# Desenv. BackEnd

**Prof. M. Marcelo Petri**

# Olá! Sejam bem vindos

| DISCIPLINA/UNIDADE CURRICULAR | Desenvolvimento Web |
|---|---|
| **TEMA DA AULA** | Programação do servidor Node.js |
| **INDICADOR** | 1. Configura o ambiente de desenvolvimento conforme as funcionalidades e características do projeto.<br>2. Cria estruturas de código utilizando linguagem de programação para back-end, de acordo com os requisitos do projeto de software. |
| CRONOGRAMA | - Arquitetura de aplicações web: conceito de back-end e sua interação com front-end.<br>- Ambiente de desenvolvimento: conceito, interface, configuração e compilação.<br>- Linguagem de programação para back-end: sintaxe, palavras reservadas, variáveis e estruturas de dados; desvios condicionais e laços de repetição.<br>- Biblioteca da linguagem: funções próprias da linguagem e suas aplicações.<br>- Orientação a objetos na linguagem de programação back-end: classes, objetos, interfaces, entre outros |
| **ATIVIDADE** | Desenvolvimento um servidor que receba por parâmetros de URL dois dados e grave em um TXT junto com o professor e exercícios. |

# Objetivos

- Entender os conceitos de Node.js
- Executar um aplicativo NodeJS
- Instalar pacotes NPM
- Exercícios

# Introdução

- O Node.js é um ambiente de tempo de execução que executa código JavaScript em servidores web — o chamado back-end (lado do servidor) — em vez de usar uma segunda linguagem, como Python ou Ruby.

- A linguagem JavaScript já é utilizada no front-end moderno dos aplicativos web, interagindo com o HTML e CSS da interface de usuário do navegador web.

- O uso do Node.js em conjunto com o JavaScript no navegador oferece a possibilidade de se empregar apenas uma linguagem de programação para todo o aplicativo.

# Introdução

- A principal razão para a existência do Node.js é a maneira como ele lida com diversas conexões simultâneas no back-end.

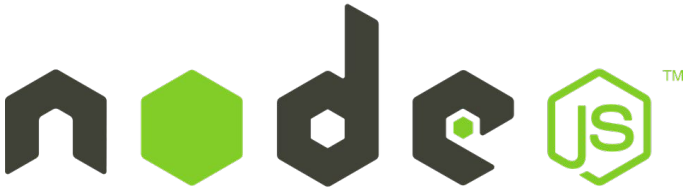
- Uma das maneiras mais comuns de um servidor de aplicativos web manipular conexões é por meio da execução de vários processos. Quando abrimos um aplicativo instalado no computador, inicia-se um processo que consome muitos recursos.

- Imagine o que acontece quando milhares de usuários estão fazendo a mesma coisa em um grande aplicativo web.

# Node.JS

- **Node.js** não é uma linguagem de programação nem um framework javascript.

- **Node.js** é um ambiente de execução de código JavaScript no lado servidor, open-source e multiplataforma




Senac

# Por quê usar Node.JS?

- **Open Source.**
- Utiliza linguagem JavaScript, que aprendemos anteriormente.
- Não precisa de um ambiente de desenvolvimento robusto.
- Utilizado por IBM, LinkedIn, Microsoft, Netflix, PayPal, Rakuten, SAP, Walmart, Cisco, etc.
- JavaScript é a linguagem com mais envios no github
- Muito bem avaliado pela comunidade de desenvolvimento no mundo.

# Por quê usar Node.JS?

- **Stackoverflow:** é um site de perguntas e respostas para programadores profissionais e entusiastas
  - https://pt.stackoverflow.com/

- **Survey:** pesquisa que abrange diversos aspectos sobre a carreira de desenvolvedor.
  - https://insights.stackoverflow.com/survey/2021

# Node.JS

- **Node.js** foi criado por Ryan Dahl, em 2009.

- Ryan Dahl buscava uma solução para o problema de acompanhar o progresso de upload de arquivos sem ter a necessidade de fazer pooling no servidor.

- Em 2009 na JSConf EU ele apresentou o Node.js e introduziu o JavaScript Server Side com I/O não bloqueante, ganhando assim o interesse da comunidade que começou a contribuir com o projeto.

# Node.JS

- Em janeiro de 2010, foi adicionado ao projeto o **npm**, um gerenciador de pacotes que tornou mais fácil para os programadores publicarem e compartilharem códigos e bibliotecas Node.js simplificando a instalação, atualização e desinstalação de módulos, aumentando rapidamente a sua popularidade.

- Em 2011, a Microsoft ajudou o projeto criando a versão Windows de Node. Daí em diante, nunca mais parou de crescer, atualmente sendo mantido pelo **Node.js Foundation**, uma organização independente e sem fins lucrativos que mantém a tecnologia com o apoio da comunidade mundial de desenvolvedores

# Node.JS - Características

- Cada requisição do Node.js não o bloqueia, atendendo a um volume absurdamente grande de requisições ao mesmo tempo, mesmo sendo sigle thread (única fila de tarefas).

- Imagine que exista apenas um fluxo de execução. Quando chega uma requisição, ela entra nesse fluxo. A máquina virtual JavaScritp verifica o que tem de ser feito, delega a atividade (consultar dados no banco, por exemplo) e volta a atender novas requisições enquanto este processamento paralelo está acontecendo. Quando a atividade termina (já temos os dados retornados pelo banco), ela volta ao fluxo principal para ser devolvida ao requisitante.

# Node.JS - Características

- Utilizado na construção de APIs ( Application Programming Interface);
- Utilizado para fazer backend de jogos, IoT e apps de mensagens;
- Node.js serve para fazer aplicações de tempo real.

# Node.JS - Configurando Ambiente

- Download do projeto node.js em https://nodejs.org/
- Após a instalação verificar se está tudo ok
  - **$ node -v**

# Node.JS - Características

- Para executar os exemplos a seguir, crie um diretório de trabalho chamado node_examplos. Abra um prompt de terminal e digite node. Se o Node.js estiver instalado corretamente, ele apresentará um prompt > onde será possível testar os comandos JavaScript interativamente.

- Este tipo de ambiente é chamado REPL, que em inglês é a sigla para “ler, avaliar, imprimir e fazer loop”.

# Node.JS - Características

- Digite a seguinte entrada no prompt >.
- Para sair do terminal Node.js, digite .exit
- Pressione a tecla Enter após cada linha e o ambiente REPL retornará os resultados de suas ações:

```
> let array = ['a', 'b', 'c', 'd'];
> array.map( (element, index) => (`Element: ${element} at index: ${index}`));

Saída:
[
     'Element: a at index: 0',
     'Element: b at index: 1',
     'Element: c at index: 2',
     'Element: d at index: 3'
]
```

# Node.JS - Características

- Para scripts e módulos mais longos, é mais prático usar um editor de texto como o VS Code.
- Crie um arquivo chamado **start.js**
- Adicione o seguinte código:

```
let array = ['a', 'b', 'c', 'd'];
array.map( (element, index) => (
    console.log(`Element: ${element} at index: ${ index}`)
));
```

- Em seguida, execute o script do shell de maneira a produzir os mesmos resultados de antes:

```
$ node ./start.js
```

# Node.JS - Características

- **Laço de eventos e thread único:**
  - No Node.js, um “verificador de eventos” está sempre verificando as operações que foram concluídas ou que estão esperando para serem processadas pelo mecanismo JavaScript. Uma operação assíncrona e longa não bloqueia as outras operações mais rápidas que vêm depois.
  - Como o Node.js evita o uso de múltiplos encadeamentos (threads), como fazem outros ambientes, ele é chamado de ambiente de thread único e, portanto, é importantíssimo que a execução ocorra sem bloqueios.

# Node.JS - Características

- **Módulos:**
  - A modularização ajuda a organizar melhor a base de código, abstrair as implementações e evitar problemas de engenharia complicados.

# Node.JS - Atividade

- Abra seu editor de texto e crie um arquivo chamado volumeCalculator.js contendo o seguinte código JavaScript:

```
const sphereVol = (radius) => {
  return 4 / 3 * Math.PI * radius
}

console.log(`A sphere with radius 3 has a ${sphereVol(3)} volume.`);
console.log(`A sphere with radius 6 has a ${sphereVol(6)} volume.`);
```

- Em seguida execute o arquivo usando o Node:

```
node volumeCalculator.js
```

# Node.JS - Atividade

- Criamos uma função simples para calcular o volume de uma esfera com base em seu raio. Imagine que também precisamos calcular o volume de um cilindro, um cone e assim por diante: percebemos rapidamente que essas funções específicas devem ser adicionadas ao arquivo volumeCalculator.js, que pode acabar se tornando uma enorme coleção de funções. Para organizar melhor a estrutura, podemos usar a ideia por trás dos módulos como pacotes de código separado.

# Node.JS - Atividade

- **Para isso, crie um arquivo separado chamado polyhedrons.js:**

```
const coneVol = (radius, height) => {
  return 1 / 3 * Math.PI * Math.pow(radius, 2) * height;
}
const cylinderVol = (radius, height) => {
  return Math.PI * Math.pow(radius, 2) * height;
}
const sphereVol = (radius) => {
  return 4 / 3 * Math.PI * Math.pow(radius, 3);
}
module.exports = {
  coneVol,
  cylinderVol,
  sphereVol
}
```

# Node.JS - Atividade

- **A seguir, no arquivo volumeCalculator.js, exclua o código antigo e substitua-o por este trecho e depois execute o nome do arquivo no ambiente Node.js (node volumeCalculator.js):**

```
const polyhedrons = require('./polyhedrons.js');

console.log(`A sphere with radius 3 has a ${polyhedrons.sphereVol(3)} volume.`);
console.log(`A cylinder with radius 3 and height 5 has a
${polyhedrons.cylinderVol(3, 5)} volume.`);
console.log(`A cone with radius 3 and height 5 has a ${polyhedrons.coneVol(3,
5)} volume.`);
```

# Node Package Manager (NPM).

- **Uma das principais tarefas do NPM é gerenciar, baixar e instalar módulos externos no projeto ou no sistema operacional.**
- **O comando npm init permite inicializar um repositório de nó.**

# Node.JS - Criando um projeto

- **$ npm init**

- Este comando vai criar o arquivo package.json que é onde está toda a configuração do seu projeto, incluindo: nome, versão, descrição, scripts a serem executados. licença, etc.

- Após executar o comando **npm init** necessário passar algumas informações básicas para montar o arquivo package.json

- Para executar um arquivo, no terminal e digite:
  - **$ node “NomeDoArquivo.js”**

# package.json

- O comando init irá gerar automaticamente um arquivo package.json escrevendo as propriedades do seu projeto/módulo.

- Uma dessas propriedades são as dependências instaladas em seu repositório local. O NPM adicionará o nome e a versão dessas dependências em package.json, junto com package-lock.json, outro arquivo usado como reserva pelo NPM no caso de package.json falhar.

- Digite o seguinte em seu terminal:

```
npm i dayjs
```

# package.json

- Agora que o módulo dayjs está instalado no diretório local, abra o console Node.js e digite as seguintes linhas:

```
const dayjs = require('dayjs');
dayjs().format('YYYY MM-DDTHH:mm:ss')
```

# Funcionalidade do servidor

- Como o Node.js controla o back-end de aplicativos web, uma de suas principais tarefas é lidar com solicitações HTTP.
- Exemplo:
  - A funcionalidade do servidor é escutar solicitações, determinar o mais rápido possível qual a resposta de que cada um precisa e retornar essa resposta ao remetente da solicitação. Esse aplicativo deve receber uma solicitação HTTP de entrada acionada pelo usuário, analisar a solicitação, realizar o cálculo, gerar a resposta e enviá-la de volta.
  - Um módulo HTTP, como o Node.js, é usado porque simplifica essas etapas, permitindo que o programador se concentre no próprio aplicativo.

# Funcionalidade do servidor

- Implementa essa funcionalidade básica, em um arquivo chamado basic_server.js:

```
const http = require('http');
const url = require('url');
const hostname = '127.0.0.1';
const port = 3000;
const server = http.createServer((req, res) => {
  const queryObject = url.parse(req.url,true).query;
  let result = parseInt(queryObject.a) + parseInt(queryObject.b);
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end(`Result: ${result}\n`);
});
server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

# Funcionalidade do servidor

- Em seguida, visite a URL a seguir em seu navegador web e veja o que acontece:

```
http://127.0.0.1:3000/numbers?a=2&b=17
```

# Resumo

- Tivemos uma visão geral do ambiente do Node.js, suas características e como ele pode ser usado para implementar programas simples. A lição inclui os seguintes conceitos:
  - O que é o Node.js e por que é usado.
  - Como executar programas do Node.js usando a linha de comando.
  - Os laços de eventos e o thread único.
  - Módulos.
  - Node Package Manager (NPM).
  - Funcionalidade de servidor.

# Atividades

- **Exemplo 01:**

```
let numero1 = 2;
let numero2 = 3;

console.log("Olá mundo");
console.log("Soma: "+ (numero1 + numero2));

function soma(num1, num2){
    return (num1 + num2);
}

console.log(soma(2, 7));
```

# Node.JS - Síncrono x Assíncrono

- Em um programa síncrono, ao rodar um código como o abaixo, caso a função readFileSync() demore muito tempo para ser executado, todo o programa fica bloqueado até que os dados sejam carregados.
- O resultado do cálculo não seria mostrado na tela, por exemplo.

```
const fs = require('fs');
let fileContent;
const someMath = 1+1;

try{
    fileContent = fs.readFileSync('big-file.txt', 'utf-8');
    console.log('O arquivo foi lido.');
} catch (err) {
    console.log(err);
}

const text = `A soma é ${someMath}`;
console.log(text);
```

# Node.JS - Síncrono x Assíncrono

- Em um programa assíncrono utilizamos uma funcionalidade chamada **high order functions.**
- As **high order functions** possibilitam passar uma função por parâmetro para outra função, as funções passadas como parâmetro serão executadas posteriormente.

```
const fs = require('fs');
const someMath = 1+1;

fs.readFile('big-file.txt', 'utf-8', function(err, content){
    if (err){
        return console.log(err);
    }
    //console.log(content);
    console.log('O arquivo foi lido.');
})

const text = `A soma é ${someMath}`;
console.log(text);
```

# Node.JS - Módulo HTTP

- Para envio e recebimento de dados através do protocolo HTTP, o Node possui um módulo que permite essa interação.

```
var http = require('http');
```

- Para criar um servidor simples:

```
var http = require('http');
//Cria um novo servidor
http.createServer(function(req, res){
    res.write('Olá mundo'); //Escreve olá mundo
    res.end(); //Termina de escrever
}).listen(8080); //Especifica a porta do servidor
```

# Node.JS - Módulo HTTP

- Exemplo 2: (localhost:8080 | localhost:8081)

```
var http = require('http');
//Cria um novo servidor

//Microserviço escrita de texto
http.createServer(function(req, res){
    res.write('Olá mundo');
    res.end();
}).listen(8080);

//Microserviço escrita HTML
http.createServer(function(req, res){
    res.writeHead(200, {'Content-Type':'text/html'});
    res.write('<p>Olá mundo</p>');
    res.end();
}).listen(8081);
```

# Node.JS - Módulo HTTP

- Exemplo 3: (localhost:8080)

```
var http = require('http');
//Cria um novo servidor

//Microserviço escrita de texto
http.createServer(function(req, res){
    res.write(req.url);
    res.end();
}).listen(8080);
```

Url requisitada

# Node.JS - Módulo URL

- Exemplo 3: (http://localhost:8080/?nome=marcelo&senha=123)

```
var http = require('http');
var url = require('url');

http.createServer(function(req, res){
    var query = url.parse(req.url, true).query;
    let nome = query.nome;
    let senha = query.senha;
    res.write(nome + " | " + senha);
    res.end();
}).listen(8080);
```

# Node.JS - Gravar arquivo txt

- Exemplo 4:

```
var http = require('http');
var url = require('url');
var fs = require('fs');

http.createServer(function(req, res){
    var query = url.parse(req.url, true).query;
    let nome = query.nome;
    let senha = query.senha;
    let texto = nome + " | " + senha;

    fs.appendFile("banco.txt", texto, function(err){
        if (err)
            throw err;
        console.log("Arquivo Atualizado!!")
    });
    res.write(nome + "|" + senha);
    res.end();
}).listen(8080);
```

# Node.JS - Gravar arquivo txt

- Exemplo 5:

```
var http = require('http');
var url = require('url');
var fs = require('fs');
http.createServer(function(req, res){
    if (req.url != '/favicon.ico'){
        var query = url.parse(req.url, true).query;
        let nome = query.nome;
        let senha = query.senha;
        let texto = nome + " | " + senha;
        fs.appendFile("banco.txt", texto, function(err){
            if (err)
                throw err;
            console.log("Arquivo Atualizado!!")
        });
        res.write(nome + "|" + senha);
        res.end();
    }
}).listen(8080);
```

# Node.JS - Gravar arquivo txt

- Exemplo 6:

```
var http = require('http');
var url = require('url');
var fs = require('fs');
http.createServer(function(req, res){
    if (req.url != '/favicon.ico'){
        var query = url.parse(req.url, true).query;
        let nome = query.nome;
        let senha = query.senha;
        if (nome != undefined && senha != undefined){
            let texto = nome + " | " + senha + "\n";
            fs.appendFile("banco.txt", texto, function(err){
                if (err)
                    throw err;
                console.log("Arquivo Atualizado!!")
            });
            res.write(nome + "|" + senha);
            res.end();
        }
    }
}).listen(8080);
```

# Node.JS - Ler arquivo txt

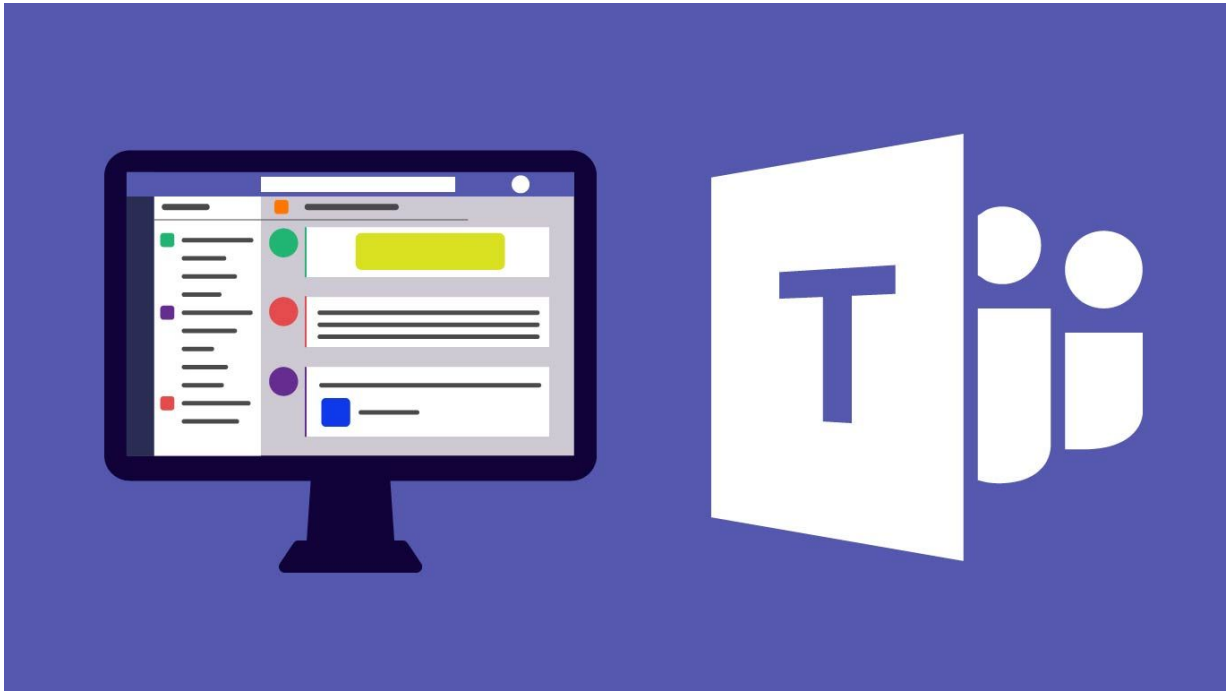
- Exemplo 7:

```
var http = require('http');
var fs = require('fs');

http.createServer(function(req, res){
    fs.readFile("banco.txt", function(err, data){
        res.write(data);
        return res.end();
    })
}).listen(8088);
```

# Atividades

- TEAMs



# Links

# Referências


- **Básicas: [OBRIGATORIAMENTE 3 BIBLIOGRAFIAS]**
- ARAÚJO, Everton Coimbra de. Desenvolvimento para web com Java. Florianópolis: Visual Books, 2010.
- QIAN et al. Desenvolvimento Web Java. LTC, 2010.
- GOMES, Yuri Marx Pereira. Java na web com JSF, Spring, Hibernate e Netbeans 6. Rio de Janeiro, RJ: Ciência Moderna, 2008.

- **Complementar: [OBRIGATORIAMENTE 5 BIBLIOGRAFIAS]**
- GONÇALVES, Edson. Desenvolvendo aplicações web com netbeans ide 6. Rio de Janeiro, RJ: Ciência Moderna, 2008.
- LUCKOW, Décio Heinzelmann; MELO, Alexandre Altair de. Programação Java para Web. 2ª Ed. São Paulo: Novatec, 2015.
- SOUZA, Thiago Hernandes de. Java mais primefaces mais ireport: desenvolvendo um CRUD para web. Rio de Janeiro, RJ: Ciência Moderna, 2013.
- KALIN, Martin. Java Web Services: implementando. Rio de Janeiro, RJ: Alta Books, 2010.
- EIS, Diego. Guia Front-End: o caminho das pedras para ser um Dev Front-End. São Paulo: Casa do Código, 2015.
- SOUZA, Alberto. Spring MVC: domine o principal framework web Java. São Paulo: Casa do Código, 2015.