ISEL
INSTITUTO SUPERIOR
DE ENGENHARIA DE LISBOA

YambaPDM

RELATÓRIO FINAL

Programação em Dispositivos Móveis

32766 – João Silvestre

31656 – Ricardo Nunes

16 de Junho de 2013

CONTEÚDO

YambaPDM

1 INTRODUÇÃO

Foi-nos proposto, na unidade curricular de Programação em Dispositivos moveis, desenvolver uma aplicação cliente do serviço Yamba. No final do desenvolvimento desta app é suposto os alunos terem conhecimentos básicos do sistema Android e estarem sensíveis a alguns dos problemas do desenvolvimento para dispositivos moveis (mais especificamente para esta plataforma). Julgamos que atingimos o objectivo e que temos o conhecimento que era expectável termos do sistema Android e de como os vários blocos se interligam.

Este relatório serve como guia do nosso trabalho e está dividido em 4 tópicos: camada de acesso ao Yamba (camada criada sobre a API de Yamba disponibilizada e serviços usados), base de dados, activities e requisitos opcionais.

2 ACESSO AO YAMBA – TWITTERASYNC

A comunicação com o serviço é feita através da UI, logo, toda esta interação será executada na *thread* da UI. A comunicação com um serviço *online* deve ser sempre considerada uma ação lenta que, quando executada na *thread* da UI, irá dar ao utilizador a sensação de bloqueio, tornando a sua experiência de utilização má. De forma a evitar isto deve-se usar uma outra *thread* para executar estas ações, o Android oferece mecanismos, tais como as AsyncTask, para facilitar este processo.

De forma a tornar todo este processo transparente na UI foi criada a classe TwitterAsync que serve de *proxy* para a classe Twitter, que implementa um modelo assíncrono para aceder ao serviço. Este permite que sejam registados *listeners* para diversas ações, sendo estes chamados na *thread* da UI pela tarefa correspondente. Por exemplo, o *listener* StatusPublishedListener é chamado quando a tarefa acaba de enviar um estado.

2.1 PRIMEIRA VERSÃO

Esta secção tem como objetivo descrever a versão inicial da API de acesso ao Yamba.

2.1.1 TwitterAsyncTask

Existe um aspeto em comum, partilhado por todas as tarefas, que implementam uma determinada ação do Yamba API, e este aspeto é o facto de todas precisarem de um objeto Twitter para poderem efetuar a ação. Desta forma, a TwitterAsyncTask garante que esse objeto exista e seja o mais atual possível, ou seja, com as configurações atuais, pois o utilizador pode mudar de utilizador e/ou serviço.

2.2 SERVICES

Numa segunda versão foram implementados serviços para obter a *timeline* e publicar estados. Esta troca permitiu uma maior segurança quanto à execução das tarefas em questão, isto pois o android pode a qualquer momento decidir terminar *activities*, seja por falta de memória ou um outro qualquer critério.

Tendo tarefas de longa execução, associadas a uma atividade, estas corriam sempre o risco de serem terminadas a meio, isto porque o android pode a qualquer momento terminar uma tarefa caso necessite de mais recursos. Um serviço tem uma maior resiliência a estes processos de terminação automáticos, isto pois foi determinado que primeiro serão sempre terminadas as atividades, principalmente se não estiverem a ser mostradas.

A troca de AsyncTasks para serviços não teve qualquer impacto nas atividades, isto porque estas fazem uso da API TwitterAsync para aceder ao serviço.

2.2.1 TimelinePullService

O serviço "TimelinePullService" corre sempre em background desde o início da aplicação até que esta seja terminada, isto permite manter a *timeline* sempre atualizada mesmo quando o utilizador não se encontra na vista. Este serviço vai ainda guardar a timeline numa base de dados SQLite, permitindo uma consulta rápida nas próximas vezes que o utilizador volte à aplicação, ao invés de ficar a espera que se carreguem os novos pode ver imediatamente os últimos guardados.

No caso de não haver acesso à internet é parado o timer que atualiza a timeline, na eventualidade do acesso à internet voltar com a aplicação aberta é enviada uma notificação ao serviço, o qual irá arrancar o timer novamente, permitindo que a timeline volte a ser atualizada.

2.2.2 UserInfoPullService

O serviço "UserInfoPullService" é responsável por obter a informação do utilizador, este serviço corre num processo separado o que faz com que se tenha que ter alguns cuidados com a sua implementação. A informação obtida do serviço tem que ser encapsulada num bundle e enviada através de um Messenger, de forma a permitir ultrapassar a barreira dos processos.

2.2.3 StatusUploadService

Este IntentService é usado para publicação de novos status no Yamba e sabe executar 2 tarefas (a distinção entre que tarefa executar é feita no Intent enviado ao serviço):

- Publicação de status – recebe a mensagem a publicar e, caso haja conectividade, publica o status, caso contrário (ou caso a publicação falhe por outra razão que não a conectividade), a mensagem é posta na base de dados da timeline mas com o atributo Published a falso – status pendentes.
- Publicação de todos os status pendentes – Esta operação lê todos os status pendentes presentes na BD e publica-os, um a um. Caso a sua publicação seja sucedida o status pendente é retirado da tabela. Esta operação apenas é usada pelo BroadcastReceiver presente em YambaApplication, sensível a mudanças na conectividade (quando detecta que existe conectividade invoca este serviço).

2.3 Tratamento de Exceções

Dado que todas as ações são executadas em background é necessário permitir que a UI possa receber estas exceções, e trata-las como desejar. O mecanismo implementado, embora não sendo o ideal, permite que isto aconteça através de um *listener* especial, registado de forma estática, que é chamado sempre que uma exceção acontece.

Mas quem chama este *listener*? Esta é mais uma das responsabilidades da TwitterAsyncTask, que apanha todas as exceções e notifica quem estiver registado no *listener*, usando a *thread* da UI.

3 DATABASEMANAGER

A API de acesso à base de dados foi dividida em três partes:

- Classe DatabaseManager, o ponto de entrada para a base de dados, onde é gerido todo o processo de criação e atualização da mesma
- Package Versions, especificação de todas as versões da base de dados, isto permite ir criando versões progressivas que o DatabaseManager sabe usar para actualizar uma qualquer versão até à mais actual
- Package Model, onde existem os objetos do modelo divididos em duas partes
    - <Tabela>.class, classe simples onde contém propriedades para as colunas da tabela
    - <Tabela>DataSource.class, classe que permite operações CRUD sobre a tabela em questão

3.1 ESTRUTURA DA BASE DE DADOS

Existe apenas uma tabela na base de dados, esta tabela é onde são armazenados todos os estados.

Timeline (ID INT, SERVERID INT, MESSAGE TEXT, PUBLICATION_DATE TEXT, REPLY_TO INT, PUBLISHED INT)

A coluna ID é uma coluna identity local permitindo identificar unicamente o estado, a coluna SERVERID é o identificador do lado do serviço Yamba. As colunas MESSAGE, PUBLICATION_DATE e REPLY_TO são a mensagem, a data de publicação e, caso tenha sido uma resposta, o identificador do estado ao qual serviu como resposta, respectivamente.

A coluna PUBLISHED é um indicador interno se o estado já foi enviado para o serviço, quando não existe conexão esta coluna permite identificar quais os estados a submeter quando a conexão for recuperada.

3.2 VERSIONAMENTO

Por cada versão da base de dados deve ser criado uma classe que estende de DatabaseVersion, esta força a criação de scripts de criação desta versão e de upgrade da versão anterior. Esta versão deve ser depois registada no DatabaseManager, adicionando-a no fim do array DATABASE_VERSIONS e alterando a referência para a versão atual (CURRENT_VERSION) para esta.

O DatabaseManager sabe então executar todas as alterações necessárias de forma a atualizar a base de dados da versão em que esta se encontra até à atual.

4 ATIVIDADES

4.1 TIMELINEACTIVITY

Esta atividade faz uso do serviço TimelinePullService para a obtenção dos estados mais recentes, fazendo depois uso de uma lista para mostrar os mesmos. Nesta atividade é possível navegar para todas as outras atividade e é ainda possível enviar o conteúdo da *timeline* na forma de um email.

4.1.1 Envio da timeline por email

O envio da *timeline* é feito recorrendo a um "intent chooser", para o tipo "message/rfc822", que permite escolher entre todos os clientes de email instalados no dispositivo. O email criado nesta versão é apenas uma versão "plain-text" dos estados, no entanto, facilmente poderia ser criado um email HTML para mostrar o conteúdo numa forma mais apelativa.

4.2 DETAILSACTIVITY

Esta activity é mostrada quando é escolhido um tweet da lista da timeline, mostrando todos os dados de um tweet:

- O texto do Tweet
- Criador do Tweet
- Data de Criação
- O ID do Tweet

Esta atividade permite também o envio do estado por email, da mesma forma que é permitido o envio da timeline.

4.3 STATUSACTIVITY

Esta activity serve para publicação de novos estados, mostrando para isso:

- Uma caixa de texto que contem o texto do novo estado;
- Um botão, a ser pressionado para envio do estado escrito;
- Uma label com o número de letras ainda disponíveis para o novo estado.

Após o pressionar o botão a mensagem é validada e, caso cumpra com os requisitos (tamanho entre 1 e o número de letras máximo definido nas preferências), a mensagem é enviada ou, caso a publicação falhe, armazenada para tentar mais tarde.

4.4 PREFERENCESACTIVITY

Activity que estende da classe PreferencesActivity, permite a configuração dos settings desta app:

- Utilizador, password e endpoint para autenticação no Yamba;
- Número de tweets máximo a guardar em memória;
- Enable e período de auto-refrescamento da timeline;
- Número de caracteres máximo num novo estado.

4.5 USERINFOACTIVITY

Esta atividade faz uso do serviço UserInfoPullService para apresentar informações sobre o utilizador. Esta associa-se ao serviço através da funcionalidade de "bind" fornecida pelo android, o que garante que este serviço só se executa enquanto a atividade está a ser usada.

Como o serviço corre num outro processo é feito uso de uma ServiceConnection para tratar da comunicação entre processos, esta faz uso dos listeners do TwitterAsync para divulgar a existência de novas informações, na tentativa de homogeneizar a comunicação com o serviço Yamba.

5 REQUISITOS OPCIONAIS

Não foi possível ao grupo implementar os requisitos opcionais, no entanto, vai-se tentar explicar como se poderia ter abordado este problema.

5.1 APPWIDGET

A widget faz uso de *broadcast* para comunicar com a aplicação, seria necessário implementar um para receber notificações do widget, e um ContentProvider para fornecer os dados armazenados.

O "broadcast receiver" quando fosse notificado que o widget estava ativo teria que se arrancar o TimelinePullService, para que o widget mostra-se constantemente os estados mais recentes. Seria depois usado um ContentProvider (que se encontra implementado em pt.isel.pdm.yamba.content.YambaTweetsProvider.java) para a obtenção dos dados armazenados na base de dados.

5.2 NOTIFICAÇÕES NA STATUSBAR

5.2.1 Estados por ler

Para a implementação destas notificações bastava notificar o utilizador sempre que se recebesse novos estados, isto pois temos os últimos estados guardados e sabe-se quando se recebem estados mais recentes. Poder-se-ia, adicionalmente, guardar o último estado lido e fazer uso deste para quando o utilizador visita-se a aplicação, por via da notificação, mostrar automaticamente o último estado por ler, caso este ainda estive-se visível na aplicação.

5.2.2 Sucesso ou Insucesso da submissão offline dos estados

Sempre que efetuada a ação de submissão dos estados offline, lançar uma notificação com base no resultado desta submissão. Em caso de falha, poderia eventualmente mostrar qual o estado no qual ocorreu o erro.

6 CONCLUSÃO

Este trabalho permitiu aprender a melhor utilizar o sistema operativo Android, este faz uso de diversos conceitos para implementar, corretamente, uma aplicação. O modelo de programação do Android é bastante versátil permitindo implementar de várias formas a mesma funcionalidade, cada forma com as suas vantagens e desvantagens.

No entanto, em algumas situações o grupo ficou preso em bugs criados por especificidades do modelo de programação do android que, muitas das vezes, induz o programador em erro. Um destes caso é, por exemplo, forçar algumas chamadas a métodos da base, algo que poderia ter sido facilmente evitado, criando um ambiente mais seguro para o programador da aplicação.