06

Testando verificação da versão

Transcrição

Implementamos o método que verifica se a versão vinda do servidor com o *Firebase*, é ou não é mais recente que a versão interna.

Mas como saberemos que esse método está funcionando como esperamos. Inicialmente podemos colocar alguns `Log.i()` que vai nos indicar o que está acontecendo.

Inicialmente vamos colocar um `Log.i("versao externa", versao)` dentro do método `sincroniza()` da classe `AlunoSincronizador`. Ele vai nos dizer qual é a versão que veio do servidor.

```
// ...

public void sincroniza(AlunoSync alunoSync) {
    String versao = alunoSync.getMomentoDaUltimaModificacao();

    Log.i("versao externa", versao);

    if(temVersaoNova(versao)) {

        preferences.salvaVersao(versao);

        AlunoDAO dao = new AlunoDAO(context);
        dao.sincroniza(alunoSync.getAlunos());
        dao.close();
    }
}

// ...
```

Agora precisamos verificar a versão interna, por isso vamos no método `temVersaoNova()`, colocamos `Log.i("versao interna", versaoInterna)`.

```
// ...

private boolean temVersaoNova(String versao) {

    if(!preferences.temVersao()){
        return true;
    }

    SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS");

    try {
        Date dataExterna = format.parse(versao);
        String versaoInterna = preferences.getVersao();

        Log.i("versao interna", versaoInterna);
```

```
            Date dataInterna = format.parse(versaoInterna);
            return dataExterna.after(dataInterna);
        } catch (ParseException e) {
            e.printStackTrace();
        }

        return false;
    }

    // ...
```

Também precisamos da versão atual, a que foi salva como versão interna. Voltamos ao método `sincroniza()` e adicionamos após o `preferences.salvaVersao(versao)` o log `Log.i("versao atual", preferences.getVersao())`

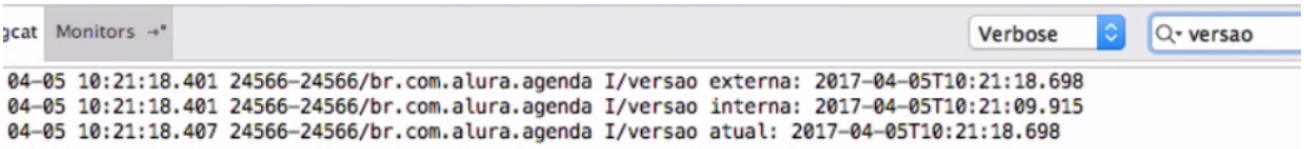
```
    // ...

    public void sincroniza(AlunoSync alunoSync) {
        String versao = alunoSync.getMomentoDaUltimaModificacao();

        Log.i("versao externa", versao);

        if(temVersaoNova(versao)) {

            preferences.salvaVersao(versao);

            Log.i("versao atual", preferences.getVersao());

            AlunoDAO dao = new AlunoDAO(context);
            dao.sincroniza(alunoSync.getAlunos());
            dao.close();
        }
    }

    // ...
```

No método `buscaAlunoCallback()` usávamos um `Log.i("versao", preferences.getVersao())` para verificar a versão. Para que ele não apareça no *Android Monitor* e nos confunda, vamos apenas comentá-lo.

```
    // ...

    @NonNull
    private Callback<AlunoSync> buscaAlunoCallback() {
        return new Callback<AlunoSync>() {
            @Override
            public void onResponse(Call<AlunoSync> call, Response<AlunoSync> response) {
                AlunoSync alunoSync = response.body();
                sincroniza(alunoSync);

    //          Log.i("versao", preferences.getVersao());

                bus.post(new AtualizaListaAlunoEvent());
                sincronizaAlunosInternos();
            }
```

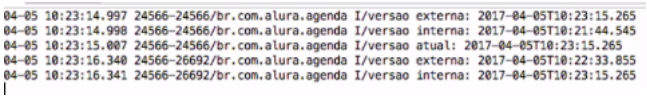
```
        @Override
        public void onFailure(Call<AlunoSync> call, Throwable t) {
            Log.e("onFailure chamado", t.getMessage());
            bus.post(new AtualizaListaAlunoEvent());
        }
    };
}

// ...
```

Executando a aplicação, percebemos tudo ainda está funcionando normalmente. Podemos fazer uma *swipe* para analisarmos o Android Monitor, percebemos que ele trouxe a versão "externa", comparou com a "interna" e colocou a mais recente.



Podemos testar mais vezes que vamos notar que ele sempre vai salvar a mais recente. Mas será que vai funcionar com o *Firebase* quando o celular perde a conexão?

Vamos colocar o celular em modo avião e editar o nosso aluno como **Paulo Silveira**. No servidor vamos colocar o mesmo aluno como **Paulo Souza**. Mas agora quando tirarmos do modo avião, rapidamente vamos fazer um *swipe*. Notamos que a aplicação pegou a versão "externa" que foi a do *swipe*, verificou com a versão "interna" e percebeu que é mais recente colocando como atual. Depois recebeu uma outra versão "externa" do *Firebase*, mas agora não substituiu porque a versão do *Firebase* é mais antiga.



Agora sim, conseguimos padronizar e evitar os problemas que poderiam acontecer por estarmos usando o *Firebase*.

Mas ainda tem alguns detalhes. Esse padrão do método `sincroniza()` estamos usando apenas o *Callback* do `buscaAlunoCallback()`, mas será que só esse *Callback* que faz essa mesma operação? Não, temos também o `sincronizaAlunosInternos()` que faz praticamente os mesmo passos.

Vamos melhorar isso, no `sincronizaAlunosInternos()`, vamos remover todas as chamadas ao `dao` de dentro do `onResponse()` e fechar a conexão com `dao.close()` após o `final List<Aluno> alunos = dao.listaNaoSincronizados()`. Agora dentro do `onResponse()`, vamos delegar para o `sincroniza(alunoSync)`.

```
// ...

private void sincronizaAlunosInternos(){
    final AlunoDAO dao = new AlunoDAO(context);

    final List<Aluno> alunos = dao.listaNaoSincronizados();

    dao.close();

    Call<AlunoSync> call = new RetrofitInicializador().getAlunoService().atualiza(alunos);
```

```
        call.enqueue(new Callback<AlunoSync>() {
            @Override
            public void onResponse(Call<AlunoSync> call, Response<AlunoSync> response) {
                AlunoSync alunoSync = response.body();
                sincroniza(alunoSync);
            }

            @Override
            public void onFailure(Call<AlunoSync> call, Throwable t) {

            }
        });


    }

    // ...
```

Não vamos alterar no `deleta()` pois ele não faz uso do `alunoSync`. Agora de fato todos os passos de conexão estão padronizados e centralizados no método sincroniza, sem correr o risco de salvar um versão antiga no banco de dados.