Inatel

PÓS-GRADUAÇÃO EM DESENVOLVIMENTO DE APLICAÇÕES PARA DISPOSITIVOS MÓVEIS E CLOUD COMPUTING

Inatel

DM107

Desenvolvimento de Web Services com segurança sob plataforma Java e PHP

Profa. Daniela E. C. de Almeida
E-mail: daniela.edvana@inatel.br
Prof. Elton Barbosa
E-mail: elton.barbosa@inatel.br

Agenda

- Apresentação da disciplina;
- Introdução a REST Web Services;
- REST ou RESTFul?;
- Protocolo HTTP.

Introdução a REST Web Services

- Surgiu no ano 2000.
- Formalização de um conjunto de melhores práticas denominadas *constraints.*
- As *constraints* tinham como objetivo determinar a forma na qual os padrões como o *Hypertext Transfer Protocol* (HTTP) e *Uniform Resource Identifier* (URI) deveriam ser modelados.

Introdução a REST Web Services

- **Cliente Servidor**

A principal característica dessa *constraint* é separar as responsabilidades de diferentes partes de um sistema.

Tipos de divisão:

- Armazenamento de dados e o *back-end* da aplicação;
- Interface do usuário e *back-end.*

Introdução a REST Web Services

- ***Stateless***

Cada requisição ao servidor não deve ter ligação com requisições anteriores ou futuras.

O protocolo HTTP segue esse modelo, porém, é muito comum o uso de *cookies* para armazenamento de sessões do lado do servidor.

Introdução a REST Web Services

- ***Cache***

Uma API RESTful deve permitir que suas respostas sejam passiveis de *cache.*

Introdução a REST Web Services

- **Interface Uniforme**

Uma das mais importantes *constraints.*

Os elementos abaixo devem ser considerados:

- Recursos;
- Mensagens auto descritivas;
- *Hypermedia.*

Introdução a REST Web Services

- **Sistemas em camadas**

Capacidade de adicionar elementos intermediários e que sejam totalmente transparentes para seus clientes.

- **Código sob demanda**

Adaptar o cliente de acordo com novos requisitos e funcionalidades.

REST ou RESTFul?

- Modelo e sobre as características (*constraints*): REST.
- Implementação que usa essas mesmas características: RESTFul.

Protocolo HTTP

- Composição de uma *Uniform Resource Locator* (URL) HTTP

<protocol>://<host>:<port>/<path>?<searchpart>

Protocolo HTTP

- Recursos

Forma única de identificar um objeto abstrato ou físico.

Detalhado especificamente pela RFC 3986.

Tem-se a modelagem de um recurso sob um substantivo. Exemplos:

/cliente/1

/produto/1

/cliente/1/notificacao

Protocolo HTTP

- **Representações**

Quando um recurso é solicitado por um cliente (ex: *browser*), o servidor executa uma série de atividades e retorna uma mensagem ao solicitante.

As representações podem ser modeladas em vários formatos, como XML, JSON, HTML e etc.

Esse formato deve suportar a utilização de *Hypermidia.*

Protocolo HTTP

- **Requisições e Respostas**

As mensagens HTTP podem ser de dois tipos:

- Requisição;
- Resposta.

Ambas seguem padrões estruturais.

Protocolo HTTP

- **Requisições e Respostas**

Composta por:

- Uma linha inicial;
- Zero ou mais linhas de cabeçalho;
- Uma linha em branco que indica o fim do cabeçalho;
- O corpo da mensagem, que é opcional a depender da situação.

Protocolo HTTP

- **Requisições e Respostas**

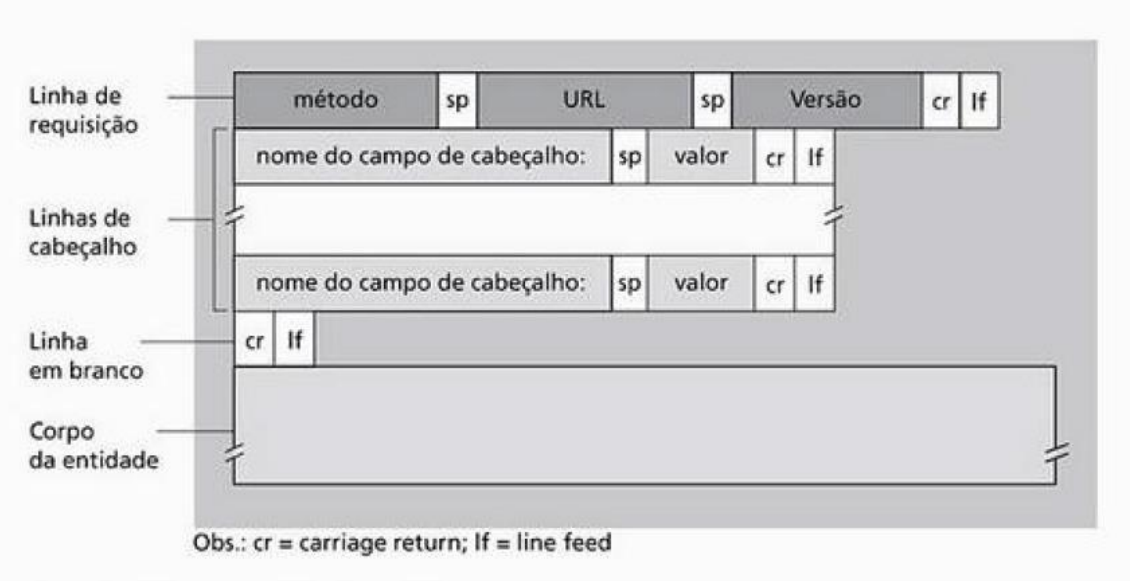

Figura 1 - Formato geral de uma mensagem de requisição HTTP

Fonte: KUROSE e ROSS, 2010.

Protocolo HTTP

- **Requisições e Respostas**

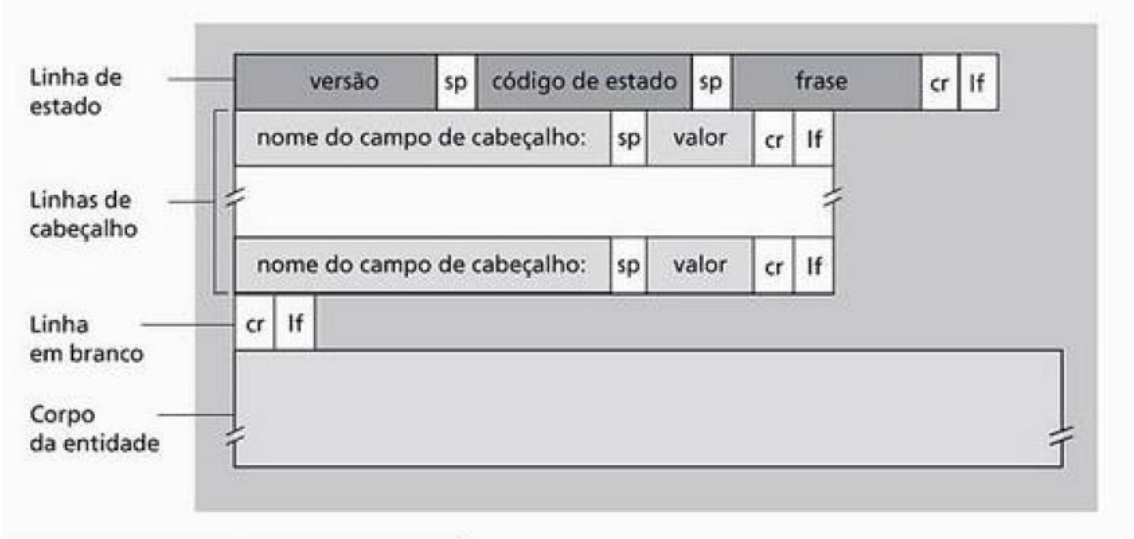

Figura 2 - Formato geral de uma mensagem de resposta HTTP

Fonte: KUROSE e ROSS, 2010.

Protocolo HTTP

- **Métodos**

A RFC 7231 especifica um conjunto de 8 métodos para a criação de uma API RESTful.

Dentre esses 8 métodos, iremos detalhar os 4 mais conhecidos.

Protocolo HTTP

- **Métodos**
  - **Get**:

    Utilizado para obter um recurso.

    Considerado idempotente.

Protocolo HTTP

- **Métodos**
  - **Post**:

    Utilizado para a criação de um recurso a partir do uso de uma representação.

Protocolo HTTP

- Métodos
  - Put:

    Utilizado para atualizar um determinado recurso.

    Em alguns cenários muitos específicos, também pode ser utilizado como forma de criação.

Protocolo HTTP

- **Métodos**
  - **Delete**:

Utilizado para a remoção de um determinado recurso.

Protocolo HTTP

- **Cabeçalhos**

  São utilizados para trafegar todo o tipo de meta informação a respeito das requisições.

  São padronizados.

  São facilmente extensíveis para comportar qualquer particularidade que uma aplicação possa requerer.

Protocolo HTTP

- **Cabeçalhos**

Cabeçalho	Tipo	Conteúdo
User-Agent	Solicitação	Informações sobre o navegador e sua plataforma
Accept	Solicitação	O tipo de páginas que o cliente pode manipular
Accept-Charset	Solicitação	Os conjuntos de caracteres aceitáveis para o cliente
Accept-Encoding	Solicitação	As codificações de páginas que o cliente pode manipular
Accept-Language	Solicitação	Os idiomas com os quais o cliente pode lidar
Host	Solicitação	O nome DNS do servidor
Authorization	Solicitação	Uma lista das credenciais do cliente
Cookie	Solicitação	Envia um cookie definido anteriormente de volta ao servidor
Date	Ambos	Data e hora em que a mensagem foi enviada
Upgrade	Ambos	O protocolo para o qual o transmissor deseja alternar
Server	Resposta	Informações sobre o servidor

Protocolo HTTP

- **Cabeçalhos**

Cabeçalho	Tipo	Conteúdo
Content-Encoding	Resposta	Como o conteúdo está codificado (por exemplo, gzip)
Content-Language	Resposta	O idioma usado na página
Content-Length	Resposta	O comprimento da página em bytes
Content-Type	Resposta	O tipo MIME da página
Last-Modified	Resposta	Data e hora da última modificação na página
Location	Resposta	Um comando para o cliente enviar sua solicitação a outro lugar
Accept-Ranges	Resposta	O servidor aceitará solicitações de intervalos de bytes
Set-Cookie	Resposta	O servidor deseja que o cliente grave um cookie

Protocolo HTTP

- **Media Types**

  Indica para o navegador qual é o tipo da informação que está sendo trafegada.

  São compostos com o seguinte formato: **tipo/subtipo.**

Protocolo HTTP

- **Media Types**

Os tipos mais comuns são:

- Application;
- Audio;
- Image;
- Text;
- Video;
- Vnd.

Protocolo HTTP

- **Media Types**

  Os *Media Types* são negociados a partir dos cabeçalhos *Accept (requisições)* e *Content-Type (respostas).*

Protocolo HTTP

- **Códigos de *status***

  Toda requisição enviada para o servidor retorna um código de status.

  São divididos em 5 famílias:

  - 1xx – Informacionais;
  - 2xx - Códigos de sucesso;
  - 3xx - Códigos de redirecionamento;
  - 4xx - Erros causados pelo cliente;
  - 5xx - Erros originados no servidor.

Protocolo HTTP

- **Códigos de *status***

De acordo com Leonard Richardson e Sam Ruby, os códigos mais importantes e mais utilizados são:

- **2xx**
  - ✓200 – OK;
  - ✓201 – *Created;*
  - ✓*202 – Accepted;*
  - ✓204 - *No Content;*
  - ✓206 - *Partial Content.*

Protocolo HTTP

- **Códigos de *status***
  - **3xx**
    - ✓301 -*Moved Permanently*;
    - ✓303 - *See Other;*
    - ✓304 - *NotModified;*
    - ✓307 - *Temporary Redirect.*

Protocolo HTTP

- **Códigos de *status***
  - **4xx**
    - ✓400 - *Bad Request*;
    - ✓401 - *Unauthorized;*
    - ✓403 - *Forbidden;*
    - ✓404 - *Not Found;*
    - ✓405 -*Method Not Allowed;*
    - ✓409 – *Conflict;*
    - ✓410 –*Gone;*
    - ✓412 - *Precondition failed;*
    - ✓415 - *Unsupported MediaType.*

Protocolo HTTP

- **Códigos de *status***
  - **5xx**
    - ✓500 - *Internal Server Error;*
    - ✓503 - *Service Unavailable*.

HTTP 1.1 x HTTP 2.0

HTTP 1.1	HTTP 2.0
Protocolo textual	Protocolo binário
Protocolo sequencial. É necessário o uso de mais de uma conexão para simular o paralelismo de requisições.	Protocolo assíncrono. Utiliza multiplexação para realizar requisições paralelas em uma única conexão.
Não prioriza requisições	Possui priorização de requisições
Apenas o cliente pode iniciar uma requisição.	Possui o mecanismo de *server push*, que permite ao servidor inferir requisições futuras e realizar o envio antecipadamente
Compressão de dados é opcional	Compressão de dados é padrão e obrigatória
Envia todos os dados de cabeçalho em cada mensagem	Utiliza compressão de cabeçalhos para enviar apenas os dados de cabeçalho que sofreram alteração ou são desconhecidos na conexão

Protocolo HTTPS

- O HTTP não é seguro.
- No HTTP os dados são enviados em texto puro.
- Dados sensíveis podem ser facilmente interceptados.

Protocolo HTTPS

- O HTTPS = HTTP + SSL/TLS (Secure Sockets Layer/Transporte Layer Security).
- No HTTPS os dados são enviados criptografados.
- Por consequência é mais seguro.
- Necessita de certificado digital do lado servidor.
- Certificado tem a chave publica, navegador criptografa os dados com a chave publica.

Protocolo HTTPS

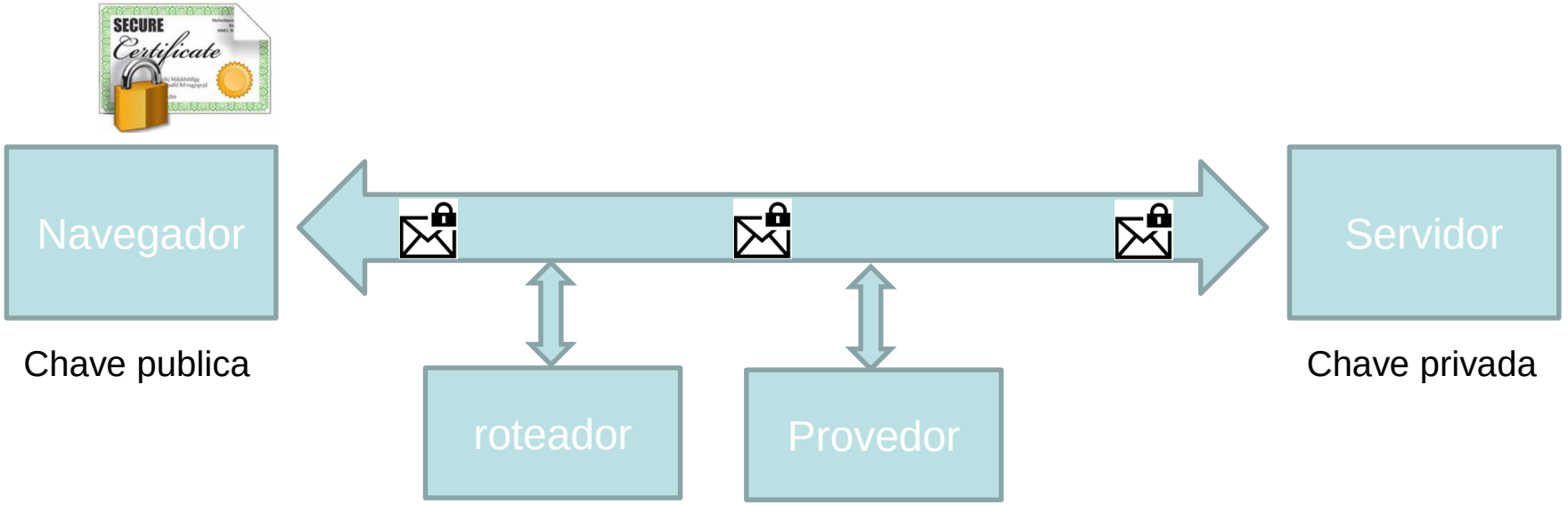

Figura 3 – Protocolo HTTPS

Protocolo HTTPS

- Quem garante que o certificado é confiável?
  Uma autoridade certificadora (CA), garante a identidade do servidor.

- Certificado tem validade?
  Sim.

- É mais lento que o HTTP convencional?
  Sim. Duas chaves diferentes é uma criptografia ***Assimétrica***

Protocolo HTTPS

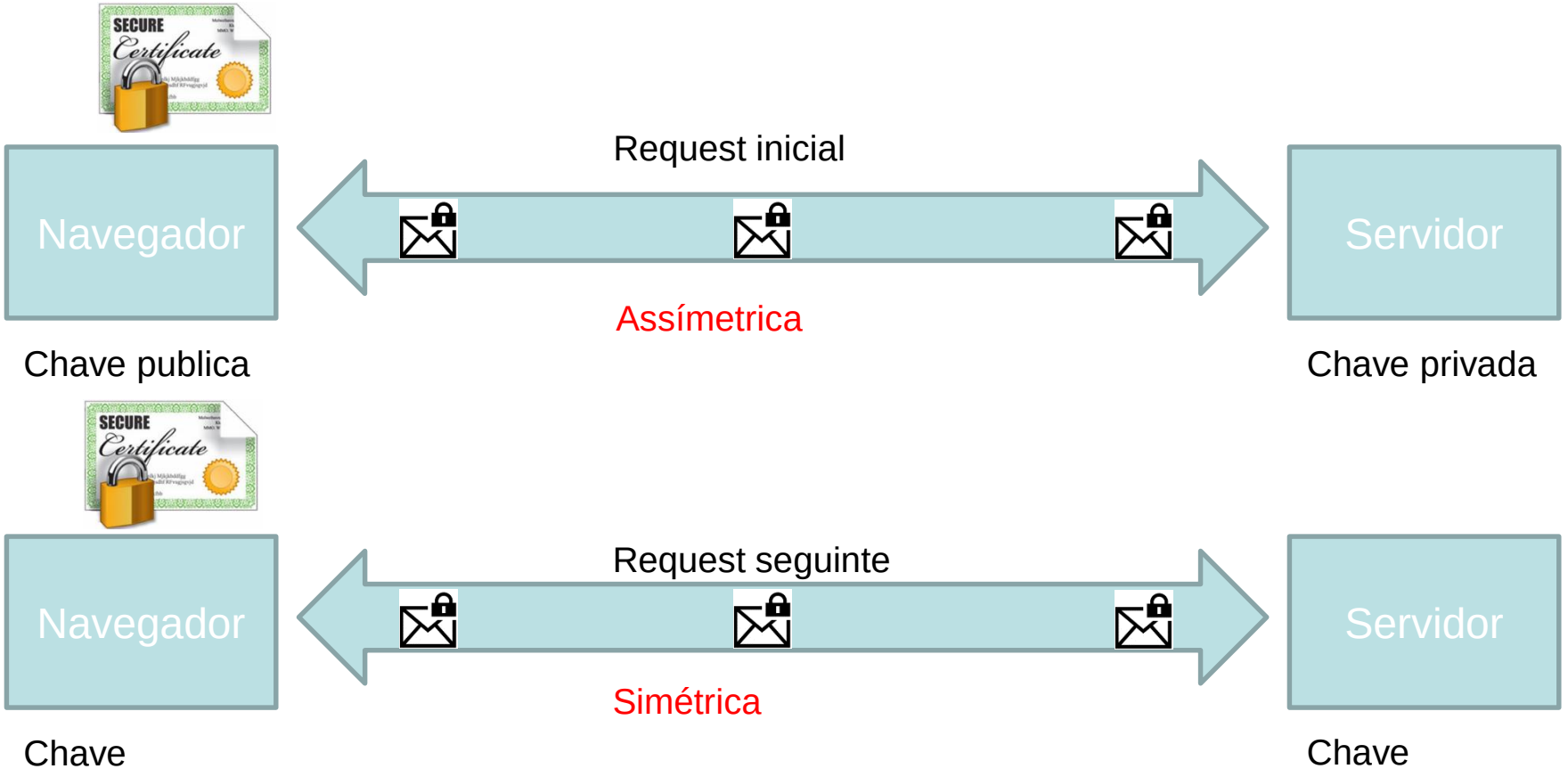

Figura 4 – Protocolo HTTPS – Chave simétrica e assimétrica

Protocolo HTTP

- **Prática 1 -** Estudo prático do protocolo HTTP através de ferramentas

- Ferramentas de depuração nos navegadores.
- Tipos de conteúdo requisitados e recebidos.
- Métodos e status code.
- Response/Headers.

Protocolo HTTP

- **Prática 1 -** Estudo prático do protocolo HTTP através de ferramentas

- Response/Body.

- Request/Headers.

- Conhecendo o Postman/CURL.

Referências

- Kurose, James F.; Ross, Keith W. **Redes de computadores e a internet:** Uma abordagem top-down. 5. ed. São Paulo: Addison Wesley, 2010. Tradução de: Computer networking a top-down approach featuing the Internet, 5th ed.

- Ruby, Sam.; Richardson, Leonard. **Restful web services**.O'Reilly, 2007.

- Saudate. Alexandre. **Rest Construa API's Inteligentes de Maneira Simples**. Casa do Código

- Souza, Jaime Freire. **Avaliação do Protocolo HTTP 2.0**. Disponível em < http://www2.uesb.br/computacao/wp-content/uploads/2014/09/MONOGRAFIA-JAIME-FINAL.pdf>. Acesso em 15 Ago. 2017.