

# API com express e Node

Desenvolvimento de Sistemas Web (DSWI6)

Prof. Luiz Gustavo Diniz de Oliveira Vêras

E-mail: [gustavo\\_veras@ifsp.edu.br](mailto:gustavo_veras@ifsp.edu.br)



# Objetivos

## ✓ Express

- ✓ Criando um servidor
- ✓ Tratadores de rotas (Routes handlers)
- ✓ O Ciclo Request/Response
- ✓ Os objetos Request e Response
- ✓ Middlewares
- ✓ Configurando rotas com objeto Router
- ✓ Tratamento de erros



# Express

Express é o framework web Node mais popular e é a biblioteca usada no desenvolvimento de vários outros frameworks Node populares.

- Veja uma lista em <https://expressjs.com/en/resources/frameworks.html>

Fornece mecanismos para:

- Escreva manipuladores para solicitações com diferentes verbos HTTP em diferentes caminhos de URL (rotas);
- Integração com “engines” de renderização de "visualização" para gerar respostas inserindo dados em modelos (Exemplo: pug);
- Defina as configurações comuns do aplicativo da Web, como a porta a ser usada para conexão e o local de *templates* usados para renderizar *views* a resposta;
- Adicione "*middleware*" de processamento de solicitação adicional em qualquer ponto do *pipeline* de manipulação de solicitação.



# Criando um servidor

Para criarmos um servidor com o express basta, em um arquivo .js:

- Importar o módulo express;
  - `const express = require("express");`
- Criar um objeto do aplicativo express;
  - `const app = express();`
- Definir rotas (paths) para requisição e seus *handlers*;
  - `app.get(path, handler);`
- Iniciar o servidor na porta desejada.
  - `app.listen(porta, callback);`



# Criando um servidor

## Hello World

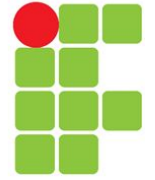
```
const express = require('express');
const app = express();
const port = 3000;

app.get('/', function(req, res) {
  res.send('Hello World!')
});

app.listen(port, function() {
  console.log(`Exemplo de app aguardando na porta ${port}!`)
});
```

# Criando um servidor

## Hello World



Importa o módulo  
"express"

Definição de uma  
rota para requisições  
get para o "path".

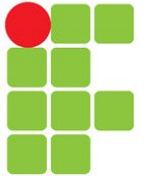
O objeto app denota  
convencionalmente o  
aplicativo Express.

Porta de escuta de  
requisições.

Inicia o servidor na  
porta especificada e  
imprime uma  
mensagem no  
console.

```
const express = require('express');  
const app = express();  
const port = 3000;  
  
app.get('/', function(req, res) {  
  res.send('Hello World!')  
});  
  
app.listen(port, function() {  
  console.log(`Exemplo de app aguardando na porta ${port}!`)  
});
```

Função callback que  
recebe os  
argumentos de  
requisição e  
resposta. Para  
retornar a resposta  
cliente invoca send();



# Tratadores de rotas

O modelo que o express é o de atribuir tratadores (handlers) para cada uma das rotas (routes) do servidor. O handler que será invocado depende:

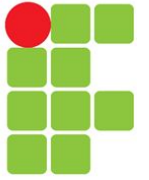
- do verbo HTTP definida na requisição
- do path (rota) definido para aquele verbo

O objeto da aplicação express oferece métodos para definir route handlers para todos os verbos http.

## Métodos de app que representam verbos HTTP

```
checkout(), copy(), delete(), get(), head(), lock(),  
merge(), mkactivity(), mkcol(), move(), m-search(),  
notify(), options(), patch(), post(), purge(), put(),  
report(), search(), subscribe(), trace(), unlock(),  
unsubscribe().
```

Existe um método especial chamado **app.all()**, que invoca um handler para qualquer um dos verbos.



# Tratadores de rotas

## GET

Exemplos:

```
app.get('/', function(req, res) {  
  //código de tratamento de uma requisição para GET  
});
```

## POST

```
app.post('/', function(req, res) {  
  //código de tratamento de uma requisição para POST  
});
```

## PUT

```
app.put('/', function(req, res) {  
  //código de tratamento de uma requisição para PUT  
});
```

## DELETE

```
app.delete('/', function(req, res) {  
  //código de tratamento de uma requisição para DELETE  
});
```

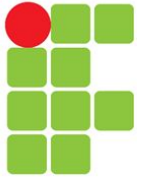




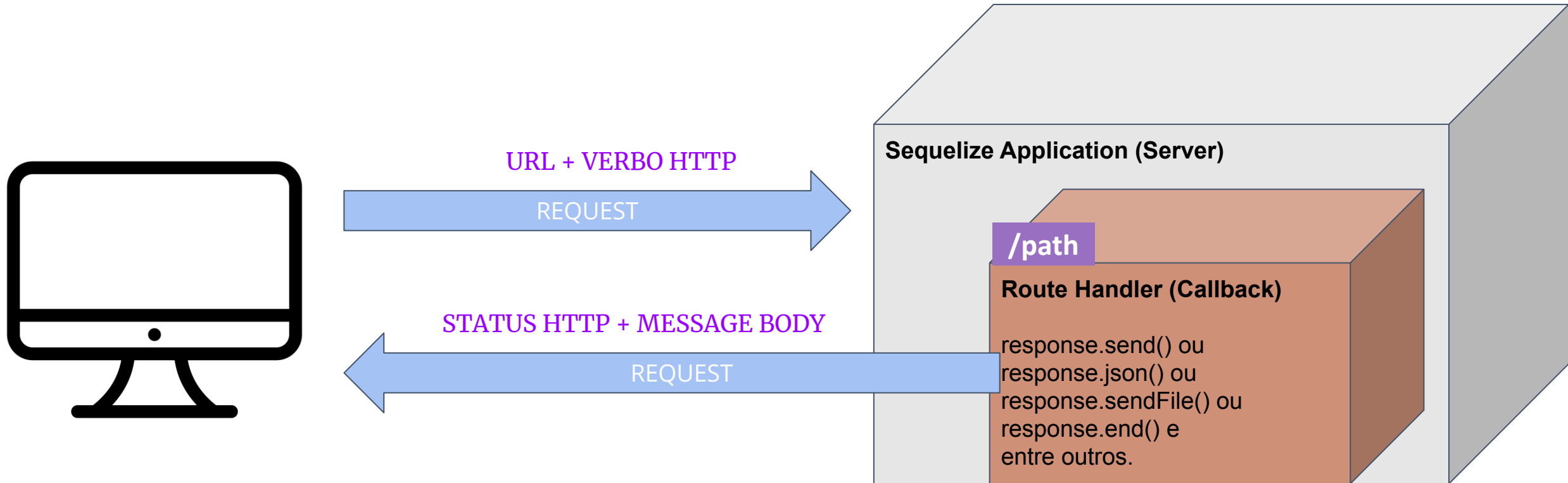
# O Ciclo Request/Response

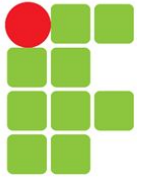
O ciclo de solicitação/resposta rastreia como a solicitação de um usuário flui pelo aplicativo.

Compreender o ciclo de solicitação/resposta é útil para descobrir quais arquivos editar ao desenvolver um aplicativo (e onde procurar quando as coisas não estiverem funcionando).

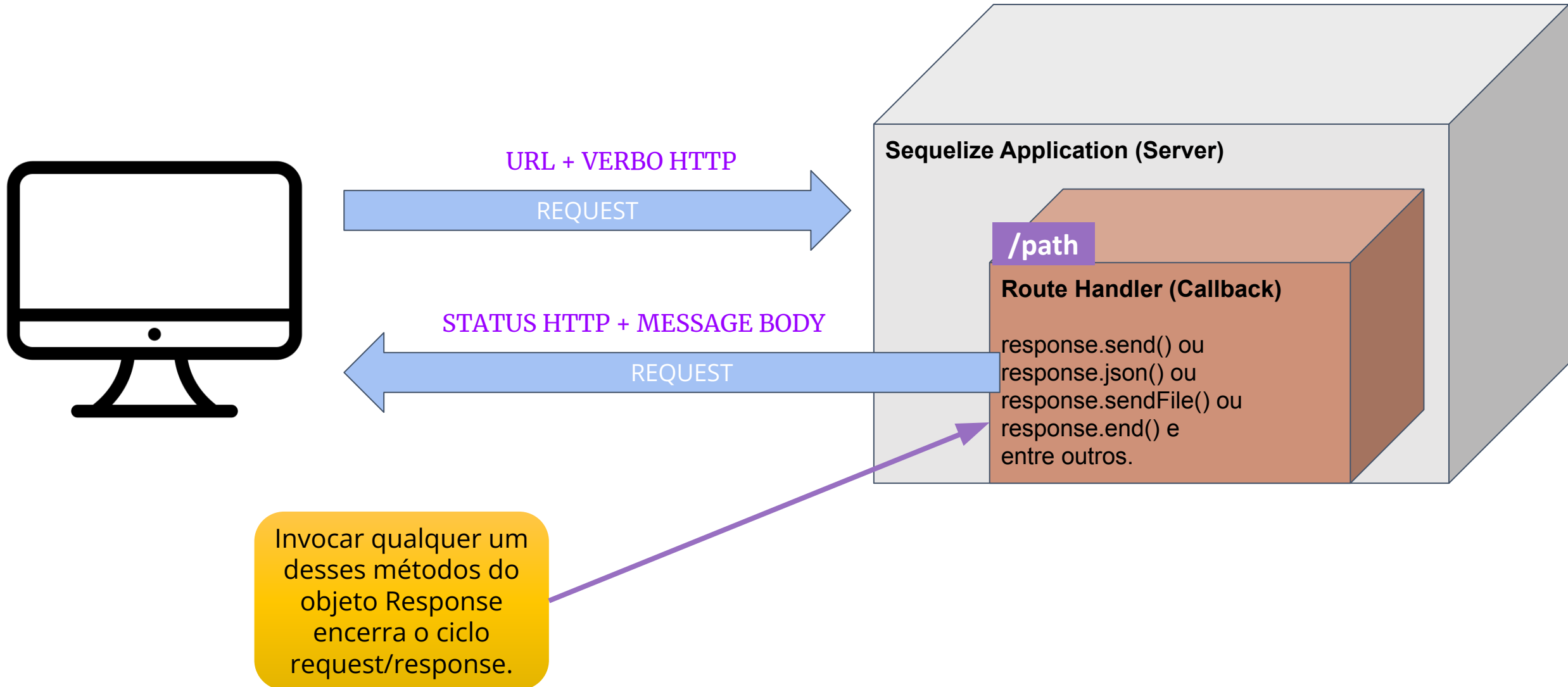


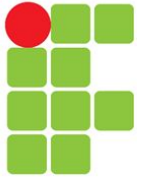
# O Ciclo Request/Response





# O Ciclo Request/Response



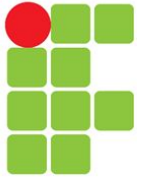


# O Ciclo Request/Response

Métodos de Response que encerram o ciclo:

Método	Descrição
<code>res.download()</code>	Solicita que um arquivo seja baixado.
<code>res.end()</code>	Finalize o processo de resposta.
<code>res.json()</code>	Envie uma resposta JSON.
<code>res.jsonp()</code>	Envie uma resposta JSON com suporte JSONP.
<code>res.redirect()</code>	Redirecionar uma solicitação.
<code>res.render()</code>	Renderize um modelo de visualização.
<code>res.send()</code>	Envie uma resposta de vários tipos.
<code>res.sendFile()</code>	Envie um arquivo como um fluxo de octetos.
<code>res.sendStatus()</code>	Defina o código de status da resposta e envie sua representação de string como o corpo da resposta.

**Fonte:** <https://expressjs.com/en/guide/routing.html#response-methods>



# O Ciclo Request/Response

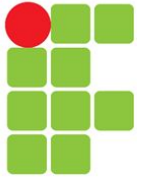
Métodos de Response que encerram o ciclo:

Método	Descrição
<a href="#">res.download()</a>	Solicita que um arquivo seja baixado.
<a href="#">reenviar()</a>	Finalize o processo de resposta.
<a href="#">res.json()</a>	Envie uma resposta JSON.
<a href="#">res.jsonp()</a>	Envie uma resposta JSON com suporte JSONP.
<a href="#">res.redirect()</a>	Redirecionar uma solicitação.
<a href="#">res.render()</a>	Renderize um modelo de visualização.
<a href="#">res.send()</a>	Envie uma resposta de vários tipos.
<a href="#">res.sendFile()</a>	Envie um arquivo como um fluxo de octetos.
<a href="#">res.sendStatus()</a>	Defina o código de status da resposta e envie sua representação de string como o corpo resposta.

Se nenhum desses métodos for chamado de um tratador de rota (route handler), a solicitação do cliente ficará suspensa.

Clique no link para ver exemplos de uso de cada um deles.

Fonte: <https://expressjs.com/en/guide/routing.html#response-methods>



# Objetos Request e Response

## Parâmetros do *path handler* (callback)

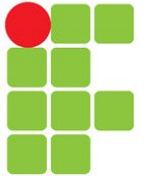
A função que trata de uma requisição recebem um objeto de requisição (*Request*) e um de resposta (*Response*).

Você pode dar qualquer nome para os parâmetros no *callback*. Eles sempre são objetos *Request* e *Response*. Porém, faz sentido nomeá-los de forma que você possa identificar o objeto com o qual está trabalhando no corpo do retorno de chamada.

Primeiro argumento:  
Objeto Request.

Segundo argumento:  
Objeto Response.

```
function(req, res) {  
  //código de tratamento de uma requisição  
}
```

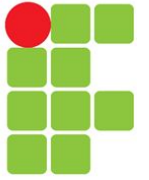


# Objetos Request e Response

## Propriedades de Request

Link: <http://expressjs.com/en/5x/api.html#req>

Propriedade	Descrição	Exemplos
req.baseUrl	O caminho de URL no qual uma instância de roteador foi montada.	<pre>const greet = express.Router()  greet.get('/jp', (req, res) =&gt; {    console.log(req.baseUrl) // /greet    res.send('Konichiwa!')  })  app.use('/greet', greet) // load the router on '/greet'</pre>
req.body	Contém pares de valores-chave de dados enviados no corpo da solicitação. Por padrão, ele é indefinido e é preenchido quando você usa middleware de análise de corpo, como <a href="#">body-parser</a> e <a href="#">multer</a> .	<pre>const app = require('express')() const bodyParser = require('body-parser') const multer = require('multer') // v1.0.5 const upload = multer() // for parsing multipart/form-data app.use(bodyParser.json()) // for parsing application/json app.use(bodyParser.urlencoded({ extended: true })) // for parsing application/x-www-form-urlencoded app.post('/profile', upload.array(), (req, res, next) =&gt; {   console.log(req.body);  res.json(req.body); })</pre>



# Objetos Request e Response

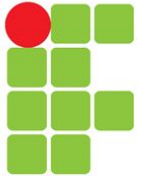
## Propriedades de Request

Link: <http://expressjs.com/en/5x/api.html#req>

Propriedade	Descrição	Exemplos
req.cookies	Ao usar o middleware <a href="#">cookie-parser</a> , essa propriedade é um objeto que contém os cookies enviados pela solicitação. Se a solicitação não contiver cookies, o padrão será {}.	<pre>// Cookie: name=tj console.dir(req.cookies.name)  // =&gt; "tj"</pre>
req.ip	Contém o endereço IP remoto da solicitação.	<pre>console.dir(req.ip)  // =&gt; "127.0.0.1"</pre>
req.params	Esta propriedade é um objeto que contém propriedades mapeadas para os “parâmetros” da rota nomeada.	<pre>// GET /user/tj para endpoint /user/:name console.dir(req.params.name)  // =&gt; "tj"</pre>

(... continuação)



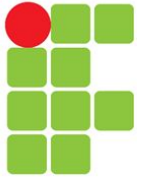


# Objetos Request e Response

## Métodos de Request

Link: <http://expressjs.com/en/5x/api.html#req>

Propriedade	Descrição	Exemplos
req.accepts(types)	Verifica se os tipos de conteúdo especificados são aceitáveis, com base no campo de cabeçalho Aceitar HTTP da solicitação. Se nenhum dos tipos de conteúdo especificados for aceitável, retorna false (nesse caso, o aplicativo deve responder com 406 "Não aceitável").	<pre>// Accept: text/*, application/json req.accepts('html') // =&gt; "html" req.accepts('text/html') // =&gt; "text/html" req.accepts(['json', 'text']) // =&gt; "json" req.accepts('application/json') // =&gt; "application/json"</pre>
req.get(field)	Retorna o campo de cabeçalho de solicitação HTTP especificado (correspondência que não diferencia maiúsculas de minúsculas).	<pre>req.get('Content-Type') // =&gt; "text/plain" req.get('content-type') // =&gt; "text/plain" req.get('Something') // =&gt; undefined</pre>

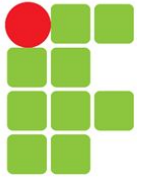


# Objetos Request e Response

## Propriedades de Response

Link: <http://expressjs.com/en/5x/api.html#res>

Propriedade	Descrição	Exemplos
res.headersSent	Propriedade booleana que indica se o aplicativo enviou cabeçalhos HTTP para a resposta.	<pre>app.get('/', (req, res) =&gt; {   console.log(res.headersSent) // false   res.send('OK')   console.log(res.headersSent) // true })</pre>
res.locals	Define o cabeçalho HTTP Content-Type para o tipo MIME conforme determinado pelo tipo especificado.	<pre>app.use((req, res, next) =&gt; {   res.locals.user = req.user   res.locals.authenticated = !req.user.anonymous   next() })</pre>



# Objetos Request e Response

## Métodos de Response

Link: <http://expressjs.com/en/5x/api.html#res>

Propriedade	Descrição	Exemplos
<code>res.cookie(name, value [, options])</code>	Define o nome do cookie como valor. O parâmetro value pode ser uma string ou objeto convertido em JSON.	<pre>res.cookie('name', 'tobi',   { domain: '.example.com', path: '/admin', secure: true }); res.cookie('rememberme', '1',   { expires: new Date(Date.now() + 900000), httpOnly: true });</pre>
<code>res.type(type)</code>	Define o cabeçalho HTTP Content-Type para o tipo MIME conforme determinado pelo tipo especificado.	<pre>res.type('.html') // =&gt; 'text/html' res.type('html') // =&gt; 'text/html' res.type('json') // =&gt; 'application/json' res.type('application/json') // =&gt; 'application/json' res.type('png') // =&gt; image/png;</pre>
<code>res.get(httpHeader)</code>	Retorna o cabeçalho de resposta HTTP especificado pelo campo. A correspondência não diferencia maiúsculas de minúsculas.	<pre>res.get('Content-Type') // =&gt; "text/plain"</pre>
<code>res.status(code)</code>	Sets the HTTP status for the response.	<pre>res.status(403).end() res.status(400).send('Bad Request') res.status(404).sendFile('/absolute/path/to/404.png')</pre>

# Objetos Request e Response

## Métodos de Response

Link: <http://expressjs.com/en/5x/api.html#res>

Os métodos do objeto Response que encerram o ciclo request/response já foram listado anteriormente.

Propriedade	Descrição	Exemplos
<code>res.cookie(name, value [, options])</code>	Define o nome do cookie como valor. O parâmetro value pode ser uma string ou objeto convertido em JSON.	<pre>res.cookie('name', 'tobi', { domain: '.example.com', path: '/admin', secure: true });  res.cookie('rememberme', '1',  { expires: new Date(Date.now() + 900000), httpOnly: true });</pre>
<code>res.type(type)</code>	Define o cabeçalho HTTP Content-Type para o tipo MIME conforme determinado pelo tipo especificado.	<pre>res.type('.html') // =&gt; 'text/html' res.type('html') // =&gt; 'text/html' res.type('json') // =&gt; 'application/json' res.type('application/json') // =&gt; 'application/json'  res.type('png') // =&gt; image/png;</pre>
<code>res.get(httpHeader)</code>	Retorna o cabeçalho de resposta HTTP especificado pelo campo. A correspondência não diferencia maiúsculas de minúsculas.	<pre>res.get('Content-Type')  // =&gt; "text/plain"</pre>
<code>res.status(code)</code>	Sets the HTTP status for the response.	<pre>res.status(403).end() res.status(400).send('Bad Request')  res.status(404).sendFile('/absolute/path/to/404.png')</pre>

# Exemplo

## POST /:nome

```
app.post('/:nome', function(req, res) {  
  // Imprime a rota deste endpoint, ip e o navegador.  
  console.log(`A rota ${req.url}  
    foi acessada pelo IP ${req.ip}  
    usando o navegador ${req.get("User-agent")}!`);  
  // Body só gera resultado se body-parser for usado como middleware  
  const data = req.body;  
  // Lê valor do path param definindo na URL  
  const nome = req.params.nome;  
  // Verifica se o client aceita tipos de texto e retorna um valor Truthy para  
  if(req.accepts("text")){  
    // Define o status HTTP da resposta  
    res.type("text");  
    // Define o status HTTP da resposta  
    res.status(200);  
    // Configura retorno de cookie com prazo de expiração  
    res.cookie("nome", "IFSP", {expires: new Date(Date.now() + 10000)});  
    // Define o status HTTP da resposta  
    res.send({  
      nome: nome,  
      data: data,  
    });  
  } else{  
    // Se o client não aceita text, configura  
    // status - 406 Not Acceptable  
    res.status(406);  
    res.end();  
  }  
});
```





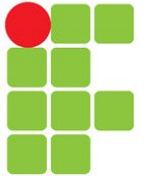
# Middlewares

## O que são?

Funções/*callbacks* que normalmente executam alguma operação na solicitação ou resposta e, em seguida, chamam a próxima função na "pilha", que pode ser mais um *middleware* ou um manipulador de rotas.

A ordem na qual o *middleware* é chamado depende do desenvolvedor do aplicativo.

Adicionamos um *middleware* ao aplicativo express com o método `app.use()`.



# Middlewares

Embora o Express em si seja bastante minimalista, os desenvolvedores criaram pacotes de *middleware* compatíveis para resolver quase qualquer problema de desenvolvimento da web.

Existem bibliotecas para trabalhar com:

- cookies
- sessões
- logins de usuários
- parâmetros de URL
- dados POST
- cabeçalhos de segurança e muito mais.

Veja uma lista de middlewares em

<https://expressjs.com/en/resources/middleware.html>



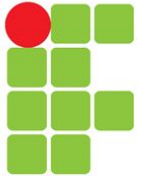
# Middlewares

É comum ver um middleware com o parâmetro adicional **next**.

```
function(req, res, next) {  
    //código de tratamento de uma requisição  
    next(); //passa o controle para o próximo middleware ou handler  
});
```

Podemos assim montar uma *stack* (pilha) de middlewares.





# Middlewares

É comum ver um middleware  
**next.**

O middleware pode executar qualquer operação, executar qualquer código, fazer alterações no objeto de solicitação e resposta e também encerrar o ciclo de request-response. Se não terminar o ciclo, ele deve chamar `next()` para passar o controle para a próxima função de middleware (ou a solicitação ficará suspensa).

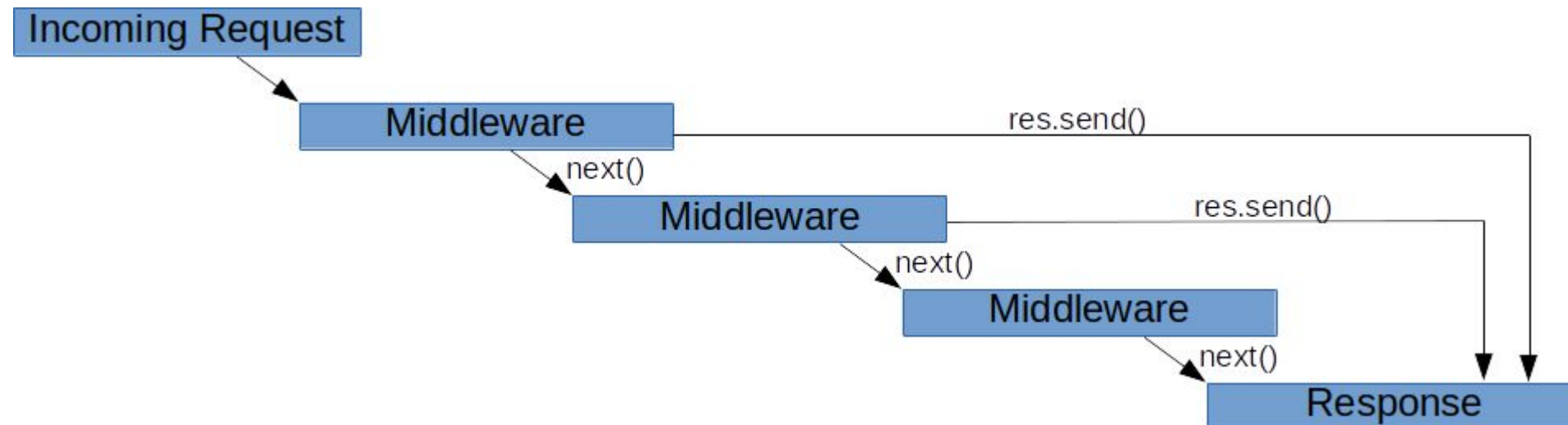
```
function(req, res, next) {  
  //código de tratamento de uma requisição  
  next(); //passa o controle para o próximo middleware ou handler  
});
```

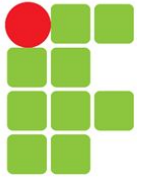
Podemos assim montar uma *stack* (pilha) de middlewares.



# Middlewares

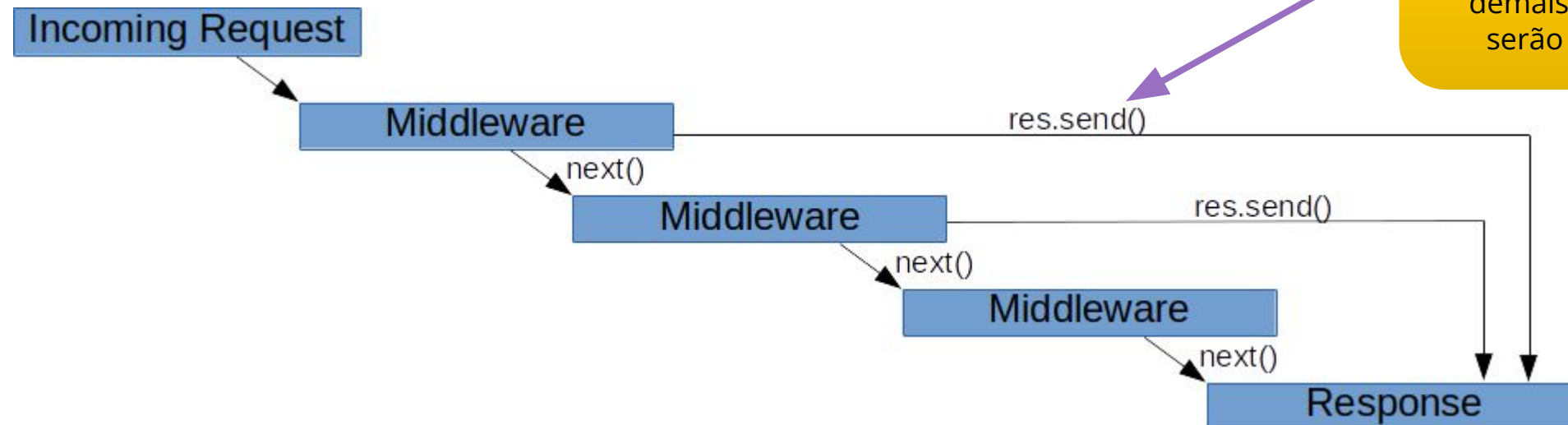
Esquema de funcionamento de uma stack de middlewares.



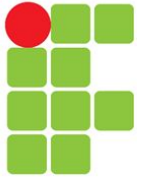


# Middlewares

Esquema de funcionamento de uma stack de middlewares.

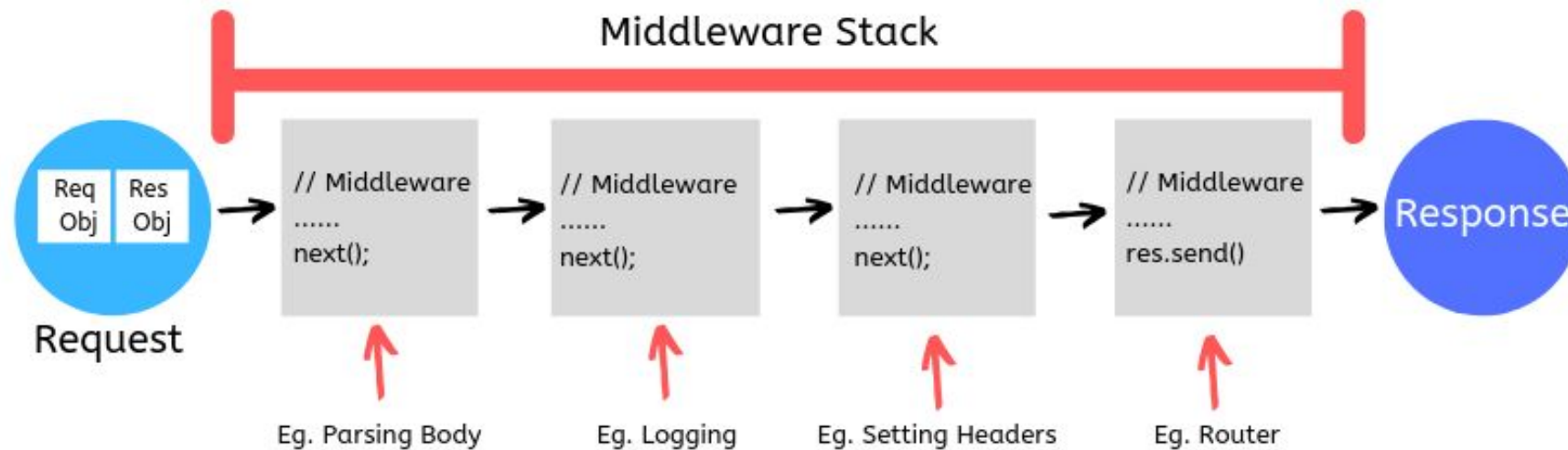


Se qualquer um dos *middlewares* invocar um método que encerra o ciclo request/response, os demais na pilha não serão executados.



# Middlewares

Esquema de funcionamento de uma stack de middlewares.



Fonte: <https://iq.opengenus.org/middlewares-in-express/>

# Exemplo

Criando um *middleware* de log.



```
const express = require('express');
const app = express();
const port = 3000;

const middlewareLog = (req, res, next) => {
  console.log("-->Middleware de Log<--");
  console.log(req.url);
  console.log(req.get("User-Agent"));
  console.log(req.ip);
  next(); //passa o controle para o próximo middleware ou handler
}

// Adicionando o middleware ao aplicativo express
app.use(middlewareLog);

// Quando este endpoint for acessado, o middleware será executado antes
app.get('/', function (req, res, next) {
  res.send('Dado retornado');
});

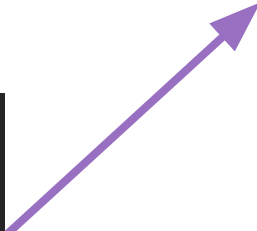
app.listen(port, function() {
  console.log(`Exemplo de app aguardando na porta ${port}!`)
});
```

# Exemplo


Definindo a ordem de execução dos *middlewares*.



```
const middlewareQueryString = (req, res, next) => {  
  console.log("-->Middleware Query String<--");  
  // No browser, acesse algo como http://localhost:3003?nome=IFSP  
  console.log(req.query);  
  next();  
}  
  
const middlewareEndpoint1 = (req, res, next) => {  
  console.log("Middleware Endpoint 1");  
  next();  
}
```



```
app.use(middlewareLog); // Executada primeiro  
app.use(middlewareQueryString); // Executada em seguida
```



```
app.use(middlewareQueryString); // Executada primeiro  
app.use(middlewareLog); // Executada em seguida
```

# Exemplo



Encerrando o ciclo no meio da stack de *middlewares*.

```
const middlewareQueryString = (req, res, next) => {  
  console.log("-->Middleware Query String<--");  
  // No browser, acesse algo como http://localhost:3000?nome=IFSP  
  console.log(req.query);  
  res.send("You shall not pass!!");  
  next();  
}  
  
const middlewareEndpoint1 = (req, res, next) => {  
  console.log("Middleware Endpoint 1");  
  next();  
}  
  
app.get('/', function (req, res, next) {  
  res.send('Dado retornado');  
});
```

Como o ciclo é encerrado aqui e a resposta enviada ao client ...

app.use(middlewareQueryString); // Executada primeiro  
app.use(middlewareLog); // Executada, mas o ciclo req/res já foi encerrado

... quando o callback do endpoint for executado o mesmo não poderá mais enviar respostas ao client e um erro será gerado.

Error [ERR\_HTTP\_HEADERS\_SENT]: Cannot set headers after they are sent to the client

# Exemplo

Também é possível configurar para uma rota específica.



```
const express = require('express');
const app = express();
const port = 3000;

const middlewareEndpoint1 = (req, res, next) => {
  console.log("Middleware Endpoint 1");
  next();
}

app.get('/', function (req, res, next) {
  res.send('Dado retornado');
});

// Quando este endpoint for acessado, o middleware será executado antes
app.get('/endpoint1', middlewareEndpoint1, function (req, res, next) {
  res.send('Endpoint 1 acessado');
});

app.listen(port, function() {
  console.log(`Exemplo de app aguardando na porta ${port}!`)
});
```



# Exemplo

Também é possível configurar para uma rota específica.



```
const express = require('express');
const app = express();
const port = 3000;
```

```
const middlewareEndpoint1 = (req, res, next) => {
  console.log("Middleware Endpoint 1");
  next();
}
```

Este middleware só será executado quando este endpoint for acessado.

```
app.get('/', function (req, res, next) {
  res.send('Dado retornado');
});
```

```
// Quando este endpoint for acessado, o middleware será executado antes
app.get('/endpoint1', middlewareEndpoint1, function (req, res, next) {
  res.send('Endpoint 1 acessado');
});
```

```
app.listen(port, function() {
  console.log(`Exemplo de app aguardando na porta ${port}!`);
});
```



# Configurando rotas com objeto Router

As rotas permitem que você corresponda a padrões específicos de caracteres em uma URL e extraia alguns valores da URL e os passe como parâmetros para o manipulador de rotas (como atributos do objeto de solicitação passados como parâmetro).

```
var router = express.Router();
```

# Exemplo



## Criando um router e adicionando no aplicativo Express

```
var router = express.Router();

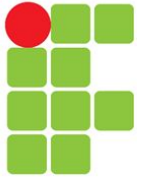
// middleware que é específico para este router
router.use(function timeLog(req, res, next) {
  console.log('Time: ', Date.now());
  next();
});

// define a home page para este router
router.get('/', function(req, res) {
  res.send('Birds home page');
});

// define a página sobre (about) passáros
router.get('/about', function(req, res) {
  res.send('About birds');
});
```

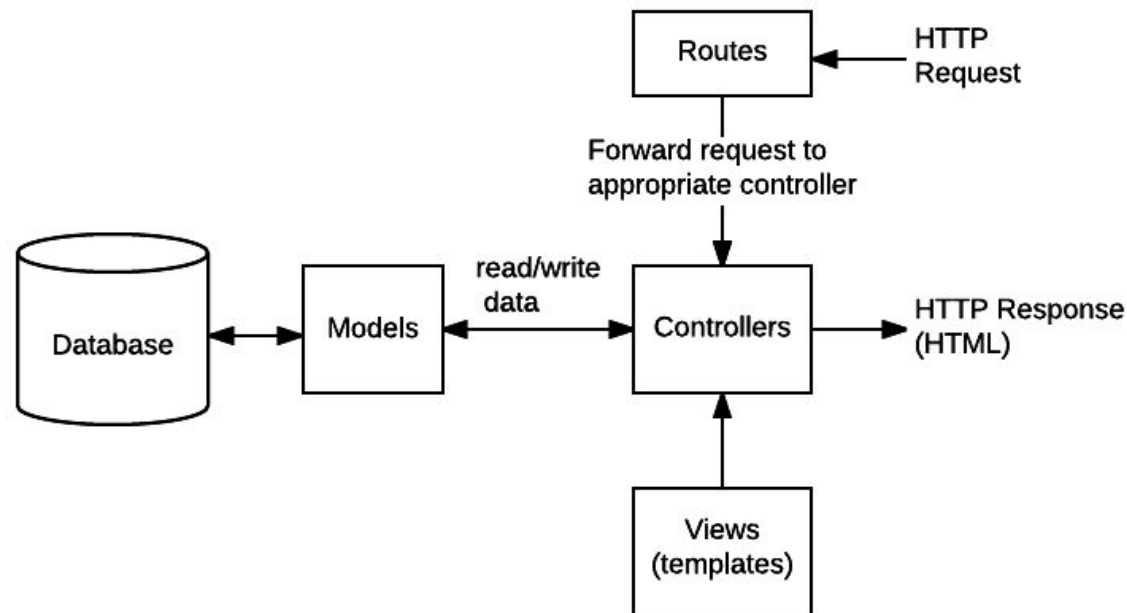


```
app.use("/passaros", passaros);
```



# Configurando rotas com objeto Router

O diagrama abaixo é fornecido como um lembrete do fluxo principal de dados e coisas que precisam ser implementadas ao lidar com uma solicitação/resposta HTTP. Além das visualizações e rotas, o diagrama mostra "controladores" — funções que separam o código para rotear solicitações do código que realmente processa solicitações.



Veja mais em:

[https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express\\_Nodejs/routes](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes)



# Tratamento de erros

Os erros são tratados por uma ou mais funções de middleware especiais que possuem quatro argumentos, em vez dos três habituais: (err, req, res, next). Por exemplo:

```
app.use(function(err, req, res, next) {  
  console.error(err.stack);  
  res.status(500).send('Algo quebrou!');  
});
```

Eles podem retornar qualquer conteúdo necessário, mas devem ser chamados depois de todos os outros `app.use()` e roteiam chamadas para que sejam o **último middleware no processo de manipulação de requisições!**

# Exemplo


## Adicionando um tratamento de erros



```
const middlewareLog = (req, res, next) => {
  console.log("-->Middleware de Log<--");
  console.log(req.url);
  console.log(req.get("User-Agent"));
  console.log(req.ip);
  next();
}

const middlewareQueryString = (req, res, next) => {
  console.log("-->Middleware Query String<--");
  // No browser, acesse algo como http://localhost:3003?nome=IFSP
  console.log(req.query);
  // Vai gerar o erro por acessar um parâmetro que não existe.
  req.query.idade.toString();
  next();
}

const middlewareErrorHandler = function(err, req, res, next) {
  console.error(err.stack);
  res.status(500).send('Algo quebrou!');
};
```



```
app.use(middlewareLog);
app.use(middlewareQueryString);
// Deve vir por último
app.use(middlewareErrorHandler);
```

# Exemplo

## Adicionando um tratamento de erros



```
const middlewareLog = (req, res, next) => {  
  console.log("-->Middleware de Log--");  
  console.log(req.url);  
  console.log(req.get("User-Agent"));  
  console.log(req.ip);  
  next();  
}
```

```
const middlewareQueryString = (req,  
  console.log("-->Middleware Query S  
  // No browser, acesse algo como ht  
  console.log(req.query);  
  // Vai gerar o erro por acessar um  
  req.query.idade.toString();  
  next();  
}
```

```
const middlewareErrorHandler = function(err, req, res, next) {  
  console.error(err.stack);  
  res.status(500).send('Algo quebrou!');  
};
```

Vale comentar que o Express vem com um manipulador de erros integrado, que cuida de quaisquer erros restantes que possam ser encontrados no aplicativo. Essa função de *middleware* de tratamento de erros padrão é incluída no final da pilha de funções de middleware.

```
Log);  
QueryString);  
tmo  
ErrorHandler);
```



# Links

## **EXPRESS TUTORIAIS**

Link: [https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express\\_Nodejs/Introduction](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction)

[https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/Express\\_Nodejs](https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/Express_Nodejs)

[https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express\\_Nodejs/Displaying\\_data](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Displaying_data)

## **ROUTES**

[https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express\\_Nodejs/routes](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes)

## **MIDDLEWARES**

<https://expressjs.com/pt-br/guide/writing-middleware.html>

## **PERFORMANCE**

<http://expressjs.com/en/advanced/best-practice-performance.html>

## **REQ OBJECT**

<https://www.digitalocean.com/community/tutorials/nodejs-req-object-in-expressjs>

## **HTTP Headers**

[Cabeçalhos HTTP - HTTP | MDN \(mozilla.org\)](#)