



Quem se prepara, não para.

Arquitetura de Aplicações Web

5º período

Professora: Michelle Hanne

Sumário

- Arquitetura REST
 - Frameworks REST
- NodeJS
 - NPM
 - Outras bibliotecas
 - Callback e Promises
 - Criando API com Node
- Atividade Learnyounode

Relembrando o HTTP

- Protocolo para comunicação entre cliente e servidor
- Modelo de requisição e resposta
- Gere recursos na web, que podem ser:
 - Páginas
 - Imagens
 - Scripts
 - Folhas de estilo
 - Fontes
 - Vídeo etc.
- Operações sobre recursos feitas pelos verbos HTTP (GET, POST...)
- Recursos identificados por URLs
- Prevê possibilidade de caching de recursos
- É totalmente stateless

Ideia do Rest

Dados como recursos

- Os dados são vistos como um recurso HTTP (assim como uma imagem, uma página HTML etc.)
- Cada informação exposta pelo banco de dados tem uma URL
- Operações (buscar, excluir, atualizar etc.) são realizadas na informação usando verbos HTTP (GET, DELETE, POST, PUT etc.)

Stateless (sem estado)

- Nenhum contexto é armazenado após o atendimento de uma requisição

“Cacheável”

- Clientes podem guardar as respostas, se interessante

Uniformidade de interface

- As operações e as URLs são padronizadas e fáceis de inferir

Ideia do Rest

API: conjunto de métodos públicos de um programa

API REST: conjunto de métodos públicos expostos por meio de um *web service* na arquitetura REST

- Proposto por Roy Fielding em 2000 (tese de doutorado)
- Abordagens: purista ou “inspirada”
- Como fazer?
 - Identifique os recursos de dados do banco
 - Identifique as operações sobre recursos que são permitidas
 - Implemente os métodos para cada recurso, respondendo possivelmente em vários formatos (.html, .json, .xml)



Arquitetura REST

Recursos:

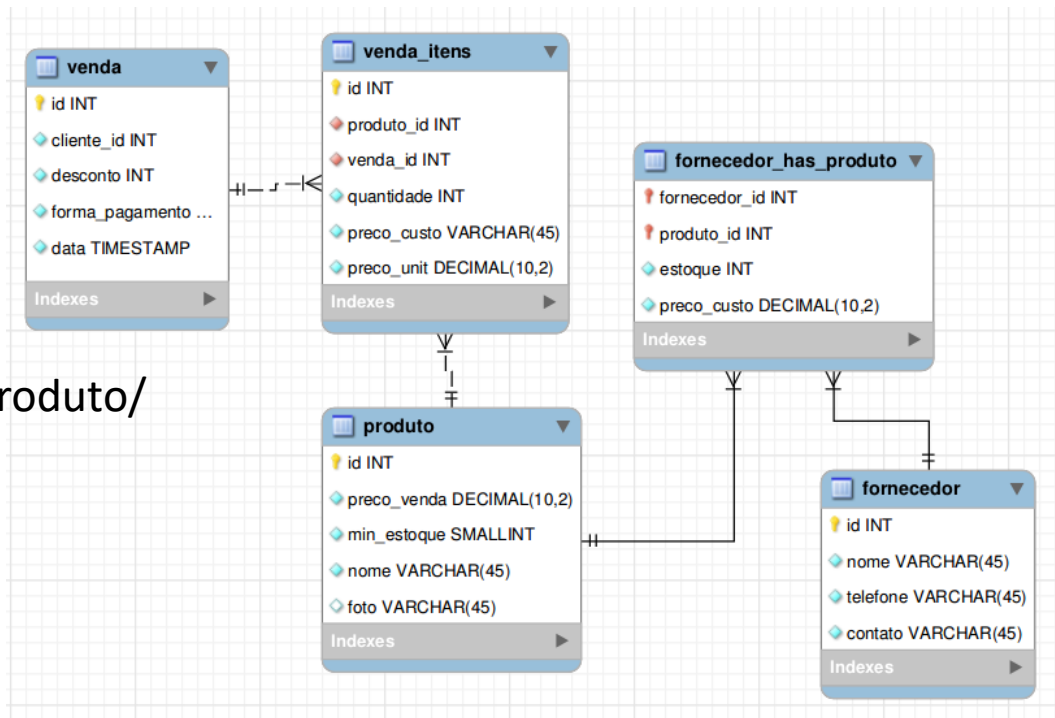
Produto (na URL /produto/)

Fornecedor (na URL /fornecedor/)

Estoque (na URL /estoque/)

Venda (na URL /venda/)

Por exemplo: controle-comercial.com/produto/



Fonte: Modelo de banco de dados disponível em <https://i.stack.imgur.com/IDNnG.png>

Arquitetura REST

Operações:

Listar todas os fornecedores (e.g., GET /fornecedor/)

Listar todos as vendas(e.g., GET /venda/)

Listar todos as produtos(e.g., GET /produto/)

Ver detalhes de um fornecedor (e.g., GET / fornecedor /50)

Inserir novo fornecedor(e.g., POST /fornecedor/)

Exclui um fornecedor completamente (e.g., DELETE / fornecedor /50)

Arquitetura REST

Usando express, podemos definir as rotas usando verbos HTTP: Vamos definir o método que lista todos os fornecedores:

```
app.get('/fornecedor/', async (req, res) => {  
  const [result] = await db.execute('SELECT * FROM fornecedor')  
  res.render('lista-fornecedor', { fornecedor: result }) })
```

Omitido aqui: (a) tratamento de erros, (b) paginação (se tiver) e (c) negociação de conteúdo (se tiver).

Arquitetura REST

A rota para detalhar um fornecedor:

```
app.get('/fornecedor/:id', async (req, res) => {  
  const id = db.escape(req.params.id) // pega o parâmetro "id" no caminho da  
  rota (eg, 4 em /fornecedor/4)  
  try {  
    const [result] = await db.execute(`SELECT * FROM fornecedor WHERE  
id=${id}`)  
    if (result.length === 0) {  
      throw new Error(`Ninguém conhece um zumbi com id ${id}.`)  
    }  
    res.render('detalhes-fornecedor', { fornecedor: result[0] })  
  } catch (error) {  
    res.send(404, "Fornecedor inexistente")  
  }  
})
```

Omitido negociação de conteúdo (se tiver).

Arquitetura REST

A rota para excluir um fornecedor:

```
app.delete('/fornecedor/:id', async (req, res) => {  
  const id = req.params.id  
  const [result] = await db.execute(`DELETE FROM fornecedor WHERE id=?`, id)  
  res.redirect('/fornecedor/') // ou 'back' para voltar à mesma URL de antes  
})
```

Omitido negociação de conteúdo (se tiver) e tratamento de erros.

Negociação de Conteúdo

É a ideia de servir diferentes representações de um mesmo recurso sob uma mesma URL. Por exemplo: mesmo recurso porém em idioma, formato ou codificação diferente.

Há diversos cabeçalhos HTTP de requisição que servem esse propósito:

- Accept
- Accept-Charset
- Accept-Encoding
- Accept-Language

O cliente especifica um ou mais deles e o servidor então escolhe qual representação enviar na resposta.

O padrão é que API REST costuma responder apenas em JSON.

Padrão de Rotas

Cada entidade que admita todas as operações CRUD:

- **GET /entidade** Lista todos os membros da entidade
- **POST /entidade** Cria nova entidade
- **GET /entidade/:id** Detalhes da entidade com certo id
- **HEAD /entidade/:id** Apenas cabeçalhos da resposta
- **DELETE /entidade/:id** Exclui a entidade com certo id
- **PUT /entidade/:id** Atualiza campos da entidade com certo id

Devem ser usados códigos de status do HTTP nas respostas. Exemplo:

200 Ok

201 Created

204 No content

400 Bad Request

401 Unauthorized

404 Not Found

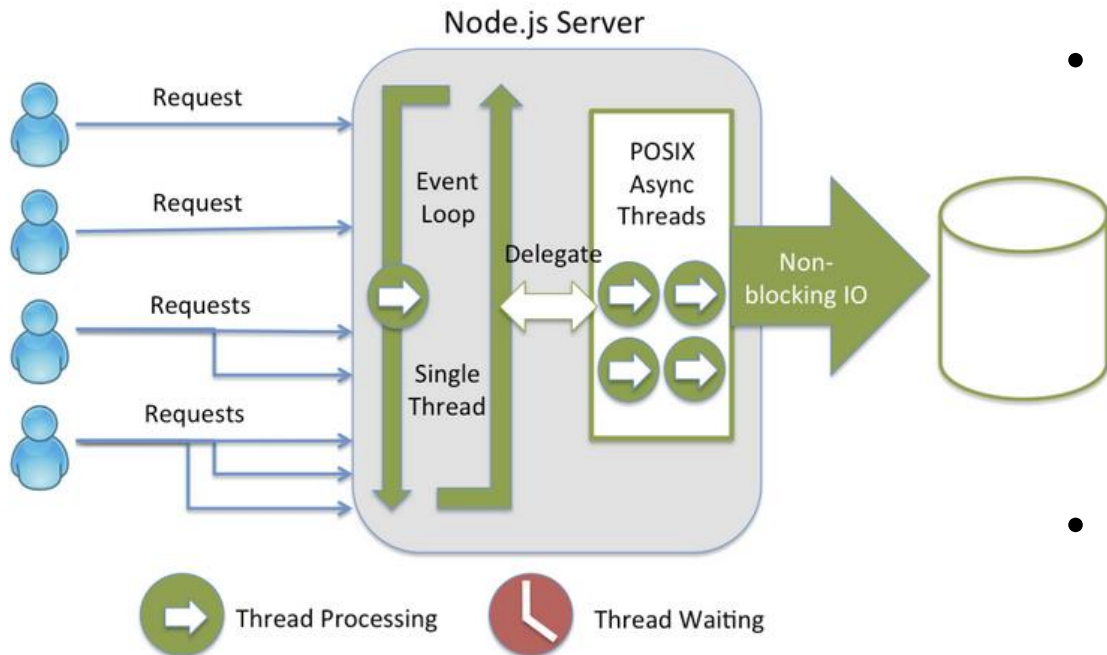
405 Method Not Allowed

Frameworks REST

	Linguagem	Baseada em recursos	Serialização	Autenticação	Cache
Django REST Framework	Python	Sim	XML, JSON, YAML, HTML	Básica, Token, Sessão, OAuth, OAuth2.0 e outros	Sim
Flask RESTful	Python	Sim	JSON	Não	Não
Restlet	Java	Sim	JSON, XML, CSV, YAML	Básica, Digest, Amazon S3, OAuth2.0	Não
Spark	Java	Não	Não	Não	Não
Sinatra	Ruby	Não	Não	Por meio de middleware Rack	Sim

- COMO CRIAR UMA API REST EM JAVA E SPRING BOOT PASSO A PASSO - <https://www.youtube.com/watch?v=0HKAqjiZveE>
- CRUD COMPLETO FLASK com SQLALCHEMY em Python - API REST e BANCO DE DADOS 2021 - <https://www.youtube.com/watch?v=WDpPGFki9UU>
- API com nodejs e express - <https://medium.com/xp-inc/https-medium-com-tiago-jlima-developer-criando-uma-api-restful-com-nodejs-e-express-9cc1a2c9d4d8>

Node.js



- O Node.js é um ***runtime environment*** de código aberto executado na **engine V8 do Chrome usando JavaScript**, com alto potencial de escalabilidade sem estar atrelado a plataformas ou dispositivos.
- Tecnologia usada para desenvolver aplicativos *server-side (back-end)*.

<https://www.youtube.com/watch?v=KtDwdoxQL4A>

Preparando o Ambiente

Instalação do Github desktop e do VS studio ou Atom

- <https://desktop.github.com/>
- <https://code.visualstudio.com/docs/?dv=win>
- <https://atom.io/>

Instalando o Node JS e o NPM

<https://nodejs.org/en/>

Testar se foi instalado corretamente

```

C:\Users\michelle_pc>node -v
v12.16.1

C:\Users\michelle_pc>npm -v
6.13.4

C:\Users\michelle_pc>
```

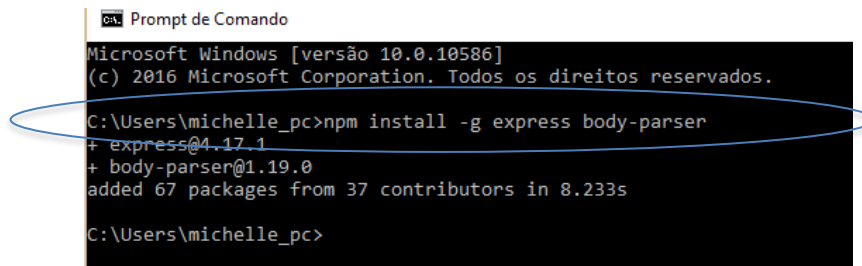
Instalando o Express.js

O Express permite criar aplicações web simples e outras aplicações.

Para instalar acesse o terminal e digite:

npm install -g express body-parser

O comando **npm install -g** instala o Express globalmente no seu sistema, acrescentar dependências com o comando **body-parser**.

A screenshot of a Windows Command Prompt window titled "Prompt de Comando". The text inside shows the command "C:\Users\michelle_pc>npm install -g express body-parser" being executed. The output indicates the successful installation of "express@4.17.1" and "body-parser@1.19.0", and that 67 packages were added from 37 contributors in 8.233s. A blue oval is drawn around the command line.

```
Prompt de Comando
Microsoft Windows [versão 10.0.10586]
(c) 2016 Microsoft Corporation. Todos os direitos reservados.

C:\Users\michelle_pc>npm install -g express body-parser
+ express@4.17.1
+ body-parser@1.19.0
added 67 packages from 37 contributors in 8.233s

C:\Users\michelle_pc>
```



- Ao instalar o Node.js, três programas são instalados:
 - O node, executador de arquivos JavaScript;
 - O npm e seu irmão mais novo npx
- O **NPM (*Node Package Manager*)** é um gerenciador de pacotes tipo Ruby Gems (ruby), NuGet (.NET), pip (python), maven/gradle (Java)
- A ideia do npm é:
 - Reutilizar programas JavaScript (📁 pacotes)
 - Gerenciar as dependências do seu projeto Node.js
 - Tornar seus programas/utilitários (i.e., pacotes) disponíveis para a comunidade

O que é um pacote

- É um “programa” Node.js
- Pode ser privado ou público (padrão)
- Quando é público, qualquer um pode instalá-lo e ver seu código fonte, caso ele esteja no GitHub, por exemplo.
- Uma pasta em seu computador é considerada um pacote se ela possui um arquivo chamado package.json:

```
{  "name": "bespoke-math",  
  "version": "1.2.0",  
  "dependencies": {  
    "katex": "^0.6.0"  
  }  
}
```

O que é um pacote

- Instalando pacotes com o npm
- Para instalar um pacote no diretório atual, usamos:

```
$ npm install <nome-do-pacote>
```

- Se quisermos instalar um pacote de forma global (acessível de qualquer lugar, como um programa executável):

```
$ npm install -g <nome>
```

- Ou então **npm install --global <nome>**

CommonJS

- Cada módulo tem o seu escopo e pode:
 - **require** coisas de outros módulos
 - **module.exports** suas próprias coisas
- Exemplo:

matematica.js

```
function fft(sinal) {  
  // transforma fourier  
  return ....;  
}  
  
module.exports = {  
  fft  
}
```

principal.js

```
const mat = require('matematica.js')  
  
console.log(mat.fft([...]))
```

Express

- Se intitulam um web framework para Node.js:
 - Rápido
 - Não opinativo (unopinionated)
 - Minimalista
 - Site oficial: <http://expressjs.com/>
- Instalando:
- `$ npm install express --save`

Servidor “*hello world*” com Express

```
import express from 'express'
const app = express()

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

const server = app.listen(3000, () => {
  const host = server.address().address;
  const port = server.address().port;

  console.log(`Listening at http://${host}:${port}`);
});
```

Express - Rotas

- O express facilita a especificação da ação a ser tomada dependendo da URL solicitada
- Uma rota é um verbo HTTP (GET, POST etc.) e uma URL
- A cada rota é associada uma callback:

```
// GET / (página inicial)
app.get('/', (request, response) => {
  response.render('index')    // desenha a view 'index'
})
// POST /contato
app.post('/contato', (request, response) => {
  // envia um email usando os dados enviados
})
```


Gerando HTML Dinamicamente

- Queremos poder gerar HTML dinamicamente. Para isso, precisamos de uma linguagem que facilite isso, ao mesmo tempo que possibilite a separação de código HTML do código nessa linguagem
- Para ser “não opinativo”, o Express não impõe uma linguagem específica:
- **ejs (.ejs, era o formato original do Express)**
- **handlebars (.hbs, [site oficial](#))**
- **pug (.pug, [site oficial](#))**

Gerando HTML Dinamicamente (EJS)

- Gerando HTML dinamicamente com ejs
- Usamos ejs (ou qualquer outro templating engine) em 3 passos:
 - Configuramos o express
 - Escrevemos arquivos HTML no formato ejs
 - Para determinadas rotas, renderizamos views
- (1) Para configurar o Express para usar ejs:
- `app.set('view engine', 'ejs')`
- (2) Escrevemos arquivos no formato .ejs em vez de .html. Trecho de um arquivo, e.g., `equipe.ejs`:

```
<ul>
```

```
<% for (let i = 0; i < users.length; i++) { %>
```

```
<li><%= users[i].nome %></li>
```

```
<% } %>
```

```
</ul>
```

Esses arquivos são chamados de views e devem ficar dentro de uma pasta que configurarmos (valor padrão: `./views`)

```
app.set('views', 'arquivos_ejs');
```

Gerando HTML Dinamicamente (EJS)

- Gerando HTML dinamicamente com ejs
- Usamos ejs (ou qualquer outro templating engine) em 3 passos:
 - Configuramos o express
 - Escrevemos arquivos HTML no formato ejs
 - Para determinadas rotas, renderizamos views
- (1) Para configurar o Express para usar ejs:
- `app.set('view engine', 'ejs')`
- (2) Escrevemos arquivos no formato .ejs em vez de .html. Trecho de um arquivo, e.g., `equipe.ejs`:

```
<ul>
  <% for (let i = 0; i < users.length; i++) { %>
    <li><%= users[i].nome %></li>
  <% } %>
</ul>
```

Gerando HTML Dinamicamente (EJS)

- (3) Ao definirmos os handlers das nossas rotas, chamamos `response.render` e passamos o nome do arquivo da view que deve ser usado (sem a extensão):

```
app.get('/equipe', function(request, response) {  
  response.render('equipe', contexto);    // vai pegar arquivos_ejs/equipe.ejs  
});
```

- É possível (e muito comum) disponibilizar dados para a view e podemos fazer isso usando o segundo parâmetro de `response.render`:

```
response.render('equipe', {  
  users: [  
    { nome: 'TJ Holowaychuk', foto: 'tj.jpg' },  
    { nome: 'Douglas Wilson', foto: 'dcw.jpg' }  
  ]  
});
```

DICA:

<http://expressjs.com/en/guide/using-template-engines.html>

Callback

A programação assíncrona não usa o retorno da função para informar que a função foi finalizada. Ela trabalha com o “estilo de passagem de continuação (continuation-passing style), CPS”. Em definição, uma função escrita neste estilo recebe como argumento uma “continuação explícita”. Quando a função resulta um valor, ela “retorna” pela chamada da função de continuação com este valor via argumento. As funções de continuação explícitas também são conhecidas como *Callback*.

Promises

Basicamente, uma vez que uma *promise* é chamada, ela inicia em **pending state (pendente)**. Isto significa que a função **caller** (que chamou a promise) continua sua execução enquanto espera pela promise terminar seu próprio processamento e retornar ao caller com algum feedback. Quando esse retorno acontece, a **promise** é **retornada em resolved state ou rejected state**.

Promises

Quando precisamos realizar várias chamadas assíncronas, podemos ter um callback hell: várias callbacks aninhadas

- Dificulta a leitura e escrita
- Suscetível a erros do programador
- Trata erros apenas por callback, dificultando a legibilidade/manutenibilidade do código
- Soluções

uso de promessas explicitamente ou com `async/await`

Promises

Uma promise é um objeto “**thenable**”, i.e., podemos invocar **.then**, passando uma função que só será chamada quando a promessa for cumprida (com êxito ou falha):

- **.then(callbackSuccess, callbackError)** pode receber 2 funções
- ...ou podemos **usar .catch** para tratar o erro de uma “**promise chain**” de forma genérica

É possível criar objetos do tipo **Promise** de forma que nós definimos quando elas estão resolvidas (com sucesso ou falha)

Problemas com Promessas

- Promessas com `.then` encadeados reduzem (mas não acabam) com callback hell 🔥
- Há possibilidade de `.catch` não capturar exceção.
- Caso `.catch` seja atrasado (eg, devido a alguma espera na criação da promessa - exemplo).
- É difícil escrever um fluxo condicional em uma cadeia de promessas.
- Depurar ainda fica um pouco difícil com promessas.
- É possível aumentar a legibilidade, se o código parecer síncrono.

AWAIT


- await substitui o .then
- Parece síncrono, mas suspende execução até a promessa ser cumprida.
- E isso não bloqueia a execução do processo (ie, é assíncrono).

```
function espera(tempo) {  
  return new Promise(resolver => {  
    setTimeout(resolver, tempo*1000)  
  })  
}
```

```
console.log('tempo = 0')  
await espera(2) // <--  
console.log('tempo = 2')  
// tempo = 0  
// ...  
// tempo = 2 (2s depois)
```

```
console.log('tempo = 0')  
espera(2).then(() => console.log('tempo = 2'))  
// tempo = 0  
// ...  
// tempo = 2 (2s depois)
```

Retorno de Await

- O valor que é resolvido pela Promise é retornado pela função
-  Exemplo: pegar dados de notícias, criar um template e mostrar
- 3x operações assíncronas em sequência:

```
mostraNoticia() << formata() << dados('noticia')
```

```
const db = { /*...*/ }  
function dados(entidade) {  
  return new Promise(resolver => {  
    // faz algo assíncrono (eg, acessa banco ou ajax)  
    // e resolve (cumpre) a promessa com o resultado  
    resolver(db[entidade])  
  })  
}
```

```
// com await 🚀  
const noticia = await dados('noticias')  
const template = await formata(noticia)  
mostraNoticia(template)
```

```
// com promise.then  
dados('noticias')  
  .then(formata)  
  .then(mostraNoticia)
```

Tratando Erros

- JavaScript possui ***try / catch***, mas eles não funcionam com **Promise**
- É necessário usar **.catch** ou passar uma **errorCallback** como 2º argumento para **.then**
- Se usarmos **await**, podemos usar ***try / catch*** sem problemas:

Se usarmos `await`, podemos usar `try / catch` sem problemas:

// com await 🍌

```
try {  
  const noticia = await dados('noticias')  
  const template = await formata(noticia)  
  mostraNoticia(template)  
} catch (erro) {  
  mostraUmaPropaganda()  
  console.error(erro)  
}
```

// com promise.catch

```
dados('noticias')  
  .then(formata)  
  .then(mostraNoticia)  
  .catch(erro => {  
    mostraUmaPropaganda()  
    console.error(erro)  
  })
```

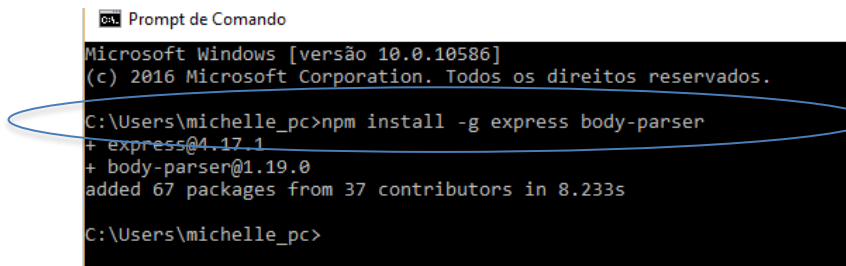
Criando uma API Node com Express.js

O Express permite criar aplicações web simples e outras aplicações.

Passo 1: Para instalar acesse o terminal e digite:

npm install -g express body-parser

O comando **npm install -g** instala o Express globalmente no seu sistema, acrescentar dependências com o comando **body-parser**.



```
cmd Prompt de Comando
Microsoft Windows [versão 10.0.10586]
(c) 2016 Microsoft Corporation. Todos os direitos reservados.

C:\Users\michelle_pc>npm install -g express body-parser
+ express@4.17.1
+ body-parser@1.19.0
added 67 packages from 37 contributors in 8.233s

C:\Users\michelle_pc>
```

Criando uma API Node com Express.js

Passo 2: Criando a pasta da API

Ainda no terminal, crie um diretório com o nome da sua aplicação e vá até o diretório criado.

```
C:\> Prompt de Comando

D:\testes_node>mkdir api-node

D:\testes_node>cd api-node

D:\testes_node\api-node>_
```

Criando uma API Node com Express.js

Passo 2: Criando a pasta da API

Ainda no terminal, crie um diretório com o nome da sua aplicação e vá até o diretório criado.

```
C:\> Prompt de Comando

D:\testes_node>mkdir api-node

D:\testes_node>cd api-node

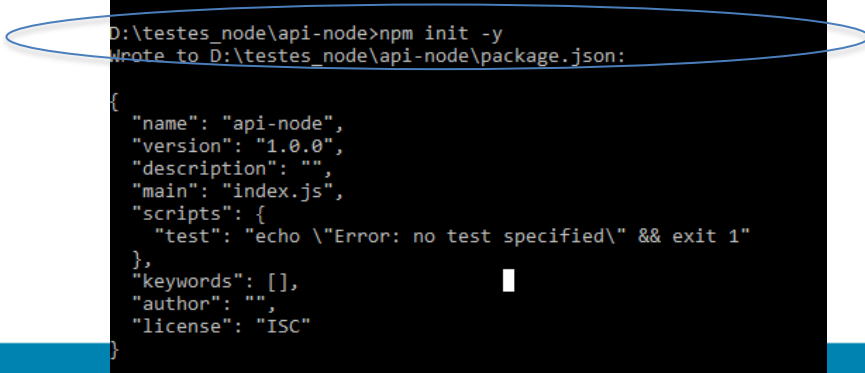
D:\testes_node\api-node>_
```

Criando uma API Node com Express.js

Passo 3: Baixe o módulo do node na sua pasta. Cria uma estrutura de diretórios dentro de node_modules.

```
npm install express
```

Passo 4: Criar o arquivo de configurações package.json



```
D:\testes_node\api-node>npm init -y
Wrote to D:\testes_node\api-node\package.json:

{
  "name": "api-node",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```


Criando uma API Node com Express.js

Passo 5: Criando o arquivo **app.js**

Agora, dentro do diretório **api-node**, abra o seu editor de textos preferido (ex. Visual Studio e crie um arquivo chamado **app.js**. Vamos precisar usar o pacote do Express que foi instalado anteriormente para poder criar a nossa API. Vejamos como vai ficar o nosso arquivo **app.js**:

Criando uma API Node com Express.js

```
var express = require('express');
var bodyParser = require('body-parser');
var app = express();

app.use(bodyParser.json());

app.get('/maps', function(req, res) {
  var dados = [
    {
      lat: -25.470991,
      lon: -49.271036
    },
    {
      lat: -0.935586,
      lon: -49.635540
    },
    {
      lat: -2.485874,
      lon: -43.128493
    }
  ];

  res.send(JSON.stringify(dados));
});

app.listen(8000, function() {
  console.log('Servidor rodando na porta 8000.');
```

Importa os pacotes instalados.

body-parser será o responsável por parsear as requisições via JSON.

configura o Express e atribui à variável app a ele

configura a forma como iremos tratar nossas requisições no Express.

rotear a URL /maps via método GET:

Quando o usuário navegar até a rota /maps, nosso servidor vai retornar um array de objetos contendo informações com latitude e longitude.

configurar uma porta onde nosso servidor web ficará escutando as chamadas HTTP.

Criando uma API Node com Express.js

Passo 6: Executar o arquivo app.js e testar no navegador

```
D:\testes_node\api-node>node app.js  
Servidor rodando na porta 8000.
```

<http://localhost:8000/maps>

← → ↻ ⓘ localhost:8000/maps

[{"lat":-25.470991,"lon":-49.271036},{"lat":-0.935586,"lon":-49.63554},{"lat":-2.485874,"lon":-43.128493}]

Learnyounode



- <https://nodeschool.io/pt-br/>
- Workshop que ensina conceitos sobre o Node

A screenshot of a Windows PowerShell terminal window. The title bar says "Windows PowerShell". The terminal has a blue background with white text. At the top, it says "LEARN YOU THE NODE.JS FOR MUCH WIN!" and "Select an exercise and hit Enter to begin". Below this is a list of exercises, each preceded by a right arrow and followed by "[COMPLETED]":
» HELLO WORLD [COMPLETED]
» BABY STEPS [COMPLETED]
» MY FIRST I/O! [COMPLETED]
» MY FIRST ASYNC I/O! [COMPLETED]
» FILTERED LS [COMPLETED]
» MAKE IT MODULAR
» HTTP CLIENT
» HTTP COLLECT
» JUGGLING ASYNC
» TIME SERVER
» HTTP FILE SERVER
» HTTP UPPERCASER
» HTTP JSON API SERVER
At the bottom, there are three options: "HELP", "CREDITS", and "EXIT".

Para instalar o learnyounode, faremos de forma global com o npm:

\$ npm install -g learnyounode

Para executar:

\$ learnyounode



- Fazer os exercícios de 1 ao número 6

Referências

<https://fegemo.github.io/cefet-web/classes/js7>

<https://medium.com/balta-io/nodejs-async-await-21ca3636252a>

<https://fegemo.github.io/cefet-web/classes/ssn6>