11

Buscando apenas alunos novos

Transcrição

Apesar de conseguirmos pegar a versão das informações do servidor, ainda estamos pegando todos os dados, e não só as informações mais recentes.

Como podemos fazer isso? Uma ideia seria cria um método que verifica se temos a versão, e se caso tenha uma versão mais nova que ainda não temos, ele pega a mais recente. Caso a aplicação não possua nenhuma versão, ele solicita todas as informações.

Na classe `AlunoSincronizador` vamos criar um método chamado `buscaTodos();` . Dentro do método vamos verificar primeiro se tem uma versão.

Para verificar precisamos da referência do *Shared Preferences* que está isolada no `onResponse()` . Vamos transformá-lo em uma atributo da classe com o atalho "Ctrl + Alt + F", selecionando a classe `AlunoSincronizador` .

O atributo foi criado, porém a instância ainda está dentro do `onResponse()` , então vamos colocá-lo no construtor da classe.

```
// ...

private final Context context;
private EventBus bus = EventBus.getDefault();
private AlunoPreferences preferences;

public AlunoSincronizador(Context context){
    this.context = context;
    preferences = new AlunoPreferences(context);
}

public void buscaTodos(){

}

// ...
```

Agora no método `buscaTodos()` , vamos verificar com um `if()` o `preferences.temVersao()` .

Como não temos o método vamos criá-lo na classe `AlunoPreferences` usando o atalho "Alt + Enter" e selecionado *create method*.

Dentro do método `temVersao()` vamos chamar o `!getVersao().isEmpty();` para verificar se está vazio, se estiver será retornado `false` , se não estiver será retornado `true` , indicando que tem versão.

```
// ...

public boolean temVersao(){
    return !getversao().isEmpty();
```

```
}

// ...
```

Agora que já conseguimos verificar se já temos uma versão, caso seja verdadeiro, vamos chamar uma outra função chamada `buscaNovos()`. Lembrando, como não temos o método implementado ainda, vamos usar os atalhos da IDE para criá-lo.

Dentro do `buscaNovos();` vamos fazer uma chamada para o servidor pedindo a versão que ainda não temos. Para fazer isso vamos fazer `new RetrofitInicializador().getAlunoService().novos();`.

```
// ...

public void buscaTodos(){
    if(preferences.temVersao()){
        buscaNovos();
    }
}

public void buscaNovos(){
    new RetrofitInicializador().getAlunoService().novos();
}

// ...
```

Ainda não implementamos o `novos()`, então usamos o atalho "Alt + Enter" para criar o método na interface `AlunoService`.

Dentro da interface `AlunoService`, vamos mudar a assinatura de `void` para `Call` e vamos anotar com a método que queremos acessar, no caso como queremos pegar vamos anotar com `@GET`. Agora o endereço que vai buscar os alunos colocaremos como `@GET("aluno/diff")`, *diff* é o que geralmente é usado quando queremos pegar a diferença de um valor que já temos. Nossa interface vai ficar assim:

```
package br.com.alura.agenda.services;

public interface AlunoService{

    @POST("aluno")
    Call<Void> insere(@Body Aluno aluno);

    @GET("aluno")
    Call<AlunoSync> lista();

    @DELETE
    Call<Void> deleta(@Path("id") String id);

    @GET("aluno/diff")
    Call novos();
}
```

Isso ainda não é o suficiente, ainda precisamos enviar nossa versão como parâmetro para o servidor consiga comparar as versões que ainda não possuímos.

```
// ...

@GET("aluno/diff")
Call novos(String versao);

// ...
```

Mas os parâmetros precisam ser alguma entidade do HTTP. O nosso parâmetro vai ser um ***HEADER***, que atua por meio de chave e valor. Então colocamos a anotação `@Header("datahora")` antes do parâmetro.

```
// ...

@GET("aluno/diff")
Call novos(@Header("datahora") String versao);

// ...
```

Utilizamos como chave o valor `"datahora"` porque a API que que estamos usando é baseada em data e hora.

Conseguimos enviar os dados para o servidor, mas além de enviar também queremos pegar o retorno. A API trabalha em cima do **DTO** então vamos colocar o retorno como `<AlunoSync>` .

```
// ...

@GET("aluno/diff")
Call<AlunoSync> novos(@Header("datahora") String versao);

// ...
```

Voltando a classe `AlunoSincronizador` a chamada do método `novos()` vai estar sinalizando que tem algo errado, e tem mesmo, não passamos a nossa versão como argumento.

Vamos chamar o `preferences.getVersao()` e guardar o retorno em uma variável chamada `versao` . Agora basta passarmos o `versao` como parâmetro. Vamos armazenar o retorno da chamada em uma variável `call` .

```
// ...

public void buscaTodos(){
    if(preferences.temVersao()){
        buscaNovos();
    }
}

public void buscaNovos(){
    String versao = preferences.getVersao();
    Call<AlunoSync> call = new RetrofitInicializador().getAlunoService().novos(versao);
}

// ...
```

Poderíamos agora chamar o `call.enqueue()` passando como argumento um função de *callback*, mas repare que vamos repetir os mesmo passos de quando pedíamos todos os alunos.

A melhor maneira de abordarmos isso é extrair toda a `callback` que já estava criada no `buscaAlunos()`. Vamos extrair para um método chamado `buscaAlunoCallback()`.

```
// ...

public void buscaTodos(){
    if(preferences.temVersao()){
        buscaNovos();
    }
}

public void buscaNovos(){
    String versao = preferences.getVersao();
    Call<AlunoSync> call = new RetrofitInicializador().getAlunoService().novos(versao);
    call.enqueue(buscaAlunoCallback());
}

public void buscaAlunos(){
    Call<AlunoSync> call = new RetrofitInicializador().getAlunoService().lista();
    call.enqueue(buscaAlunoCallback());
}

@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());
        }
    };
}

// ...
```

Não podemos esquecer que no método `buscaTodos();` tratando apenas para buscar novos alunos, precisamos que caso a validação retorne falso, buscar todos os alunos. Então vamos adicionar um `else` e colocar dentro uma chamada para o método `buscaAlunos();`.

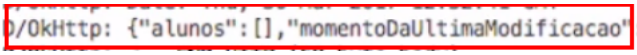
```
// ...

public void buscaTodos(){
    if(preferences.temVersao()){
        buscaNovos();
    } else {
        buscaAlunos();
    }
}

// ...
```

Note que não faz mais sentido chamar o método `buscaAluno()` fora da classe `AlunoSincronizador`, devemos chamar sempre o `buscaTodos()`. Vamos colocar a visibilidade do `buscaAluno()` como `private`.

Precisamos mudar todas as chamadas ao `buscaAlunos()` que forma feitas fora da classe para `buscaTodos()`. Então como mudamos para privado, basta usarmos o atalho "Ctrl + F9" para o ***Gradle*** reconstruir o projeto e verificar todos os pontos onde vão quebrar com a mudança da visibilidade. Após o *Gradle* localizar todos, podemos alterar as chamadas para `buscaTodos()`.

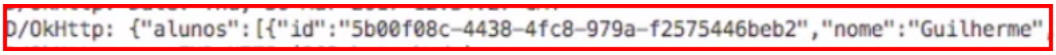
Após refatorar, rodamos o *Gradle* novamente para que ele valide se tudo está funcionando corretamente.

Com tudo certo, vamos executar a aplicação e testar. Com a aplicação aberta, fazemos um *swipe* e analisando o Android Monitor, vemos que ele não trouxe nenhum aluno.



Ele não retornou nada porque não temos nenhum contato novo no servidor. Para testar que realmente está funcionado, vamos acessar o servidor.

Para que o teste seja perceptível, é necessário **desabilitar** o ***firebase***. Com o *firebase* desabilitado, vamos adicionar um novo aluno no servidor e fazer um *swipe* no aplicativo. E ele busca apenas o novo aluno.



Conseguimos de fato pedir os alunos pela versão, melhorando e economizando a banda do usuário.