**2ª Serie**

**Programação em Dispositivos Moveis**

Ricardo Nunes - 31656

João Silvestre – 32766

14/05/2012

*Instituto Superior Engenharia de Lisboa*



Engenheiro: Pedro Pereira

Engenharia Informática e de Computadores

Semestre de Verão 2011/2012

Índice

[Introdução 4](#_Toc324796330)

[Estrutura do Trabalho 5](#_Toc324796331)

[TwitterAsync 5](#_Toc324796332)

[StatusUploadService 5](#_Toc324796333)

[GetTimelineService 5](#_Toc324796334)

[UserInfoPullService 5](#_Toc324796335)

[UserInfoActivity 6](#_Toc324796336)

[AsyncUserInfo 6](#_Toc324796337)

[Conclusão 7](#_Toc324796338)

# Introdução

Neste trabalho foi alterada a componente de comunicação da aplicação Yamba, recorrendo a serviços. Foi também implementado uma nova componente da aplicação, responsável por mostrar os dados do utilizador.

Neste documento encontram-se explicadas as implementações, e a forma como estas foram integradas com a aplicação.

# Estrutura do Trabalho

## TwitterAsync

Houve uma pequena alteração na implementação desta classe, todos os métodos passam agora a necessitar de saber o contexto (*Android* *Context*) em que foram chamados, para que se possa fazer o *startService*.

## StatusUploadService

A responsabilidade deste serviço é de fazer o *upload* do estado do utilizador, este serviço foi implementado com base na classe *Service*. Foi criada uma *AsyncTask*, muito parecida com a já existente mas que para o serviço quando concluí o envio, sendo que o acto de parar o serviço é feito num bloco *finally* para garantir que mesmo em casos de erro o serviço é terminado.

Na integração, com a aplicação, bastou apenas alterar a implementação do método *updateStatusAsync*, sendo que agora em vez de iniciar uma *AsyncTask* este vai começar o serviço, fazendo *startService* sobre o contexto passado no parâmetro.

## GetTimelineService

Este serviço foi implementado recorrendo a uma *TimerTask*, usando um *Timer*, esta permite uma forma fácil de agendar tarefas para serem executadas de uma forma recorrente.

Dado que este serviço é dependente de algumas preferências, este encontra-se à escuta de modificações nestas. Se a actualização automática for activada/desactivada é feito um cancelamento da tarefa que se encontra agendada, é feito um novo agendamento tendo em conta a escolha do utilizador. Quando o tempo entre actualizações é mudado, caso a actualização automática esteja activa, é também feito um novo agendamento da tarefa.

A integração com a aplicação foi feita da mesma forma que a usada no *StatusUploadService*.

## UserInfoPullService

O *bounded* *service* responsável por devolver os dados necessários pela *UserInfoActivity*. Como corre num processo separado, apenas aceita pedidos através de mensagens. Implementa uma operação (GET\_USER\_INFO) que tem como retorno um *bundle* com os 4 campos pedidos.

Por cada novo pedido recebido obtém o utilizador actual de *TwitterAsync* e usa os métodos disponíveis nesta classe (*Twitter*.*User*) para obter o número de status, subscrições e subscritores associados ao utilizador actual.

### Instâncias TwitterAsync

O TwitterAsync disponibiliza métodos estáticos para obter a instância, de forma a garantir que só existe uma. No entanto, como este serviço corre num novo processo o contexto estático é diferente. No entanto, ambos partilham as mesmas *preferences*, logo estes vão estará trabalhar sobre a mesma conta.

## UserInfoActivity

Esta *activity* é responsável por mostrar o nome de utilizador, número de status, subscrições e subscritores do utilizador activo na aplicação e é apenas acessível pelo menu da aplicação.

Os dados são obtidos de forma assíncrona usando *AsyncUserInfo*, pedidos após invocação do método *onResume* desta *activity*. De maneira a pedir os dados é necessário enviar um *GetUserInfoCompletedListener* (*listener* a notificar quando os dados forem retornados de *UserInfoPullService*) que, nesta implementação, afecta o valor das *TextView* presentes no formulário com o valor correspondente.

## AsyncUserInfo

Esta classe é a ponte entre a *UserInfoActivity* e o *UserInfoPullService*, responsável pela interacção com o serviço através do envio e recepção de mensagens.

Após receber um novo pedido de *UserInfoActivity*, guarda o *GetUserInfoCompletedListener* recebido, gera e envia uma *Message* para o *UserInfoPullService* e retorna de imediato. Após receber a resposta de *UserInfoPullService* o *bundle* presente na resposta é interpretado e o *listener* é notificado.

# Conclusão

A estrutura de comunicação, com o serviço, implementada na fase anterior permitiu que estas alterações fossem realizadas de uma forma fácil e quase transparente para a aplicação. No entanto, a forma como o *bounded* *service* foi implementado carece de cuidados, e espera-se conseguir uma melhor integração deste, com o resto da aplicação, na próxima fase.