




# Hands On

---



# Gregory Silveira Lagranha

Engenheiro de Software 

Formando em Engenharia de Software - **PUCRS**

AGES I 

AGES II 

AGES III 

AGES IV



[gregory@triider.com](mailto:gregory@triider.com)

[gregory.lagranha@acad.pucrs.br](mailto:gregory.lagranha@acad.pucrs.br)



# O que será abordado?

- O que é Node.js?
- O que são NPM e Yarn?
- Qual a diferença entre Javascript, ECMAScript e Typescript?
- O que é API, REST e RESTful?
- Frontend X Backend. Onde está o Node.js?

# Node.js

- Interpretador de Javascript assíncrono orientado a eventos
- Criado por Ryan Dahl em 2009
- Tem como características principais: **Alta escalabilidade, flexibilidade e baixo custo**
- Utilizado por grandes players do mercado como Netflix, Uber, LinkedIn, Walmart, entre outros



# NPM / Yarn

- NPM (Node Package Manager) e Yarn são gerenciadores de pacotes/dependências do Node.js
- Existem outros gerenciadores de pacotes, porém, NPM e Yarn são os mais conhecidos
- Esses pacotes aceleram o desenvolvimento de aplicações, fazendo com que o desenvolvedor tenha um vasto repositório de código para ser reaproveitado





JavaScript™



# Javascript

- Não tem relação com Java
- Foi criada em 1995 por Brendan Eich
- Linguagem de programação de alto nível com tipagem dinâmica fraca
- Utilizada em conjunto com HTML e CSS, permite a criação de conteúdo que se atualiza dinamicamente





# ECMAScript

- Hoje, Javascript é na verdade ECMAScript



ES

# ECMAScript

- ECMAScript é uma linguagem de programação de uso geral padronizada pela Ecma International, de acordo com o documento ECMA-262
- Surgiu com o objetivo de normatizar a linguagem Javascript, porém, como o nome Javascript já era patenteado pela Sun Microsystem (hoje Oracle), foi batizada como ECMAScript
- A versão 6 do ECMAScript (ES6 / ES2015) foi a versão que teve o maior número de mudanças na linguagem
- O ECMAScript está atualmente na versão 11 (ES11 / ES2020)



```
1  var variavel1 = ""; // var ou let
2  variavel1 = "Hello";
3
4  function getHelloWorld() {
5      const space = " "; // Não pode ser alterado
6      let variavel2 = "";
7      variavel2 = "world";
8      return variavel1 + space + variavel2;
9  }
10
11  // console.log(space); // Erroooooou
12  // console.log(variavel2); // Erroooooou
13
14  console.log(getHelloWorld());
```

```
~/puc/hand-on-nodejs
> node index.js
Hello world
```

ES

```
const object = {  
  name: "Gregory",  
  lastName: "Lagranha",  
  birthday: new Date("1993-10-27").toISOString(),  
  active: true  
};  
  
console.log('object \n', object);
```

~/puc/hand-on-nodejs

> node index.js

object

```
{ name: 'Gregory',  
  lastName: 'Lagranha',  
  birthday: '1993-10-27T00:00:00.000Z',  
  active: true }
```

ES

```
const array = ["item 1", "item 2", "item 3"];

for (let i = 0; i < array.length; i++) {
  console.log("for loop", array[i]);
}

let j = 0;
while (j < array.length) {
  console.log("while loop", array[j]);
  j++;
}

array.forEach((a) => console.log("forEach loop", a));
```

~/puc/hand-on-nodejs

```
> node index.js
for loop item 1
for loop item 2
for loop item 3
while loop item 1
while loop item 2
while loop item 3
forEach loop item 1
forEach loop item 2
forEach loop item 3
```

ES

```
console.log("1 == '1' > ", 1 == '1');  
console.log("1 === '1' > ", 1 === '1');  
console.log("1 === parseInt('1') > ", 1 === parseInt('1'));
```

**hands-on-nodejs** on  **master** **[?]**

```
> node index.js  
1 == '1' > true  
1 === '1' > false  
1 === parseInt('1') > true
```

**ES**

```
function soma(n1, n2) {  
  return n1 + n2;  
}  
  
const somaArrowFunction = (n1, n2) => n1 + n2;  
// const somaArrowFunction = (n1, n2) => {  
//   return n1 + n2  
// };  
  
console.log('soma(1, 2) > ', soma(1, 2));  
console.log('somaArrowFunction(1, 2) > ', somaArrowFunction(1, 2));
```

**hands-on-nodejs** on  master **[?]**

```
> node index.js  
soma(1, 2) > 3  
somaArrowFunction(1, 2) > 3
```

**ES**

```
function sum (n1, n2, callback) {  
  const n = n1 + n2;  
  return callback(n);  
}  
  
function checkIfGreaterThan10(n1) {  
  return n1 > 10;  
}  
  
console.log('Callback example > ', sum(2, 9, checkIfGreaterThan10));
```

```
hands-on-nodejs on ↗ master [?]  
> node index.js  
Callback example > true
```

ES



```
function asyncMulti (n1, n2) {  
  return new Promise((resolve, reject) => {  
    setTimeout(() => {  
      if (typeof n1 === 'number' && typeof n2 === 'number') {  
        resolve(n1 * n2);  
      } else {  
        reject("Error")  
      }  
    }, 3000)  
  })  
}
```

```
console.log(asyncMulti(2, 2))
```

**hands-on-nodejs** on  **master** **[?]**

```
> node index.js
```

```
Promise { <pending> }
```

**ES**

```
function asyncMulti (n1, n2) {  
  return new Promise((resolve, reject) => {  
    setTimeout(() => {  
      if (typeof n1 === 'number' && typeof n2 === 'number') {  
        resolve(n1 * n2);  
      } else {  
        reject("Ops! Ocorreu um erro")  
      }  
    }, 3000)  
  })  
}  
  
asyncMulti(2, 2).then(result => console.log('Resultado: ', result))  
asyncMulti(2, 'A')  
  .then(result => console.log('Resultado: ', result))  
  .catch(error => console.log('Erro: ', error))
```

```
hands-on-nodejs on ↗ master [?] took 3s  
> node index.js  
Resultado: 4  
Erro: Ops! Ocorreu um erro
```

ES

```
async function main() {  
  try {  
    const [result1, result2] = await Promise.all([asyncMulti(2, 2), asyncMulti(2, 'A')])  
    console.log('Resultado 1: ', result1)  
    console.log('Resultado 2: ', result2)  
  } catch (error) {  
    console.log('Erro: ', error)  
  }  
}  
  
main();
```

```
hands-on-nodejs on ↗ master [?] took 3s  
> node index.js  
Erro: Ops! Ocorreu um erro
```

ES

```
async function main() {  
  try {  
    const result1 = await asyncMulti(2, 2);  
    console.log('Resultado 1: ', result1)  
    const result2 = await asyncMulti(2, 'A');  
    console.log('Resultado 2: ', result2)  
  } catch (error) {  
    console.log('Erro: ', error)  
  }  
}  
  
main();
```

**hands-on-nodejs** on  master [?]

**> node index.js**

Resultado 1: 4

Erro: Ops! Ocorreu um erro

**ES**

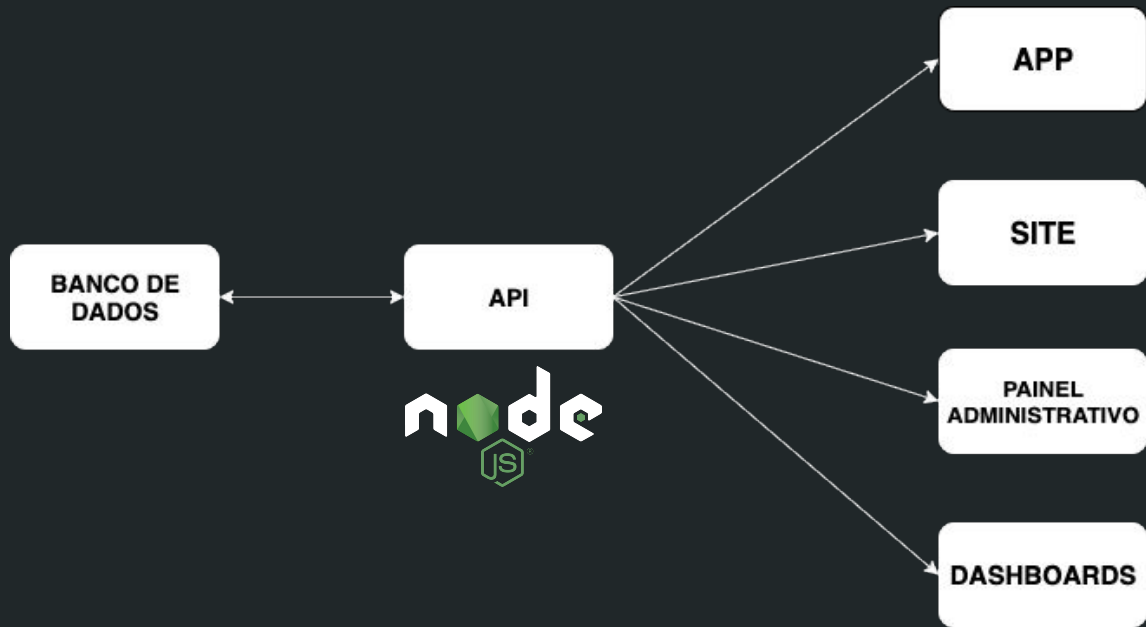
# Typescript

- É um *superconjunto* do Javascript, que adiciona uma tipagem estática ao Javascript
- No final, o código será compilado para Javascript



# API

- Uma API (Application Programming Interface) trata-se de um conjunto de rotinas e padrões que permite a comunicação entre aplicações e usuários



# API - Representação

```
1  {  
2  |    "address": {  
3  |        "street": "Rua Estremosa",  
4  |        "Cidade": "Cachoeirinha"  
5  |    }  
6  }
```

## JSON

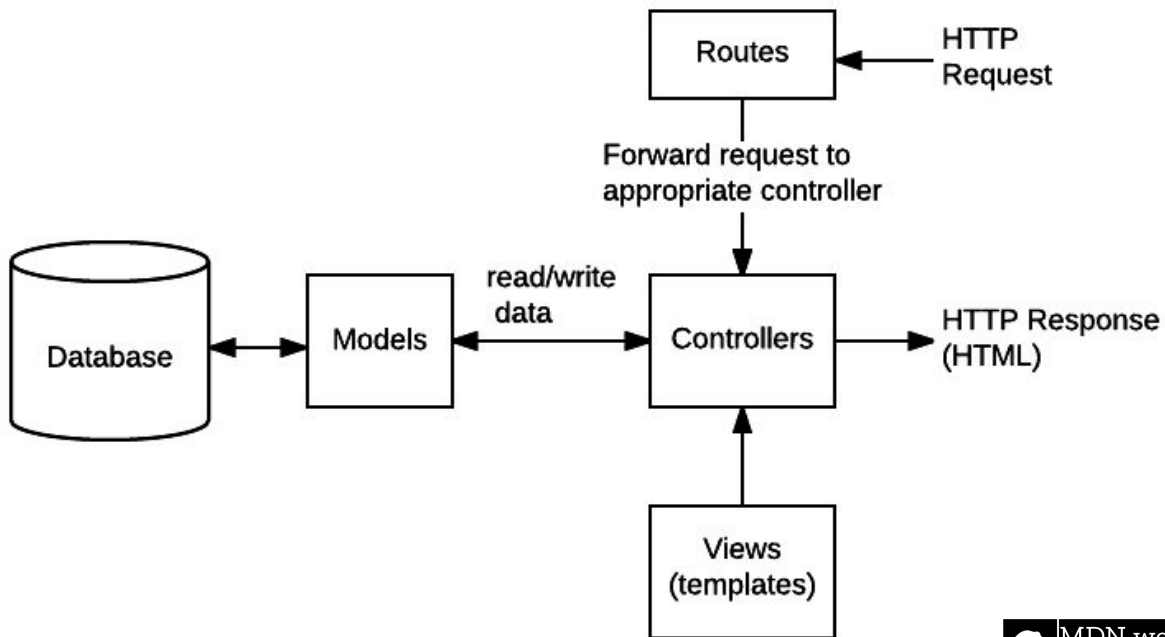
JavaScript Object Notation

```
1  <address>  
2  |    <street>  
3  |        Rua Estremosa  
4  |    </street>  
5  |    <city>  
6  |        Cachoeirinha  
7  |    </city>  
8  </address>
```

## XML

Extensible Markup Language

# API - Estrutura





# API REST / RESTful

- REST (Representational State Transfer) é um conjunto de boas práticas utilizadas nas requisições HTTP realizadas por uma API em uma aplicação web
- Sistemas que utilizando os princípios REST são chamados de RESTful
- As principais requisições HTTP utilizadas são:
  - **POST:** para criar dados no servidor;
  - **GET:** para buscar dados no servidor;
  - **DELETE:** para excluir dados no servidor;
  - **PUT:** para atualizar dados no servidor;

# API REST / RESTful

Launchpad

POST Create user X

GET Get all users

GET Get user by id

PUT Update user

▶ Create user

POST http://localhost:3001/api/v1/user

Params Authorization Headers **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON**

```
1 {
2   "user": {
3     "name": "joao",
4     "register": "171110992",
5     "course": "React",
6     "birthDate": "2017-08-09",
7     "cep": "94965240",
8     "address": "rua holanda",
9     "number": "1711",
10    "address_detail": "",
11    "district": "Marechal Rondon",
12    "status": true,
13    "image": "/assets/teste.png"
14  }
15 }
```

# Testando a API

- Para testar o funcionamento da API RESTful, podemos utilizar outros softwares que irão nos ajudar a validar se as requisições estão de acordo com o esperado



POSTMAN



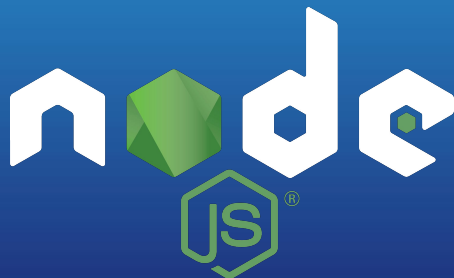
Insomnia



Postwoman



FRONTEND



BACK END

# Dúvidas?

---

# Hands On

---

# Hands On

---

Criar um CRUD (Create, Read, Update, Delete) de usuários para a AGES conforme o layout desenvolvido no figma.

Utilize este repositório do github como ponto de partida

## Vá além

Express



socket.io



Jest



redis



mongoDB



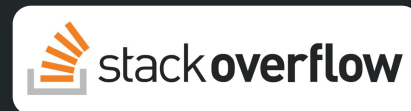
nest



ADONIS



# Onde estudar?



# CHALLENGES

---

## Sua vez!

O Cássio notou que existe a possibilidade de termos mais de um usuário com o mesmo número de matrícula (register).

Você precisa modificar as rotas de criação/edição de usuários para verificar se existe um usuário no banco de dados (user.json) com o mesmo número de matrícula (register) do usuário recebido nestas rotas. Caso exista, deve-se retornar um erro para o client, se não, o usuário pode ser criado/editado.

# Plus

Durante o hands on, notamos que temos rotas que buscam/alteram/deletam informações do usuário. Mas e se o usuário não existir no banco de dados (user.json)?

Analise as rotas e ache uma alternativa, retornando um erro ou uma informação para o front-end que o usuário não foi encontrado ou não existe.

**Obs:** Estude sobre os status das respostas do http [aqui](#).

# Obrigado

---

[gregory@triider.com](mailto:gregory@triider.com)

[gregory.lagranha@acad.pucrs.br](mailto:gregory.lagranha@acad.pucrs.br)

