04

Salvando apenas a data mais recente

Transcrição

Fizemos o primeiro passo que é centralizar o método responsável por sincronizar o aluno. Porém, vimos que o *Firebase* funciona de uma forma que não conseguimos controlar. Pode acontecer que quando recuperarmos a rede, o envio da informação ocorra depois de minutos, ou até mesmo, segure a informação - já sendo antiga no momento em que for enviada para aplicação.

Por correr esse risco, verificaremos se a versão recebida é mais recente do que a já disponível.

No método `sincroniza()` da classe `AlunoSincronizador()`, faremos uma verificação adicionando `if(temVersaoNova(versao))`.

```
// ...

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();
    }
}

// ...
```

Como o método `temVersaoNova(versao)` não existe, vamos criá-lo dentro da classe `AlunoSincronizador`.

```
// ...

private boolean temVersaoNova(String versao) {
    return false;
}

//...
```

Dentro do método `temVersaoNova(versao)`, ficará toda a lógica de verificação. O primeiro passo é verificar se não possuímos uma versão, afinal, se o usuário instalou a aplicação pela primeira vez, ele não terá uma versão antiga, por isso, faz sentido aproveitar a versão que o servidor possui.

Realizaremos a verificação usando `if(!preferences.temVersao())`. Observe que o método havia sido criado.

```
//...

private boolean temVersaoNova(String versao) {

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

    return false;
}

// ...
```

Caso a primeira verificação resulte falsa, saberemos que temos uma versão. Desta forma, precisamos identificar se a versão que estamos recebendo como parâmetro é mais recente que a versão interna.

Se fizermos um *swipe* e analisarmos o *log* no *Android Monitor*, notaremos que o atributo `"momentoDaUltimaVerificacao"` está em um formato de "data e hora".

Com esse formato é possível fazer a verificação, o problema é que ela está como uma *string*. Por isso, usaremos uma API do Java que extrai os valores e transforma-os em um objeto do tipo de datas. No entanto, criaremos um formatador de datas antes.

Vamos instanciar o formatador com `new SimpleDateFormat()`. Dentro do construtor, usaremos o padrão (***pattern***) que deverá ser utilizado. Respeitando o formato da *string* da data, vamos utilizar o formato `"yyyy-MM-dd'T'HH:mm:ss.SSS"`, ficando como:

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

Agora é só extrair o retorno da instancia em uma variável.

```
// ...

private boolean temVersaoNova(String versao) {

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

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

    return false;
}

// ...
```

Em seguida, usaremos o objeto `format` para formatar as datas com `format.parse(versao)`. O problema é que com essa conversão lançaremos uma ***Exception***, porque o Java não tem certeza se o *pattern* enviado está correto, então, vamos envolvê-lo com um `try / catch`.

```
// ...

private boolean temVersaoNova(String versao) {

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

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

    try{
        format.parse(versao);
    } catch(ParseException e) {
        e.printStackTrace();
    }

    return false;
}

// ...
```

O `format.parse()` retorna um objeto do tipo `Date`, que será guardado na variável `dataExterna`. Agora que temos a data externa, pegaremos a data interna. Ao invocarmos o `preferences.getVersao()`, o retorno será um objeto do tipo `String` que guardaremos em uma variável chamada de `versaoInterna`.

Agora é só chamar o `Date dataInterna = format.parse(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();
        Date dataInterna = format.parse(versaoInterna);
    } catch(ParseException e) {
        e.printStackTrace();
    }

    return false;
}

// ...
```

Com as duas datas, verificaremos se a `dataExterna` virá depois da `dataInterna`. Podemos usar os métodos que a classe `Date` nos oferece, então, usaremos `dataExterna.after(dataInterna)`. O melhor é que o método nos retorna um `boolean`, então, já iremos retorná-lo.

```
// ...

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();
        Date dataInterna = format.parse(versaoInterna);

        return dataExterna.after(dataInterna);
    } catch(ParseException e) {
        e.printStackTrace();
    }

    return false;
}

// ...
```

Assim estamos deixando tudo padronizado!