UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Fundamentos de Express

QXD0020 - Desenvolvimento de Software para Web

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

Agenda

- Introdução
- Criando um projeto usando Express
- Middleware
- Criando um servidor web com Node & Express

Introdução

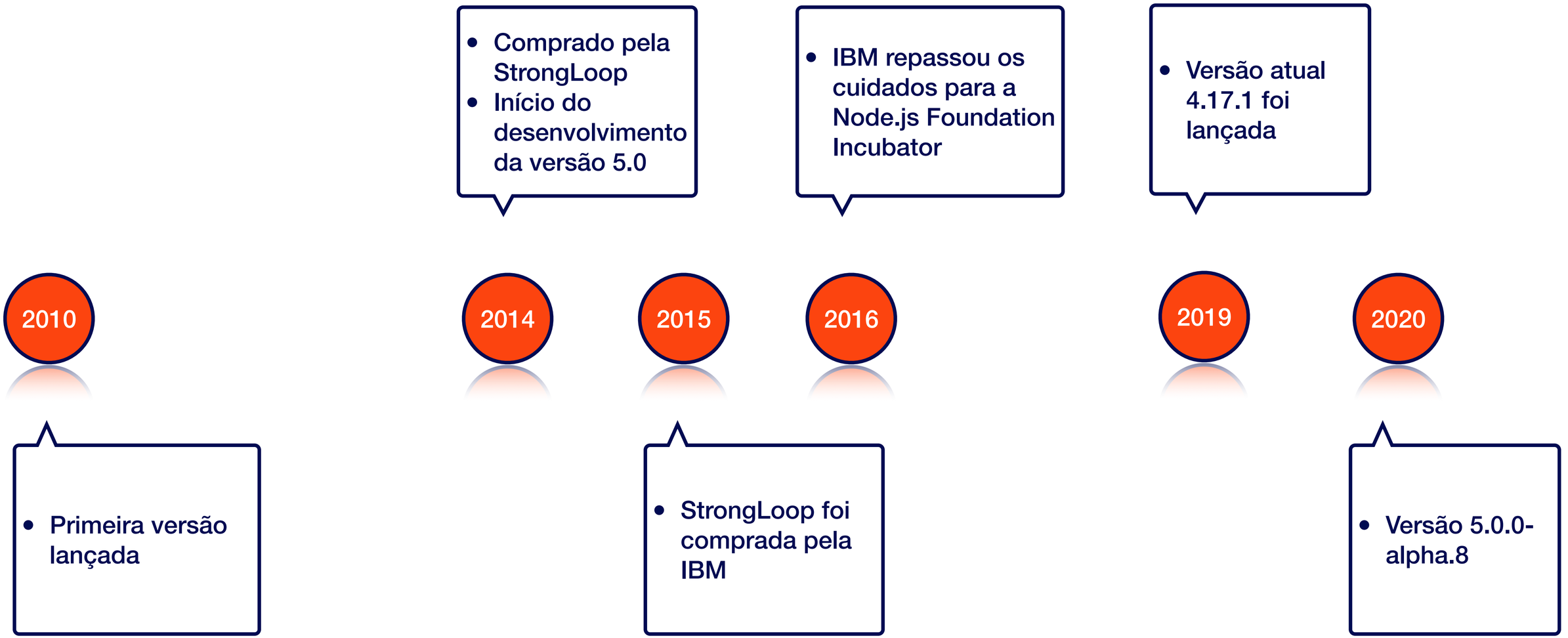
Introdução

Express

- “Um framework web para Node, rápido, minimalista e não opinativo”
- Um framework projetado para a criação de aplicações web e APIs utilizando o Node.js
  - Inspirado no Sinatra (Ruby)
- Facilita a organização da arquitetura das aplicações e facilita o desenvolvimento das mesmas
- Padrão de facto entre as opções de servidor web em Node

Introdução

História - Timeline



Introdução

Motivação

- Algumas tarefas comuns no desenvolvimento web não são suportadas diretamente pelo Node
  - Gerenciamento de recursos estáticos
  - Template engines
  - Suporte ao diversos métodos HTTP
  - Gerenciamento de Rotas
- Faz parte de umas das stacks mais utilizadas atualmente MEVN (Mongo, Express, Vue.js e Node)

Introdução

Características

- Open-source
- Light-weight server-side (minimalista)
- Sistema completo de rotas
- Tratamento de exceções dentro da aplicação
- Gerencia os diversos tipos (method) de requisições HTTP
- Da suporte a diversas “view engines”

Introdução

Vantagens

- Menor tempo de desenvolvimento
- Alta escalabilidade
- Flexibilidade
- Consistência entre as linguagens de backend e frontend
- Gerenciamento de requisições concorrentes
- Grande comunidade de desenvolvedores

Introdução

Desvantagens

- Muito trabalho manual
- Falta de padronização
  - A flexibilidade do Express é uma espada de dois gumes
  - Há pacotes de middleware para resolver quase qualquer problema
  - Utilizar os pacotes corretos para cada situação às vezes se torna um grande desafio
  - Não há "caminho certo" para estruturar um aplicativo

Criando um projeto com Express

Criando um projeto com Express

Criando um projeto

```
npm init --yes
npm install express
npm install -D typescript
npm install -D @types/node @types/express

npx tsc --init
npm install -D nodemon ts-node
```

Criando um projeto com Express

Configurando os scripts

- O seu primeiro programa em Express

```
{
  "scripts": {
    "build": "npx tsc",
    "start": "node dist/index.js",
    "dev": "nodemon src/index.ts"
  }
}
```

Criando um projeto com Express

Hello World

```
import express from 'express';
const app = express();
const PORT = 8000;
app.get('/', (req, res) => res.send('Express + TypeScript Server'));
app.listen(PORT, () => {
  console.log(`⚡[server]: Server is running at https://localhost:${PORT}`);
});
```

Importa o módulo do Express

Cria uma aplicação Express

Cria uma rota

Inicia o servidor

```
npm run dev
```

Criando um projeto com Express

O objeto app

Propriedade/Método	Descrição
app.set(name, value)	Define propriedades específicas da aplicação
app.get(name)	Recupera os valores definidos por meio da chamada app.set()
app.enable(name)	Habilitar um configuração na aplicação
app.disable(name)	Desabilita uma configuração na aplicação
app.enabled(name)	Verifica se uma configuração está habilitada
app.disabled(name)	Verifica se uma configuração está desabilitada
app.configure([env], callback)	Configura a aplicação condicionalmente de acordo com ambiente de desenvolvimento
app.use([path]	Carrega um middleware na aplicação
app.engine(ext, callback)	Regista um engine template na aplicação

Criando um projeto com Express

O objeto app

Propriedade/Método	Descrição
app.VERB(path, [callback...], callback)	Define uma rota de acordo com o método HTTP e como tratá-la
app.all(path, [callback...], callback)	Define uma rota para todos método HTTP e como tratá-la
app.locals	Armazena todas as variáveis visíveis em views
app.render(view, [options], callback)	Renderiza um view da aplicação
app.routes	A lista de todas as rotas da aplicação
app.listen	Realiza a ligação e passa a esperar por conexões

Criando um projeto com Express

O objeto request

Propriedade/Método	Descrição
req.params	Armazena os valores do parâmetros nomeados na rotaparameters
req.params(name)	Retorna o valor de parameters nomeados em rotas de GET ou POST
req.query	Armazena os valores enviados via GET
req.body	Armazena os valores enviados via POST
req.files	Armazena arquivos enviados via formulário de upload
req.route	Prover detalhes da rota atual
req.cookies	Armazena os valores dos cookies

Criando um projeto com Express

O objeto request

Propriedade/Método	Descrição
req.ip	O endereço IP do cliente
req.path	O path requisitado
req.host	O hostname contido no cabeçalho HTTP
req.protocol	O protocolo utilizado para realizar a requisição
req.secure	Verifica se a conexão é segura
req.url	A url requisitada junto com os parâmetros enviados na query

Criando um projeto com Express

O objeto response

Propriedade/Método	Descrição
res.status(code)	Define o código HTTP da resposta
res.set(field, [value])	Define campos no cabeçalho HTTP
res.get(header)	Recupera informação do cabeçalho HTTP
res.cookie(name, value, [options])	Define um cookie no cliente
res.clearCookie(name,	Deleta um cookie no cliente
res.redirect([status], url)	Redireciona o cliente para uma URL
res.location	O valor da localização presente no cabeçalho HTTP

Criando um projeto com Express

O objeto response

Propriedade/Método	Descrição
res.send([bodylstatus], [body])	Envia uma reposta HTTP com um código de resposta opcional
res.json([statuslbody], [body])	Envia um JSON como resposta HTTP com um código de resposta opcional
res.type(type)	Define o tipo da media da resposta HTTP
res.attachment([filename])	Informa presença de um anexo no cabeçalho HTTP Content-Disposition
res.sendfile(path, [options], [callback]])	Envia um arquivo para o cliente
res.download(path, [filename], [callback])	Solicita que o cliente baixe um arquivo
res.render(view, [locals], callback)	Renderiza uma view

Criando um projeto com Express

Rotas

- Manga Store
  - Listar todos os mangás
  - Adicionar novos mangás
  - Mostrar os detalhes de um mangá
  - Editar/Atualizar um mangá
  - Remover um mangá

Criando um projeto com Express

Rotas

Tarefa/Funcionalidade	HTTP Method	URL
Listar mangás	GET	/mangas
Formulário p/ adicionar um mangá	GET	/mangas/novo
Adicionar um mangá	POST	/mangas
Ver detalhes de um mangá	GET	/mangas/:id
Formulário p/ editar um mangá	GET	/mangas/:id/editar
Atualizar um mangá	PUT	/mangas/:id
Remover um mangá	DELETE	/mangas/:id

Criando um projeto com Express

Rotas

```
import express from 'express';
const app = express();
const PORT = 8000;
app.get('/', (req, res) => res.send('Express + TypeScript Server’));
app.get('/mangas', (req, res) => ???);
app.get(‘/mangas/novo', (req, res) => ???);
app.post('/mangas', (req, res) => ???);
app.get('/mangas/:id', (req, res) => ???);
app.get(‘/mangas:id/editar’, (req, res) => ???);
app.put('/mangas/:id', (req, res) => ???);
app.delete(‘/mangas/:id', (req, res) => ???);
app.listen(PORT, () => {
  console.log(`⚡[server]: Server is running at https://localhost:${PORT}
`);
});
```

Criando um projeto com Express

Rotas e parâmetros

- Inevitavelmente será preciso enviar informações via url
  - Id de uma entidade no banco de dados
  - Informações para filtrar os dados do banco de dados
  - Informação para realizar a paginação do resultado de uma consulta

```
Route path: /users/:userId/books/:bookId
Request URL: http://localhost:3000/users/34/books/8989
req.params: { "userId": "34", "bookId": "8989" }
```

```
Route path: /flights/:from-:to
Request URL: http://localhost:3000/flights/LAX-SFO
req.params: { "from": "LAX", "to": "SFO" }
```

Criando um projeto com Express

Roteadores

- Frequentemente chamados de mini-app
- Utilizados para lidar com rotas de maneira modular

```
import express from 'express';
const router = Router()
router.get('/', (req, res) => ???);
router.get('/novo', (req, res) => ???);
router.post('/', (req, res) => ???);
router.get('/:id', (req, res) => ???);
router.get(':id/editar', (req, res) => ???);
router.put('/:id', (req, res) => ???);
router.delete('/:id', (req, res) => ???);

module.exports = router
```

Criando um projeto com Express

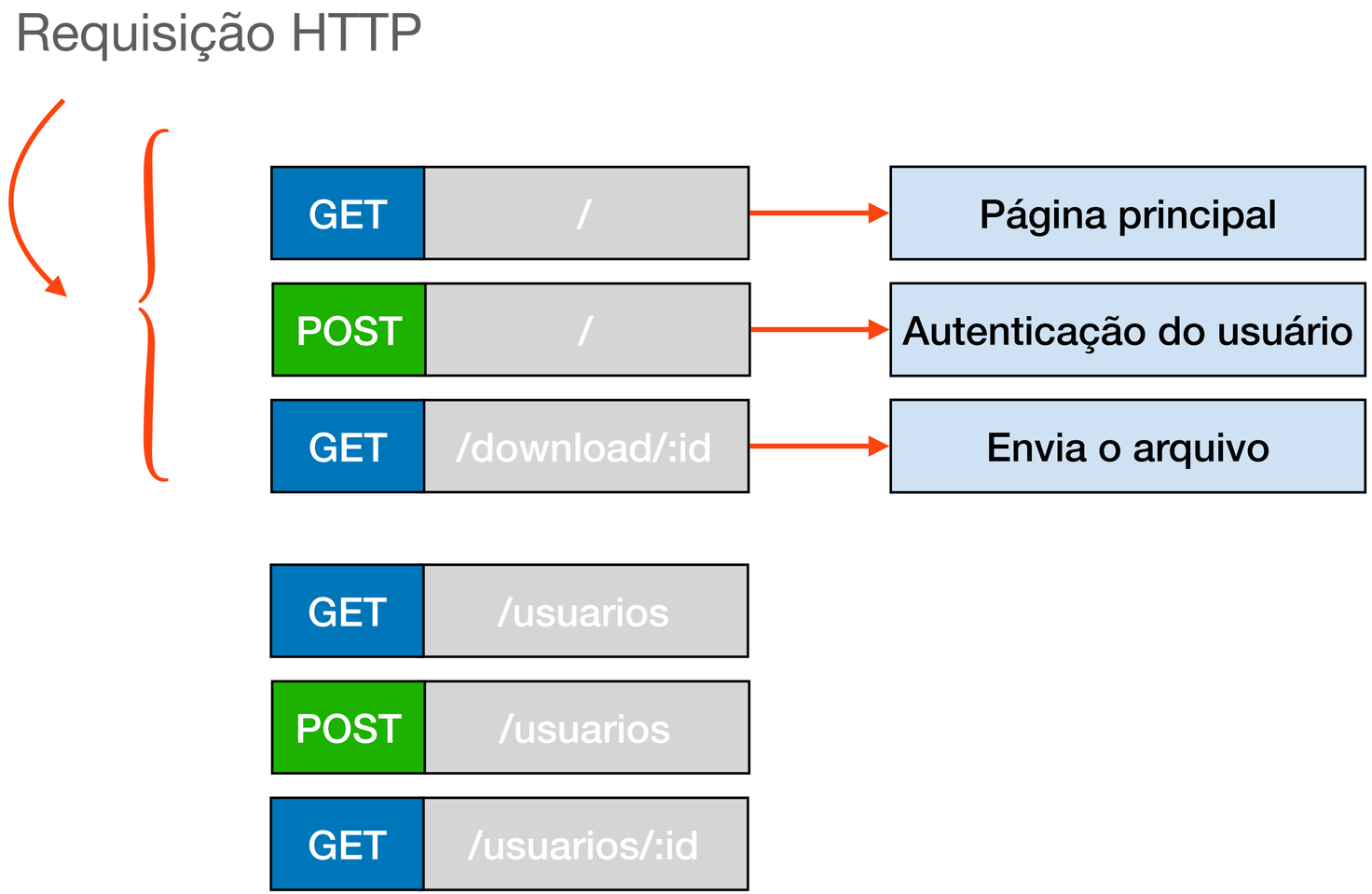
Roteadores

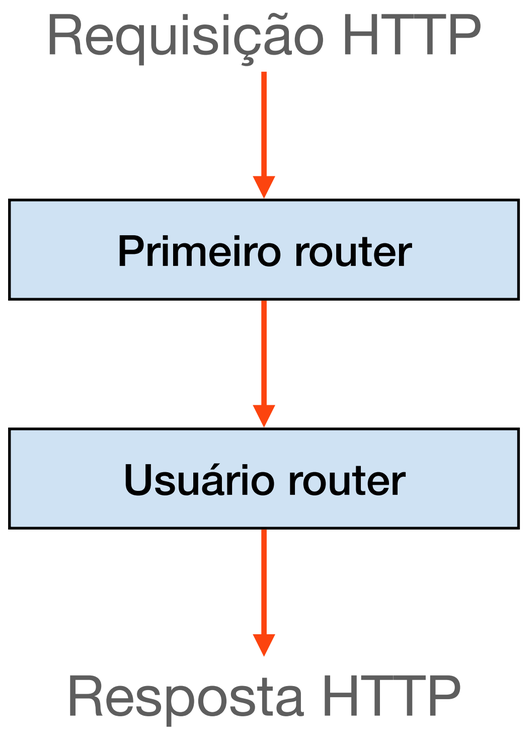
- Frequentemente chamados de mini-app

```
import express from 'express';
import userRouter from './routes/userRoutes'

const app = express();
const PORT = 8000;

app.get('/', (req, res) => res.send('Express + TypeScript Server’));

app.use('/usuarios', userRouter)

app.listen(PORT, () => {
  console.log(`⚡[server]: Server is running at https://localhost:${PORT}
`);
});
```

Criando um projeto com Express

Roteadores





Middlewares

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();
 });
```

Middlewares

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

Built-in middleware

- 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())
```

Middlewares

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

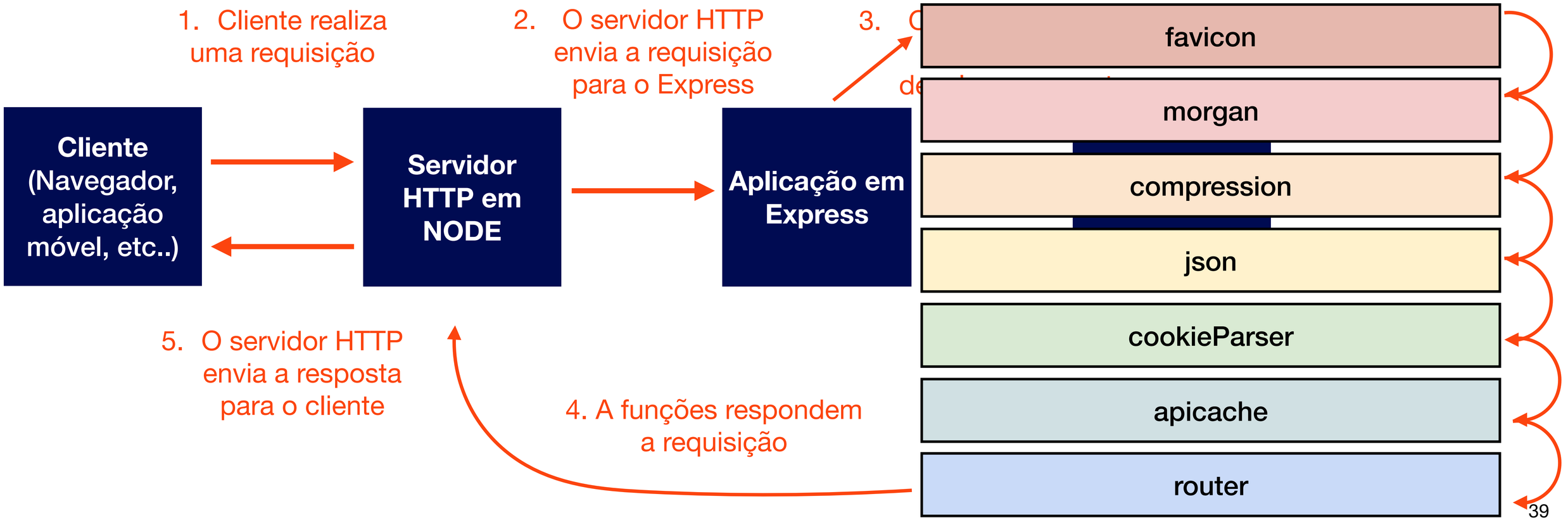
Disponíveis no Express

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

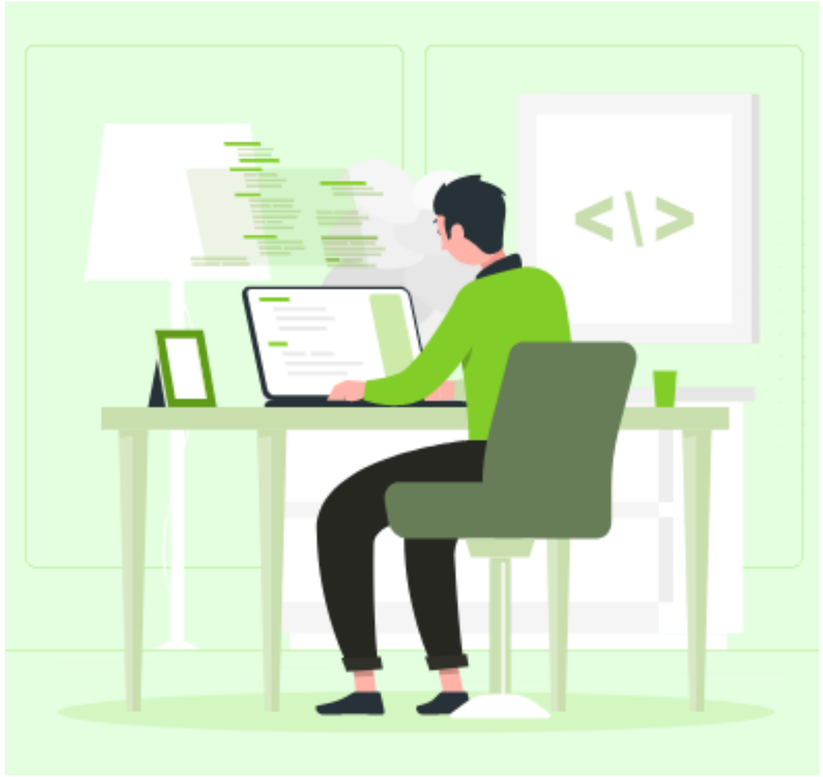
Middlewares

Fluxo da requisição

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



Criando um servidor web com Node & Express



Referências

- Express Web Application Development
- Introdução Express/Node
- How to set up TypeScript with Node.js and Express

Por hoje é só