Mão na massa com Retrofit

*Começando deste ponto? Você pode fazer o* [*DOWNLOAD*](https://s3.amazonaws.com/caelum-online-public/android-chat/stages/capitulo-3.zip) *do projeto completo do capítulo anterior e continuar seus estudos a partir deste capítulo.*

Da forma que fizemos no vídeo anterior, resolvemos o problema. Mas temos alguns problemas com relação a esse código.

* Precisamos criar e manipular *Threads* na mão;
* Sempre que for necessário jogar uma mensagem na tela ou atualizar algum componente com o resultado da requisição, precisamos estar na *UI Thread*. E para isso precisamos carregar sempre uma dependência do contexto onde estamos.

Podemos resolver esses problemas com um componente do Android já conhecido por nós: a **AsyncTask**. Que nos permite realizar tarefas pesadas em background através do método **doInBackground** e processar a resposta no método **onPostExecute**. Mas usando as **AsyncTask**, deixamos de lado um problema:

* Precisamos lidar com uma API de mais baixo nível pra construir e disparar requisições HTTP;
* Precisamos fazer na mão o *Marshal*/*Unmarshal* de JSON.

## **Facilitando a vida com Retrofit**

Em busca de resolver esses problemas comuns no dia-a-dia, a *Square* desenvolveu uma biblioteca chamada **Retrofit**. A ideia do Retrofit é criar, através de mágica (ou metaprogramação), classes clientes baseados em um contrato (interface). Abstraindo de nós detalhes de como montar toda a conexão HTTP até o serviço!

## **Configurando as dependências do Retrofit**

Vamos começar então, configurando as dependências do Retrofit no projeto:

No arquivo **build.gradle** (**Módulo**) vamos adicionar a dependência do Retrofit e da biblioteca que vai cuidar do *Marshal*/*Unmarshal* de JSON:

dependencies {  
  
 ...  
 compile 'com.squareup.retrofit2:retrofit:2.0.2'  
 compile 'com.squareup.retrofit2:converter-gson:2.0.2'  
 ...  
  
}

## **Botando a mão na massa**

Como conversamos, o **Retrofit** se baseia em uma interface que diz as operações que esse serviço possui.

No nosso projeto, já temos alguma classe que possui as funcionalidades da nossa API? Sim, a classe ChatService.

Nos métodos dessa classe nós montamos a requisição HTTP, construímos um JSON e criamos Threads. Como vimos, o Retrofit tem como objetivo de nos abstrair de toda essa complexidade.

O primeiro passo será apagar os atributos e toda a implementação dos métodos dessa classe, para transformá-la em uma interface:

public interface ChatService {  
  
 void enviar(final Mensagem mensagem);  
 void ouvirMensagens();  
}

Agora precisamos adicionar algumas informações sobre as requisições. Por exemplo, quando o usuário executar o método **enviar(Mensagem mensagem)**, queremos na verdade disparar um **@POST** para **polling**, enviando a mensagem no corpo (**@Body**) da requisição:

@POST("polling")  
void enviar(@Body Mensagem mensagem);

Do mesmo jeito, queremos fazer um **@GET** para **polling** no método **ouvirMensagens()**.

@GET("polling")  
void ouvirMensagens();

Nesse momento, o código da MainActivity quebra, já que ChatService agora é uma interface, não temos como instanciá-la.

Já dissemos que o responsável por implementar essa interface será o Retrofit. Então precisamos pedir para ele criar uma implementação da classe **ChatService** para nós:

chatService = retrofit.create(ChatService.class);

Mas tenho certeza que você deve ter se perguntado: "Como ele vai encontrar a URL completa do serviço?". E como ele vai saber que queremos que ele converta os objetos para JSON?

Precisamos configurar o Retrofit para isso. Para essa finalidade, podemos usar o **Retrofit.Builder**. Para ele, passamos a URL através do método **baseUrl**, e a biblioteca que usaremos para transformar objetos em JSON e vice-versa, através do método **addConverterFactory**:

// MainActivity.java  
  
Retrofit retrofit = new Retrofit.Builder()  
 // Altere para o seu IP  
 .baseUrl("http://192.168.0.208:8080/")  
 .addConverterFactory(GsonConverterFactory.create())  
 .build();  
  
chatService = retrofit.create(ChatService.class);

Se você tentar rodar esse código verá que o Retrofit vai lançar uma exceção:

Caused by: java.lang.IllegalArgumentException: Service methods cannot return void.

Por que não podemos retornar **void**? Veremos no próximo vídeo :)