01

Introdução da remoção do aluno offline

Transcrição

Conseguimos implementar as funções de inserir e editar um aluno em nossa aplicação, mesmo estando offline. E remover, qual seria o comportamento? Recebemos a mensagem de que não é possível deletar o aluno.

Por que isso está acontecendo? Acessaremos a classe `ListaAlunoActivity`, que implementa esse menu de contexto. Dentro do `deletar.setOnMenuItemClickListener()`, nós pediremos uma conexão com o servidor e **apenas quando ela ocorre com sucesso**, deletaremos o aluno.

Esse não é o comportamento que desejamos, por isso vamos remover todo o código responsável por deletar o aluno e colocá-lo fora da *callback*.

```
// ...

MenuItem deletar = menu.add("Deletar");
deletar.setOnMenuItemClickListener(new MenuItem.OnMenuItemClickListener() {

    AlunoDAO dao = new AlunoDAO(ListaAlunosActivity.this);
    dao.deleta(aluno);
    dao.close();
    carregaLista();

    Call<Void> call = new RetrofitInicializador().getAlunoService().deleta(aluno.getId());

    call.enqueue(new Callback<Void>() {
        @Override
        public void onResponse(Call<Void> call, Response<Void> reponse){

        }

        @Override
        public void onFailure(Call<Void> call, Throwable t){
            Toast.makeText(ListaAlunosActivity.this, "Nao foi possivel remover o aluno", Toast.l
        }
    });

    return false;
});

// ...
```

Outro detalhe é que estamos instanciando a `call`, chamando o método `call.enqueue()` e criando a função de *callback* com um método que não deveria ter essa responsabilidade.

Vamos isolar usando o atalho "Ctrl + Alt + M", nomeando o método como `deleta`, depois, clicaremos em "OK".

```
// ...

MenuItem deletar = menu.add("Deletar");
deletar.setOnMenuItemClickListener(new MenuItem.OnMenuItemClickListener() {

    AlunoDAO dao = new AlunoDAO(ListaAlunosActivity.this);
    dao.deleta(aluno);
    dao.close();
    carregaLista();

    deleta(aluno);

    return false;
});

// ...
```

O novo método extraído ficará:

```
// ...

private void deleta(Aluno aluno){
    Call<Void> call = new RetrofitInicializador().getAlunoService().deleta(aluno.getId());

    call.enqueue(new Callback<Void>() {
        @Override
        public void onResponse(Call<Void> call, Response<Void> reponse){

        }

        @Override
        public void onFailure(Call<Void> call, Throwable t){
            Toast.makeText(ListaAlunosActivity.this, "Nao foi possivel remover o aluno", Toast.l
        }
    });
}

// ...
```

Porém, o método `deleta()` também não é responsabilidade da classe `ListaAlunosActivity` e, sim, da `AlunoSincronizador`. Poderíamos delegar o método, porém delegamos quando queremos criar uma classe nova. Então o ideal seria mover o método.

Selecionaremos o método `deleta()`, usando o atalho "Ctrl + Alt + Shift + t", e escolhemos a opção "Move". Ao fazermos isso, aparecerão algumas opções de classe onde poderemos mover o método, uma delas é `AlunoSincronizador`. Após selecionarmos a classe, iremos clicar em "Refactor". O método `deleta()` foi movido para a classe `AlunoSincronizador`.

Em seguida, removeremos o parâmetro `ListaAlunosActivity listaAlunosActivity` e trocar o argumento dentro `Toast.makeText()` para `context`.

```
// ...
```

```
public void deleta(Aluno aluno){
    Call<Void> call = new RetrofitInicializador().getAlunoService().deleta(aluno.getId());

    call.enqueue(new Callback<Void>() {
        @Override
        public void onResponse(Call<Void> call, Response<Void> response){

        }

        @Override
        public void onFailure(Call<Void> call, Throwable t){
            Toast.makeText(context, "Nao foi possivel remover o aluno", Toast.LENGTH_SHORT).shou
        }
    });
}

// ...
```

Com o método em seu devido lugar, podemos voltar a classe `ListaAlunosActivity`, e ajustar alguns detalhes que mudaram ao mover o método. Nossa classe ficará da seguinte maneira:

```
// ...

MenuItem deletar = menu.add("Deletar");
deletar.setOnMenuItemClickListener(new MenuItem.OnMenuItemClickListener() {

    AlunoDAO dao = new AlunoDAO(ListaAlunosActivity.this);
    dao.deleta(aluno);
    dao.close();
    carregaLista();

    sincronizador.deleta(aluno);

    return false;
});

// ...
```

Executaremos a nossa aplicação, repetindo o passo de colocar no modo avião. Com o celular offline, vamos tentar remover um aluno. Apesar de recebermos a mensagem que não foi possível remover, o aluno foi removido. Como não faz sentido mantermos essa frase, podemos removê-la do método `deleta()`, da classe `AlunoSincronizador`.

```
// ...

public void deleta(Aluno aluno){
    Call<Void> call = new RetrofitInicializador().getAlunoService().deleta(aluno.getId());

    call.enqueue(new Callback<Void>() {
        @Override
        public void onResponse(Call<Void> call, Response<Void> reponse){

        }
```

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

            }
        });
    }

    // ...
```

Executaremos a aplicação novamente. Tiraremos o celular do modo avião, e mesmo forçando o *swipe*, o aluno não foi excluído do servidor. E o pior, como foi excluído apenas localmente, com o *swipe*, o aluno deveria ser puxado de volta para o celular.

Mas por que isso aconteceu? Agora estamos trabalhando com versionamento de dados, e no momento estamos com a mesma versão tanto no servidor quanto no celular. Desta forma, mesmo deletando o aluno localmente, não estamos obtendo uma versão nova. Podemos trazer de volta o aluno excluído alterando alguma informação dele no servidor e fazendo um *swipe* no celular.

Para resolver esse problema, em vez de deletar, marcaremos o aluno como "desativado" e não iremos deixá-lo visível. Assim, quando a aplicação voltar a ficar online, o servidor será avisado de que o aluno foi desativado e vai removê-lo, confirmando a remoção. Após a aplicação receber a resposta, o dado será removido internamente na aplicação.

Esse conceito de remoção é conhecido como ***Soft Delete***, que já vimos anteriormente. Nosso próximo passo será implementar isso.