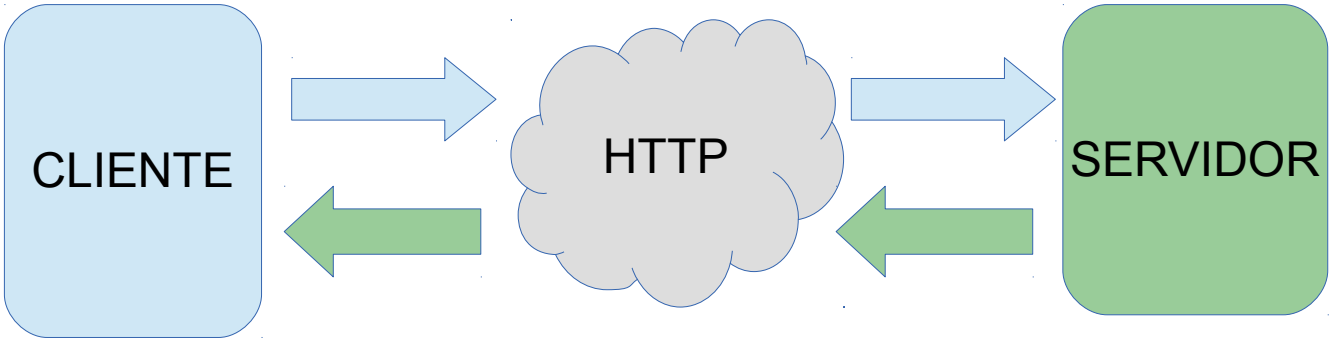


RESTful Webservices (parte 1 - Teoria)

Workshop TQI – JULHO/2014
Alexandre Fidélis Vieira Bitencourt
https://github.com/alexandrefvb/restful-webservices-workshop

Agenda

- O que é REST?
  - Introdução
  - REST x RPC
  - Os 6 princípios de arquiteturas REST
  - Idempotência e segurança
  - Verbos HTTP
  - Nomenclatura de resources e padrões de URIs
  - Richardson Maturity Model
  - Referências
  - Dúvidas?

Introdução

- REST = **RE**presentational **S**tate **T**ransfer
- É um estilo arquitetural descrito por Roy Fielding em sua dissertação de doutourado.
- "O nome 'REpresentational State Transfer' pretende evocar uma imagem de como um aplicativo da Web bem projetado se comporta: uma rede de páginas web (um estado da máquina virtual), onde o usuário progride através do aplicativo, selecionando os links (transições de estado), resultando na página seguinte (que representa o estado seguinte da aplicação) que está sendo transferido para o usuário e processado para a sua utilização."

REST x RPC

- Algumas APIs se dizem REST mas na verdade são exemplos de RPC!
- Utilizar HTTP como meio de transporte entre o cliente e o servidor não é o único requisito de aplicações REST.
- Existem várias implementações de RPC (SOAP, XML-RPC, Spring Remoting, etc) que utilizam HTTP como meio de transporte e não podem ser consideradas RESTful.
- REST posui o foco em resources (coisas) enquanto RPC possui o foco em operações (ações).
- Seis princípios que são inerentes ao estilo arquitetural REST.

Os 6 princípios REST

- Interface Uniforme
- Client-Server
- Stateless
- Cacheable
- Multicamadas
- Código sob demanda (opcional)

Interface Uniforme

- **Baseada em resources (objetos)**
  - Ex: Para um e-commerce possíveis resources são: Usuários, Pedidos, Produtos, etc.
- **São identificados por uma ou mais URIs**
  - Ex: /usuarios/1/pedidos
- **São manipulados através de representações (JSON, XML, HTML, YAML).**
  - A representação de um resource, incluíndo os metadados devem conter todas as informações necessárias para a manipulação do mesmo.
- **Os verbos HTTP (GET, POST, PUT, DELETE) da requisição representam a ação a ser tomada.**
- **Mensagens auto-descritivas**
  - As mensagens trocadas pelo cliente e servidor devem conter as informações sobre como devem ser processadas (Media types, cache, etc).

Interface Uniforme

Hypermedia as the Engine of ApplicatIon State (HATEOAS)

- O estado no cliente deve ser entregue através do corpo, parâmetros query string, headers e a URI da requisição.
- O estado no servidor deve ser entregue através do corpo, status e headers da resposta http.
- Além disso, links devem ser incluídos no corpo ou headers da resposta para prover ao cliente as URIs necessárias para obtenção do próprio resource e dos resources relacionados ao mesmo.

Client-Server

- Arquiteturas REST são exemplos de arquiteturas Client-Server.
- Separação de responsabilidades.
- A Interface Uniforme é a ligação entre o cliente e o servidor.
- Permite que vários clientes acessem os resources independentemente da tecnologia utilizada na implementação dos mesmos.

Stateless

- O o estado do cliente não pode ser armazenado no servidor!
- Cada requisição deve conter toda a informação necessária para realizar a operação desejada.
  - Mensagens auto descritivas
- Caso seja necessário manter o estado entre as requisições, este estado deve ser mantido no cliente.
  - Cookies, Headers, parâmetros ou mesmo o corpo das requisições e respostas podem conter o estado do cliente.

Cacheable

- As representações retornadas pelo servidor podem ser “cacheadas“ no cliente.
- Evita que requisições desnecessárias sejam enviadas ao servidor melhorando a escalabilidade da aplicação.
- As respostas podem de ser cacheadas forma implícita (convenção) ou explicita (através dos cabeçalhos de controle de cache enviados pelo servidor).
- O tempo de cache de uma representação de um resource depende da natureza do mesmo podendo inclusive não ser cacheável.

Multicamadas

- O cliente não pode assumir que possui uma conexão direta com o servidor.
- Podem haver várias camadas intermediárias entre o cliente e o servidor que processa a requisição.
- Os servidores intermediários permitem por exemplo o balanceamento de carga, caches e políticas de segurança melhorando a escalabilidade dos serviços.

Código sob demanda (opcional)

- É o único princípio opcional entre os 6 apresentados.
- Consiste em o servidor estender a funcionalidade apresentada pelo cliente de forma temporária através do envio de lógicas que o cliente pode executar.
- Ex: Java applets, JavaScript, Flash, SilverLight, etc.

Idempotência e segurança

- Uma operação é dita idempotente quando ao ser executada por vezes seguidas produz sempre o mesmo resultado.
- Uma operação é dita segura quando é utilizada apenas para obtenção de informações e ao ser executada não altera o estado do sistema a não ser informações de log ou cache que não refletem o estado do recurso sendo consultado.

Verbos HTTP

- Os mais utilizados são: POST, GET, PUT e DELETE.
- Representam respectivamente as operações básicas de criação, obtenção, atualização e remoção de resources.

GET

- Utilizado para obter uma representação de um resource.
  - Ex: GET http://www.api.com.br/clientes/1 Obtém a representação do cliente com o id 1.
- É uma operação considerada segura e idempotente, portanto não deve alterar o estado do resource sendo acessado.
- Em caso de sucesso retorna o status 200 (OK) com a representação do resource (json, xml, etc) no corpo da resposta.
- Em caso de erro pode retornar 404 (Not found) quando o recurso não existir ou 400 caso a requisição não esteja correta.

PUT

- Utilizado para atualizar um resource existente ou criar um novo resource caso a URI do mesmo seja previamente conhecida.
- O corpo da requisição deve conter uma representação do resource com o estado para o qual o mesmo deve ser atualizado.
- É uma operação idempotente e não segura pois altera o estado do resource e o resultado final será a atualização/criação do resource com aquele estado.
- Retorna 200, 201 ou 204 em caso de sucesso dependendo do que é retornado no corpo da resposta. Em caso de erro códigos comuns de retorno são 400 e 404.
- Para operações não idempotentes ou para criar um resource cuja URI será gerada pelo servidor recomenda-se a utilização de POST.

POST

- Utilizado para criar um novo resource.
- Na requisição deve ser enviada a representação do resource a ser criado.
- É uma operação não idempotente e não segura.
- A URI do resource criado é gerada pelo servidor e deve ser incluída na resposta.
- O status da resposta em caso de sucesso deve ser 201 (Created). Em caso de erro os status mais comuns são 400 (Bad request) e 409 (Conflict).

DELETE

- Remove o resource identificado pela URI.
- É uma operação não segura e idempotente. (Após a execução da operação várias vezes o resultado final é que recurso não mais existe).
- Os status de retorno recomendados são 200 (ok) e 204 (No content) em caso de sucesso e 404 caso o resource não exista.

Nomenclatura de resources e padrões de URIs

- Essencialmente uma API RESTful é uma coleção de URIs para as quais são enviadas requisições HTTP para obtenção e manipulação de representações recursos associados à URI.
- Ao decidir quais são os resources do sistema devem ser levados em consideração os substantivos e não os verbos. Em outras palavras as URIs devem identificar coisas e não ações.
- Exemplos de resources de uma rede social:
  - Usuários da rede social
  - Posts dos usuários
  - Usuários seguidores de um usuário

Nomenclatura de resources e padrões de URIs

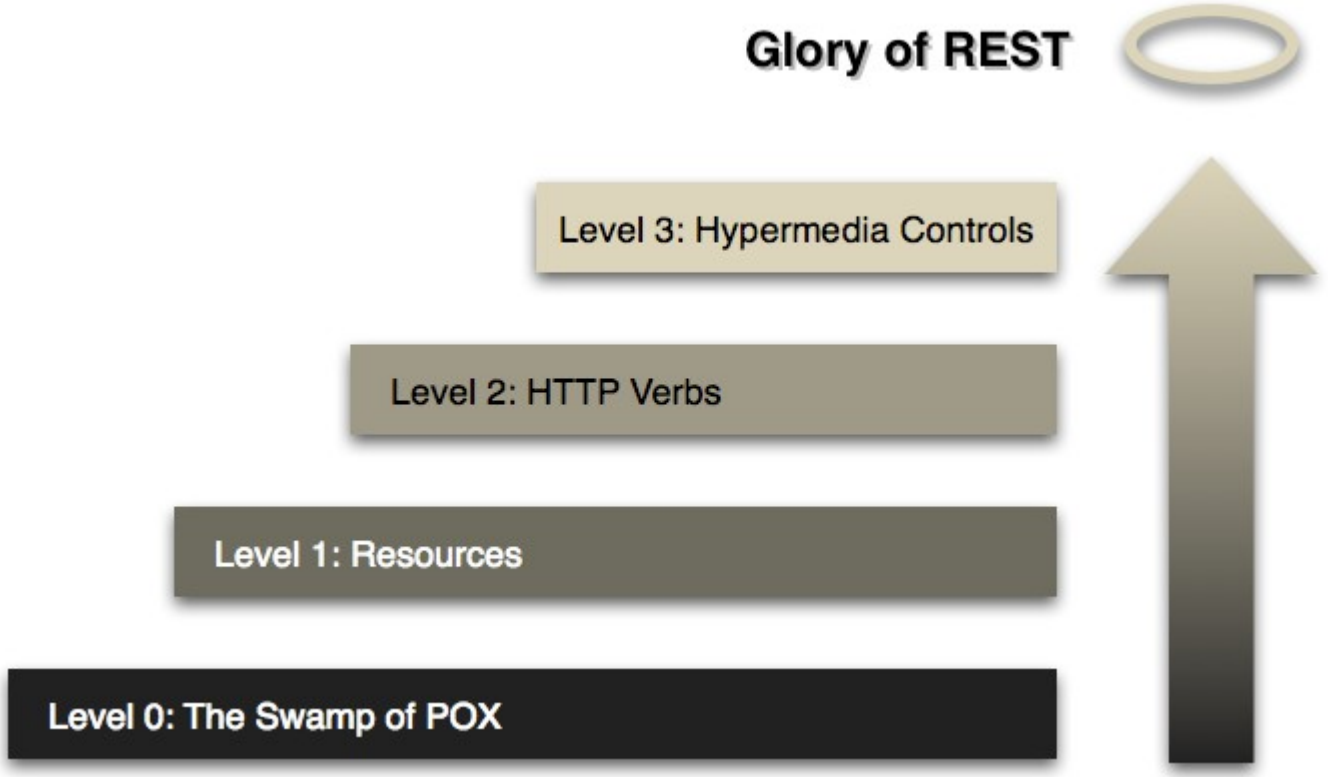
- Um resource deve ter pelo menos uma URI que o identifica.
- Um resource pode ser identificado por mais de uma URI.
- As URIs devem ser criadas de forma previsível e hierárquica de modo que as relações entre os resources sejam estabelecidas de forma clara.
- As URIs devem ser definidas pensando no cliente e não nos dados armazenados pelo sistema.

Nomenclatura de resources e padrões de URIs

- Pluralização x Singularização
  - /clientes/{id}
  - /cliente/{id}
- Hierarquia das URIs
  - /clientes
  - /clientes/{id}
  - /clientes/{id}/pedidos
  - /clientes/{id}/pedidos/{id}/items/{id}

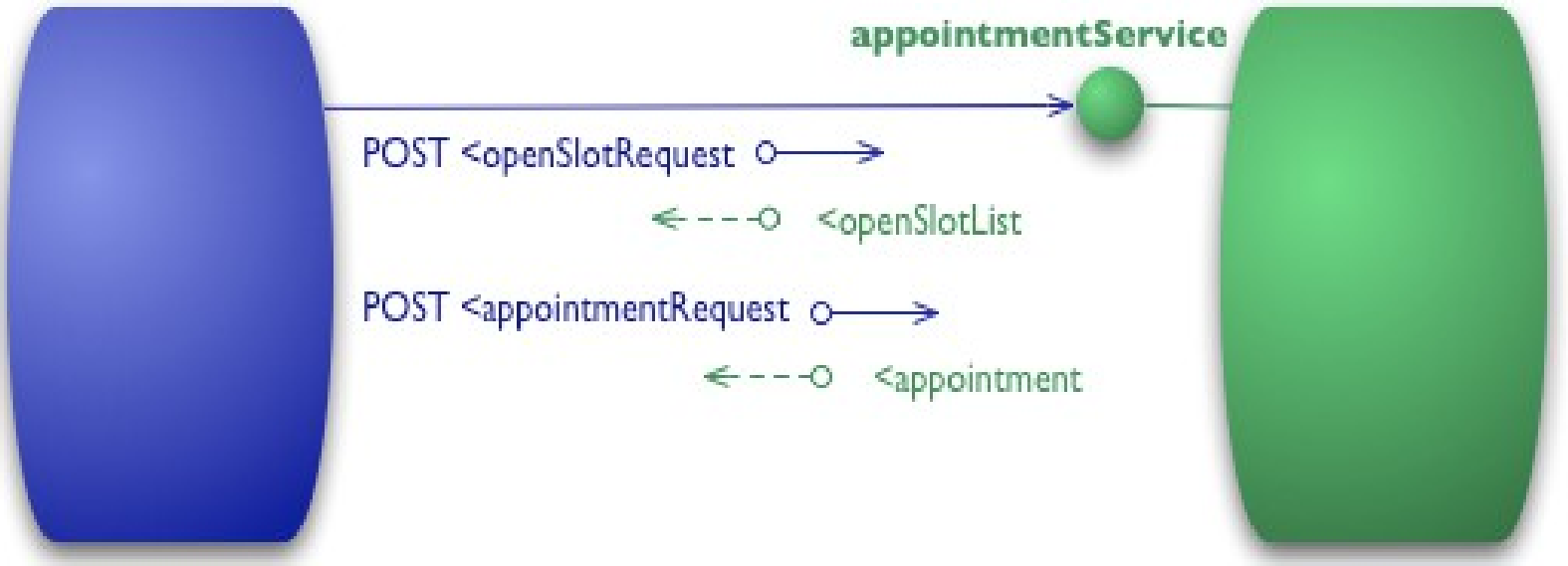
Richadson Maturity Model

- Desenvolvido por Leonard Richadson, define o nível de maturidade de uma API REST.



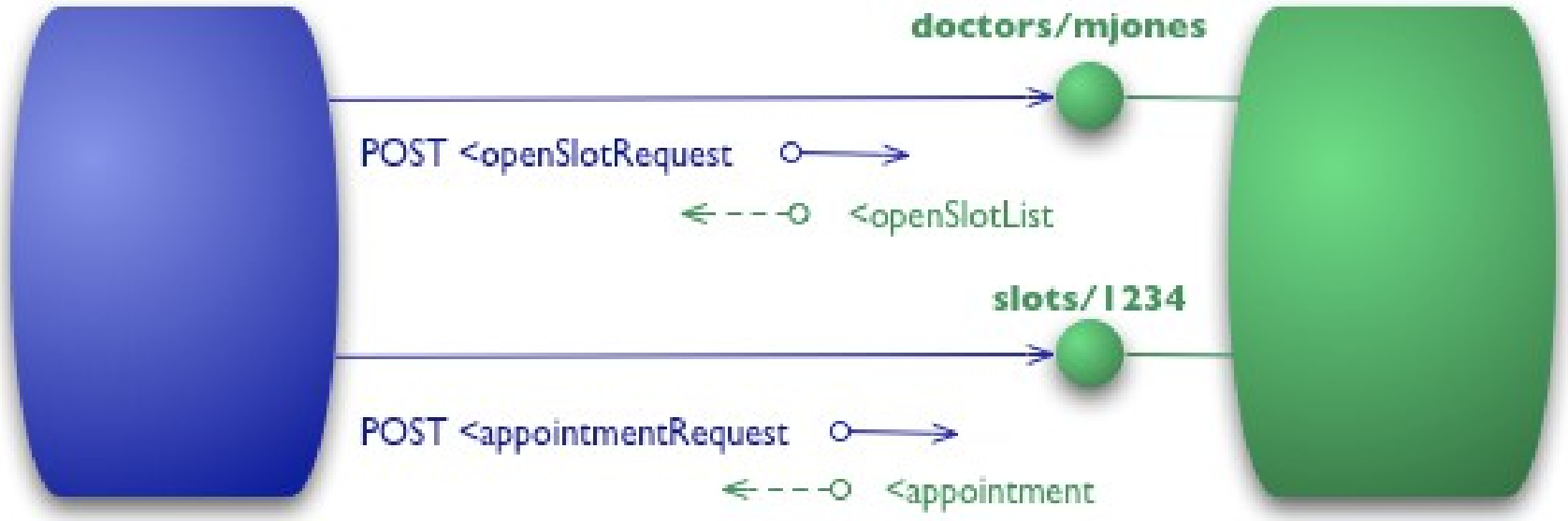
Level 0

- Utiliza o protocolo HTTP como transporte, mas possui um único ponto de entrada onde as requisições são enviadas. Pode ser considerado um exemplo de RPC.



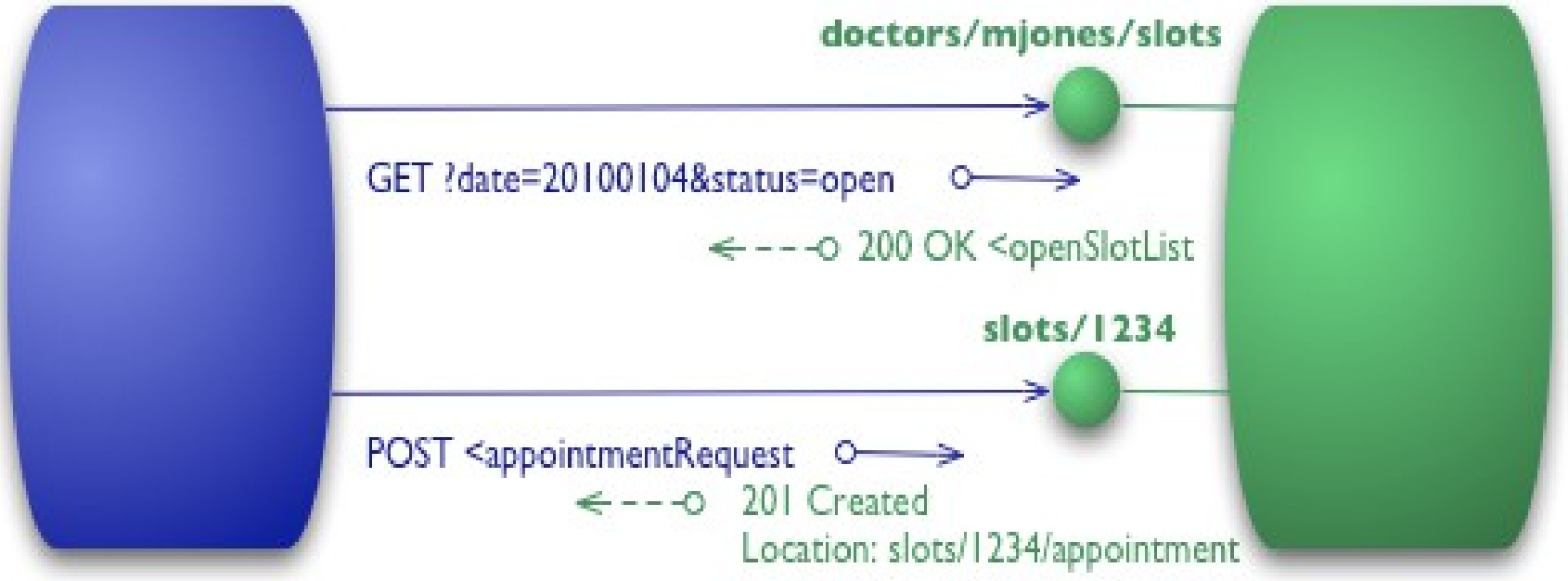
Level 1 - Resources

- As requisições são envidas para URIs que representam os resources e não para um único ponto de entrada.



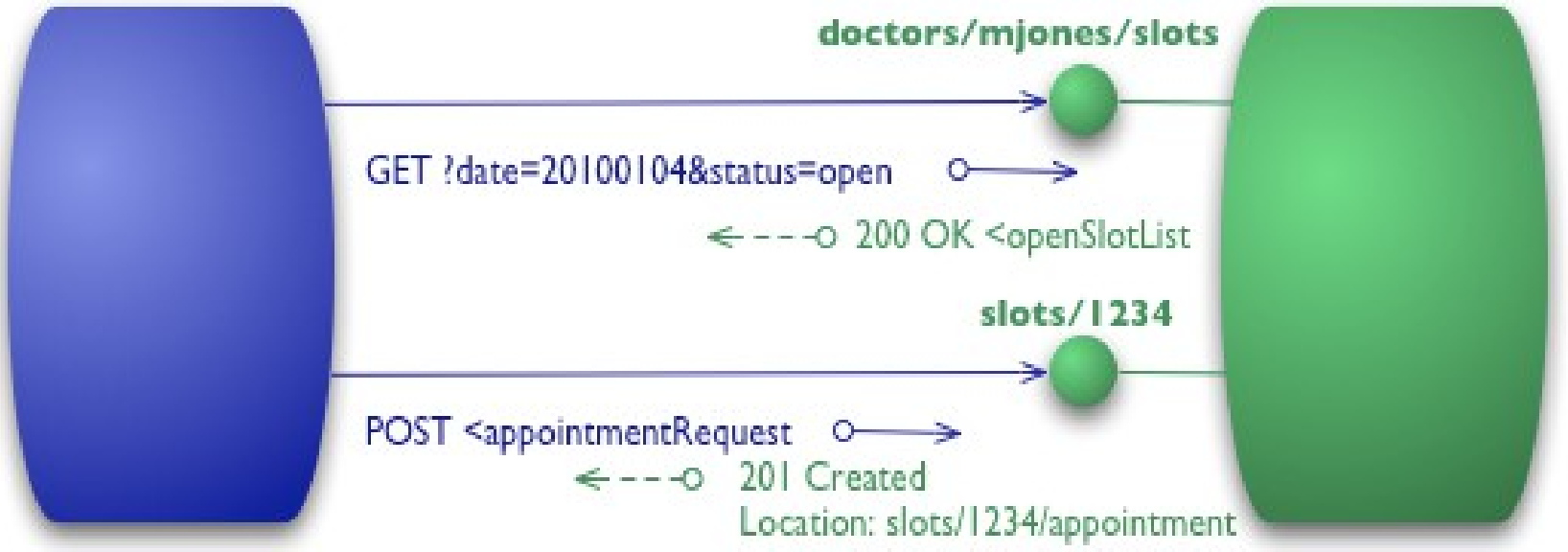
Level 2 – Verbos HTTP

- Faz o uso correto dos verbos HTTP para representar a operação sendo executada.



Level 3 – Hypermedia controls

- HATEOAS. Fornece links para as transições possíveis.



Referências

- http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
- http://restapitutorial.com
- http://martinfowler.com/articles/richardsonMaturityModel.html

Dúvidas?