

<?php



RESTful

Arquitetura RESTful utilizando PHP

Prof. Luiz Henrique de Angeli
 luizdeangeli@gmail.com
 luiz.angeli@unicesumar.edu.br
 facebook.com/luizdeangeli
 @luizdeangeli

Conteúdo da Disciplina

- Entender o funcionamento do protocolo HTTP
- Conhecer o funcionamento REST/RESTful
- Conhecer e utilizar o CURL
- Utilizar um framework para criar rotas
- Criar um aplicação RESTful utilizando o framework em PHP
- Criar o front-end para consumir o RESTful criado.
- Trabalhando com Cache



Quem sou?

- [Luiz Henrique de Angeli](#)
- Graduado em [Processamento de Dados](#) pela Unicesumar – Centro Universitário Cesumar (2006)
- Pós-Graduado em [Desenvolvimento de Sistemas Orientado a Objetos Java](#) pela Unicesumar – Centro Universitário Cesumar (2010)
- Professor:
 - [Graduação](#) desde 2007, Ministrando disciplinas na área de desenvolvimento: PHP, HTML, CSS, JS, DELPHI, SHELL SCRIPT, C, PASCAL;
 - [Pós-graduação](#) na área de desenvolvimento para WEB;
- Head de Desenvolvimento de Sistemas EAD.
- Desenvolvimento para WEB desde 2005




Calendário da Disciplina

16/02/2018
23/02/2018
09/03/2018



Aula 1

- Protocolo HTTP
- REST
- REST x SOAP
- JSON
- XML
- RESTful
- CURL
- Utilizando a Biblioteca libcurl do PHP
- Configurando o Apache
- Criando uma aplicação RESTful



RFC

Request for Comments

RFC

- RFC é um acrônimo para o inglês ***Request for Comments*** que pode ser traduzido do inglês com o significado de Request for Change, ou Requisição de Mudança.
- O RFC é um documento que descreve os padrões de cada protocolo da Internet a serem considerados um padrão. Os assuntos variam desde especificações, padrões e normas técnicas.

RFC

- Os RFC's são documentos técnicos desenvolvidos e mantidos pelo IETF (Internet Engineering Task Force) instituição que especifica os padrões que serão implementados e utilizados em toda a internet.
- São documentos públicos de acesso a qualquer pessoa para ler, comentar, enviar sugestões e/ou relatar experiências sobre o assunto.

RFC

- Site oficial: <http://www.faqs.org/rfcs>.
- O processo de desenvolvimento de um RFC também está descrito no RFC 2026.



Protocolo HTTP

HTTP - História



- **HTTP** é o acrônico de *HyperText Transfer Protocol* (Protocolo de Transferência de Hipertexto)
- Este **protocolo** é a base sobre qual a web esta construída.
- Foi criado por **Timothy John Berners-Lee** juntamente com a linguagem de marcação HTML em 1990.
- Toda transação HTTP consiste em uma **solicitação** e **resposta**.



HTTP - História

- HTTP trabalha na porta **80** ou **8080** e **443**.
- Utiliza o protocolo TCP para transporte. TCP é **Transmission Control Protocol** (Protocolo de controle de transmissão)
- O protocolo não guarda estado (o acesso de uma URI para outra não é mantido)

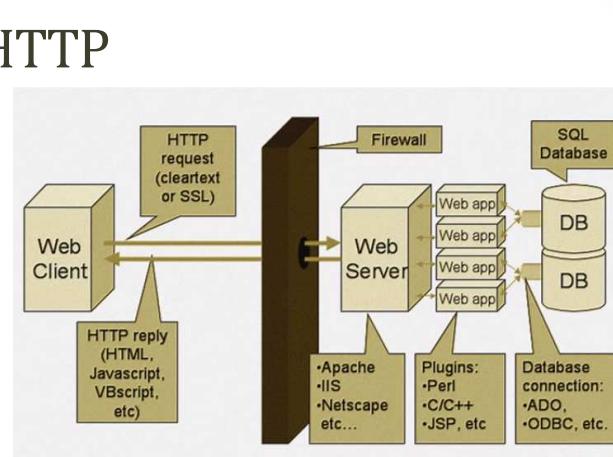
HTTP - História



- **Componentes do WWW.**

- **Protocolo HTTP:** Responsável pelas mensagens de requisição e resposta entre clientes e servidores;
- **Clientes:** Responsáveis pela interação do usuário para acesso aos dados disponibilizados;
- **Servidores:** Responsáveis por disponibilizar os arquivos e por gerar conteúdo dinâmico.
- **Linguagem HTML:** Linguagem interpretada nos clientes.

HTTP



HTTP - História



- **(1991) - HTTP/0.9 (<http://www.w3.org/Protocols/HTTP/AsImplemented.html>)**

- Transferência de dados somente no formato de texto ASCII;
- Suporte somente ao método GET;
- Versão inicial com algumas falhas.

MIME Media Types:
 text/html
 text/plain
 application/pdf
<http://www.iana.org/assignments/media-types/media-types.xhtml>

- **(1996) - HTTP/1.0 (<http://www.faqs.org/rfcs/rfc1945.html>)**

- RFC 1945
- Suporte a objetos multimídia, métodos adicionais e cabeçalho HTTP.
- Possibilitou uso de formulário interativos (PUT e POST).
- Uso acadêmico e comercial.

HTTP - História

- **(199X) - HTTP/1.0 +**

- Gambiarras criadas durante a década de 90 por desenvolvedores de clientes e servidores HTTP para melhorar o protocolo.
- As melhorias extraoficiais: conexões persistentes, suporte a hosts virtuais e conexões a proxies.

- **(1999) - HTTP/1.1 (<http://www.faqs.org/rfcs/rfc2616.html>)**

- Implementações extraoficiais do 1.0+ viraram oficiais;
- Versão do protocolo (RFC 2616);
- Melhorias do projeto do protocolo;
- Introdução de melhorias em desempenho e otimizações de funcionalidades;
- Suporte a aplicações mais sofisticadas (cabeçalhos diferentes, transferir arquivos)

HTTP - História



- **(2015) - HTTP/2.0**

- Versão do protocolo (RFC 7540);
- Melhor compressão de campos do header (cabeçalho);
- Maior velocidade na carga de páginas;
- Criptografia mais forte;
- Transmissões simultâneas (multiplexing) de pedidos (requests) com prioridade;
- Redução na latência de rede;
- Aumento no posicionamento (page-rank) nos resultado de busca devido a fatos como: a página carregar mais rapidamente, ter criptografia e compressão;

HTTP - História



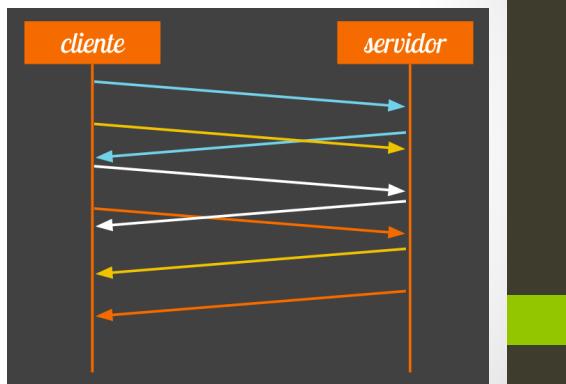
- **(2015) - HTTP/2.0**

- **Compressão automática:** O GZIP é padrão e obrigatório;
- No HTTP 2.0, os headers são binários e comprimidos usando um algoritmo chamado HPACK;
 - Tanto o request quanto o response levam vários cabeçalhos que não são comprimidos;
 - No HTTP 1.1 e ainda viajam em texto puro.
- Permite uso sem SSL, mas na prática todo mundo vai suportar apenas conexões seguras HTTPS.
- Mandamos apenas os cabeçalhos que mudarem entre requisições.

HTTP - História



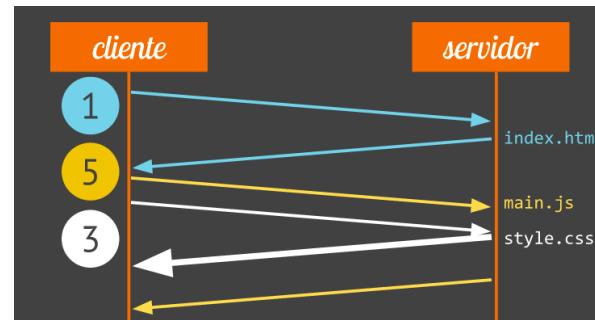
- Paralelização de requests com multiplexing



HTTP - História

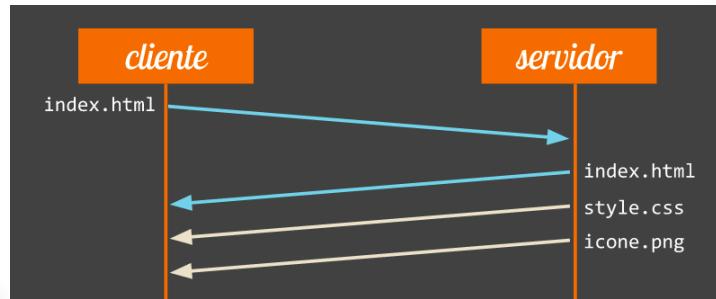


- Priorização de requests



HTTP - História

- Server-Push



HTTP - História

- Entendendo o *multiplexing*

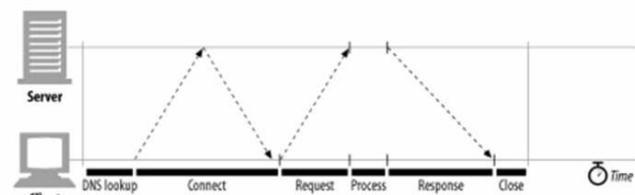


HTTP - Funcionamento

- O HTTP funciona como um protocolo de requisição-resposta no modelo computacional cliente-servidor.
- Um navegador web, por exemplo, pode ser o cliente e uma aplicação em um computador que hospeda um site da web pode ser o servidor.
- O cliente submete uma mensagem de requisição HTTP para o servidor, que retorna uma mensagem resposta para o cliente.
- A resposta contém informações de estado completas sobre a requisição e pode também conter o conteúdo solicitado no corpo de sua mensagem.

HTTP - Funcionamento

Linha do tempo de uma requisição HTTP



HTTP - Funcionamento



- O protocolo HTTP é composto de várias partes:
 - A URI para qual a solicitação foi direcionada:
 - O verbo usado:
 - Cabeçalhos e Códigos de status
 - Corpo da resposta.
- **Vamos detalhar estes itens nos próximos slides!**

URI x URL x URN



URL (<http://www.faqs.org/rfcs/rfc1738.html>)



- **URL – Uniform Resource Locator** (Localizador de Recursos Universal)
- Como o próprio nome diz, se refere ao local, o Host que você quer acessar determinado recurso.
- O objetivo da URL é associar um endereço remoto com um nome de recurso na Internet.
- Exemplo de URL
 - webdev.com.br
 - webdev.org
 - webdev.xyz

URN (<http://www.faqs.org/rfcs/rfc8141.html>)



- **URN – Uniform Resource Name** (Nome de Recursos Universal)
- É o nome do recurso que será acessado e também fará parte da URI.
 - home.html
 - contato.php
 - servicos.html
 - /api/v01/exemplo/usuarios

URI

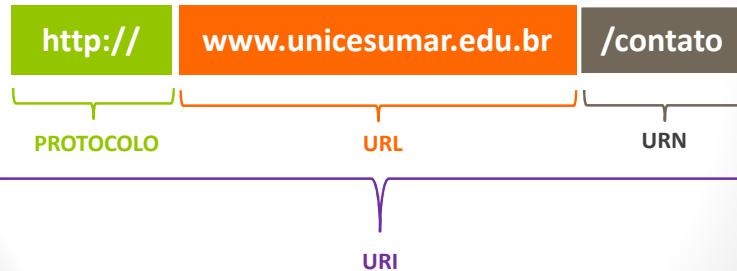
(<http://www.faqs.org/rfcs/rfc3986.html>)



- **URI – Uniform Resource Identifier** (Identificador de Recursos Universal);
- Como diz o próprio nome, é o identificador do recurso.
- Pode ser uma imagem, uma página, etc, pois tudo o que está disponível na internet precisa de um identificador único para que não seja confundido.
- Exemplos de URI
 - <http://woliveiras.com.br/desenvolvedor-front-end/>
 - http://pt.wikipedia.org/wiki/Hypertext_Transfer_Protocol

URI

- Então a **URI** une o Protocolo, URL e URN



URI

URI



- **URI** é o endereço de um recurso disponível em uma rede, seja a rede internet ou intranet, e significa em inglês *Universal Resource Identifier* ou **Identificador Universal de Recurso** em português.
- Em outras palavras, uri é um endereço virtual com um caminho que indica onde está o que o usuário procura.
- Pode ser tanto um arquivo, como uma máquina, uma página, um site, uma pasta etc.
- O formato das URI é definido pela norma RFC 3986 (<https://tools.ietf.org/html/rfc3986>).



URI

- Sintaxe:

```
esquema://domínio:porta/caminho/recurso?query_string#fragmento
• O esquema é o protocolo. Poderá ser HTTP, HTTPS, FTP etc.
• O domínio é o endereço da máquina.
• A porta é o ponto lógico no qual se pode executar a conexão com o servidor. (opcional)
• O caminho especifica o local onde se encontra o recurso.
• A query string é um conjunto de um ou mais pares "pergunta-resposta" ou "parâmetro-argumento" (ex: nome=fulano). (opcional)
• O fragmento é uma parte ou posição específica dentro do recurso. (opcional)
```



Verbos

(operações ou métodos)

Verbos

- Verbos HTTP nos permitem enviar a intenção, juntamente com a URI, para que possamos instruir o servidor sobre o que fazer com ela.
- O protocolo HTTP define um conjunto de métodos que o cliente pode invocar, que funcionam como comandos enviados ao servidor web.
- O HTTP possui oito métodos disponíveis (**GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS e CONNECT**)



Verbos

- GET**: recupera informações sobre o recurso identificado pela URI. Ex: listar produtos, visualizar o produto 45.
 - Uma requisição GET não deve modificar nenhum recurso do seu sistema, ou seja, não deve ter nenhum efeito colateral, você apenas recupera informações do sistema.
- POST** - adiciona informações usando o recurso da URI passada. Ex: adicionar um produto.
 - Pode adicionar informações a um recurso ou criar um novo recurso.



Verbos

- **PUT** - adiciona (ou modifica) um recurso na URI passada. Ex: atualizar um produto.
 - A diferença fundamental entre um PUT e um POST é que no POST a URI significa o lugar que vai tratar a informação, e no PUT significa o lugar em que a informação será armazenada.
- **DELETE** - remove o recurso representado pela URI passada. Ex: remover um produto.



Verbos

- **HEAD, OPTIONS e TRACE** - recuperam metadados da URI passada. Respectivamente o Header, quais métodos são possíveis e informações de debug.
- **CONNECT** - Para uso com um proxy que possa se tornar um túnel SSL.



Cabeçalhos



Cabeçalho

- Em solicitações e respostas web normalmente o **corpo é parte mais importante** e, com frequência a parte com mais conteúdo.
- Mas os **cabeçalhos** proporcionam informações fundamentais tanto para a solicitação quanto para a resposta, o que permite o servidor e cliente se comunicarem de forma eficiente.



Cabeçalho HTTP

- **Campos de cabeçalho** HTTP são componentes da seção de cabeçalho de solicitação e resposta de mensagens no HTTP.
- Eles definem os parâmetros de funcionamento de uma transação HTTP;
- São pares de nome e valor separados por dois pontos;
- A seguir vamos dar uma olhada nos campos de cabeçalho mais comuns.



Cabeçalho

Cabeçalho de Solicitud

```
> GET /api/v1/cargos HTTP/1.1
> User-Agent: curl/7.33.0
> Host: api.transparencia.org.br
> App-Token: zgaGuYXU9aoC
> Content-Type: application/json
> Accept: application/json
```

Verbo URN Versão HTTP
URL Headers

Cabeçalho da Resposta

```
< HTTP/1.1 200 OK
< Date: Sat, 10 Jan 2015 20:55:33 GMT
< Connection: close
< X-Powered-By: PHP/5.3.3-7+squeeze22
* Server Apache/2.2.16 (Debian) is not
blacklisted
< Server: Apache/2.2.16 (Debian)
< Content-Type: application/json
< Content-Length: 424
```

Versão HTTP Status Conteúdo



Cabeçalho HTTP



| Cabeçalho | Solicitação | Resposta | Observações |
|----------------|-------------|----------|---|
| Accept | SIM | | Mostra os formatos, com uma indicação da preferência, de maneira que o cliente solicitante possa entender. Os cabeçalhos adicionais Accept-Charset, Accept-Encoding e Accept-Language estão relacionados. |
| Authorization | SIM | | São informações de formato livre para provar a identidade de um usuário. São usadas em autenticação básica. |
| Cookie | | | Os cookies são pares chave/valor enviados juntamente com cada solicitação, separados por ponto e vírgula. |
| Content-Length | SIM | SIM | Qualquer solicitação ou resposta com conteúdo no corpo também deve incluir Content-Length com o tamanho em bytes no cabeçalho. Em geral a biblioteca HTTP vai calcular isso. |
| Content-Type | SIM | SIM | Qualquer solicitação ou resposta com conteúdo no corpo deve incluir este cabeçalho para prover informações sobre o formato deste conteúdo. |

Cabeçalho HTTP



| Cabeçalho | Solicitação | Resposta | Observações |
|---------------|-------------|----------|--|
| Etag | | SIM | É um identificador da versão do recurso que está sendo retornado. |
| Last-Modified | | SIM | Fornecce informações sobre quando esse recurso foi atualizado pela última vez. |
| Location | | SIM | Fornecce informações sobre uma localização e é usado com os códigos de status 300 quando houver redirecionamento, ou com 201/202 para dar informações sobre a localização do novo recurso. |
| Set-Cookie | | SIM | Envia cookies para que sejam armazenados no cliente e enviados de volta em um cabeçalho Cookie, juntamente com as solicitações posteriores. |
| User-Agent | SIM | | Fornecce informações sobre o software cliente que está fazendo a solicitação. |

Lista de todos os campos na RFC: <http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html>

Lista dos campos com exemplos: http://en.wikipedia.org/wiki/List_of_HTTP_header_fields

Cabeçalho HTTP



- Quando uma solicitação é realizada, o servidor retornará um **código de status**.
- Esse código fornece informações sobre o status da solicitação.

| Total de faixas de códigos | Categoria |
|----------------------------|------------------|
| 100-199 | Informacional |
| 200-299 | Sucesso |
| 300-399 | Redirecionamento |
| 400-499 | Erro no Cliente |
| 500-599 | Erro no Servidor |

Cabeçalho HTTP 1XX



| Código | Observações |
|--------------------------|---|
| 100 (Continuar) | O solicitante deve continuar com a solicitação. O servidor retorna esse código para indicar que recebeu a primeira parte de uma solicitação e que está aguardando o restante. |
| 101 (Mudando protocolos) | O solicitante pediu ao servidor para mudar os protocolos e o servidor está reconhecendo a informação para, então, executá-la. |

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html> e <https://support.google.com/webmasters/answer/40132?hl=pt-BR>

Cabeçalho HTTP 2XX



| Código | Observações |
|----------------------------------|---|
| 200 (Bem-sucedido) | O servidor processou a solicitação. Geralmente, isso significa que o servidor forneceu a página solicitada. |
| 201 (Criado) | A solicitação foi bem-sucedida e o servidor criou um novo recurso. |
| 202 (Aceito) | O servidor aceitou a solicitação, mas ainda não a processou. |
| 203 (Informação não autorizável) | O servidor processou a solicitação com sucesso, mas está retornando informações que podem ser de outra fonte. |
| 204 (Sem conteúdo) | O servidor processou a solicitação com sucesso, mas não está retornando nenhum conteúdo. |
| 205 (Reconfigurar conteúdo) | O servidor processou a solicitação com sucesso, mas não está retornando nenhum conteúdo. Ao contrário da 204, esta resposta exige que o solicitante reconfigure o modo de exibição do documento (por exemplo, limpe um formulário para uma nova entrada). |
| 206 (Conteúdo parcial) | O servidor processou uma solicitação parcial GET com sucesso. |

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html> e <https://support.google.com/webmasters/answer/40132?hl=pt-BR>

Cabeçalho HTTP 3XX



| Código | Observações |
|------------------------------|--|
| 300 (Múltipla escolha) | O servidor tem várias ações disponíveis com base na solicitação. Ele pode escolher uma ação com base no solicitante (user agent) ou pode apresentar uma lista para que o solicitante escolha uma ação. |
| 301 (Movido permanentemente) | A página solicitada foi movida permanentemente para um novo local. Quando o servidor retornar essa resposta (como uma resposta para uma solicitação GET ou HEAD), ele automaticamente direcionará o solicitante para o novo local. Você deve usar esse código para fazer com que o Googlebot saiba que uma página ou um site foi permanentemente movido para um novo local. |
| 302 (Movido temporariamente) | O servidor está respondendo à solicitação de uma página de uma localidade diferente, mas o solicitante deve continuar a usar o local original para solicitações futuras. Esse código é semelhante ao 301 com relação a uma solicitação GET ou HEAD, pois direciona automaticamente o solicitante para um local diferente. No entanto, você não deve usá-lo para informar ao Googlebot que uma página ou um site foi movido, porque o Googlebot continuará rastreando e indexando o local original. |

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html> e <https://support.google.com/webmasters/answer/40132?hl=pt-BR>

Cabeçalho HTTP 3XX



| Código | Observações |
|-----------------------------------|--|
| 303 (Consultar outro local) | O servidor retornará esse código quando o solicitante precisar fazer uma solicitação GET separadamente para outro local para obter a resposta. Para todas as outras solicitações (com exceção de HEAD), o servidor direciona automaticamente para o outro local. |
| 304 (Não modificado) | A página solicitada não foi modificada desde a última solicitação. Quando o servidor retorna essa resposta, ele não retorna o conteúdo da página. Você deverá configurar o servidor para retornar essa resposta (chamada de cabeçalho If-Modified-Since HTTP) quando uma página não tiver sido alterada desde a última vez que o solicitante fez o pedido. Isso economiza largura de banda e evita sobrecarga, pois o servidor pode informar ao Googlebot que uma página não foi alterada desde o último rastreamento. |
| 305 (Utilizar proxy) | O solicitante poderá acessar a página solicitada utilizando um proxy. Quando o servidor retornar essa resposta, também indicará qual proxy o solicitante deverá usar. |
| 307 (Redirecionamento temporário) | O servidor está respondendo à solicitação de uma página de uma localidade diferente, mas o solicitante deve continuar a usar o local original para solicitações futuras. Esse código é semelhante ao 301 com relação a uma solicitação GET ou HEAD, pois direciona automaticamente o solicitante para um local diferente. No entanto, você não deve usá-lo para informar ao Googlebot que uma página ou um site foi movido, porque o Googlebot continuará rastreando e indexando o local original. |

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html> e <https://support.google.com/webmasters/answer/40132?hl=pt-BR>

Cabeçalho HTTP 4XX



| Código | Observações |
|----------------------------|--|
| 400 (Solicitação inválida) | O servidor não entendeu a sintaxe da solicitação. |
| 401 (Não autorizado) | A solicitação requer autenticação. O servidor pode retornar essa resposta para uma página que necessita de login. |
| 403 (Proibido) | O servidor está recusando a solicitação. Se você observar que o Googlebot recebeu esse código de status ao tentar rastrear páginas válidas de seu site (é possível ver isso na página Erros de rastreamento em Integridade nas Ferramentas do Google para webmasters), é possível que o servidor ou host esteja bloqueando o acesso do Googlebot. |
| 404 (Não encontrado) | O servidor não encontrou a página solicitada. Por exemplo, o servidor retornará esse código com frequência se a solicitação for para uma página que não existe mais no servidor. Se você não tiver um arquivo robots.txt em seu site e vir esse status na página URLs bloqueados nas Ferramentas do Google para webmasters, esse será o status correto. No entanto, se você tiver um arquivo robots.txt e notar esse status, esse arquivo poderá estar nomeado incorretamente ou no local errado. Ele deve estar no nível superior do domínio e ter o nome robots.txt. Se você vir esse status para URLs que o Googlebot tentou rastrear, provavelmente o Googlebot seguiu um link inválido de outra página (que pode ser antigo ou ter erros de digitação). |

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html> e <https://support.google.com/webmasters/answer/40132?hl=pt-BR>

Cabeçalho HTTP 4XX



| Código | Observações |
|--|---|
| 405 (Método não permitido) | O método especificado na solicitação não é permitido. |
| 406 (Não aceitável) | A página solicitada não pode responder com as características de conteúdo solicitadas. |
| 407 (Autenticação de proxy necessária) | Esse código de status é semelhante ao 401 (não autorizado), mas especifica que o solicitante deve autenticar usando uma proxy. Quando o servidor retornar essa resposta, também indicará qual proxy o solicitante deverá usar. |
| 408 (Tempo limite da solicitação) | O servidor atingiu o tempo limite ao aguardar a solicitação. |
| 409 (Conflito) | O servidor encontrou um conflito ao cumprir a solicitação. É necessário que o servidor inclua informações sobre o conflito na resposta. O servidor retorna esse código em resposta a uma solicitação PUT que entra em conflito com uma solicitação anterior, além de uma lista de diferenças entre as solicitações. |

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html> e <https://support.google.com/webmasters/answer/40132?hl=pt-BR>

Cabeçalho HTTP 4XX



| Código | Observações |
|--|--|
| 410 (Desaparecido) | O servidor retornará essa resposta quando o recurso solicitado tiver sido removido permanentemente. É semelhante ao código 404 (Não encontrado), mas às vezes é usado no lugar de um 404 para recursos que tenham existido anteriormente. Se o recurso foi movido permanentemente, você deve usar o código 301 para especificar o novo local do recurso. |
| 411 (Comprimento necessário) | O servidor não aceitará a solicitação sem um campo de cabeçalho "Comprimento-do-Conteúdo" válido. |
| 412 (Falha na pré-condição) | O servidor não cumpre uma das pré-condições que o solicitante coloca na solicitação. |
| 413 (Entidade de solicitação muito grande) | O servidor não pode processar a solicitação porque ela é muito grande para a capacidade do servidor. |
| 414 (o URI solicitado é muito longo) | O URI solicitado (geralmente um URL) é muito longo para ser processado pelo servidor. |

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html> e <https://support.google.com/webmasters/answer/40132?hl=pt-BR>

Cabeçalho HTTP 4XX



| Código | Observações |
|---------------------------------------|--|
| 415 (Tipo de mídia incompatível) | A solicitação está em um formato não compatível com a página solicitada. |
| 416 (Faixa solicitada insatisfatória) | O servidor retorna esse código de status se a solicitação for para uma faixa não disponível para a página. |
| 417 (Falha na expectativa) | O servidor não pode cumprir os requisitos do campo "Expectativa" do c |

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html> e <https://support.google.com/webmasters/answer/40132?hl=pt-BR>

Cabeçalho HTTP 5XX



| Código | Observações |
|--------------------------------|--|
| 500 (Erro interno do servidor) | O servidor encontrou um erro e não pode completar a solicitação. |
| 501 (Não implementado) | O servidor não tem a funcionalidade para atender à solicitação. Por exemplo, o servidor poderá retornar esse código quando não reconhecer o método de solicitação. |
| 502 (Gateway inválido) | O servidor estava operando como gateway ou proxy e recebeu uma resposta inválida do servidor superior. |
| 503 (Serviço indisponível) | O servidor está indisponível no momento (por sobrecarga ou inatividade para manutenção). Geralmente, esse status é temporário. |
| 504 (Tempo limite do gateway) | O servidor estava operando como gateway ou proxy e não recebeu uma solicitação do servidor superior a tempo. |
| 505 (Versão HTTP incompatível) | O servidor não é compatível com a versão do protocolo HTTP usada na solicitação. |

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html> e <https://support.google.com/webmasters/answer/40132?hl=pt-BR>

HTTP - Funcionamento



- Resumo do HTTP



Como funciona uma requisição HTTP.mp4

Segurança HTTP



Ataques mais comuns ao WWW

- Captura de tráfego sem criptografia
 - Usuário e Senha de Webmail são alvos principais
- Negação de Serviço
 - Gerar um grande número de requisições, causando sobrecarga no servidor.
- Exploração de vulnerabilidade
 - Falhas na aplicações dinâmicas que interagem com B.D. e S.O..
 - O problema esta no desenvolvedor e não na linguagem.



REST

O que é? De onde vem? O que comem? Como se reproduzem?

Para que possamos entender como o RESTful pode nos ajudar temos que inicialmente aprender sobre o que é REST.



REST

- REST é um termo definido por **Roy Fielding** (roy.gbiv.com) em sua tese de doutorado em 2000 no qual ele descreve sobre um **estilo de arquitetura** de software sobre um sistema operado em rede. (http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm)
- REST consiste em um conjunto de ideias sobre como os dados podem ser transferidos de modo elegante.
- O HTTP, geralmente funciona muito bem com os critérios do REST.



REST

- REST: *Representational State Transfer* em português *Transferência de Estado Representativo*
- É um estilo/modelo de arquitetura de software
- REST NÃO É:
 - só para criar APIs;
 - um padrão;
 - um protocolo;
 - tecnologia;
 - framework;



REST

Arquitetura de Software:

Como você organiza elementos de software para alcançar certos objetivos. ← Dados, Processamento, Conexão →

Performance, escalabilidade, simplicidade. → Perry e Wolf (1992)

Estilo Arquitetural

Como você alcança certos objetivos com determinadas regras de organização dos elementos de software.

REST

ISTO NÃO É UM CACHIMBO

A Traição das Imagens, René François Ghislain Magritte, 1929

Isto é então uma representação de um cachimbo.

É desta forma que o REST trabalha.

REST

- Para entender melhor, exploraremos o exemplo do Facebook.
- Devido ao seu grande crescimento como rede social, ele encontrou diversas oportunidades que o ajudaram a crescer ainda mais.
- Essas oportunidades têm em comum a necessidade de web services para suprir informações para as aplicações externas.
- Desta forma o Facebook escolheu a utilização do REST.
- Integrações com outros aplicativos, dispositivos móveis e internet.

REST

- À medida que um sistema web vai evoluindo, é comum que se torne necessário uma nova forma de se comunicar com ele.
- Porém, pode chegar o momento em que desejamos tornar possível que outras aplicações se comuniquem com nosso sistema.

Princípios do REST

- Interface Uniforme:** Com o objetivo de obter uma interface uniforme, REST define quatro requisitos de interface:
 - identificação de recursos:** cada recurso deve possuir um identificador universal (URI);
 - representação de recursos:** XML, JSON, TEXT
 - mensagens auto-descritivas:** Os pedidos e respostas devem conter meta-dados que indicam como o conteúdo deve ser tratado (cabeçalhos, autenticação, etc.);
 - hipermídia como mecanismo de estado da aplicação:** As representações obtidas devem possuir hiperlinks que permitam a navegação do cliente pelos recursos.*

*Este pré-requisito é o menos cumprido por aplicações autointituladas RESTful.

Princípios do REST

- **Cliente-Servidor:** esta característica é mais comumente encontrada em aplicações. Um cliente envia uma requisição para o servidor. O servidor então pode tanto rejeitar como executar o serviço solicitado, e retornar uma resposta ao cliente.
- **Stateless (Sem estado):** A comunicação deve ser feita sem o armazenamento de qualquer tipo de estado no servidor. Portanto, estados de sessão, quando necessários, devem ser totalmente mantidos no cliente.



Princípios do REST

- **Cache:** uma forma de diminuir o impacto da desvantagem trazida pela redução de desempenho é a utilização de cache.
- **Multicamada:** com o intuito de aperfeiçoar o requisito de escalabilidade da Internet, foi adicionado ao estilo REST a característica de divisão em camadas:
- **Code-On-Demand:** O cliente deve ser capaz de executar scripts armazenados no servidor de forma a estender as funcionalidades do cliente.



Princípios do REST

- **Listagem 1:** Exemplos de URN que não seguem os princípios do REST.
 - /listarProdutos = URN para listar todos os produtos
 - /obterProduto/1 = URN para obter o produto 1
 - /criarProduto = URN para criar produto
 - /deletarProduto = URN para deletar produto

Não utilizar verbos nas URNs
- **Listagem 2:** Exemplos de URN que seguem os princípios do REST.
 - /produtos = URN para ações sobre recursos do tipo Produto
 - /produtos/1 = URN para ações sobre um produto específico



REST

- Alguns fatores de por que utilizar REST:
 - **Escalabilidade:** Web cache
 - **Performance Alta:** Web proxy
 - **Alta Disponibilidade:** Load Balancer
 - **Permitir evolução sem parar o sistema:** Load Balancer
 - **Permitir evolução sem quebrar os clientes:** HTML, JSON, XML
 - **Segurança:** HTTPS



SOAP x REST

SOAP

- Simple Object Access Protocol (Protocolo Simples de Acesso a Objetos)
- é um protocolo para troca de informações estruturadas em uma plataforma descentralizada e distribuída;
- Ele se baseia na Linguagem de Marcação (XML);
- Assim como o HTTP o SOAP é outra forma de colocar mensagens em envelopes.

Desvantagens do SOAP

- O SOAP é uma especificação de um protocolo estruturado em XML. Elas naturalmente ocupam mais rede;
- A complexidade na comunicação com um web service SOAP também é um problema para aplicações Mobile e HTML5, visto que são necessárias bibliotecas específicas, utilização de geradores de código e ainda a manutenção do código gerado nas diferentes plataformas;

Desvantagens do SOAP

- O SOAP foi construído com o pensamento da comunicação entre servidores, em que é assumido que eles são naturalmente seguros;
- O REST é mais simples, foi projetado para ser usado em clientes “magros”, o que o torna ideal para utilização em dispositivos com capacidades limitadas;

Desvantagens do SOAP

- As respostas do REST são cacheáveis, diferente do SOAP. Isso dá um grande aumento de performance em clientes simples;
- Enquanto o SOAP utiliza apenas XML, o REST pode se comunicar através de diversos formatos, sendo JSON o mais usado.
- A transferência de dados com o uso de JSON causa uma carga menor na rede, além de ser mais facilmente consumido por clientes construídos com qualquer linguagem, em especial HTML5.

{ JSON }

JSON

- JSON é a abreviatura de JavaScript Object Notation (Notação de Objetos JavaScript).
- Não se deixe enganar pelo nome, embora seja como se fosse específico de JavaScript, esse formato pode ser facilmente lido e escrito por uma ampla variedade de linguagens.
- É um formato bem simples, leve que pode representar dados aninhados e estruturados.

JSON

- Por exemplo, se houvesse um conjunto de dados que tivesse o seguinte aspecto:
 - mensagem
 - en : "hello friend"
 - pt : "olá amigo"
- Em JSON, esses dados se pareceriam com:
 - {"mensagem" : {"en":"hello friend", "pt":"olá amigo"}}

Quando optar pelo JSON

- O JSON oferece uma indicação bem clara da estrutura de dados original e disponibiliza os dados que estão nessa estrutura.
- Mas não nos oferece nenhuma informação sobre o tipo de dado. Com frequência, isso não é importante.
- O ponto mais forte do JSON está no fato de ser um formato de dados simples. Ele não exige muito espaço para armazenamento em comparação com o XML.

JSON

- Também é possível mostrar uma lista de itens, considere a seguinte lista:
 - Leite, Arroz, Ovos, Banana, Maça
- Em JSON, sua representação seria:
 - `["Leite", "Arroz", "Ovos", "Banana", "Maça"]`
- Se a lista acima fosse, o valor de uma propriedade, então colchetes e chaves estariam presentes:
 - `{"lista": ["Leite", "Arroz", "Ovos", "Banana", "Maça"]}`

Quando optar pelo JSON

- O JSON também não é grande demais para ser transferido por dispositivos móveis ou por uma conexão de dados lenta e irregular.
- Como o formato JSON é bem simples e compacto em termos de processamento, decodificá-lo não é custoso, o que torna ideal para dispositivos menos potentes como telefone.

Quando optar pelo JSON

- Use JSON quando as informações sobre o formato exato dos dados não forem cruciais e quando não for necessário exigir muito esforço para decodificá-lo.
- É ótimo para aplicativos web ou aplicativos móveis e é claro, se você estiver fornecendo dados para um consumidor JavaScript, pois ele lida esse formato de dados de modo nativo e rápido.

Lidando com o JSON em PHP

- Em PHP você pode utilizar a função `json_encode` para transformar um array ou um objeto em um JSON válido.

```
echo json_encode(array("message"=>"hello you"));
```

- Para lidar com dados JSON de entrada e transformá-lo em uma estrutura que você possa utilizar, basta usar a função `json_decode`, passando a string contendo o JSON com primeiro argumento.

```
var_dump(json_decode('{"message":"hello you"}'));
```

Lidando com o JSON em PHP

- Exemplo de retorno de um array de objetos

```
$objetos = array();
$objeto = new stdClass();
$objeto->id = 10;
$objeto->nome = "Luiz Henrique";
$objetos[] = $objeto;

$objeto = new stdClass();
$objeto->id = 15;
$objeto->nome = "Maria da Silva";
$objetos[] = $objeto;

echo json_encode($objetos);
```

Lidando com o JSON em PHP

- Exemplo de retorno de um objeto

```
$objeto = new stdClass();
$objeto->id = 10;
$objeto->nome = "Luiz Henrique";

echo json_encode($objeto);
```

Lidando com o JSON em PHP

- Exemplo de retorno de um array de objetos

Lidando com o JSON em PHP

- Ao utilizar o `json_encode` o php retorna a representação JSON de um valor.
- Porém ao utilizar a função em uma classe com atributos privados (`private`) estes dados não serão apresentados.
- Para resolver este problema o php oferece uma interface `JsonSerializable` para implementar o método `jsonSerialize` que retorne os atributos e valores.
- Esta interface esta disponível acima da versão 5.4 do PHP.

http://php.net/manual/pt_BR/jsonserializable.jsonserialize.php

Lidando com o JSON em PHP

- Exemplo de utilização do [JsonSerializable](#)

```
class Pessoa implements JsonSerializable {

    private $nome = "Luiz Henrique";

    public function jsonSerialize() {
        return get_object_vars($this);
    }

}

$pessoa = new Pessoa();
echo json_encode($pessoa);
```

XML

< XML />

XML

XML

- O XML é outro formato de dados bem comum, usado com APIs.
- O XML é um formato prolixo; a pontuação adicional e o escopo de atributos, os dados de caracteres e as tags aninhadas podem gerar dados um pouco mais extensos do que gerados por outro formato.
- O XML possui muito mais recursos que o JSON e pode representar muito mais dados.

XML

XML

- A representação de uma lista é:

```
<!xml version="1.0"?>
<lista>
    <item>Leite</item>
    <item>Arroz</item>
    <item>Ovos</item>
    <item>Banana</item>
    <item>Maça</item>
</lista>
```

Quando optar pelo XML

XML

- A capacidade do XML de representar atributos, filhos e dados de caracteres permitem uma maneira mais poderosa e descritiva de representar dados em relação ao JSON.
- O XML pode incluir informações sobre tipos de dados e tipos customizados de dados, e cada elemento pode ter atributos que incluem mais informações ainda.
- O formato de dados mais extenso não representa problema quando trabalhamos com máquinas potentes e conexões rápidas de rede.

Quando optar pelo XML

XML

- O XML é popular entre muitas plataformas corporativas de tecnologia como Java, Oracle e .NET e por isso usuários dessas tecnologias frequentemente requisitarão o XML como formato preferido.

Lidando com o XML em PHP

XML

- Trabalhar com o XML em PHP não é tão fácil quanto trabalhar com JSON. Há muitas maneiras de trabalhar o XML e todas elas em situações diferentes. Há três abordagens principais:
 - [SimpleXML](#) é a maneira mais acessível é fácil de usar e entender, mas, possui algumas limitações.
 - [DOM](#) é mais poderoso e, sendo assim, mais complicado de usar, existe funções prontas e é muito ser utilizado junto com o SimpleXML para suprir as limitações deste.
 - [XMLReader](#), [XMLWriter](#) e [XMLParser](#) são maneiras de baixo nível para lidar com o XML, são complicadas e não são intuitivas, mas tem uma vantagem, não carregam todo o documento XML na memória de uma só vez.

Lidando com o XML em PHP

XML

```
<?php

$lista = array("Leite", "Arroz", "Ovos", "Banana", "Maça");

$xml = new SimpleXMLElement("<lista />");

foreach ($lista as $item) {
    $xml->addChild("item", $item);
}

//para uma saída em arquivo XML
$dom = dom_import_simplexml($xml)->ownerDocument;
$dom->formatOutput = true;
echo $dom->saveXML();

?>
```

RESTful



RESTful

- Os Web Services RESTful são serviços construídos com o estilo de arquitetura REST.
- A construção de Webservices com a abordagem RESTful está surgindo como uma alternativa popular ao uso de tecnologias baseadas em SOAP para implantação de serviços na Internet, por ser mais leve e ter a capacidade de transmitir dados diretamente via HTTP.



REST x RESTful

- REST é um modelo de arquitetura de software
- RESTful é uma implementação de um "webservice" simples utilizando o HTTP e os princípios REST.



Como o RESTful funciona

- No RESTful, a requisição e a resposta podem ser configuradas da seguinte forma:
 - A entrada é definida pela URI de acesso ao servidor, e essa URI obedece aos métodos HTTP;
 - Quando um cliente faz uma requisição ao serviço RESTful, ele aguarda normalmente uma resposta que, na maioria das vezes é no formato JSON.
 - Poderia ser um texto puro ou XML.



Operações

- Uma das características mais importantes de REST é que você tenha um conjunto pequeno e fixo de operações bem definidas, gerando uma interface uniforme.
- Basicamente temos a seguinte convenção.
 - **GET** é usado para listar dados, listar registro ou um cálculo.
 - **POST** é usado para adicionar dados
 - **PUT** é usado para editar dados.
 - **DELETE** é usado para deletar dados.

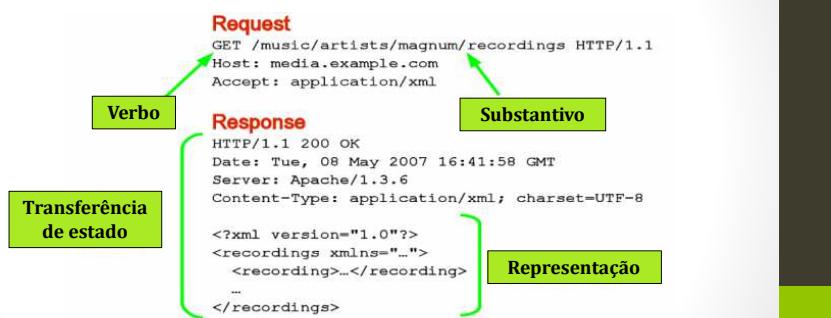


Operações (exemplo)

| Convenção | Descrição | Método |
|---------------------------------|---|---------------|
| GET /tarefas | Obter todas as tarefas | getAll |
| GET /tarefas/findAll/Manutencao | Obter todas as tarefas por meio de uma busca por manutenção | findAll |
| GET /tarefas/1 | Obter a tarefa cuja o id é 1 | getById |
| POST /tarefas/ | Adicionar uma tarefa | addTarefas |
| PUT /tarefas/1 | Editar uma tarefa cujo o id é 1 | editTarefas |
| DELETE /tarefas/1 | Deletar uma tarefa cujo id é 1 | deleteTarefas |
| GET /tarefas/concluidas | Chamar o método concluídas da classe tarefas | concluidas |



Exemplo Requisição e Resposta



Recursos do HTTP em REST

- O REST extrai o máximo dos melhores recursos do HTTP, colocando todos os metadados sobre a solicitação e a resposta nos cabeçalhos e reservando o corpo principal para o conteúdo.
- Isso significa que um serviço RESTful implementado corretamente fará o uso de **verbos**, **códigos de status** e **cabeçalhos**.



Ler Registros em REST



- Para acessar representações de recursos, uso o verbo **GET** aplicando a uma coleção ou a um recurso individual, sem enviar nenhum conteúdo junto a solicitação GET.
- Os recursos geralmente aparecerão exatamente com a mesma estrutura, em uma coleção ou individual.

Ler Registros em REST



- O código de status será igual a **200** caso o(s) registro(s) tenha(m) sido obtido(s) com sucesso.
- Outros códigos também podem ser usados, como **302 “Found”** (Encontrado) ou **304 “Not Modified”** (Não Modificado)

Ler Registros em REST



- Porém se o registro não for encontrado o problema será retornado.
- **Pode-se utilizar:**
 - **404 “Not Found”** (Não encontrado) para indicar que o registro não foi encontrado ou que não existe.
 - **401 “Not Authorized”** (Não autorizado) se o usuário não estiver autenticado.
 - **403 “Forbidden”** (Proibido) para usuário autenticados sem permissão de acesso ao registro.

Criar Recursos em REST



- Os recursos são criados por meio de uma solicitação **POST** para a coleção à qual o novo recurso irá pertencer.
- O corpo da solicitação conterá uma representação de um novo recurso.
- Se o recurso for criado de modo bem-sucedido, um código de status indicando sucesso será incluído na resposta.

Criar Recursos em REST



- É comum escolher um código de status igual a **201** que significa “**Created**”, (**Criado**).
- Quando um novo recurso for criado, e retornar uma representação do novo recurso. É perfeitamente válido retornar um **200** que significa “**Accepted, but not completed**” (**Aceito, mas não completado**)
- Se o recurso não poder ser criado, podemos utilizar os códigos **400 “Bad Request”** (**Solicitação inválida**) ou **406, Not Acceptable** (**Não Aceitável**)

Atualizar Registros em REST



- Editar registro com serviços RESTful consiste em um processo com múltiplos passos.
 - Em primeiro lugar, o recurso deve ser obtido com **GET**.
 - Em seguida, a representação do recurso poderá ser alterada conforme necessária.
 - E o recurso deve ser colocado de volta usando **PUT** em seu URI.
- De maneira idêntica ao que ocorre quando um recurso é criado usando o POST, a solicitação PUT incluirá a representação do recurso no corpo.

Apagar Registros em REST



- O verbo **DELETE** é enviado em uma solicitação para o URI do item a ser apagado, sem a necessidade de haver conteúdo no corpo.
- Muitos serviços retornaram **200** para “**OK**” ou simplesmente **“204”** para “**Not Content**” (**Sem Conteúdo**), quando o item for apagado com sucesso.
- Ou **404 “Not Found”** (**Não encontrado**) se o item não existir.
- <http://www.restapitutorial.com/httpstatuscodes.html>

Segurança em REST



Segurança em REST



- Hoje em dia o desenvolvimento de componentes com APIs é cada vez mais frequente e um sistema normalmente se integra com um ou mais sistemas da mesma empresa.
- Se suas APIs abrem possibilidades de interação em outros sistemas da empresa, os cuidados devem ser redobrados.

Mecanismos de segurança comuns



Restrição de acesso pela infraestrutura de rede

- Em APIs internas, às vezes o controle da segurança fica totalmente implementado pela infraestrutura.
- Restrições de acesso por IP implementados em roteadores, firewalls e balanceadores de carga.
- Restrição também de protocolos de comunicação permitidos e portas.

Mecanismos de segurança comuns



Comunicações HTTPS

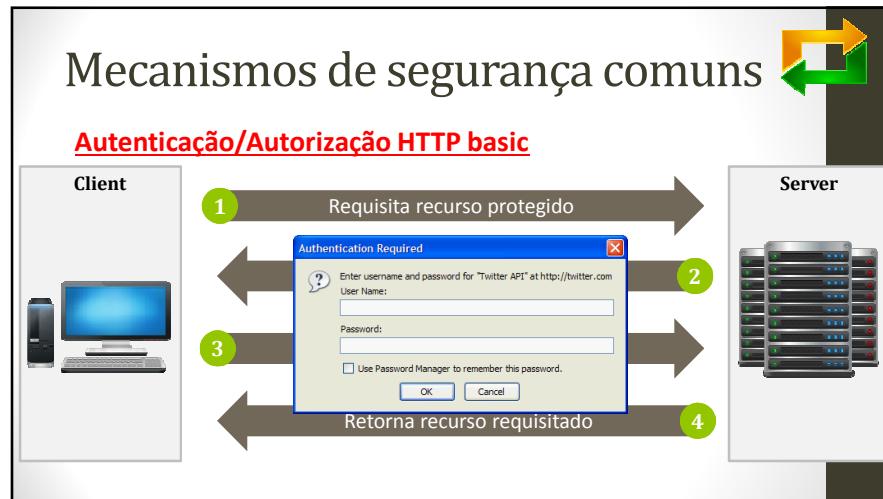
- O HTTPS é um protocolo para comunicações seguras que vem sendo amplamente utilizado na internet há muitos anos.
- Ele provê autenticação entre usuários finais e servidores, e também na comunicação entre servidores.
- Servidores que se comunicam via HTTPS possuem chance muito menor de brechas de segurança.

Mecanismos de segurança comuns



Autenticação/Autorização HTTP basic

- A autenticação HTTP basic é uma forma bem simples de um cliente informar suas credenciais de acesso ao fazer uma requisição HTTP.
- Não requer cookies, identificadores de sessão ou páginas de login. São usados somente cabeçalhos HTTP estáticos.
- O usuário e senha do cliente são enviados ao servidor em Base64.
- Esta forma é vulnerável a interceptações na redes. Porém, o uso de HTTPS para proteger o canal resolve este problema.



Mecanismos de segurança comuns

Autenticação/Autorização HTTP basic

```

<?php
if (!isset($_SERVER['PHP_AUTH_USER'])) {
    header('WWW-Authenticate: Basic realm="My Realm"');
    header('HTTP/1.0 401 Unauthorized');
    echo 'Texto enviado caso o usuário clique no botão Cancelar';
    exit;
} else {
    echo "<p>Olá, ($_SERVER['PHP_AUTH_USER']).</p>";
    echo "<p>Você digitou ($_SERVER['PHP_AUTH_PW']) como sua senha.</p>";
}
?>

```

http://php.net/manual/pt_BR/features.http-auth.php



Mecanismos de segurança comuns

Autenticação/Autorização HTTP Digest

```

.....
if (empty($_SERVER['PHP_AUTH_DIGEST'])) {
    header('HTTP/1.1 401 Unauthorized');
    header('WWW-Authenticate: Digest realm="'.$realm.
        '",qop="auth",nonce="'.uniqid().'",opaque="'.md5($realm).'");
    die('Texto enviado caso o usuário clique no botão Cancelar');
}
.....

```

http://php.net/manual/pt_BR/features.http-auth.php

Mecanismos de segurança comuns



Autenticação/Autorização através de Certificados

- A autenticação/autorização através de certificados também do lado do cliente é um refinamento adicional de segurança.
- Esta forma é bastante segura, porém o trabalho e custo de lidar com certificados dos 2 lados é razoável, sendo adequada só em cenários muito sensíveis de segurança.

Mecanismos de segurança comuns



Token-based authorization

- Esta é uma forma simples e segura de controlar autenticação/autorização de serviços entre servidores, embora não seja um padrão.
- Como funciona:**
 - Aplicação cliente autentica-se no servidor informando suas credenciais;
 - Aplicação servidora valida credenciais e guarda um token, vinculando-o ao IP de origem do cliente e uma data validade;
 - Servidor envia mensagem de sucesso para o cliente e envia o token;
 - Cliente invoca serviços no servidor sempre enviando o token;
 - Servidor valida cada requisição checando se foi enviado um token válido;

Mecanismos de segurança comuns



Token-based authorization

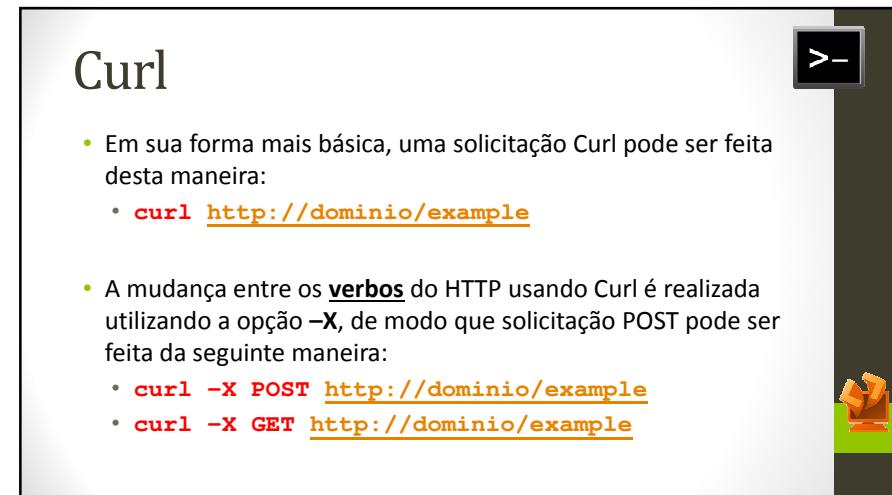
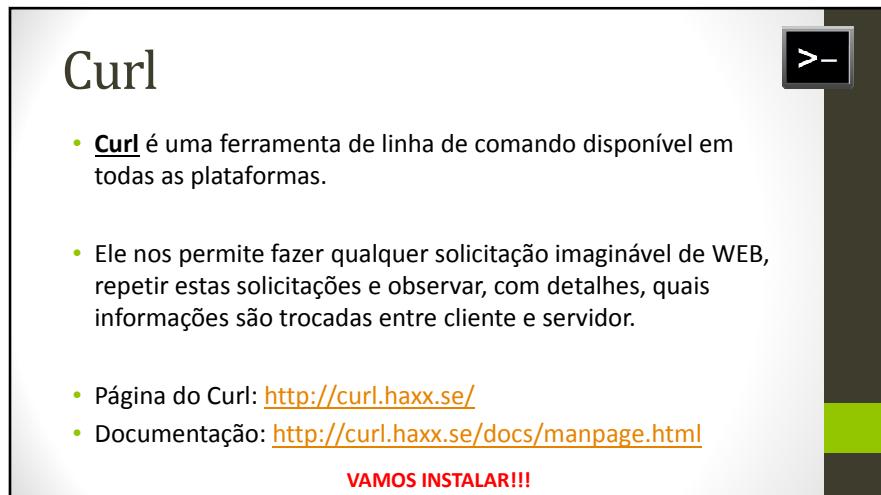
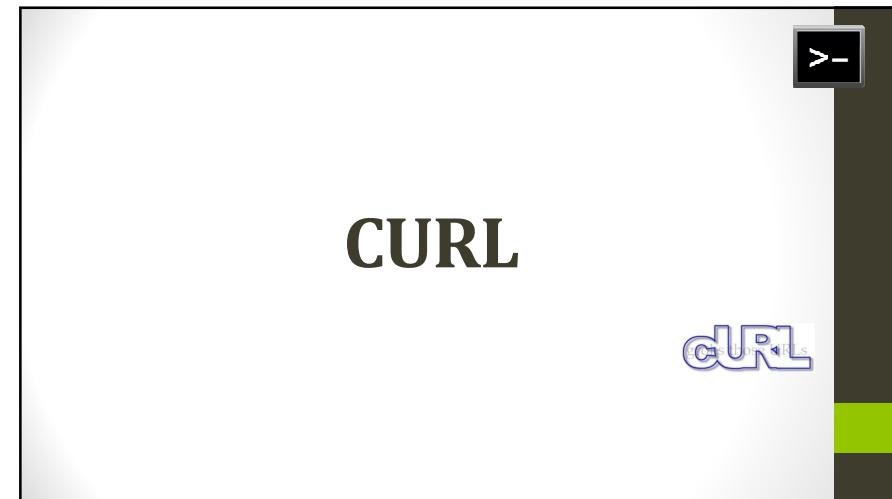
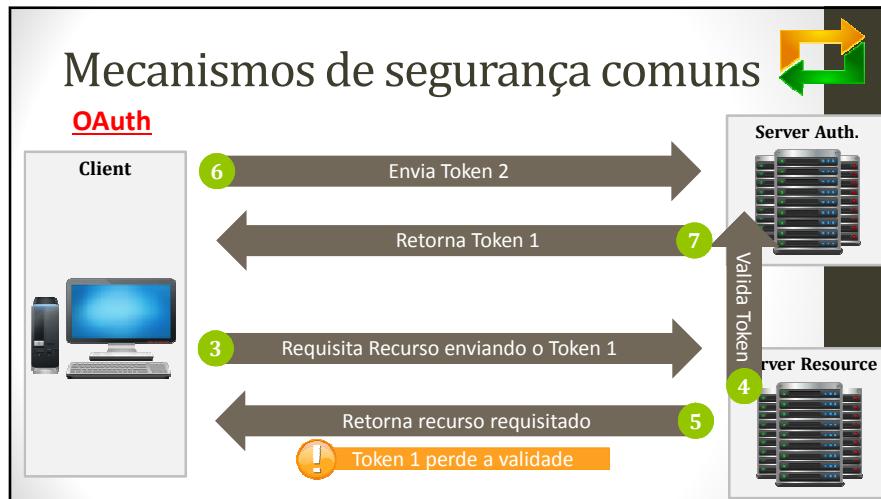


Mecanismos de segurança comuns



OAuth

- O OAuth é um padrão aberto para autorização.
- Ele provê meios de autorizar acesso de terceiros a seus recursos sem informar suas credenciais.
- Normalmente através de redirects e confirmações por parte dos usuários.
- Embora seja seguro, o OAuth não tem adesão tão grande devido à complexidade na implementação.



Curl

- Existe algumas opções úteis que podem ser usadas para obter mais informações do Curl, além do corpo da resposta.
- A opção `-v` mostrará tudo: cabeçalho da solicitação e cabeçalho de resposta e o corpo da resposta.
 - `curl http://dominio/example -v`
- A opção `-i` mostrará o cabeçalho de resposta da solicitação.
 - `curl http://dominio/example -i`

>-



Curl

- Também é possível enviar um JSON utilizando a opção `-d`
 - `curl -X POST -H "Content-Type: application/json" http://dominio/example -d "{\"nome\":\"Luiz Henrique\"}"`
- Para recuperar a informação no PHP é necessário utilizar o comando `file_get_contents("php://input")`:

```
$data = file_get_contents("php://input");
$data = json_decode($data);
print_r($data);
```

>-



Curl

- O modo mais simples para trabalhar com dados no Curl é enviar os dados juntamente com uma solicitação em pares chave e valor.
 - `curl -X POST http://dominio/example -d name="Luiz Henrique" -d email="luisdeangeli@gmail.com" -d message="Olá Mundo"`
- Utilizando GET direto na URL:
 - `curl -X GET http://dominio/example?name=Luiz&idade=15`

>-



Curl

- Trabalhar com os recursos estendidos do HTTP exige a capacidade de trabalhar com vários cabeçalhos.
- O Curl permite enviar qualquer cabeçalhos desejado usando a opção `-H`.
 - `curl -H "Accept: text/html" http://dominio/example`

>-



Curl



- Outros exemplos de API RESTful:
- [Twitter](https://dev.twitter.com/rest/public): <https://dev.twitter.com/rest/public>
- [Google Translate](https://cloud.google.com/translate/v2/using_rest): https://cloud.google.com/translate/v2/using_rest
- [Github](https://developer.github.com/v3/): <https://developer.github.com/v3/>
- [Instagram](http://instagram.com/developer/api-console/): <http://instagram.com/developer/api-console/>
- [Youtube](https://developers.google.com/youtube/v3/sample_requests): https://developers.google.com/youtube/v3/sample_requests



Cabeçalhos em PHP

Cabeçalho em PHP



- O PHP permite customizar o cabeçalho de resposta para solicitação através da função [header](#).
- Através desta função é possível enviar o código de status da requisição, o Content-Type, o charset utilizado, etc.
- **Observação:** A função header deve ser executada antes de qualquer saída de conteúdo (html, json, xml, etc.).
- Documentação: http://php.net/manual/pt_BR/function.header.php
- Lista de possíveis cabeçalhos: http://en.wikipedia.org/wiki/List_of_HTTP_header_fields

Cabeçalho em PHP



- Vamos utilizar o PHP para criar alguns cabeçalhos e verificar o resultado da saída utilizando o Curl.

```
<?php
    header ("HTTP/1.1 404 Not Found");
    header ("HTTP/1.1 403 Forbidden");
    header ("HTTP/1.1 400 Bad Request");
    header ("HTTP/1.1 401 Unauthorized");
    header ("HTTP/1.1 302 Found");
    header ("HTTP/1.1 200 OK");

    header ("App-Token: zgaGuYXU9aoC");
    header ("Content-Type: application/json; charset=utf-8");
    exit();
?>
```



Utilizando a biblioteca Curl do PHP



curl

Exemplo do Curl do PHP utilizando o GET

```
$url = "http://api.transparencia.org.br/api/v1/cargos/";
$ch = curl_init($url);

curl_setopt($ch, CURLOPT_HTTPHEADER, array(
    "App-Token: zgaGuYXU9aoC",
    "Content-Type: application/json",
    "Accept: application/json"
));

curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
$resultado = curl_exec($ch);
curl_close($ch);

echo $resultado;
```



curl

- O PHP suporta libcurl, uma biblioteca criada por Daniel Stenberg, que permite que você conecte-se e comunique-se com diferentes tipos de servidor usando diferentes tipos de protocolos
- O libcurl atualmente suporta os protocolos http, https, ftp, gopher, telnet, dict, file, e ldap. libcurl também suporta certificados HTTPS, HTTP POST, HTTP PUT, upload via FTP, upload HTTP por formulário, proxies, cookies, e autenticação com usuário e senha.
- Você pode utilizar esta biblioteca para acessar APIs RESTful externas através do PHP.
- http://php.net/manual/pt_BR/book.curl.php

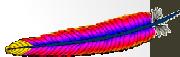


Configurando o Servidor APACHE



mod_rewrite

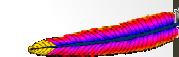
- Para trabalharmos com alguns frameworks é necessário ativar a regra **mod_rewrite** do apache.
- Este módulo utiliza um motor de reescrita baseada em regras, com base em um analisador de expressão regular.
- Permite:**
 - reescrever as regras;
 - usar expressões regulares para analisar a URI solicitada;
 - direcionar para uma URI diferente antes de interpretação.



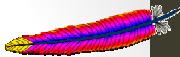
mod_rewrite

- Para saber se o módulo está instalado no servidor, podemos utilizar o **phpinfo()** e verificar se o **mod_rewrite** está carregado na seção **Loaded Modules**.

| | |
|------------------------|--|
| Apache Version | Apache/2.2.14 (Ubuntu) |
| Apache API Version | 20051115 |
| Server Administrator | webmaster@localhost |
| Hostname:Port | lucid64.hsd1.ca.comcast.net:80 |
| User/Group | www-data/www-data |
| Max Requests Per Child | 0 - Keep Alive: on - Max Per Connection: 100 |
| Timeouts | Connection: 300 - Keep-Alive: 15 |
| Virtual Server | Yes |
| Server Root | /etc/apache2 |
| Loaded Modules | core mod_log_config mod_logio prefork http_core mod_so mod_alias mod_authn basic mod_authn_file mod_authz_default mod_authz_groupfile mod_authz_host mod_authz_user mod_autoindex mod_cgi mod_deflate mod_dir mod_env mod_mime mod_negotiation mod_php5 mod_reqtimeout mod_rewrite mod_setenvif mod_status |



mod_rewrite



- Para ativar a regra **mod_rewrite** do apache acesse o arquivo **httpd.conf** ou no arquivo de configuração do apache similar.
- Localize e remova o caractere de comentário (#) da linha **LoadModule rewrite_module modules/mod_rewrite.so** ou utilize o designe da ferramenta de servidor caso exista.

htaccess

- O que é o htaccess?**
- htaccess (hypertext access)** é um arquivo de orientação/leitura utilizado pelo servidor Apache nas hospedagens.
- Com o htaccess você pode aplicar mudanças específicas em diretórios distintos de parâmetros como por exemplo register_globals, magic_quotes_gpc, include path e muitos outros.
- Normalmente o arquivo **htaccess** é inserido de modo oculto nos diretórios com o acréscimo de um ponto "." a frente do seu nome.
Ex.: **.htaccess**



htaccess

- O que podemos escrever em um arquivo *htaccess*?
- A documentação do htaccess esta disponível no site do apache em <http://httpd.apache.org/docs/2.2/pt-br/howto/htaccess.html>

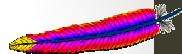


htaccess

- Vamos realizar alguns testes!
- ```
#Exemplo 4 : Liberando somente para um IP específico
ErrorDocument 403 http://www.yahoo.com/
Order deny,allow
Deny from all
Allow from 10.0.2.3

#Exemplo 5 : Redirecionamento
Redirect teste.html http://www.example.com/new-articles.html

#Exemplo 6 : Bloqueando a listagem de pastas
Options -Indexes
```



# htaccess

- Vamos realizar alguns testes!

```
#Exemplo 1 : Customizar o Erro 404
ErrorDocument 404 "<p>Error 404</p><p> Página Não encontrada.</p>"
```

```
#Exemplo 2 : Customizar o Erro 404 em um arquivo
ErrorDocument 404 /WEBDEV/htaccess/404.html
```

```
#Exemplo 3 : Bloquear um IP de acessar o servidor
order allow,deny
deny from 10.0.2.2
allow from all
```



# htaccess

- Vamos realizar alguns testes!

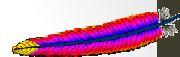
```
#Exemplo : Criando login e senha para acesso ao sistema
#Gerar Login e Senha: http://www.htaccesstools.com/htpasswd-generator/
AuthType Basic
AuthName "My Protected Area"
AuthUserFile .htpasswd
Require valid-user
```



```
admin:$apr1$0jzhOLYC$.NUMRdqNcdx/P22sFqC7o1
```



## htaccess



- Alguns exemplos de códigos para *htaccess* disponíveis na web.
  - <http://blog.dreamhosters.com/kbase/index.cgi?area=3083>
  - <http://www.askapache.com/htaccess/htaccess.html>
  - <http://www.htaccesstools.com/>

## htaccess



- Existem duas razões principais para **evitar** o uso de arquivos .htaccess.
  1. A primeira delas é a performance. Quando **AllowOverride** é configurado para permitir o uso de arquivos .htaccess, o Apache procura em todos diretórios por arquivos .htaccess.
  2. A segunda consideração é relativa à segurança. Você está permitindo que os usuários modifiquem as configurações do servidor, o que pode resultar em mudanças que podem fugir ao seu controle.

## htaccess



- Quando **não** usar arquivos .htaccess
  - No geral, você nunca deve usar arquivos .htaccess a não ser que você não tenha acesso ao arquivo de configuração principal do servidor. Ex: Provedores de site.
  - De modo geral, o uso de arquivos .htaccess deve ser evitado quando possível. Quaisquer configurações que você considerar acrescentar em um arquivo .htaccess, podem ser efetivamente colocadas em uma seção **<Directory>** no arquivo principal de configuração de seu servidor.

## PSR



## PSR



- Há muitas maneiras de se programar, cada um pode ter técnicas e preferências diferentes ao codificar;
- Por exemplo você pode achar que é melhor usar uma identação com tab size = 4, já outro programador poderia achar melhor tab size = 2, e outro ainda poderia nem usar tab e gostar mesmo é dos espaços.
- Programadores que seguem seu próprio padrão ou nenhum padrão, podem fazer coisas de maneira que não seja a melhor para outros.

## PSR



- Quem cuida disso é o [PHP-FIG](#) (Framework Interop Group).
- O FIG é um grupo composto por representantes de grandes projetos em PHP, tais como como o CakePHP, Doctrine, Symfony 2, Drupal e Zend Framework 2.
  - <http://www.php-fig.org/members/>
- Este grupo busca criar padrões que todos esses projetos possam seguir, definindo assim um “formato global” para projetos PHP

## PSR



- Se você desenvolvedor PHP há algum tempo, certamente já teve contato com projetos escritos em todo e qualquer tipo de estrutura
- Isso faz com que os desenvolvedores percam muito do seu precioso tempo tentando do projeto em questão.
- E se existisse uma série de regras globais para todos os projetos PHP seguirem, a fim de se obter uma estrutura padrão de diretórios, carregamento automático de classes e até da formatação do código?

## PSR



- Esses padrões são criados com foco nos projetos participantes, mas podem ser facilmente implementados em qualquer projeto que precisemos desenvolver.
- E o que ganhamos com isso é a manutenibilidade, ou seja, a facilidade que teremos para realizar processos de manutenção nesses projetos.

## PSR

- Esses padrões são chamados de PSR - PHP Standard Recommendation (ou padrão recomendado para PHP).
- Cada um deles especifica um conjunto de regras diferentes, entretanto, eles podem ser utilizados em conjunto, uma vez que cada conjunto de regras define uma prática específica.
- <http://www.php-fig.org/psr/>



## Porque utilizar PSR's?

- As PSR's não são uma imposição de como você deve programar, são recomendações, você não é obrigado a utilizá-las, no entanto considere alguns motivos para conhecê-las e adotá-las:
- Se você desenvolve algo que outros programadores terão acesso, com certeza eles preferem algo que seja fácil de ler e trabalhar.
- Se você pretende utilizar frameworks/projetos como os já mencionados logo acima, eles e vários outros implementam PSR.



## PSR – Status / Estágios

Cada PSR possui ou passa por um dos seguintes status/estágios:

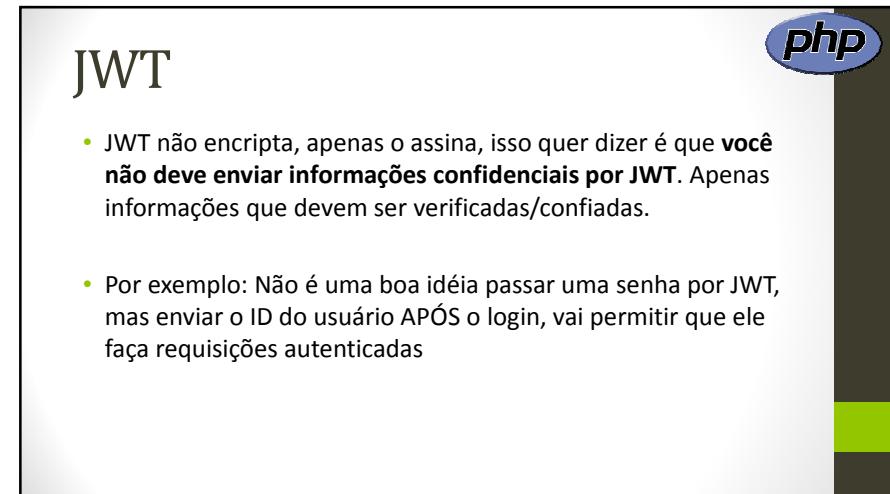
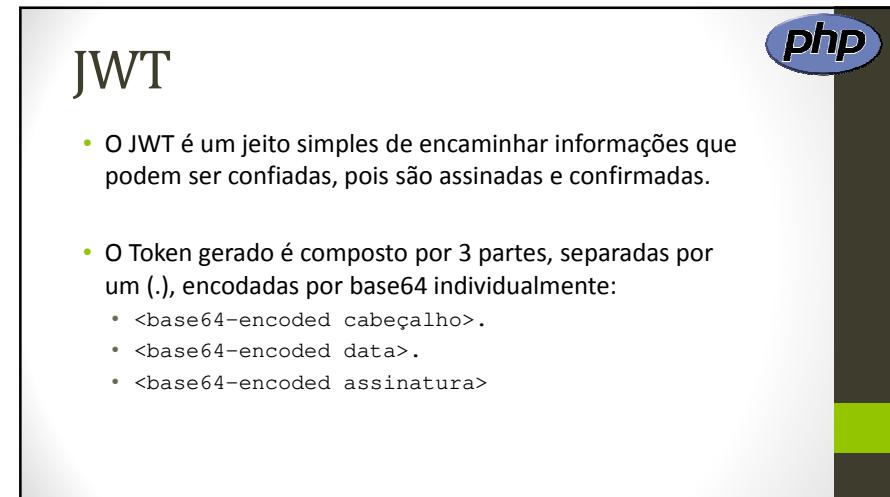
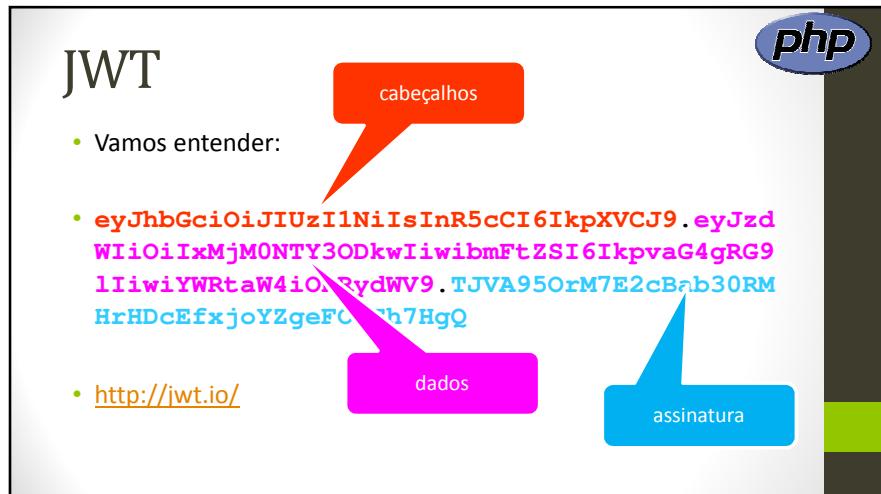
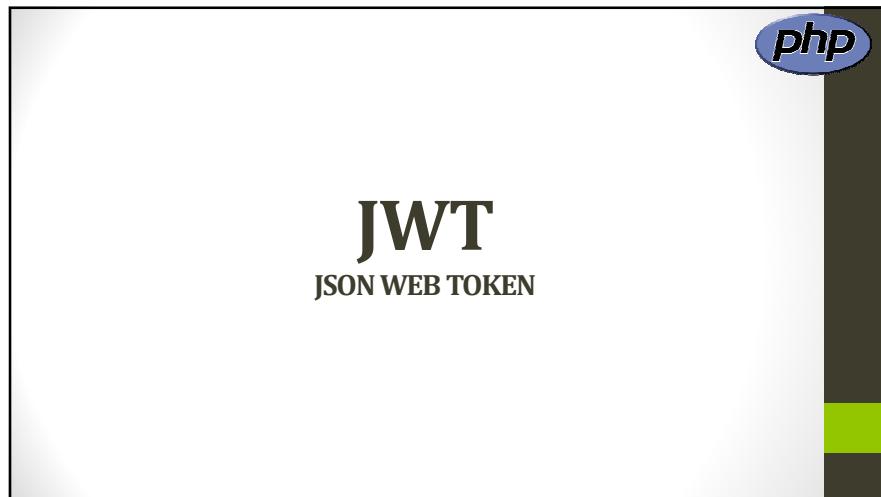
| STATUS     | Descrição                                                                                                                                                                                                            |
|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DEPRECATED | Era uma recomendação mas por estar obsoleta não é mais recomendada, isso acontece porque depois de aceita não pode ser alterada, mas pode ser substituída por outra.                                                 |
| ACCEPTED   | Uma PSR oficial, aceita, aprovada e recomendada.                                                                                                                                                                     |
| REVIEW     | Se não voltar para o DRAFT, fica aqui pelo menos 2 semanas para os membros se familiarizarem, se concordarem que a proposta está pronta para se tornar uma PSR é realizada uma votação de aceitação.                 |
| DRAFT      | Aqui estão discutindo e trabalhando em uma proposta. Abordagens alternativas podem ser sugeridas e discutidas a qualquer momento.                                                                                    |
| PRE-DRAFT  | Nessa fase o objetivo é determinar se a maioria do PHP-FIG está interessada em publicar uma PSR para um conceito proposto. Os interessados podem discutir uma possível proposta, incluindo possíveis implementações. |



## Porque utilizar PSR's?

- Se você não trabalha em equipe ou trabalha em um grupo pequeno ou é um programador independente, pode achar que não faz sentido utilizá-las, mas se algum dia você chegar a trabalhar com equipes grandes, essas recomendações podem ser algo comum entre os programadores PHP e conhecê-las o pode tornar mais aceito.
- Imagina você em uma entrevista de emprego onde o entrevistador faz perguntas sobre PSR e você não conhece nem as principais.
- Muitas pessoas já utilizam essas recomendações.





## JWT

- Instalação utilizando o composer:
- <https://packagist.org/packages/firebase/php-jwt>

```
{
 "require": {
 "firebase/php-jwt": "*"
 }
}
```



## JWT

Vamos criar um exemplo de validação:

- gerar.php
- validar.php



## POSTMAN



## POSTMAN

Download:

<https://app.getpostman.com/app/download/win64>



# Framework SLIM



## Slim

- Existem dezenas de frameworks que implementam a arquitetura REST
- Vamos começar com o **Slim Framework**, que é bastante leve e prático, possuindo como principal característica a implementação RESTful.
- O **Slim** é uma microestrutura que permite que os desenvolvedores escrevam rapidamente aplicações web RESTful e APIs.

## Slim



- O Slim é um framework PHP leve e ágil usada para construir aplicações web menores.
- A ideia dele é fornecer apenas a parte de rotas e views simplificadas, sem se preocupar com a parte de modelos e coisas mais complexas.
- Suporte ao PSR-7 de interfaces para **Requisição** e **Resposta**.
  - <http://www.slimframework.com/docs/concepts/value-objects.html>
  - <http://www.php-fig.org/psr/psr-7/>

## Slim



## Slim

- Então vamos instalar o Slim e realizar alguns testes do funcionamento do framework.
- Vamos utilizar o composer para instalar o Slim.

**Slim**  
a micro framework for PHP

Slim is a PHP micro framework that helps you quickly write simple yet powerful web applications and APIs.

```
use Psr\Http\Message\ServerRequestInterface as Request;
use Psr\Http\Message\ResponseInterface as Response;
require __DIR__ . '/vendor/autoload.php';

$app = new \Slim\App();
$app->get('/hello/{name}', function (Request $request, Response $response) {
 $name = $request->getAttribute('name');
 $response->getBody()->write("Hello, $name");
 return $response;
});
$app->run();
```

<https://packagist.org/packages/slim/slim>

# Slim

No arquivo *index.php* vamos programar o seguinte código:

```
<?php

require_once '../vendor/autoload.php';

use Psr\Http\Message\RequestInterface as Request;
use Psr\Http\Message\ResponseInterface as Response;

$app = new \Slim\App();

//nossas rotas serão configuradas nesta área!

$app->run();
```



## Exercício de Fixação



- Vamos praticar um pouco!**

- Criar 4 rotas **GET** que recebam dois números como parâmetro na URI e apresente o resultado da operação destes números; Ex: /soma/2/2, /subtracao/2/2
- Criar uma rota **POST** que receba 3 parâmetro (um número, o operador, outro número). Apresente o resultado do calculo do operador enviado. Apresente erro 404 caso os dados não sejam enviados corretamente. Ex: /calculadora/ , ("10", "1", "5").

# Slim



- Rotas**

- Parâmetro Obrigatório
- Grupos
- Parâmetro Opcional
- Parâmetro com ER

- Request**

- Middleware
- Error Handler
- JWT

- Response**

- Cabeçalhos
- Código de Status
- Trabalhando com JSON

## Exercício de Fixação



- Vamos praticar um pouco!**

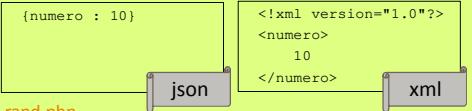
- Crie um classe **CalculadoraController** que contenha os métodos das 4 operações básica da matemática:
  - Soma()
  - Substracao()
  - Multiplicacao()
  - Divisao()
 Cada função deve retornar o resultado do calculo para a rota do *Slim* que apresentará em formato JSON.

## Exercício de Fixação

**Vamos praticar um pouco!**

- Criar uma rota GET que receba uma cabeçalho “Token” que deve ser igual XYZ123, caso não seja apresente erro 403, receba também o cabeçalho “Accept” com as seguintes possibilidades “application/json” ou “application/xml”.

Faça a programação para gerar um número aleatório de 0 a 100 utilizando o random e retornar o número no formato desejado (json ou xml). Caso o “Accept” enviar seja diferente dos dois aceito apresente erro 404.



http://php.net/manual/pt\_BR/function.rand.php

## Slim

- Exemplo simples de cliente com Mysql**
  - Exemplo de cada rota: GET, POST, PUT, DELETE



## Exercício de Fixação

Utilizando o exemplo de Cliente com Mysql construa a aplicação RESTful:

| Base                | Objeto                              | Rotas               |
|---------------------|-------------------------------------|---------------------|
| Cargo               | {idCargo, nome}                     | CRUD Completo       |
| Colaborador         | {idColaborador, nome, cpf, idCargo} | CRUD Completo       |
| Despesa Colaborador | {id, valor, idColaborador, data}    | Inserção e Exclusão |

## Aula 2

- Criar uma aplicação utilizando o Slim.



# Slim

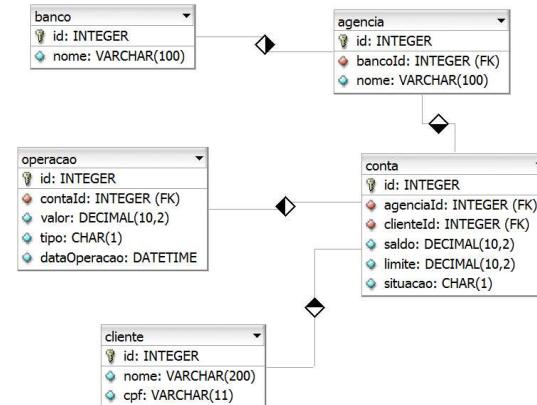
Vamos começar!

1. Utilizando o Doctrine e o Framework Slim, criar as rotas para manter os CRUDs de:
  - Cliente, Banco, Agência e Conta
2. Criar a rota de operações de Crédito e Débito;
  - Ao realizar uma inserção na tabela de operação a trigger vai atualizar o saldo e também bloquear saque quando não existir limite.



# Slim

- DER



# Memcached



A ferramenta de software livre *memcached* é um cache para armazenar informações frequentemente usadas para evitar o carregamento (e processamento) de informações de origens mais lentas, como discos ou um banco de dados.

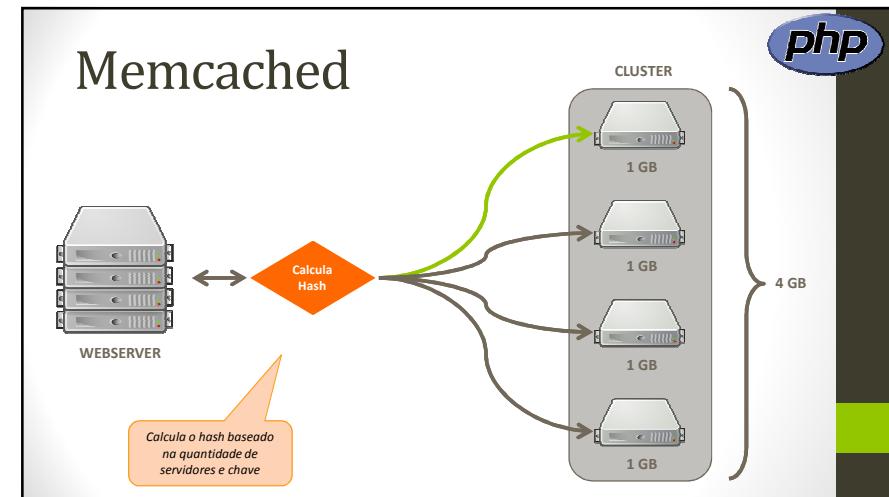
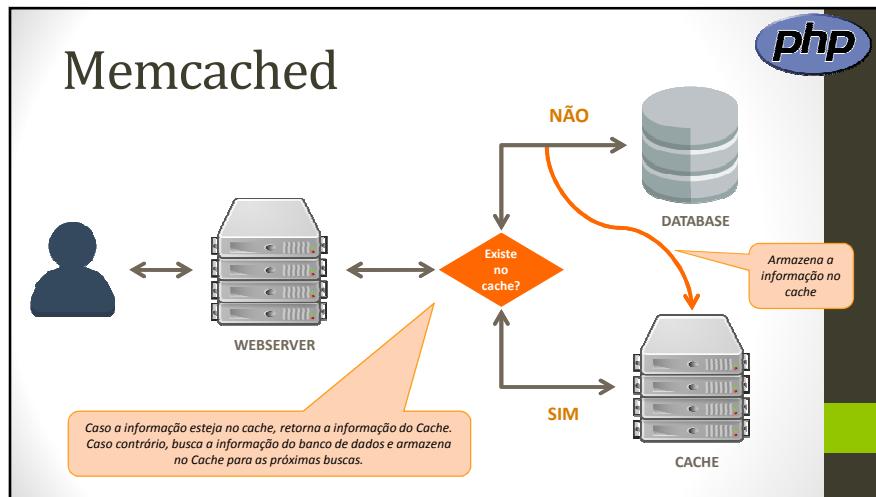
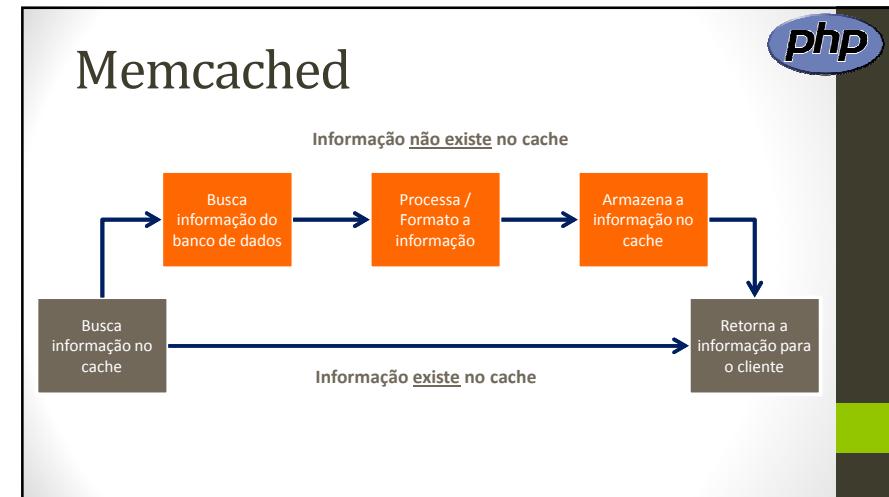
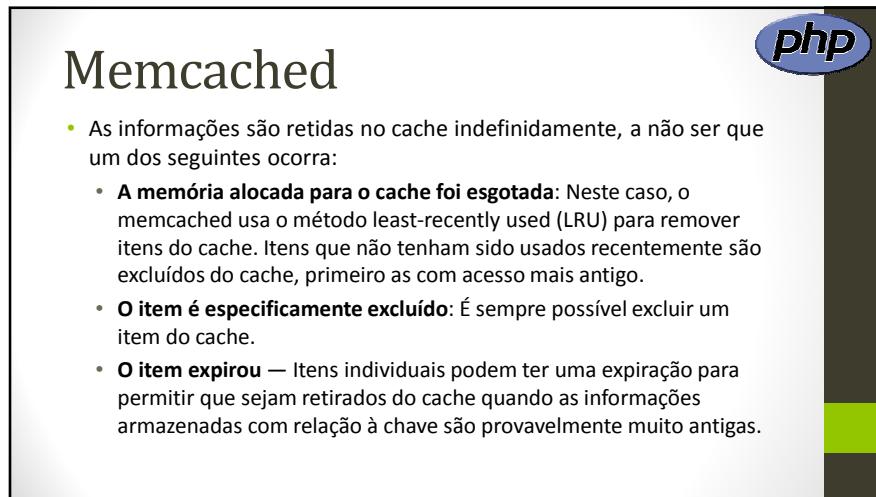
Ela pode ser implementada em uma situação dedicada ou como um método para usar memória sobressalente em um ambiente existente

# Memcached



O método de armazenamento com o memcached é um par simples de **palavra-chave/valor**, similar ao hash ou matriz associativa disponível em várias linguagens.

As informações são **armazenadas** no memcached fornecendo a chave e o valor, e **recuperadas** solicitando as informações pela chave especificada.



## Memcached

Leitura do texto da IBM:

<https://www.ibm.com/developerworks/bropensource/library/os-memcached/>



## Memcached

4) Baixar e instalar o memcache para Windows:

- 32 bits: <http://downloads.northscale.com/memcached-win32-1.4.4-14.zip>
- 64 bits: <http://downloads.northscale.com/memcached-win64-1.4.4-14.zip>

Comandos do Memcache:

```
memcached.exe -d install
memcached.exe -d start
memcached.exe -d stop
```

5) Baixar o phpmemcacheadmin e colocar na pasta o htdocs do xamp

<https://code.google.com/archive/p/phpmemcacheadmin/downloads>



## Memcached

1) Realizar o download Biblioteca Memcache para PHP:

<http://windows.php.net/downloads/pecl/releases/memcache/3.0.8/>

- 32 bits: [php\\_memcache-3.0.8-5.6-ts-vc11-x86.zip](php_memcache-3.0.8-5.6-ts-vc11-x86.zip)
- 64 bits: [php\\_memcache-3.0.8-5.6-ts-vc11-x64.zip](php_memcache-3.0.8-5.6-ts-vc11-x64.zip)

2) Copiar o arquivo **php\_memcache.dll** para a pasta: C:\tools\xampp\php\ext

3) Adicionar o Memcached na biblioteca no php.ini:

*Copiar o texto abaixo e inserir no final do arquivo do php.ini e reiniciar o apache.*

```
extension=php_memcache.dll
[Memcache]
memcache.allow_failover = 1
memcache.max_failover_attempts=20
memcache.chunk_size=8192
memcache.default_port = 11211
```



## Memcached

Exemplo de como **registrar** a informação no cache

```
<?php

$m = new Memcache();
$m->addServer('localhost', 11211);

$m->set('int', 99, MEMCACHE_COMPRESSED, 10);
$m->set('string', 'a simple string');
$m->set('array', array(11, 12));
```



## Memcached



Exemplo de como **recuperar** a informação do cache

```
<?php

$m = new Memcache();
$m->addServer('localhost', 11211);

var_dump($m->get('int'));
var_dump($m->get('string'));
var_dump($m->get('array'));
```



## Documentação



...

## Aula 3



- Implementando o front-end utilizando Angular.

## Aplicando padrões e implementando o front-end



# AngularJS

<https://angularjs.org>

<https://docs.angularjs.org>



## O que é AngularJS?

- Este framework é mantido pelo [Google](#).
- Possui algumas particularidades interessantes, que o fazem um framework muito poderoso.
- Uma dessas particularidades é que ele funciona como uma extensão ao documento HTML, adicionando novos parâmetros e interagindo de forma dinâmica com vários elementos.



## Instalando o AngularJS?

- É preciso apenas duas alterações na estrutura de um documento HTML para que possamos ter o AngularJS instalado.
  - A primeira, é incluir a biblioteca javascript no cabeçalho do documento.
  - A segunda, é incluir a propriedade ng-app no elemento html em que queremos “ativar” o angularJS.

```
<html ng-app>
 <head>
 <script src="angular.min.js"></script>
 </head>
 <body></body>
</html>
```



## Principais características

### DataBind

- Uma das principais vantagens do AngularJS é o seu DataBind.
- Este termo é compreendido como uma forma de ligar automaticamente uma variável qualquer a uma outra.
- Geralmente, o DataBind é usado para ligar uma variável do JavaScript (ou um objeto) a algum elemento do documento HTML.

# Principais características



## DataBind

```
<html ng-app>
 <head>
 <script src="angular.min.js"></script>
 </head>
 <body>
 Olá <input type="text" ng-model="meuNome"/>
 <hr/>
 <h1>Olá {{meuNome}} </h1>
 </body>
</html>
```

Utilizamos para *DataBind* a propriedade **ng-model**, para informar que este elemento estará ligado a uma variável do AngularJS

# Principais características



## Controller

```
<html ng-app="app">
 <head>
 <script src="angular.min.js"></script>
 <script src="simplesController.js"></script>
 </head>
 <body ng-controller="simplesController">
 Olá <input type="text" ng-model="usuario.nome"/>
 <hr/>
 <h1>Olá {{usuario.nome}}</h1>
 </body>
</html>
```

Usamos a propriedade **ng-controller** para dizer que, todo elemento abaixo do será gerenciado pelo controller.

# Principais características



## Controller

- Um **controller** é, na maioria das vezes, um arquivo JavaScript que contém funcionalidades pertinentes à alguma parte do documento HTML.
- Não existe uma regra para o **controller**, como por exemplo ter um **controller** por arquivo HTML, mas sim uma forma de sintetizar as regras de negócio (funções javascript) em um lugar separado ao documento HTML.

# Principais características



## Controller

```
var app = angular.module('app', []);
app.controller('simplesController', function($scope) {
 $scope.usuario = {nome: "Daniel"};
});
```

simpleController.js

O arquivo **simpleController.js** contém a criação da app e a indicação do controller, que é criado de forma modularizada

Neste exemplo estamos utilizando o **\$scope**, vamos falar sobre ele no próximo slide.

# Principais características



## Controller

- Neste controller, temos o parâmetro **\$scope** que é um “ponteiro” para a aplicação em si, ou seja, \$scope significa a própria página html.
- Como o *controller* foi declarado no elemento <body>, \$scope é usado para todo este elemento.
- Usa-se o \$scope para criar uma conexão entre o model e a view, como foi feito no exemplo utilizando o objeto user.

# Principais características



## Métodos do Controller

```
<script type="text/javascript">

 var app = angular.module('app', []);
 app.controller('contadorController', function($scope) {
 $scope.contador = 0;
 $scope.adicionarUm = function() {
 $scope.contador++;
 }
 });
</script>
```

# Principais características



# Principais características

## Métodos do Controller

O controller é usado também para manipular regras de negócio que podem ou não alterar os models.

```
<html ng-app="app">
 <head>
 <title>Hello Counter</title>
 <script src="angular.min.js"></script>
 </head>
 <body ng-controller="contadorController">
 Add 1
 <p>Valor do Contador: {{contador}}</p>
 </body>
</html>
```

# Principais características



# Principais características

## Loops

Outra característica do AngularJS é utilizar templates para que se possa adicionar conteúdo dinâmico.

```
<!DOCTYPE html>
<html ng-app="app">
 <head>
 <script src="angular.min.js"></script>
 </head>
 <body ng-controller="loopController">

 <li ng-repeat="fruta in frutas">{{fruta}}

 </body>
</html>
```

# Principais características



## Loops

```
<script type="text/javascript">

 var app = angular.module('app', []);
 app.controller('loopController', function ($scope) {
 $scope.frutas = ['banana', 'maça', 'laranja'];
 });

</script>
```



# Principais características



## Formulários

```
<form name="myForm">

 Found errors in the form!

 <input type="text" ng-model="name" name="Name" value="Your Name" required/>
 <button ng-disabled="myForm.$invalid"/>Save</button>
</form>
```

O uso do `myForm.$invalid` é um recurso do AngularJS que define se um formulário está inválido ou não.

# Principais características



## Formulários

- Existem diversas características que um formulário contém, tais como validação, mensagens de erro, formato dos campos, entre outros.
- Neste caso, usamos o AngularJS de diferentes formas, e usamos vários parâmetros `ng` para controlar todo o processo.

# Principais características

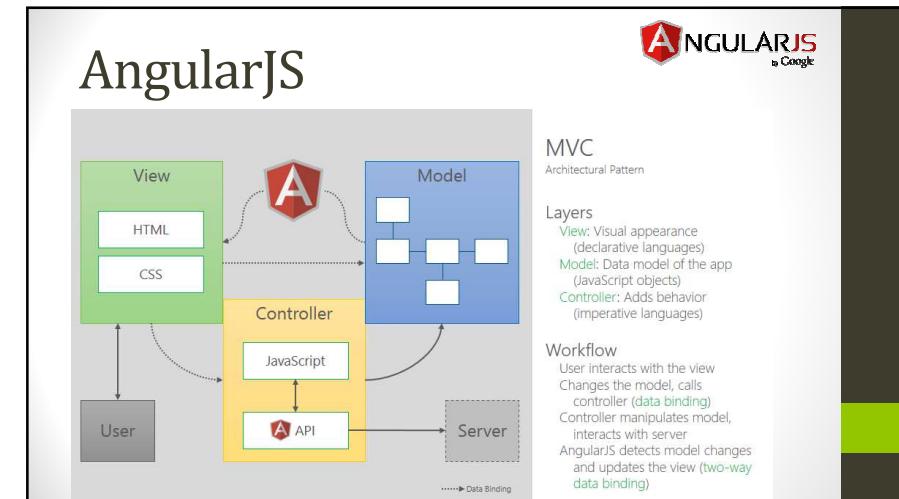
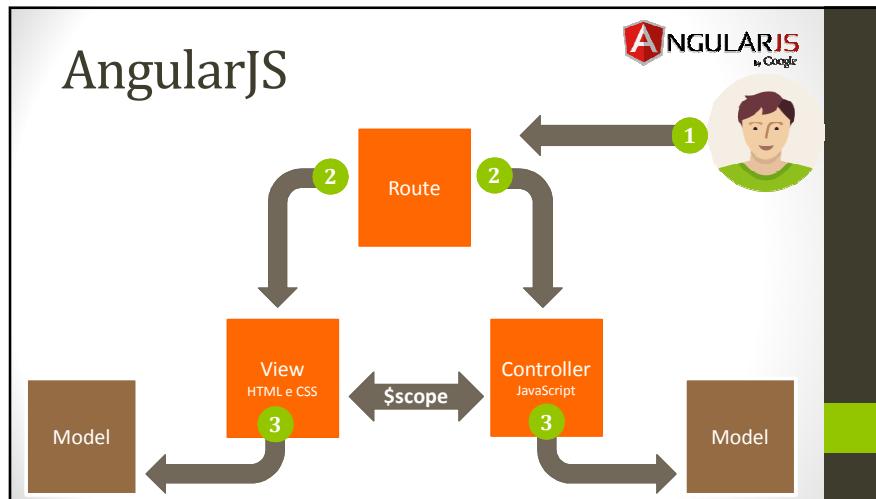


# Principais características



## Rotas

- O AngularJS possui um recurso chamado **Deep Linking**, que consiste em criar rotas na URI do documento HTML para manipular partes do código HTML de forma independente, podendo assim separar ainda mais as camadas da sua aplicação.
- O uso de DeepLinking usa Ajax para carregar templates de forma dinâmica, então é necessário que todo o exemplo seja testado em um servidor web.



## AngularJS

### MVC

```
$scope.model = {
 id : 1,
 nome : "Luiz",
 idade : 25
}
```

## AngularJS

### MVC

```

 <li ng-repeat="usuario in usuarios">
 {{usuario.nome}}


```

## AngularJS



### MVC

```
function controller($scope) {
 $scope.model = {...}
 $scope.click =
 function() {}
}
```

## Conectando AngularJS ao servidor



- O AngularJS fornece duas formas distintas de trabalhar com estas conexões.
  - A primeira delas, e mais simples, é através do serviço \$http, que pode ser injetado em um controller.
  - A segunda forma é através do serviço \$resource que é uma abstração RESTful, funcionando como um data source
- Em nossos exemplos vamos utilizar a primeira forma com o \$http.

## Conectando AngularJS ao servidor



- Assim como é feito com jQuery e javascript puro, a melhor forma de obter e enviar dados para o servidor é através de **Ajax** e o formato de dados para se usar nesta comunicação é **JSON**.
- Existem diversas formas de conexão entre cliente e servidor, e vamos utilizar um conceito chamado **RESTful**, que é uma comunicação **HTTP** que segue um padrão bastante simples, utilizando cabeçalhos **HTTP** como **POST, GET, PUT, DELETE**.

## Uso do \$http



- \$http é uma implementação ajax através do XMLHttpRequest utilizando JSONP. Iremos sempre usar JSON para troca de dados entre cliente e servidor.
- **Os métodos disponíveis são:**
  - \$http.get
  - \$http.head
  - \$http.post
  - \$http.put
  - \$http.delete
  - \$http.jsonp

[https://docs.angularjs.org/api/ng/service/\\$http](https://docs.angularjs.org/api/ng/service/$http)

## Uso do \$http



- Para todos estes métodos, o AngularJS configura automaticamente o cabeçalho da requisição HTTP.
- Por exemplo, em uma requisição **POST** os cabeçalhos preenchidos são:
  - Accept: application/json, text/plain
  - X-Requested-With: XMLHttpRequest
  - Content-Type: application/json
- Além dos cabeçalhos, o AngularJS também serializa o objeto JSON que é repassado entre as requisições.

## Uso do \$http



- A forma mais simples de uso do \$http é:

```
$http({method:'GET', url:'/someUrl'}).success(function(data) {
});
```

- O método get pode ser generalizado para:

```
$http.get('/someUrl').success(function(data) {
});
```

## AngularJS?



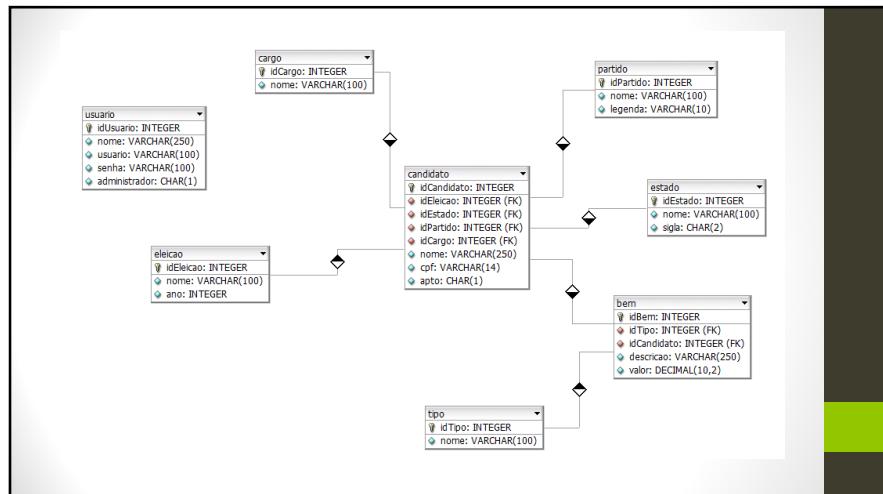
- Agora vamos criar uma aplicação com o AngularJS para consumir o back-end criando anteriormente.
- Vamos utilizar a pasta **angular**.

## Exercício



- Criar o cadastro de usuário**
  - Front-end e Back-end
  - Vincular o login ao usuário criado.
  - Usuário:
    - id, nome, usuário e senha

# Avaliação Final



## Avaliação Final - Sistema de eleições

- **Utilizando o Slim e Mysql**

- Criar as rotas para persistir as informações;
- Usuário administrador deve ter acesso total;
- Usuário que não seja administrador somente leitura;
- Criar o controle de Token utilizando o JWT;
- Busca de Candidato
  - Por Eleição, UF e Cargo: </candidato/2018/PR/1>
  - Por Eleição, UF e Partido: </candidato/2018/PR/5>
  - Por Eleição, UF : </candidato/2018/PR>

## Referências

- Web Services em PHP, Lorna Jane Mitchell, 2013
- Criando Sistemas com RESTful com PHP e Jquery, 2013
- RESTful Serviços WEB, Leonard Richard e Sam Ruby, 2007
- AngularJS na prática, Daniel Schmitz e Douglas Lira, 2014
- Links:
  - [https://netbeans.org/kb/docs/websvc/rest\\_pt\\_BR.html](https://netbeans.org/kb/docs/websvc/rest_pt_BR.html)
  - <http://arquitetura.takenet.com.br/blog/rest-e-hypermedia-apis>
  - [http://www.ics.uci.edu/~fielding/pubs/dissertation/rest\\_arch\\_style.htm](http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm)
  - <http://www.rodrigocalado.com.br/o-que-e-rest-um-resumo-do-assunto-caracteristicas-conceitos-vantagens-e-desvantagens-prefiro-dizer-que-e-uma-rapida-introducao-ao-assunto/>
  - <http://sidneylimafilho.com.br/post/rest-parte-1/>

?>

# Obrigado!!!



**Prof. Luiz Henrique de Angeli**  
luizdeangeli@gmail.com  
luiz.angeli@unicesumar.edu.br  
facebook.com/luizdeangeli