Unidade 7

# 

Aula 1 - Networking - Parte 1

Até agora estivemos preso a aplicativos que trabalham offline (sem conexão com a internet) mas sabemos que a maioria dos apps que temos hoje em dia operam de maneira online.

A conectividade é tudo. Nos primórdios dos smartphones, antes da popularização da internet móvel, a quantidade de aplicativos era bem limitada, e entre um modelo de celular e outro sempre víamos mais do mesmo.

Com a popularização das redes móveis e conectividade dos smartphones, as possibilidades se multiplicaram. Hoje em dia a maioria dos aplicativos estão conectados com a internet de certa forma, seja salvando as preferências do usuário, atualizando o conteúdo do app ou até mesmo para jogos online. Definitivamente nossos aplicativos precisam estar conectados à internet.

Para utilizarmos este recurso, você precisará entender alguns conceitos do mundo web, como o protocolo HTTP, Restful e a camada Service do nosso aplicativo. Então vamos lá.

## REST

O primeiro conceito que precisamos entender é a maneira como um aplicativo conversa com um servidor. Para um app manter seus dados salvos, para serem resgatados em um outro smartphone ou plataforma, é necessário que um servidor opere a seu favor. Este servidor irá guardar os dados, cuidar das regras de negócios, fazer autenticação de usuários atualizar conteúdo e muito mais. Em palavras mais breves o servidor retira toda a complexidade existente do lado do app e a manipula no servidor.

Se referir a esta entidade que “faz tudo” como servidor é um pouco simplista. O nome desta entidade é **web-server.** Arquiteturalmente falando o web-server está no **backend** e o nosso aplicativo está no **frontend.** Dizemos *frontend* pois não se trata apenas do aplicativo, mas de todos os softwares que consomem os web-servers. Por outro lado, o web-server não é o *backend****,*** mas ele está contido nele juntamente com uma série de outros componentes (banco de dados, servidor de arquivos, outros web-servers, etc).

Bom, definidos os termos, vamos agora entender como o *frontend* conversa com o *backend.*

Existem dois estilos arquiteturais que definem como o *frontend* deve ser comunicar com o *backend:* o REST e o SOAP.

SOAP (Simple Object Access Protocol, em português Protocolo Simples de Acesso a Objetos) é um protocolo para troca de informações estruturadas em uma plataforma descentralizada e distribuída. Ele se baseia na Linguagem de Marcação Extensível (XML) para seu formato de mensagem.

REST (Representational State Transfer, em português Transferência de Estado Representacional), é uma abstração da arquitetura da World Wide Web, mais precisamente, é um estilo arquitetural que consiste de um conjunto coordenado de restrições arquiteturais aplicadas a componentes, conectores e elementos de dados dentro de um sistema de hipermídia distribuído. REST é o mais utilizado atualmente e é ele que iremos detalhar.

O objetivo do REST é ignorar os detalhes da implementação dos web-services e oferecer uma interface limpa, com alguns métodos, para o usuário (frontend) consumir. Como dissemos acima, o web-service guarda toda a complexidade de uma aplicação, mas deve existir uma forma de acessar os seus métodos. Este meio é o REST

Ele é frequentemente aplicado à web services fornecendo APIs para acesso a um serviço qualquer na web. Ele usa integralmente as mensagens HTTP para se comunicar através do que já é definido no protocolo sem precisar "inventar" novos protocolos específicos para aquela aplicação.

Você trabalha essencialmente com componentes, conectores e dados. Veja algumas das características do REST:

* Ele usa o protocolo HTTP (verbos, accept headers, códigos de estado HTTP, Content-Type) de forma explícita e representativa para se comunicar. URIs são usados para expor a estrutura do serviço. Utiliza uma notação comum para transferência de dados como XML ou JSON. Iremos detalhar mais este trecho adiante.
* Não possui estado entre essas comunicações, ou seja, cada comunicação é independente e uniforme (padronizada) precisando passar toda informação necessária.
* Ele deve facilitar o cache de conteúdo no cliente.
* Deve ter clara definição do que faz parte do cliente e do servidor. O frontend não precisa saber como o backend armazena dados, por exemplo. Assim cada implementação não depende da outra e se torna mais escalável.
* Permite o uso em camadas também facilitando a escalabilidade, confiabilidade e segurança.
* Frequentemente é criado com alguma forma de extensibilidade.

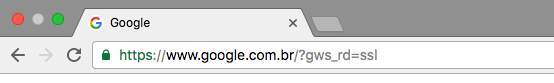
Quando a aplicação cumpre todos os itens acima, dizemos que ela é Restful, ou seja, tem capacidade de fazer REST

Falhando em um dos cinco primeiros itens, a arquitetura não pode ser classificada formalmente como RESTful. Mas nem todo mundo se apega ao formalismo.

## O protocolo HTTP

Já que REST é um conceito arquitetural para comunicação entre plataformas, ele precisa de um protocolo que o regularize.

REST usa como protocolo o HTTP (*Hypertext Transfer Protocol*). Você já deve ter ouvido ou visto esta sigla antes. Ela aparece no seu navegador ao lado da barra de endereço:



Nesta imagem vemos a sigla https (*Hypertext Transfer Protocol Secure*), que é uma extensão do http.

*HyperText Transfer Protocol* que em português significa "Protocolo de Transferência de Hipertexto". É um protocolo de comunicação entre sistemas de informação que permite a transferência de dados entre redes de computadores, principalmente na *World Wide Web (Internet)*.

O HTTP é o protocolo utilizado para transferência de páginas HTML do computador para a Internet. Por isso, os endereços dos websites (URL) utilizam no início a expressão "http://", definindo o protocolo usado. Esta informação é necessária para estabelecer a comunicação entre a URL e o servidor Web que armazena os dados, enviando então a página HTML solicitada pelo usuário.

Para que a transferência de dados na Internet seja realizada, o protocolo HTTP necessita estar agregado a outros dois protocolos de rede: TCP (*Transmission Control Protocol*) e IP (*Internet* *Protocol*). Esses dois últimos protocolos formam o modelo TCP/IP, necessário para a conexão entre computadores clientes-servidores.

### Métodos HTTP

Quando estamos navegando na web, a todo momento o nosso navegador está enviando requisições para um servidor e o servidor, por sua vez, nos devolve uma resposta em um formato específico ou realiza uma ação de acordo com o que pedirmos para ele fazer.  
  
Nas requisições, especificamos o que chamamos de método HTTP ou verbo. Na versão 1.1 do protocolo HTTP(que é a que todos usamos atualmente) temos 9 verbos diferentes.

* **GET**: Requisita um representação do recurso especificado (O mesmo recurso pode ter várias representações, ao exemplo de serviços que retornam XML e JSON).
* **POST**: Envia uma entidade e requisita que o servidor aceita-a como subordinada do recurso identificado pela URI.
* **PUT**: Requisita que um entidade seja armazenada embaixo da URI fornecida. Se a URI se refere a um recurso que já existe, ele é modificado; se a URI não aponta para um recurso existente, então o servidor pode criar o recurso com essa URI.
* **DELETE**: Apaga o recurso especificado.
* **HEAD**: Retorna os cabeçalhos de uma resposta (sem o corpo contendo o recurso)
* **TRACE**: Ecoa de volta a requisição recebida para que o cliente veja se houveram mudanças e adições feitas por servidores intermediários.
* **OPTIONS**: Retorna os métodos HTTP que o servidor suporta para a URL especificada.
* **CONNECT**: Converte a requisição de conexão para um túnel TCP/IP transparente, usualmente para facilitar comunicação criptografada com SSL (HTTPS) através de um proxy HTTP não criptografado.
* **PATCH**: Usado para aplicar modificações parciais a um recurso.

Agora você vai entender o que estamos tentando fazer este tempo todo. Vamos ao exemplo:

Imagine que nossa intenção é criar um app para consulta de filmes. Você cadastraria todos os filmes existentes? Ou pelo menos os filmes dos últimos dez anos? Se sim, boa sorte, pois isto vai tomar uma boa parte do seu tempo. Se você respondeu não, como 99% das pessoas respondem a esta pergunta, você precisa começar a pensar em onde conseguir tantos filmes.

A notícia boa é que existem web-servers que disponibilizam estas informações, e de forma totalmente grátis. Vamos tentar pegar os dados do popular filme **Clube da luta.** Abra no seu navegador o exemplo abaixo:

<https://api.themoviedb.org/3/movie/550?api_key=d272326e467344029e68e3c4ff0b4059>

Implicitamente você está fazendo uma requisição **GET** usando o **HTTPS** e como resultado você receberá um arquivo JSON todo bagunçado. Para melhor visualização, nós pegamos este **JSON** e deixamos de uma forma mais legível para você:

{

"adult": false,

"backdrop\_path": "/8uO0gUM8aNqYLs1OsTBQiXu0fEv.jpg",

"belongs\_to\_collection": null,

"budget": 63000000,

"genres": [

{

"id": 18,

"name": "Drama"

}

],

"homepage": "http://www.foxmovies.com/movies/fight-club",

"id": 550,

"imdb\_id": "tt0137523",

"original\_language": "en",

"original\_title": "Fight Club",

"overview": "A ticking-time-bomb insomniac and a slippery soap salesman channel primal male aggression into a shocking new form of therapy. Their concept catches on, with underground \"fight clubs\" forming in every town, until an eccentric gets in the way and ignites an out-of-control spiral toward oblivion.",

"popularity": 5.56269,

"poster\_path": "/811DjJTon9gD6hZ8nCjSitaIXFQ.jpg",

"production\_companies": [

{

"name": "Regency Enterprises",

"id": 508

},

{

"name": "Fox 2000 Pictures",

"id": 711

},

{

"name": "Taurus Film",

"id": 20555

},

{

"name": "Linson Films",

"id": 54050

},

{

"name": "Atman Entertainment",

"id": 54051

},

{

"name": "Knickerbocker Films",

"id": 54052

}

],

"production\_countries": [

{

"iso\_3166\_1": "DE",

"name": "Germany"

},

{

"iso\_3166\_1": "US",

"name": "United States of America"

}

],

"release\_date": "1999-10-14",

"revenue": 100853753,

"runtime": 139,

"spoken\_languages": [

{

"iso\_639\_1": "en",

"name": "English"

}

],

"status": "Released",

"tagline": "How much can you know about yourself if you've never been in a fight?",

"title": "Fight Club",

"video": false,

"vote\_average": 8.1,

"vote\_count": 5357

}

O resultado desta requisição é um JSON veja que ele realmente possui algumas informações sobre o filme (em inglês). Mas o que é JSON?

## JSON

JSON (JavaScript Object Notation) é uma notação que descreve um objeto JavaScript. JavaScript é uma linguagem amplamente utilizada na web e seus objetos são definidos através da notação JSON.

Esta notação é muito usada, até mesmo nas aplicações mobile por ser leve (arquivos geralmente pequenos) e diminui o tempo de resposta das requisições (pois são menos dados trafegando na rede) e é de fácil compreenção.

JSON usa o padrão “chave-valor” onde para cada chave há um valor. Vejamos um trecho do exemplo acima:

"tagline": "How much can you know about yourself if you've never been in a fight?"

"title": "Fight Club"

"tagline"é a chave para o valor "How much can you..." e é a "title" chave para o valor "Fight Club"

No começo pode parecer estranho, mas logo você pega o jeito. Como o foco do nosso curso não é explicar conceitos de conectividade, passamos por este assunto rapidamente para que possamos fazer uso na aula seguinte, mas aconselhamos que você procure por leituras complementares, em especial sobre o HTTP e o JSON.

## A biblioteca Gson

Na aula 1 da unidade 6, criamos a interface da tela **Entrada (Inbox)**, para preencher as células da ListView criamos um array de objetos Email, pré-definidos. Na vida real não será assim, pois neste caso os emails deveriam vir de um backend em formato JSON. Preparando para isto, vamos aprender agora como manipular um objeto JSON usando a biblioteca Gson.

Gson é uma biblioteca Java que pode ser utilizada para converter objetos em JSON (chamamos este processo de serialização). É possível também fazer o processo inverso (chamado deserialização). O Gson pode trabalhar com objetos arbitrários incluindo objetos pré-existentes que não temos acesso ao código (código fechado).

Existem algumas outras bibliotecas que realizam esta função, porém a maioria requer que seja inserido uma série de anotações Java nas nossas classes. E isto é uma coisa que não conseguiríamos fazer se não tivéssemos acesso ao código-fonte destas classes. A maioria delas também não dão suporte aos Generics do Java, mas o Gson considera estes dois empasses.

### Objetivos do Gson

* Providenciar simples métodos **toJson()** e **fromJson()** para converter objetos em JSON e vice-versa
* Permite realizar conversões em classes com código fechado
* Suporte aos Generics Java
* Permite uma representação personalizada para os objetos
* Suporta objetos complexos arbitrariamente.

Vamos mudar nosso exemplo e passar a usar o JSON como fonte de dados da nossa lista de email. Com isso estaremos a um passo de obter os emails pela internet, via HTTP.

1. Preparamos um projeto como ponto de partida para você. Descompacte o arquivo **Unidade\_7\_-\_Aula\_1\_-\_Exemplo\_1.zip** e abra o projeto no seu Android Studio.
2. Adicionamos um arquivo JSON ao projeto contendo os emails que serão listados. Este arquivo se encontra na pasta assets (**.../app/src/main/assets**), mesma pasta onde colocamos as fontes na unidade passada. Dê uma olhada nele e entenda como é feito um arquivo JSON.
3. Como sempre, adicione uma nova dependência no seu arquivo **build.gradle (Module: app)**:

compile **'com.google.code.gson:gson:2.6.2'**

1. Para transformar este JSON em emails, fazemos o procedimento chamado Deserialização, ou também conhecido como Parsing. Primeiro devemos ler o arquivo, tranformando-o em String, depois o convertemos para um tipo de array especial, o JsonArray, e por último, através deste array criamos um ArrayList de objetos Email. Entre na classe **Utils.java** e insira o método responsável por ler um arquivo do diretório do projeto (chamado bundle) e devolve em formato String:

**public static** String loadJSONFromAsset(String fileName) {

String json = **null**;

**try** {

InputStream inputStream = MailListApplication.*getInstance*().getApplicationContext().getAssets().open(fileName);

**int** size = inputStream.available();

**byte**[] buffer = **new byte**[size];

inputStream.read(buffer);

inputStream.close();

json = **new** String(buffer, **"UTF-8"**);

} **catch** (IOException ex) {

ex.printStackTrace();

**return null**;

}

**return** json;

}

1. Tendo nosso JSON em formato de String o Gson entra em ação. Ainda na classe Utils, adicione o seguinte método:

**public static** ArrayList<Email> parseEmails (String jsonString){

*// Novo parser (serializa/deserializa)*

JsonParser parser = **new** JsonParser();

*// Obtém elemento JSON através parser da jsonString*

JsonElement element = parser.parse(jsonString);

*// Inicializa um ArrayList para guardar os objetos Email*

ArrayList<Email> emailsList = **new** ArrayList<>();

*// Verifica se o elemento JSON contém apenas um JsonObject ou um conjunto deles*

**if** (element.isJsonArray()) {

JsonArray emails = element.getAsJsonArray();

**for** (**int** i = 0; i < emails.size(); i++) {

*// Obtém o JsonObject correspondente do index i do array email*

JsonObject emailJson = emails.get(i).getAsJsonObject();

*// Obtém os valores de cada chave do JsonObject*

String name = emailJson.get(**"from"**).getAsString();

String subject = emailJson.get(**"subject"**).getAsString();

String message = emailJson.get(**"message"**).getAsString();

*// Instancia e adiciona um novo objeto Email na lista de e-mails*

emailsList.add(**new** Email(subject, name, message));

}

}

*// Retorna a lista de e-mails*

**return** emailsList;

}

**Entendendo o código:** A primeira coisa a ser feita é a instanciação de um parser de JSON (JsonParser) e através dele deserializamos o JSON em formato de String usando o JsonElement. JsonElements podem conter um JsonArray ou um JsonObject. Um JsonArray é composto por JsonObjects, e no nosso caso, um JsonObject representa um único e-mail isolado. Como temos uma lista de e-mails, utilizamos o JsonArray. Tendo identificado que nosso JsonElement é um JsonArray, iteramos por este array e instanciamos novos objetos Email baseado no conteúdo de cada JsonObject contido neste JsonArray.

1. Feito isto basta usarmos na nossa classe **InboxFragment**. Primeiro apague o array de e-mails
2. Agora substitua o conteúdo do método **initListView()** pelo seguinte:

**public void** initListView() {

String json = Utils.*loadJSONFromAsset*(**"emails.json"**);

**listView**.setAdapter(**new** InboxAdapter(**this**.getContext(), Utils.*parseEmails*(json)));

}

1. Rode a aplicação e veja o resultado.

Você não verá diferenças na interface do aplicativo pois apenas mudamos o jeito de popular nossa ListView. Apesar do uso do Gson neste momento parecer não ter necessidade, ele acabou limpando a classe InboxFragment. Se tivéssemos 20 mil itens para serem apresentados na lista a classe ficaria muito grande, e ficaria difícil gerenciá-la. Separar os dados da lógica é uma boa prática.

Você verá a real utilidade do Gson na aula seguinte, quando puxarmos os dados (e-mails) da internet.

Aula 2 - Networking - Parte 2

# 

## Padrão de Projeto - Observer

### Objetivo

* Definir uma dependência um-para-muitos entre objetos, de modo que quando um objeto muda de estado, todos os seus dependentes são notificados e atualizados automaticamente.
* Encapsular os componentes núcleo em uma abstração Subject, e componentes variáveis em uma hierarquia Observer.
* A parte "View" do Model-View-Controller.

### Problema

Precisamos interceptar ou receber notificações quando um objeto muda de estado de forma que o fluxo principal do programa não precise esperar por estas notificações, ou seja, de forma assíncrona.

### Discussão

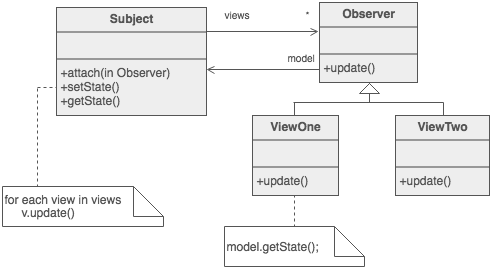
O padrão Observer define um objeto que é o "guardião" do modelo de dados e / ou lógica de negócios (o Subject). Também delega todas as funcionalidades de “View” para desacoplar e dostintiguir objetos Observers. Observers registram-se juntamente com os Subjects ao qual foram cridos. Sempre que o Subject muda, um broadcast é disparado para todos os observers registados.

O protocolo descrito acima especifica um modelo de interação "puxar". Ao invés do Subject “empurrar” o que mudou aos Observers, cada Observer se encarrega de “puxar”.

O padrão Observer capta a maior parte da arquitetura MVC que tem sido uma parte da comunidade Smalltalk por anos.

### Estrutura

O Subject representa a abstração central. O Observer representa a abstração variável do contexto. O Subject solicita aos objetos Observers que realizem tomem uma ação quando um evento acontece. Cada observador pode chamar de volta o Subject, conforme necessário.



## Biblioteca Ion

## Biblioteca Glide

## Considerações finais