Protocolo HTTP em Java

Introdução

- O protocolo HTTP (*HyperText Transfer Protocol*) é, juntamente com o POP3 e SMTP, um dos protocolos do nível de aplicação mais usados na Internet
- Base da *World Wide Web* (WWW) desde a década de 1990
- Recorre ao TCP e ao porto 80 por omissão
- Uma sessão HTTP é uma sequência de transações do tipo pedido/resposta baseadas em mensagens de texto (ascii)
- Permite, entre outras operações, a obtenção de dados/recursos (ficheiros, resultados de consultas a bases de dados, conteúdos gerados de forma dinâmica, etc.)

Introdução

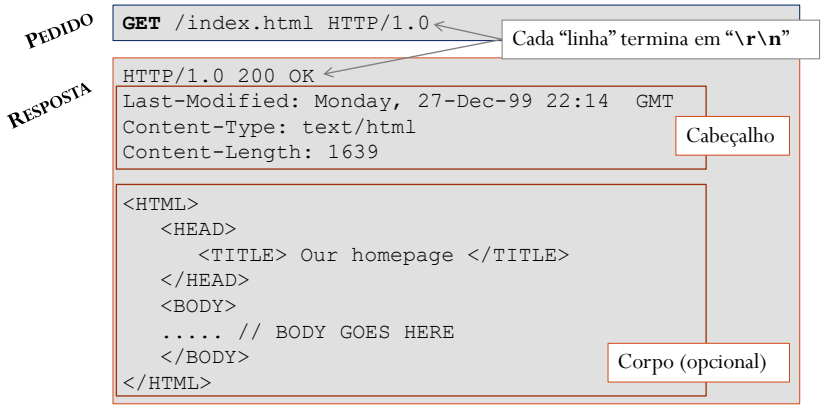
- Protocolo do tipo *stateless*
- Na versão 1.0 do HTTP (HTTP/1.0), que data de 1996, era estabelecida uma nova ligação TCP para cada pedido
- As versões 1.1, 2 e 3 surgiram, respetivamente, em 1997, 2015 e 2022
- A versão 1.1 (1997) introduziu as ligações TCP permanentes, reutilizadas por mais do que um pedido, o que permite reduzir a latência
- O HTTP/2 mantém compatibilidade com o HTTP/1.1, mas incorpora mecanismos para aumentar o desempenho (compressão automática, vários pedidos em simultâneo numa mesma ligação, etc.)

Introdução

- O HTTP/3 mantém a mesma semântica das versões anteriores e a capacidade de multiplexagem de vários pedidos como na versão 2, mas usa o QUIC em vez de TCP para efeitos de transporte:
  - Fornece latências menores no âmbito de conexões HTTP
  - Executa vários fluxos HTTP sobre UDP de forma independente
  - Inclui, para cada fluxo, mecanismos de deteção de perda de pacotes e de retransmissão

Introdução

- Exemplo de uma transacção entre um cliente e um servidor HTTP (obtenção do conteúdo do ficheiro "index.html")



URI

- *Uniform Resource Identifier*
- Permite identificar um recurso
- Formato:

  scheme:[//[user[:password]@]host[:port]][/path][?query][#fragment]
- Exemplos:
  - https://netp.isec.pt/netpa/page?stage=difhomestage
  - http://www.isec.pt:80
  - ftp://delta.isec.pt
  - mailto://jm3@isec.pt
  - file://c:/temp
  - rmi://services.isec.pt/ServicosAcademicos

Pedidos e Resposta

- Mensagens de texto (ascii)
- Estrutura genérica
  - Uma linha inicial (terminada em <CR><LF>)
  - Campos opcionais de cabeçalho (terminados em <CR><LF>)
  - Uma linha vazia (apenas com <CR><LF>)
  - Um corpo opcional de mensagem

Pedidos e Resposta

- **Pedidos**
  - Linha de pedido (terminada em <CR><LF>)

URI absoluta

```
GET http://w.b.com/widget/xpto/3/ HTTP/1.1
```

Caminho absoluto (mais comum)

```
GET /widget/xpto/3/ HTTP/1.1
Host: w.b.com
```

  - Campos opcionais do cabeçalho do pedido (terminados em <CR><LF>)
  - Linha vazia (<CR><LF>)
  - Corpo opcional do pedido

Pedidos e Resposta

- **Respostas**
  - Linha de status com código numérico e informação textual (terminada em <CR><LF>)
  - Campos opcionais do cabeçalho da resposta (terminados em <CR><LF>)
  - Linha vazia (<CR><LF>)
  - Corpo opcional da resposta

Campos de cabeçalho

- Informação (expansível) sobre pedidos/respostas/conteúdos
- Formato genérico: *nome_do_campo*: *valor*<CR><LF>

Pedido

```
GET /widget/xpto/3/ HTTP/1.1
Host: w.b.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:57.0) Gecko/20100101
            Firefox/57.0
Accept: text/html,application/xhtml+xml,application/xml,*/*
Accept-Language: en-US,en
```

Resposta

```
HTTP/1.1 200 OK
Content-Type: application/javascript; charset=utf-8
Date: Fri, 15 Dec 2017 14:10:09 GMT
Last-Modified: Tue, 25 Jul 2017 08:44:04 GMT
Server: Apache/2.4.7 (Ubuntu)
Cache-Control: no-cache
Content-Length: 115
```

Campos de cabeçalho

Pedido

```
POST /cgi-bin/process.cgi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Content-Type: text/xml; charset=utf-8
Content-Length: 97
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

<?xml version="1.0" encoding="utf-8"?>
<string xmlns="http://clearforest.com/">string</string>
```

https://www.tutorialspoint.com/http/http_requests.htm

Métodos/verbos HTTP

- GET
  - Solicita uma representação do recurso indicado na URI
  - Não deve ter qualquer efeito sobre os dados
- HEAD
  - Resposta semelhante ao GET, mas sem o corpo da mensagem
  - Apenas com a linha de status e o cabeçalho
- POST
  - Solicita o envio de dados ao servidor (atualização de dados, *upload* de um ficheiro, etc.), sendo estes incluídos no corpo da mensagem

Métodos/verbos HTTP

```
HEAD /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

https://www.tutorialspoint.com/http/http_methods.htm

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
Content-Length: 88
Content-Type: text/html
Connection: Closed
```


Métodos/verbos HTTP

Pedido

```
POST /cgi-bin/process.cgi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Content-Type: text/xml; charset=utf-8
Content-Length: 97
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

<?xml version="1.0" encoding="utf-8"?>
<string xmlns="http://clearforest.com/">string</string>
```

Métodos/verbos HTTP

Resposta

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
Content-Length: 75
Content-Type: text/html
Connection: Closed

<html>
<body>
<h1>Request Processed Successfully</h1>
</body>
</html>
```

https://www.tutorialspoint.com/http/http_methods.htm

Métodos/verbos HTTP

- PUT
  - Solicita que um determinado recurso, identificado através da URI fornecida, seja armazenado no servidor
  - Quando a URI se refere a um recurso já existente, este é substituído
- DELETE
  - Solicita a eliminação do referido recurso
- TRACE
  - Envia o pedido de volta ao cliente para que este possa verificar se algum servidor intermédio operou alguma modificação (usado durante a fase de desenvolvimento)

Métodos/verbos HTTP

```
PUT /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Connection: Keep-Alive
Content-type: text/html
Content-Length: 58

<html>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

https://www.tutorialspoint.com/http/http_methods.htm

```
HTTP/1.1 201 Created
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Content-type: text/html
Content-length: 66
Connection: Closed

<html>
<body>
<h1>The file was created.</h1>
</body>
</html>
```


Métodos/verbos HTTP

```
DELETE /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Connection: Keep-Alive
```

https://www.tutorialspoint.com/http/http_methods.htm

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Content-type: text/html
Content-length: 57
Connection: Closed

<html>
<body>
<h1>URL deleted.</h1>
</body>
</html>
```

Métodos/verbos HTTP

- OPTIONS
  - Devolve a lista dos métodos HTTP suportados pelo servidor
- CONNECT
  - Estabelece uma ligação/túnel ao servidor indicado na URI (usado pelo HTTPS na travessia de *proxies* sem suporte para SSL)
- PATCH
  - Aplica modificações parciais a um determinado recurso
  - Para substituir um recurso, deve ser usado o método PUT
- Com exceção do POST, todos os métodos devem ser *idempotents* (múltiplas invocações idênticas deixam o sistema no mesmo estado que uma única invocação)

Métodos/verbos HTTP

curl -i -X OPTIONS http://moodle.isec.pt/

```
OPTIONS HTTP/1.1
Host: moodle.isec.pt
User-Agent: curl/7.55.1
Accept: */*
```

```
HTTP/1.1 200 OK
Server: nginx/1.15.6
Date: Sun, 02 Dec 2018 19:00:34 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 0
Connection: keep-alive
Allow: GET,HEAD,POST,OPTIONS,TRACE
```

```
CONNECT www.tutorialspoint.com HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
```

https://www.tutorialspoint.com/http/http_methods.htm

```
HTTP/1.1 200 Connection established
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
```

Métodos/verbos HTTP

```
PATCH /widgets/abc123 HTTP/1.1
Host: api.example.com
Content-Length: ...
Content-Type: application/json-patch

[
  {"replace": "/count", "value": 5}
]
```

https://www.mnot.net/blog/2012/09/05/patch

```
HTTP/1.1 200 OK
Content-Type: text/plain
Connection: close

Your patch succeeded. Yay!
```


Códigos de resposta

- Nas respostas, a linha de status inclui um código numérico de status de resposta (extensível) e uma descrição textual
- **1XX** – Informação (pedido recebido e processo em curso)
- **2XX** – Pedido recebido e processado com sucesso
- **3XX** – Redireccionamento (ações adicionais são necessárias para concluir o pedido)
- **4XX** – Erro do lado cliente (pedido com erro de sintaxe ou sem possibilidade de ser satisfeito)
- **5XX** – Erro do lado servidor (pedido válido mas sem possibilidade de ser satisfeito)

Códigos de resposta

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
Content-Length: 58
Content-Type: text/html
Connection: Closed

<html>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

https://www.tutorialspoint.com/http/http_responses.htm

Códigos de resposta

```
HTTP/1.1 404 Not Found
Date: Sun, 18 Oct 2012 10:36:20 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 226
Connection: Closed
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html>
<head>
   <title>404 Not Found</title>
</head>
<body>
   <h1>Not Found</h1>
   <p>The requested URL /t.html was not found on this server.</p>
</body>
</html>
```

https://www.tutorialspoint.com/http/http_responses.htm

Códigos de resposta

```
HTTP/1.1 400 Bad Request
Date: Sun, 18 Oct 2012 10:36:20 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 329
Content-Type: text/html; charset=iso-8859-1
Connection: Closed

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html>
<head>
   <title>400 Bad Request</title>
</head>
<body>
   <h1>Bad Request</h1>
   <p>Your browser sent a request that this server could not understand.</p>
   <p>The request line contained invalid characters following the protocol
string.</p>
</body>
</html>
```

https://www.tutorialspoint.com/http/http_responses.htm

Códigos de resposta

100 Continue 101 Switching Protocols . . .	200 OK 201 Created 202 Accepted 203 Non-authoritative Information . . .	300 Multiple Choices 301 Moved Permanently 302 Found 303 See Other 304 Not Modified 305 Use Proxy . . .

400 Bad Request 401 Unauthorized 402 Payment Required 403 Forbidden 404 Not Found . . .	500 Internal Server Error 501 Not Implemented 502 Bad Gateway 503 Service Unavailable . . .

Alguns métodos de autenticação HTTP

- **Autenticação básica**
  - É o método mais simples
  - Um cliente fornece um nome e uma *password* em cada pedido
  - É usado o campo ***Authorization*** no cabeçalho do pedido, sendo o seu valor construído da seguinte forma:
    - Nome do utilizador e *password* concatenados: ***username: password***
    - Aplica-se a codificação Base64 ao resultado da concatenação
    - A palavra-chave ***Basic*** é colocada antes desse valor codificado
  - *Username* john e *password* secret ("john:secret"):

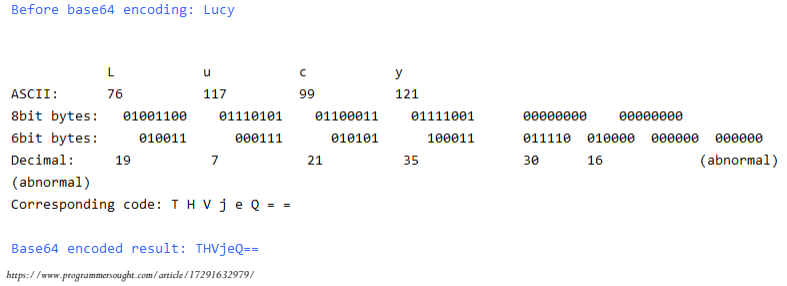
```
Authorization: Basic am9objpzZWNyZXQ=
```


Alguns métodos de autenticação HTTP

- Princípios da codificação Base64
  - 3 bytes consecutivos da mensagem original (24 bits) são agrupados em 4 palavras de 6 bits cada
  - 6 bits resultam em 64 combinações:

Value	Char	Value	Char	Value	Char	Value	Char
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

Alguns métodos de autenticação HTTP



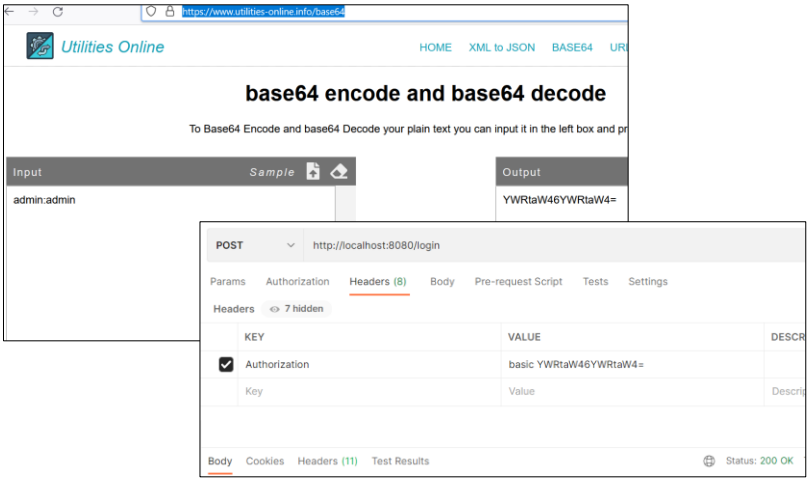
https://www.programmersought.com/article/17291632979/

```
import java.util.Base64;
...
byte[] bytes = "A minha mensagem".getBytes();
String encodedString = Base64.getEncoder().encodeToString(bytes);
bytes = Base64.getDecoder().decode(encodedString);
String decodedString = new String(bytes);
```


Alguns métodos de autenticação HTTP

- Principal desvantagem: as credenciais são incluídas em cada pedido
- Usar ligações seguras (HTTPS/SSL) pode não ser suficiente para garantir que as credenciais não sejam expostas
- Não existe qualquer mecanismo de *logout* associado nem de expiração de credenciais (a solução passa por solicitar ao utilizador que mude a sua *password*)

Alguns métodos de autenticação HTTP



Alguns métodos de autenticação HTTP

- ***Cookies***
  - Em resposta a um pedido HTTP, o servidor envia um *cookie* através do campo ***Set-Cookie*** no cabeçalho da resposta

```
Set-Cookie: critical-css-home=1576165734; expires=Wed, 15-Jan-2020
14:03:16 GMT; Max-Age=2592000; path=/; secure
```

  - O cliente (e.g., *browser*), guarda-o num ficheiro e transmite-o em cada pedido posterior no campo ***Cookie*** dos cabeçalhos dos pedidos
  - Para usar *cookies* para efeitos de autenticação, estes devem ser **assinados** de modo a permitir que o servidor possa detetar qualquer modificação introduzida do lado cliente

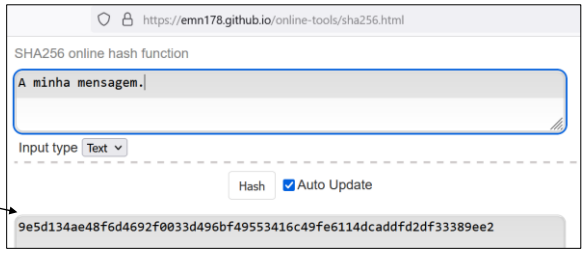
Alguns métodos de autenticação HTTP

- Conceitos de assinatura digital
  - Informação acrescentada a uma mensagem
  - Permite que o recetor verifique se o conteúdo da mensagem não foi alterado desde a sua emissão, bem como a identidade do emissor
  - Não tem como objetivo encriptar o conteúdo das mensagens!
  - Exemplos comuns: HS256 e RS256
  - **HS256** (assinatura HMAC com função de *hash* SHA-256)
    - HMAC - Hash-based Message Authentication Code
      - Criptografia simétrica
      - Chave secreta partilhada entre clientes e servidores

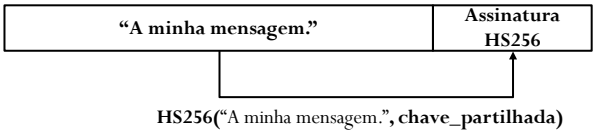
Alguns métodos de autenticação HTTP

- SHA - Secure Hash Algorithm
  - Produz uma sequência fixa de *bytes* com base num conteúdo com tamanho arbitrário
  - Permite, por exemplo, verificar se a informação foi alterada/adulterada desde a sua emissão
  - SHA-256: 256 *bits* (32 *bytes*)

Código hash SHA-256



Alguns métodos de autenticação HTTP

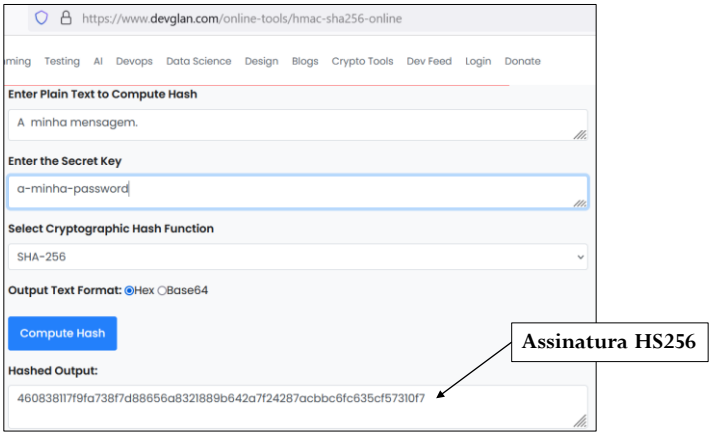


HS256("A minha mensagem.", chave_partilhada)

Criptografia simétrica + SHA-256 aplicados à mensagem → Código hash com 256 bits →

Criptografia simétrica + SHA-256 aplicados ao código hash obtido → Assinatura com 256 bits

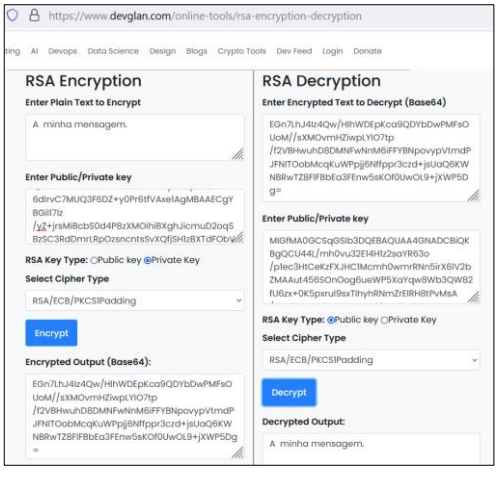
Alguns métodos de autenticação HTTP



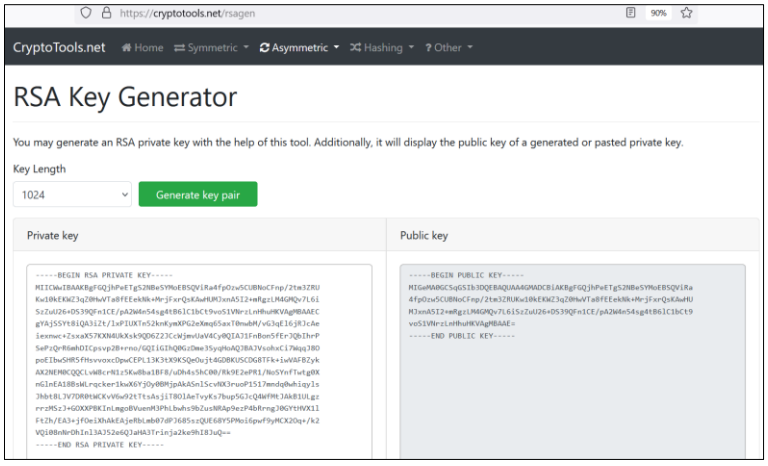
Alguns métodos de autenticação HTTP

- **RS256** (Assinatura RSA com função de *hash* SHA-256)
  - RSA - Rivest–Shamir–Adleman (nomes dos autores)
    - Algoritmo criptográfico assimétrico
    - Par de chaves: pública + privada
    - O que é encriptado com uma das chave pode apenas ser desencriptado com a outra
    - Mensagens encriptadas com a chave pública (“conhecida de todos”) apenas descodificáveis por quem tem a chave privada
      → **Confidencialidade**
    - Mensagens encriptadas com a chave privada descodificáveis com a chave pública (“conhecida de todos”)
      → **Autenticação**

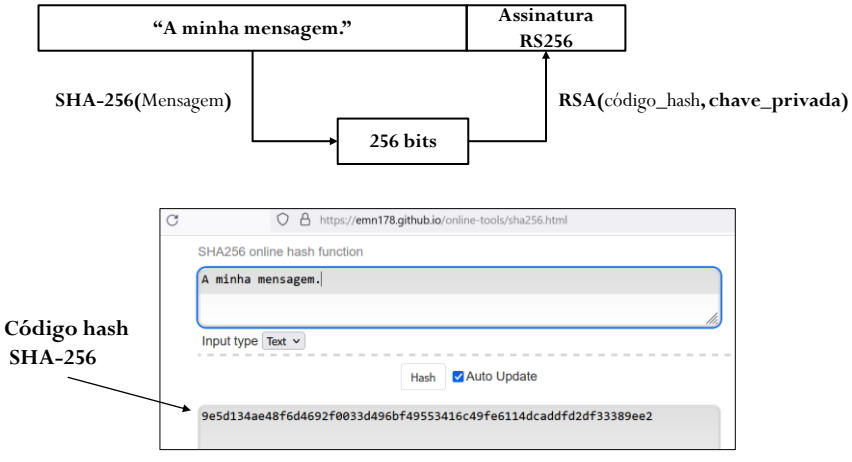
Alguns métodos de autenticação HTTP



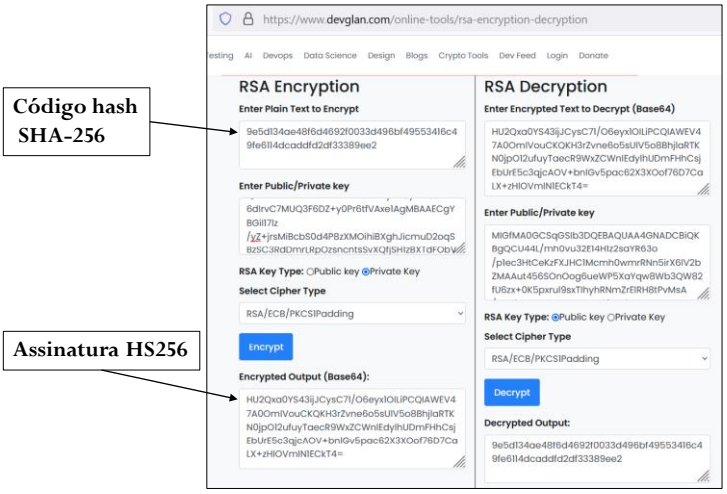
Alguns métodos de autenticação HTTP



Alguns métodos de autenticação HTTP



Alguns métodos de autenticação HTTP



Alguns métodos de autenticação HTTP

- **Tokens**
  - Um utilizador submete as suas credenciais (*username* e *password*) num pedido HTTP
  - Em resposta, se estas forem validadas pelo servidor, recebe um *token*, geralmente, no corpo da mensagem
  - Os *tokens* são passados em todos os pedidos subsequentes, no campo ***Authorization*** do cabeçalho, evitando recorrer às credenciais e a um processo repetido de autenticação
  - Podem ter um tempo de validade limitado e contêm toda a informação necessária (i.e., *username*), sendo apropriados para ambientes *stateless*

Alguns métodos de autenticação HTTP

- Uma aplicação pode, por exemplo, fornecer um *token* obtido a outra aplicação, dando-lhe, assim, acesso a recursos específicos, sem ter de fornecer as suas credenciais
- Exemplo de acesso baseado na utilização de um *token*:
  - Se não for fornecido qualquer *token* ou for fornecido um *token* inválido no acesso a um recurso protegido:

```
GET /hello-world HTTP/1.1
Host: localhost
User-Agent: curl/7.55.1
Accept: */*
```

```
HTTP/1.1 401
Content-Type: application/json;charset=UTF-8
Date: Tue, 4 Dec 2019 11:07:49 GMT

{"timestamp":"2019-12-
4T011:07:49.034+0000","status":401,"error":"Unauthorized",
"message":"Access Denied","path":"/hello-world"}
```


Alguns métodos de autenticação HTTP

- Quando é acedido, com credenciais válidas, o recurso destinado a retornar um *token* (*/user/login* neste exemplo):

```
POST /user/login HTTP/1.1
Host: localhost
User-Agent: curl/7.55.1
Accept: */*
Content-Type: application/json
Content-Length: 47

{"username":"Jose","password":"123"}
```

```
HTTP/1.1 200
Content-Type: application/json;charset=UTF-8
Date: Tue, 4 Dec 2019 11:33:57 GMT

{"token":"PD FH1leoFr0YkCy2ayuK68ITMalGrQQdGgrKZwn3SSKrA="}
```

Alguns métodos de autenticação HTTP

- O mesmo exemplo com autenticação básica HTTP:

```
POST /user/login HTTP/1.1
Host: localhost
User-Agent: curl/7.55.1
Authorization: Basic Sm9zZToxMjM=
Accept: */*
Content-Type: application/json
Content-Length: 47
```

```
HTTP/1.1 200
Content-Type: application/json;charset=UTF-8
Date: Tue, 4 Dec 2019 11:33:57 GMT

{"token":"PD FH1leoFr0YkCy2ayuK68ITMalGrQQdGgrKZwn3SSKrA="}
```


Alguns métodos de autenticação HTTP

- Se voltar a ser acedido, com o *token* obtido, o recurso protegido */hello-world*:

```
GET /hello-world HTTP/1.1
Host: localhost
User-Agent: curl/7.55.1
Accept: */*
Authorization: PD FH1leoFr0YkCy2ayuK68ITMalGrQQdGgrKZwn3SSKrA=
```

```
HTTP/1.1 200
Content-Type: text/plain;charset=UTF-8
Content-Length: 12
Date: Tue, 4 Dec 2019 11:41:50 GMT

Hello world!
```

Alguns métodos de autenticação HTTP

- **JWT - JSON Web Token**
  - Abordagem amplamente adoptada
  - *Token* constituído por três componentes separados por um ponto:
    - **Cabeçalho** com identificação do tipo de *token* e do algoritmo de *hash* usada para produzir a assinatura
    - **Conteúdo** (*payload*) com os dados (as reivindicações/*claims*)
    - **Assinatura**
      - Algoritmos comuns: HS256 e RS256
      - HS256(base64Encode(header) + "." + base64Encode(payload) , secret)
      - RS256(base64Encode(header) + "." + base64Encode(payload) , private_key)

Alguns métodos de autenticação HTTP

```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;

public class HMAC {
  static public byte[] calcHmacSha256(String chave, String msg){
    byte[] hmacSha256 = null;
    try {
      Mac mac = Mac.getInstance("HmacSHA256");
      SecretKeySpec sKeySpec = new SecretKeySpec(chave.getBytes(),
                                                 "HmacSHA256");
      mac.init(sKeySpec);
      hmacSha256 = mac.doFinal(msg.getBytes());
    } catch (Exception ex) {
      throw new RuntimeException("Calculo do hmac-sha256 falhou", ex);
    }
    return hmacSha256; //assinatura de msg
  }
}
```

Alguns métodos de autenticação HTTP

```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;

public class HMAC {
  static public byte[] calcHmacSha256(String chave, String msg){
    byte[] hmacSha256 = null;
    try {
//A assinatura também pode ser codificada em Base64
import java.util.Base64;

String hash = Base64.getEncoder().
                      encodeToString(HMAC.calcHmacSha256(
                              "chave123!", "A minha mensagem");
    }
    return hmacSha256; //assinatura de msg
  }
}
```


Alguns métodos de autenticação HTTP

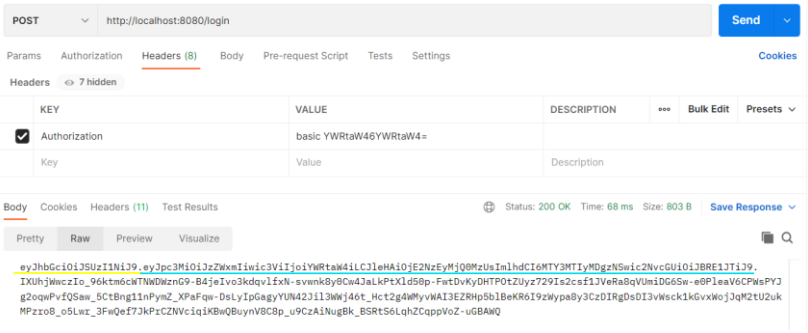
- Exemplo de um token JWT com:
  - **Cabeçalho**: `{"alg": "HS256", "typ": "JWT"}`
  - ***Payload***: `{"sub":"1234567890", "name": "John Doe", "admin": true}`
  - **Assinatura**: gerada através do algoritmo HS256 com a chave secreta partilhada `a-minha-password-a-minha-password`

```
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiYWRtaW4iOnRydWV9.7kj8xkuLZlYNNu0XhYg4nIp8FY7kGNtDW22t5yHA7Xw
```

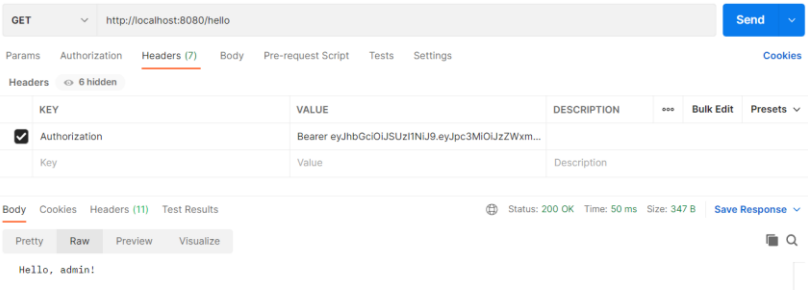
  - Se a assinatura também for codificada em Base64

```
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiYWRtaW4iOnRydWV9.N2tqOHhrdUxabFlOTnUwWGhZZzRuSXA4Rlk3a0dOdERXMjJ0NX
lIQTdYdw==
```

Alguns métodos de autenticação HTTP



Alguns métodos de autenticação HTTP



Alguns métodos de autenticação HTTP

- **Assinaturas nos pedidos HTTP**
  - Sem uma ligação segura (e.g., HTTPS), os *cookies* ou *tokens* podem ser expostos e usados por atacantes
  - Nas API web, pode assinar-se todos os pedidos
    - Sequência *hash*, que engloba a totalidade do pedido, encriptada com uma chave (algoritmo simétrico ou assimétrico)
  - Esta assinatura é acrescentada ao pedido sob a forma de uma *query* na URI ou no cabeçalho
  - Um atacante não consegue assinar os seus pedidos porque não conhece a chave, sendo estes descartados pelo serviço

java.net.URL

- Encapsula endereços no formato URL (*Uniform Resource Locator*):***protocolo://destino[:porto]/caminho[#referência]***
- Permite obter os recursos

Especifica uma parte ou uma posição no documento/recurso global

```
• URL(String url_str) throws java.net.MalformedURLException
• URL(String protocol, String host, String file) throws ...
• URL(String protocol, String host, int port, String file) throws ...
• ...
```

java.net.URL

```
public static void main(String args[])
{
    int argc = args.length;

    if (argc != 1){
        System.out.println ("Syntax : java URLParser url ");
        return;
    }

    try{
        URL myURL = new URL ( args[0] );

        System.out.println ("Protocol : " + myURL.getProtocol() );
        System.out.println ("Hostname : " + myURL.getHost() );
        System.out.println ("Port     : " + myURL.getPort() );
        System.out.println ("Filename : " + myURL.getFile() );
    }catch (MalformedURLException e){
        System.err.println ("Unable to parse URL!");
        return;
    }
}
```


java.net.URL

```
import java.net.*;
import java.io.*;

public class FetchURL
{
    public static void main(String args[]) throws Exception
    {
        int argc = args.length;

        if (argc != 1){
            System.out.println ("Syntax: java FetchURL url");
            return;
        }

        try{
            URL myURL = new URL ( args[0] );
            InputStream in = myURL.openStream();

            BufferedInputStream bufIn = new BufferedInputStream(in);
```

java.net.URL

```
            while((data = bufIn.read())!= -1){ //!EOF
                System.out.print ((char)data);
            }

            System.out.println ("\nHit <Enter> to continue");
            System.in.read();

        }catch (MalformedURLException e){
            System.err.println ("Unable to parse URL!");
            return;
        }catch (IOException ioe){
            System.err.println ("I/O Error : " + ioe);
            return;
        }
    }
}
```


java.net.URLConnection

- Envio e receção de pedidos HTTP
- Superclasse (abstrata) de todas as classes que representam uma ligação com uma URL
- Não possui construtores públicos, apenas do tipo *protected*

```
import java.net.*;
import java.io.*;

public class FetchURLConnection
{
    public static void main(String args[]) throws Exception
    {
        int argc = args.length;

        if (argc != 1){
            System.out.println ("Syntax: java FetchURLConnection url");
            return;
        }
```

java.net.URLConnection

```
        try{
            java.net.URL myURL = new URL ( args[0] );

            URLConnection connection = myURL.openConnection();
            connection.connect();

            // DISPLAY THE MIME CONTENT-TYPE (E.G. TEXT/HTML)
            String MIME = connection.getContentType();
            System.out.println ("Content-type: " + MIME);

            // DISPLAY, IF AVAILABLE, THE CONTENT LENGTH
            int contentLength = connection.getContentLength();
            if (contentLength != -1){
                System.out.println ("Content-length: " + contentLength);
            }

            // Pause for user
            System.out.println ("Hit enter to continue");
            System.in.read();
```


java.net.URLConnection

```
            // READ THE CONTENTS OF THE RESOURCE FROM THE CONNECTION
            InputStream in = connection.getInputStream();
            // BUFFER THE STREAM, FOR BETTER PERFORMANCE
            BufferedInputStream bufIn = new BufferedInputStream(in);

            while((data = bufIn.read())>0){
                System.out.print ( (char) data);
            }
        }catch (MalformedURLException e){
            System.err.println ("Unable to parse URL!");
            return;
        }catch (IOException e){
            System.err.println ("I/O Error : " + ioe);
            return;
        }
    }
}
```

java.net.URLConnection

- Existem muitos outros campos que podem ser inspeccionados no cabeçalhos
- É possível modificar campos nos cabeçalhos dos pedidos HTTP antes de serem submetidos

```
import java.net.*;
import java.io.*;

public class HTTPHeaders
{
   public static void main(String args[]) throws Exception
   {
      int argc = args.length;

      if (argc != 1){
         System.out.println ("Syntax: java HTTPHeaders url");
         return;
      }
```


java.net.URLConnection

```
      try{
         URL myURL = new URL ( args[0] );
         URLConnection connection = myURL.openConnection();

         // Set some basic request fields
         // Identify the application as Netscape compatible
         connection.setRequestProperty ("User-Agent", "Mozilla/4.0
                                 (compatible; JavaApp)");

         // Set use-caches field, to prevent caching
         connection.setUseCaches(false);

         // Now open a connection
         connection.connect();

         // Examine request properties, to verify their settings
         System.out.println ("Request properties....\n");
         System.out.println ("User-Agent: " +
                     connection.getRequestProperty("User-Agent"));
```

java.net.URLConnection

```
            System.out.println (); System.out.println ();

            // EXAMINE RESPONSE PROPERTIES, TO SEE THEIR SETTINGS
            System.out.println ("Response properties....\n");

            int i = 1;

            // SEARCH THROUGH EACH HEADER FIELD, UNTIL NO MORE EXIST
            while ( connection.getHeaderField ( i ) != null ){

                // GET THE NAME OF THIS HEADER FIELD
                String headerName = connection.getHeaderFieldKey(i);

                // GET THE VALUE OF THIS HEADER FIELD
                String headerValue = connection.getHeaderField(i);

                // OUTPUT HEADER FIELD KEY, AND HEADER FIELD VALUE
                System.out.println ( headerName + ": " + headerValue);

                i++;
            }
```


java.net.URLConnection

```
            System.out.println ("Hit enter to continue");
            System.in.read();

        }catch (MalformedURLException e){
            System.err.println ("Unable to parse URL!");
            return;
        }catch (IOException e){
            System.err.println ("I/O Error : " + e);
            return;
        }
    }
}
```

java.net.HttpURLConnection

- Subclasse de URLConnection
- Permite aceder aos códigos de estado das mensagens de resposta
- Inclui vários atributos estáticos que representam os possíveis códigos de resposta
- Cada instância é usada para efetuar um único pedido

```
public static final int HTTP_OK = 200;
public static final int HTTP_CREATED = 201;
public static final int HTTP_ACCEPTED = 202;
public static final int HTTP_NOT_AUTHORITATIVE = 203;
public static final int HTTP_NO_CONTENT = 204;
//...
```


java.net.HttpURLConnection

Método	Objectivo
void **disconnect**()	Desliga a ligação caso esteja activa
String **getRequestMethod**()	Devolve o método usado para os pedidos (e.g., "GET")
int **getResponseCode**()	Devolve o código de resposta
String **getResponseMessage**()	Devolve a mensagem de resposta (e.g., "OK")
void **setRequestMethod**(String method)	Define o método a usar para os pedidos (pode gerar uma java.net.**ProtocolException** e deve ser invocado antes do *connect()*)
static boolean **getFollowRedirects**()	Indica se indicações HTTP de redireccionamento devem ser automaticamente seguidas
static void **setFollowRedirects**(boolean flag)	Define se respostas HTTP de redireccionamento devem ser automaticamente seguidas (pode gerar uma java.net. **SecurityException** e deve se invocado antes do *connect()*)
...	...

java.net.HttpURLConnection

```
import java.net.*;
import java.io.*;

public class UsingHttpURLConnection
{
   public static void main(String args[]) throws Exception
   {
      int argc = args.length;

      if (argc != 1){
         System.out.println ("Syntax: java UsingHttpURLConnection url");
         return;
      }

      try{

         URL myURL = new URL ( args[0] );
         URLConnection connection = myURL.openConnection();
```


java.net.HttpURLConnection

```
         if (connection instanceof HttpURLConnection){

            HttpURLConnection hConnection;
            hConnection = (HttpURLConnection) connection;

            // DISABLE AUTOMATIC REDIRECTION, TO SEE THE STATUS HEADER
            hConnection.setFollowRedirects(false);

            // CONNECT TO SERVER
            hConnection.connect();

            // CHECK TO SEE IF A PROXY SERVER IS BEING USED
            if (hConnection.usingProxy()){
               System.out.println ("Proxy server used");
            }else{
               System.out.println ("No proxy server used");
            }
```

java.net.HttpURLConnection

```
            // GET THE STATUS MESSAGE
            String msg = hConnection.getResponseMessage();

            // GET THE STATUS CODE
            int   code = hConnection.getResponseCode();

            // IF A 'NORMAL' RESPONSE
            if ( code == HttpURLConnection.HTTP_OK ){
               System.out.println ("Normal response returned : " +
                  code + " " + msg );
            }else{
               System.out.println ("Abnormal response returned : "
                  + code + " " + msg );
            }

            System.out.println ("Hit enter to continue");
            System.in.read();
         }else{
            System.err.println ("Invalid transport protocol - not http!");
            return;
         }
```


java.net.HttpURLConnection

```
      }catch (MalformedURLException e){
         System.err.println ("Unable to parse URL!");
         return;
      } catch (IOException e){
         System.err.println ("I/O Error : " + e);
         return;
      }
   }
}
```

Quando uma URI identifica uma API / *Web Service* REST...

- Como exemplo, um *Web Service* REST que devolve citações sob a forma de documentos JSON (*JavaScript Object Notation*) com os seguintes pares atributo-valor

```
{
   type: "success",
   value: {
      id: 10,
      quote: "Spring Boot makes stand alone apps easy."
   }
}
```


Quando uma URI identifica uma API / *Web Service* REST...

http://gturnquist-quoters.cfapps.io/api/random (recurso alvo do verbo HTTP GET: uma citação aleatória)

```
GET /api/random HTTP/1.1
Host: gturnquist-quoters.cfapps.io
User-Agent: curl/7.55.1
Accept: */*
```

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Date: Sun, 02 Dec 2018 23:28:16 GMT
Content-Length: 197
Connection: keep-alive

{"type":"success","value":{"id":6,"quote":"It embraces
convention over configuration, providing an experience on
par with frameworks that excel at early-stage development,
such as Ruby on Rails."}}
```

Quando uma URI identifica uma API / *Web Service* REST...

http://gturnquist-quoters.cfapps.io/api/3 (recurso alvo do verbo HTTP GET: a citação com identificado 3)

```
GET /api/3 HTTP/1.1
Host: gturnquist-quoters.cfapps.io
User-Agent: curl/7.55.1
Accept: */*
```

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Date: Sun, 02 Dec 2018 23:40:03 GMT
Content-Length: 177
Connection: keep-alive

{"type":"success","value":{"id":3,"quote":"Spring has come
quite a ways in addressing developer enjoyment and ease of
use since the last time I built an application using
it."}}
```


Quando uma URI identifica uma API / *Web Service* REST...

http://gturnquist-quoters.cfapps.io/api (recurso alvo do verbo HTTP GET: todas as citações)

```
GET /api HTTP/1.1
Host: gturnquist-quoters.cfapps.io
User-Agent: curl/7.55.1
Accept: */*
```

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Date: Sun, 02 Dec 2018 23:42:30 GMT
Content-Length: 1918
Connection: keep-alive

[{"type":"success","value":{"id":1,"quote":"Working with
Spring Boot is like pair-programming with the Spring
developers."}},{"type":"success","value":{"id":2,"quote":"
With Boot you deploy everywhere you can find a JVM
basically."}},{"type":"success","value":{"id":3,"quote":"S
pring has come quite a ways in addressing developer
enjoyment and ease of use since the last time I built an
application using it."}},..]
```

Quando uma URI identifica uma API / *Web Service* REST...

- Exemplo de um programa que invoca a API REST anterior e constrói um objeto com base no documento JSON recebido

```
public class Value {

    private Long id;
    private String quote;

    public Value() {}
    public Long getId() {return this.id;}
    public String getQuote() {return this.quote;}
    public void setId(Long id) {this.id = id;}
    public void setQuote(String quote) {this.quote = quote;}
    @Override
    public String toString() {
        return "Value{id=" + id + ", quote='" + quote + '\'' + '}';
    }

}
```


Quando uma URI identifica uma API / *Web Service* REST...

```
public class Quote {

    private String type;
    private Value value;

    public Quote() {}
    public String getType() {return type;}
    public void setType(String type) {this.type = type;}
    public Value getValue() {return value;}
    public void setValue(Value value) {this.value = value;}
    @Override
    public String toString() {
        return "Quote{type='" + type + '\'' + ", value=" + value + '}';
    }

}
```

Quando uma URI identifica uma API / *Web Service* REST...

```
import java.io.*;
import java.net.*;
import com.google.gson.*;
import javax.json.*;

public class SingleQuoteConsumer {

   public static void main(String args[]) throws MalformedURLException,
                                                          IOException {
      String quoteId = args.length >0 ? args[0] : "random";
      getQuoteFromQuoteService(quoteId);
   }

   public static void getQuoteFromQuoteService(String quoteId) throws
                                  MalformedURLException, IOException {

      String uri = "http://gturnquist-quoters.cfapps.io/api/"+quoteId;
      URL url = new URL(uri);
```


Quando uma URI identifica uma API / *Web Service* REST...

```
      HttpURLConnection connection;
      connection = (HttpURLConnection) url.openConnection();
      connection.setRequestMethod("GET");
      connection.setRequestProperty("Accept", "application/xml, */*");

      InputStream in = connection.getInputStream();

      JsonReader jsonReader = Json.createReader(in);
      JsonObject object = jsonReader.readObject();

      jsonReader.close(); connection.disconnect();

      Gson gson = new GsonBuilder().create();
      Quote q = gson.fromJson(object.toString(), Quote.class);

      System.out.println("Tipo: " + q.getType());
      System.out.println("Id: " + q.getValue().getId());
      System.out.println("Citacao: " + q.getValue().getQuote());
   }
}
```

Quando uma URI identifica uma API / *Web Service* REST...

```
import java.io.*;
import java.net.*;
import com.google.gson.*;
import javax.json.*;

public class MultipleQuotesConsumer {

    public static void main(String args[]) throws MalformedURLException,
                                                   IOException {
        getQuoteFromRandomQuoteService();
    }

    public static void getQuoteFromRandomQuoteService() throws
                                MalformedURLException, IOException {
        String uri = "http://gturnquist-quoters.cfapps.io/api";

        URL url = new URL(uri);
        HttpURLConnection connection;
```


Quando uma URI identifica uma API / *Web Service* REST...

```
        connection = (HttpURLConnection)url.openConnection();
        connection.setRequestMethod("GET");
        connection.setRequestProperty("Accept", "application/xml, */*");

        InputStream jsonStream = connection.getInputStream();

        JsonReader jsonReader = Json.createReader(jsonStream);
        JsonArray array = jsonReader.readArray();

        jsonReader.close(); connection.disconnect();

        for(int i=0; i<array.size(); i++){
            JsonObject object = array.getJsonObject(i);
            Gson gson = new GsonBuilder().create();
            Quote q = gson.fromJson(object.toString(), Quote.class);

            System.out.println("Citacao: " + q.getValue().getQuote());
        }
    }
}
```

Bibliografia

- REILLY, David; REILLY, Michael - *Java Network Programming & Distributed Computing* - Addison-Wesley
- http://download.oracle.com/javase/tutorial/essential/