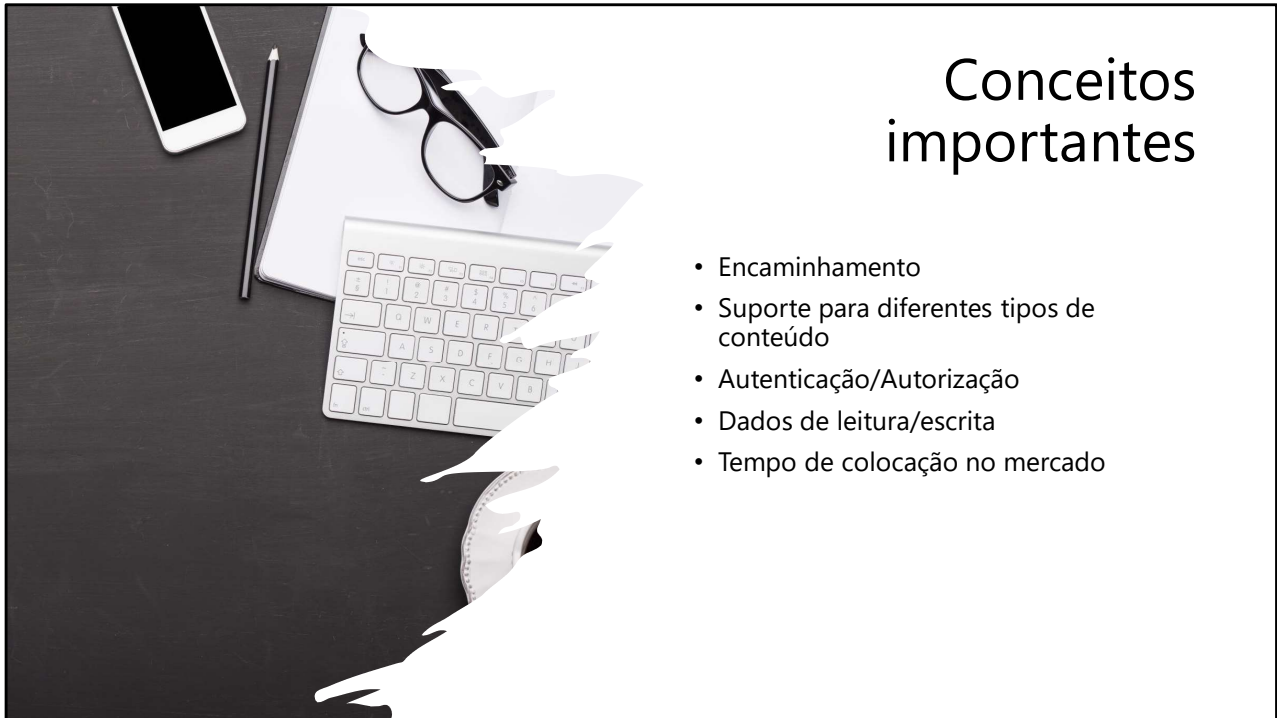




Criar uma API Web com o Node.js



Conceitos importantes

- Encaminhamento
- Suporte para diferentes tipos de conteúdo
- Autenticação/Autorização
- Dados de leitura/escrita
- Tempo de colocação no mercado

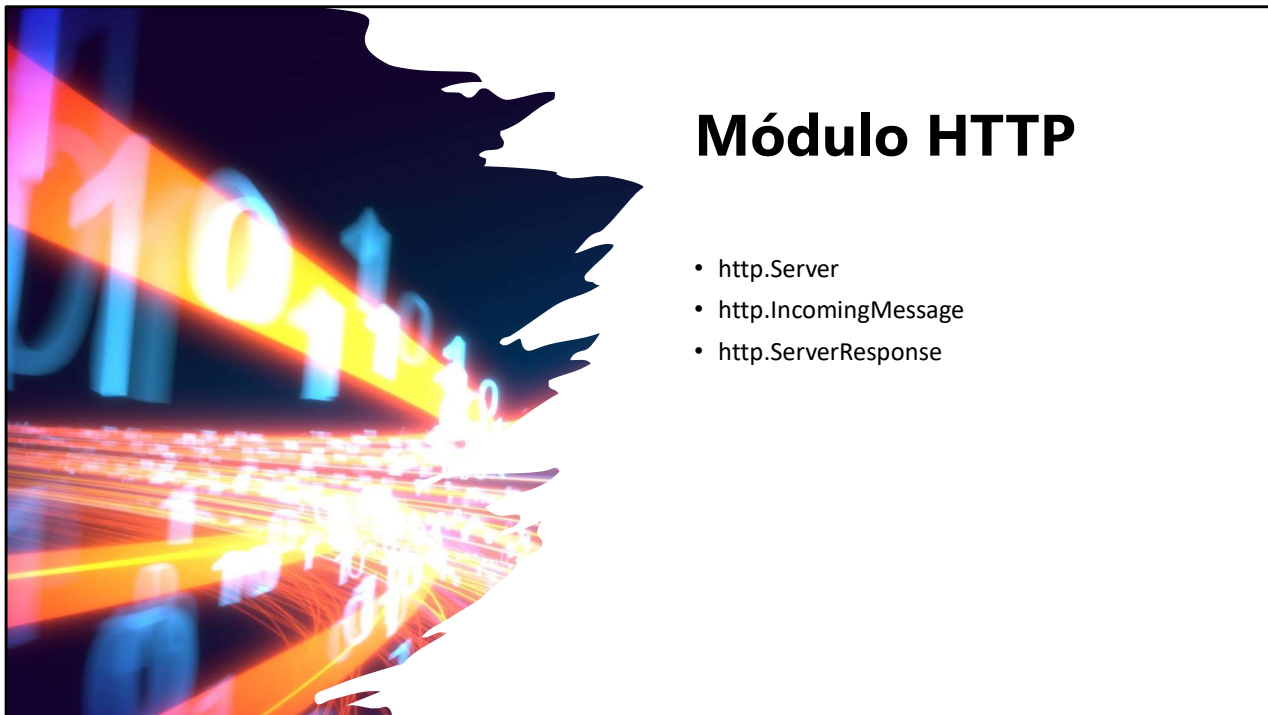
•**Encaminhamento:** a aplicação é dividida em secções diferentes com base em partes do endereço URL.

•**Suporte para diferentes tipos de conteúdo:** os dados a servir podem existir em formatos de ficheiro diferentes, como texto simples, JSON, HTML, CSV e muito mais.

•**Autenticação/Autorização:** alguns dados podem ser confidenciais. É possível que um utilizador precise de iniciar sessão ou ter uma função ou nível de permissão específico para aceder aos dados.

•**Dados de leitura/escrita:** normalmente, os utilizadores precisam de ver e adicionar dados ao sistema. Para adicionar dados, os utilizadores podem introduzir dados num formulário ou carregar ficheiros.

•**Tempo de colocação no mercado:** para criar aplicações Web e APIs de forma eficiente, escolha ferramentas e bibliotecas que proporcionam soluções para problemas comuns. Estas escolhas ajudam o programador a dedicar o máximo de tempo possível aos requisitos de negócio do trabalho.



Módulo HTTP

- `http.Server`
- `http.IncomingMessage`
- `http.ServerResponse`

O Node.js inclui um módulo HTTP incorporado. É um módulo relativamente pequeno que processa a maioria dos tipos de pedido. Suporta tipos comuns de dados, como cabeçalhos, URL e payloads.

`http.Server`: representa uma instância de um Servidor HTTP. Este objeto precisa de ser instruído para ouvir vários eventos numa porta e endereço específicos.

`http.IncomingMessage`: este objeto é um fluxo legível criado por `http.Server` ou `http.ClientRequest`. Utilize-o para aceder ao estado, aos cabeçalhos e aos dados.

`http.ServerResponse`: este objeto é um fluxo criado internamente pelo Servidor HTTP. Esta classe define o aspeto da resposta, por exemplo, o tipo de cabeçalhos e o conteúdo da resposta.

Este exemplo configura a aplicação com os seguintes passos:

1.Criar o servidor: o `createServer()` método cria uma instância da `http.Server` classe.

2.Implementar a chamada de retorno: o `createServer()` método espera uma função conhecida como *chamada de retorno*. Quando a chamada de retorno é

invocada, fornecemos o método com instâncias das `http.IncomingMessage` classes e `http.ServerResponse` . Neste exemplo, fornecemos as req instâncias e res :

1. **Pedido de cliente:** o req objeto investiga que cabeçalhos e dados foram enviados no pedido do cliente.
2. **Resposta** do servidor: o servidor cria uma resposta ao indicar ao objeto os res dados e os cabeçalhos de resposta com os qual deve responder.

1.Começar a escutar pedidos: o `listen()` método é invocado com uma porta especificada. Após a chamada para o `listen()` método, o servidor está pronto para aceitar pedidos de cliente.

Módulo HTTP

```
const http = require('http');
const PORT = 3000;

const server = http.createServer((req, res) => {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('hello world');
});

server.listen(PORT, () => {
  console.log(`listening on port ${PORT}`)
})
```


Este exemplo configura a aplicação com os seguintes passos:

1.Criar o servidor: o `createServer()` método cria uma instância da `http.Server` classe.

2.Implementar a chamada de retorno: o `createServer()` método espera uma função conhecida como *chamada de retorno*. Quando a chamada de retorno é invocada, fornecemos o método com instâncias das `http.IncomingMessage` classes e `http.ServerResponse` . Neste exemplo, fornecemos as `req` instâncias e `res` :

- 1. Pedido de cliente:** o `req` objeto investiga que cabeçalhos e dados foram enviados no pedido do cliente.
- 2. Resposta** do servidor: o servidor cria uma resposta ao indicar ao objeto os `res` dados e os cabeçalhos de resposta com os qual deve responder.

3.Começar a escutar pedidos: o `listen()` método é invocado com uma porta especificada. Após a chamada para o `listen()` método, o servidor está pronto para aceitar pedidos de cliente.



Fluxos

- definem a forma como os dados são transportados
- os dados são enviados, segmento a segmento, do cliente para o servidor e, do servidor para o cliente
- permitem que o servidor consiga processar muitos pedidos em simultâneo
- é uma estrutura de dados fundamental no Node.js que pode ler e escrever dados e enviar e receber mensagens ou *eventos*

```
req.on('data', (chunk) => {  
  console.log('You received a chunk of data', chunk)  
})  
  
res.end('some data')
```

Os *fluxos* não são um conceito Node.js, mas sim um conceito de sistema operativo. Os fluxos definem a forma como os dados são transportados. Os dados são enviados, segmento a segmento, do cliente para o servidor e, do servidor para o cliente. Os fluxos permitem que o servidor consiga processar muitos pedidos em simultâneo.

No nosso exemplo, os req parâmetros e res são fluxos. Utilize o on() método para ouvir dados recebidos de um pedido de cliente como este
Utilize o end() método para os dados enviados de volta para o cliente no fluxo de resposta do res objeto

Arquitetura Express

- Boas funcionalidades
- Afasta a complexidade
- Resolve problemas comuns da Web
- Considerado fidedigno por milhões de programadores

É uma escolha perfeitamente válida para aplicações Web mais pequenas. Se uma aplicação se tornar grande, uma arquitetura como o Express poderá ajudar a criar a arquitetura de forma dimensionável.

Depois de criar algumas aplicações Web, irá reparar que resolve os mesmos problemas repetidamente. São habituais os problemas de gestão de rotas, a autenticação e a autorização e a gestão de erros. Por esta altura, irá começar à procura de uma biblioteca ou arquitetura que aborde alguns ou todos estes problemas.

•**Boas funcionalidades:** o Express tem um conjunto de funcionalidades que lhe permite ser mais rápido e produtivo.

•**Afasta a complexidade:** o Express afasta conceitos complicados, como fluxos, o que torna toda a experiência de programação muito mais fácil.

•**Resolve problemas comuns da Web:** o Express ajuda-o com problemas comuns, como a gestão de rotas, a colocação em cache e o redirecionamento.

•**Considerado fidedigno por milhões de programadores:** segundo o GitHub, 6,8 milhões de programadores estão atualmente a utilizar o Express para as aplicações Web.

Gestão de rotas no Express

```
http://localhost:3000/products
```

Arquitetura Express utiliza:.....

- URL
- Rota
- Verbos de HTTP

O Express ajuda a registar rotas e a emparelhá-las com os verbos HTTP adequados para organizar a aplicação Web.

O termo localhost no URL refere-se ao seu próprio computador. Um URL com mais aspeto de produção pode ter mudado o termo localhost para um nome de domínio como microsoft.com. A parte final do URL é a *rota*. Decide um local específico para ir no servidor. Neste caso, a rota é /products.

Verbos HTTP como post, put e get descrevem a ação pretendida pelo cliente. Cada verbo HTTP tem um significado específico para o que deve acontecer aos dados.

Gestão de rotas no Express

```
app.get('/products', (req, res) => {  
  // handle the request  
})  
  
app.post('/products', (req, res) => {  
  // handle the request  
})
```

O verbo get HTTP significa que um utilizador quer ler dados. O verbo post HTTP significa que pretende escrever dados. Dividir a sua aplicação para que diferentes emparelhamentos de rotas e verbos executem diferentes partes de código faz sentido. Este conceito será abordado mais detalhadamente mais tarde.

Servir diferentes tipos de conteúdo

```
res.send('plain text')
```

```
res.json({ id: 1, name: "Catcher in the Rye" })
```

```
res.writeHead(200, { 'Content-Type': 'application/json' });  
res.end(JSON.stringify({ id: 1, name: "Catcher in the Rye" })))
```

Express suporta muitos formatos de conteúdo que podem ser devolvidos para um cliente de chamada. O `res` objeto inclui um conjunto de funções auxiliares para devolver diferentes tipos de dados. Para devolver texto simples, teria de utilizar o método `send()` da seguinte forma:

Para outros tipos de dados, como JSON, existem métodos dedicados que garantem que o tipo de conteúdo e as conversões de dados corretos são efetuados. Para devolver JSON no Express, utilize o `json()` método da seguinte forma:

O `Content-Type` cabeçalho em HTTP está definido e a resposta também é convertida de um objeto JavaScript para uma versão *stringify* antes de ser devolvida ao cliente de chamadas.

Se compararmos os dois exemplos de código, poderá ver que o Express guarda algumas linhas de escrita, ao utilizar métodos auxiliares para tipos de ficheiros comuns, como JSON e HTML.

Criar uma aplicação Express

1. Instanciar a aplicação
2. Definir rotas e processadores de rotas
3. Configurar o middleware
4. Iniciar a aplicação

1.Instanciar a aplicação: crie uma instância da aplicação Web. Por esta altura, a instância não pode ser executada, mas já tem algo que pode expandir.

2.Definir rotas e processadores de rotas: defina que rotas a aplicação deve ouvir. Uma rota faz parte do URL. Por exemplo, no URL `http://localhost:8000/products`, a parte da rota é `/products`. O Express utiliza rotas diferentes para executar diferentes partes do código. Outros exemplos de rotas são a barra `/`, também conhecida como rota predefinida, e `/orders`. As rotas serão exploradas com mais detalhe posteriormente neste módulo.

3.Configurar o middleware: o middleware é um bloco de código que pode ser executado antes ou depois de um pedido. Também pode utilizar o middleware para lidar com a autenticação/autorização ou para adicionar uma funcionalidade à aplicação.

4.Iniciar a aplicação: define uma porta e, em seguida, instrui a aplicação para ouvir essa porta. Agora, a aplicação está pronta para receber pedidos.

Para começar a programar uma aplicação Node.js com a arquitetura Express, precisa de a instalar como uma dependência. Também é recomendado que inicialize primeiro um projeto Node.js, para que todas as dependências

transferidas sejam incluídas no ficheiro **package.json**. É uma recomendação geral para todas as aplicações programadas para o runtime Node.js. As vantagens desta ação surgem ao emitir o código para um repositório como o GitHub. Qualquer pessoa a obter o código do GitHub pode facilmente utilizar o código que escreveu ao instalar as respetivas dependências.

Criar uma aplicação Express

```
Bash
npm init -y
npm install express
```

```
Bash
"dependencies": {
  "express": "^4.18.1"
```

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

app.get('/', (req, res) => res.send('Hello World!'));

app.listen(port, () => console.log(`Example app listening on port ${port}!`));
```

1. Abra um terminal e introduza os seguintes comandos:

2. Bash Copiar

3. `npm init -y` `npm install express` O `init` comando cria um ficheiro **package.json** predefinido para o projeto Node.js. O `install` comando instala a arquitetura Express.

4. Num editor de código, abra o ficheiro `package.json`.

5. `dependencies` Na secção, localize a `express` entrada:

6. Bash Copiar

7. `"dependencies": { "express": "^4.18.1"` Esta entrada indica que a arquitetura Express está instalada.

1. Feche o ficheiro.

8. Num editor de código, crie um ficheiro com o nome **app.js** adicione o seguinte código:

9. JavaScript Copiar

10. `const express = require('express');` `const app = express();` `const port = 3000;` `app.get('/', (req, res) => res.send('Hello World!'));` `app.listen(port, () => console.log('Example app listening on port ${port}!'));` O código cria uma instância de uma aplicação Express ao invocar o `express()` método .

11. Repare como o código configura uma rota para barra / com a sintaxe:

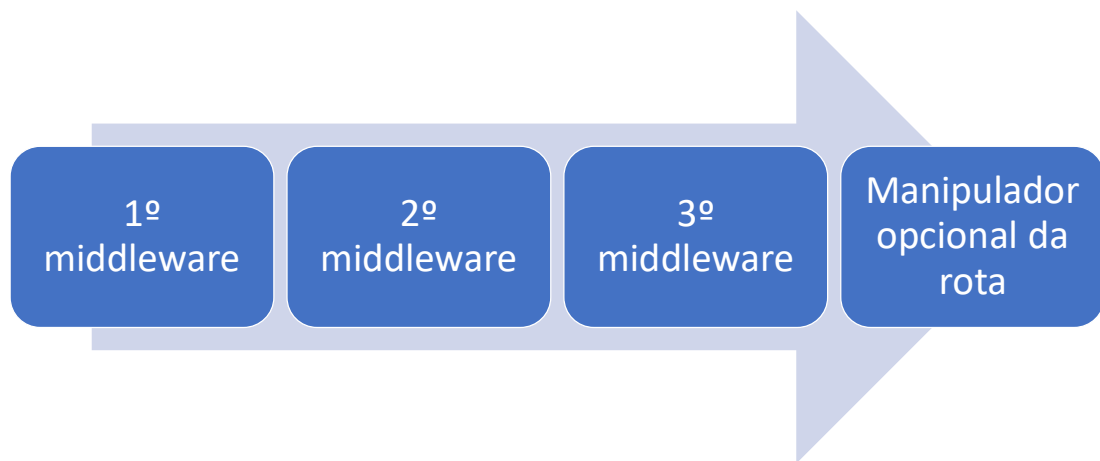
12. `app.get('/', (req, res) => res.send('Hello World!'));`
13. Depois de configurar a rota, o código inicia a aplicação Web ao invocar o `listen()` método :
14. `app.listen(port, () => console.log(Exemplo de escuta de aplicação na porta ${port}!));`
15. Guarde as alterações feitas ao ficheiro `app.js` e feche o ficheiro.
16. No terminal, execute o seguinte comando para iniciar a aplicação Web Express:
17. Bash Copiar
18. `node app.js` Deverá ver o seguinte resultado:
19. Saída Copiar
20. `Example app listening on port 3000!` Este resultado significa que a aplicação está operacional e pronta para receber pedidos.
21. Num browser, aceda a `http://localhost:3000`. Deverá ver o seguinte resultado:
22. Saída Copiar
23. `Hello World!`
24. No terminal, prima `Ctrl + C` para parar o programa Web Express.

Criar uma aplicação Web que devolve dados JSON

```
app.get("/products", (req,res) => {  
  const products = [  
    {  
      id: 1,  
      name: "hammer",  
    },  
    {  
      id: 2,  
      name: "screwdriver",  
    },  
    {  
      id: 3,  
      name: "wrench",  
    },  
  ];  
  res.json(products);  
});
```

```
const express = require("express");  
const app = express();  
const port = 3000;  
  
app.get("/", (req, res) => res.send("Hello World!"));  
  
app.get("/products", (req,res) => {  
  const products = [  
    {  
      id: 1,  
      name: "hammer",  
    },  
    {  
      id: 2,  
      name: "screwdriver",  
    },  
    ,  
    {  
      id: 3,  
      name: "wrench",  
    },  
  ],  
  res.json(products);  
})  
  
app.listen(port, () => console.log(`Example app listening on port ${port}!`));
```

Middlewares



Acima você pode ver como uma requisição passa por uma aplicação Express. Ela atravessa três middlewares. Cada um pode modificá-la, então baseado na regra de negócio, o terceiro middleware pode mandar de volta uma resposta ou passá-la para um manipulador de rota.

Middleware

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

app.use((request, response, next) => {
  console.log(request.headers)
  next()
})

app.use((request, response, next) => {
  request.chance = Math.random()
  next()
})

app.get('/', (request, response) => {
  response.json({
    chance: request.chance
  })
})

app.listen(3000)
```

Coisas para serem notadas aqui:

- app.use: é assim que você define middlewares - ela recebe uma função com três parâmetros, o primeiro sendo a requisição, o segundo sendo a resposta e o terceiro sendo o callback para o **próximo**(next). Chamando next o Express saberá que ele pode ir para o próximo middleware ou gerenciador de rota.
- o primeiro middleware irá logar os cabeçalhos e imediatamente chamar o próximo.
- o segundo adiciona uma propriedade a mais - **isso é uma das melhores características do padrão middleware**. Seus middlewares podem adicionar informações a mais ao objeto da requisição que os próximos middlewares poderão ler/alterar.