º 35418

LI51D – LEIC

Índice

[Introdução 3](#_Toc409347688)

[IselAppServer 4](#_Toc409347689)

[IselApp 6](#_Toc409347690)

[Utils 6](#_Toc409347691)

[Entidades 6](#_Toc409347692)

[CustomAdapters 6](#_Toc409347693)

[AsyncTasks 7](#_Toc409347694)

[ViewModels 7](#_Toc409347695)

[ClassesActivities 7](#_Toc409347696)

[NewsActivities 7](#_Toc409347697)

[ParticipantsActivities 7](#_Toc409347698)

[WorkItemsActivities 7](#_Toc409347699)

[Fragments 7](#_Toc409347700)

[ListModels 8](#_Toc409347701)

[Handlers 8](#_Toc409347702)

[Receivers 8](#_Toc409347703)

[Services 8](#_Toc409347704)

[SyncAdapters 8](#_Toc409347705)

[Funcionamento Geral 9](#_Toc409347706)

# Introdução

O objetivo deste trabalho é a implementação de uma aplicação Android que interage com a ferramenta de ensino Thoth, através da sua API. As funcionalidades presentes na aplicação são:

* A escolha de turmas;
* O visionamento das notícias e dos trabalhos associados a cada uma das turmas escolhidas;
* E ainda é possível ver os participantes de cada turma.

Para a melhor utilização desta aplicação foram criados 2 projetos, o IselAppServer e o IselApp, para que o utilizador possa aceder à informação sem ser necessário ligação à Internet.

As soluções utilizadas para solucionar os diversos desafios encontrados durante a implementação da aplicação são descritas ao longo deste relatório.

# IselAppServer

Esta aplicação foi criada de forma a implementar o Content Provider e a criar a base de dados que dá suporte à informação disponibilizada ao utilizador. É nesta base de dados que a informação fica armazenada para que o utilizador possa aceder-lhe sem que seja necessário fazer qualquer tipo de conexão à Internet. Criou-se uma base de dados SQLite, de nome **IselApp.db**. Para fazer a gestão desta base de dados é necessário implementar um Content Provider.

Na base de dados foram criadas 3 tabelas:

* A tabela **classes**, que armazena a informação relativa às turmas existentes no Thoth;
* A tabela **news**, que armazena a informação relativa às notícias das turmas selecionadas pelo utilizador;
* A tabela **workItems**, que armazena a informação relativa aos trabalhos associados a cada turma escolhida pelo utilizador.

O modelo Entidade-Associação destas tabelas está definido na Figura 1.

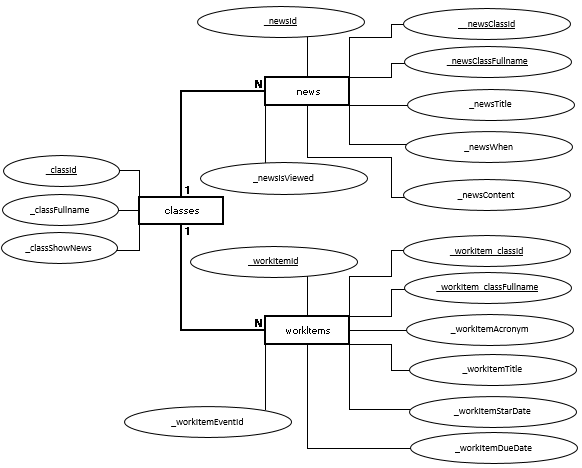


Figura 1- Modelo EA

Para criar a base de dados **IselApp.db** foi definida a classe **IselAppOpenHelper**. De modo a que a classe tenha os requisitos necessários para criar a base de dados é preciso que estenda de **SQLiteOpenHelper**, classe que permite a criação e a gestão de bases de dados. É nesta classe que é criado o ficheiro da base de dados assim como as tabelas que suportam o modelo de dados obtido a partir do Thoth.

Para além desta classe, foram ainda definidas mais 2 classes: a classe **IselAppContentProviderContract** onde está definido o valor do campo **Authority** associado a este Content Provider, que é utilizado pela outra classe implementada, a classe **IselAppContentProvider**. Esta última classe referida é responsável por fazer toda a gestão da base de dados da aplicação (inserts, deletes, updates e querys) através de uma instância da classe **IselAppOpenHelper**, do campo **Authority** e de um **Uri** que identifica univocamente cada uma das possíveis tabelas presentes na base de dados. A combinação destes elementos possibilita a correta gestão da base de dados da aplicação.

Na classe IselAppContentProvider, para além da implementação dos métodos insert, delete, updates e query, estão também implementados 2 métodos que permitem fazer operações de forma transacional. Estes métodos são:

* **applyBatch**: recebe um conjunto de operações e efetua-as todas dentro de uma transação de forma a garantir que todas as operações sejam feitas com sucesso e que caso haja uma exceção todas as operações já executadas sejam anuladas.
* **bulkInsert**: método semelhante ao anterior mas apenas executa operações de insert, também de forma transacional.

# IselApp

É neste projeto que está feita toda a implementação das funcionalidades tidas como objetivo desta aplicação. Para o correto e eficiente funcionamento desta aplicação foram implementadas diversas soluções, recorrendo aos mais variados objetos disponibilizados pelo Android (AsyncTask, Service, Fragments, …).

Para melhor organização do projeto, foram criados vários packages, de forma a separar as classes por funcionalidades.

## Utils

Sendo necessário realizar algumas operações auxiliares, foi criado este package, onde se encontram diversas utilidades:

* **RequestsToThoth** – realiza os diferentes tipos de pedidos ao Thoth (turmas, notícias, trabalhos, participantes);
* **NotificationLaunch** – realiza o lançamento das notificações;
* **CalendarEvents** – auxilia na adição de eventos ao calendário dos trabalhos (workItems) a serem entregues.

## Entidades

De forma a guardar a informação obtida através do Content Provider ou de um pedido HTTP para auxiliar na manipulação dos dados ao longo da execução da aplicação, foram criadas 4 classes representando cada uma delas uma entidade diferente. As classes são:

* **ClassItem** - armazena os dados associados a uma turma;
* **NewsItem** – armazena os dados associados a uma notícia;
* **WorkItem** - armazena os dados associados a um trabalho;
* **ParticipantItem** – armazena os dados associados a um participante.

Todas estas classes implementam a interface **Serializable**, estando o motivo explicado mais à frente no relatório.

## CustomAdapters

De forma a relacionar a view de cada tipo, com a informação conseguida, foram criados Adapters para cada entidade existente:

* **ClassesCursorAdapter** – relaciona a view das turmas com a informação das mesmas;
* **NewsCustomAdapter** – relaciona a view das notícias com a informação das mesmas;
* **ParticipantsCustomAdapter** – relaciona a view dos participantes com a informação dos mesmos;
* **WorkItemCustomAdapter** – relaciona a view dos trabalhos com a informação dos mesmos.

## AsyncTasks

Para obter toda a informação da plataforma Thoth, sobre as entidades existentes neste trabalho foram criadas diferentes AsyncTasks:

* **NewsAsyncTask** – obtém a informação sobre as notícias existentes de uma dada turma;
* **ParticipantsAsyncTask** – obtém a informação sobre os participantes existentes numa dada turma;
* **WorkItemsAsyncTask** – obtém a informação sobre os trabalhos existentes para uma dada turma.

## ViewModels

Para associar os widgets das views de cada tipo, com a informação necessária a ser apresentada, foram criados os seguintes ViewModels:

* **ClassesViewModel** - disponibiliza os widgets da view referente às turmas;
* **NewsViewModel** – disponibiliza os widgets da view referente às notícias;
* **ParticipantViewModel** – disponibiliza os widgets da view referente aos participantes;
* **WorkItemViewModel** – disponibiliza os widgets da view referente aos trabalhos.

## ClassesActivities

Contém a classe **SettingsActivity** onde é implementada toda a funcionalidade referente ao lançamento da atividade onde é possível o utilizador selecionar as turmas.

## NewsActivities

Contém as classes **MainActivity** e **NewsItemActivity**, onde são implementadas as funcionalidades referentes ao lançamento das atividades que mostram a lista de todas as notícias de uma turma, e cada notícia individualmente, respetivamente.

## ParticipantsActivities

Contém as classes **ParticipantsActivity** e **ParticipantItemActivity**, onde são implementadas as funcionalidades referentes ao lançamento das atividades que mostram a lista de todos os participantes de uma dada turma e cada participante individualmente, respetivamente.

## WorkItemsActivities

Contém as classes **WorkItemsActivities** e **WorkItemLinkActivity**, onde são implementadas funcionalidades referentes ao lançamento das atividades que mostram a lista de todos os trabalhos de uma determinada turma, e a página de internet do trabalho em si, respetivamente.

## Fragments

Para apresentação das informações das notícias e participantes foram criados fragmentos, para visualização dos itens por lista e individualmente:

* Para as notícias:
  + **NewsItemListFragment** – Fragmento a partir do qual é apresentada a lista das notícias de uma dada turma;
  + **NewsItemFragment** – Fragmento a partir do qual é apresentada um notícia individualmente;
* Para os participantes:
  + **ParticipantItemListFragment** – Fragmento a partir do qual é apresentada a lista dos participantes de uma dada turma;
  + **ParticipantItemFragment** – Fragmento a partir do qual é apresentado um participante individualmente.

## ListModels

Para obtenção das informações das notícias e participantes de uma dada turma, de forma que essa informação seja passada aos fragmentos, existem dois modelos auxiliares, que suportam a obtenção da mesma: **NewsListModel** e **ParticipantListModel**.

## Handlers

O processo de obtenção da imagem do participante é algo que requer um cuidado especial, pois devido ao número de imagens que são necessárias carregar, associado ao facto de por cada imagem se efetuar um pedido HTTP, torna o processo muito demorado o que impossibilita que seja feito na UI thread. Devido a este problema foram criadas três novas classes de forma a implementar uma solução que recorre a Loopers e Handlers:

* **ImageHandlerThread** - Possibilita a criação de uma nova *thread* que possui um Looper, onde é efetuado o processo de obtenção das imagens;
* **ImageHandler** - Associada ao looper originado pela **ImageHandlerThread** tem como função receber as diversas mensagens e processá-las;
* **SetViewHandler** - Associada ao mainLooper, tem como única responsabilidade associar a imagem obtida do pedido HTTP ao ImageView presente no layout correspondente.

## Receivers

Contém a classe **IselAppReceiver**, que tem como função ficar alerta quando houver mudanças no estado da rede, e assim lançar o SyncAdapter, para atualização da informação existente no ContentProvider.

## Services

Contém a classe **IselAppService**, onde é feita a atualização da informação no ContentProvider, após serem selecionadas ou desseleccionadas turmas, atualizando as notícias e os trabalhos da mesma.

## SyncAdapters

O componente SyncAdapter foi criada para encapsular o código de transferência de dados entre a aplicação e o Content Provider. Para tal foi necessária a criação de algumas classes:

* **IselAppSyncAdapter** – Onde se encontra o código de atualização da informação, obtida da plataforma Thoth, existente no Content Provider;
* **IselAppSyncService** – Componente que permite que a framework do SyncAdapter corra o código existente na classe **IselAppSyncAdapter**.
* **IselAppAuthenticator** - A framework SyncAdapter assume que é necessário uma conta com acesso login para poder haver transferência de informação, daí a existência de uma classe Authenticator. Embora nesta aplicação não seja necessário a existência de uma conta, este componente tem de ser criado na mesma.
* **IselAppAuthenticatorService** – Para que o SyncAdapter tenha acesso ao Authenticator é necessário criar um bound Service. Este retorna um objeto Android binder que permite ao SyncAdapter chamar o Authenticator e assim fazer transferência de informação.

# Funcionamento Geral

## Notícias

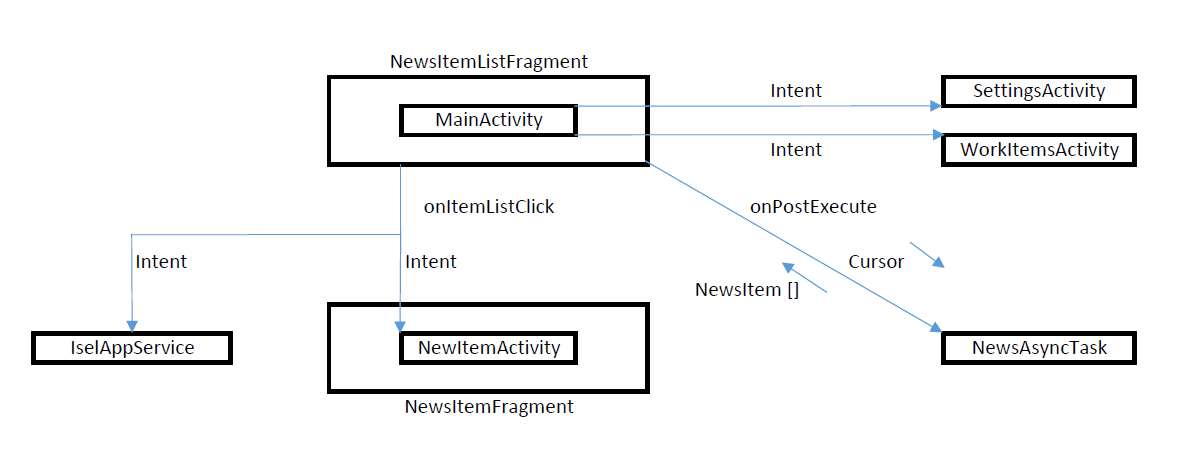


Figura 2 – Schema referente às atividades das Notícias

## Turmas