Integração com webservice

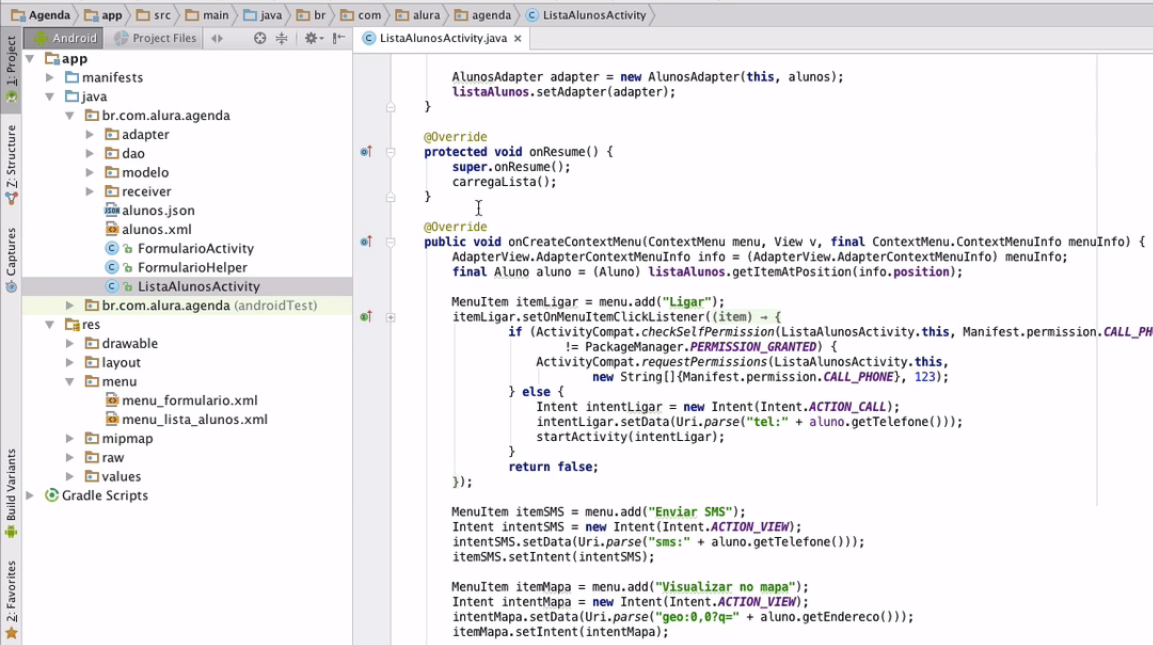
Até o momento, a nossa aplicação tem uma quantidade de dados pequena. Uma quantidade de cadastro que poderíamos considerar grande seria ter mil alunos cadastrados - que não seria pesado para o banco de dados SQLite. No entanto, existem aplicações que precisam trabalhar com um número grande de dados ou que precisam realizar tarefas não indicadas para o próprio dispositivo. Por exemplo, fazer um cálculo muito complexo.

Nestes casos, o recomendável é tirar o processamento da aplicação e enviá-lo para um serviço remoto. Um situação assim acontece quando vamos baixar os posts do Facebook, que estão em um servidor para a aplicação. Após baixados uma vez, já ficarão no cache local.

Na nossa aplicação, também devemos saber nos comunicar com um serviço externo, para salvar dados ou fazer algum cálculo mais complexo. Agora, nós iremos simular que temos um tipo de tarefa não realizável no celular, que será enviada para um servidor externo. Nosso foco será em fazer a **comunicação** e não a tarefa no servidor - isto será visto em **Java Web**.

Nós cadastramos vários alunos na Agenda e todos têm uma nota. Nós iremos mandar estes dados para o servidor, que irá calcular a média das avaliações e irá devolver o resultado. O cálculo é simples, mas iremos simular que poderia ser complexo - o procedimento seria o mesmo, com uma tarefa diferente.

Para fazermos isto, adicionaremos um botão de menu que irá disparar a ação. Iremos voltar ao arquivo ListaAlunosActivity.java.



Iremos sobrescrever o método onCreateOptionsMenu e depois, iremos inflar o menu.

@Override  
public boolean onCreateOptionsMenu(Menu menu) {  
 return super.onCreateOptionsMenu(menu);  
}

Temos o arquivo menu\_lista\_alunos.xml, que o Android Studio cria por padrão.

Se o Android Studio não criou o arquivo menu\_lista\_alunos.xml automaticamente, basta você ir na pasta /res/menu clicar com o botão direito, New > Menu Resource File colocar o nome menu\_lista\_alunos e colocar esse conteúdo nele:

<?xml version="1.0" encoding="utf-8"?>  
<menu xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 tools:context=".ListaAlunosActivity">  
  
 <item android:id="@+id/menu\_enviar\_notas"  
 android:title="Enviar Notas"  
 android:icon="@drawable/ic\_enviar"  
 app:showAsAction="always" />  
  
</menu>

Vamos inflar o Menu com um getMenuInflater, vamos especificar o xml do menu que queremos inflar e onde queremos fazer isto. Iremos devolver um true para dizer que o menu precisa aparecer para abrir a aplicação.

@Override  
public boolean onCreateOptionsMenu(Menu menu) {  
 getMenuInflater().inflate(R.menu.menu\_lista\_alunos, menu);  
  
 return true;  
}

Teremos que editar o xml. Por padrão, o Android incluirá apenas o item Settings.

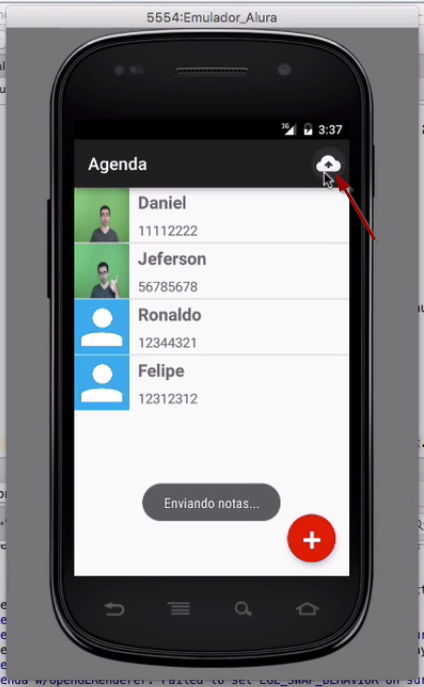
<menu xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools" tools:context=".ListaAlunosActivity">  
 <item  
 android:id="@+id/menu\_enviar\_notas"  
 android:title="Enviar notas"  
 android:icon="@drawable/ic\_enviar"  
 app:showAsAction="always" />  
  
</menu>

Vamos sobrescrever o método onOptionItemSelected, que irá definir o comportamento do menu. Dentro, incluiremos um switch para descobrirmos qual item foi clicado.

@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
 switch (item.getItemId()) {  
 case R.id.menu\_enviar\_notas:  
 Toast.makeText(this, "Enviando notas...", Toast.LENGTH\_LONG).show();  
 break;  
 }  
 return super.onOptionsItemSelected(item);  
}

Adicionando um Toast, com uma duração longo, que usaremos quando estivermos mostrando algo útil.

Se rodarmos a aplicação novamente, veremos o aplicativo com um ícone novo e exibindo o Toast.

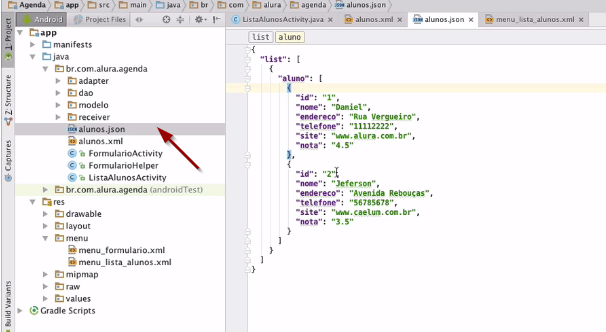


A tarefa que queremos realizar é encontrar uma forma de enviar os alunos para o servidor. A primeira coisa que devemos fazer é conseguir converter os alunos, que são objetos Java, em algo com formato texto, para trafegar os dados pela internet. Para isto, poderíamos usar o formato .txt, .csv ou um formato mais conhecido, o .xml - como o alunos.xml, que já estava preparado aqui.

<alunos>  
 <aluno>  
 <id></id>  
 <nome>Daniel</nome>  
 <endereco>Rua Vergueiro</endereco>  
 <telefone>11112222</telefone>  
 <site>www.alura.com.br</site>  
 <nota>4.5</nota>  
 </aluno>  
</alunos>

Trafegar arquivos .xml é bastante comum. Porém, dispositivos móveis geralmente podem ter conexões com a internet de menor capacidade do que um usuário no desktop. O ideal é encontrarmos formas de minimizarmos a quantidade de dados enviados.

Uma opção é o formato .json, comumente usado quando queremos trafegar dados. Poderíamos replicar o nosso arquivo, mas com uma sintaxe um pouco diferente do anterior.



Nele, reduzimos o uso de chaves e colchetes, o que diminui o número de bytes, tanto para enviar como para receber.

Vamos criar uma classe que irá fazer a conversão do arquivo.

Voltaremos para o ListaAlunosActivity.java, e adicionaremos AlunoDAO, no método onOptionsItemSelected. Também chamaremos o buscaAlunos e guardaremos um lista de alunos.

@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
 switch (item.getItemId()) {  
 case R.id.menu\_enviar\_notas:  
 AlunoDAO dao = new AlunoDAO(this);  
 List<Aluno> alunos = dao.buscaAlunos();  
 dao.close();  
  
  
 Toast.makeText(this, "Enviando notas...", Toast.LENGTH\_LONG).show();  
 break;  
 }  
}

Depois, adicionaremos um conversor. Queremos que ele devolva um json em formato de String. Para verificar se a conversão está correta, em vez de exibirmos Enviando notas..., iremos exibir a *string* json.

@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
 switch (item.getItemId()) {  
 case R.id.menu\_enviar\_notas:  
 AlunoDAO dao = new AlunoDAO(this);  
 List<Aluno> alunos = dao.buscaAlunos();  
 dao.close();  
  
 String json = conversor.converteParaJSON(alunos);  
  
 Toast.makeText(this, "Enviando notas...", Toast.LENGTH\_LONG).show();  
 break;  
 }  
}

Antes, precisamos criar o conversor. Vamos chamá-lo de AlunoConverter e vou instanciá-lo.

AlunoConverter conversor = new AlunoConverter();  
String json = conversor.converteParaJSON(alunos);

Vamos criar o método AlunoConverter e pedir para colocá-lo no pacote converter. Ele já irá sugerir uma assinatura que faça isto.

public class AlunoConverter {  
 public String converteParaJSON(List<Aluno> alunos) {  
 return null;  
  
 }  
}

Em seguida, iremos montar o json no AlunoConverter. Podemos adicionar a parte de *string* de aluno.json, mas ele ficaria muito difícil de ler e de fazer a concatenação:

public class AlunoConverter {  
 public String converteParaJSON(List<Aluno> alunos) {  
 String json = "(/n" +  
 " \"list\": [\n" +  
 " {\n" +  
 " \"aluno\": ["  
 return null;  
  
 }  
}

Para não nos preocuparmos se a concatenação está correta, existe uma classe no Android que irá nos ajudar a montar o json corretamente. Nós iremos usar a classe JSONStringer, que é como um concatenador, mas não abriremos chaves ou colchetes. No JSONStringer, nós iremos abrir com o método js.object(). Precisamos agora adicionar uma chave, que estará adicionada a algum valor. Faremos isto adicionando key e o nome da chave, no nosso caso, list. Em seguida, iremos inserir array.

public class AlunoConverter {  
 public String converteParaJSON(List<Aluno> alunos) {  
 JSONStringer js = new JSONStringer();  
  
 js.object().key("list").array().object().key("aluno").array();  
 }  
}

Até aqui, já fizemos o seguinte trecho no arquivo alunos.json.

{  
  
 "list": [  
 {  
 "aluno": [  
 {

Para trabalhar com a classe JSONStringer, teremos que tratar o exception do object. Faremos isto usando o comando ALT + ENTER e selecionando "Surround with try/catch".

public class AlunoConverter {  
 public String converteParaJSON(List<Aluno> alunos) {  
 JSONStringer js = new JSONStringer();  
  
 try {  
 js.object().key("list").array().object().key("aluno").array();  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
  
 return null;  
 }  
}

Iremos adicionar os alunos da lista. Como teremos que percorre-la, adicionaremos um for. Para cada aluno, teremos que abrir um objeto e depois incluir a chave e o valor. Vamos incluir os dados:

try {  
 js.object().key("list").array().object().key("aluno").array();  
 for (Aluno aluno : alunos) {  
 js.object();  
 js.key("nome").value(aluno.getNome());  
 js.key("nota").value(aluno.getNota());  
 js.endObject();  
 }  
 js.endArray().endObject().endArray().endObject();  
} catch (JSONException e) {  
 e.printStrackTrace();  
 }

Observe que nós fechamos as listas e os objetos.

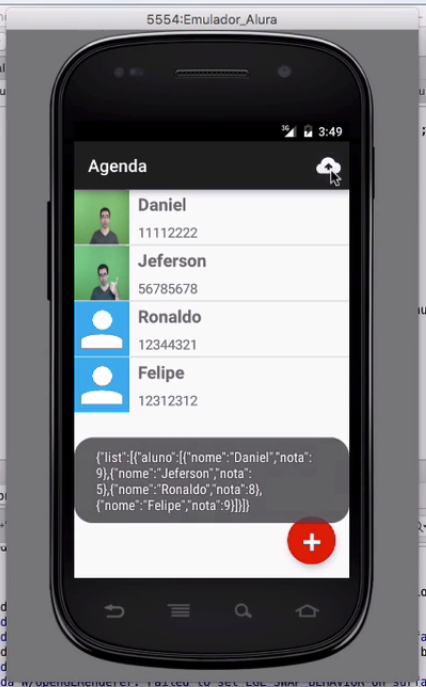
Nós apenas replicamos, o formato feito no json. Agora, precisamos devolver a *string*, que iremos montar com o toString.

try {  
 js.object().key("list").array().object().key("aluno").array();  
 for (Aluno aluno : alunos) {  
 js.object();  
 js.key("nome").value(aluno.getNome());  
 js.key("nota").value(aluno.getNota());  
 js.endObject();  
 }  
 js.endArray().endObject().endArray().endObject();  
} catch (JSONException e) {  
 e.printStrackTrace();  
}  
  
return js.toString();  
}  
}

Agora, a gente vai só passar o json pro Toast

@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
 switch (item.getItemId()) {  
 case R.id.menu\_enviar\_notas:  
 AlunoDAO dao = new AlunoDAO(this);  
 List<Aluno> alunos = dao.buscaAlunos();  
 dao.close();  
  
 String json = conversor.converteParaJSON(alunos);  
  
 Toast.makeText(this, "json, Toast.LENGTH\_LONG).show();  
 break;  
 }  
}

Tudo está pronto, vamos testar se o novo ícone funciona.



Após clicarmos, veremos que aparecerá o nome dos alunos e as notas. Com isto estamos prontos para fazer a comunicação no servidor.