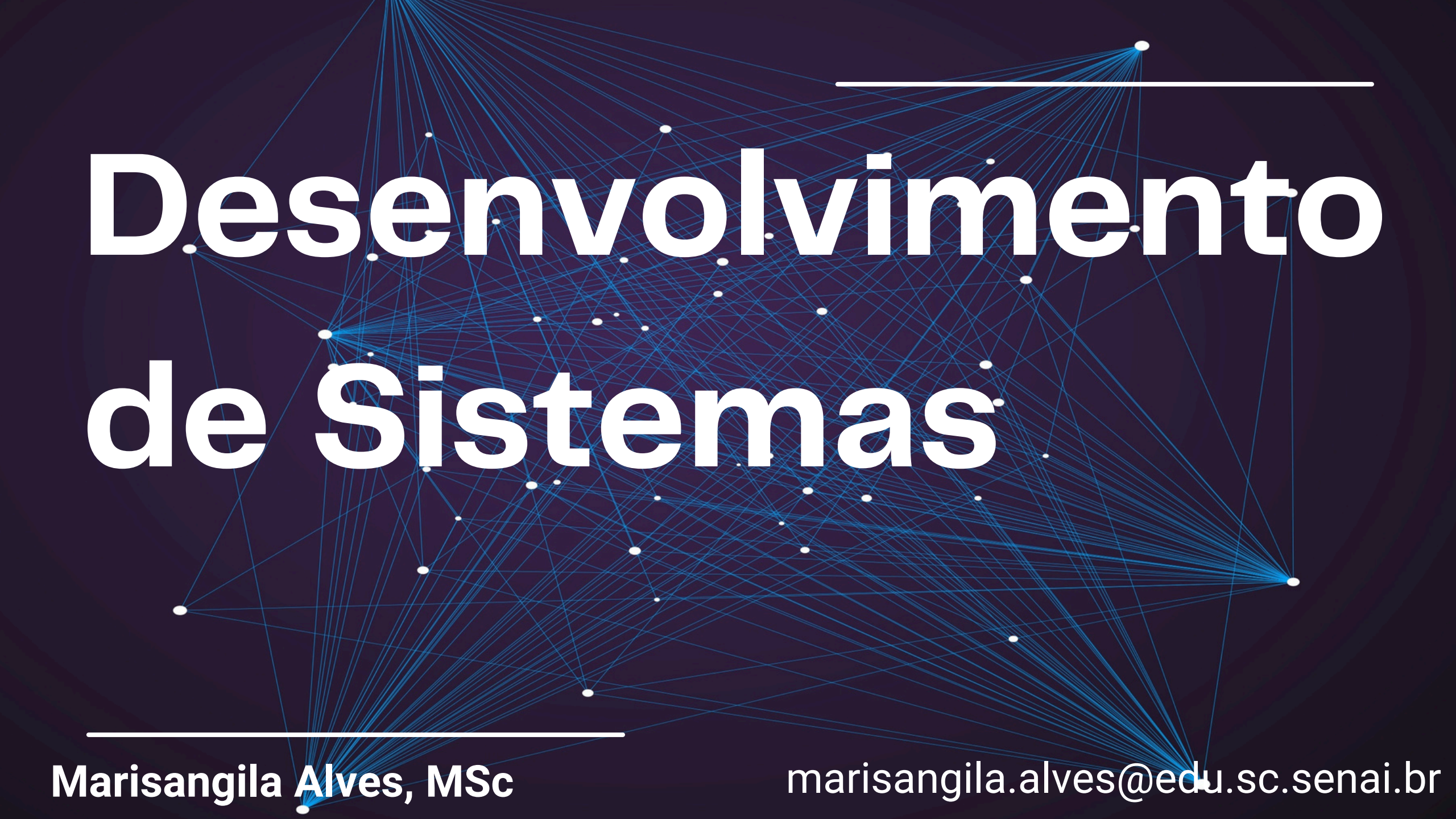


Desenvolvimento de Sistemas

Marisangila Alves, MSc

marisangila.alves@edu.sc.senai.br

REST

REST *(Representational State Transfer)*

REST é uma arquitetura de software que define um conjunto de princípios para a construção de Web Services.

REST *(Representational State Transfer)*

- Alternativa mais leve em relação ao SOAP.
- Recursos são identificados por uma URI *(Uniform Resource Identifie)* e acessados através de mensagens HTTP.

REST *(Representational State Transfer)*

- Exemplo de URI para um web service REST:

https://www.exemplo.com/servico/clientes/123

REST *(Representational State Transfer)*

Imagine que nosso *endpoint* seja:

http://exemplo.com/MyService/Add

Então teremos uma chamada:

http://exemplo.com/MyService/Add?a=10&b=5

REST *(Representational State Transfer)*

- Para realizar operações sobre um recurso(método), podem ser utilizados os verbos HTTP :
  - GET.
  - POST.
  - PUT.
  - DELETE.

REST *(Representational State Transfer)*

- Podemos enviar informações adicionais enviando um arquivo JSON.

JSON*(JavaScript Object Notation)*

```
{
    "nome": "João",
    "idade": 30,
    "profissao": "Engenheiro"
}
```

REST *(Representational State Transfer)*

- Podemos enviar informações adicionais enviando um arquivo JSON.
- Alternativa ao uso do XML.
- Mais compacto que XML.

RESTful

- Tese de Ph.D de Roy Fielding (cap 5)
- Formalização de um conjunto de boas práticas.
- Define padrões para que o HTTP e URI sejam modelados.

RESTful

- Cliente Servidor:
  - Divisão de responsabilidades.
  - Interface de usuário e *backend*
  - *Backend* e Banco de dados.
  - *Frameworks*: Angulas ou ReactJS.

RESTful

- *Stateless*:
  - Requisições devem ser independentes.
  - Os dados de autenticação devem estar no cliente.
  - Exemplo: JWT *(JSON Web Token):*
    - *Token -> cookie*

RESTful

- Cache.
- Interface Uniforme:
  - Utilização dos métodos e códigos de retorno e nomenclatura de recursos.
- Sistemas em Camadas:
  - Transparência para o cliente.
- Código sob Demanda.

REST ou RESTful?

- Para o modelo:
  - REST
- Para a implementação de uma aplicação:
  - RESTful

Representação de Recursos

- Um recurso é utilizado para identificar de forma única um objeto abstrato ou físico.
- Utiliza-se substantivos no plural.

Representação de Recursos

1. Listar todos os produtos: GET /produtos
2. Obter detalhes de um produto específico: GET /produtos/{id} Exemplo: GET /produtos/123
3. Criar um novo produto: POST /produtos
4. Atualizar um produto existente: PUT /produtos/{id} Exemplo: PUT /produtos/123
5. Excluir um produto: DELETE /produtos/{id} Exemplo: DELETE /produtos/123
6. Listar todos os clientes: GET /clientes
7. Obter detalhes de um cliente específico: GET /clientes/{id} Exemplo: GET /clientes/456
8. Criar um novo cliente: POST /clientes
9. Atualizar um cliente existente: PUT /clientes/{id} Exemplo: PUT /clientes/456
10. Excluir um cliente: DELETE /clientes/{id} Exemplo: DELETE /clientes/456

Representação de Recursos

Normalmente um recurso pode ser representado por um JSON (hypermedia)

Exemplo de recursos: /cliente/1

```
{ "nome": "Joao da Silva" }
```

Diversos tipos de reprentação podem ser aceitas como: XML, HTML ...

O campo Accept do cabeçalho HTTP pode definir qual formato pode ser aceito.

```
GET /api/produtos HTTP/1.1
Host: example.com
Accept: application/json
```

Representação de Recursos

O método GET é utilizado quando existe a necessidade de se obter um recurso. Ele é considerado idempotente, ou seja, independente da quantidade de vezes que é executado sob um recurso, o resultado sempre será o mesmo.

Exemplo:

```
GET /clientes/1 HTTP/1.1
```

Essa chamada irá retornar uma representação para o recurso “/clientes/1”

Representação de Recursos

POST:

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

Exemplo:

```
POST /clientes HTTP/1.1
{
  "Cliente": {
    "Nome": "João da Silva"
    ...
  }
}
```

Representação de Recursos

PUT:

O método PUT é utilizado como forma de atualizar um determinado recurso.

A semântica da atualização aqui é de todo o recurso. Exemplo:

```
PUT /clientes HTTP/1.1
{
  "Cliente": {
    "Nome": "João da Silva"
    ...
  }
}
```

Representação de Recursos

DELETE:

O delete tem como finalidade a remoção de um determinado recurso.

Exemplo:

DELETE /clientes/1 HTTP/1.1

Modelo de Maturidade Richardson

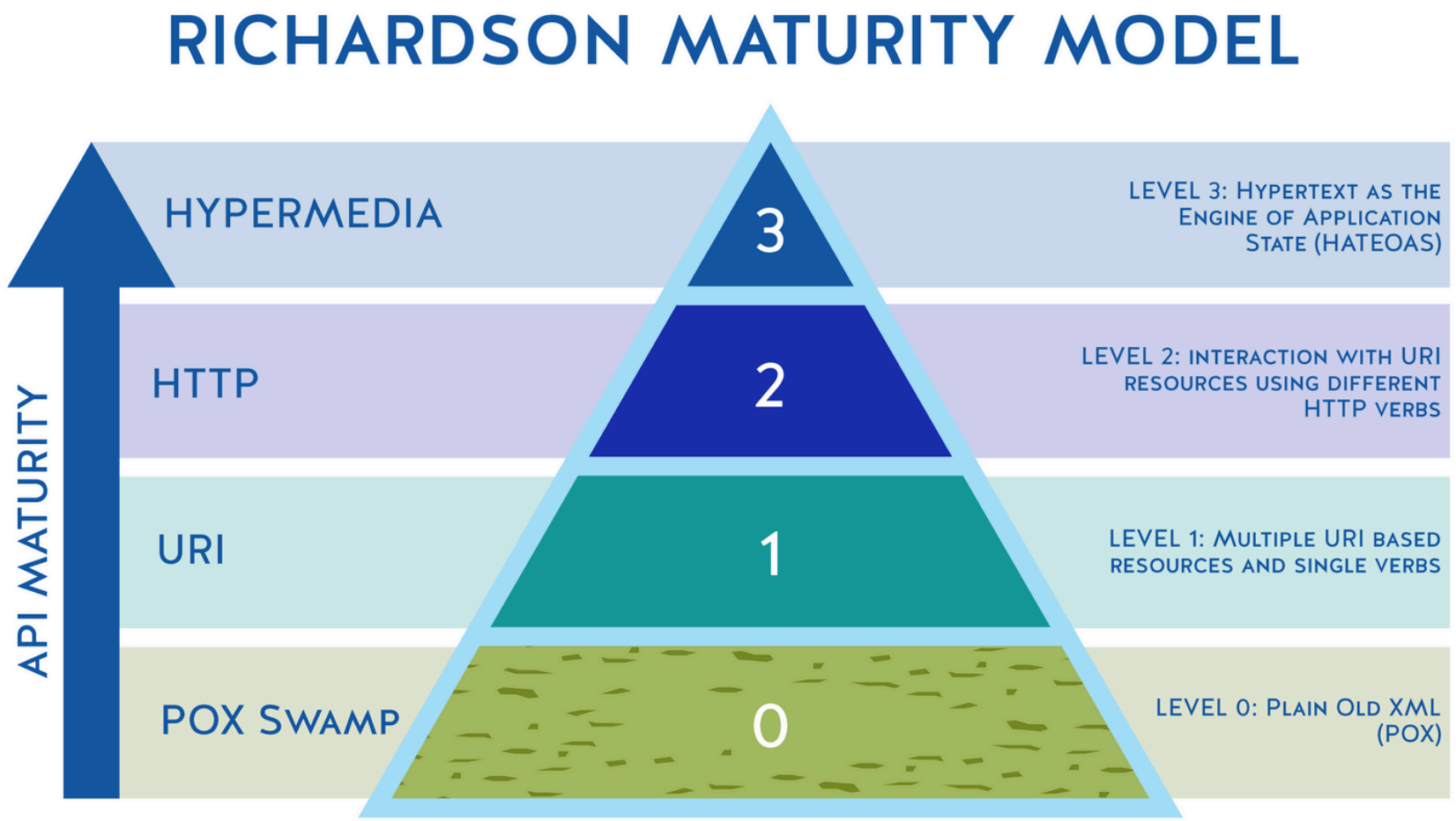
Leonard Richardson analisou centenas de projetos de web services e dividiu em 4 categorias.

- Fatores de análise:
  - URI.
  - Método HTTP.
  - Hypermedia (HATEOAS - hipermídia como o mecanismo do estado do aplicativo).

Modelo de Maturidade Richardson

Quanto mais uso dessas características, indo de encontro às restrições REST, mais REST é a API.

Modelo de Maturidade Richardson



Nível 0 (POX)

É importante lembrar, que não é o formato da mensagem que define ou não um sistema REST. Veja abaixo um exemplo de API em nível 0:

```
POST /salvarCliente HTTP/1.1
<Cliente>
<Nome>João da Silva</Nome>
...
</Cliente>
```

Apesar da utilização aparentemente correta do verbo POST HTTP na criação de um recurso, a URI não está modelada da forma correta.

Nível 0 (POX)

RPC (POX)		
Verbo HTTP	**URI**	**Ação**
GET	/buscarCliente/1	Visualizar
POST	/salvarCliente	Criar
POST	/alterarCliente/1	Alterar
GET/POST	/deletarCliente/1	Remover

REST		
Verbo HTTP	**URI**	**Ação**
GET	/cliente/1	Visualizar
POST	/cliente	Criar
PUT	/cliente/1	Alterar
DELETE	/cliente/1	Remover

Nível 1

Um primeiro passo em direção a REST é a modelagem adequada de recursos e utilização dos mesmos para interação com uma API. Um exemplo de chamada adequada nesse nível de maturidade pode ser:

```
POST /clientes HTTP/1.1
<Cliente>
<Nome>João da Silva</Nome>
...
</Cliente>
```

É importante notar a modelagem correta do recurso “cliente” (ou clientes). Modelando corretamente os recursos, precisamos usar os métodos HTTP da forma certa

Nível 2

Nesse nível, o HTTP deixa de exercer um papel apenas de transporte e passa a exercer um papel semântico na API, ou seja, seus verbos passam a ser utilizados com o propósito no qual foram criados.

Criando um cliente:

```
POST /cliente HTTP/1.1
<Cliente>
<Nome>João da Silva</Nome>
...
</Cliente>
```

Nível 2

O servidor deverá também agora retornar uma mensagem que reflete o uso adequado do protocolo HTTP, como segue abaixo:

HTTP/1.1 201 Created
Location: /cliente/1

É importante notar a utilização correta da resposta “201 Created”.
Nada mais adequado que uma resposta que informe que o recurso foi criado com sucesso. E a presença do header “Location” informa em qual endereço o recurso criado se encontra disponível.
Com o endereço fornecido (/cliente/1), nós podemos agora obter esse recurso.

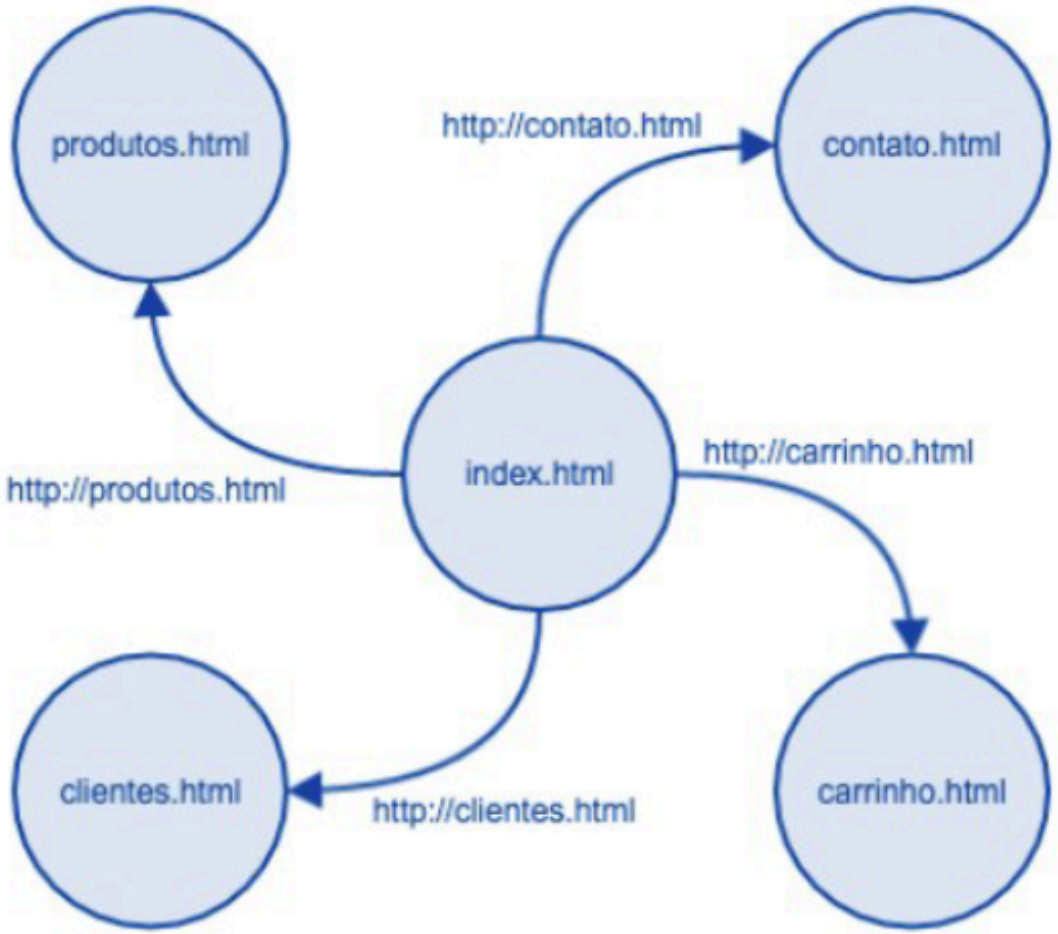
Nível 3 (HATEOAS)

Existe uma série de perguntas sobre HATEOAS (*Hypermedia as the Engine of Application State* – Hipermedia como “motor” do estado da Aplicação).

- Ele enfatiza o uso de links hipermídia para permitir a descoberta dinâmica e a navegação entre os recursos de um serviço web.

- Ao usar o HATEOAS, o cliente não precisa ter conhecimento prévio sobre a estrutura da API ou os pontos de extremidade disponíveis. Em vez disso, o cliente pode navegar pelo serviço web de forma dinâmica, seguindo os links hipermídia fornecidos nas respostas

Nível 3 (HATEOAS)



Nível 3 (HATEOAS)

GET /cliente/1 HTTP/1.1

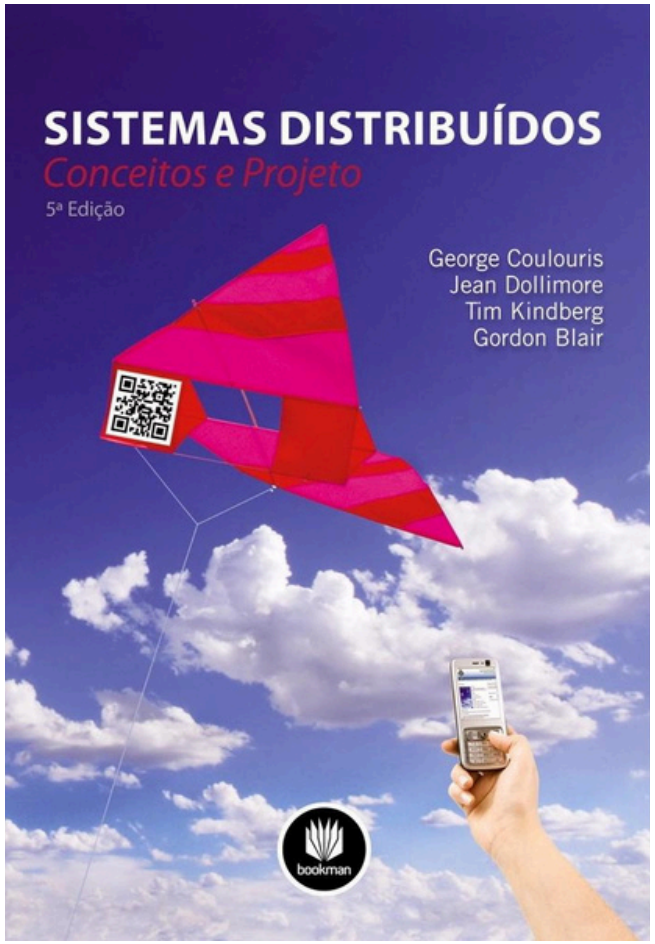
```
HTTP/1.1 200 OK
{
“id”: 1,
“nome”: “João da Silva”,
“links”: [
{ “rel”: “delete”, “link”: “/cliente/1”},
{ “rel”: “update”, “link”: “/cliente/1”}
]
}
```

Nível 3 (HATEOAS)

No exemplo, podemos ver a chamada ao recurso “/cliente/1”. O retorno
evidenciado demonstra a representação do estado e quais as possíveis ações a
serem realizadas.

Leitura Recomendada

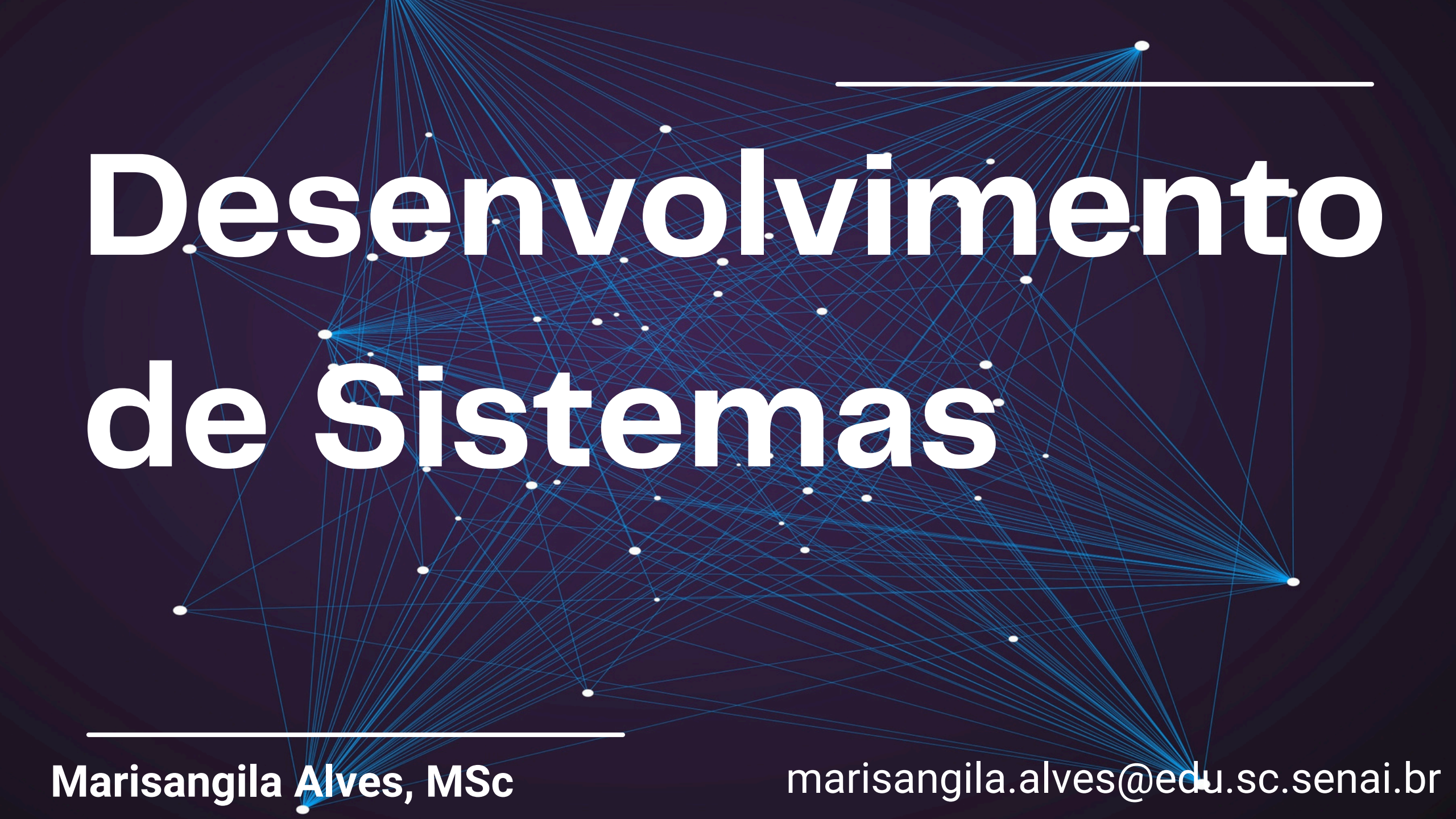
Capítulo 9 do livro:
Sistemas distribuídos Conceitos e Projeto



Referências

Gupta, Lokesh. Richardson Maturity Model – REST API Tutorial. Restfulapi.net, 5 Nov. 2023. Disponível em: https://restfulapi.net/richardson-maturity-model/. Acesso em: 29 out. 2025.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T.; BLAIR, G. Sistemas Distribuídos: Conceitos e Projeto, 5 a Edição. Bookman, 2013



Desenvolvimento de Sistemas

Marisangila Alves, MSc

marisangila.alves@edu.sc.senai.br