



CRIANDO APPS CONECTADOS

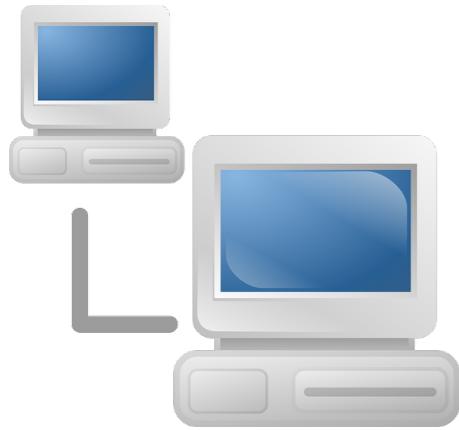
INTRODUÇÃO À REDES



REDES DE COMPUTADORES

Breve História

Breve História

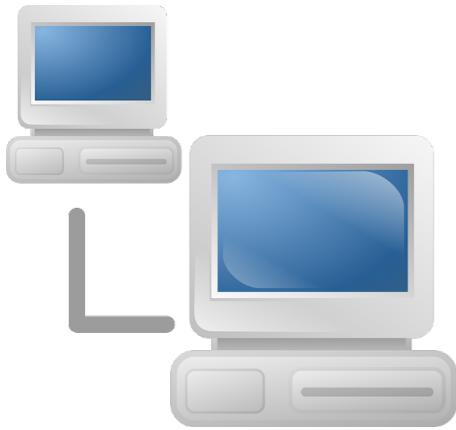


Seus primórdios estão ligados aos meios acadêmicos e militares.

No início da década de 70 o ARPANET foi criado com o objetivo de bases militares e centros de pesquisas.

O ARPANET foi o embrião da Internet, que tomou proporções comerciais na década de 90.

Breve História



Também na década de 90 começaram a se consolidar as tecnologias que permitiam a criação de redes locais e domésticas a baixo custo.

Nos anos 2000 tivemos a grande oferta de conexões de banda larga, tornando a Internet uma *commodity*.

Em meados dos anos 2000, houve grande disseminação da tecnologia 3G, que abriu caminho para o nosso cenário atual de

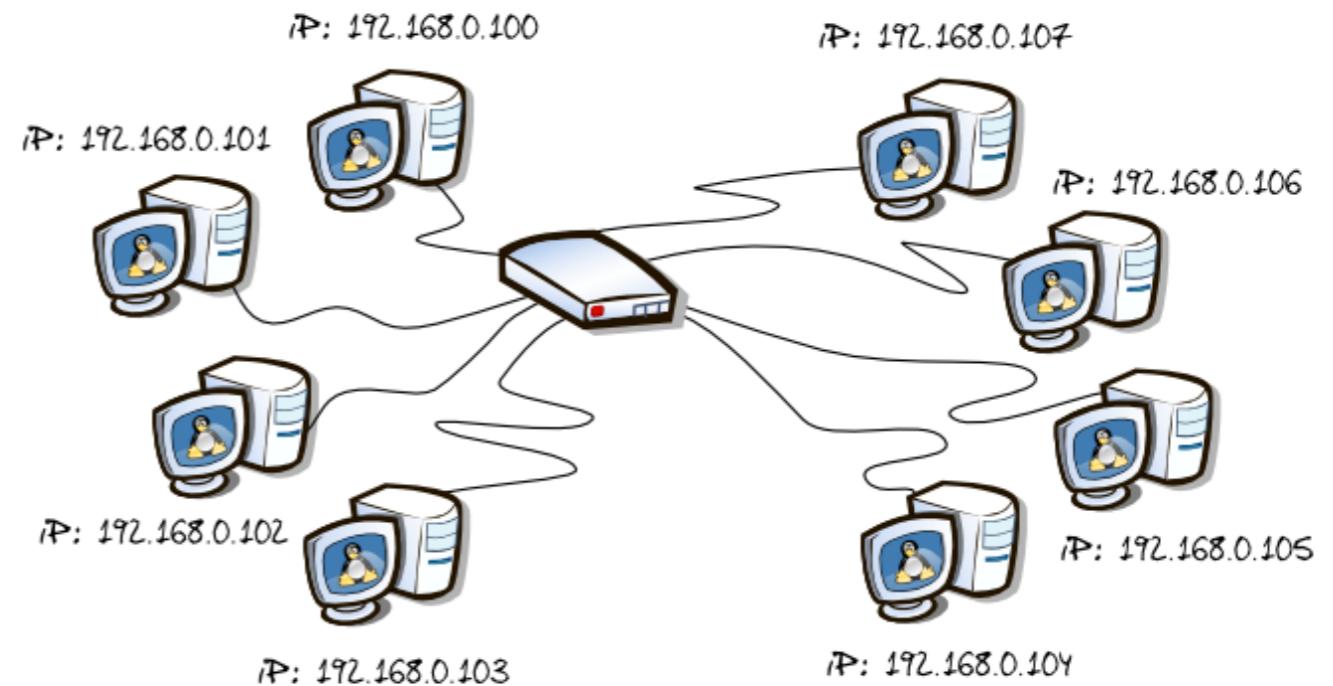
COMUNICAÇÕES EM REDE

**Modelos OSI
Protocolos mais comuns
Conectividade dos dispositivos**

Protocolos mais comuns

IP - Internet Protocol

Protocolo usado para endereçamento de dispositivos de rede



Protocolos mais comuns

TCP - Transfer Control Protocol

Protocolo básico para comunicação em rede

Oferece controle de fluxo e recursos de confiabilidade que garantem entrega.

Em geral não programamos para TCP diretamente, mas usamos algum protocolo construído em cima dele (como **HTTP**, **FTP**, **SSH**, **POP3**, **SMTP**, **IMAP**, etc.)

Complementado pelo protocolo IP, por isso



Protocolos mais comuns

UDP - User Datagram Protocol

Outro protocolo básico para comunicação em rede.

Em contrapartida ao **TCP**, o **UDP** é usado quando precisamos otimizar a comunicação entre dispositivos sem a preocupação com a confiabilidade dos dados.

Não tem garantia de entrega.

Muito usado em aplicações de Streaming.



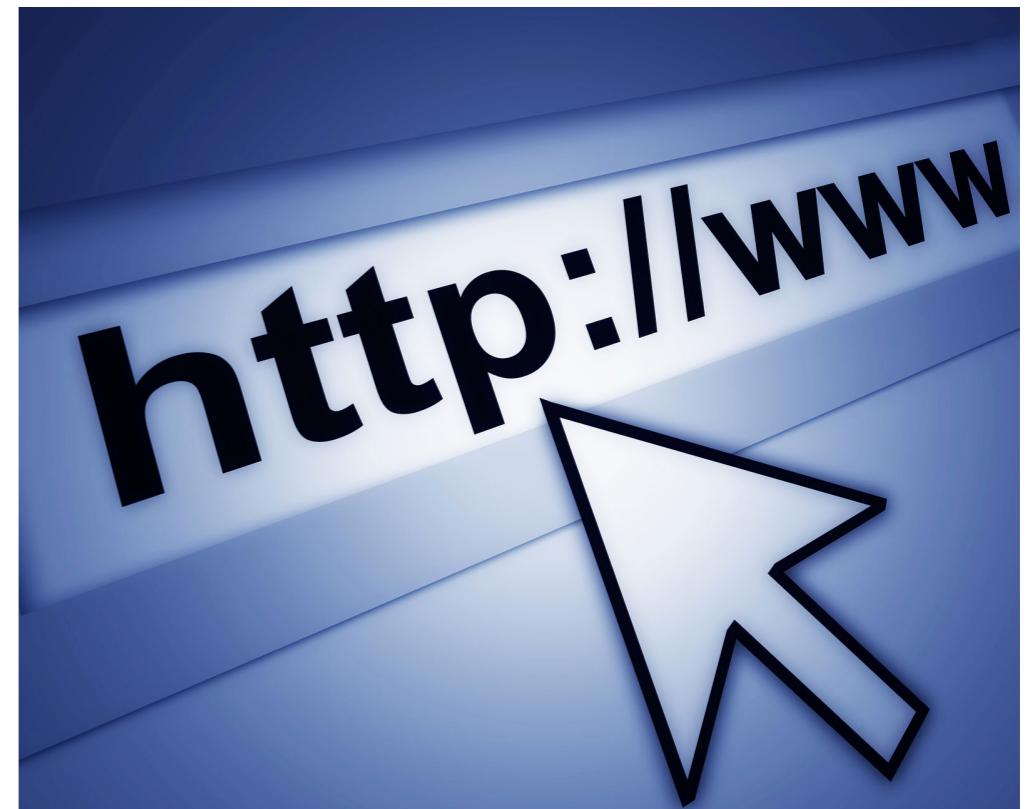
Protocolos mais comuns

HTTP - Hypertext Transfer Protocol

Protocolo base da **World Wide Web**, usado na navegação entre browsers.

Trafega dados em formato de texto (arquivos **HTML**, scripts Javascript, arquivos **XML** e **JSON**).

Será explorado com detalhes nas próximas sessões.



Protocolos mais comuns

TLS/SSL - Transport Layer Security/Secure Socket Layers

Protocolo de segurança na comunicação.

Cria um canal de comunicação criptografado, que inviabiliza a captura dos dados entre as partes que estão se comunicando.

É a base do chamado **HTTP/S**, usado nos sites seguros.



Protocolos mais comuns

DNS - Domain Name Services

Protocolo usado para endereçamento da Internet.

Responsável por traduzir os nomes de domínios que usamos (www.google.com, www.apple.com) em endereços IP dos computadores que desejamos acessar.

Protocolo fundamental para o funcionamento da Internet.



Protocolos mais comuns

DHCP - Dynamic Host Configuration Protocol

Protocolo usado para automatizar a configuração de dispositivos de rede.

Responsável por atribuir endereçamento e outras configurações de rede quando dispositivos se conectam.

Trabalha em conjunto com o protocolo IP.



Conectividade dos dispositivos

Redes Celulares

A maioria dos dispositivos modernos conta com conectividade via rede celular, incluindo alguns modelos de Tablets.

Em geral tem maior latência e velocidade mais baixa, as conexões são menos confiáveis e com tendência a queda ou



Conectividade dos dispositivos

Redes Celulares

Em geral relegadas a operações simples.

Devemos evitar o uso da rede celular quando precisamos tráfegar um grande volume de dados ou quando precisamos de maior estabilidade.



Conectividade dos dispositivos

Redes Wi-fi

Sigla de Wireless Fidelity.

São redes locais com acesso sem fio.

Em geral são mais confiáveis que as Redes Celulares e mais adequadas para fluxo de dados.



“

Ao trabalhar com Apps conectados é importante sempre verificar o estado de conectividade do dispositivo, bem como dimensionar as cargas de trabalho de acordo com a rede ativa no momento.

PROTOCOLO HTTP



Protocolo HTTP

É o protocolo base de comunicação com a Internet.

Por sua simplicidade tanto para criação como para consumo de serviços, tornou-se o padrão para criação de API's Web.

É um protocolo chamado *Stateless*, que significa que as conexões não são persistentes e o estado não é mantido entre chamados feitas entre um mesmo cliente para um servidor.

Modelo Funcional do HTTP

Toda a comunicação HTTP parte de um cliente, que faz uma solicitação, o chamado *Request*.

O servidor recebe e interpreta essa solicitação, processando e produzindo o resultado que é retornado na forma de um *Response*.



Modelo Funcional do HTTP

Logo o fluxo básico da comunicação HTTP

1. Cliente

- Monta um Request
- Abre uma conexão com o Servidor de destino
- Envia o Request para o servidor

2. Servidor

- Recebe o Request
- Interpreta os dados da solicitação
- Processa a solicitação
- Monta um Response
- Envia o Response de volta para o cliente

3. Conexão entre o cliente e o servidor é encerrada



ENTENDENDO O REQUEST

Entendendo o Request

Um *Request* contém todas as características da solicitação a um serviço, tais como:

Dados do solicitante, como o endereço IP e o Agent (por exemplo o Browser que está sendo usado para iniciar a chamada)

O método (Method) da chamada, dentre um conjunto previsto de métodos como **GET** (obter), **POST** (enviar), **PUT** (atualizar), **DELETE** (excluir), etc.



Entendendo o Request

O recurso (**Resource**) que está sendo solicitado, como o caminho para um arquivo ou recurso que o cliente deseja obter.

Um conjunto de cabeçalhos que podem incluir informações complementares descrevendo características.

Entendendo o Request

Como toda a comunicação que acontece em HTTP é em formato textual, todo o *Request* é descrito dentro de um formato bem definido. Uma chamada para o site do Google teria o seguinte formato:

```
GET / HTTP/1.1
Host: www.google.com
Accept: */*
Content-Type: text/html
Content-Length: 0
```

O exemplo ao lado solicita a página principal do Google.

ENTENDENDO O RESPONSE

Entendendo o Response

O **Response** encapsula todas as informações de retorno para uma solicitação feita através de um **Request**.

O **Response** é dividido em duas partes:

Response Header

Response Body

Response Header

O cabeçalho da resposta contém uma série de Metadados para utilização por parte do cliente. Dentre as principais informações, encontramos:

Versão do protocolo HTTP

Código de Status: um código numérico sinalizando o Status do retorno. É onde identificamos se a solicitação foi atendida com sucesso ou se aconteceu alguma modificação ou erro.

Data da resposta (usada para fins de cache) Content-Type: o tipo de conteúdo que esta sendo retornado no

Response Body

Contém o conteúdo da resposta em si. Quando uma página é solicitada, por exemplo, o retorno é o código HTML que a representa. Quando uma imagem é solicitada o que é retornado é uma codificação da imagem.

Exemplo

```
HTTP/1.1 200 OK
Date: Fri, 11 Nov 2016 20:37:31 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
P3P: CP="This is not a P3P policy! See https://
www.google.com/support/accounts/answer/151657?hl=en for
more info."
Server: gws
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
Set-Cookie
NID=90=tHcIDnNn4Va02m_k6L34gRpAYeCs-1NJzCI8SW2kPBIRVPosr
Qc7ioFZjM-280qU4C91KJm1-lc71h0tU18f_ZX-
GeIWrqyXatm9c3a8zlsM0gRF93LSSD_zhqQE8YPJbQpZEuJVf8bGg;
expires=Sat, 13-May-2017 20:37:31 GMT; path=/;
domain=.google.com; HttpOnly
Accept-Ranges: none
Vary: Accept-Encoding
Transfer-Encoding: chunked
```

**Exemplo do
cabeçalho de
retorno de um
Request a página
principal do
Google:**



Exemplo

```
<!doctype html><html itemscope="" itemtype="http://  
schema.org/WebPage" lang="en"><head><meta  
content="Search the world's information, including  
webpages, images, videos and more. Google has many  
special features to help you find exactly what you're  
looking for." name="description"><meta content="noodp"  
name="robots"><meta content="text/html; charset=UTF-8"  
http-equiv="Content-Type"><meta content="/logos/doodles/  
2016/veterans-day-2016-6213878699524096-hp.png"  
itemprop="image"><meta content="Honoring our veterans  
#GoogleDoodle" property="og:description">...
```

**Exemplo do corpo
do *Response*.**



URI's

URI's

Sigla de Uniform Resource Identifier (Identificar Uniforme de Recurso).

scheme: nome do protocolo do objeto (**http**, **recursoremoto** (de rede) em **ftp**, **mailto**, etc.)

user: opcional, nome do usuário para acessar o recurso

Forma geral de um URL:

password: senha do usuário

`scheme://[user:password@]host[:port]/path[?query]
[#fragment]`

host: endereço do computador aonde o

query: um conjunto opcional de parâmetros que pode ser

dor para o recurso.

fragment: referência para um fragmento do conteúdo.

URL's

URL's

Sigla de **Uniforme Resource Locator** (Localizador Uniforme de Recurso).

Também referenciado como endereço web, é a **aplicação do padrão de URL's para endereços de Internet, em especial HTTP e HTTPS**

- <http://www.google.com>

Forma geral de um URL:

- [https://twitter.com/search?
q=Swift%20Language&src=typd](https://twitter.com/search?q=Swift%20Language&src=typd)

http[s]://[host][path][query][#fragment]

- [https://www.google.com.br/maps/search/Centro+Europeu/
@-25.4262029,-49.2803484,15z/data=!3m1!4b1](https://www.google.com.br/maps/search/Centro+Europeu/@-25.4262029,-49.2803484,15z/data=!3m1!4b1)

PADRÕES PARA TRANSMISSÃO DE DADOS

Aplicações conectadas modernas acabaram por construir diversos padrões para as transmissões de dados entre dispositivos. Vamos conhecer as principais.



FORMATO XML

Formato XML



Sigla de eXtensible Markup Language foi apresentada em meados de 1990 pela W3C como um padrão para representação de documentos.

Formato XML

Foi amplamente adotado pelo mercado graças as suas características:

Representada puramente por texto.

Simples de escrever e de ler, tanto para pessoas quanto para os computadores.

Simples de criar regras de validação através de DTD's ou Schemas.

Simples de integrar e transformar através de XSL.



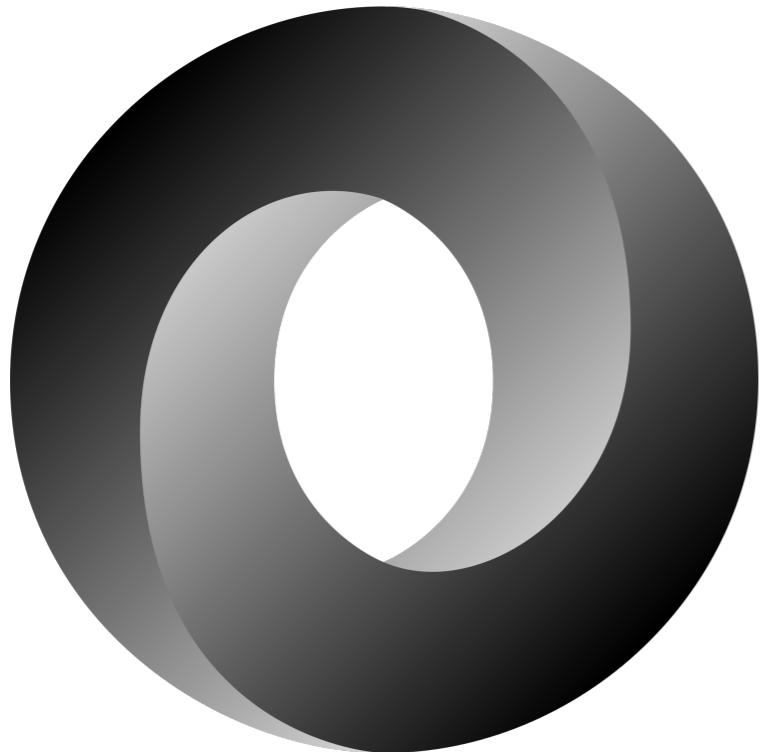
Formato XML - Exemplo

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<curso name="Curso de Criação de Apps">
    <descricao>Curso de Criação de Apps do Centro
    Europeu.</descricao>
    <disciplinas>
        <disciplina nome="Introdução a Programação"
cargaHoraria="24" />
        <disciplina nome="Criando Apps para iOS"
cargaHoraria="16" />
        <disciplina nome="Criando Apps para Android"
cargaHoraria="16" />
        <disciplina nome="Criando Apps Conectados"
cargaHorario="12" />
    </disciplinas>
    <turmas>
        <turma data="2016-08-01" periodo="integral">
            <aluno nome="Aluno 1" />
            <aluno nome="Aluno 2" />
        </turma>
        <turma data="2017-02-01" periodo="noturno">
            <aluno nome="Aluno A" />
            <aluno nome="Aluno B" />
        </turma>
    </turmas>
</curso>
```



FORMATO JSON

Formato JSON

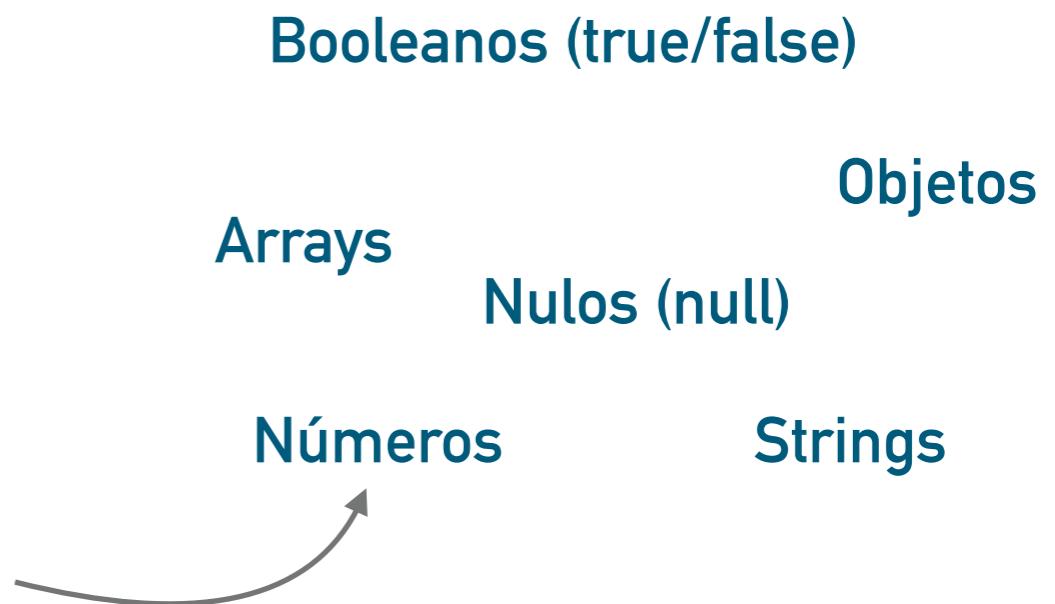


Sigla de JavaScript Object Notation, é um padrão derivado da linguagem JavaScript, que usa a notação que recebe esse acrônimo para descrever objetos da linguagem. O JSON exclui a parte funcional da linguagem, deixando apenas os elementos necessários para descrever o chamado Documento JSON.

Formato JSON

Documentos JSON descrevem dois tipos de elementos principais: Objetos e Arrays.

Objetos: uma lista de propriedades do objeto representados por uma par de chave e valor. As chaves são sempre Strings, enquanto os valores podem ser dos seguintes tipos:



Arrays: são listas de objetos JSON

“

Fazendo um paralelo com as estruturas de dados que conhecemos, um documento JSON pode ser facilmente visualizado como um Array (caso seu elemento raiz seja uma Array) ou um Dictionary (caso seu elemento raiz seja um Objeto).



Formato JSON

O JSON é um formato extremamente compacto, fácil de ler e de escrever, que nos permite expressar estruturas bastante complexas.

Ele tem servido como base para um novo conceito de bancos de dados chamados de NOSQL, onde as informações são armazenadas em documentos JSON ao invés de tabelas com linhas e colunas.



Formato JSON

Desde de meados de 2010 o JSON praticamente se tornou o padrão para representar as informações em API's Web.

Exemplo:

```
{  
    "name": "Curso de Criação de Apps",  
    "descricao": "Curso de Criação de Apps do Centro Europeu.",  
    "disciplinas": [  
        {  
            "nome": "Introdução a Programação",  
            "cargaHoraria": 24,  
        },  
        {  
            "nome": "Criado Apps para iOS",  
            "cargaHoraria": 16,  
        },  
    ],  
    "turmas": [  
        {  
            "data": "2016-08-01",  
            "periodo": "integral",  
            "alunos": [  
                "Aluno 1",  
                "Aluno 2",  
            ]  
        },  
        {  
            "data": "2016-08-01",  
            "periodo": "noturno",  
            "alunos": [  
                "Aluno 1",  
                "Aluno 2",  
            ]  
        },  
    ],  
}
```



CONTEÚDOS BINÁRIOS

Conteúdos Binários

Embora as transmissões de conteúdos binários tenha especial relevância na transmissão de conteúdos de mídia em especial, a complexidade de trabalhar com ela fez com que ela ficasse relegada a cenários específicos como a transmissão de mídia.

PADRÕES DE COMUNICAÇÃO NA WEB



SOAP

Sigla de Simple Object Access Protocol

Criado no final dos anos 90 como padrão para comunicação via HTTP.

As mensagens são trocadas usando o padrão XML.



SOAP

Protocolo básico dos Web Services, modelo que tomou força em meados de 2000.

A comunicação SOAP acontece através de mensagens nos padrões Request/Response tal como no HTTP



Exemplo de SOAP Request

```
POST /Quotation HTTP/1.0
Host: www.xyz.org
Content-Type: text/xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap-env:Envelope xmlns:soap-env="http://www.w3.org/
2001/12/soap-envelope" soap-env:encodingStyle="http://
www.w3.org/2001/12/soap-encoding">

    <soap-env:Body xmlns:m="http://www.xyz.org/
quotations" >

        <m:GetQuotation>
            <m:QuotationsName>Apple</m:QuotationsName>

            </m:GetQuotation>
        </soap-env:Body>

    </soap-env:Envelope>
```

Exemplo de SOAP Response

```
HTTP/1.0 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap-env:Envelope xmlns:SOAP-ENV="http://www.w3.org/
2001/12/soap-envelope" soap-env:encodingStyle="http://
www.w3.org/2001/12/soap-encoding" >
    <soap-env:Body xmlns:m="http://www.xyz.org/quotation"
>
        <m:GetQuotationResponse>
            <m:Quotation>Here is the quotation</
m:Quotation>
            </m:GetQuotationResponse>
        </soap-env:Body>
    </soap-env:Envelope>
```

REST

REST

Sigla de **Representation State Transfer**

Também costuma ser referenciado como **RESTful**

De acordo com o método HTTP que usamos com essa URL
Sua característica mais importante é o uso das
a API produz diferentes resultados:
características do HTTP para descrever os serviços.

GET: lista os itens dessa biblioteca.

POST: insere um novo registro na lista de itens da
Tomemos como exemplo um site de registro de livros.
biblioteca.

PUT: substitui a lista de itens por uma nova lista

DELETE: exclui a lista de itens



REST

Extendemos o uso dos URL's para referir a um item específico dentro da lista:

<http://bookstand.com/library/laranja-mecanica>

De acordo com o método HTTP que usamos podemos operar sobre esse item em específico de maneiras diferentes:

GET: obtém as informações detalhadas desse item.

PUT: substitui informações desse items.

DELETE: exclui esse registro.



COMUNICAÇÃO HTTP COM ANDROID



Comunicação HTTP com Android

O Android inclui suporte nativo para comunicação HTTP.

Esse suporte nos permite construir *Requests HTTP* e obter o Response usando a classe **HttpURLConnection**.

Esse método porém é verboso, cansativo e propenso a erro, portanto ao invés de estudá-lo vamos usar um biblioteca criada para facilitar esse tipo de operação, chamada **Volley**.

Operações em Plano de Fundo

Toda a operação que seu App for executar que utilize a rede deve fazê- lo em plano de fundo, ou seja, em uma *thread* à parte da principal.

Isso é importante para não travar o funcionamento do App.



Operações em Plano de Fundo

Essa é uma imposição do sistema operacional Android e qualquer tentativa de executar operações de rede na *thread* principal irá gerar um erro de execução (**NetworkOnMainThreadException**).

Felizmente a biblioteca **Volley** que utilizamos nesse curso cuida disso para nós!



PERMISSÕES

Permissões

É necessário solicitar permissão para acessar a rede e poder se comunicar com a Internet em Android. Isso é feito incluindo as diretivas de permissão no arquivo **AndroidManifest.xml**:

Onde:



```
<uses-permission  
    android:name="android.permission.ACCESS_NETWORK_STATE" />  
ACCESS_NETWORK_STATE: permite verificar o estado  
de conectividade do dispositivo.
```

```
<uses-permission  
    android:name="android.permission.INTERNET" />  
INTERNET: permite acessar a Internet.
```

A BIBLIOTECA VOLLEY

A Biblioteca Volley

Biblioteca criada pelo Google para facilitar a comunicação HTTP em Android.

Trabalha de maneira totalmente assíncrona, de maneira que o desenvolvedor não precisa se preocupar com a questão de threads.



A Biblioteca Volley

Tem um mecanismo integrado para agendamento e encadeamento de chamados.

Permite priorizar e cancelamento de requisições.

Deserializa automaticamente o resultado dos chamados.



Usando o Volley

Os passos para usar o Volley são os seguintes:

1

Criar ou Obter uma fila de requisições (RequestQueue).

2

Criar um objeto do tipo **Request<T>** que descreve:

- Configurações do HTTP, tais como URL, método, etc.
- Bloco de código a ser executado quando a requisição é concluída.
- Bloco de código a ser executado caso a requisição falhe.

3

Adicionar o objeto a fila de requisições.

“

*A classe **Request<T>** é abstrata, o que significa que ela não pode ser diretamente instanciada. Ao invés disso, trabalhamos com as classes derivadas dela.*



Usando o Volley

Os tipos de Request mais comuns do Volley são:

StringRequest: executa um Request HTTP obtendo o corpo (body) do Response como uma String.

JsonObjectRequest: executa o Request e converte o corpo do Response em um objeto do tipo **JSONObject** (classe nativa do Android para representar JSON).

JsonArrayRequest: semelhante ao anterior, porém o resultado é retornado é convertido para um objeto do tipo **JSONArray**.

“

É importante conhecer a API com a qual você está trabalhando para utilizar a versão especializada do Request mais adequada.



Usando o Volley

Ao inserir o seu *Request* na fila de processamento do *Volley*, ele irá executar a operação de rede em plano de fundo, chamado os métodos de sucesso ou falha tão logo o processamento da rede finalize.

Exemplo:

```
// Obtém uma nova fila de processamento
RequestQueue queue = Volley.newRequestQueue(this);

// Cria um Request para obter o conteúdo HTML da página
// inicial do Google
StringRequest request = new
StringRequest(Request.Method.GET, "http://
www.google.com",
    new Response.Listener<String>() {
        @Override
        public void onResponse(String response) {
            // Exibe o conteúdo no Logcat
            Log.i("VOLLEY_TEST", response);
        }
    },
    new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError
error) {
            // Exibe o erro no Logcat
            Log.e("VOLLEY_TEST",
error.getLocalizedMessage());
        }
    });
}

// Adiciona o Request a fila
queue.add(request);
```

COMUNICAÇÃO HTTP COM IOS



Comunicação HTTP com iOS

O iOS inclui um conjunto de bibliotecas padrões para trabalhar com comunicação HTTP, comumente chamados de **NSURLSession**.

Tal como no Android recomenda-se que toda a comunicação HTTP seja feita em segundo plano, embora o iOS não obrigue isso.

Desde o iOS 9 a Apple força que toda a comunicação seja feita usando HTTPS.



Comunicação HTTP com iOS

O recurso chamado ***App Transport Security*** (ATS) é habilitado por padrão e deve ser explicitamente desligado nas configurações do Aplicativo, no arquivo ***Info.plist***, adicionando a chave:

```
<key>NSAppTransportSecurity</key>
<dict>
<key>NSAllowsArbitraryLoads</key><true/>
</dict>
```



“

Importante: a partir de 2017 a Apple não aceitará mais submissões no Apple Store de Apps que façam comunicação com servidores considerados “não seguros” (ou seja, que não utilizam HTTPS).



“

Existem diversas bibliotecas para comunicação HTTP com iOS, como AFNetworking e Alamofire, sendo a última totalmente escrita em Swift. Se precisar de recursos mais avançados valem uma olhada!



ENTENDENDO O NSURLSESSION

Entendendo o NSURLConnection

Ao longo dos anos as bibliotecas para trabalhar com HTTP em iOS foram evoluindo junto com a plataforma.

Anteriormente eram verbosas, cansativas e propensas a erro, tal como no Android.



Entendendo o NSURLConnection

NSURLSession é uma classe da biblioteca básica do iOS, responsável pela criação de objetos **NSURLSession** e por suas configurações.

A classe principal e responsável por realizar as requisições (**requests**).

Tal como o anterior, porém armazena o conteúdo do Response dentro de um arquivo.

Obtém os dados de uma requisição e os armazena em memória para uso dentro do programa.

Permite enviar arquivos para um serviço Web (ex: enviar foto de perfil).

NSURLSession

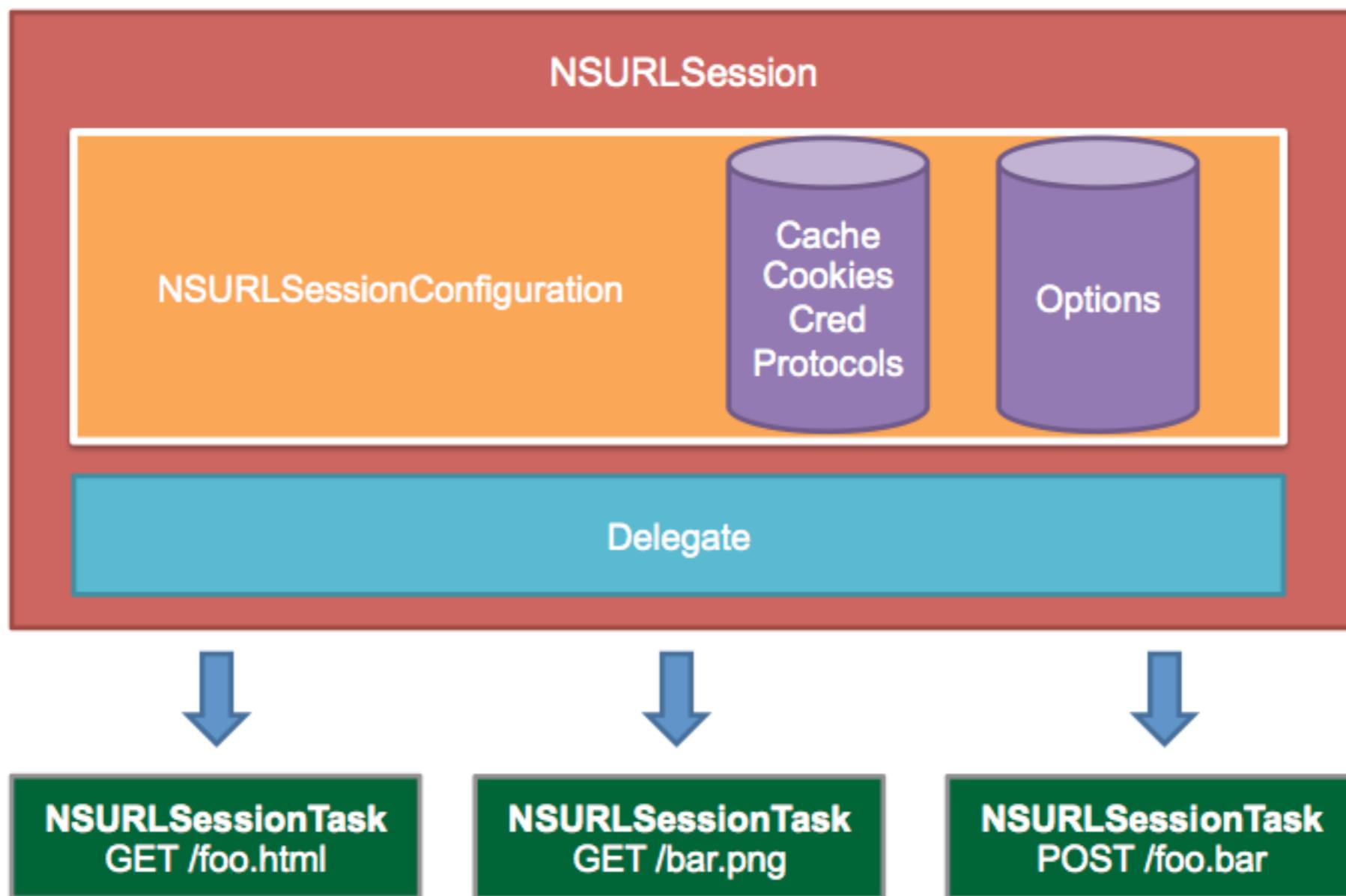
NSURLSessionConfiguration

NSURLSessionUploadTask

NSURLSessionDownload



Entendendo o NSURLConnection



Entendendo o **NSURLSession**

Os passos para usar o **NSURLSession** são os seguintes:

1.

Criar um objeto **NSURLSession** usando uma das 3 configurações padrões da classe **NSURLSessionConfiguration** (default, ephemeral ou background).

2.

Criar um objeto do tipo **NSURL** com a URL do serviço que deseja acessar.

3.

Criar uma Task usando os métodos `dataTaskWithURL`, `downloadTaskWithURL` ou `uploadTaskURL` do **NSURLSession**, passando a URL criada no passo anterior e configurando o código a ser executado quando o processamento finalizar.

4.

Chamar o método **resume** da Task criada para dar início a requisição.



Exemplo

```
// Cria um objeto de Sessão
let session = NSURLSession(configuration:
NSURLSessionConfiguration.defaultSessionConfiguration())
// Cria a URL do recurso que desejamos acessar
let url = NSURL(string: "http://www.google.com")!
// Cria uma task para ser executada pela Sessão
let task = session.dataTaskWithURL(url) {
    data, response, error in
    // Verifica se houve algum erro, do contrário
    imprime o conteúdo retornado.
    if let error = error {
        print(error.localizedDescription)
    } else if let httpResponse = response as?
    NSHTTPURLResponse {
        if httpResponse.statusCode == 200 {
            print(String(data))
        }
    }
}
// Inicia o Request
task.resume()
```