**3ª Serie**

**Programação em Dispositivos Moveis**

Ricardo Nunes - 31656

João Silvestre – 32766

09/06/2012

*Instituto Superior Engenharia de Lisboa*



Engenheiro: Pedro Pereira

Engenharia Informática e de Computadores

Semestre de Verão 2011/2012

Índice

[Introdução 4](#_Toc327031184)

[Estrutura do Trabalho 5](#_Toc327031185)

[Armazenamento do *timeline* em memória persistente 5](#_Toc327031186)

[Submissão de status em modo offline 6](#_Toc327031187)

[Partilha de status por email 6](#_Toc327031188)

[Conclusão 7](#_Toc327031189)

# Introdução

Nesta fase foram implementadas três novas funcionalidades à aplicação: armazenamento de status em memória persistente, submissão de status em modo offline e partilha de status por email.

Como veremos no capítulo seguinte, todas as funcionalidades foram implementadas sem alterações à estrutura usada nas duas primeiras fases.

# Estrutura do Trabalho

## Armazenamento do *timeline* em memória persistente

Este módulo é responsável por armazenar qualquer novo *tweet* em memória persistente e com isto permitir que o utilizador veja o seu *timeline*, mesmo quando não tiver o *wi-fi* ligado.

De maneira a dar continuidade à lógica aplicada nas fases anteriores, o único módulo que teve alterações foi o serviço responsável por obter a *timeline* do serviço *Yamba* (*GetTimelineService*) que, até à 2ª fase, tinha a responsabilidade de retornar o *timeline* do utilizador registado em *TwitterAsync* (com *user* e *password* configurável como *preferences*).

Tendo em conta as novas funcionalidades deste serviço, o mesmo foi alterado para que tenha uma fonte de dados de instancia (neste caso para ler *Tweet*) que é actualizada com os dados recebidos do serviço *Yamba* e de onde os dados para preencher a *TimelineActivity* são lidos, independentemente do sucesso da ligação com o serviço *Yamba*.

Como repositório de *Tweet* usámos o SGBD SQLite e a respectiva biblioteca de acesso presente na API do Android. De todas as classes criadas para suporte desta infra-estrutura destacam-se:

**TweetsSqliteHelper –** Estende de *SQLiteOpenHelper* e é responsável pela criação da BD. Permite a leitura de variáveis que representam o a posição e o nome de cada campo e foi construída de maneira a que se o programador alterar apenas estes valores todos os módulos que os usem continuam a funcionar correctamente, desde que alterados para valores válidos no seu contexto.

**CursorToObjectAdapter<T>** – Interface de um módulo que recebe uma instância de Cursor e converte-o para um objecto. Útil aquando da leitura do resultado de uma *query*. Tem *CursorToTweetAdapter* como única implementação, que cria, preenche e retorna um novo *Tweet* com o tuplo actual do *Cursor*. Como o acesso de um valor num *Cursor* é feito tendo em conta sua posição, este módulo usa as variáveis de posição de campo disponíveis em *TweetsSqliteHelper*.

**ObjectToContentValueAdapter<T> -** Interface de um módulo que converte um objecto para um *ContentValues*, tipo usado para inserção na API de SQLite do Android. Tem como única implementação *TweetToContentValuesAdapter* que retorna um *ContentValues* preenchido com os valores do *Tweet* recebido por parâmetro. Usa os nomes de campos disponíveis em *TweetsSqliteHelper*.

**BaseSqliteSimpleDataSource<T> -** Classe que tem como entrada um *SQLiteOpenHelper*, o nome de uma tabela e as classes necessárias para a conversão entre os objectos a guardar na tabela e os tipos usados pela API do SQLite (*CursorToObjectAdapter* e *ObjectToContentValuesAdapter*) e permite inserção e leitura na BD.

## Submissão de status em modo offline

Tal como a funcionalidade explicada anteriormente, apenas o serviço responsável pela operação foi alterado, neste caso, *StatusUploadService*.

Até esta fase o serviço tentava publicar o novo *status* usando a API do Yamba. Com o objectivo pedido em mente, o serviço foi alterado de maneira a que, caso exista alguma falha na publicação do *status* (*status* retornado pela API igual a *null* ou alguma excepção lançada durante o processo) esse mesmo seja guardado, o serviço registar-se como *BroadcastReceiver* de uma *CONNECTIVY\_ACTION* que, quando detecta que o *wi-fi* já está ligado, repete o processo de publicação do *status*.

## Partilha de status por email

Para esta funcionalidade optámos por tirar proveito dos clientes de *email* presentes no dispositivo. Para tal, adicionámos um botão à vista *Details* que lança a uma *activity chooser* onde o utilizador pode escolher o cliente de *email* pretendido para partilha do *status*. Nesta implementação apenas o conteúdo do *email* (*status* do *tweet*) é enviado entre *activities*.

# Conclusão

Todos os objectivos obrigatórios propostos pelo enunciado foram atingidos. O tratamento de excepções foi melhorado desde a ultima fase mas, apesar de servir para o que foi proposto, podia ser melhorado.

Nesta fase tivemos de contornar o problema de não haver construtor publico de *Status* com o uso de *Tweet* para a *timeline* que nos obrigou a fazer algumas pequenas alterações ao código, que agora converte de forma *lazy* os Status recebidos da API do *Yamba* para *Tweet*, o único tipo usado para representar um *tweet* nesta aplicação.