UniSENAI

Módulo 2: Fundamentos do Node.js

Módulo 2: Fundamentos do Node.js

Objetivo:

Explorar conceitos fundamentais que capacitam os desenvolvedores a criar aplicações eficientes com Node.js, aproveitando seus recursos nativos e padrões de programação.

Aula 2.1: O Loop de Eventos e a Programação Assíncrona

Conceito

O loop de eventos é o mecanismo central que permite ao Node.js executar código de maneira assíncrona. Ele monitora eventos e gerencia o fluxo de execução, permitindo que tarefas I/O sejam delegadas enquanto outras são processadas.

Componentes Chave

1. Call Stack: Onde funções são executadas.
2. Callback Queue: Fila que armazena tarefas aguardando execução.
3. Thread Pool: Gerencia tarefas de I/O complexas e operações em background.
4. Event Loop: Liga o Call Stack à Callback Queue, garantindo que as tarefas sejam processadas de maneira ordenada.

Exemplo Prático

```
console.log('Início');

setTimeout(() => {
  console.log('Executado depois de 2 segundos');
}, 2000);

console.log('Fim');
```

Saída:

```
Início
Fim
Executado depois de 2 segundos
```

Importância da Programação Assíncrona

Permite que aplicações processem grandes volumes de tarefas simultaneamente sem bloquear a execução do código principal.

Aula 2.2: Módulos no Node.js: CommonJS vs ES Modules

O que são Módulos?

Os módulos são blocos reutilizáveis de código que ajudam na organização e manutenção de projetos.

CommonJS

- Padrão histórico do Node.js.
- Usa `require` para importar módulos e `module.exports` para exportá-los.

Exemplo:

```
// arquivo.js
module.exports = function() {
  console.log('Módulo CommonJS');
};

// app.js
const modulo = require('./arquivo');
modulo();
```

ES Modules (ESM)

- Padrão moderno (ECMAScript).
- Usa `import` e `export`.

Exemplo:

```
// arquivo.mjs
export const saudacao = 'Olá, ES Modules!';

// app.mjs
import { saudacao } from './arquivo.mjs';
console.log(saudacao);
```

Diferenças Principais

- CommonJS: Executa módulos de maneira sincrona.
- ESM: Opera de maneira assíncrona e é mais compatível com navegadores modernos.

Quando usar:

- Use CommonJS para aplicações legadas ou completamente server-side.
- Prefira ESM para projetos modernos ou com compatibilidade web.

Aula 2.3: Manipulação de Arquivos e Diretórios com o módulo `fs`

Conceito

O módulo `fs` (File System) permite interagir com o sistema de arquivos para criar, ler, atualizar e excluir arquivos e diretórios.

Principais Métodos

- **Síncronos**: Bloqueiam a execução até que a operação seja concluída.
- **Assíncronos**: Executam tarefas em background.

Exemplos Práticos

Criar e Ler Arquivos:

const fs = require('fs');

```
const fs = require('fs');

// Criar arquivo
fs.writeFileSync('arquivo.txt', 'Conteúdo inicial');

// Ler arquivo
const conteudo = fs.readFileSync('arquivo.txt', 'utf8');
console.log(conteudo);
```

Assíncrono:

```
fs.writeFile('arquivo.txt', 'Conteúdo atualizado', (err) => {
  if (err) throw err;
  console.log('Arquivo criado com sucesso!');
});
```

Aplicabilidade

- Processamento de arquivos de log.
- Manipulação de uploads.

Aula 2.4: Trabalhando com Streams e Buffers

Conceito

Streams permitem o processamento de dados em pedaços (chunks), ideal para grandes volumes de informações.

Tipos de Streams

- **Readable**: Leitura de dados (ex.: arquivos).
- **Writable**: Escrita de dados.
- **Duplex**: Ambos (ex.: conexão TCP).
- **Transform**: Modifica os dados durante o fluxo.

Exemplo Prático

Leitura de Arquivo com Stream:

```
const fs = require('fs');

const stream = fs.createReadStream('grandeArquivo.txt', 'utf8');
stream.on('data', (chunk) => {
  console.log('Novo pedaço recebido:', chunk);
});
```

Buffers

Buffers armazenam temporariamente dados binários durante o processamento. São amplamente usados em streams.

Exemplo:

```
const buffer = Buffer.from('Node.js');
console.log(buffer.toString());
```

Aula 2.5: Tratamento de Erros e Estratégias de Depuração

Tipos de Erros

1. **Erros de Sintaxe:** Ocorrem durante a compilação.
2. **Erros de Runtime:** Ocorrem durante a execução do código.
3. **Erros Lógicos:** Resultados inesperados devido à lógica incorreta.

Métodos de Tratamento

- **Try/Catch:**

```
try {
  const data = fs.readFileSync('inexistente.txt', 'utf8');
} catch (err) {
  console.error('Erro ao ler o arquivo:', err.message);
}
```

- **Eventos de Erro:**

```
const stream = fs.createReadStream('arquivo.txt');
stream.on('error', (err) => {
  console.error('Erro no stream:', err.message);
});
```

Depuração

- Use `console.log()` para verificar variáveis.
- Ferramentas como o `--inspect` para debugging interativo:

```
node --inspect app.js
```

Aula 2.6: Trabalhando com o módulo `http` para criar servidores básicos

Conceito

O módulo `http` permite criar servidores que recebem e respondem a requisições HTTP.

Exemplo Básico

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

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Olá, mundo!');
});

server.listen(3000, () => {
  console.log('Servidor rodando em http://localhost:3000');
});
```

Aplicabilidade

- Criação de servidores personalizados.
- Base para frameworks como Express.js.

Lista de Exercícios

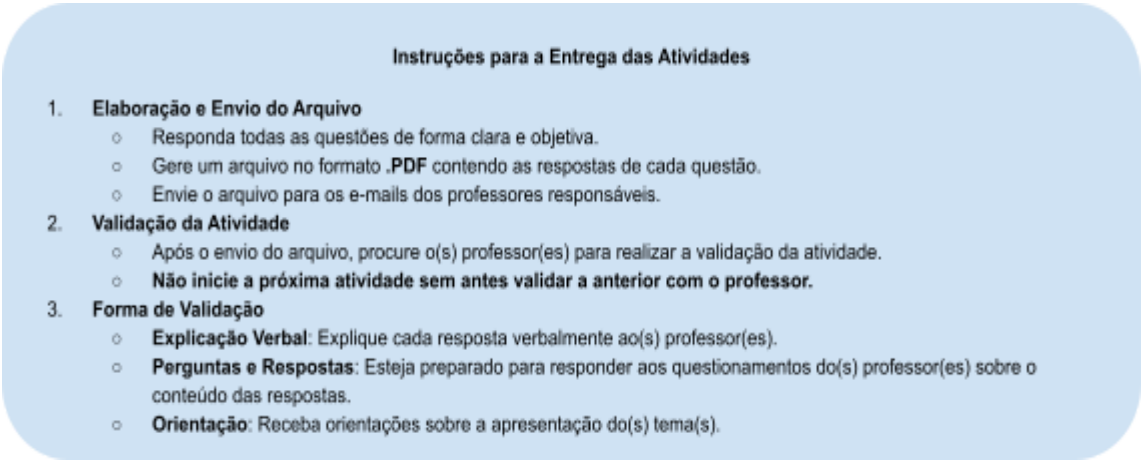
Questões Teóricas

1. O que é o loop de eventos e qual sua importância no Node.js?
2. Diferencie CommonJS de ES Modules.
3. O que são Streams e Buffers?
4. Liste três tipos de Streams e suas aplicações.
5. Explique a diferença entre erros de runtime e erros de sintaxe.
6. Quais são as vantagens de usar Streams para processamento de dados?
7. Como o módulo `fs` pode ser usado para manipular arquivos?
8. O que acontece quando um erro não é tratado em um stream?
9. Qual é a função do `http` no Node.js?
10. Explique o papel do Thread Pool no Node.js.

Questões Práticas

1. Crie um servidor HTTP que responda com "Bem-vindo ao Node.js!".
2. Escreva um código para criar e ler um arquivo usando o módulo `fs`.
3. Implemente um stream para processar dados de um arquivo grande.

4. Utilize Buffers para manipular uma string e convertê-la em dados binários.
5. Crie um módulo CommonJS que exporta uma função simples e importe-o em outro arquivo.
6. Configure e leia um arquivo JSON usando o módulo `fs`.
7. Crie um script que utilize try/catch para tratar erros ao acessar um arquivo inexistente.
8. Implemente um stream Duplex para modificar dados durante a leitura e escrita.
9. Configure um servidor HTTP que responda de forma diferente para cada rota.
10. Debug um erro em um script usando o `--inspect`.



Instruções para a Entrega das Atividades

1. **Elaboração e Envio do Arquivo**
   - Responda todas as questões de forma clara e objetiva.
   - Gere um arquivo no formato **.PDF** contendo as respostas de cada questão.
   - Envie o arquivo para os e-mails dos professores responsáveis.
2. **Validação da Atividade**
   - Após o envio do arquivo, procure o(s) professor(es) para realizar a validação da atividade.
   - **Não inicie a próxima atividade sem antes validar a anterior com o professor.**
3. **Forma de Validação**
   - **Explicação Verbal**: Explique cada resposta verbalmente ao(s) professor(es).
   - **Perguntas e Respostas**: Esteja preparado para responder aos questionamentos do(s) professor(es) sobre o conteúdo das respostas.
   - **Orientação**: Receba orientações sobre a apresentação do(s) tema(s).