



UNIVERSIDADE
FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Criando uma API REST

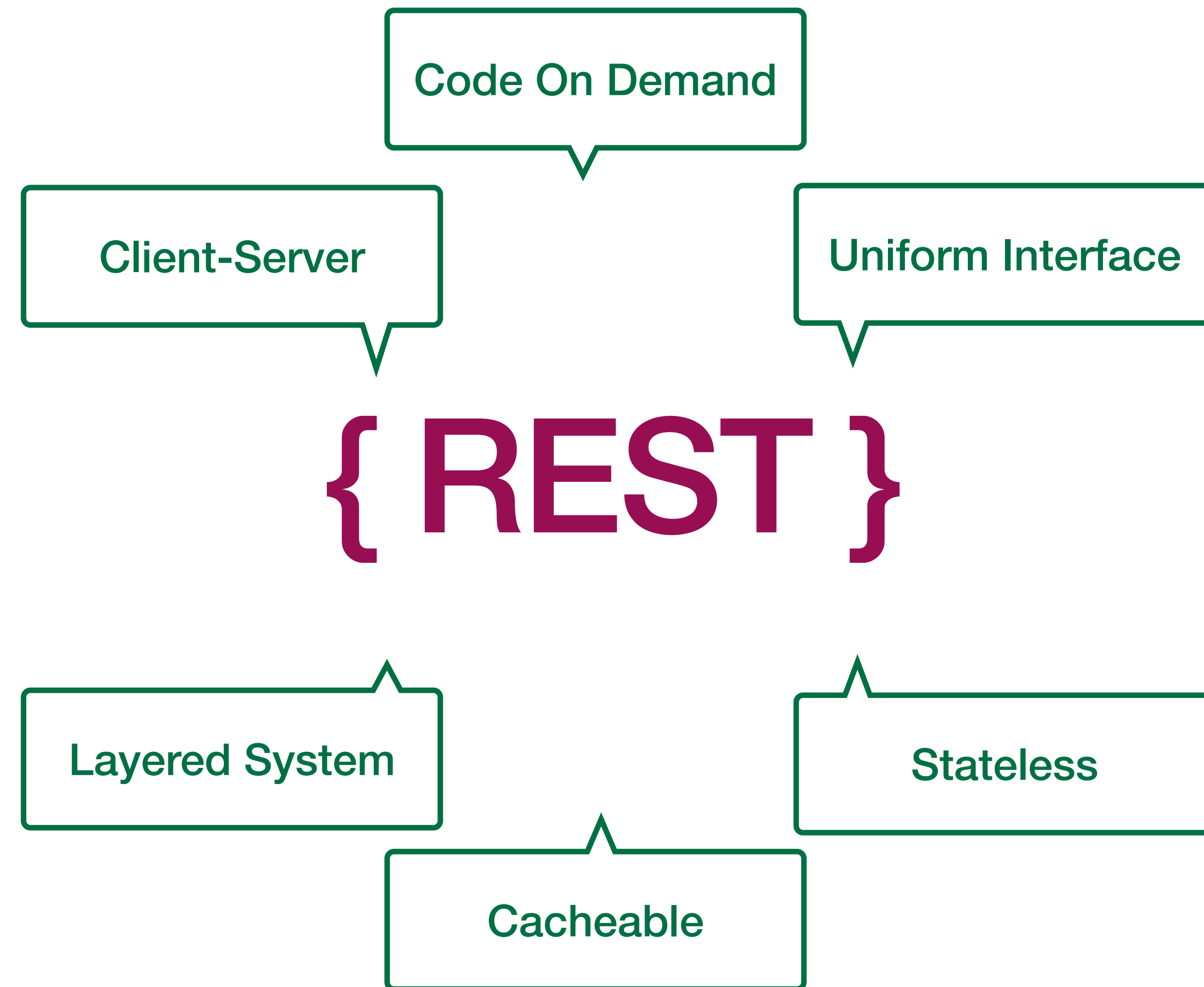
QXD0193 - Projetos de Interfaces Web

Prof. Bruno Góis Mateus (brunomateus@ufc.br)

Agenda

- Os princípios da arquitetura REST
- Projetando uma API REST
- Middlewares

Os princípios da arquitetura REST



Os princípios da arquitetura REST

Cliente-Servidor

- Trata da separação de responsabilidades
- **Cliente** e **Servidor** podem evoluir de forma independente
 - Independentemente das **tecnologias e linguagens utilizadas**
- O contrato entre eles se mantém intacto

Os princípios da arquitetura REST

Interface Uniforme (Uniform Interface)

- Toda comunicação entre **clientes** e **servidores** se dá por meio de interfaces
- Se algum componente não segue o padrão estabelecido a comunicação pode falhar
- Para estar conforme com este princípio 4 restrições devem ser seguidas
 - *Identificação de recursos*
 - *Manipulação de recursos através de representações*
 - *Mensagens auto descritivas*
 - *Hypermedia as the engine of application state (HATEOAs)*

Os princípios da arquitetura REST

Interface Uniforme (Uniform Interface)

- Identificação de recursos
 - Cada recurso distinto deve ser unicamente identificado por meio de uma URI
- Manipulação de recursos através de representações
 - Clientes manipulam representações de recursos que podem ser representados de forma diferentes. Ex: HTML, JSON, XML
 - O formato é apenas a forma de interação

Os princípios da arquitetura REST

Interface Uniforme (Uniform Interface)

- *Mensagens auto descritivas*
 - Cada representação de um recurso deve conter as informações necessárias para descrever como aquela mensagem deve ser processada
 - A mensagem também deve conter informações sobre as ações que os clients podem tomar em relação ao recurso desejado
- *Hypermedia as the engine of application state (HATEOAs)*
 - A representação de um estado de um recurso deve conter links para outros recursos relacionados
 - Desta forma, um cliente pode encontrar e navegar entre recursos

Os princípios da arquitetura REST

HATEOAs

```
GET /accounts/12345 HTTP/1.1  
Host: bank.example.com
```

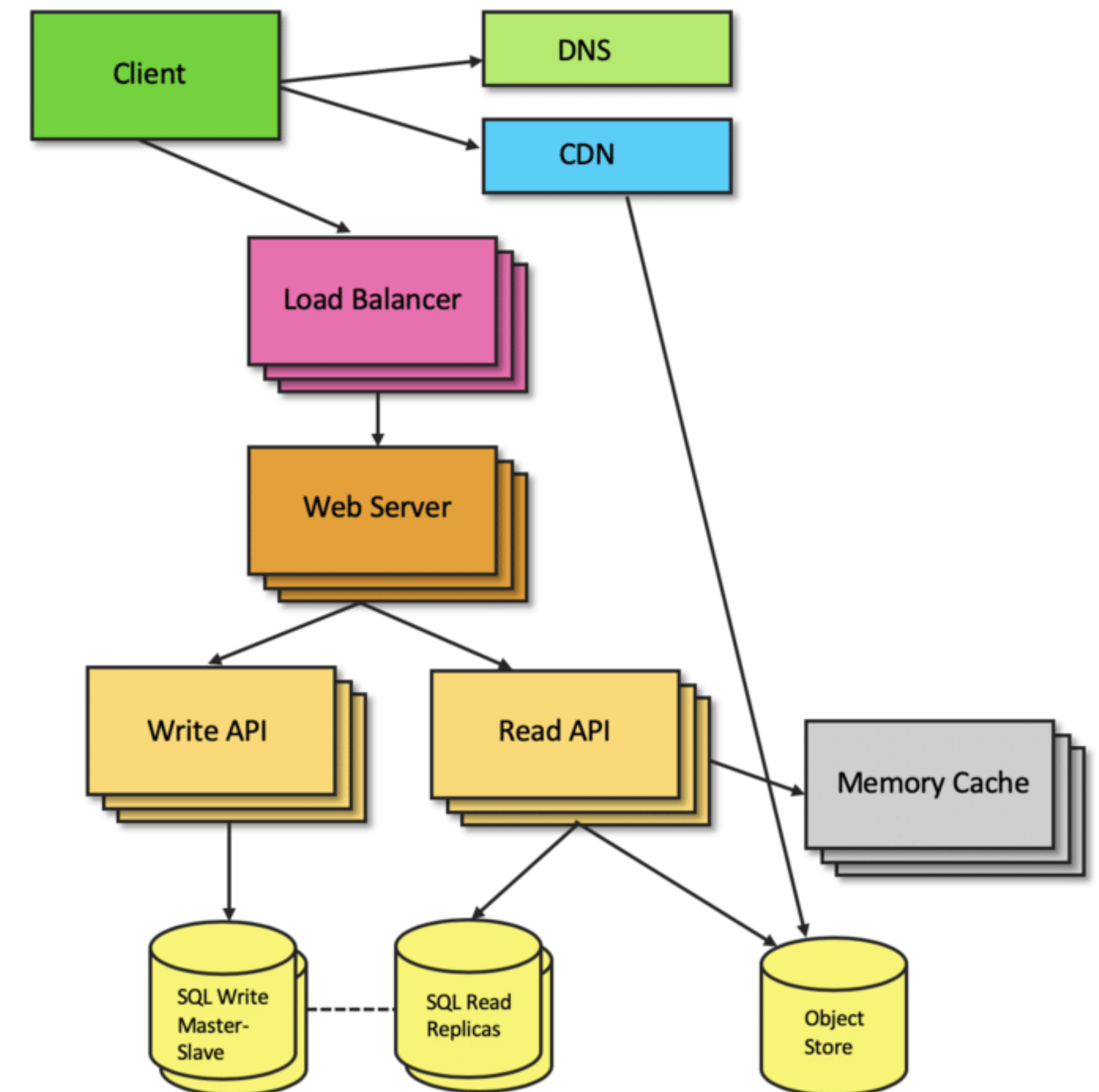
```
HTTP/1.1 200 OK
```

```
{  
  "account": {  
    "account_number": 12345,  
    "balance": {  
      "currency": "usd",  
      "value": 100.00  
    },  
    "links": {  
      "deposits": "/accounts/12345/deposits",  
      "withdrawals": "/accounts/12345/withdrawals",  
      "transfers": "/accounts/12345/transfers",  
      "close-requests": "/accounts/12345/close-requests"  
    }  
  }  
}
```


Os princípios da arquitetura REST

Sistema em camadas

- Não assuma que o cliente está se conectado diretamente ao servidor
- A API deve ser projetada de forma que nem o **cliente** nem o **servidor** saibam se eles estão se comunicando diretamente ou com um intermediário
- Ex: Múltiplas camadas de **servidores**.



Os princípios da arquitetura REST

Cache

- O servidor deve informar a *cacheability* dos dados de cada resposta
- O cache *pode existir em qualquer lugar da rede que liga o cliente ao servidor*:
 - Client side (navegador), Server side e Intermediary side (CDN)
- De forma geral reduz o custo da web (reduz o tráfego na rede)
 - Reduz a *latência* percebida pelo cliente
 - Aumenta a *disponibilidade e confiabilidade* da aplicação
 - Aumenta a *escalabilidade* da aplicação

Os princípios da arquitetura REST

Sem estados (*Stateless*)

- Toda requisição realizada deve conter toda a informação necessária para que ela seja entendida
 - O **servidor** não deve possuir conhecimento sobre requisições feitas previamente
 - O **servidor** não deve armazenar informações sobre as requisições
- A complexidade de gerir os estados deve ficar no **cliente**
- O **servidor** pode atender um número muito maior de **clientes**

Os princípios da arquitetura REST

Código sobre demanda (Opcional)

- Na maioria das vezes o **servidor** responde com **recursos estáticos**, no entanto, em certos casos o **servidor** deve poder incluir **código executável**
 - Java Applets
- No entanto, isso gera um acoplamento entre o cliente e o servidor
 - O cliente precisa entender o código enviado
 - Por essa razão este é o único opcional

Os princípios da arquitetura REST

REST vs RESTful

- Uma API Web em conformidade com a arquitetura REST é uma API RESTful
- APIs REST bem projetadas podem atrair desenvolvedores clientes para usar serviços da web
- No mercado aberto de hoje, onde os serviços web rivais competem por atenção, um design de API REST esteticamente agradável é obrigatório

Projetando uma API REST



Projetando uma API REST

O formato da URI

- A barra (/) deve ser utilizada para indicar hierarquia entre recursos

`http://api.canvas.restapi.org/shapes/polygons/quadrilaterals/squares`

- A barra (/) não dever ser utilizada no final das URIs

`http://api.canvas.restapi.org/shapes/`



`http://api.canvas.restapi.org/shapes`



Projetando uma API REST

O formato da URI

- A URI preferencialmente devem ser escritas em letras minúsculas

`http://api.example.restapi.org/my-folder/my-doc`

`HTTP://API.EXAMPLE.RESTAPI.ORG/my-folder/my-doc`

`http://api.example.restapi.org/My-Folder/my-doc`

São a URI

São URIs diferentes,
nos levam a recursos diferentes

- A extensão de arquivos não devem aparecer nas URIs

Projetando uma API REST

Arquétipos de Recursos (*Resource Archetypes*)

- Uma API REST possui 4 arquétipos de recursos
 - Documento (*Document*)
 - Coleção (*Collection*)
 - Loja (*Store*)
 - Controlador (*Controller*)

Projetando uma API REST

Documento

- Algo singular, como uma **instância de objeto** ou um **registro do banco de dados**

```
http://api.soccer.restapi.org/leagues/seattle
```

```
http://api.soccer.restapi.org/leagues/seattle/teams/trebuchet
```

```
http://api.soccer.restapi.org/leagues/seattle/teams/trebuchet/players/mike
```

Projetando uma API REST

Coleção

- Diretório ou coleção de recursos
 - Os clientes podem propor a adição de um novo recurso

```
http://api.soccer.restapi.org/leagues
```

```
http://api.soccer.restapi.org/leagues/seattle/teams
```

```
http://api.soccer.restapi.org/leagues/seattle/teams/trebuchet/players
```

Projetando uma API REST

Loja

- Uma **Store** nunca gera uma nova URI
 - Cada URI de **Store** é escolhida pelo cliente
 - Repositório escolhido gerenciado pelos clientes

```
PUT /users/1234/favorites/alonso
```

Projetando uma API REST

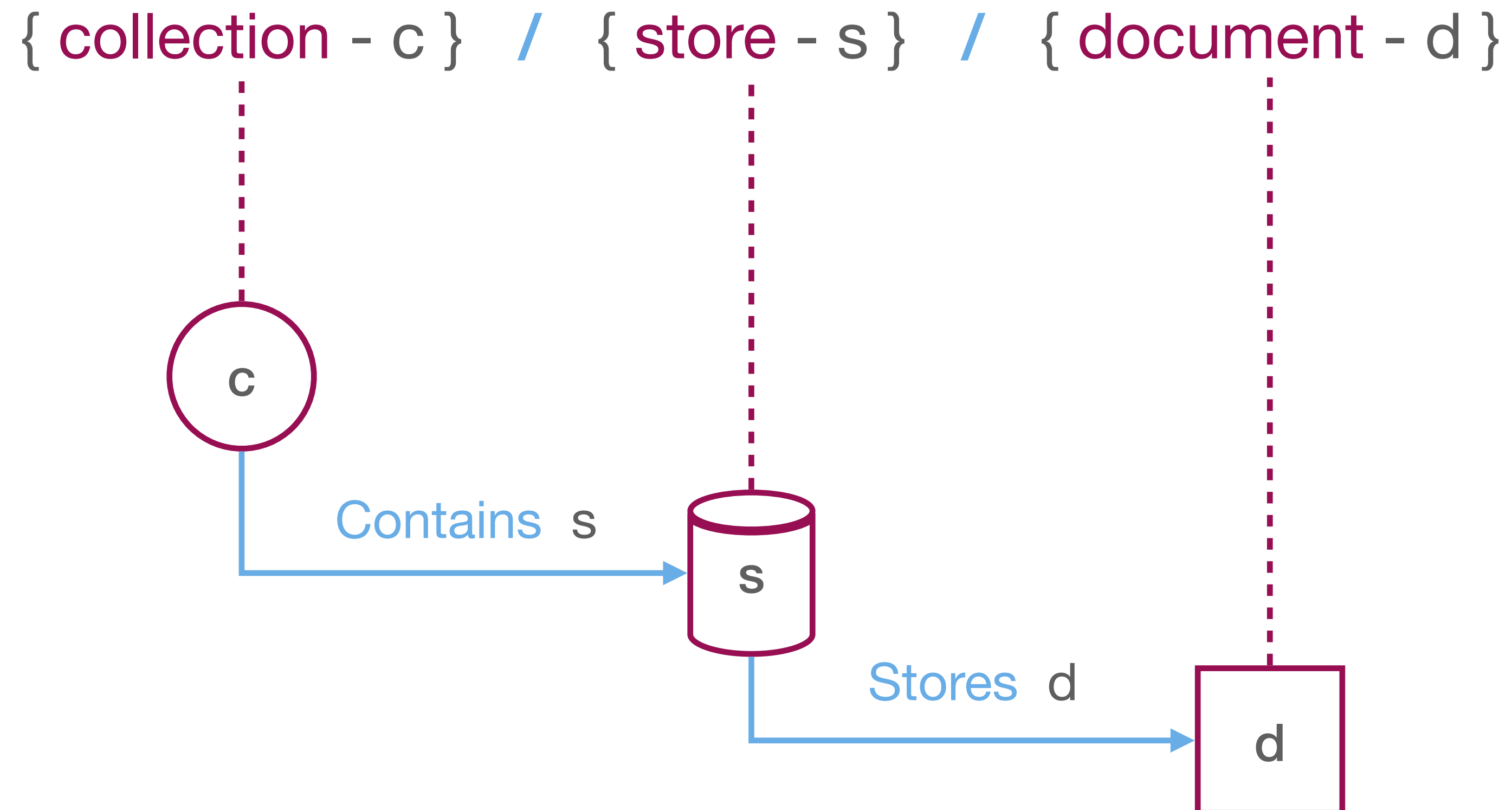
Controlador

- São similares a **funções executáveis**
 - Possuem parâmetros e retorno, i.e., entrada e saída
- Em geral o nome do controlador **faz parte do último segmento da URI**

```
POST /alerts/245743/resend
```

Projetando uma API REST

URI Path Design



Projetando uma API REST

URI Path Design

- Nomes no **singular** devem ser usados para nomear **documentos**
- Nomes no **plural** devem ser usados para nomear **coleções**
- Nomes no **plural** devem ser usados para nomear **stores**
- Um **verbo** deve ser utilizado para nomear um **controller**

PUT `/users/1234/favorites/alonso`

POST `http://api.college.org/students/register`

Projetando uma API REST

URI Path Design

- Os nomes de funções **CRUD** não devem ser usados em **URIs**

DELETE /users/1234



GET /deleteUser?id=1234

GET /deleteUser/1234

DELETE /deleteUser/1234

POST /users/1234/delete



Projetando uma API REST

URI Query Design

- A query string pode ser utilizada para filtrar collections or stores
- A query string deve ser utilizada para paginar os resultados de collections ou stores

```
GET /users?role=admin
```

```
GET /users?pageSize=25&pageStartIndex=50
```

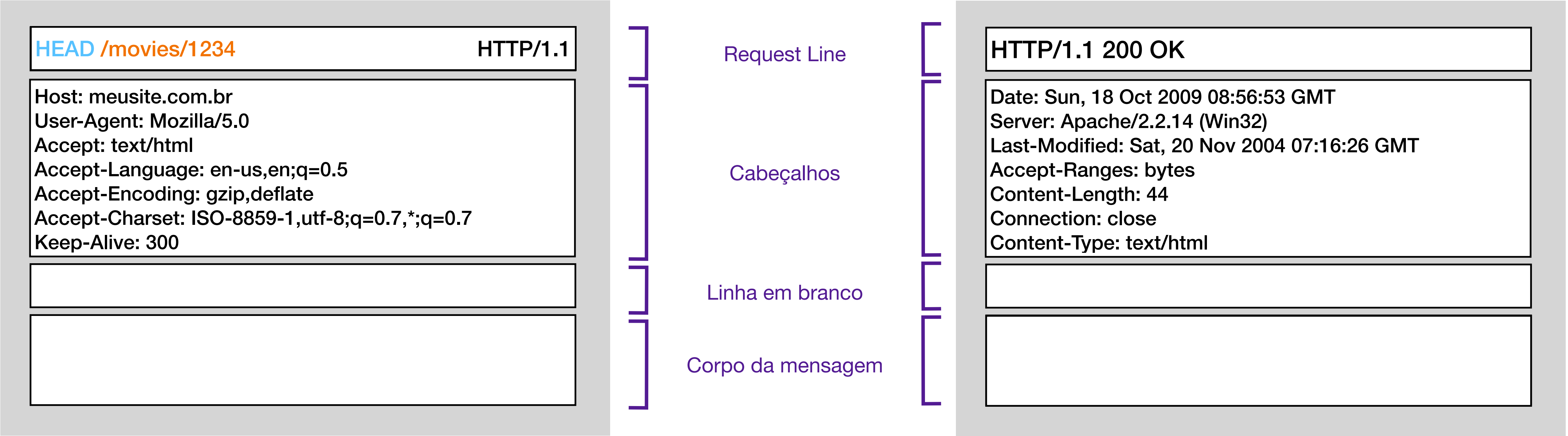
Projetando uma API REST

Interagindo com o HTTP

Finalidade	HTTP Method
Recuperar a representação de um recurso	GET
Criar recurso / Executar um controller	POST
Atualizar um recurso	PUT
Excluir um recurso	DELETE
Recuperar os metadados associados a um recurso	HEAD
Recuperar os metadados associados a um recurso que descrevem as possíveis interações	OPTIONS

Projetando uma API REST

HEAD



Projetando uma API REST

Options

OPTIONS /movies/1234	HTTP/1.1
Host: meusite.com.br User-Agent: Mozilla/5.0 Accept: text/html Accept-Language: en-us,en;q=0.5 Accept-Encoding: gzip,deflate Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7 Keep-Alive: 300	

HTTP/1.1 200 OK	
Date: Sun, 18 Oct 2009 08:56:53 GMT Server: Apache/2.2.14 (Win32) Last-Modified: Sat, 20 Nov 2004 07:16:26 GMT Accept-Ranges: bytes Content-Length: 44 Connection: close Allow: GET, PUT, DELETE	

Se comunicando com uma API REST

Códigos de status HTTP

Grupo	Código	Quando
1xx - Respostas informativas	Raramente são utilizadas	Raramente são utilizadas
2xx - Envias em caso de sucesso	200 OK	Código mais utilizado. Requisição processada com sucesso
	201 Created	Indica que um novo registro foi criado. Usando em respostas a requisições POST
	202 Accepted	Indica que uma ação assíncrona iniciou com sucesso
	204 No content	Usado quando o corpo é intencionalmente vazio. Usado com requisições PUT, POST e DELETE.

Se comunicando com uma API REST

Códigos de status HTTP

Grupo	Código	Quando
3xx - Define respostas de redirecionamento	301 Moved Permanently	Informa que o recurso A agora é o recurso B
	304 Not modified	Resposta utilizada em cenários de cache. Informa ao cliente que a resposta não foi modificada. Portanto, o cliente pode usar a mesma versão em cache da resposta
	307 Temporary redirect	Indica que a API não irá processar a requisição. Uma nova requisição deve ser feita para URI indicada no Header

Se comunicando com uma API REST

Códigos de status HTTP

Grupo	Código	Quando
4xx - Informa erros no lado do cliente	400 Bad Request	Indica que o servidor não conseguiu entender a requisição, devido a sua sintaxe ou estrutura inválida
	401 Unauthorized	Informa que existe uma camada de segurança para recurso solicitado, e que as credenciais informadas requisição estão incorretas
	403 Forbidden	Informa que as credenciais foram reconhecidas ao mesmo tempo que indica que o cliente não tem permissão para acessar o recurso

Se comunicando com uma API REST

Códigos de status HTTP

Grupo	Código	Quando
4xx - Informa erros no lado do cliente	404 Not Found	Informa que o servidor não encontrou o recurso solicitado
	405 Method Not Allowed	Informa que o recurso específico não suporta o método HTTP utilizado
	406 Not Acceptable	Indica que o cliente requisitou dados em um formato de media não aceito
	429 Too Many Requests	Não é tão comum, mas pode ser utilizar para informar que o cliente excedeu o limite permitido de requisições

Se comunicando com uma API REST

Códigos de status HTTP

Grupo	Código	Quando
5xx - Enviadas quando ocorre um erro no lado do servidor	500 Internal Server Error	Erro mais genérico do grupo. Informa que o servidor encontrou um cenário inesperado de erro com o qual não soube lidar
	503 Service Unavailable	Normalmente é utilizado para informar que o servidor está fora do ar, em manutenção ou sobrecarregado

Se comunicando com uma API REST

Headers

- Proveem informações sobre o recurso requisitados
- Indicam algo sobre a mensagem que está sendo enviada
- Regras
 - **Content-Type** deve ser utilizado
 - **Content-Length** deve ser utilizado
 - **Last-Modified** deve ser utilizado em respostas
 - **Cache-Control, Expires e Date** devem ser usados promover o uso de cache

Se comunicando com uma API REST

Headers

Request Header	Response Header	Effect	Exemplos
Accept	Content-Type	Tipo de media	application/json text/html multipart/form-data
Accept-Language	Content-Language	Idioma	en-US, fr;q=0.9 en-GB
Accept-Encoding	Content-Encoding	Compressão	gzip, br compress deflate identity
Accept-Charset	Content-Type charset parameter	Codificação dos caracteres	text/html; charset=utf-8

Projetando uma API REST

Accept & Content-Type

GET /movies/1234

HTTP/1.1

Host: meusite.com.br
User-Agent: Mozilla/5.0
Accept: application/json
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300

HTTP/1.1 200 OK

Date: Sun, 18 Oct 2009 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Sat, 20 Nov 2004 07:16:26 GMT
Accept-Ranges: bytes
Content-Length: 44
Connection: close
Content-Type: application/json

```
{  
  "id": 1234,  
  "title": "Inception",  
  "quotes": [  
    "Dreams feel real while we're in them."  
  ]  
}
```

Middleware

ex

Middlewares

- É uma **função** que trata uma **requisição ou uma resposta** HTTP em uma aplicação Express
 - Pode manipular a requisição ou a resposta
 - Pode realizar uma ação isolada
 - Pode finaliza o fluxo da requisição ao retornar uma resposta
 - Pode passar o controle da requisição ao próximo middleware
- Para carregar um middleware chamamos: **app.use()**

Middlewares

- Uma **aplicativo Express** é essencialmente uma série de chamadas as funções de middleware
- Tais funções que têm acesso a:
 - Ao objeto de requisição (req),
 - Ao objeto de resposta (res)
 - À próxima função de middleware no ciclo de solicitação-resposta da aplicação
 - Comumente indicada por uma variável chamada next

```
app.use(function(req, res, next) {  
  console.log('Request from: ' + req.ip);  
  next();  
});
```

Middleware

Tipos de Middleware

- Em uma aplicação Express podemos ter diversos tipos de middlewares
 - Application-level middleware
 - Router-level middleware
 - Error-handling middleware
 - Built-in middleware
 - Third-party middleware

Middlewares

Application-level middleware

```
const express = require('express')
const app = express()

app.use((req, res, next) => {
  console.log('Time:', Date.now())
  next()
})
```

Middlewares

Router-level middleware

- Funcionam da mesma maneira que application level middleware
- Estão ligado a um Router do Express

```
const express = require('express')
const app = express()
const router = express.Router()
```

```
// a middleware function with no mount path. This code is executed for every
request to the router
router.use((req, res, next) => {
  console.log('Time:', Date.now())
  next()
})
```

Middlewares

Error-handling middleware

- Sempre recebem quatro argumentos
 - Mesmo quando algum deles não é necessário
- Sem os argumentos, ele não será capaz de lidar com os erros

```
app.use((err, req, res, next) => {  
  console.error(err.stack)  
  res.status(500).send('Something broke!')  
})
```

Middlewares

Built-in middleware

- São middleware que são disponíveis no código do Express
- Exemplos:
 - `express.static`
 - `express.json` (a partir do Express 4.16.0+)
 - `express.urlencoded` (a partir do Express 4.16.0+)

```
app.use((err, req, res, next) => {  
  console.error(err.stack)  
  res.status(500).send('Something broke!')  
})
```

Middlewares

Built-in middleware

```
const options = {
  dotfiles: 'ignore',
  etag: false,
  extensions: ['htm', 'html'],
  index: false,
  maxAge: '1d',
  redirect: false,
  setHeaders: function (res, path, stat) {
    res.set('x-timestamp', Date.now())
  }
}

app.use(express.static('public', options))
```

Middlewares

Third-party

- São criados por terceiros para adicionar novas funcionalidades ao Express
- É necessário instalar o módulo Node.js para ter acesso a funcionalidade

```
$ npm install cookie-parser
```

```
const express = require('express')  
const app = express()  
const cookieParser = require('cookie-parser')
```

```
// load the cookie-parsing middleware  
app.use(cookieParser())
```

Middleware

Disponíveis no Express

Middleware	Descrição
router	Sistema de rotas da aplicação
morgan	Realiza o log das requisições HTTP
compression	Comprime as respostas HTTP
json	Realizar o parse de application/json
urlencode	Realiza o parse de application/x-www-form-urlencoded
multer	Realiza o parse de multipart/form-data
bodyParser	Realiza o parse do body usando os middlewares json, url encoded e multipart
timeout	Defina um período de tempo limite para o processamento da solicitação HTTP

Middlewares

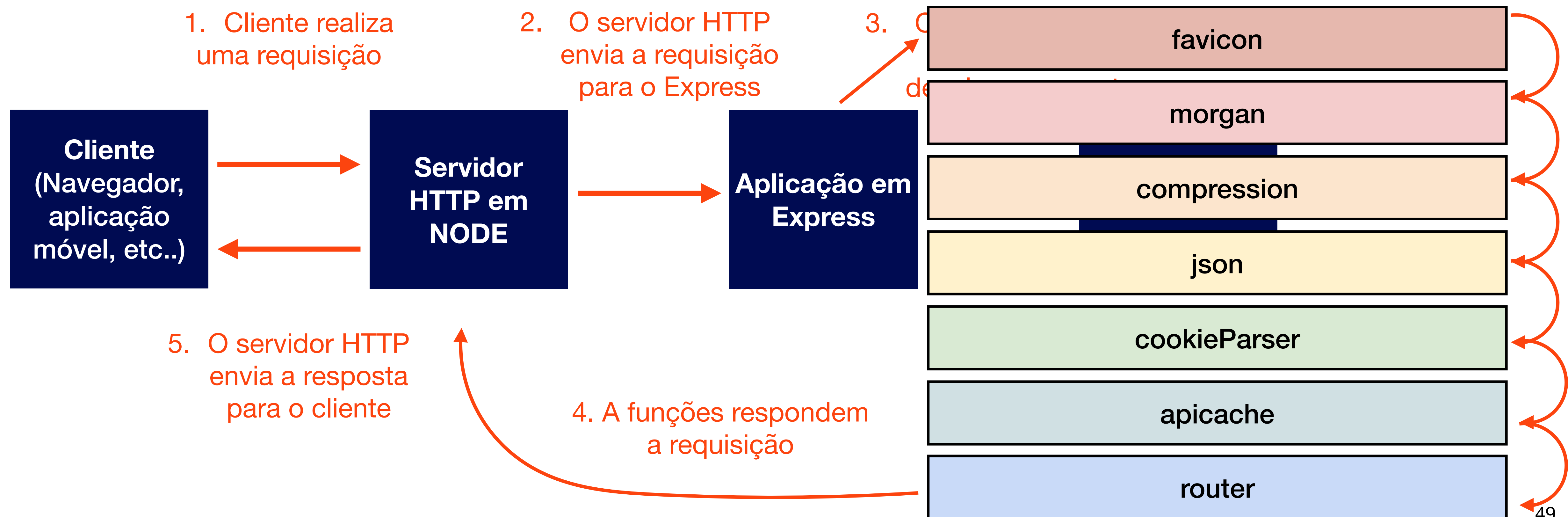
Disponíveis no Express

Middleware	Descrição
<code>cookieParser</code>	Realizar o parse de cookies
<code>session</code>	Da suporte a sessões
<code>cookieSession</code>	Da suporte a cookie de sessão
<code>responseTime</code>	Grava o tempo de resposta do servidor
<code>serve-static</code>	Configura o diretório de recursos estáticos do servidor
<code>serve-favicon</code>	Serve o favicon do website
<code>errorHandler</code>	Gera o stacktrace de erros utilizando HTML

Middlewares

Fluxo da requisição

- Existe apenas um ponto de entrada em aplicações Node + Express



Referências

- O que é uma API (interface de programação de aplicações)?
- REST API Design Rulebook, Mark Masse
- O que é a API REST e como ela difere de outros tipos?
- What is REST
- What is a REST API?
- What is the difference between POST and PUT in HTTP?

Referências

- [Restful API guidelines](#)
- [Exploring REST API Architecture](#)
- [REST API vs RESTful API: Which One Leads in Web App Development?](#)
- [A anatomia de uma API RESTful](#)
- [API REST: o que é e como montar uma API sem complicação?](#)
- [Using Middleware](#)

Por hoje é só