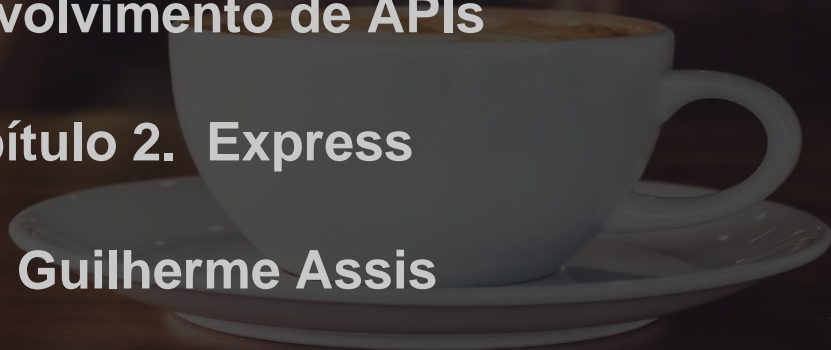


Desenvolvimento de APIs

Capítulo 2. Express

Prof. Guilherme Assis





Aula 2.1. Instalação

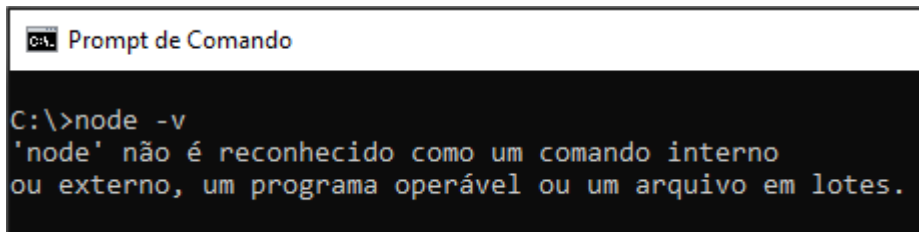
Nesta aula

☐ Introdução.

☐ Instalação.

- ExpressJS é um framework web para o Node.js.
- Desenvolvimento de aplicações mais rápido e fácil ao comparado com o desenvolvimento somente com o Node.js.
- Em seu site, ele se autodescreve com um framework web rápido, flexível e minimalista para o Node.js.
- Facilita o roteamento da aplicação, baseado nos métodos HTTP e URLs;
 - Roteamento se refere a como os endpoints respondem as solicitações.

- Antes de instalar é o Express, é necessário ter o Node.js instalado no computador.

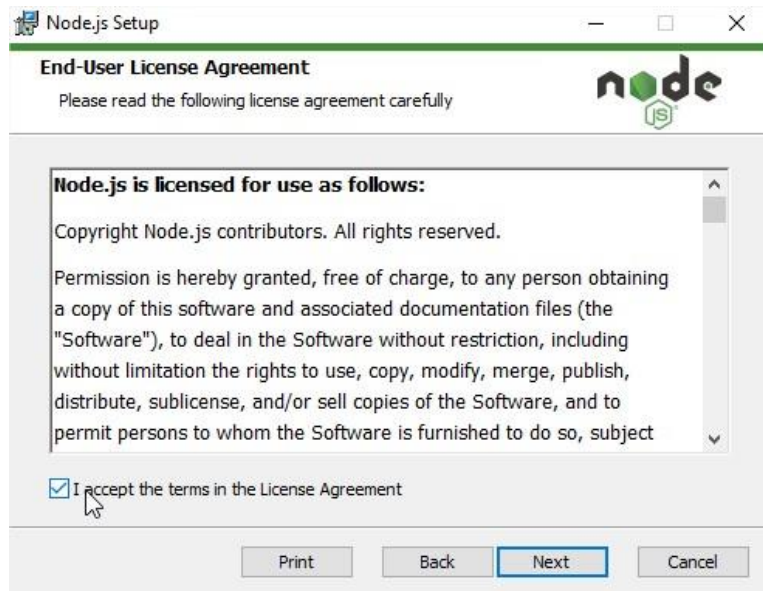
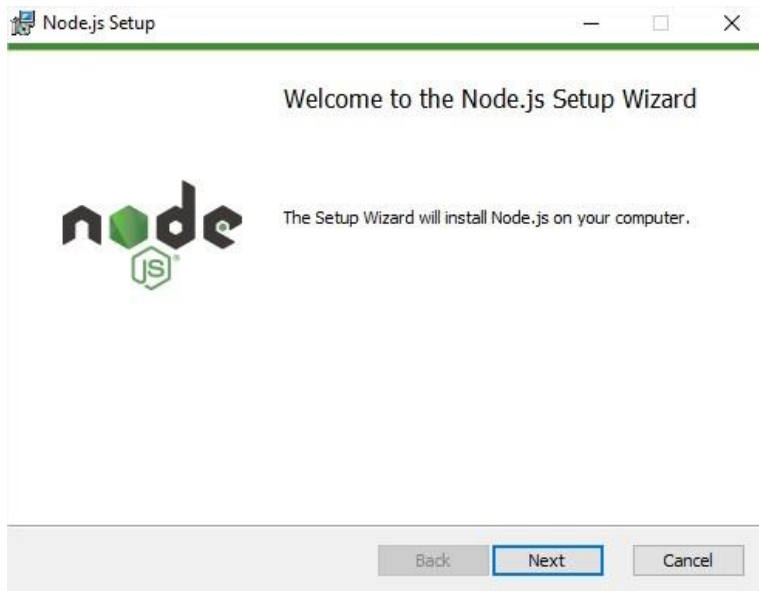


```
Prompt de Comando

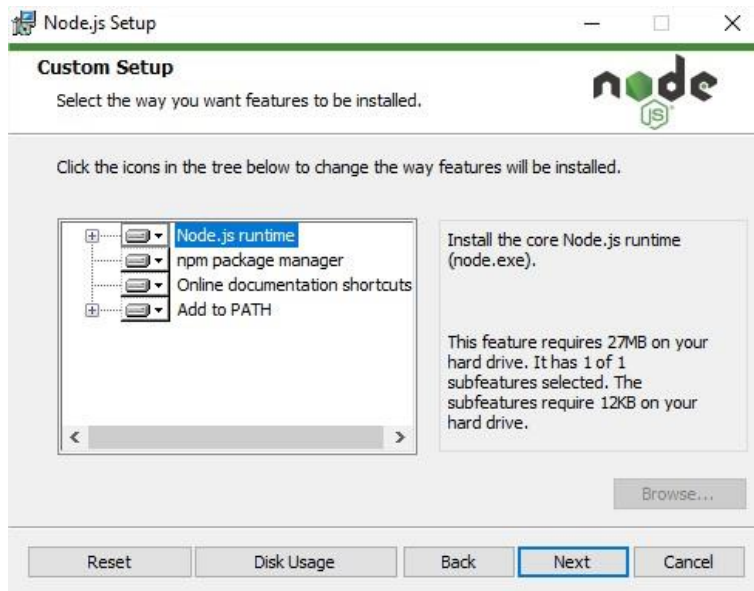
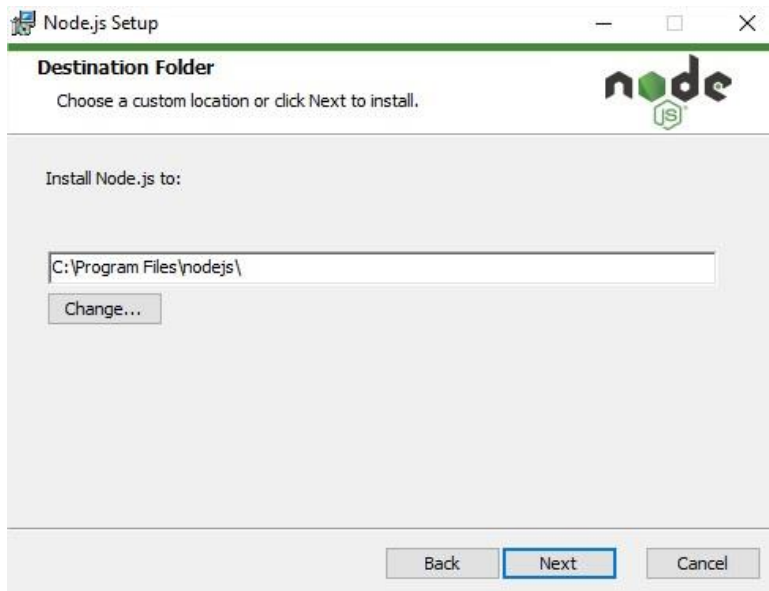
C:\>node -v
'node' não é reconhecido como um comando interno
ou externo, um programa operável ou um arquivo em lotes.
```

- Caso não tenha, baixar o instalador correspondente ao seu sistema operacional no site: <https://nodejs.org/en/>.

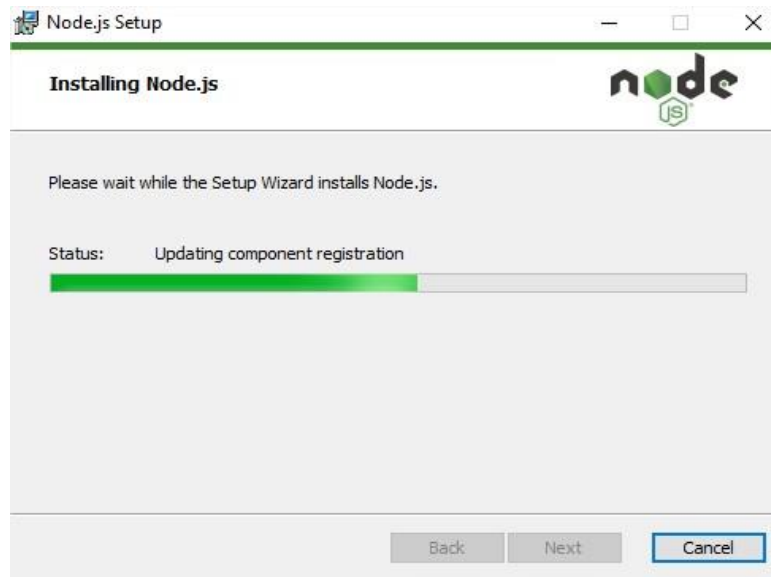
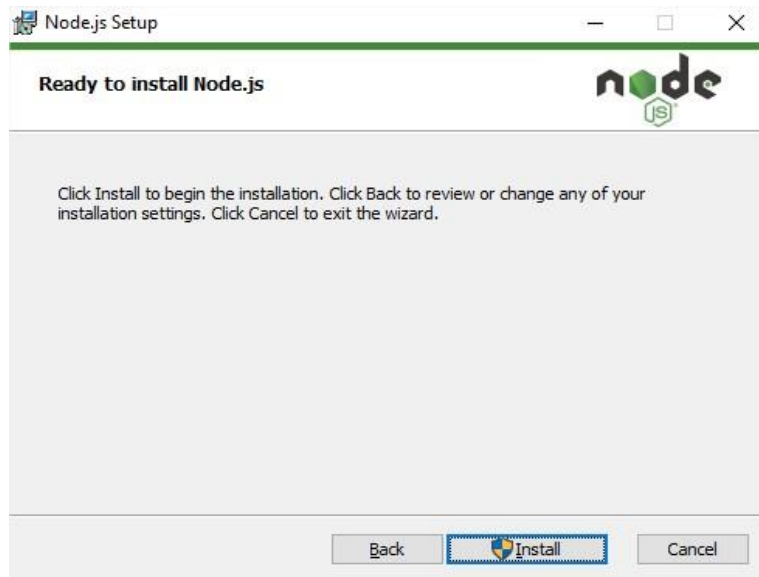
Instalação Node.js



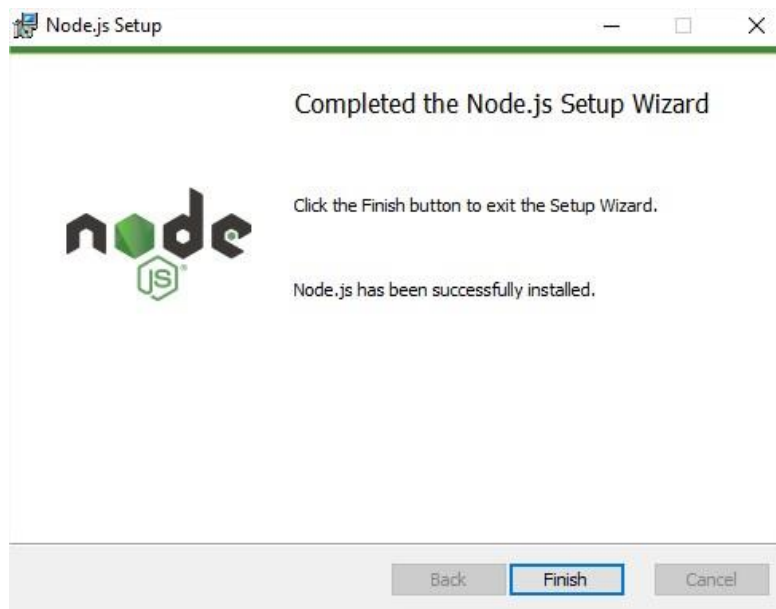
Instalação Node.js



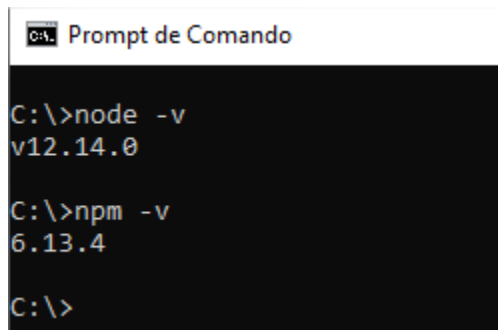
Instalação Node.js



Instalação Node.js



- Para conferir se a instalação deu certo, basta executar novamente o comando `node -v`.
- Verifique também se o NPM foi instalado através do comando `npm -v`. A instalação dele vem junto do instalador do Node.js.



```
C:\>node -v
v12.14.0

C:\>npm -v
6.13.4

C:\>
```

Instalação Express

- Para instalar o Express, primeiro é necessário iniciar um projeto Node e adicioná-lo como dependência.
- `npm init`
- `npm install express`

```
{ } package.json > ...
1  {
2    "name": "grades-control-api",
3    "version": "1.0.0",
4    "description": "",
5    "main": "src/app.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "author": "",
10   "license": "ISC",
11   "dependencies": {
12     "express": "^4.17.1"
13   }
14 }
15
```

Conclusão

☒ Introdução.

☒ Instalação.

Próxima aula

☐ Hello World.



Aula 2.2. Hello World

☐ Hello World.

- Uma rota define a forma como a aplicação responde a requisição de um cliente para determinado endpoint.
 - Composto por uma URI e pelo método HTTP da requisição;
 - `app.method(path, handler)`.
- Vamos criar um “Hello World” de exemplo no projeto criado anteriormente.

Conclusão

☒ Hello World.

- ❑ Configurações iniciais.

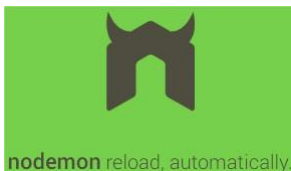


Aula 2.3. Configurações iniciais

- ❑ Configurações iniciais.

- Nos exemplos anteriores foi utilizado o CommonJS através do método `require` para importação e exportação de módulos.
- O padrão atual do JavaScript são os módulos do ECMAScript 6.
- Node.js está começando a suportar este modo, bastando habilitar uma flag no momento.
 - `node --experimental-modules src/app.js`
 - Incluir “type”: “module” no `package.json`

- Para debugar o projeto, é preciso acrescentar a flag “experimental-flags” dentro da propriedade “runtimeArgs” no arquivo launch.json.
- Para debugar o projeto basta inserir um breakpoint e clicar no símbolo de um inseto.
- A ferramenta Nodemon possibilita que o código seja reiniciado automaticamente quando houver alterações;
 - É possível desconsiderar alguns arquivos, como arquivos de log.



- ☑ Configurações iniciais.

Próxima aula

☐ Rotas.



Aula 2.4. Rotas

☐ Rotas.

- Vimos anteriormente um exemplo simples de criação de rota, a partir de uma instância do Express.

```
1  const express = require('express')
2  const app = express()
3  const port = 3000
4
5  app.get('/', (req, res) => res.send('Hello World!'))
6
7  app.listen(port, () => console.log(`App listening on port ${port}!`))
```

- Veremos outras opções possíveis para configuração de rotas com o Express.

- É possível capturar todos os métodos HTTP para o mesmo endpoint utilizando o método “all”.

```
app.all('/testAll', (req, res) => {  
  res.send(req.method);  
});
```

- O caminho para uma rota pode ser definido como uma string, um padrão de string ou uma expressão regular.
- No caso do padrão é possível utilizar os seguintes caracteres:
 - ?, +, * e ()
 - - e . são tratados normalmente

- “?”: indica que a letra imediatamente anterior a ele é opcional.
- “+”: indica que a letra imediatamente anterior a ela pode ser repetida diversas vezes naquela posição.
- “*”: indica que naquela posição pode ocorrer qualquer string.
- “()”: indica que a string dentro do parênteses será tratada como uma unidade.

- Através das rotas também é possível capturar parâmetros.
- Os parâmetros podem ser obtidos através da propriedade “params” do objeto da requisição.
- Ao definir uma rota que espera um parâmetro, basta colocar um “:” antes do nome do parâmetro.

- Uma outra característica das rotas é que possível fazer com que mais de uma função seja executada para determinada requisição.
- Elas são executadas na ordem que foram inseridas, e a execução passa para a próxima quando o método `next()` é invocado.

- Rotas que respondem ao mesmo endereço, mudando apenas o tipo do método HTTP podem ser agrupadas sob o método “route”.

- ☒ Método all.
- ☒ Caracteres especiais.
- ☒ Parâmetros na rota.
- ☒ Next.
- ☒ Route.

Próxima aula

☐ Middlewares.



Aula 2.5. Middlewares

☐ Middlewares.

- Funções de middleware são funções que tem acesso ao seguinte:
 - Objeto de solicitação (req);
 - Objeto de resposta (res);
 - Próxima função de middleware no ciclo da requisição e resposta do aplicativo (next).
- Podem executar qualquer código, fazer mudanças nos objetos de solicitação, encerrar o ciclo e chamar a próxima função de middleware na pilha.

- Ela pode ser utilizada para interceptar chamadas em específico ou qualquer chamada.
- Elas são as funções que são executadas quando determinada rota é atingida.

```
var express = require('express');  
var app = express();
```

O método HTTP para o qual a função de middleware é aplicada.

Caminho (rota) para o qual a função de middleware é aplicada.

A função de middleware.

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

Argumento de retorno de chamada para a função de middleware, chamado de "next" por convenção.

```
app.listen(3000);
```

Argumento de **resposta** HTTP para a função de middleware, chamado de "res" por convenção.

Argumento de **solicitação** HTTP para a função de middleware, chamado de "req" por convenção.

Fonte: <http://expressjs.com/>

- Nível da aplicação.
- Nível do roteador.

Conclusão

☒ Middlewares.

Próxima aula

☐ Tratamento de erros.



Aula 2.6. Tratamento de erros

- ☐ Tratamento de erros.

- Tratamento de erros é uma parte muito importante de uma API.
- Um erro pode ser originado a partir de vários pontos.
- É importante que a API seja capaz de se recuperar de um erro e informar adequadamente ao usuário o que ocorreu.
- O Express faz um tratamento padrão caso nenhum outro tenha sido especificado.

- Caso o erro tenha sido gerado a partir de um código assíncrono e deseje utilizar o tratamento padrão, é preciso passar o erro para o “next”.
- O Express permite que o desenvolvedor escreva as próprias funções para tratamento de erro;
 - Basta adicionar um quarto parâmetro na função de middleware.

- O middleware para tratamento de exceção deve ser configurado por último na instância do Express;
 - Assim ele receberá erros gerados em todas as definições anteriores.
- É permitido que exista várias funções para tratamento de erros;
 - Basta chamar o “next” passando o objeto de erro como parâmetro, para enviar o fluxo para a próxima função;
 - Neste caso a última função de tratamento deverá encerrar a requisição através do objeto de resposta.

- ☑ Tratamento de erros.
- ☑ Tratamento padrão.
- ☑ Tratamento em código assíncrono.
- ☑ Tratamentos customizados.
- ☑ Várias funções para tratamento.

■ Próxima aula

☐ Gravação de logs.



Aula 2.7. Gravação de logs

Nesta aula

- ☐ Gravação de logs.

- Uma funcionalidade muito importante para uma API é a gravação de logs.
- Métodos do console nativo do JavaScript, como `console.log`, `console.error` e `console.warn`:
 - Não é possível desativar os logs;
 - Não é possível definir o nível de logs;
 - Funções síncronas.
- Existem várias bibliotecas de log para o Node que tentam oferecer uma solução de log mais completa.

- Ele é uma biblioteca que suporta vários tipos de transporte.
- Permite a configuração de formatos de log.
- Suporta 7 níveis de log:
 - error: 0;
 - warn: 1;
 - info: 2;
 - http: 3;
 - verbose: 4;
 - debug: 5;
 - silly: 6.

Conclusão

☒ Gravação de logs.

☒ Winston.

- ☐ Servindo arquivos estáticos.



Aula 2.8. Servindo arquivos estáticos

- ☐ Servindo arquivos estáticos.

- Uma funcionalidade interessante do Express é que ele permite que sejam servidos arquivos estáticos.
- `express.static`, que recebe como parâmetro o diretório raiz de onde estão localizados os arquivos partindo da raiz da aplicação;
 - Pode-se utilizar este métodos várias vezes para servir vários diretórios;
 - Pode-se criar um diretório virtual, passando como parâmetro o nome desejado.

Conclusão

☒ Servindo arquivos estático.

- ❑ Capítulo 3 – Construção de uma API.